

**CENTRO DE INVESTIGACION CIENTIFICA Y DE
EDUCACION SUPERIOR DE ENSENADA**

**ANALISIS E IMPLANTACION DE FORTH PARA
PROGRAMACION DE PROCESOS DE CONTROL**

**T E S I S
MAESTRIA EN CIENCIAS**

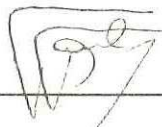
JESUS ALFONSO ESPARZA BETANCOURTH

Ensenada, Baja California, Enero de 1989.

RESUMEN de la Tesis de Jesús Alfonso Esparza Betancourth presentada como requisito parcial para la obtención del grado de MAESTRO EN CIENCIAS DE LA COMPUTACION. Ensenada, Baja California, México. Enero de 1989.

ANALISIS E IMPLANTACION DE FORTH PARA PROGRAMACION
DE PROCESOS DE CONTROL.

Resumen aprobado por:



M. en C. Fernando Favela Vara
Director de Tesis

La necesidad de crear aplicaciones de control en el menor tiempo posible ha llevado a realizar una herramienta programática para un entorno limitado de recursos mecánicos y de energía, basándose en el lenguaje Forth. El dialecto desarrollado permite crear aplicaciones basadas en el microprocesador MC146805E2 de Motorola el cual se encuentra en la computadora en una tarjeta CUTCMOS, desarrollada en CICESE. La metodología empleada para construir un dialecto de Forth comprendió el análisis del lenguaje mismo así como la selección de la mejor técnica, de acuerdo a las limitantes existentes (de programación y mecánicas), para realizar su implantación. Una vez seleccionada la técnica, se procedió a escribir las primitivas del dialecto y determinar la parte de la memoria donde deberían residir cada una de ellas. Para optimizar el espacio de memoria disponible se creó un mecanismo de intercambio de bancos de memoria que permitió expandir el área destinada para la creación de nuevas definiciones hasta 32 K, distribuidos en 8 bancos, cada uno de 4 K de longitud. El dialecto implantado contiene primitivas que permiten desarrollar aplicaciones escritas en Forth bajo la norma de 1979, además de palabras de adquisición de datos de un convertidor analógico-digital.

CENTRO DE INVESTIGACION CIENTIFICA Y DE
EDUCACION SUPERIOR DE ENSENADA.

CENTRO DE COMPUTO

ANALISIS E IMPLANTACION DE FORTH PARA
PROGRAMACION DE PROCESOS DE CONTROL.

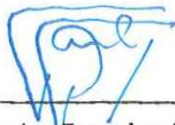
TESIS

que para cubrir parcialmente los requisitos necesarios para
obtener el grado de MAESTRO EN CIENCIAS presenta:

JESUS ALFONSO ESPARZA BETANCOURTH

Ensenada, Baja California, Enero de 1989.

TESIS APROBADA PARA SU DEFENSA POR:



M.C. Fernando Favela Vara, Director del Comité



M.C. Carlos Roberto Duarte Muñoz, Miembro del Comité



M.C. José Luis Briseño Cervantes, Miembro del Comité



M.C. Jaime Sánchez García, Miembro del Comité



Ing. Salvador Castañeda Avila, Jefe del Centro de Cómputo Electrónico



M.C. Cuauhtémoc Nava Button, Director Académico

Tesis presentada en Enero 23, 1989

DEDICATORIA

Para Ma. Esther y Alfonso,
quienes siempre han estado conmigo,
a pesar de la distancia.

"Un poco de lucidez, mucho realismo y un amigo
para cuando todo va realmente mal
son lo verdaderamente necesario
para eludir las malas rachas."

Para Cuauhtémoc Sepúlveda,
por ser ese amigo.

AGRADECIMIENTOS

Al M. en C. Fernando Favela Vara por poner objetivos más altos cada vez que se alcanzaban otros.

Al personal de la Escuela Superior de Ciencias por todo el apoyo otorgado, especialmente a Alejandro Martínez y Carmen Pérez F.

Al personal de Biblioteca de CICESE por las facilidades durante la búsqueda de información.

A Carlos Duarte y Armando Valenzuela por las primeras noticias de FORTH.

A todos aquellos que me ayudaron a encontrar nuevas publicaciones sobre FORTH.

CONTENIDO

	<u>Página</u>
I INTRODUCCION	1
II DESCRIPCION DE FORTH	4
II.1 Generalidades	4
II.2 El interprete externo	22
II.3 La compilación de Forth	29
III IMPLEMENTACION DE UN LENGUAJE	38
III.1 Introducción	38
III.2 Código hilado	41
III.2.1 Descripción	41
III.2.2 Código hilado subrutina	42
III.2.3 Código hilado directo	45
III.2.4 Código hilado indirecto	46
III.2.5 Código hilado por señal	49
III.3 Selección de la técnica	49
IV IMPLEMENTACION DE FORTH PARA CUTCMOS	53
V USO DE FORTH/05	64
VI PRUEBAS Y CONCLUSIONES	68
LITERATURA CITADA	72
APENDICES	73
A: Listado de FORTH/05.	73
B: Formatos S1-S9 y HEX.	116
C: Programa de conversión de formatos.	118
D: Primitivas del dialecto FORTH/05.	120
E: Instalación de FORTH/05.	134

LISTA DE FIGURAS

<u>Figura</u>	<u>Página</u>
1. Ejecución de la palabra CUADRADO.	5
2. Recorridos de ejecución a) para CUADRADO y b) para CUBO.	8
3. Operaciones con VARIABLE: a) Inserción (!) y b) Búsqueda (@).	9
4. Mapa de memoria de FORTH.	12
5. Formato de la palabra en FORTH. El campo de código de FORTH guarda la dirección de la subrutina en lenguaje de máquina para una clase de palabra en particular. El campo de parámetros guarda los datos por ser procesados por tal subrutina. A menudo, tales datos son direcciones a otras palabras que son ejecutadas.	13
6. Estructuras de varias palabras hijo. Las diferentes palabras de definición producen diferentes formatos de palabras. La palabra de definición determina la estructura del campo de parámetros y la subrutina a la cual apunta la dirección del campo de código.	16
7. Jerarquía de las palabras en FORTH.	17
8. El intérprete externo.	24
9. Ejecución Hilada.	27
10. Código hilado subrutina.	44
11. Código hilado directo.	47
12. Código hilado indirecto.	48
13. Código hilado por señal.	50
14. Proceso de Ejecución de Código hilado directo.	51
15. Proceso de Ejecución de Código hilado indirecto.	51
16. Proceso de Ejecución de Código hilado por señal.	52
17. Mapa de memoria de microcomputadora.	54

LISTA DE FIGURAS (Continuación)

<u>Figura</u>	<u>Página</u>
18. Representación interna dentro del diccionario.	57
19. Mapa de memoria de FORTH.	61
20. Proceso de cargado de FORTH/05.	63
21. Programa para graficar.	69

ANALISIS E IMPLANTACION DE FORTH PARA PROGRAMACION DE PROCESOS DE CONTROL

I INTRODUCCION

La programación de procesos de control utilizando un computador digital es una tarea que indudablemente está relacionada con el conocimiento profundo del procesador que se ha de utilizar, ya que por su naturaleza, la mayoría de los procesos controlados requieren una alta velocidad de respuesta (tiempo real) y las opciones de solución invariablemente conducen a la utilización del lenguaje ensamblador. Esto provoca que el desarrollo de aplicaciones consuma bastante tiempo y que las soluciones obtenidas sean altamente dependientes de la máquina usada y por lo tanto, de poca transportabilidad.

Si agregamos que el entorno de funcionamiento de dichas aplicaciones es a menudo poco favorable (por ejemplo, sistemas portátiles en zonas aisladas geográficamente) tenemos problemas para los cuales se tienen pocos recursos para resolverlos.

Ante este panorama, se planeó desarrollar una herramienta programática que facilitara realizar aplicaciones de control en menor tiempo al permitir al usuario-programador interactuar con el computador más a nivel de humano que de máquina, lo que resultaría en soluciones algorítmicamente

transportables a otros sistemas, reduciéndose su dependencia del procesador.

Las características del recurso programático deberán satisfacer las necesidades de rápida ejecución, ser de naturaleza estructurada, fácilmente modificable y transportable.

La característica principal de la máquina en la cual se utilizará este recurso es su rendimiento favorable en situaciones adversas de suministro de energía.

Al analizar los recursos con los que se contaban, sobre todo materiales, se decidió utilizar un computador en una tarjeta desarrollada en CICESE basada en el microprocesador MC146805E2 (Duarte C., Favela Vara F.(1983)), que ofrece la ventaja de consumir poca energía y cuyo procesador está especialmente diseñado para controlar procesos (Fattal, J. (1982)).

Las características del recurso programático nos conducían a pensar en un lenguaje de alto nivel, probablemente C o Pascal, pero habiendo seleccionado el computador, estos quedaban eliminados por requerir una alta cantidad de recursos mecánicos. Además, se pensaba en algo que fuera compacto por la limitada capacidad de direccionamiento del procesador (8K). De los lenguajes

existentes pocos se ajustaban a estas características. Entre ellos encontramos PL/9, MPL, APL y Forth. PL/9, MPL y APL son lenguajes parecidos a Pascal, del cual toman las estructuras de selección y repetición así como el concepto de dividir el programa en procedimientos. Estos lenguajes son bastante rápidos solo que están orientados a usarse en un procesador específico (por ejemplo, MPL se emplea en el procesador 6800 de Motorola, exclusivamente (Anónimo, (1978))).

Forth es a menudo criticado por ser difícil de aprender, casi imposible de leer y obtuso pero mostraba bastantes atractivos: su naturaleza de meta-lenguaje, es decir, su capacidad de crear nuevos lenguajes; su pequeño tamaño al no requerir cantidades grandes de memoria; el ser sumamente rápido (Rather, E. y Moore, C. (1976)) y ser muy sensible, ya que es prácticamente imposible usarlo sin hacerle algún cambio, fueron características que nos condujeron a seleccionarlo. Además, Forth es poco conocido en México, por lo que presentaba una oportunidad para darlo a conocer. Así, el objetivo de esta tesis fue desarrollar un dialecto de Forth, que se denominó **FORTH/05** por ser una versión para el microprocesador MC146805E2, que permitiera llevar a cabo aplicaciones de control de procesos en menor tiempo que el empleado si se utilizara un ensamblador.

II DESCRIPCION DE FORTH

II.1 Generalidades

FORTH es un intérprete y un compilador a la vez. Es un lenguaje interactivo consistente de un diccionario de palabras que son sus funciones. Cuando son invocadas las palabras al ser tecleadas, se analizan y ejecutan inmediatamente (es decir, son interpretadas), o si son usadas en la definición de otra palabra, se lleva a cabo su compilación. La compilación también ocurre interactivamente en el teclado o desde código fuente en disco. Escribir un programa consiste solamente en definir nuevas palabras, cada una de las cuales puede considerarse como una subrutina. Las palabras se combinan hasta que, eventualmente, se define una palabra final la cual al ser ejecutada, es el programa. Esto es, un programa en FORTH en su forma más altamente estructurada es simplemente una extensión del lenguaje mismo, compilado en piezas pequeñas, cada una de las cuales puede depurarse en su misma definición. Los programas se desarrollan simultáneamente de arriba-a-abajo (siguiendo un diagrama de flujo) y de abajo-a-arriba (escribiendo y probando las secciones).

Otro tratamiento poco común es el uso de una estructura de pila (último en entrar, primero en salir) para la transferencia de argumentos entre funciones (se pueden usar

nombres de variables pero sus valores se colocan en la pila antes que se ejecute la función). El uso de esta estructura es el resultado de la aplicación de la notación postfija (también llamada notación polaca inversa), la cual radicalmente simplifica el código fuente y su ejecución. FORTH por ello ofrece un entorno creativo en el cual los programas se desarrollan pieza por pieza como parte de un proceso general de solución de problemas.

La pila juega un papel central en todas las operaciones de FORTH. Si el usuario teclea

5 DUP * .

se desplegará el número 25. La inserción de 5 coloca el número 5 en la pila; las siguientes tres palabras duplican (DUP) el 5, multiplican (*) los dos números, y despliegan en la pantalla (.) el resultado. Lo que ocurre en la pila se presenta en la figura 1.

, : CUADRADO DUP * . ;

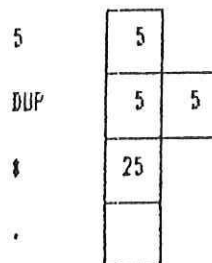


Figura 1. Ejecución de la palabra CUADRADO

FORTH analiza la cuerda de datos de entrada, usando el espacio en blanco como delimitador (así que todas las palabras y números deben ser separadas por al menos un espacio). En el análisis de la cuerda de datos, FORTH busca la palabra en el diccionario. Si la cuerda de caracteres es una palabra válida (existe en el diccionario), se ejecuta. Si no lo es, FORTH determina si la cuerda representa un número válido, en cuyo caso coloca el número en la pila. Si la cuerda no es una palabra o un número válido, se envía un mensaje de error a la terminal.

La compilación consiste simplemente en definir nuevas palabras (creando nuevas entradas al diccionario). Cada nueva palabra puede probarse antes que se agregue la siguiente, lográndose así el desarrollo y depuración del programa más rápidamente. Esto es, si el usuario teclea

: CUADRADO DUP * ;

la palabra CUADRADO sería agregada al diccionario, ya que la palabra **:** (dos puntos) está encargada de realizar esta función. Tecleando

5 CUADRADO .

aparecería en el monitor 25 porque CUADRADO ejecuta las palabras empleadas en su definición. CUADRADO puede ser empleada en nuevas definiciones. Por ejemplo:

: CUBO DUP CUADRADO * ;

cuando se use como

5 CUBO .

desplegará en el monitor el número 125. El recorrido de la ejecución aparece en las figuras 2a y 2b.

CUADRADO y CUBO se agregan al diccionario por la acción de `:` (dos puntos); que es la palabra que se emplea más frecuentemente dentro del grupo de palabras de definición que crean nuevas palabras. `:` cambia temporalmente la acción del lenguaje. La primera cuerda de caracteres después de `:` se agrega al diccionario como el nombre de una nueva palabra. Las palabras que siguen en la cuerda de datos no son ejecutadas, pero son compiladas en la definición de la nueva palabra hasta que se encuentra un `;` (punto y coma), el cual indica el término de la definición.

Las palabras en una definición son compiladas de una manera poco usual: sus direcciones se compilan en memoria. Cuando se teclea la nueva palabra, la ejecución salta de dirección en dirección. El ejecutar un programa saltando de subrutina en subrutina como se especifica por medio de una lista de direcciones, se llama ejecución hilada, y es por ello que a **FORTH** se le cataloga como un lenguaje interpretado en forma hilada.

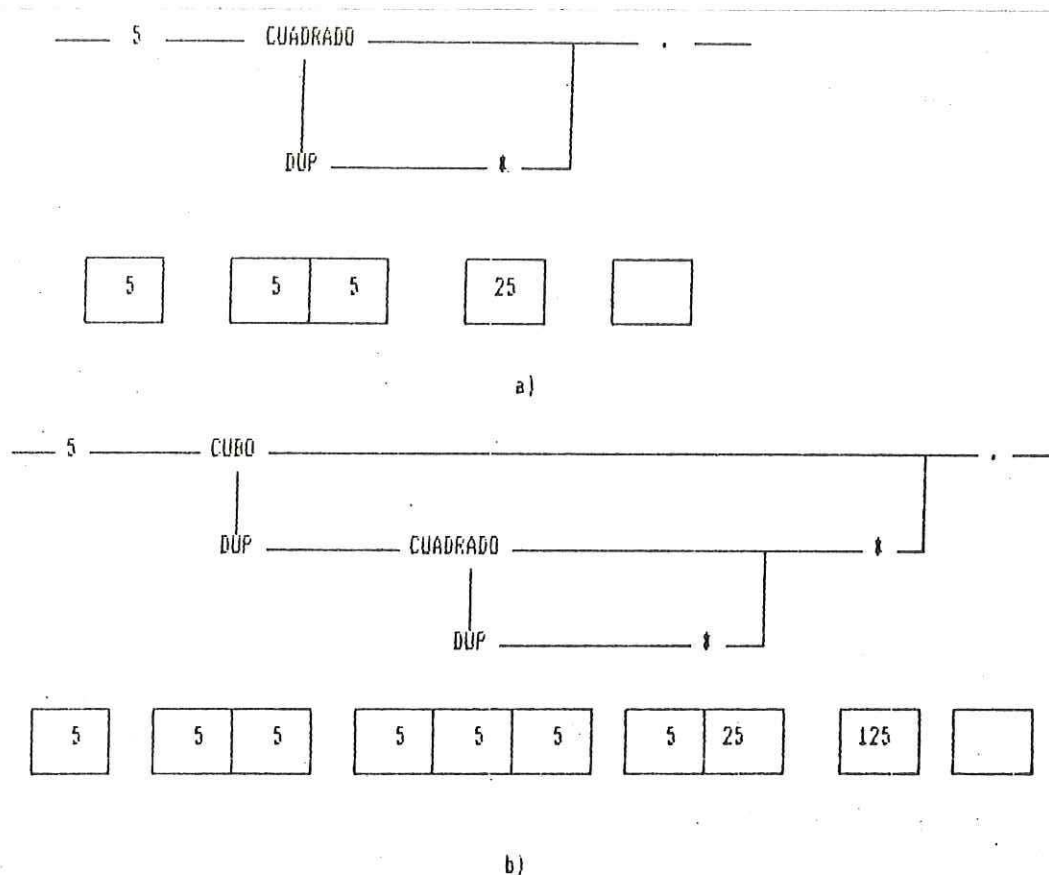


Figura 2. Recorridos de ejecución
a) para CUADRADO y
b) para CUBO

Existen varias palabras de definición, aparte de !.

La palabra de definición VARIABLE, cuando se usa como

VARIABLE MIVARIABLE

crea una nueva palabra MIVARIABLE que asigna espacio para guardar un número entero. Cuando MIVARIABLE se ejecuta, la dirección donde el número se almacena es colocada en la pila. La palabra ! (que se denomina store) coloca al número que se encuentra como segundo elemento de la pila en la

localidad de memoria que es direccionada por el valor que esta en la cima. La palabra @ (significa fetch -busca y trae-) toma una dirección de la pila y devuelve (es decir, coloca en la cima de la pila) el número que está guardado en aquella dirección. Esto es

6 MIVARIABLE !

seguida de

MIVARIABLE @ CUADRADO .

desplegará en el monitor el número 36 (ver figura 3).

Asumiendo que MIVARIABLE tiene la dirección 1000H

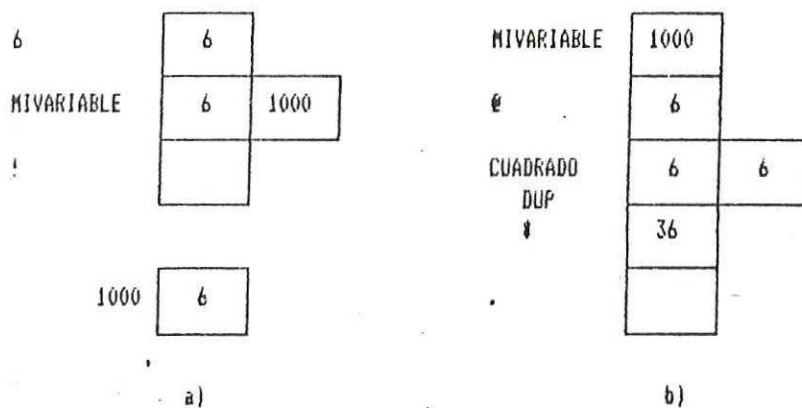


Figura 3. Operaciones con VARIABLE: a) inserción (!) y b) búsqueda (@).

Una palabra de definición más poderosa lo es CODE la cual permite hacer definiciones en lenguaje de máquina. CODE usa un diccionario diferente de mnemónicos de lenguaje

ensamblador para crear nuevas primitivas. (Una primitiva es una subrutina en lenguaje de máquina que se ejecuta sin hilar). Por ejemplo SWAP, que es una de las palabras que manipulan el contenido de la pila -intercambia los dos números que se encuentran en la cima de ella-, puede definirse como

```
CODE SWAP BX POP AX POP BX PUSH AX PUSH END-CODE
```

Note que el ensamblador (en este caso para el microprocesador 8088) también usa notación postfija. Esto ilustra un punto más importante: **FORTH** está escrito en su mayoría en **FORTH**. Un dialecto se desarrolla definiendo un pequeño kernel o conjunto de definiciones básicas usando un ensamblador. Después, ya que todas las palabras de bajo nivel se han definido, todas las demás se definen usando el mismo **FORTH**, ya sea a través del vocabulario de ensamblador, con definiciones de dos puntos o, raramente, a través de otros medios (como lo puede ser CODE).

El programador de **FORTH**, como el programador de otros lenguajes ricos en procedimientos como Pascal, C o Lisp, define procedimientos (palabras) más poderosos y generales al problema por resolver en términos de otros más simples que ya están definidos. Pero el usuario de **FORTH** está mejor habilitado para emplear este estilo porque es mínimo el tiempo y los costos en espacio de memoria en las llamadas a procedimientos en **FORTH**. Las invocaciones de **FORTH** emplean

rutinas pequeñas y rápidas en lenguaje de máquina y el direccionamiento directo a través de rutas hiladas elimina la duplicación de código y desperdicio de espacio.

El mapa de memoria de un segmento típico de **FORTH** tiene cuatro elementos principales (figura 4): el diccionario, el cual crece hacia adelante a partir de localidades bajas de memoria; las pilas de parámetros y de retorno (parameter and return stack), que crecen hacia abajo desde localidades altas de memoria; un pequeño kernel, que reside abajo del diccionario y que contiene el código de inicialización, algunas rutinas en lenguaje de máquina y tablas del sistema y dos o más áreas de almacenamiento temporal para bloques (un bloque es la unidad básica de almacenamiento en disco de **FORTH**). La estructura del diccionario es la parte más grande e importante de **FORTH**, ya que es la que determina como funciona el lenguaje.

El diccionario y la pila de parámetros son los extremos opuestos de memoria disponible, permitiéndose así cada uno crecer sin interferir con el otro. La forma compilada de las palabras de **FORTH** dentro del diccionario es un buen punto de partida para comprender el proceso de compilación y de ejecución de **FORTH**.



Figura 4. Mapa de memoria de FORTH

La figura 5 muestra el formato en el cual todas las palabras son guardadas en memoria. Una palabra compilada consiste de cuatro campos -nombre, enlace, código y parámetros- cada uno ubicado en las localidades progresivas más altas. Los dos primeros campos en conjunto son llamados el encabezado de la definición; los siguientes dos constituyen el cuerpo.

El encabezado permite a **FORTH** localizar e identificar una palabra antes de su ejecución o compilación dentro de otra palabra. El cuerpo mantiene la información que determina lo que la palabra hará cuando sea ejecutada.

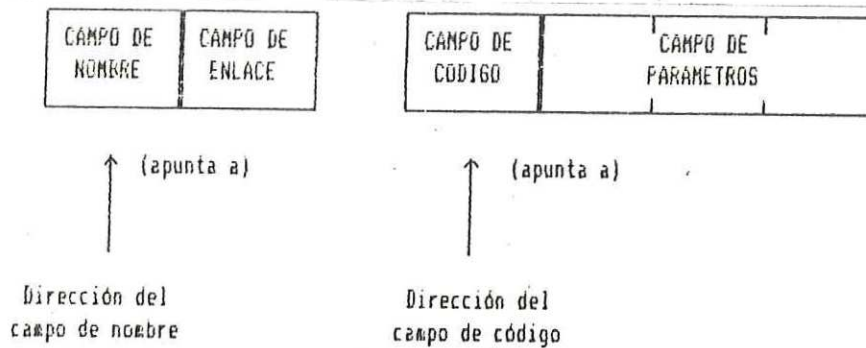


Figura 5. Formato de la palabra en FORTH

El campo de código de FORTH guarda la dirección de la subrutina en lenguaje de máquina para una clase de palabra en particular. El campo de parámetros guarda los datos por ser procesados por tal subrutina. A menudo, tales datos son direcciones a otras palabras que son ejecutadas.

El campo de nombre contiene el nombre de la palabra, que es el identificador para la búsqueda en el diccionario. **FORTH** permite el uso de cualquier carácter ASCII a excepción del espacio (que es un delimitador), un retorno de carro o la alimentación de línea para formar el nombre. Esto puede cambiar en diversos dialectos o implantaciones.

El siguiente campo es el de enlace, el cual contiene la dirección del campo de nombre (NFA, Name Field Address) de la anterior palabra definida que se encuentra en el diccionario. Este campo determina el orden de búsqueda. Cuando una palabra es referida, se inicia la búsqueda de su definición a partir de la cima del diccionario (la última definición en éste). Esta es una búsqueda en una lista ligada, desde las localidades altas de memoria a las bajas.

El cuerpo de la palabra es un mini-programa que contiene información del procedimiento y de los datos. El campo de código apunta al procedimiento, que es una subrutina en lenguaje de máquina llamada también la primitiva de ejecución. El campo de parámetros contiene los datos y su estructura varía de palabra en palabra (ver figura 6). La dirección a la cual el campo de código apunta, - es decir, la primitiva en lenguaje de máquina que debe ser ejecutada - depende de la palabra de definición empleada para crear la palabra. Esto es, la primitiva de ejecución de todas las palabras definidas a través de : (dos puntos) será la misma pero será diferente que la usada para las palabras definidas usando VARIABLE.

El campo de parámetros guarda los datos bajo los cuales operará la primitiva de ejecución. En definiciones de dos puntos, los datos son en realidad una lista de direcciones de las palabras usadas para definir la nueva palabra, que terminan con la dirección de una palabra especial llamada EXIT que concluye la ejecución de la lista. La compilación de una definición de dos puntos, por lo tanto, consiste en colocar las direcciones de las palabras usadas en la definición secuencialmente dentro del campo de parámetros de la nueva palabra. La primitiva de ejecución controla la ejecución de la lista de direcciones guardadas en el campo de parámetros.

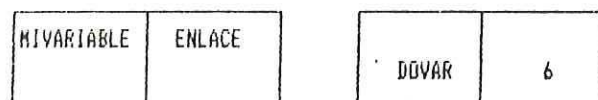
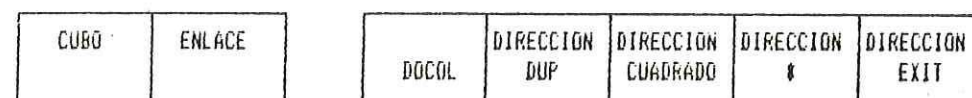
La figura 6 muestra el contenido de una de las anteriores palabras definidas como ejemplo: CUBO. Su campo de parámetros mantiene las direcciones de los campos de código de DUP, CUADRADO, y *. Esto es, para definiciones de dos puntos, las direcciones guardadas en el campo de parámetros de la palabra determina la secuencia hilada de ejecución, y en cierto modo, el código apuntado a cada dirección de campo de código es un mini-intérprete que actúa sobre el campo de parámetros. En contraste, para una palabra creada usando VARIABLE, el campo de parámetros simplemente contiene el valor almacenado en la variable, y el campo de código apunta a la dirección de una rutina rudimentaria en lenguaje de máquina que regresa la dirección del campo de parámetros.

El contenido de los campos de código y parámetros de una palabra definida por el vocabulario de ensamblador de **FORTH** merece atención. El campo de parámetros contiene la rutina en lenguaje de máquina que será ejecutada cuando la palabra sea invocada. Debido a que el contenido del campo de código de cualquier palabra apunta al código de máquina que será ejecutado por la palabra (en el caso de una definición por medio de CODE), el apuntador simplemente apunta al inicio del campo de parámetros.

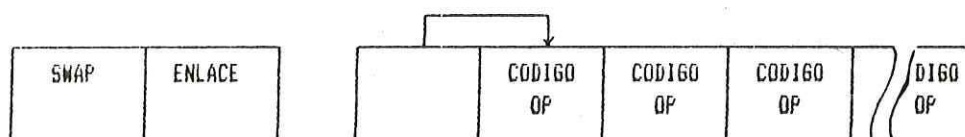
PALABRA PADRE

: (DOS PUNTOS)

VARIABLE



CODE



CONSTANT

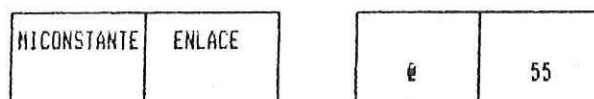


Figura 6. Estructura de varias palabras hijo

Las diferentes palabras de definición producen diferentes formatos de palabras. La palabra de definición determina la estructura del campo de parámetros y la subrutina a la cual apunta la dirección del campo de código.

FORTH proporciona un ambiente dinámico en el cual aún las formas más fundamentales pueden ser redefinidas. Un programador que necesite ocuparse superficialmente con las propiedades básicas del lenguaje debe, sin embargo,

comprender la estructura de tres niveles de las palabras de **FORTH** (figura 7).

Las palabras-hijo tales como CUBO y CUADRADO son el tipo más común de palabras. Debido a que **:** fue empleada para definirlas, ésta es su palabra-padre, y como tienen el mismo padre, son palabras-hermanas. Las palabras-hermanas tienen la misma primitiva de ejecución y la misma estructura de campo de parámetros. La primitiva para **:** se denomina DOCOL.

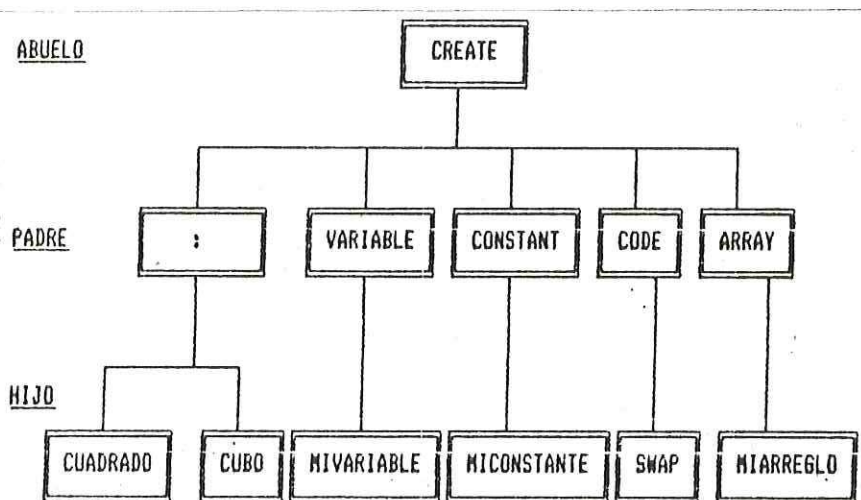


Figura 7. Jerarquía de las palabras en FORTH

El mecanismo para redefinición de formas fundamentales (palabras-padre) es CREATE, la cual puede construir nuevas palabras-padre. Esto es, CREATE la cual es una palabra de definición, se denomina palabra-abuelo. Cuando CREATE forma una nueva palabra, construye un campo de parámetros como se

especifica en la definición de la palabra y enlaza la nueva palabra a una primitiva de ejecución adecuada. La definición de la palabra-padre VARIABLE es:

: VARIABLE CREATE 2 ALLOT ;

Cuando VARIABLE se ejecuta para crear la palabra-hijo, CREATE analiza la siguiente cuerda (MIVARIABLE en el ejemplo anterior) en el campo de nombre. Esto también crea un campo de código que contiene un apuntador a una rutina que regresa la dirección del campo de parámetros a la pila (esta rutina a menudo se denomina DOVAR). 2 ALLOT asigna un espacio de dos octetos en el campo de parámetros donde se conservará el valor de la variable. Ahora, la palabra creada con VARIABLE, y secundariamente con CREATE, coloca la dirección del campo de parámetros en la pila, como se hizo con MIVARIABLE en el ejemplo anterior. Todos los hijos de VARIABLE son hermanos: sus campos de código apuntan a DOVAR.

Por si misma, CREATE meramente especifica una estructura de parámetro y una acción por omisión de la palabra-hijo, DOVAR. Cuando se combina con la palabra DOES>, CREATE puede especificar la acción de la palabra-hijo, como se ilustra en la definición de la palabra-padre CONSTANT. CONSTANT es una palabra de definición, la cual en lugar de regresar la dirección del campo de parámetros, regresa el contenido de este campo. CONSTANT espera encontrar un número en la pila que, a su vez, será puesto en el campo de parámetros de la

nueva palabra. Esto es

55 CONSTANT MICONSTANTE

define MICONSTANTE tal que cuando se ejecute ponga el 55 dentro de la pila.

Probablemente CONSTANT debería definirse usando CODE por velocidad de ejecución, pero puede hacerse en FORTH de alto nivel como:

```
: CONSTANT CREATE , DOES> @ ;
```

Cuando CONSTANT se ejecuta para crear la palabra-hijo, CREATE actúa como lo hizo con VARIABLE, creando un encabezado y un campo de código. Por omisión, CREATE siempre apunta al campo de código de DOVAR. La palabra, entonces asigna dos octetos para el campo de parámetros y pone en éste el número que se encuentra en la cima de la pila.

Pero CONSTANT debe decir al hijo que busque el número en el campo de parámetros y lo ponga en la pila. DOES> realiza esto por CONSTANT. Especifica la acción del tiempo de corrida para la palabra-hijo, en este caso buscar y traer el valor almacenado en el campo de parámetros. DOES> compila las palabras que le siguen dentro de la definición de la palabra-padre, tal como si ellas fueran palabras en una definición de dos puntos; después coloca la dirección de una secuencia diferente de código ejecutable dentro del campo de código de la palabra-hijo, sustituyendo la dirección (DOVAR) puesta allí por CREATE.

La nueva secuencia de código coloca la dirección del campo de parámetros en la pila (justo donde DOVAR lo hizo), luego ejecuta las palabras compiladas después de DOES> en la palabra-padre como si ellas estuvieran en una definición de dos puntos. El resultado neto es que MICONSTANTE primero coloca su dirección del campo de parámetros en la pila, luego @ (busca y trae) el número almacenado en el campo de parámetros a la pila. Simplemente, las palabras entre CREATE y DOES> se ejecutan cuando la palabra de definición (en este caso, CONSTANT), se ejecuta; las palabras que siguen a DOES> se ejecutan cuando lo hace la palabra-hijo.

En otro ejemplo, la palabra de definición ARRAY se redefine para crear palabras-hijo que son arreglos lineales (vectores) tales que la dirección de un elemento en el arreglo será regresado a la pila si la palabra-hijo se presenta con el número del elemento que ha de ser devuelto. ARRAY debería presentarse con el número de elementos en el arreglo. Esto es

10 ARRAY MIARREGLO

crearía la palabra MIARREGLO.

3925 5 MIARREGLO !

almacenaría 3925 en el elemento 5 del arreglo lineal MIARREGLO, y

5 MIARREGLO @ .

desplegaría en el monitor el contenido del elemento 5, o sea 3925. La definición de ARRAY es:

```
: ARRAY CREATE 2 * ALLOT DOES> SWAP 2 * + :
```

Cuando se ejecuta ARRAY para crear MIARREGLO, se ejecuta el 2 * ALLOT, asignando 2 veces 10, o 20, octetos (el espacio para 10 enteros) en el campo de parámetros de MIARREGLO. Cuando se ejecuta MIARREGLO, la dirección de su campo de parámetros se coloca primero en la pila, luego

```
SWAP 2 * +
```

se ejecuta, calculando la dirección de los elementos y dejándola en la pila.

La construcción CREATE... DOES> se considera una de las herramientas más poderosas de FORTH debido a que crea palabras-padre que definen palabras-hijo con categorías completamente nuevas de acción. Aún más poderosa, sin embargo, es la palabra ;CODE, la cual se emplea como DOES>, pero es seguida de mnemónicos de lenguaje ensamblador, permitiendo que la acción de las palabras-hijo se definan en lenguaje de máquina.

El diccionario de FORTH, por lo tanto, consiste de una lista ligada de nombres de palabras. Cada nombre se asocia con un campo de código que contiene un apuntador a la secuencia de código de máquina que realiza la acción de la palabra. La lista también está asociada con un campo de parámetros que guarda cierto tipo de datos que determina el

comportamiento exacto de la palabra.

El intérprete de **FORTH** es a la vez responsable de la ejecución de las subrutinas representadas por las palabras y de la compilación de las nuevas palabras.

II.2 El intérprete externo

Un intérprete traduce código fuente a una forma ejecutable y luego inmediatamente realiza las acciones indicadas. En Forth este proceso tiene dos diferentes fases:

- 1) la localización de las palabras en el diccionario y
- 2) el control de su ejecución o compilación de una manera ordenada.

La localización de las palabras y la decisión de si debe ejecutarse o compilarse es responsabilidad del intérprete externo.

La acción del intérprete externo es inicializada, en la mayoría de los dialectos, por **INTERPRET**; pero ya sea invocada por una palabra específica o no, el proceso es similar. La operación de **INTERPRET** (figura 8) fué parcialmente revelada en la descripción de la estructura del diccionario. Primero analiza la cuerda de entrada, luego busca la cuerda en el diccionario. Si la búsqueda es exitosa, la dirección del campo de código se coloca en la pila (realizada por la

palabra FIND).

Si la cuerda no fue encontrada (la colocación de un 0 en la pila por la palabra FIND), INTERPRET prueba la cuerda como un número válido. Si la cuerda es un número, INTERPRET lo convierte de su forma en ASCII a binaria, y lo coloca en la pila (la conversión la realiza a través de la palabra NUMBER). Cuando la cuerda de caracteres es una palabra o un número, el siguiente paso depende de si FORTH está en modo de compilación o en modo de ejecución. En cualquier caso, INTERPRET se ejecuta hasta que se agota la cuerda de entrada. Si la cuerda de caracteres no es ni una palabra ni un número, el intérprete externo aborta la acción, es decir, ignora el resto de la cuerda de entrada, vacía la pila y genera un mensaje de error.

El intérprete externo sabe cuando FORTH está en modo de compilación o en modo de ejecución por el valor que tiene una variable del sistema denominada STATE. Si STATE @ regresa a la pila un valor diferente de cero, se realiza la compilación; si se regresa 0, FIND busca la palabra. Si FIND es exitosa, la palabra asociada con la dirección del campo de código que se devuelve se ejecuta.

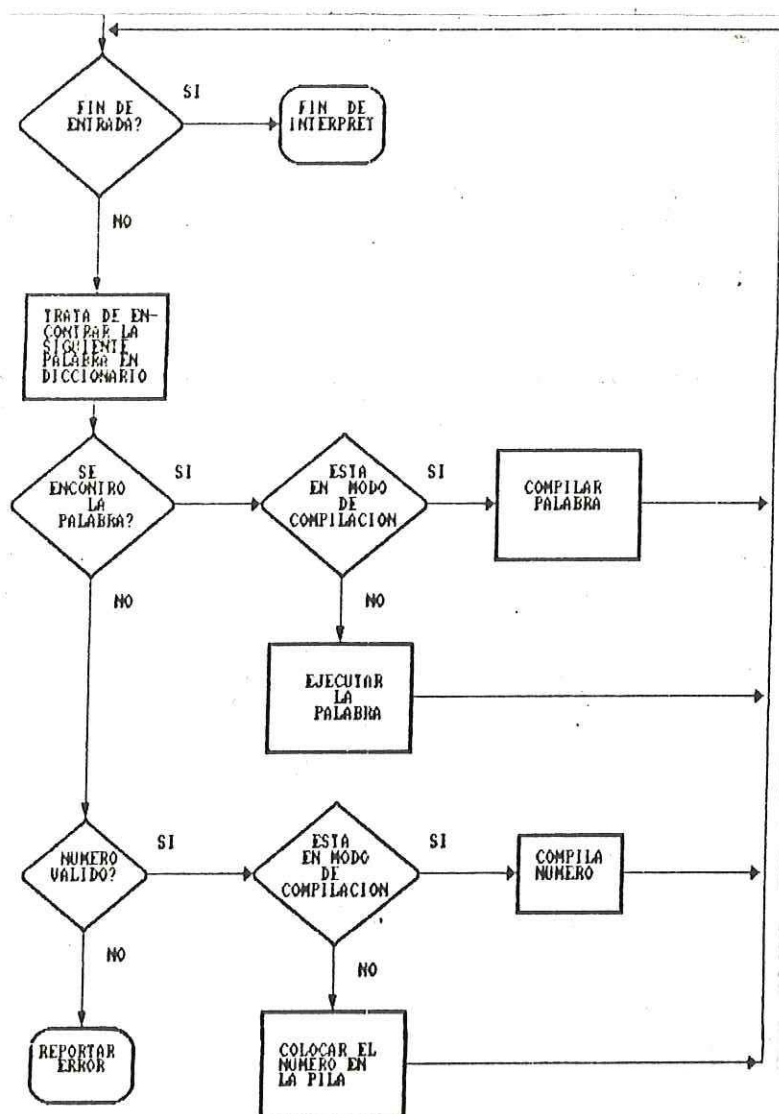


Figura 8. El Intérprete Externo

Durante este proceso, la palabra EXECUTE ejecuta la palabra asociada con la dirección del campo de código que se encuentra en la cima de la pila. En la norma FORTH-83 (Forth Standards Team (1983)), la palabra ' (tick) regresa la dirección del campo de código a la pila. En la norma 79, FIND hace esta tarea. Si

' DUP .

se tecleara y la dirección 15CH fuera desplegada en el monitor, al teclear

5 15C EXECUTE . .

o

5 ' DUP EXECUTE . .

o

5 DUP . .

todas desplegarán el mismo resultado. Cada una desplegará 5 5 - esto es, 5 se duplica y se despliega dos veces-. El flujo de ejecución entre las palabras en la definición se controla por el intérprete interno. Un ejemplo usando

: CUADRADO DUP * ;

y

: CUBO DUP CUADRADO * ;

ilustra mejor como EXECUTE invoca al intérprete interno. Cuando EXECUTE ejecuta CUBO, empieza a seguir el hilo de la definición (mostrada en la figura 9). La ejecución de las primitivas (DUP y *) se realiza automáticamente por el intérprete interno y no involucra EXECUTE. El intérprete eventualmente regresa a INTERPRET.

FORTH mantiene la pista de todas las rutas hiladas a través del apuntador de instrucciones (IP). Este registra el progreso del intérprete a través de la palabra actual en FORTH. Cuando una palabra invoca a otra, el antiguo apuntador de instrucciones se coloca en la pila de retorno y el IP se actualiza para reflejar la posición de la nueva palabra.

Específicamente, considere como un ejemplo el caso de una definición de dos puntos. Cada vez que se llama a una palabra en Forth dentro de otra, la dirección de la CFA (Code Field Address, dirección del campo de código) de la siguiente palabra en el campo de parámetros de la palabra invocadora se guarda en la pila de retorno y se inicia la ejecución de la palabra invocada. El apuntador de instrucción IP controla el progreso de la ejecución dentro de una palabra de dos puntos, indicando el siguiente elemento del campo de parámetros por procesar a menos que la palabra invocada sea una palabra de dos puntos. Para describir los apuntadores de las rutas hiladas de otra manera, la ruta de ejecución dentro del campo de parámetros se rastrea a través del apuntador de instrucción. Pero cuando una definición de dos puntos llama a otra, el IP debe rastrear los contenidos de los campos de parámetros en la palabra que se invoca. La localidad a la cual debe regresar en la palabra invocadora se coloca en la pila de retorno.

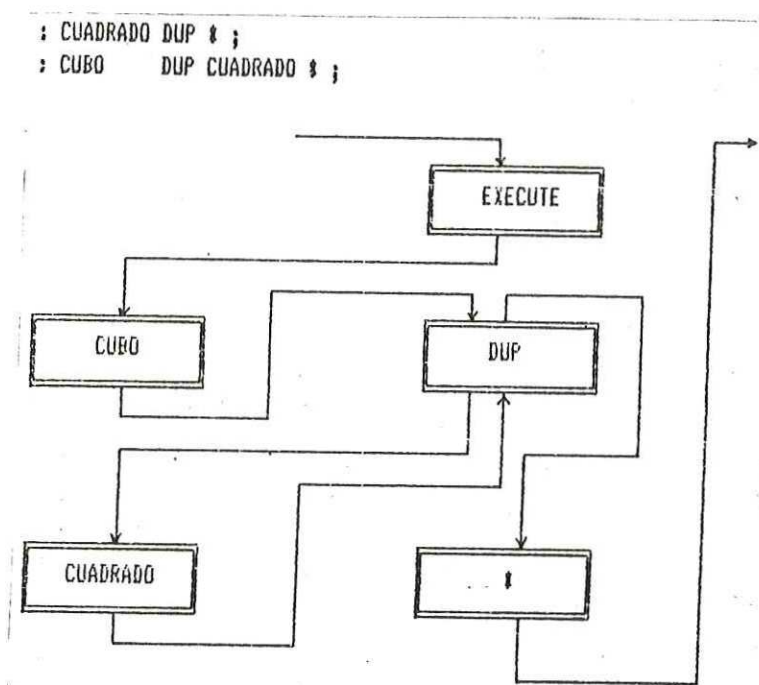


Figura 9. Ejecución Hilada

Ahora podemos seguir la ejecución del ejemplo: Cuando CUBO se ejecuta, la subrutina apuntada por la dirección del campo de código se ejecuta (como se mencionó anteriormente, este código de máquina a menudo se llama DOCOL -por DO COLon , hacer definición dos puntos-). La primera acción de DOCOL es buscar la siguiente dirección que será ejecutada en INTERPRET (la cual está en el IP) y colocarla en la pila de retorno. Esto asegurará el regreso al intérprete externo cuando se termine la ejecución ya que en ese momento será sacada de la pila. Después usa la rutina llamada NEXT para

apuntar el IP a la primera dirección del campo de código en el campo de parámetros en CUBO, la cual es DUP (una primitiva), y la ejecución se transfiere a la dirección apuntada por tal dirección del campo de código; la dirección del campo de código de DUP apunta al código de máquina que será ejecutado.

Todas las primitivas de ejecución terminan con NEXT, la cual incrementa el valor de la dirección en IP en dos octetos, apuntando a la siguiente dirección del campo de código en el campo de parámetros de la palabra invocadora. En este caso, la dirección del campo de código de CUADRADO.

La ejecución se transfiere de nuevo a DOCOL, esta vez desde la dirección del campo de código de CUADRADO. NEXT primero suma dos octetos a la dirección en IP y coloca la dirección resultante en la pila de retorno (la siguiente dirección del campo de código en CUBO que será cuando CUADRADO termine), luego apunta el IP a la primera dirección del campo de código en el campo de parámetros de CUADRADO, que es DUP. DUP se ejecuta de nuevo, con NEXT incrementando el IP a *, la cual se ejecuta con su NEXT apuntando a la última dirección del campo de código en el campo de parámetros en CUADRADO.

La última dirección del campo de código en cada definición de dos puntos (la que coloca *i*) es la de EXIT.

Esta palabra saca de la pila de retorno el valor colocado en la cima, la cual en este caso es la dirección del campo de código de la siguiente palabra a ejecutarse en el campo de parámetros en CUBO (que es *), puesto ahí por el DOCOL de CUADRADO. Cuando se encuentra el EXIT de CUADRADO, los dos elementos en la cima de la pila de retorno contienen la siguiente dirección del campo de código en el campo de parámetros en CUBO que está siendo ejecutada, y la dirección que será ejecutada en seguida por INTERPRET. EXIT regresa a la palabra (que es apuntada por la pila de retorno) que invocó la palabra que se termina. Ahora se ejecuta * en CUBO, con NEXT apuntando a la salida de CUBO; este entonces saca de la pila de retorno la dirección que mantiene la ruta hilada hacia INTERPRET, que termina así la ejecución de CUBO y regresa al intérprete externo.

II.3 La compilación de Forth

La operación de compilación puede dividirse en dos partes: 1) la producción del encabezado, la cual debe hacerse para toda palabra definida, y 2) la producción del cuerpo, los detalles en los cuales varía de palabra en palabra.

La parte del encabezado es simple, pero dependiente del dialecto. La producción del campo de código depende de la palabra de definición usada. El proceso más interesante es sin duda la compilación del campo de parámetros de una

definición de dos puntos, la cual consiste simplemente en la creación de un campo de código para la nueva palabra que contiene la dirección de DOCOL y el campo de parámetros que contiene las direcciones de los campos de código de las palabras en la definición.

La compilación de Forth siempre toma lugar en la cima del diccionario (Forth tiene un compilador incremental). La dirección en la cima del diccionario se guarda en una localidad en memoria y el contenido de ella se coloca en la pila por la palabra HERE. HERE es una palabra que mantiene un apuntador a la cima del diccionario. Este es un apuntador interno que se incrementa. Si se escribe:

10 ALLOT

incrementaría HERE en 10 octetos. Como mencionamos anteriormente, la palabra `_` copia el número en la pila a la cima del diccionario e incrementa HERE en 2. Se puede definir como

: , HERE ! 2 ALLOT ;

después de que se instala el encabezado para una definición de dos puntos, HERE apunta a donde debe ir el campo de código. El campo de código se producirá con una construcción tal como

' DOCOL 2 + ,

Recuerde que `_` regresa la dirección del campo de código de la palabra, en este caso la dirección de DOCOL. Luego, `2 +`

calcula la dirección del campo de parámetros de DOCOL (dos octetos más que la dirección del campo de código). Finalmente, deposita la dirección del campo de parámetros en HERE e incrementa HERE en 2. El campo de código de la palabra de dos puntos ahora contiene un apuntador a la rutina en lenguaje de máquina de DOCOL. Todo lo que queda es colocar las direcciones de los campos de código de las palabras usadas en la definición del código fuente de la palabra de dos puntos dentro del campo de parámetros de su entrada al diccionario. Esto es complementado por el intérprete externo cuando analiza las palabras de la definición en el código fuente.

INTERPRET verifica la cuerda de caracteres en el diccionario. Si está presente, su dirección del campo de código se almacena en la pila. Si STATE @ regresa un valor diferente de cero -es decir, FORTH está en modo de compilación- INTERPRET ejecuta la palabra, y conserva su dirección del campo de código en HERE. Esto se repite hasta que todas las palabras en la definición se compilen como direcciones del campo de código en el campo de parámetros y hasta que se encuentra un ; que compila la dirección del campo de código de EXIT. (Si STATE @ devuelve 0, FORTH está en modo de ejecución, e INTERPRET usa EXECUTE para ejecutar la palabra en lugar de compilarla).

Para aclarar el proceso de compilación considere la porción de INTERPRET que realiza tales acciones. Por cada palabra de la definición que se analiza, INTERPRET ejecuta lo siguiente:

STATE @ IF , ELSE EXECUTE THEN

(La construcción IF xxx ELSE yyy THEN zzz permite ejecutar la instrucción xxx si el valor en la pila es diferente de cero, ejecutar la instrucción yyy si es cero; y ejecutar zzz después de que alguna de ellas se haya realizado). Cuando IF encuentra un valor no cero devuelto por STATE @ (Forth está en modo de compilación), se ejecuta la palabra `.`. Si se devuelve cero, se ejecuta EXECUTE que se encuentra entre el ELSE y el THEN, ejecutándose la palabra. Esto es, el compilador para las palabras de dos puntos es parte de la definición de INTERPRET y es muy simple.

Pero qué pasa si se devuelve un número a la pila por INTERPRET, es decir la cuerda analizada es un número. El número debe ser compilado pero de tal manera que sea devuelto a la pila durante la ejecución de la palabra de dos puntos (el número no puede ser tratado como una dirección del campo de código). Esta es una tarea que realiza LITERAL, que se define como:

: LITERAL STATE @ IF COMPILE LIT , THEN ; IMMEDIATE

LITERAL brinda la capacidad de retrazar la ejecución, una propiedad que se regula en su definición tanto por IMMEDIATE

como por COMPILE. IMMEDIATE fuerza la ejecución de una palabra que de otra manera debería compilarse, en este caso, LITERAL. Sin la especificación de IMMEDIATE en su definición, LITERAL solamente compilaría dentro de la definición antes del número. Pero esto no es suficiente; LITERAL debe llevarse a cabo al tiempo de compilación para procesar el número. Es posible extender esta propiedad inmediata con la palabra COMPILE, que fuerza que la palabra inmediata LIT sea compilada dentro de la definición de la palabra-hijo de LITERAL.

Cuando se ejecuta LITERAL, ésta verifica si STATE indica que FORTH está en modo de compilación. Si es así, COMPILE LIT compila la dirección del campo de código de LIT en HERE. LIT es una primitiva que busca y trae los dos octetos que siguen a su dirección del campo de código compilado (el número compilado por la palabra `┘`) a la pila e incrementa el IP en dos, causando que el número compilado sea omitido por el intérprete interno. LITERAL es un ejemplo de una palabra compilada, como lo son las palabras `┘` y ALLOT. Por ello, un número se compila dentro del diccionario como cuatro octetos; los primeros dos son la dirección del campo de código de LIT, los cuales buscan los siguientes dos octetos- el número solo- y lo colocan en la pila.

El ambiente presentado anteriormente brinda al programador un gran poder de expresión. Es tan expresivo que las construcciones que el usuario esperaría encontrar a nivel

más interno del sistema Forth pueden definirse (o redefinirse) por el usuario. Considere la estructura de Forth para el salto condicional, IF..ELSE..THEN. Cuando IF ve un valor diferente de cero en la pila (considerado como una condición verdadera), se ejecutan las palabras entre IF y ELSE. Si el número es cero (falso), las palabras entre IF y ELSE se omiten y las que están entre ELSE y THEN se ejecutan. La ejecución continua en cualquier caso con las palabras después de THEN.

Por ejemplo, las siguientes secciones de código verifican que el número en la cima de la pila sea par o impar al dividirlo entre dos. En Pascal, este procedimiento se escribiría:

```
procedure par_impar(i:integer);
begin
  if i mod 2 <> 0 then writeln('impar')
    else writeln('par');
end;
```

En FORTH se escribiría:

```
PAR/IMPAR? 2 MOD IF ." Impar" ELSE ." Par" THEN ;
```

La operación 2 MOD (módulo) deja el residuo de la división entre 2 en la pila. ." analiza lo que sigue en la cuerda de entrada hasta encontrar " y despliega en el monitor

la cuerda de caracteres formada.

IF es una palabra inmediata que, en lugar de compilarse (guardando la dirección de su campo de código en el diccionario), implanta la estructura de salto condicional. **IF** compila la CFA de **OBRANCH**, una primitiva que hace el salto. Si la pila guarda un 0, la acción de **OBRANCH** es buscar el siguiente número en el campo de parámetros e incrementar el IP de tal manera que salte sobre ese número de CFA; salta a un punto después de la CFA de la palabra que inmediatamente precede a **ELSE** o a **THEN**. Si un valor diferente de cero esta en la pila, **OBRANCH** salta un número y sigue en la siguiente CFA. Después de compilar **OBRANCH**, **IF** pone a **HERE** en la pila y deja espacio para **ELSE** o **THEN** al colocar el número de CFA que será omitida al compilar un cero, usando la palabra **┘**. Una posible definición de **IF** sería como sigue:

```
: IF ?COMP COMPILE OBRANCH HERE 0, ; IMMEDIATE
```

?COMP aborta con un mensaje de error si Forth no está en modo de compilación. Sería definida antes de **IF** como

```
: ?COMP STATE @ 0= ABORT" Solo compilacion!" ;
```

0= es un operador de comparación que devuelve un valor verdadero (1 o -1, dependiendo del estándar de Forth empleado) y **ABORT"** aborta después de desplegar el mensaje de la cuerda de caracteres formada hasta encontrar ". Después de que **IF** se ejecuta, **HERE** se deja en la pila y las subsecuentes palabras se compilan normalmente hasta **THEN**, la cual puede

definirse como:

: THEN ?COMP HERE OVER - SWAP ! ; IMMEDIATE

HERE coloca la dirección de la cima del diccionario, después de la compilación de las palabras entre IF y THEN, en la pila. OVER copia el segundo número en la pila (la dirección que regresó HERE en la ejecución de IF) a la cima de la pila; - (menos) calcula que tan lejos de la cima del diccionario se movió, el cual es el número de octetos que debe brincarse por OBRANCH; y SWAP ! guarda la cuenta en la localidad reservada para el 0, en IF. THEN no compila nada, simplemente termina la acción de IF.

Los servicios de la compilación de IF..THEN pueden ser más claros considerando el contenido de un campo de parámetros de una palabra muy simple, por ejemplo

: PRUEBA IF . THEN ;

El primer elemento en el campo de parámetros será la CFA de OBRANCH. Esta es seguida de 4, el número de octetos que OBRANCH debe brincarse si ve un valor falso (debe brincar el número 4 mismo y la CFA de que le sigue). Los únicos elementos compilados por IF y THEN fueron OBRANCH y el número de octetos que debe brincar éste.

ELSE es similar a la combinación de IF y THEN, pero usa BRANCH en lugar de OBRANCH (BRANCH es el salto incondicional); su definición es

: ELSE ?COMP COMPILE BRANCH HERE 0 , SWAP HERE OVER - SWAP !

; IMMEDIATE

Algunos dialectos pueden usar diferentes procedimientos,
pero el principio es el mismo.

III IMPLANTACION DE UN LENGUAJE

III.1 Introducción

Entre un lenguaje de alto nivel y la correspondiente arquitectura de máquina en la cual se apoyará existe una gran diversidad de técnicas de implantación de lenguajes que van desde las ya clásicas de interpretación y compilación hasta las más recientes de lenguaje intermedio y código hilado. El criterio de selección de la técnica más adecuada para implantar un lenguaje estará basado en factores importantes propios del desempeño que se desea, como lo son: la velocidad de ejecución, el almacenamiento de programas y el uso de los recursos del procesador, además del tamaño del código generado y el tamaño de la propia implantación. Así, las técnicas de implantación de un lenguaje pueden dividirse lógicamente en dos categorías: traducción e interpretación.

En las técnicas de traducción se reemplazan elementos de la sintaxis de alto nivel con instrucciones de bajo nivel que realizan una operación equivalente. La traducción resultante se ejecuta como resultado final del proceso. Bajo esta técnica se encuentran los compiladores y ensambladores que tradicionalmente traducen sus entradas a código a nivel de máquina.

Las técnicas de interpretación ejecutan directamente los programas en lenguaje de alto nivel. El intérprete es un

programa que ve al programa fuente de alto nivel como una serie de códigos de operación que guían su ejecución. El sistema interpretativo aparece al usuario como una máquina virtual (es decir, una máquina ficticia con determinadas características) que tiene la arquitectura del lenguaje de alto nivel.

En algunas ocasiones, la técnica de implantación seleccionada es una combinación de las expuestas con anterioridad, utilizando lo que se conoce como lenguaje intermedio, bajo el cual se traducen programas en lenguaje de alto nivel a un lenguaje que es fácil de tratar con él y sintácticamente diferente del original, lo que permite emplear técnicas de optimización de código. Ya que la traducción a lenguaje intermedio es independiente de la máquina sobre la que se va a utilizar, varios compiladores diferentes necesitan producir solo el código más simple del lenguaje intermedio para una misma máquina. Así, este programa en lenguaje intermedio puede ser utilizado por un generador de código para obtener el mismo programa objeto que produciría un compilador, o ser interpretado en lugar de realizarse tal traducción, lo cual implica la utilización de rutinas de lenguaje de máquina complejas y poderosas para minimizar el tiempo de interpretación, pero la independencia de la máquina se complementa mejor teniendo rutinas en lenguaje de máquina simples y fáciles de escribir. Esta misma

combinación entre la independencia de la máquina y la velocidad de ejecución debe considerarse en el diseño de cualquier lenguaje intermedio.

Cualquiera que sea el tratamiento que se siga para manipular un programa en lenguaje intermedio, esta técnica ofrece la ventaja de la independencia de la máquina respecto al lenguaje fuente y permite la transportabilidad del programa (la habilidad de ejecutar el mismo programa fuente en computadores completamente diferentes).

Un lenguaje intermedio está constituido por un conjunto de operaciones primitivas (las cuales en combinación, pueden expresar un algoritmo) y de operaciones de almacenamiento tanto para datos como del programa. El programa en lenguaje intermedio (LI) puede usar un número fijo de localidades de memoria para simular registros de propósito general, pero entonces se necesitan rutinas para cargar (y almacenar) cada registro de la memoria, así como también rutinas que simplemente muevan valores entre registros. Si el lenguaje intermedio se enfoca a la complejidad del lenguaje de máquina original, su uso es de dudoso valor.

Un enfoque que simplifica un conjunto de instrucciones es la arquitectura de "dirección cero" o de pila. En esta arquitectura todas las operaciones obtendrán valores de la pila y los resultados se devolverán colocándolos en ella. Se

necesitan solo dos operaciones con la memoria: el sacar de la pila y almacenar (en memoria), y la operación de cargar de memoria y colocar en la pila. Diseñando una arquitectura de dirección cero en el lenguaje intermedio, la localidad de transferencia de parámetros es implícita y no necesita ser parte de la representación del propio lenguaje intermedio.

III.2 Código hilado

III.2.1 Descripción

El código hilado es una técnica de implantación de lenguaje intermedio que organiza el control de flujo de programas como una secuencia de invocaciones a subrutinas. Ningún otro aspecto se representa con código hilado. El código hilado se aplica especialmente a la interpretación. El proceso de interpretación consiste en transferir el control a las rutinas seleccionadas por los códigos de operación del código hilado. Las funciones disponibles en el lenguaje intermedio se proporcionan por las subrutinas que se invocan y no son sólo parte inherente del código hilado.

Los lenguajes intermedios de código hilado son aplicados especialmente a la implantación de máquinas virtuales incorporadas en arquitecturas de dirección cero. Así, la técnica de usar código hilado para implantar un lenguaje puede usarse por ejemplo para implantar Pascal (usando el lenguaje intermedio P-code (Aho ,A. et al (1986)),

intérpretes de LISP o, por supuesto, FORTH. Podemos clasificar cuatro tipos de código hilado: subrutina, directo, indirecto y por señal.

Todas las variedades de código hilado consisten en una estructura de datos que es una secuencia de identificadores únicos de subrutinas. Tradicionalmente, el código hilado se ha mantenido cercano al nivel de máquina y ha incluido apuntadores reales a las subrutinas (las cuales pueden ser escritas en lenguaje intermedio o en código de máquina). También tradicionalmente, una parte de los recursos del procesador -en particular los registros del procesador- se han dedicado al uso del intérprete del código hilado. Como debemos ver, no se necesitan ni apuntadores absolutos ni recursos de registros para implantar un código hilado.

III.2.2 Código hilado subrutina

Una secuencia de llamadas a subrutinas sin contener ninguna instrucción implanta un lenguaje intermedio. Cada llamada a subrutina puede considerarse una operación simple de lenguaje intermedio, la cual no necesita estar relacionada a la arquitectura de la máquina empleada. El código hilado subrutina (CHS) es un mecanismo de control que se maneja con facilidad a nivel de circuitería de la máquina.

La organización del programa consistente en llamadas a subrutinas raramente se usa por programadores pero algunas veces es empleada por compiladores. Es el lenguaje intermedio

más general posible y conserva las ventajas de independencia de máquina al no generar lenguaje de máquina en línea (la diferencia en la forma de instrucciones de llamado y regreso de subrutinas en varios computadores es usualmente trivial).

El código hilado subrutina incurrirá en menor tiempo de ejecución que la mayoría de los lenguajes intermedios ya que su interpretación se maneja mecánicamente en vez de una secuencia de instrucciones. Aún más, CHS puede ser optimizado usando código de máquina en línea para operaciones donde es excesivo el gasto en subrutinas, una ventaja que no se puede obtener en otros tipos de códigos hilados. Por supuesto que el código optimizado ya no es independiente de la máquina; el paso adicional de traducción convierte al lenguaje intermedio en código objeto para una máquina en particular. Ver figura 10.

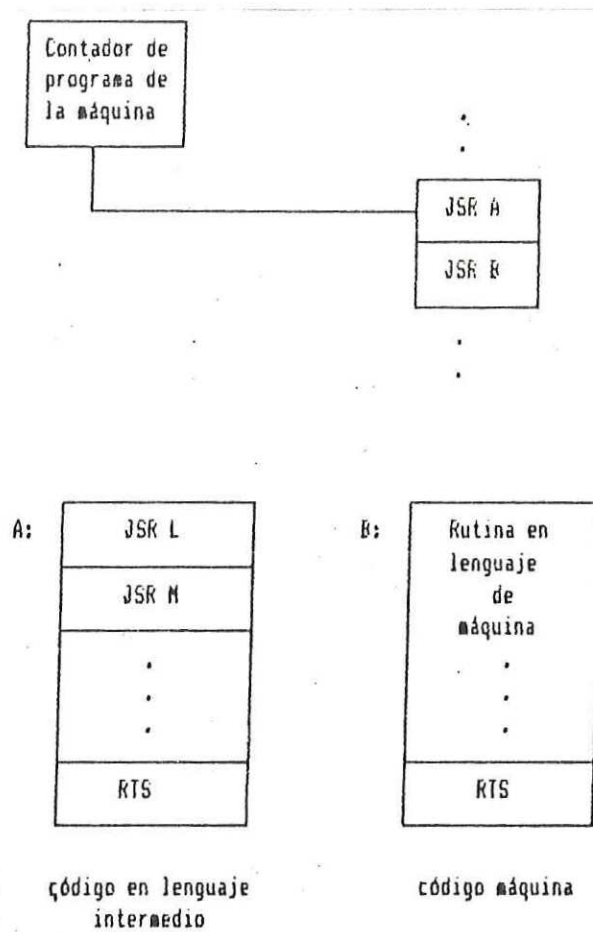


Figura 10. Código Hilado Subrutina

III.2.3 Código hilado directo

El código hilado directo (CHD) puede considerarse como una secuencia de llamadas a subrutinas en lenguaje de máquina con el código de operación de invocación (CALL) eliminado. Esto resulta en una lista de direcciones, cada una de las cuales apunta a una subrutina en lenguaje de máquina. Ya que los programas hilados directos no incluyen códigos de operación, debe escribirse un programa corto en lenguaje de máquina para leer la siguiente dirección en la lista y transferir el control a dicha dirección. Las implantaciones tradicionales de código hilado directo no permiten el uso de verdaderas subrutinas a nivel de máquina sino que requieren que cada subrutina termine ejecutando la operación NEXT.

Con el objeto de llamar a rutinas en hilado directo, se debe incluir código en lenguaje de máquina (ejecutando las instrucciones por el CALL) al principio de cada rutina hilada directa para poner el valor actual del apuntador de la lista en una pila de dirección, cargar el registro del apuntador de la lista con la dirección de inicio de la lista de rutinas direccionada por la rutina hilada directa recién iniciada y ejecutar la operación NEXT.

La operación NEXT (codificada como código de máquina en línea) causa que la computadora ejecute la rutina apuntada por el apuntador de lista, a menos que la rutina apuntada sea otra rutina de lenguaje intermedio o una rutina de lenguaje

de máquina.

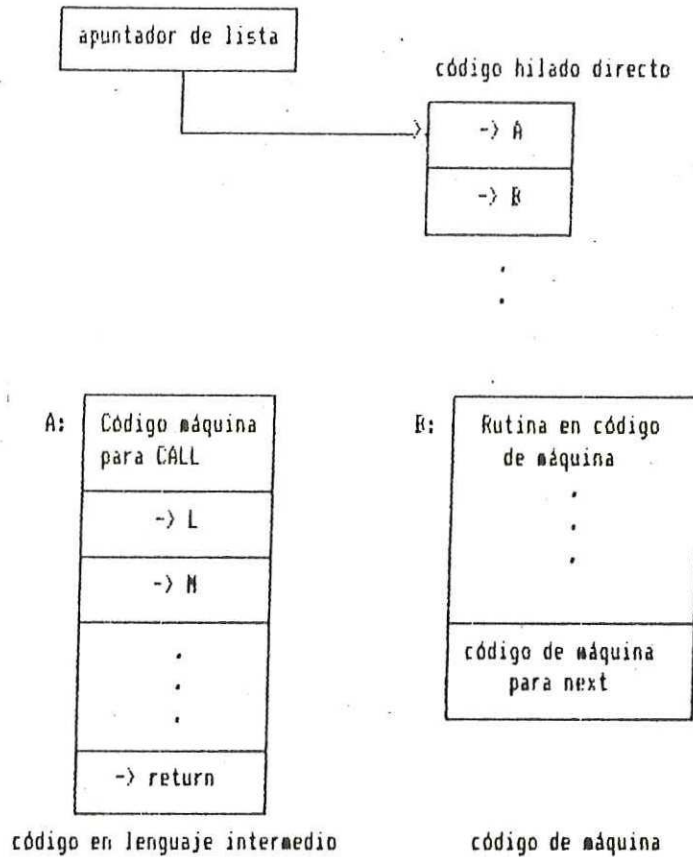
Con el objeto de regresar a un nivel de anidación más alto, el último elemento de la lista en la rutina de lenguaje intermedio apunta al código para la operación RETURN. Cuando esta operación se ejecuta por la operación NEXT, recupera el valor anterior del apuntador de la lista de la pila, luego ejecuta la operación NEXT, la cual en cambio ejecuta la primera rutina pasando a esa que la computadora esta regresando.

Por ello el código hilado directo se implanta en tres operaciones: NEXT, CALL y RETURN. (Figura 11).

III.2.4 Código hilado indirecto

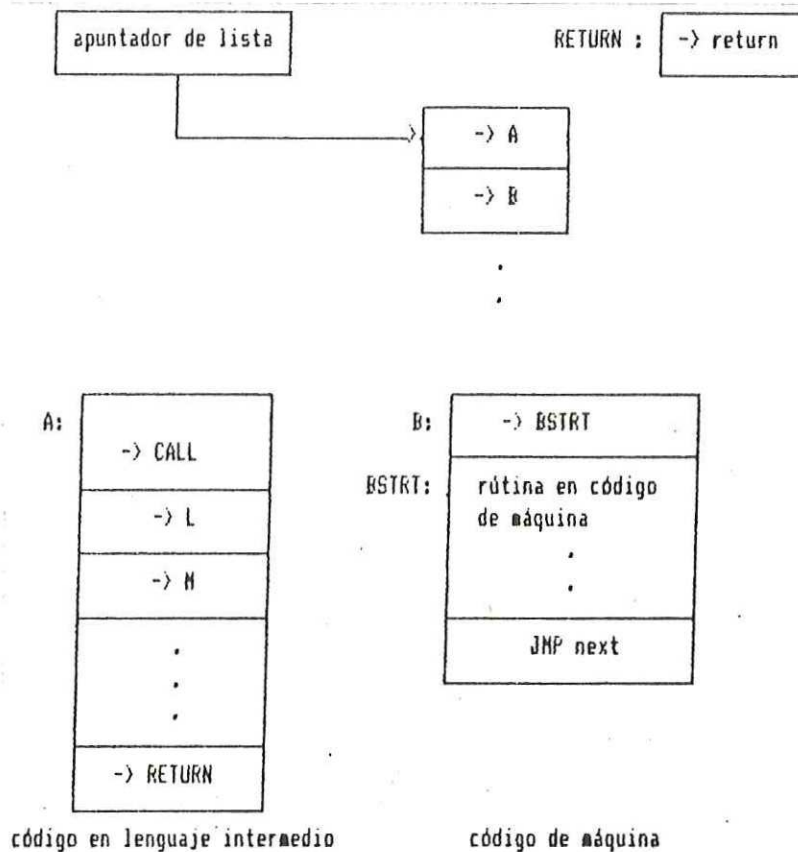
El código hilado indirecto (CHI) consiste en una lista de direcciones, pero en la cual cada dirección apunta a otra dirección, la cual a su vez apunta a la rutina en código de máquina. (ver figura 12). Comparando código hilado indirecto con el directo, en el indirecto el intérprete debe ir a un nivel extra de indirección. Las subrutinas del lenguaje intermedio en hilado directo no contienen código en lenguaje de máquina para la operación CALL. Una ventaja del código hilado indirecto es que un compilador empleando esta técnica solo necesita producir apuntadores. Al manipular solo apuntadores, el compilador genera código en lenguaje intermedio que no incluye código en lenguaje de máquina en si

Asuma que $\rightarrow A$ corresponde a un apuntador a la localidad de memoria designada por la etiqueta A



<u>RUTINA</u>	<u>OPERACIONES</u>
next	1. copia el elemento actual de la lista al almacenamiento temporal 2. cambia el apuntador de la lista al siguiente elemento dentro de ella 3. salta al código de máquina en la dirección en el almacenamiento temporal
CALL	1. coloca en la pila el apuntador de lista actual 2. coloca la dirección de la lista de subrutinas en lenguaje intermedio como apuntador de lista 3. Ejecuta "next"
return	1. coloca como apuntador de lista el valor en la cima de la pila 2. Ejecuta "next"

Figura 11. Código Hilado Directo

RUTINAOPERACIONES

- | | |
|--------|--|
| next | <ol style="list-style-type: none"> 1. copia el elemento actual de la lista al almacenamiento temporal indirecto 2. cambia el apuntador de la lista al siguiente elemento dentro de ella 3. carga en el almacenamiento temporal de código con el elemento en la dirección del almacenamiento temporal indirecto 4. salta al código de máquina en la dirección en el almacenamiento temporal de código |
| CALL | <ol style="list-style-type: none"> 1. coloca en la pila el apuntador de lista actual 2. apunta el almacenamiento temporal indirecto al siguiente elemento de la lista 3. carga apuntador de lista actual del almacenamiento temporal indirecto 4. Ejecuta "next" |
| return | <ol style="list-style-type: none"> 1. coloca como apuntador de lista el valor en la cima de la pila 2. Ejecuta "next" |

Figura 12. Código Hilado Indirecto

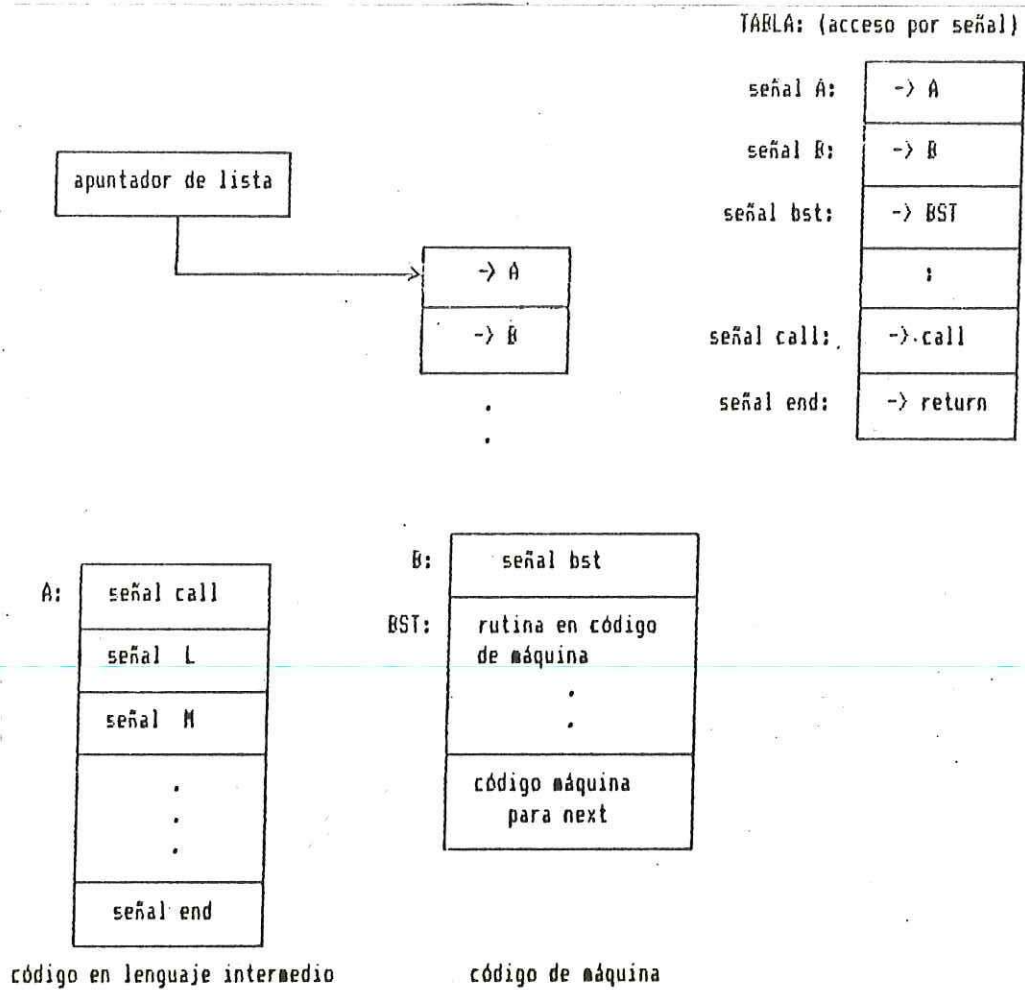
mismo; por ello es independiente de la máquina donde debe trabajar. Sin embargo, la desventaja es que el intérprete tiene el gasto de un nivel extra de direccionamiento indirecto.

III.2.5 Código hilado por señal

Las variedades de código hilado mencionadas previamente contienen apuntadores que fueron direcciones reales a subrutinas en memoria. El emplear las direcciones de memoria para seleccionar rutinas desperdicia almacenamiento porque el número de subrutinas en el sistema es bastante más pequeño que el número de localidades de memoria. Se puede obtener un ahorro en el tamaño del programa en lenguaje intermedio usando señales cortas para identificar las subrutinas que se invocan. Típicamente, el código hilado por señal puede implantarse usando la señal misma como índice apuntador dentro de una tabla de direcciones a subrutinas. (figura 13).

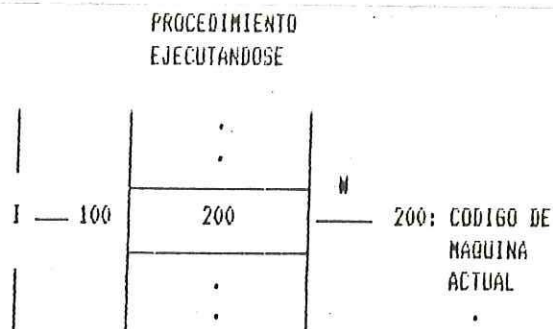
III.3 Selección de la técnica

Siendo FORTH un lenguaje que se implanta bajo alguna de las técnicas expuestas anteriormente, se debía seleccionar la que ofreciera mejores rendimientos con los recursos con los que se contaban y que debían ser optimizados para obtener un buen producto. En las figuras 14, 15 y 16 se encuentran ejemplos de el direccionamiento realizado en el código hilado directo, indirecto y por señal.



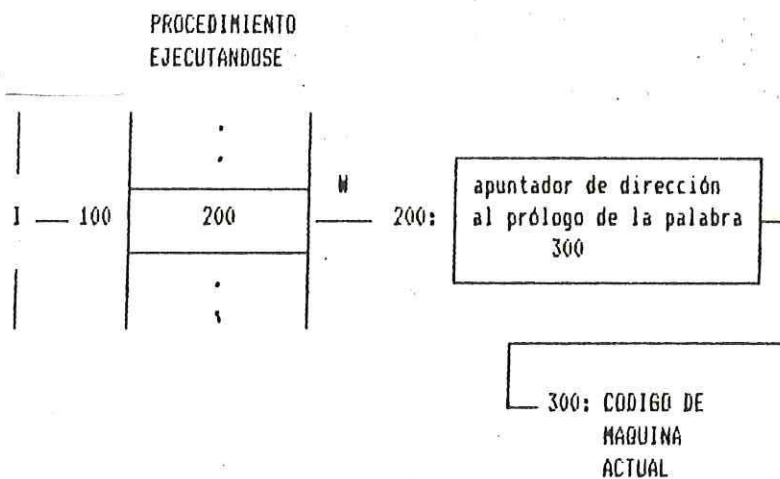
<u>RUTINA</u>	<u>OPERACIONES</u>
next	1. Obtiene señal actual de la lista 2. cambia el apuntador de lista al siguiente elemento dentro de ella 3. busca la dirección correspondiente a la señal 4. obtiene la señal indirecta en la dirección 5. busca la dirección correspondiente a la señal indirecta 6. salta al código máquina en esa dirección
call	1. coloca en la pila el apuntador de lista actual 2. carga el apuntador de lista con el inicio de nueva lista 3. Ejecuta "next"
return	1. coloca como apuntador de lista actual el valor en la cima de la pila 2. Ejecuta "next"

Figura 13. Código Hilado por señal



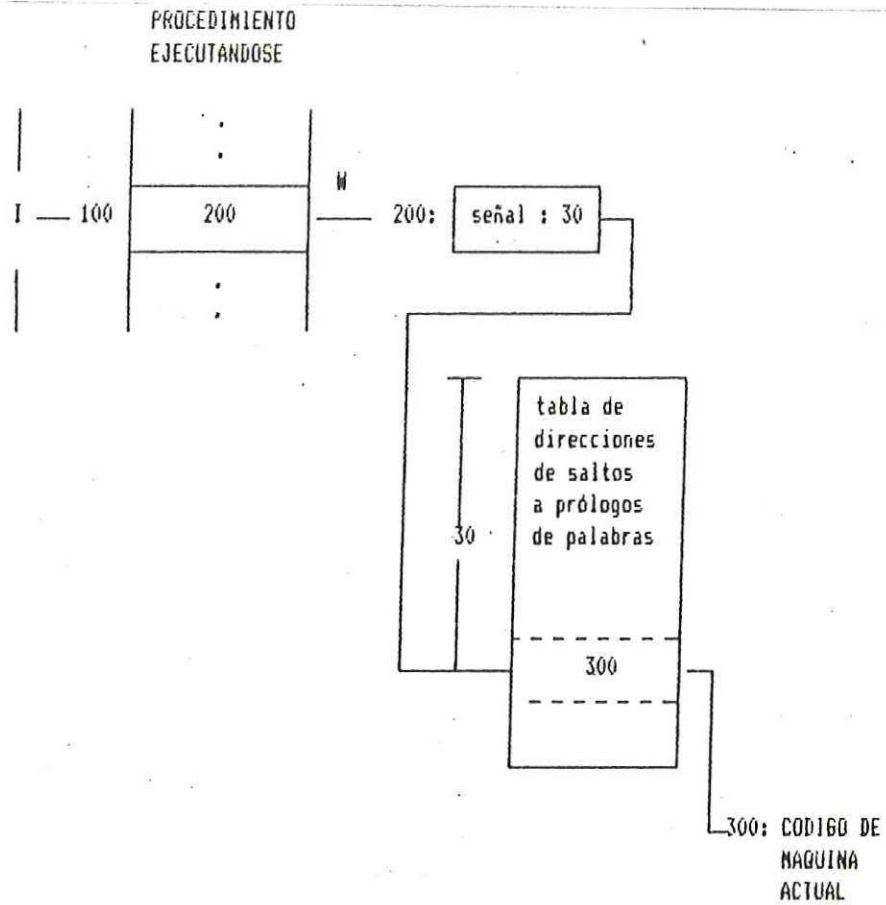
next: W ← contenido de la localidad I
 I ← I + 1
 transferir ejecución a localidad W

Figura 14. Proceso de Ejecución de Código hilado directo



next: W ← contenido de la localidad I
 I ← I + 1
 X ← contenido de la localidad W
 transferir ejecución a localidad X

Figura 15. Proceso de Ejecución de Código hilado indirecto



next: $W \leftarrow$ contenido de la localidad I
 $I \leftarrow I + 1$
 $X \leftarrow$ contenido de la localidad W
 $Y \leftarrow$ contenido de la tabla con índice X
 transferir ejecución a localidad Y

Figura 16. Proceso de Ejecución de Código hilado por Señal

IV IMPLANTACION DE FORTH PARA CUTCMOS

CUTCMOS es un computador en una tarjeta basado en el procesador MC146805E2 de Motorola operando a una velocidad de 5MHz (1 μ seg por ciclo de máquina); tiene un área de memoria de 8 K. CUTCMOS tiene incluido un reloj de tiempo real. Tiene 1 K de memoria RAM, 8 líneas digitales de entrada/salida, 8 canales analógicos, 4 K de memoria externa accesibles a través de una ventana, lo que le permite direccionar sobre 8 bancos de memoria, uno a la vez, dando la posibilidad de direccionar 32 K; y 2 K de memoria EPROM donde reside un monitor. La memoria restante es utilizada por el procesador y por el reloj (Figura 17). Ante este esquema, se tienen disponibles sólo 5 K de memoria para la implantación de Forth, la cual requiere de:

- * Un intérprete externo para comunicación con el usuario,
- * Un intérprete interno que ejecuta cada palabra,
- * Un conjunto de palabras básicas para manipular la pila, hacer definiciones, entrada/salida, operaciones aritméticas y adquisición de datos,
- * La pila de datos, (también llamada pila de parámetros)
- * La pila de retorno, (usada al ejecutar palabras)
- * Un área de almacenamiento de cuerda de entrada de datos y
- * Area para nuevas definiciones.

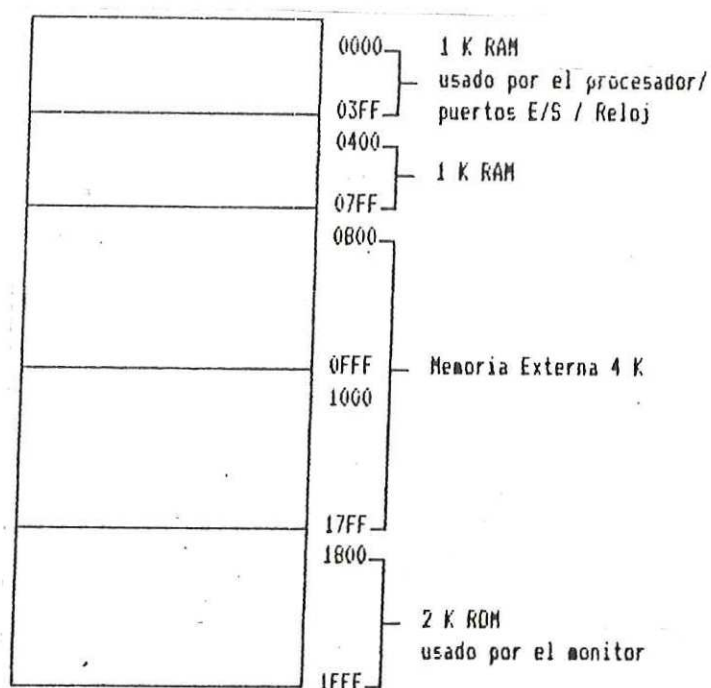


Figura 17. Mapa de Memoria de Microcomputadora

Las palabras básicas están en su mayoría escritas en ensamblador, y varias de ellas están escritas en Forth (Haydon, G.(1983)). Esto es posible ya que como se mencionó anteriormente sólo basta escribir las palabras esenciales y las demás palabras básicas se escriben en base a las esenciales, siguiendo las reglas de Forth. Una clase de instrucciones que son particularmente útiles en la implantación de FORTH lo son los saltos indirectos, los cuales hacen que el procesador salte a una dirección que estaba apuntada por el contenido de una localidad y esta es

identificada por un registro o por otra localidad de memoria. Algunos procesadores tienen una familia completa de instrucciones indirectas (para escribir y leer, además de la transferencia de control que se logra con los saltos) pero en el 6805 no existen por lo que tuvieron que ser simulados mediante código automodificable. La técnica de implantación utilizada es código hilado directo (Ritter, T. (1980)). Aunque no es la técnica más comunmente usada para desarrollar dialectos de Forth (normalmente se utiliza código hilado indirecto), debido a la carencia de instrucciones de indirección se opto por la técnica de código hilado directo.

La estructura de una palabra en FORTH está constituida por un encabezado y un cuerpo. Con el propósito de conservar el uso de la memoria del sistema, la estructura del encabezado se modificó levemente. Sólo el contador de caracteres y los primeros tres caracteres de la palabra se guardan en el encabezado. El octeto destinado para el contador de caracteres tiene la siguiente estructura: el número de caracteres se representa con los cinco bits de orden menor, por lo que se permiten tener palabras de hasta 31 caracteres de longitud. El sexto bit es el bit de marca y tiene el valor de 1 cuando la palabra está marcada y es utilizado en palabras como -FIND y otras más. Una palabra se marca cuando su definición ha sido incompleta y por lo tanto colocada en el diccionario erroneamente. Si la palabra

incompleta no se marcara nunca sería encontrada en el diccionario ya que propiamente no existe por el error de su definición y ocuparía espacio no recuperable.

El séptimo bit no tiene uso definido.

El octavo es 0 normalmente pero es puesto en 1 en palabras inmediatas que deben ejecutarse al momento en que se compilan como parte de una definición. Este bit también se le conoce como bit de precedencia.

Aunque esta estructura no viola las normas no es la más usual. Tradicionalmente el octavo bit siempre es puesto en 1 y es el séptimo bit el bit de precedencia. La forma tradicional no se utilizó para hacer a WORD más pequeña, ya que ésta enmascara el bit de mayor orden en cada octeto que examina en el campo del nombre de una palabra dada cuando trata de encontrarla en el diccionario. En el formato tradicional, el octeto de conteo tiene una máscara diferente de lo que es el resto del campo del nombre.

Así, el formato del encabezado consta de:

- * octeto de conteo de caracteres,
- * los primeros tres caracteres del nombre,
- * un apuntador al campo de nombre de la palabra FORTH
previa y finalmente
- * el código mismo para la palabra.

Por ejemplo, la definición PROCESO puede hacerse como

: PROCESO 1PASO 2PASO 3PASO ;

y su representación interna dentro del diccionario sería la mostrada en la figura 18.

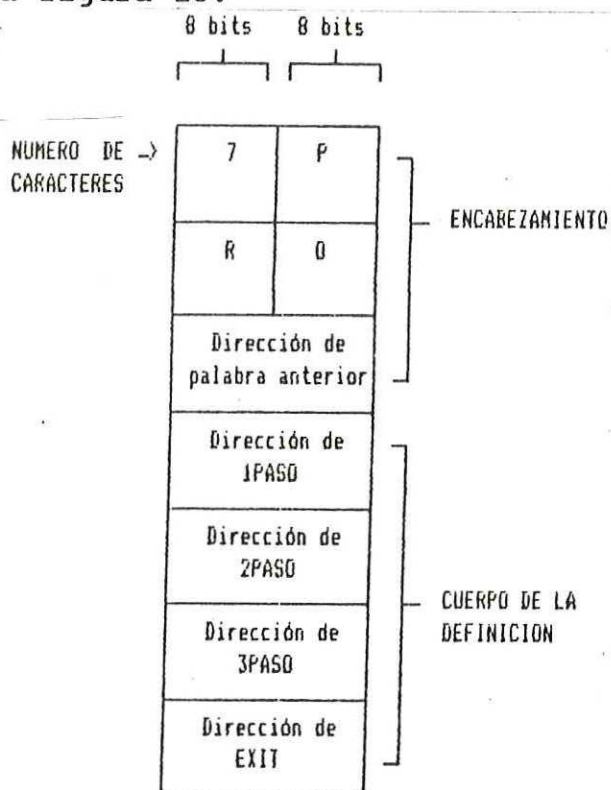


Figura 18. Representación Interna dentro del Diccionario

Aunque no existe una recomendación normativa sobre la implantación del intérprete de Forth, comunmente éste se hace en forma hilada indirecta. En nuestro caso el intérprete de FORTH está implantado como un intérprete hilado directo. El enfoque hilado directo significa que el código empieza donde tradicionalmente habría un apuntador a la dirección del campo de código, de ahí que la palabra ⌞ (tick) regrese la dirección del campo de código y no la dirección del apuntador a éste. Esto significa que debe ponerse cuidado al usar

palabras de FORTH que manipulan direcciones, como lo son la dirección del campo de código, del campo de parámetros y del campo de nombre, ya que su uso bajo el dialecto FORTH/05 es sustancialmente diferente.

Aunque se guardan sólo los tres primeros caracteres de cada palabra, el número de caracteres permite identificar a las palabras diferentes cuyos primeros tres caracteres sean iguales, siempre y cuando no tengan el mismo número de caracteres. Existe un apuntador a la última palabra que ingresó al diccionario y a partir de ésta se buscan las palabras cuando se van a ejecutar. La primera palabra en el diccionario en su parte de dirección de palabra anterior tiene una dirección nula y eventualmente indica el fin del diccionario y de una búsqueda sin éxito. La búsqueda en el diccionario de una palabra es similar al proceso de recorrer una lista ligada.

Por la naturaleza de los recursos con los que se contaban y la propia del lenguaje FORTH se encontraron limitantes mecánicas severas para llevar a cabo la implantación del lenguaje. Como se mencionó anteriormente solo se contaban con 5 K de memoria (los restantes 3 K los ocupan el procesador y el monitor) y los requisitos básicos del lenguaje competían por ella, por lo que debieron marcarse prioridades sobre su uso. Se decidió que las pilas de retorno

y de datos así como la cuerda de entrada debían estar en memoria modificable (lectura/escritura), ocupando para ello 560 octetos. El intérprete externo y las definiciones básicas requerían memoria fija (solo lectura). El intérprete interno por contener secciones de código automodificable debería copiarse a memoria de lectura/escritura para poder ser ejecutado; para ello se contaba con alrededor de 460 octetos, los cuales fueron suficientes para almacenar esa porción de código. El espacio que consumió la realización de los intérpretes y el conjunto básico de definiciones (que en adelante llamaré primitivas) fue de 4 K, por lo que no se contaba con memoria para realizar una función importante del lenguaje: la expansión del diccionario. Sin esto, la realización de aplicaciones, que no son más que nuevas definiciones al diccionario, no podría llevarse a cabo. Ante esta situación se decidió utilizar segmentos de 4 K de memoria y considerar como tales a las primitivas y a las nuevas definiciones.

La administración de los segmentos se basó en la capacidad de direccionar diferentes bancos de memoria a través de una ventana entre las direcciones 0800H-17FFH. Al seleccionar el banco (entre 0 y 7), el contenido de éste se accede como si estuviera en las localidades de la ventana, aunque físicamente correspondan otras direcciones. Cada palabra en el diccionario es referida por su dirección

lógica. La dirección lógica está constituida como sigue: los tres bits más significativos indican el banco y los trece restantes forman la dirección en la ventana donde será accedida. Por ejemplo, supongamos que la palabra PROCESO se encuentra en la dirección 4800H; ésta es su dirección lógica. Para el procesador, se encuentra en el banco 2 (010, los bits más significativos) y a partir de la 0800H. La dirección física se forma tomando los 3 bits del banco y de los trece bits restantes, se descarta el décimo tercero e invierte el décimo segundo. A esta dirección de 15 bits se le coloca un 1 como bit más significativo, que indica la utilización de bancos de memoria. Así, para la dirección lógica 4800H, obtenemos la dirección A000H como su correspondiente dirección física en bancos de memoria. El mapa de memoria de Forth se presenta en la figura 19.

Ahora bien, la naturaleza de las primitivas y las nuevas definiciones son diferentes: las primeras se colocan en ROM y las segundas en RAM. Así tenemos un banco con ROM y en los restantes se coloca RAM en ellos. Los bancos de memoria están colocados en una tarjeta de expansión de memoria CMOS llamada MEM-64, que se comunica a través del ducto C-44 con el computador. La tarjeta cuenta con 8 bases de 28 patas que pueden usarse para colocar circuitos integrados de memoria RAM 6264 o EPROM 27C64, de ahí que pueda configurarse con una mezcla de EPROM y RAM sin modificación.

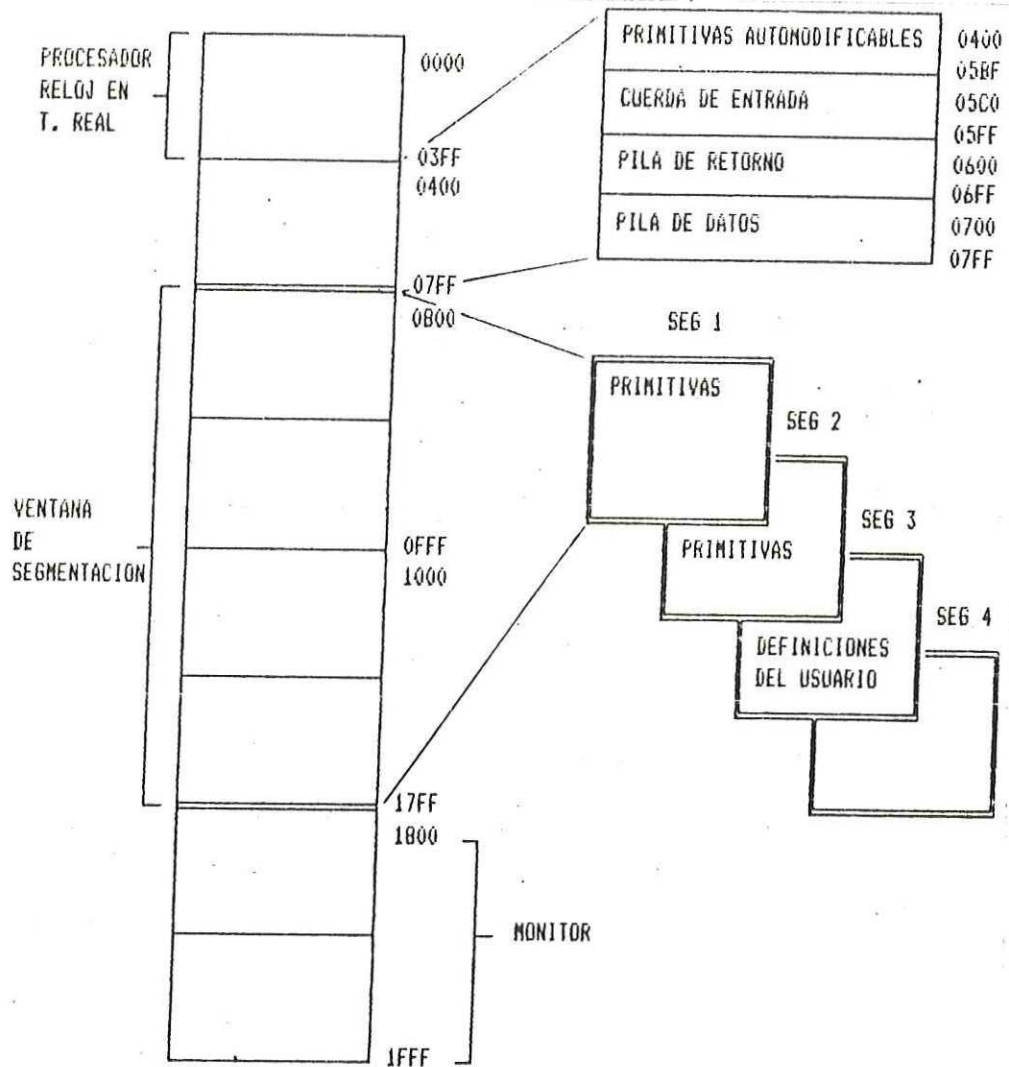


Figura 19. Mapa de Memoria de FORTH

Para realizar las pruebas se utilizó un emulador de EPROM conocido como ROMULATOR. Esto permite cargar un programa ensamblado utilizando la interfase RS-232. En nuestro caso, el dialecto FORTH/05 fue codificado y ensamblado utilizando una computadora IBM-PC, empleando el

ensamblador AS5, que nos generó código en el formato S1S9 de Motorola. El ROMULATOR carga datos que se encuentran en el formato HEX de Intel, por lo que hubo que hacer la conversión a través de un programa (ver apéndice C). Los formatos S1S9 y HEX pueden consultarse en el apéndice B.

Existe un aspecto importante en la codificación. Se mencionó que los bancos son accedidos a través de una ventana entre las localidades 0800H y 17FFH, pero existen referencias a localidades que no se encuentran entre estas direcciones como lo son algunas variables del sistema y la porción de código automodificable. Estas porciones se codificaron con las direcciones reales donde se ejecutarían pero que no se encontrarían en ellas antes de iniciar el empleo del dialecto. Por ello, deberían editarse manualmente las direcciones de compilación en el formato S1S9 para que al inicializarse el sistema esta porción fuera copiada de ROM a RAM a su posición donde fueron ensamblados originalmente. Esta no fue la única edición de direcciones que se realizó. El programa debía cargarse a través del ROMULATOR en el banco 0, de ahí que le correspondan las direcciones a partir de la localidad 0000H hasta la 0FFFFH. Por esta razón se modificaron las direcciones de tal manera que las direcciones en el espacio 0800H a 17FFH correspondieran a las ubicadas a partir de la dirección 0000H. Estas modificaciones se realizaron sobre el código en formato S1S9. Después se aplicó el

programa de conversión a formato HEX y se cargó el dialecto en el banco 0 del MEM-64 a través del ROMULATOR. Durante las pruebas se designó al banco 2 como el área de nuevas definiciones al diccionario, por lo que ésta aparece en el listado del programa (apéndice A) como la dirección 4800H que por los mecanismos de direccionamiento anteriormente explicados serían accedidos como el banco 2. El proceso de cargado del dialecto FORTH/05 se muestra en la figura 20.

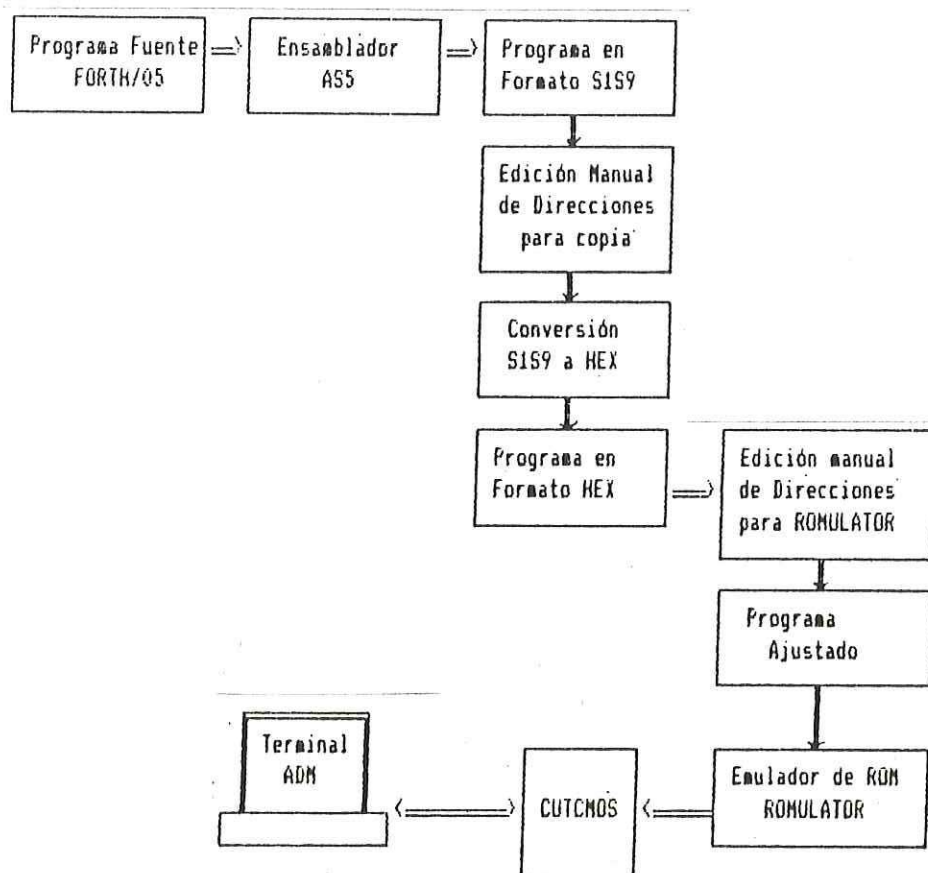


Figura 20. Proceso de Cargado de FORTH/05

V USO DE FORTH/05

El uso de FORTH/05 es a través de la inicialización del computador y la puesta en marcha del monitor, por medio del cual se puede enviar ejecutar el dialecto transfiriendo el control a la porción de código correspondiente con comandos propios del monitor. La inicialización de FORTH/05 causa que se coloquen los apuntadores al diccionario y las variables de FORTH a sus valores por omisión y se realice la copia de ROM a RAM del intérprete interno tal como se describió anteriormente. Aparece el mensaje de FORTH en la terminal que se utiliza como medio de despliegue visual y el indicador OK para empezar a aceptar ordenes que serán analizadas por el intérprete externo.

Con el propósito de minimizar el tamaño del sistema, se ha reducido a una cantidad mínima la verificación de errores. Esta se aplica cuando no se encuentra una palabra en el diccionario, devolviéndose la palabra tecleada seguida de un signo de interrogación. Cuando esto sucede la pila de datos no se afecta, no así la pila de retorno, la cual se reinicializa para evitar devolverse a rutinas fuera del intérprete externo y que ocasionarían errores graves en el sistema. El intérprete se queda en espera de nuevas ordenes por parte del usuario. Si el sistema estaba en modo de compilación cuando ocurre el error, se saldrá de éste para ponerse en estado de ejecución, por lo que deberá iniciarse

de nuevo la definición desde el principio y no desde el punto donde ocurrió el error.

Los errores en la pila no se notifican al usuario pero el comportamiento de las pilas es circular, lo que evita que se invadan otras áreas que pudieran destruir el sistema, afectándose solo el contenido de la pila.

Otros errores no controlados lo son las definiciones incompletas que pueden ser compiladas pero cuya ejecución afecta al sistema. Esto también ocurre con la ejecución inadecuada de palabras que sólo se utilizan en estado de compilación.

Otra situación no controlada como error es la redefinición de palabras, por lo que se permite su compilación sin que se notifique el error. Forth/05 ejecutará la primera que encuentre, que será la definida más recientemente.

Aunque FORTH/05 ha seguido la mayoría de las normas existen pequeñas diferencias con estos en algunos aspectos.

La palabra <NUMBER> que trata la entrada del intérprete como número, no reconoce números de doble precisión: todos son considerados como números de precisión sencilla de dos octetos. La razón es obvia ya que se reduce así el consumo de memoria en el sistema. Además, <NUMBER> toma una dirección

como entrada y regresa una bandera falsa (0) si no puede realizar la conversión ó una verdadera (FFFFH) si se puede realizar la conversión.

Otra diferencia es la palabra NOT. Esta regresa el complemento a uno del valor en la pila de datos. Normalmente NOT regresa un valor verdadero (FFFFH) si el valor en la pila es cero y falso (0) si no lo es, por lo que se considera sinónimo de 0=. Por esto NOT puede usarse para negar el resultado de alguna prueba lógica (por ejemplo 10 = NOT ...) y funcionar adecuadamente pero fallar si se emplea en lugar de 0=. La justificación a esta diferencia es que en procesos de control es muy útil tener una palabra que realice el complemento a uno.

La palabra BASE es una variable de un octeto en lugar de ser de dos octetos. Así que debe manipularse usando C@ y C! en lugar de @ y !. En la misma situación se encuentra >IN.

La palabra TIB es una constante, no una variable por lo que la localidad del área de almacenamiento de la cuerda de entrada no puede cambiarse.

El conjunto de palabras disponibles del dialecto FORTH/05 así como su significado puede consultarse en el apéndice D.

Las palabras de FORTH/05 pueden ser clasificadas en

varios grupos de acuerdo a la función que realizan:

- + manipulación de la pila,
- + comparación,
- + aritméticas y lógicas,
- + manipulación de memoria,
- + estructuras de control,
- + conversión numérica,
- + de definición de palabras,
- + manejo de diccionario,
- + de compilación,
- + de adquisición de datos y
- + diversas.

FORTH/05 ofrece un conjunto de palabras básicas que podemos encontrar en cualquier otra instalación de FORTH, a excepción de las palabras relativas a la administración de entrada/salida en almacenamiento en masa, ya que éste no se cuenta en el sistema desarrollado. Las funciones de adquisición de datos no forman parte de la norma, a pesar de que FORTH tiene una trayectoria notable en este campo. En FORTH/05 existe una palabra llamada LEEADC que permite leer la entrada en alguno de los ocho canales de un convertidor analógico digital, lográndose adquirir datos analógicos de dispositivos para controlar su funcionamiento.

VI PRUEBAS Y CONCLUSIONES

Una de las aplicaciones desarrolladas en este campo utilizando FORTH/05 lo es un programa de graficado empleando un control analógico bidimensional y una terminal gráfica ADM. La aplicación consiste en graficar en la terminal el desplazamiento obtenido al manipular el control con referencia a un plano cartesiano. El punto asociado a un par de coordenadas X,Y corresponde a las dos salidas analógicas del control que son alimentadas a dos canales diferentes del convertidor analógico digital. El programa en FORTH/05 lee los datos de cada canal y las lecturas digitales obtenidas son transformadas a una secuencia de cuatro caracteres que la terminal interpreta como un punto que debe graficar en el plano X-Y.

El desarrollo de esta aplicación, cuyo listado se encuentra a continuación, se hizo en forma modular ascendente, definiendo palabras en base a las cuales se construye la aplicación, que se conoce como la palabra REGISTRO y cuya ejecución hará que la terminal ADM se coloque en modo gráfico (a través de la palabra VGRAF) y dentro de un ciclo infinito BEGIN..UNTIL se lean los canales 1 y 2 del convertidor como las coordenadas X,Y respectivamente y se genere la gráfica al ejecutarse la palabra PLOT.

```

VARIABLE YE
: AJUSTA 1 U# 2 U/MOD SWAP 4 # + 18 OR ;
: /.SWAP 1 U# ROT U/MOD SWAP DROP ;
: CANAL AJUSTA LEEADC ;
: >YE YE ! ;
: <YE YE @ ;
: COORD DUP IF AND SWAP 3E0 AND 20 / ;
: CUATRO >YE COORD <YE COORD ;
: X-Y 20 OR EMIT 60 OR EMIT 20 OR EMIT 40 OR EMIT ;
: PLOT CUATRO X-Y ;
: VGRAF 10 EMIT ;
: REGISTRO VGRAF BEGIN 1 CANAL 4 # 2 CANAL 3 # PLOT 0 UNTIL ;

```

Figura 21. Programa para graficar

Existen otras palabras involucradas, como COORD, CUATRO, X-Y, que no aparecen dentro de la definición de REGISTRO pero que son utilizadas por alguna de las palabras que la conforman.

Dentro de las operaciones que se requieren para transformar un par de lecturas X,Y en el punto gráfico en la terminal ADM se encuentra la división de dos enteros, solo que esta operación no pertenece a las primitivas de FORTH/05. Por tal razón hubo que definirse en base a palabras que si existen y ser invocada como una palabra que deberá definirse dentro de la aplicación que la requiera. La razón de no haberla definido antes como primitiva estriba en que siendo

una palabra que puede definirse en base a otras y contándose con recursos de memoria limitados no es justificable consumir éste recurso innecesariamente, lo que constituye una de las filosofías de desarrollo de aplicaciones de FORTH: deben descartarse del sistema todas aquellas palabras que no sean utilizadas, con el fin de obtener instalaciones más cortas, con menor consumo de memoria.

La búsqueda infructuosa de una palabra en el diccionario es del orden de $(1018*n + 168)$ ciclos de máquina para n palabras en el diccionario. Este tiempo solo tiene repercusión al momento de compilar la aplicación puesto que es el momento en que debe localizarse cada palabra que conforma su definición y conservar su dirección de inicio para colocarla dentro del cuerpo de la nueva palabra por definirse. Al momento de ejecutarse, la única búsqueda que se realiza es la de la aplicación misma que ya contiene la dirección de todas las palabras que la conforman. Como las nuevas palabras se encuentran en la cima del diccionario, el tiempo necesario para localizarlas es corto puesto que si consideramos que n es el número de palabras que se encuentran definidas después de la palabra buscada, el tiempo de búsqueda será de $1018*(n+1)+168$ ciclos de máquina.

Consideramos que FORTH/05 cumple con el objetivo establecido al inicio de su desarrollo: es una herramienta programática compacta implantada en 4 K de memoria, con un

conjunto de instrucciones suficiente para desarrollar aplicaciones de control y un tiempo aceptable de ejecución al tener un eficiente manejo de las llamadas a rutinas hiladas. Aún falta que las aplicaciones desarrolladas se conserven de manera permanente a voluntad del usuario pero se está en proceso de hacerlo. Su campo de aplicación es amplio por ser una buena opción para programar procesos de control pero no sólo eso: puede ser una herramienta didáctica para aprender a programar en Forth. A medida que Forth sea conocido y aceptado, en esa medida FORTH/05 encontrará nuevos campos de aplicación.

LITERATURA CITADA

- Aho, A. V., Sethi, R., Ullman, J. D.. 1987. Compilers, principles, techniques, and tools. Addison-Wesley Publishing Company. Reading, MA. 796 pp.
- Anónimo. 1978. M6800 Resident MPL Compiler Reference Manual. Microsystems. Phoenix AZ . 103 pp.
- Duarte Muñoz, C., Favela Vara, F.. 1983. Microcomputadora de bajo consumo de energía para aplicaciones de instrumentación y control. MEXICON '83 Memorias: 17.3.1-17.3.4.
- Fattal, J.. 1982. A new 8-bit high performance low power microprocessor. Motorola Systems News. 2 :
- Forth Standards Team. 1983. Forth-83 Standard. Mountain View Press, Inc. Mountain View, CA. 78 pp.
- Haydon, G. 1983. All about Forth, an annotated glossary of common Forth ideograms. Mountain View Press, Inc. Mountain View, Ca. 278 pp.
- Rather, E., Moore, C. 1976. The Forth aproach to operating systems. ACM'76 Proceedings. October. 233-240.
- Ritter, T. 1980. Varieties of threaded code for language implementation. BYTE, the small systems journal. 5(8). 206-227.

APENDICE A: Listado de FORTH/05

* DIALECTO DE FORTH PARA MC146805E2 *				
* FORTH/05 *				
* *				

*				
0800	SEG1	EQU	\$0800	DIRECCION DE INICIO DE SEGMENTO 1
1000	SEG2	EQU	\$1000	DIRECCION INICIO SEGMENTO 2
0000	CR	EQU	\$00	RETORNO DE CARRO
000A	LF	EQU	\$0A	ALIMENTACION DE LINEA
0020	BL	EQU	\$20	ESPACIO EN BLANCO
0008	BS	EQU	\$08	RETROCESO
007F	DEL	EQU	\$7F	ELIMINACION
0004	DDR	EQU	4	OFFSET PARA EL REGISTRO DE DETECCION DE DATOS
0000	PRTOA	EQU	0	PUERTO E/S 0
0001	PRTOB	EQU	1	PUERTO E/S 1
0001	PUT	EQU	PRTOB	PUERTO SERIE E/S
007F	INPILA	EQU	\$7F	VALOR INICIAL DEL APUNTADOR DE PILA
007A	PILA	EQU	INPILA-5	CIMA DE LA PILA
2000	TAMMEM	EQU	\$2000	DIRECCION DE TAMAÑO DE ESPACIO EN MEMORIA
0700	SPO	EQU	\$0700	DIRECCION DE INICIO DE LA PILA DE DATOS
0600	RPO	EQU	\$0600	DIRECCION DE INICIO DE LA PILA DE RETORNO
05D0	ICE	EQU	\$05D0	DIRECCION DE INICIO CUERDA DE ENTRADA
1F31	TRAE	EQU	\$1F31	LECTURA DEL PUERTO SERIE
1F65	ENVIA	EQU	\$1F65	ESCRITURA EN EL PUERTO SERIE
193E	ANIN	EQU	\$193E	LECTURA DEL CONVERTIDOR A/D
0014		ORG	\$14	
0014	XTEMP	EQU	\$14	INDICE TEMPORAL
0015	ATEMP	EQU	\$15	TEMPORAL EMPLEADO EN PUTDEC
0016	CUENTA	EQU	\$16	NUMERO DE BITS
0017	OBTR	EQU	\$17	TEMPORAL PARA GUARDAR Y DESCARTAR
001D	CTAOCT	EQU	\$1D	CONTADOR DE OCTETOS
001F	WTIME	EQU	\$1F	AREAS TEMPORALES DIVERSAS
0020	PH	EQU	\$20	*
0021	PL	EQU	\$21	*
0022	TEMPA	EQU	\$22	*
0023	TEMPB	EQU	\$23	*
0024	QH	EQU	\$24	*
0025	QL	EQU	\$25	*
0026	TEMP	EQU	\$26	*
0027	TERM	EQU	\$27	*
*				
0028		ORG	\$0028	
0028 00	ENTRA	FCB	#0	DONDE FORTH BUSCARA PARA ENTRADAS
0029 00	SAL	FCB	#0	*
002A 00	CONTRD	FCB	#0	
002B 48 00	DP	FDB	#\$4800	VALOR INICIAL APUNTADOR DEL DICCIONARIO
002D 00 00	INICIO	FDB	#0	VECTOR DE ENCENDIDO
002F 00 00	IP	FDB	#0	APUNTADOR DE INSTRUCCIONES DE FORTH

0031 00	RP	FCB	#0	APUNTADOR DE LA PILA DE RETORNO
0032 00	SP	FCB	#0	APUNTADOR DE LA PILA DE DATOS
0033 10	BASE	FCB	#\$10	BASE NUMERICA - HEXADECIMAL
0034	USER	EQU	*	ESPACIO PARA VARIABLES DEL USUARIO
0000	FENCE	EQU	0	USER + 0
0034 00 00		FDB	#0	INICIALIZACION DE VARIABLES EN USER
0002	STATE	EQU	2	USER + 2
0036 00 00		FDB	#0	
0004	FORTH	EQU	4	USER + 4
0038 00 00		FDB	#0	*
0006	CONTEXT	EQU	6	USER + 6
003A 00 38		FDB	USER+FORTH	*
0008	CURRENT	EQU	8	USER + 8
003C 00 38		FDB	USER+FORTH	*
000A	HLD	EQU	\$0A	USER + \$0A
003E 00 00		FDB	#0	
	*			
	*			
0400		ORG	\$400	
	* inicio del interprete interno			
0400 BE 31	DOCOL	LDX	RP	COLOCA IP EN LA PILA DE RETORNO
	*			
0400	DOCOL1	EQU	DOCOL	GUARDA EN LA PILA
0402 5A		DECX		DE RETORNO LA DIRECCION
0403 B6 30		LDA	IP+1	DE LA RUTINA QUE
0405 D7 06 00		STA	RP0,X	INVOCO A DOCOL Y CUYA
0408 5A		DECX		EJECUCION SE SUSPENDE
0409 B6 2F		LDA	IP	HASTA LA CULMINACION DE LAS
040B D7 06 00		STA	RP0,X	RUTINAS QUE LA CONFORMAN
040E BF 31		STX	RP	
0410 C6 04 5C		LDA	NEXT1+2	AVANZA EL APUNTADOR DE INSTRUCCIONES
0413 AB 03		ADD	#3	3 OCTETOS ADELANTE PARA LA INVOCACION
0415 B7 30		STA	IP+1	A DOCOL QUE OCUPA ESA CANTIDAD DE OCTETO.
0417 C6 04 5B		LDA	NEXT1+1	IP MANTIENE LA DIRECCION
041A A9 00		ADC	#0	DE LA SIGUIENTE INSTRUCCION DESPUES
041C B7 2F		STA	IP	DE DOCOL
041E B6 30	NEXT	LDA	IP+1	
0420 C7 04 2D		STA	CA+2	OBTIENE EL OCTETO MENOS SIGNIFICATIVO
0423 B6 2F		LDA	IP	DE LA INSTRUCCION PROXIMA A
0425 C7 04 2C		STA	CA+1	EJECUTAR A TRAVES DE NEXT1
042B CD 05 A6		JSR	BANCO	
042B C6 07 00	CA	LDA	SP0	
042E C7 04 5B		STA	NEXT1+1	
0431 B6 30		LDA	IP+1	OBTIENE EL OCTETO MAS SIGNIFICATIVO
0433 AB 01		ADD	#1	DE LA INSTRUCCION PROXIMA A
0435 C7 04 44		STA	CA2+2	EJECUTAR A TRAVES DE NEXT1
0438 B6 2F		LDA	IP	
043A A9 00		ADC	#0	
043C C7 04 43		STA	CA2+1	
043F CD 05 A6		JSR	BANCO	
0442 C6 07 00	CA2	LDA	SP0	

0445 C7 04 5C		STA	NEXT1+2	
0448 B6 30		LDA	IP+1	ACTUALIZA EL APUNTADOR DE
044A AB 02		ADD	#2	INSTRUCCIONES A LA SIGUIENTE
044C B7 30		STA	IP+1	DIRECCION DENTRO DE LA
044E B6 2F		LDA	IP	DEFINICION
0450 A9 00		ADC	#0	
0452 B7 2F		STA	IP	
0454 C6 04 5B	NEXT0	LDA	NEXT1+1	
0457 CD 05 A6		JSR	BANCO	ACTUALIZA EL BANCO DONDE SE ENCUENTRA
045A CC 16 B7	NEXT1	JMP	COLD	LA PALABRA QUE SE EJECUTARA A CONTINUACION
	*			
	*	CODIGO AUTO-MODIFICABLE		
	*			
045D B7 17	CARGAO	STA	OBTR	ALMACENA TEMPORALMENTE EL ACUMULADOR
045F C6 04 68		LDA	CARGA+1	
0462 CD 05 A6		JSR	BANCO	IDENTIFICA EL BANCO DESEADO
0465 B6 17		LDA	OBTR	RECUPERA EL ACUMULADOR
0467 D7 07 00	CARGA	STA	SPO,X	GUARDA A EN LA DIR. ESPECIFICADA (MUEVE A A HERE+X)
046A B7 17	PONO	STA	OBTR	VUELVE AL BANCO 0
046C 4F		CLRA		SIN PERDER EL CONTENIDO
046D CD 05 A6		JSR	BANCO	DEL ACUMULADOR.
0470 B6 17		LDA	OBTR	
0472 81		RTS		
	*			
	*			
0473 C6 04 7A	OBTO	LDA	OBT+1	IDENTIFICA Y CAMBIA
0476 CD 05 A6		JSR	BANCO	AL CAMBIO DESEADO.
0479 D6 07 00	OBT	LDA	SPO,X	CARGA EN A EL CONTENIDO DE HERE+X
047C CD 04 6A		JSR	PONO	VUELVE AL BANCO 0
047F 81		RTS		
	*			
0480 04		FCB	4	TYPE - AUTO MODIFICABLE
0481 54 59 50		FCC	'TYP'	TYPE (dir n --)
0484 00 00		FDB	0	imprime cuerda de n caracteres
0486 BE 32	TYPE	LDX	SP	empezando en la direccion dir,
0488 5C		INCX		DESCARTA OCTETO ALTO
0489 D6 07 00		LDA	SPO,X	
048C B7 2A		STA	CONTR	LA CIMA ES EL CONTADOR DE CARACTERES
048E 5C		INCX		
048F D6 07 00		LDA	SPO,X	CARGA EL OCTETO ALTO DE LA DIRECCION
0492 C7 04 AE		STA	TYSCR+1	EN TYSCR+1
0495 5C		INCX		
0496 D6 07 00		LDA	SPO,X	CARGA EL OCTETO BAJO DE LA DIRECCION
0499 5C		INCX		
049A C7 04 AF		STA	TYSCR+2	EN TYSCR+2
049D BF 32		STX	SP	ACTUALIZA EL APUNTADOR DE LA PILA
049F 3F 29		CLR	SAL	DESPUES DE SACAR LOS DOS VALORES EN LA CIMA.
04A1 3D 2A		TST	CONTR	Y PONE CERO EL NUMERO DE CARACTERES IMPRESOS
04A3 27 19		BEQ	TXIT	SI ES IGUAL AL CONTADOR, TERMINA TYPE
04A5 BE 29	TLOOP	LDX	SAL	
04A7 C6 04 AE		LDA	TYSCR+1	

04AA CD 05 A6		JSR BANCO	ACTUALIZA EL BANCO
04AD D6 07 00	TYSCR	LDA SPO,X	CARGA UN CARACTER
04B0 CD 04 6A		JSR PONO	PONE EN BANCO O Y
04B3 CD 17 5A		JSR SALCARA	LO IMPRIME.
04B6 3C 29		INC SAL	INCREMENTA CONTADOR DE CARACTERES IMPRESOS
04B8 B6 29		LDA SAL	COMPARA SI SE
04BA B0 2A		SUB CONTDR	LLEGO AL FINAL.
04BC 2B E7		BMI TLOOP	SI AUN NO, REGRESA A TLOOP
04BE CC 04 1E	TXIT	JMP NEXT	SINO TERMINA TYPE.
	*		
	*		
04C1 06		FCB 6	<FIND>
04C2 3C 46 49		FCC '<FI'	busca en el diccionario una palabra
04C5 04 80		FDB TYPE-6	
04C7 BE 32	FIN6	LDX SP	
04C9 D6 07 00		LDA SPO,X	
04CC C7 04 7A		STA OBT+1	OBTIENE OCTETO ALTO DE DIR2 Y
04CF 5C		INCX	LO GUARDA EN OBT+1
04D0 D6 07 00		LDA SPO,X	
04D3 C7 04 7B		STA OBT+2	OBTIENE OCTETO BAJO DE DIR2 Y LO GUARDA
04D6 5C		INCX	EN OBT+2
04D7 D6 07 00		LDA SPO,X	OBTIENE OCTETO ALTO DE DIR1 Y LO
04DA 5C		INCX	
04DB C7 05 0C		STA FINSCR+1	GUARDA EN FINSCR+1 Y EN
04DE C7 04 F4		STA FINCNT+1	FINCNT+1
04E1 D6 07 00		LDA SPO,X	OBTIENE OCTETO BAJO DE DIR 1 Y LO
04E4 5C		INCX	GUARDA EN
04E5 C7 05 0D		STA FINSCR+2	FINSCR+2 Y EN
04E8 C7 04 F5		STA FINCNT+2	FINCNT+2
04EB BF 32		STX SP	ACTUALIZA EL APUNTADOR DE PILA
04ED C6 04 F4		LDA FINCNT+1	
04F0 CD 05 A6		JSR BANCO	IDENTIFICA EL BANCO DONDE DEBE BUSCAR
04F3 C6 07 00	FINCNT	LDA SPO	CARGA EL PRIMER OCTETO DE LA LOCALIDAD DIR1
04F6 B7 2A		STA CONTDR	LO GUARDA COMO CONTADOR
04F8 4D		TSTA	SI ES CERO,
04F9 27 3C		BEQ NONE	NO HAY PALABRA Y SALE.
04FB 5F	FINLP1	CLR X	
04FC CD 04 73	FINLP2	JSR OBT0	OBTIENE EL CARACTER EN DIR2
04FF A4 7F		AND #\$7F	ENMASCARA EL OCTAVO BIT
0501 B7 26		STA TEMP	Y LO GUARDA TEMPORALMENTE
0503 C6 05 0C		LDA FINSCR+1	
0506 CD 05 A6		JSR BANCO	ACTUALIZA EL BANCO DONDE DEBE BUSCAR
0509 B6 26		LDA TEMP	RECUPERA EL CARACTER DE DIR2
050B D1 07 00	FINSCR	CMP SPO,X	COMPARA CON EL CARACTER EN DIR1
050E 26 0B		BNE NFND	SI NO SON IGUALES, NO SE ENCONTRO
0510 A3 03		CPX #3	X=3? ENTONCES SALE COMO
0512 27 28		BEQ FOUND	ENCONTRADO
0514 B3 2A		CPX CONTDR	X=CONTADOR?
0516 27 24		BEQ FOUND	SI LO ES, ENCONTRO LA PALABRA
0518 5C		INCX	SINO AVANZA AL SIGUIENTE
0519 20 E1		BRA FINLP2	CARACTER

051B AE 04	NFND	LDX #4	AVANZA SOBRE CONTADOR Y CAMPO DE NOMBRE(1 Y 3 OCTETOS)
051D CD 04 73		JSR OBTO	VA AL SIGUIENTE ELEMENTO EN EL DICCIONARIO
0520 B7 15		STA ATEMP	OCTETO ALTO DIR. SIG. PALABRA EN DICCIONARIO
0522 5C		INCX	
0523 CD 04 73		JSR OBTO	OCTETO BAJO DIR. SIG. PALABRA EN DICCIONARIO
0526 BA 15		ORA ATEMP	AMBOS SON CEROS?
0528 27 0D		BEQ NONE	SI LO SON, SE TERMINO EL DICCIONARIO.
052A CD 04 73		JSR OBTO	
052D C7 04 7B		STA OBT+2	SINO, MUEVE LA DIRECCION DEL SIGUIENTE
0530 B6 15		LDA ATEMP	ELEMENTO A OBT+1 Y OBT+2
0532 C7 04 7A		STA OBT+1	
0535 20 C4		BRA FINLP1	E INTENTA DE NUEVO
0537 BE 32	NONE	LDX SP	
0539 4F		CLRA	COLOCA BANDERA FALSA
053A 20 27		BRA FQUIT	Y VE A FQUIT
053C BE 32	FOUND	LDX SP	
053E C6 04 7B		LDA OBT+2	OBTIENE LA DIRECCION DE INICIO DE LA PALABRA
0541 AB 06		ADD #6	(DIR1 + 1 (CONTADOR) + 3 (NOMBRE) + 2 (SIGUIENTE)
0543 5A		DECX	
0544 D7 07 00		STA SPO,X	GUARDA OCTETO BAJO DE DIRECCION
0547 C6 04 7A		LDA OBT+1	
054A A9 00		ADC #0	
054C 5A		DECX	
054D D7 07 00		STA SPO,X	GUARDA OCTETO ALTO DE DIRECCION
0550 BF 32		STX SP	
0552 5F		CLR X	
0553 CD 04 73		JSR OBTO	OBTIENE EL CONTADOR DE CARACTERES
0556 BE 32		LDX SP	
0558 5A		DECX	
0559 D7 07 00		STA SPO,X	LO GUARDA EN LA PILA DE DATOS
055C 4F		CLRA	
055D 5A		DECX	
055E D7 07 00		STA SPO,X	GUARDA 0 COMO OCTETO ALTO DEL CONTADOR
0561 A6 FF		LDA #\$FF	CARGA BANDERA VERDADERA
0563 5A	FQUIT	DECX	COLOCA LA
0564 D7 07 00		STA SPO,X	BANDERA COMO
0567 5A		DECX	EL ELEMENTO
0568 D7 07 00		STA SPO,X	EN LA CIMA DE LA PILA Y
056B BF 32		STX SP	ACTUALIZA EL APUNTADOR.
056D CC 04 1E		JMP NEXT	
0570 04	*	FCB 4	
0571 42 52 41		FCC 'BRA'	BRAN SALTO INCONDICIONAL ADELANTE O ATRAS
0574 04 C1		FDB FIN6-6	
0576 B6 2F	BRAN	LDA IP	CARGA OCTETO ALTO DEL APUNTADOR DE INSTRUCCION
0578 C7 05 8F		STA BRSC1+1	EN DIRECCION ALTA DE BRSC1+1 Y
057B C7 05 9D		STA BRSC2+1	BRSC2+2
057E B6 30		LDA IP+1	CARGA EL OCTETO BAJO DE IP
0580 C7 05 90		STA BRSC1+2	LO GUARDA COMO OCTETO BAJO DE DOS DIRECCIONES
0583 C7 05 9E		STA BRSC2+2	
0586 AE 01		LDX #1	

0815 A6 64	LDA #L	PREPARA DIRECCION DEL PROMPT
0817 5A	DECX	DE FORTH (OK)
0818 D7 07 00	STA SP0,X	
081B A6 16	LDA #H1	
081D 5A	DECX	
081E D7 07 00	STA SP0,X	
0821 BF 32	STX SP	
0823 B6 2D	SIGUE LDA INICIO	
0825 B7 2F	STA IP	COLOCA IP AL INICIO DEL
0827 B6 2E	LDA INICIO+1	INTERPRETE
0829 B7 30	STA IP+1	
082B CC 04 1E	JMP NEXT	
082E B6 2B	QERR LDA DP	
0830 C7 04 68	STA CARGA+1	
0833 B6 2C	LDA DP+1	
0835 C7 04 69	STA CARGA+2	
0838 5F	CLR X	
0839 CD 04 73	JSR OBTO	
083C 4C	INCA	
083D 4C	INCA	
083E CD 04 5D	JSR CARGAO	
0841 97	TAX	
0842 A6 3F	LDA #\$3F	AGREGA SIGNO DE INTERROGACION A<- '?'
0844 CD 04 5D	JSR CARGAO	
0847 BE 32	LDX SP	
0849 B6 2C	LDA DP+1	
084B 5A	DECX	
084C D7 07 00	STA SP0,X	
084F B6 2B	LDA DP	
0851 5A	DECX	
0852 D7 07 00	STA SP0,X	
0855 BF 32	STX SP	
0857 CC 16 FA	JMP WARM	
	★	
085A 04	FCB 4	
085B 51 55 49	FCC 'QUI'	QUIT (--)
085E 17 DB	FDB DFND-6	Limpia la pila de retorno, se pone en modo de
0860 20 B1	QUIT BRA QEXIT	ejecucion y regresa control a terminal
	★	
0862 05	FCB 5	COUNT (dir -- dir+1 n)
0863 43 4F 55	FCC 'COU'	Deja la direccion dir+1 y el contador de carac
0866 08 5A	FDB QUIT-6	teres que esta al inicio de dir.
0868 CC 04 00	COU5 JMP DOCOL	
086B 15 98	FDB DUP3	
086D 14 8D	FDB ONEP	
086F 17 14	FDB SWAP	
0871 13 F2	FDB BUTC	
0873 12 09	FDB EXIT	
	★	
0875 01	FCB 1	0 (-- 0)
0876 30 20 20	FCC '0 '	Coloca un cero en la pila de datos

0879 08 62		FDB COU5-6	
087B BE 32	ZERO	LDX SP	
087D CC 0A 15		JMP SPOS	
★			
0880 01		FCB 1	1 (-- 1)
0881 31 20 20		FCC '1 '	Coloca uno en la pila de datos
0884 08 75		FDB ZERO-6	
0886 BE 32	ONE	LDX SP	
0888 A6 01		LDA #1	
088A 5A	ESCRIBE	DECX	
088B D7 07 00	ESC2	STA SPO,X	
088E 5A		DECX	
088F 4F		CLRA	
0890 D7 07 00		STA SPO,X	
0893 BF 32		STX SP	
0895 CC 04 1E		JMP NEXT	
★			
0898 01		FCB 1	2 (-- 2)
0899 32 20 20		FCC '2 '	Coloca un 2 en la pila de datos
089C 08 80		FDB ONE-6	
089E BE 32	TWO	LDX SP	
08A0 A6 02		LDA #2	
08A2 CC 08 8A		JMP ESCRIBE	
★			
08A5 02		FCB 2	2+ (n -- n+2)
08A6 32 2B 20		FCC '2+ '	Suma 2 al valor en la cima de la pila de datos
08A9 08 98		FDB TWO-6	
08AB BE 32	TWOP	LDX SP	
08AD 5C		INCX	
08AE A6 02		LDA #2	
08B0 CC 14 92		JMP COLOCA	
★			
08B3 81		FCB \$81	[(--)
08B4 5B 20 20		FCC '['	Termina el modo de compilacion. El texto en la cuerda de entrada se ejecuta a continuacion.
08B7 08 A5		FDB TWOP-6	
08B9 CC 04 00	LBRAK	JMP DOCOL	
08BC 08 7B		FDB ZERO	
08BE 14 C3		FDB STA5	
08C0 14 EB		FDB STO	
08C2 12 09		FDB EXIT	
★			
08C4 01		FCB 1	] (--)
08C5 5D 20 20		FCC ']' '	Coloca el modo de compilacion. El texto en la cuerda de entrada sera compilado a continuacion
08C8 08 B3		FDB LBRAK-6	
08CA CC 04 00	RBRAK	JMP DOCOL	
08CD 16 1B		FDB LIT3	
08CF 00 C0		FDB \$C0	
08D1 14 C3		FDB STA5	
08D3 14 EB		FDB STO	
08D5 12 09		FDB EXIT	
★			

08D7 0B		FCB 11	DEFINITIONS (--)
08D8 44 45 46		FCC 'DEF'	Establece el vocabulario donde seran
08DB 08 C4		FDB RBRAK-6	incluidas las nuevas definiciones.
08DD CC 04 00	DEFS	JMP DOCOL	
08E0 14 CD		FDB CON7	
08E2 14 1C		FDB FTCH	
08E4 14 D7		FDB CUR7	
08E6 14 EB		FDB STO	
08E8 12 09		FDB EXIT	
08EA 01		FCB 1	+ (n1 n2 -- n1+n2)
08EB 2B 20 20		FCC '+ '	Realiza la suma con los dos valores
08EE 08 D7		FDB DEFS-6	en la cima de la pila de datos.
08F0 BE 32	PLUS	LDX SP	
08F2 D6 07 00		LDA SP0,X	
08F5 5C		INCX	
08F6 B7 20		STA PH	
08F8 D6 07 00		LDA SP0,X	
08FB 5C		INCX	
08FC BF 32		STX SP	
08FE 5C		INCX	
08FF DB 07 00		ADD SP0,X	
0902 D7 07 00		STA SP0,X	
0905 BE 32		LDX SP	
0907 B6 20		LDA PH	
0909 D9 07 00		ADC SP0,X	
090C D7 07 00		STA SP0,X	
090F CC 04 1E		JMP NEXT	
0912 01	*	FCB 1	- (n1 n2 -- n1-n2)
0913 2D 20 20		FCC '- '	Realiza la operacion de resta con los
0916 08 EA		FDB PLUS-6	valores en la cima de la pila
0918 BE 32	MINUS	LDX SP	
091A D6 07 00		LDA SP0,X	
091D 5C		INCX	
091E B7 20		STA PH	
0920 D6 07 00		LDA SP0,X	
0923 5C		INCX	
0924 B7 21		STA PL	
0926 BF 32		STX SP	
0928 5C		INCX	
0929 D6 07 00		LDA SP0,X	
092C B0 21		SUB PL	
092E D7 07 00		STA SP0,X	
0931 BE 32		LDX SP	
0933 D6 07 00		LDA SP0,X	
0936 B2 20		SBC PH	
0938 D7 07 00		STA SP0,X	
093B CC 04 1E		JMP NEXT	
093E 02	*	FCB 2	U* (n1 n2 -- n1*n2)

093F 55 2A 20		FCC 'U* '
0942 09 12		FDB MINUS-6
0944 BE 32	UMULT	LDX SP
0946 D6 07 00		LDA SPO,X
0949 5C		INCX
094A B7 20		STA PH
094C D6 07 00		LDA SPO,X
094F 5C		INCX
0950 B7 21		STA PL
0952 D6 07 00		LDA SPO,X
0955 5C		INCX
0956 B7 24		STA QH
0958 D6 07 00		LDA SPO,X
095B B7 25		STA QL
095D BF 14		STX XTEMP
095F CD 13 1B		JSR MPY16
0962 BE 14		LDX XTEMP
0964 B6 25		LDA QL
0966 D7 07 00		STA SPO,X
0969 B6 24		LDA QH
096B 5A		DECX
096C D7 07 00		STA SPO,X
096F B6 23		LDA TEMPB
0971 5A		DECX
0972 D7 07 00		STA SPO,X
0975 B6 22		LDA TEMPB
0977 5A		DECX
0978 D7 07 00		STA SPO,X
097B BF 32		STX SP
097D CC 04 1E		JMP NEXT
0020	*	
	SEC	EQU PH
	*	
0980 B6 22	DIV16	LDA SEC+2
0982 BE 20		LDX SEC
0984 BF 22		STX SEC+2
0986 48		LSLA
0987 B7 20		STA SEC
0989 B6 23		LDA SEC+3
098B BE 21		LDX SEC+1
098D BF 23		STX SEC+3
098F 49		ROLA
0990 B7 21		STA SEC+1
0992 A6 10		LDA #10
0994 B7 26		STA TEMP
0996 39 22	DBEG	ROL SEC+2
0998 39 23		ROL SEC+3
099A B6 22		LDA SEC+2
099C B0 25		SUB QL
099E 97		TAX
099F B6 23		LDA SEC+3

Realiza la multiplicacion de dos numeros sin signo. El resultado tambien es un numero sin signo.

09A1 B2 24		SBC QH
09A3 25 07		BCS DSKIP
09A5 BF 22		STX SEC+2
09A7 B7 23		STA SEC+3
09A9 99		SEC
09AA 20 01		BRA DSKIP+1
09AC 98	DSKIP	CLC
09AD 39 20		ROL SEC
09AF 39 21		ROL SEC+1
09B1 3A 26		DEC TEMP
09B3 26 E1		BNE DBEG
09B5 81		RTS

*

09B6 05		FCB 5
09B7 55 2F 4D		FCC 'U/M'
09BA 09 3E		FDB UMULT-6
09BC BE 32	UDMD	LDX SP
09BE D6 07 00		LDA SP0,X
09C1 5C		INCX
09C2 B7 24		STA QH
09C4 D6 07 00		LDA SP0,X
09C7 5C		INCX
09C8 B7 25		STA QL
09CA D6 07 00		LDA SP0,X
09CD 5C		INCX
09CE B7 21		STA SEC+1
09D0 D6 07 00		LDA SP0,X
09D3 5C		INCX
09D4 B7 20		STA SEC
09D6 D6 07 00		LDA SP0,X
09D9 5C		INCX
09DA B7 23		STA SEC+3
09DC D6 07 00		LDA SP0,X
09DF B7 22		STA SEC+2
09E1 BF 32		STX SP
09E3 AD 9B		BSR DIV16
09E5 BE 32		LDX SP
09E7 B6 22		LDA SEC+2
09E9 D7 07 00		STA SP0,X
09EC B6 23		LDA SEC+3
09EE 5A		DECX
09EF D7 07 00		STA SP0,X
09F2 B6 20		LDA SEC
09F4 5A		DECX
09F5 D7 07 00		STA SP0,X
09F8 B6 21		LDA SEC+1
09FA 5A		DECX
09FB D7 07 00		STA SP0,X
09FE BF 32		STX SP
0A00 CC 04 1E		JMP NEXT

*

U/MOD (n1 n2 -- n3 n4)

Realiza la division sin signo de un numero
doble entre uno sencillo, dejando el residuo
en n3 y el cociente en n4

0A03 04		FCB 4
0A04 53 2D 3E		FCC 'S->'
0A07 09 B6		FDB UDMD-6
0A09 BE 32	STOD	LDX SP
0A0B D6 07 00		LDA SP0,X
0A0E 4D		TSTA
0A0F 2A 04		BPL SPOS
0A11 A6 FF		LDA #\$FF
0A13 20 01		BRA SEXIT
0A15 4F	SPOS	CLRA
0A16 5A	SEXIT	DECX
0A17 D7 07 00		STA SP0,X
0A1A 5A		DECX
0A1B D7 07 00		STA SP0,X
0A1E BF 32		STX SP
0A20 CC 04 1E		JMP NEXT

S->D (n -- d)

Forma un numero doble con signo a partir de uno sencillo con signo.

0A23 03		FCB 3
0A24 50 41 44		FCC 'PAD'
0A27 0A 03		FDB STOD-6
0A29 CC 04 00	PAD	JMP DOCOL
0A2C 14 64		FDB HERE
0A2E 16 1B		FDB LIT3
0A30 00 44		FDB \$0044
0A32 08 F0		FDB PLUS
0A34 12 09		FDB EXIT

PAD (-- dir)

Devuelve la direccion del area de descarte empleada para guardar cuerdas de caracteres para procesamiento inmediato.

0A36 02		FCB 2
0A37 3C 23 20		FCC '<# '
0A3A 0A 23		FDB PAD-6
0A3C CC 04 00	LSHP	JMP DOCOL
0A3F 0A 29		FDB PAD
0A41 14 A9		FDB HLD3
0A43 14 EB		FDB STO
0A45 12 09		FDB EXIT

<# (d1 -- d1)

Inicializa la salida con formato de un numero.

0A47 04		FCB 4
0A48 4F 56 45		FCC 'OVE'
0A4B 0A 36		FDB LSHP-6
0A4D BE 32	OVER	LDX SP
0A4F 5C		INCX
0A50 5C		INCX
0A51 D6 07 00		LDA SP0,X
0A54 5C		INCX
0A55 B7 15		STA ATEMP
0A57 D6 07 00		LDA SP0,X
0A5A BE 32		LDX SP
0A5C 5A		DECX
0A5D D7 07 00		STA SP0,X
0A60 B6 15		LDA ATEMP
0A62 5A		DECX

OVER (n1 n2 -- n1 n2 n1)

Deja una copia del segundo valor en la pila de datos.

0A63 D7 07 00		STA SP0,X	
0A66 BF 32		STX SP	
0A68 CC 04 1E		JMP NEXT	
	*		
0A6B 02		FCB 2	#> (n -- dir n)
0A6C 23 3E 20		FCC '#> '	Termina la conversion de salida numerica
0A6F 0A 47		FDB OVER-6	con formato. La cuerda de salida se encuentra
0A71 CC 04 00	SPGR	JMP DOCOL	en dir y tiene n caracteres.
0A74 13 E3		FDB DROP	
0A76 13 E3		FDB DROP	
0A78 14 A9		FDB HLD3	
0A7A 14 1C		FDB FTCH	
0A7C 0A 29		FDB PAD	
0A7E 0A 4D		FDB OVER	
0A80 09 18		FDB MINUS	
0A82 12 09		FDB EXIT	
	*		
0A84 02		FCB 2	>R (n --)
0A85 3E 52 20		FCC '>R '	transfiere n a la pila de retorno.
0A88 0A 6B		FDB SPGR-6	
0A8A BE 32	TOR	LDX SP	
0A8C D6 07 00		LDA SP0,X	
0A8F 5C		INCX	
0A90 B7 15		STA ATEMP	
0A92 D6 07 00		LDA SP0,X	
0A95 5C		INCX	
0A96 BF 32		STX SP	
0A98 BE 31		LDX RP	
0A9A 5A		DECX	
0A9B D7 06 00		STA RP0,X	
0A9E B6 15		LDA ATEMP	
0AA0 5A		DECX	
0AA1 D7 06 00		STA RP0,X	
0AA4 BF 31		STX RP	
0AA6 CC 04 1E		JMP NEXT	
	*		
0AA9 02		FCB 2	R> (-- n)
0AAA 52 3E 20		FCC 'R> '	Saca el valor en la cima de la pila de
0AAD 0A 84		FDB TOR-6	retorno y lo coloca en la cima de la
0AAF BE 31	RTO	LDX RP	pila de datos.
0AB1 D6 06 00		LDA RP0,X	
0AB4 5C		INCX	
0AB5 B7 15		STA ATEMP	
0AB7 D6 06 00		LDA RP0,X	
0ABA 5C		INCX	
0ABB BF 31		STX RP	
0ABD BE 32		LDX SP	
0ABF 5A		DECX	
0AC0 D7 07 00		STA SP0,X	
0AC3 B6 15		LDA ATEMP	
0AC5 5A		DECX	

0AC6 D7 07 00		STA SP0,X	
0AC9 BF 32		STX SP	
0ACB CC 04 1E		JMP NEXT	
	*		
0ACE 02		FCB 2	R0 (-- n)
0ACF 52 40 20		FCC 'R0 '	Copia el numero en la cima de la pila de
0AD2 0A A9		FDB RTO-6	retorno a la pila de datos.
0AD4 BE 31	RFTCH	LDX RP	
0AD6 D6 06 00		LDA RPO,X	
0AD9 5C		INCX	
0ADA B7 15		STA ATEMP	
0ADC D6 06 00		LDA RPO,X	
0ADF BE 32		LDX SP	
0AE1 5A		DECX	
0AE2 D7 07 00		STA SP0,X	
0AE5 B6 15		LDA ATEMP	
0AE7 5A		DECX	
0AE8 D7 07 00		STA SP0,X	
0AEB BF 32		STX SP	
0AED CC 04 1E		JMP NEXT	
	*		
0AF0 03		FCB 3	ROT (n1 n2 n3 -- n2 n3 n1)
0AF1 52 4F 54		FCC 'ROT'	Rota los tres valores en la cima de la
0AF4 0A CE		FDB RFTCH-6	pila de datos, dejando el tercero en la
0AF6 CC 04 00	ROT	JMP DOCOL	cima.
0AF9 0A 8A		FDB TOR	
0AFB 17 14		FDB SWAP	
0AFD 0A AF		FDB RTO	
0AFF 17 14		FDB SWAP	
0B01 12 09		FDB EXIT	
	*		
0B03 04		FCB 4	HOLD (car --)
0B04 48 4F 4C		FCC 'HOL'	Inserta el caracter car en la cuerda de
0B07 0A F0		FDB ROT-6	salida con formato.
0B09 CC 04 00	HOLD	JMP DOCOL	
0B0C 16 1B		FDB LIT3	
0B0E FF FF		FDB \$FFFF	
0B10 14 A9		FDB HLD3	
0B12 15 B9		FDB PLSTO	
0B14 14 A9		FDB HLD3	
0B16 14 1C		FDB FTCH	
0B18 15 1B		FDB CSTO	
0B1A 12 09		FDB EXIT	
	*		
0B1C 05		FCB 5	M/MOD (n1 n2 -- n3 n4)
0B1D 4D 2F 4D		FCC 'M/M'	Divide un numero de doble precision n1
0B20 0B 03		FDB HOLD-6	entre uno de precision sencilla n2,
0B22 CC 04 00	MDM5	JMP DOCOL	dejando el cociente doble en n4 y el
0B25 0A 8A		FDB TOR	residuo sencillo en n3.
0B27 0B 7B		FDB ZERO	
0B29 0A D4		FDB RFTCH	

0B2B 09 BC		FDB UDMD	
0B2D 0A AF		FDB RTO	
0B2F 17 14		FDB SWAP	
0B31 0A 8A		FDB TOR	
0B33 09 BC		FDB UDMD	
0B35 0A AF		FDB RTO	
0B37 12 09		FDB EXIT	
★			
0B39 04		FCB 4	BASE (-- dir)
0B3A 42 41 53		FCC 'BAS'	Deja la direccion de la variable que
0B3D 0B 1C		FDB MDM5-6	contine la base de conversion numerica.
0B3F CC 04 00	BAS4	JMP DOCOL	
0B42 16 1B		FDB LIT3	
0B44 00 33		FDB BASE	
0B46 12 09		FDB EXIT	
★			
0B48 06		FCB 6	SMUDGE (--)
0B49 53 4D 55		FCC 'SMU'	Se usa para marcar una palabra, por si
0B4C 0B 39		FDB BAS4-6	no se completa su definicion.
0B4E CC 04 00	SMUDG	JMP DOCOL	
0B51 15 FB		FDB LAT6	
0B53 16 1B		FDB LIT3	
0B55 00 20		FDB \$0020	
0B57 17 9D		FDB TOG6	
0B59 12 09		FDB EXIT	
★			
0B5B 03		FCB 3	ABS (n -- n)
0B5C 41 42 53		FCC 'ABS'	Coloca el valor absoluto del numero
0B5F 0B 48		FDB SMUDG-6	en la cima en la pila de datos.
0B61 BE 32	ABS	LDX SP	
0B63 D6 07 00		LDA SP0,X	
0B66 4D		TSTA	
0B67 2A 0C		BPL ABXIT	
0B69 43		COMA	
0B6A D7 07 00		STA SP0,X	
0B6D 5C		INCX	
0B6E D6 07 00		LDA SP0,X	
0B71 40		NEGA	
0B72 D7 07 00		STA SP0,X	
0B75 CC 04 1E	ABXIT	JMP NEXT	
★			
0B78 02		FCB 2	0< (n -- bandera)
0B79 30 3C 20		FCC '0<'	deja una bandera verdadera si n es menor
0B7C 0B 5B		FDB ABS-6	que cero.
0B7E BE 32	ZLESS	LDX SP	
0B80 D6 07 00		LDA SP0,X	
0B83 4D		TSTA	
0B84 2A 04		BPL ZLPOS	
0B86 A6 FF		LDA #\$FF	
0B88 20 01		BRA ZLXIT	
0B8A 4F	ZLPOS	CLRA	

0B8B D7 07 00	ZLXIT	STA SP0,X	
0B8E 5C		INCX	
0B8F D7 07 00		STA SP0,X	
0B92 CC 04 1E		JMP NEXT	
*			
0B95 02		FCB 2	0= (n -- bandera)
0B96 30 3D 20		FCC '0= '	Deja una bandera verdadera si n es igual a cero.
0B99 0B 78		FDB ZLESS-6	
0B9B BE 32	ZEQ	LDX SP	
0B9D D6 07 00		LDA SP0,X	
0BA0 5C		INCX	
0BA1 DA 07 00		ORA SP0,X	
0BA4 26 04		BNE ZEN	
0BA6 A6 FF		LDA #\$FF	
0BA8 20 01		BRA ZEXIT	
0BAA 4F	ZEN	CLRA	
0BAB D7 07 00	ZEXIT	STA SP0,X	
0BAE 5A		DECX	
0BAF D7 07 00		STA SP0,X	
0BB2 CC 04 1E		JMP NEXT	
*			
0BB5 01		FCB 1	< (n1 n2 -- bandera)
0BB6 3C 20 20		FCC '< '	Deja una bandera verdadera si n1 es menor que n2.
0BB9 0B 95		FDB ZEQ-6	
0BBB CC 04 00	LESS	JMP DOCOL	
0BBE 09 18		FDB MINUS	
0BC0 0B 7E		FDB ZLESS	
0BC2 12 09		FDB EXIT	
*			
0BC4 01		FCB 1	> (n1 n2 -- bandera)
0BC5 3E 20 20		FCC '> '	Deja una bandera verdadera si n1 es mayor que n2.
0BC8 0B 85		FDB LESS-6	
0BCA CC 04 00	GREAT	JMP DOCOL	
0BCD 17 14		FDB SWAP	
0BCF 0B 8B		FDB LESS	
0BD1 12 09		FDB EXIT	
*			
0BD3 01		FCB 1	= (n1 n2 -- bandera)
0BD4 3D 20 20		FCC '= '	Deja una bandera verdadera si n1 es igual a n2 .
0BD7 0B C4		FDB GREAT-6	
0BD9 CC 04 00	EQUAL	JMP DOCOL	
0BDC 09 18		FDB MINUS	
0BDE 0B 9B		FDB ZEQ	
0BE0 12 09		FDB EXIT	
*			
0BE2 04		FCB 4	SIGN (n --)
0BE3 53 49 47		FCC 'SIG'	Inserta el caracter ascii "-" dentro de la cuerda de salida con formato si n es negativo.
0BE6 0B D3		FDB EQUAL-6	
0BE8 CC 04 00	SIGN	JMP DOCOL	
0BEB 0B 7E		FDB ZLESS	
0BED 11 E2		FDB ZBRAN	

0BEF 00 08		FDB \$0008	
0BF1 16 1B		FDB LIT3	
0BF3 00 2D		FDB \$002D	
0BF5 0B 09		FDB HOLD	
0BF7 12 09		FDB EXIT	
*			
0BF9 06		FCB 6	NEGATE (n -- -n)
0BFA 4E 45 47		FCC 'NEG'	Deja el complemento a dos de un numero.
0BFD 0B E2		FDB SIGN-6	
0BFF BE 32	NEG6	LDX SP	
0C01 D6 07 00		LDA SP0,X	
0C04 43		COMA	
0C05 D7 07 00		STA SP0,X	
0C08 5C		INCX	
0C09 D6 07 00		LDA SP0,X	
0C0C 40		NEGA	
0C0D D7 07 00		STA SP0,X	
0C10 CC 04 1E		JMP NEXT	
*			
0C13 02		FCB 2	+ - (n1 n2 -- n3)
0C14 2B 2D 20		FCC '+- '	Aplica el signo de n2 a n1, el cual se
0C17 0B F9		FDB NEG6-6	deja en n3.
0C19 CC 04 00	PLMI	JMP DOCOL	
0C1C 0B 7E		FDB ZLESS	
0C1E 11 E2		FDB ZBRAN	
0C20 00 04		FDB \$0004	
0C22 0B FF		FDB NEG6	
0C24 12 09		FDB EXIT	
*			
0C26 01		FCB 1	# (n1 -- n2)
0C27 23 20 20		FCC '# '	n2 es el cociente de la division de n1 entre
0C2A 0C 13		FDB PLMI-6	la base que se este utilizando. se emplea
0C2C CC 04 00	SHARP	JMP DOCOL	en la conversion de salida numerica.
0C2F 0B 3F		FDB BAS4	
0C31 13 F2		FDB BUTC	
0C33 0B 22		FDB MDM5	
0C35 0A F6		FDB ROT	
0C37 16 1B		FDB LIT3	
0C39 00 09		FDB \$0009	
0C3B 0A 4D		FDB OVER	
0C3D 0B BB		FDB LESS	
0C3F 11 E2		FDB ZBRAN	
0C41 00 08		FDB \$0008	
0C43 16 1B		FDB LIT3	
0C45 00 07		FDB \$0007	
0C47 0B F0		FDB PLUS	
0C49 16 1B		FDB LIT3	
0C4B 00 30		FDB \$0030	
0C4D 0B F0		FDB PLUS	
0C4F 0B 09		FDB HOLD	
0C51 12 09		FDB EXIT	

0C53 02	*	FCB 2	OR (n1 n2 -- n3)
0C54 4F 52 20		FCC 'OR '	Deja en n3 el resultado de la operacion o
0C57 0C 26		FDB SHARP-6	logica de bits.
0C59 BE 32	OR2	LDX SP	
0C5B D6 07 00		LDA SP0,X	
0C5E 5C		INCX	
0C5F 5C		INCX	
0C60 DA 07 00		ORA SP0,X	
0C63 D7 07 00		STA SP0,X	
0C66 5A		DECX	
0C67 D6 07 00		LDA SP0,X	
0C6A 5C		INCX	
0C6B 5C		INCX	
0C6C DA 07 00		ORA SP0,X	
0C6F D7 07 00		STA SP0,X	
0C72 5A		DECX	
0C73 BF 32		STX SP	
0C75 CC 04 1E		JMP NEXT	
0C78 03	*	FCB 3	AND (n1 n2 -- n3)
0C79 41 4E 44		FCC 'AND'	Deja en n3 el resultado de la operacion y
0C7C 0C 53		FDB OR2-6	logica de bits de n1 y n2.
0C7E BE 32	AND3	LDX SP	
0C80 D6 07 00		LDA SP0,X	
0C83 5C		INCX	
0C84 5C		INCX	
0C85 D4 07 00		AND SP0,X	
0C88 D7 07 00		STA SP0,X	
0C8B 5A		DECX	
0C8C D6 07 00		LDA SP0,X	
0C8F 5C		INCX	
0C90 5C		INCX	
0C91 D4 07 00		AND SP0,X	
0C94 D7 07 00		STA SP0,X	
0C97 5A		DECX	
0C98 BF 32		STX SP	
0C9A CC 04 1E		JMP NEXT	
0C9D 03	*	FCB 3	XOR (n1 n2 -- n3)
0C9E 58 4F 52		FCC 'XOR'	Deja en n3 el resultado de la operacion o
0CA1 0C 78		FDB AND3-6	exclusiva de bits de n1 y n2.
0CA3 BE 32	XOR3	LDX SP	
0CA5 D6 07 00		LDA SP0,X	
0CA8 5C		INCX	
0CA9 5C		INCX	
0CAA D8 07 00		EOR SP0,X	
0CAD D7 07 00		STA SP0,X	
0CB0 5A		DECX	
0CB1 D6 07 00		LDA SP0,X	
0CB4 5C		INCX	

OCB5 5C		INCX	
OCB6 D8 07 00		EOR SP0,X	
OCB9 D7 07 00		STA SP0,X	
OCBC 5A		DECX	
OCBD BF 32		STX SP	
OCBF CC 04 1E		JMP NEXT	
*			
OCC2 04		FCB 4	DDUP (n -- n n)
OCC3 44 44 55		FCC 'DDU'	Duplica el numero de precision doble que
OCC6 0C 9D		FDB XOR3-6	se encuentra en la cima de la pila.
OCC8 CC 04 00	DDUP	JMP DOCOL	
OCCB 0A 4D		FDB OVER	
OCCD 0A 4D		FDB OVER	
OCCF 12 09		FDB EXIT	
*			
OCD1 02		FCB 2	#S (n -- 0 0)
OCD2 23 53 20		FCC '#S '	Convierte los digitos de un numero doble sin
OCD5 0C C2		FDB DDUP-6	signo a caracteres que agrega a la cuerda
OCD7 CC 04 00	SHRPS	JMP DOCOL	de salida con formato. Solo se usa entre
OCDA 0C 2C		FDB SHARP	<# y #>.
OCDC 0C C8		FDB DDUP	
OCDE 0C 59		FDB OR2	
OCE0 0B 9B		FDB ZEQ	
OCE2 11 E2		FDB ZBRAN	
OCE4 FF F6		FDB \$FFF6	
OCE6 12 09		FDB EXIT	
*			
OCE8 01		FCB 1	. (n --)
OCE9 2E 20 20		FCC '. '	Despliega n de acuerdo a la base actual
OCEC 0C D1		FDB SHRPS-6	como una cuerda de caracteres sin formato.
OCEE CC 04 00	DOT	JMP DOCOL	
OCF1 15 98		FDB DUP3	
OCF3 15 98		FDB DUP3	
OCF5 0B 61		FDB ABS	
OCF7 0A 09		FDB STOD	
OCF9 0A 3C		FDB LSHP	
OCFB 0C D7		FDB SHRPS	
OCFD 0A F6		FDB ROT	
OCFF 0B E8		FDB SIGN	
OD01 0A 71		FDB SPGR	
OD03 04 86		FDB TYPE	
OD05 13 E3		FDB DROP	
OD07 12 A4		FDB BL2	
OD09 12 8F		FDB EMIT	
ODOB 12 09		FDB EXIT	
*			
ODOD 07		FCB 7	COMPILE (--)
ODOE 43 4F 4D		FCC 'COM'	Coloca la direccion de la palabra que le
OD11 0C E8		FDB DOT-6	sigue dentro del diccionario.
OD13 CC 04 00	COMP	JMP DOCOL	
OD16 0A AF		FDB RTO	

OD18 15 98		FDB DUP3	
OD1A 08 AB		FDB TWOP	
OD1C 0A 8A		FDB TOR	
OD1E 14 1C		FDB FTCH	
OD20 15 3F		FDB COMA	
OD22 12 09		FDB EXIT	
★			
OD24 81		FCB \$81	; (--)
OD25 3B 20 20		FCC ' ; '	Termina la definicion de dos puntos y
OD28 0D 0D		FDB COMP-6	
OD2A CC 04 00	SEMI	JMP DOCOL	
OD2D 0D 13		FDB COMP	suspende la compilacion.
OD2F 12 09		FDB EXIT	
OD31 0B 4E		FDB SMUDG	
OD33 0B 09		FDB LBRAK	
OD35 12 09		FDB EXIT	
★			
OD37 01		FCB 1	: (--)
OD38 3A 20 20		FCC ' : '	inicia definicion de dos puntos.
OD3B 0D 24		FDB SEMI-6	
OD3D CC 04 00	COLON	JMP DOCOL	
OD40 14 D7		FDB CUR7	
OD42 14 1C		FDB FTCH	
OD44 14 CD		FDB CON7	
OD46 14 EB		FDB STO	
OD48 17 7E		FDB CRE6	
OD4A 0B 4E		FDB SMUDG	
OD4C 16 1B		FDB LIT3	
OD4E 00 CC		FDB \$00CC	
OD50 15 77		FDB CCOMA	
OD52 0D 13		FDB COMP	
OD54 04 00		FDB DOCOL	
OD56 0B CA		FDB RBRAK	
OD58 12 09		FDB EXIT	
★			
OD5A 81		FCB \$81	' (n --)
OD5B 27		FCB \$27	Asigna dos octetos en el diccionario y
OD5C 20 20		FCC ' '	guarda n alli.
OD5E 0D 37		FDB COLON-6	
OD60 CC 04 00	TICK	JMP DOCOL	
OD63 17 E1		FDB DFND	
OD65 11 E2		FDB ZBRAN	
OD67 00 06		FDB \$0006	
OD69 13 E3		FDB DROP	
OD6B 12 09		FDB EXIT	
OD6D 0B 00		FDB QUES	
★			
OD6F BE 32	DOVAR	LDX SP	crea una variable
OD71 C6 04 5C		LDA NEXT1+2	
OD74 AB 03		ADD #3	
OD76 5A		DECX	

0D77 D7 07 00		STA SP0,X	
0D7A C6 04 5B		LDA NEXT1+1	
0D7D A9 00		ADC #0	
0D7F 5A		DECX	
0D80 D7 07 00		STA SP0,X	
0D83 BF 32		STX SP	
0D85 CC 04 1E		JMP NEXT	
★			
0D88 08		FCB 8	VARIABLE (--)
0D89 56 41 52		FCC 'VAR'	crea una nueva entrada en el diccionario
0D8C 0D 5A		FDB TICK-6	y asigna espacio para guardar un valor.
0D8E CC 04 00	VAR8	JMP DOCOL	
0D91 17 7E		FDB CRE6	
0D93 16 1B		FDB LIT3	
0D95 00 CC		FDB \$00CC	
0D97 15 77		FDB CCOMA	
0D99 0D 13		FDB COMP	
0D9B 05 B8		FDB HAZVAR	
0D9D 08 9E		FDB TWO	
0D9F 16 0C		FDB ALL5	
0DA1 12 09		FDB EXIT	
★			
0DA3 C6 04 5C	DOCON	LDA NEXT1+2	Crea una constante
0DA6 AB 03		ADD #3	
0DAB C7 04 7B		STA OBT+2	
0DAB C6 04 5B		LDA NEXT1+1	
0DAE A9 00		ADC #0	
0DB0 C7 04 7A		STA OBT+1	
0DB3 AE 01		LDX #1	
0DB5 CD 04 73		JSR OBT0	
0DB8 BE 32		LDX SP	
0DBA 5A		DECX	
0DBB D7 07 00		STA SP0,X	
0DBE BF 32		STX SP	
0DC0 5F		CLR X	
0DC1 CD 04 73		JSR OBT0	
0DC4 BE 32		LDX SP	
0DC6 5A		DECX	
0DC7 D7 07 00		STA SP0,X	
0DCA BF 32		STX SP	
0DCC CC 04 1E		JMP NEXT	
★			
0DCF 08		FCB 8	CONSTANT (n --)
0DD0 43 4F 4E		FCC 'CON'	Crea una entrada en el diccionario para
0DD3 0D 88		FDB VAR8-6	una constante y coloca n en esa localidad.
0DD5 CC 04 00	CON8	JMP DOCOL	
0DD8 17 7E		FDB CRE6	
0DDA 16 1B		FDB LIT3	
0DDC 00 CC		FDB \$00CC	
0DDE 15 77		FDB CCOMA	
0DE0 0D 13		FDB COMP	

ODE2 05 BE		FDB HAZCON	
ODE4 15 3F		FDB COMA	
ODE6 12 09		FDB EXIT	
	*		
ODE8 01		FCB 1	* (n1 n2 -- n1*n2)
ODE9 2A 20 20		FCC '* '	Realiza la multiplicacion de dos numeros
ODEC 0D CF		FDB CON8-6	en la cima de la pila de datos.
ODEE CC 04 00	MULT	JMP DOCOL	
ODF1 09 44		FDB UMULT	
ODF3 13 E3		FDB DROP	
ODF5 12 09		FDB EXIT	
	*		
ODF7 89		FCB \$89	[COMPILE] (--)
ODF8 5B 43 4F		FCC '[CO'	Realiza la compilacion de la siguiente
ODFB 0D E8		FDB MULT-6	palabra.
ODFD CC 04 00	BCOM9	JMP DOCOL	
OE00 0D 60		FDB TICK	
OE02 15 3F		FDB COMA	
OE04 12 09		FDB EXIT	
	*		
OE06 85		FCB \$85	BEGIN
OE07 42 45 47		FCC 'BEG'	
OE0A 0D F7		FDB BCOM9-6	
OE0C CC 04 00	BEGN	JMP DOCOL	
OE0F 14 64		FDB HERE	
OE11 12 09		FDB EXIT	
	*		
OE13 85		FCB \$85	AGAIN
OE14 41 47 41		FCC 'AGA'	
OE17 0E 06		FDB BEGN-6	
OE19 CC 04 00	AGAIN	JMP DOCOL	
OE1C 0D 13		FDB COMP	
OE1E 05 76		FDB BRAN	
OE20 14 64		FDB HERE	
OE22 09 18		FDB MINUS	
OE24 15 3F		FDB COMA	
OE26 12 09		FDB EXIT	
	*		
OE28 85		FCB \$85	UNTIL
OE29 55 4E 54		FCC 'UNT'	
OE2C 0E 13		FDB AGAIN-6	
OE2E CC 04 00	UNTIL	JMP DOCOL	
OE31 0D 13		FDB COMP	
OE33 11 E2		FDB ZBRAN	
OE35 14 64		FDB HERE	
OE37 09 18		FDB MINUS	
OE39 15 3F		FDB COMA	
OE3B 12 09		FDB EXIT	
	*		
OE3D 82		FCB \$82	IF
OE3E 49 46 20		FCC 'IF '	

0E41 0E 28		FDB UNTIL-6	
0E43 CC 04 00	IF2	JMP DOCOL	
0E46 0D 13		FDB COMP	
0E48 11 E2		FDB ZBRAN	
0E4A 14 64		FDB HERE	
0E4C 08 7B		FDB ZERO	
0E4E 15 3F		FDB COMA	
0E50 12 09		FDB EXIT	
	*		
0E52 84		FCB \$84	THEN
0E53 54 4B 45		FCC 'THE'	
0E56 0E 3D		FDB IF2-6	
0E58 CC 04 00	THEN	JMP DOCOL	
0E5B 14 64		FDB HERE	
0E5D 0A 4D		FDB OVER	
0E5F 09 18		FDB MINUS	
0E61 17 14		FDB SWAP	
0E63 14 EB		FDB STO	
0E65 12 09		FDB EXIT	
	*		
0E67 84		FCB \$84	ELSE
0E68 45 4C 53		FCC 'ELS'	
0E6B 0E 52		FDB THEN-6	
0E6D CC 04 00	ELSE	JMP DOCOL	
0E70 0D 13		FDB COMP	
0E72 05 76		FDB BRAN	
0E74 14 64		FDB HERE	
0E76 08 7B		FDB ZERO	
0E78 15 3F		FDB COMA	
0E7A 17 14		FDB SWAP	
0E7C 0E 58		FDB THEN	
0E7E 12 09		FDB EXIT	
	*		
0E80 85		FCB \$85	WHILE
0E81 57 4B 49		FCC 'WHI'	
0E84 0E 67		FDB ELSE-6	
0E86 CC 04 00	WHILE	JMP DOCOL	
0E89 0E 43		FDB IF2	
0E8B 12 09		FDB EXIT	
	*		
0E8D 86		FCB \$86	REPEAT
0E8E 52 45 50		FCC 'REP'	
0E91 0E 80		FDB WHILE-6	
0E93 CC 04 00	REPET	JMP DOCOL	
0E96 0A 8A		FDB TOR	
0E98 0E 19		FDB AGAIN	
0E9A 0A AF		FDB RTO	
0E9C 0E 58		FDB THEN	
0E9E 12 09		FDB EXIT	
	*		
0EA0 04		FCB 4	<."> (--)

0EA1 3C 2E 22		FCC '<.''	Compilada por .", la cual transmite
0EA4 0E 8D		FDB REPET-6	la siguiente línea de texto al dispo-
0EA6 CC 04 00	BDOTQ	JMP DOCOL	sitivo de salida seleccionado.
0EA9 0A D4		FDB RFTCH	
0EAB 08 68		FDB COU5	
0EAD 15 98		FDB DUP3	
0EAF 14 8D		FDB ONEP	
0EB1 0A AF		FDB RTO	
0EB3 08 F0		FDB PLUS	
0EB5 0A 8A		FDB TOR	
0EB7 04 86		FDB TYPE	
0EB9 12 09		FDB EXIT	
	*		
0EBB 03		FCB 3	TIB (-- dir)
0EBC 54 49 42		FCC 'TIB'	Variable del usuario que contiene la
0EBF 0E A0		FDB BDOTQ-6	direccion de inicio de la cuerda de
0EC1 CC 04 00	TIB3	JMP DOCOL	entrada.
0EC4 16 1B		FDB LIT3	
0EC6 05 D0		FDB ICE	
0EC8 12 09		FDB EXIT	
	*		
0ECA 03		FCB 3	>IN (-- dir)
0ECB 3E 49 4E		FCC '>IN'	Deja la direccion de la variable que
0ECE 0E BB		FDB TIB3-6	contiene el apuntador actual dentro
0ED0 CC 04 00	FRIN	JMP DOCOL	de la cuerda de entrada.
0ED3 16 1B		FDB LIT3	
0ED5 00 28		FDB ENTRA	
0ED7 12 09		FDB EXIT+	
	*		
0ED9 07		FCB 7	'STREAM (-- dir)
0EDA 27		FCB \$27	Regresa la diercción del siguiente
0EDB 53 54		FCC 'ST'	caracter en la cuerda de entrada.
0EDD 0E CA		FDB FRIN-6	
0EDF CC 04 00	TSTRM	JMP DOCOL	
0EE2 0E C1		FDB TIB3	
0EE4 0E D0		FDB FRIN	
0EE6 13 F2		FDB BUTC	
0EE8 08 F0		FDB PLUS	
0EEA 12 09		FDB EXIT	
	*		
0EEC 04		FCB 4	<DO> (n1 n2 --)
0EED 3C 44 4F		FCC '<DO'	Mueve los parametros de control de ciclo
0EF0 0E D9		FDB TSTRM-6	a la pila de retorno.
0EF2 A6 04	BDO	LDA #4	
0EF4 B7 2A		STA CONTDR	
0EF6 BB 32		ADD SP	
0EF8 B7 32		STA SP	
0EFA BE 32	DOAGIN	LDX SP	
0EFC 5A		DECX	
0EFD D6 07 00		LDA SP0,X	
0F00 BF 32		STX SP	

0F02 BE 31		LDX RP	
0F04 5A		DECX	
0F05 D7 06 00		STA RPO,X	
0F08 BF 31		STX RP	
0F0A 3A 2A		DEC CONTDR	
0F0C 26 EC		BNE DOAGIN	
0F0E A6 04		LDA #4	
0F10 BB 32		ADD SP	
0F12 B7 32		STA SP	
0F14 CC 04 1E		JMP NEXT	
	*		
0F17 06		FCB 6	<LOOP> (--)
0F18 3C 4C 4F		FCC '<LO'	Incrementa los indices de los ciclos y prueba
0F1B 0E EC		FDB BDO-6	si este ha llegado a su fin.
0F1D 3F 20	BLOP	CLR PH	
0F1F A6 01		LDA #1	
0F21 B7 21		STA PL	
0F23 BE 31	LOOPS	LDX RP	
0F25 5C		INCX	
0F26 B6 21		LDA PL	
0F28 DB 06 00		ADD RPO,X	
0F2B D7 06 00		STA RPO,X	
0F2E 5A		DECX	
0F2F B6 20		LDA PH	
0F31 D9 06 00		ADC RPO,X	
0F34 D7 06 00		STA RPO,X	
0F37 5C		INCX	
0F38 D6 06 00		LDA RPO,X	
0F3B 5C		INCX	
0F3C 5C		INCX	
0F3D D0 06 00		SUB RPO,X	
0F40 BE 31		LDX RP	
0F42 D6 06 00		LDA RPO,X	
0F45 5C		INCX	
0F46 5C		INCX	
0F47 D2 06 00		SBC RPO,X	
0F4A B8 20		EOR PH	
0F4C 2B 07		BMI LAGIN	
0F4E 5C		INCX	
0F4F 5C		INCX	
0F50 BF 31		STX RP	
0F52 CC 11 F4		JMP ZBREX	
0F55 CC 05 76	LAGIN	JMP BRAN	
	*		
0F58 07		FCB 7	<+LOOP> (n --)
0F59 3C 2B 4C		FCC '<+L'	Incrementa el indice del ciclo en n y
0F5C 0F 17		FDB BLOP-6	prueba si este ha terminado.
0F5E BE 32	BPLOP	LDX SP	
0F60 D6 07 00		LDA SPO,X	
0F63 5C		INCX	
0F64 B7 20		STA PH	

0F66 D6 07 00		LDA SP0,X	
0F69 5C		INCX	
0F6A B7 21		STA PL	
0F6C BF 32		STX SP	
0F6E 20 B3		BRA LOOPS	
	*		
0F70 82		FCB \$82	DO
0F71 44 4F 20		FCC 'DO '	
0F74 0F 58		FDB BPLOP-6	
0F76 CC 04 00	DO	JMP DOCOL	
0F79 0D 13		FDB COMP	
0F7B 0E F2		FDB BDO	
0F7D 14 64		FDB HERE	
0F7F 12 09		FDB EXIT	
	*		
0F81 84		FCB \$84	LOOP
0F82 4C 4F 4F		FCC 'LOO'	
0F85 0F 70		FDB DO-6	
0F87 CC 04 00	LOOP	JMP DOCOL	
0F8A 0D 13		FDB COMP	
0F8C 0F 1D		FDB BLOP	
0F8E 14 64		FDB HERE	
0F90 09 18		FDB MINUS	
0F92 15 3F		FDB COMA	
0F94 12 09		FDB EXIT	
	*		
0F96 85		FCB \$85	+LOOP
0F97 2B 4C 4F		FCC '+LO'	
0F9A 0F 81		FDB LOOP-6	
0F9C CC 04 00	PLOOP	JMP DOCOL	
0F9F 0D 13		FDB COMP	
0FA1 0F 5E		FDB BPLOP	
0FA3 14 64		FDB HERE	
0FA5 09 18		FDB MINUS	
0FA7 15 3F		FDB COMA	
0FA9 12 09		FDB EXIT	
	*		
0FAB 07		FCB 7	DNEGATE (n1 -- -n1)
0FAC 44 4E 45		FCC 'DNE'	Deja el complemento a dos de un numero de
0FAF 0F 96		FDB PLOOP-6	precision doble.
0FB1 A6 03	DNEG7	LDA #3	
0FB3 B7 2A		STA CONTDR	
0FB5 BE 32		LDX SP	
0FB7 D6 07 00	DNLP	LDA SP0,X	
0FBA 43		COMA	
0FBB D7 07 00		STA SP0,X	
0FBE 5C		INCX	
0FBF 3A 2A		DEC CONTDR	
0FC1 26 F4		BNE DNLP	
0FC3 D6 07 00		LDA SP0,X	
0FC6 40		NEGA	

```

0FC7 D7 07 00      STA SP0,X
0FCA CC 04 1E      JMP NEXT

*

0FCD 81            FCB $81          I ( -- n )
0FCE 49 20 20      FCC 'I '        Copia el indice de ciclo a la pila de
0FD1 0F AB          FDB DNEG7-6     datos.
0FD3 CC 04 00      I1  JMP DOCOL
0FD6 0D 13          FDB COMP
0FDB 0A D4          FDB RFTCH
0FDA 12 09          FDB EXIT

*

0FDC 06            FCB $6          LEEADC ( n -- n )
0FDD 4C 45 45      FCC 'LEE'        Lee del convertidor analogico digital
0FE0 0F CD          FDB 11-6
0FE2 BE 32      LEER  LDX SP
0FE4 5C            INCX
0FE5 D6 07 00      LDA SP0,X
0FE8 97            TAX
0FE9 CD 19 3E      JSR ANIN
0FEC 9F            TXA
0FED BE 32          LDX SP
0FEF 5C            INCX
OFF0 CC 08 8B      JMP ESC2

*
*****
*   SEGMENTO DOS DEL PROGRAMA   *
*****

11DC              ORG SEG2+OFFSET+24
0FDC              ULTIMO EQU      LEER-6          ULTIMA ENTRADA AL DICCIONARIO

*
*

11DC 07            FCB 7          OBRANCH ( f -- )
11DD 30 42 52      FCC 'OBR'      procedimiento al tiempo de corrida
11E0 05 70          FDB BRAN-6     para salto condicional. si f es falso
11E2 BE 32      ZBRAN  LDX SP        para salto condicional. si f es falso
11E4 D6 07 00      LDA SP0,X        (cero), el siguiente parametro en li-
11E7 5C            INCX              nea se suma al apuntador del interpre-
11E8 DA 07 00      ORA SP0,X        te para saltar adelante o atras. Es
11EB 5C            INCX              compiladopor IF, UNTIL y WHILE
11EC BF 32          STX SP
11EE 4D            TSTA
11EF 26 03          BNE ZBREX
11F1 CC 05 76      JMP BRAN
11F4 B6 30      ZBREX  LDA IP+1
11F6 AB 02          ADD #2
11F8 B7 30          STA IP+1
11FA B6 2F          LDA IP
11FC A9 00          ADC #0
11FE B7 2F          STA IP
1200 CC 04 1E      JMP NEXT

*

```


1203 04		FCB 4	EXIT (--)
1204 45 58 49		FCC 'EXI'	termina la ejecucion de una
1207 11 DC		FDB ZBRAN-6	definicion de dos puntos.
1209 BE 31	EXIT	LDX RP	(no puede ser usada dentro
120B D6 06 00		LDA RPO,X	de un DO...LOOP)
120E 5C		INCX	
120F B7 2F		STA IP	
1211 D6 06 00		LDA RPO,X	
1214 5C		INCX	
1215 B7 30		STA IP+1	
1217 BF 31		STX RP	
1219 CC 04 1E		JMP NEXT	
	*		
121C 07		FCB 7	EXECUTE (dir --)
121D 45 58 45		FCC 'EXE'	ejecuta la entrada al diccionario
1220 12 03		FDB EXIT-6	en la direccion dir en la pila.
1222 BE 32	EXE7	LDX SP	
1224 D6 07 00		LDA SPO,X	
1227 5C		INCX	
1228 C7 04 5B		STA NEXT1+1	
122B D6 07 00		LDA SPO,X	
122E 5C		INCX	
122F C7 04 5C		STA NEXT1+2	
1232 BF 32		STX SP	
1234 CC 04 54		JMP NEXT0	
	*		
1237 CD 17 62	INLINE	JSR CRLF	
123A A6 20		LDA #BL	
123C 3F 2A		CLR CONTDR	
123E 5F		CLR X	
123F D7 05 D0	INLP1	STA ICE,X	
1242 5C		INCX	
1243 A3 2E		CPX #\$2E	
1245 26 F8		BNE INLP1	
1247 3F 28		CLR ENTRA	
1249 5F		CLR X	
124A CD 17 52	INLP2	JSR OBTCARA	
124D A1 7F		CMP #DEL	
124F 26 1B		BNE INTST2	
1251 A3 00	INDEL	CPX #0	
1253 27 F5		BEQ INLP2	
1255 5A		DECX	
1256 A6 20		LDA #BL	
1258 D7 05 D0		STA ICE,X	
125B A6 08		LDA #BS	
125D CD 17 5A		JSR SALCARA	
1260 A6 20		LDA #BL	
1262 CD 17 5A		JSR SALCARA	
1265 A6 08		LDA #BS	
1267 CD 17 5A		JSR SALCARA	
126A 20 DE		BRA INLP2	

126C A1 08	INTST2	CMP #BS	
126E 27 E1		BEQ INDEL	
1270 A1 0D		CMP #CR	
1272 27 0D		BEQ INEX	
1274 D7 05 D0		STA ICE,X	
1277 A3 2D		CPX #\$2D	
1279 24 01		BHS INSKP	
127B 5C		INCX	
127C CD 17 5A	INSKP	JSR SALCARA	
127F 20 C9		BRA INLP2	
1281 A6 20	INEX	LDA #BL	
1283 CD 17 5A		JSR SALCARA	
1286 CC 04 1E		JMP NEXT	
*			
1289 04		FCB 4	EMIT (char --)
128A 45 4D 49		FCC 'EMI'	imprime el caracter correspondiente
128D 12 1C		FDB EXE7-6	al valor ascii en la pila
128F BE 32	EMIT	LDX SP	
1291 5C		INCX	
1292 D6 07 00		LDA SP0,X	
1295 5C		INCX	
1296 BF 32		STX SP	
1298 CD 17 5A		JSR SALCARA	
129B CC 04 1E		JMP NEXT	
*			
129E 02		FCB 2	BL (-- c)
129F 42 4C 20		FCC 'BL'	deja el valor ascii del blanco
12A2 12 89		FDB EMIT-6	en la pila
12A4 BE 32	BL2	LDX SP	
12A6 A6 20		LDA #BL	
12A8 5A		DECX	
12A9 D7 07 00		STA SP0,X	
12AC 4F		CLRA	
12AD 5A		DECX	
12AE D7 07 00		STA SP0,X	
12B1 BF 32		STX SP	
12B3 CC 04 1E		JMP NEXT	
*			
12B6 04		FCB 4	WORD (char -- dir)
12B7 57 4F 52		FCC 'WOR'	recibe caracteres de entrada
12BA 12 9E		FDB BL2-6	hasta que se encuentra el
12BC B6 2B	WORD	LDA DP	delimitador no cero o hasta
12BE C7 04 68		STA CARGA+1	que la entrada se agota, ignorando
12C1 B6 2C		LDA DP+1	otros delimitadores. Los caracteres
12C3 C7 04 69		STA CARGA+2	se guardan en una cuerda empacada
12C6 3F 2A		CLR CONTDR	con el contador de caracteres
12C8 4F		CLRA	en la posicion del primer caracter.
12C9 5F		CLR	la direccion de inicio de la cuerda
12CA CD 04 5D		JSR CARGAO	se deja en la pila.
12CD BE 32		LDX SP	
12CF 5C		INCX	

12D0 D6 07 00		LDA SPO,X
12D3 B7 27		STA TERM
12D5 5C		INCX
12D6 BF 32		STX SP
12D8 BE 28		LDX ENTRA
12DA A1 20		CMP #BL
12DC 26 0A		BNE TOK
12DE D6 05 D0	IGNBL	LDA ICE,X
12E1 A1 20		CMP #BL
12E3 26 03		BNE TOK
12E5 5C		INCX
12E6 20 F6		BRA IGNBL
12E8 3C 2A	TOK	INC CONTDR
12EA D6 05 D0		LDA ICE,X
12ED BF 14		STX XTEMP
12EF BE 2A		LDX CONTDR
12F1 CD 04 5D		JSR CARGAO
12F4 BE 14		LDX XTEMP
12F6 5C		INCX
12F7 A1 80		CMP #\$80
12F9 27 04		BEQ WORXIT
12FB B1 27		CMP TERM
12FD 26 E9		BNE TOK
12FF BF 28	WORXIT	STX ENTRA
1301 B6 2A		LDA CONTDR
1303 4A		DECA
1304 5F		CLR X
1305 CD 04 5D		JSR CARGAO
1308 B6 2C		LDA DP+1
130A BE 32		LDX SP
130C 5A		DECX
130D D7 07 00		STA SPO,X
1310 B6 2B		LDA DP
1312 5A		DECX
1313 D7 07 00		STA SPO,X
1316 BF 32		STX SP
1318 CC 04 1E		JMP NEXT
	*	
131B AE 10	MPY16	LDX #\$10
131D 3F 22		CLR TEMP A
131F 3F 23		CLR TEMP B
1321 36 24		ROR QH
1323 36 25		ROR QL
1325 24 0C	MPYNXT	BCC ROTAT
1327 B6 23		LDA TEMP B
1329 BB 21		ADD PL
132B B7 23		STA TEMP B
132D B6 22		LDA TEMP A
132F B9 20		ADC PH
1331 B7 22		STA TEMP A
1333 36 22	ROTAT	ROR TEMP A

1335 36 23	ROR TEMPB
1337 36 24	ROR QH
1339 36 25	ROR QL
133B 5A	DECX
133C 26 E7	BNE MPYNXT
133E 81	RTS

*

133F 08	FCB 8
1340 3C 4E 55	FCC '<NU'
1343 12 B6	FDB WORD-6
1345 BE 32	LDX SP
1347 D6 07 00	LDA SPO,X
134A 5C	INCX
134B C7 04 7A	STA OBT+1
134E D6 07 00	LDA SPO,X
1351 5C	INCX
1352 C7 04 7B	STA OBT+2
1355 BF 32	STX SP
1357 5F	CLR X
1358 CD 04 73	JSR OBT0
135B B7 2A	STA CONTDR
135D 4D	TSTA
135E 27 31	BEQ NOTNO
1360 5C	INCX
1361 3F 26	CLR TEMP
1363 3F 24	CLR QH
1365 3F 25	CLR QL
1367 3F 20	CLR PH
1369 B6 33	LDA BASE
136B B7 21	STA PL
136D CD 04 73	JSR OBT0
1370 A1 2D	CMP #\$2D
1372 26 0A	BNE NUMSKP
1374 3A 26	DEC TEMP
1376 5C	INCX
1377 B3 2A	CPX CONTDR
1379 22 16	BHI NOTNO
137B CD 04 73	JSR OBT0
137E 5C	INCX
137F A0 30	SUB #\$30
1381 2B 0E	BMI NOTNO
1383 A1 0A	CMP #\$0A
1385 2B 06	BMI NUMB
1387 A1 11	CMP #\$11
1389 2B 06	BMI NOTNO
138B A0 07	SUB #7
138D B1 33	CMP BASE
138F 25 10	BLO ANUMB
1391 4F	CLRA
1392 BE 32	LDX SP
1394 5A	DECX

NUMB

NUMSKP

NUMB

NOTNO

NUMXT

<NUMBER> (dir -- d)
 convierte el contador y la cuerda
 de caracteres en dir a entero de
 16 bits, usando la base actual. si
 la condicion falla, existe condicion
 de error. La cuerda puede tener un
 guion para indicar que es negativo.

1395 D7 07 00		STA SP0,X	
1398 5A		DECX	
1399 D7 07 00		STA SP0,X	
139C BF 32		STX SP	
139E CC 04 1E		JMP NEXT	
13A1 BF 14	ANUMB	STX XTEMP	
13A3 B7 15		STA ATEMP	
13A5 CD 13 1B		JSR MPY16	
13A8 B6 15		LDA ATEMP	
13AA BB 25		ADD QL	
13AC B7 25		STA QL	
13AE B6 24		LDA QH	
13B0 A9 00		ADC #0	
13B2 B7 24		STA QH	
13B4 BE 14		LDX XTEMP	
13B6 B3 2A		CPX CONTDR	
13B8 22 05		BHI NUMOK	
13BA CD 04 73		JSR OBTO	
13BD 20 BF		BRA NUMSKP	
13BF B6 26	NUMOK	LDA TEMP	
13C1 4D		TSTA	
13C2 27 07		BEQ NUMPOS	
13C4 4F		CLRA	
13C5 30 25		NEG QL	
13C7 B2 24		SBC QH	
13C9 B7 24		STA QH	
13CB BE 32	NUMPOS	LDX SP	
13CD B6 25		LDA QL	
13CF 5A		DECX	
13D0 D7 07 00		STA SP0,X	
13D3 B6 24		LDA QH	
13D5 5A		DECX	
13D6 D7 07 00		STA SP0,X	
13D9 A6 FF		LDA #\$FF	
13DB 20 B7		BRA NUMXT+2	
*			
13DD 04		FCB 4	DROP (n --)
13DE 44 52 4F		FCC 'DRO'	descarta el numero en la cima
13E1 13 3F		FDB NUMB-6	de la pila
13E3 BE 32	DROP	LDX SP	
13E5 5C		INCX	
13E6 5C		INCX	
13E7 BF 32		STX SP	
13E9 CC 04 1E		JMP NEXT	
*			
13EC 02		FCB 2	C@ (dir -- byte)
13ED 43 40 20		FCC 'C@ '	busca el byte menos significativo
13F0 13 DD		FDB DROP-6	en dir y lo coloca en la pila
13F2 BE 32	BUTC	LDX SP	
13F4 D6 07 00		LDA SP0,X	
13F7 5C		INCX	

13FB C7 04 7A		STA OBT+1	
13FB D6 07 00		LDA SPO,X	
13FE C7 04 7B		STA OBT+2	
1401 BF 32		STX SP	
1403 5F		CLR X	
1404 CD 04 73		JSR OBT0	
1407 BE 32		LDX SP	
1409 D7 07 00		STA SPO,X	
140C 4F		CLRA	
140D 5A		DECX	
140E D7 07 00		STA SPO,X	
1411 BF 32		STX SP	
1413 CC 04 1E		JMP NEXT	
	*		
1416 01		FCB 1	@ (dir -- n)
1417 40 20 20		FCC 'a '	coloca en la pila el contenido
141A 13 EC		FDB BUTC-6	de dir
141C BE 32	FTCH	LDX SP	
141E D6 07 00		LDA SPO,X	
1421 5C		INCX	
1422 C7 04 7A		STA OBT+1	
1425 D6 07 00		LDA SPO,X	
1428 C7 04 7B		STA OBT+2	
142B BF 32		STX SP	
142D AE 01		LDX #1	
142F CD 04 73		JSR OBT0	
1432 BE 32		LDX SP	
1434 D7 07 00		STA SPO,X	
1437 5F		CLR X	
1438 CD 04 73		JSR OBT0	
143B BE 32		LDX SP	
143D 5A		DECX	
143E D7 07 00		STA SPO,X	
1441 BF 32		STX SP	
1443 CC 04 1E		JMP NEXT	
	*		
1446 02		FCB 2	DP (-- dir)
1447 44 50 20		FCC 'DP '	variable del usuario, el apuntador al
144A 14 16		FDB FTCH-6	diccionario que contiene la direccion
144C BE 32	DP2	LDX SP	del siguiente espacio libre de memoria
144E A6 2B		LDA #DP	arriba del diccionario. puede leerse por
1450 5A		DECX	HERE y alterarse por ALLOT
1451 D7 07 00		STA SPO,X	
1454 4F		CLRA	
1455 5A		DECX	
1456 D7 07 00		STA SPO,X	
1459 BF 32		STX SP	
145B CC 04 1E		JMP NEXT	
	*		
145E 04		FCB 4	HERE (-- dir)
145F 48 45 52		FCC 'HER'	regresa la direccion de la siguiente

1462 14 46		FDB DP2-6	localidad disponible en el diccionario
1464 CC 04 00	HERE	JMP DOCOL	
1467 14 4C		FDB DP2	
1469 14 1C		FDB FTCH	
146B 12 09		FDB EXIT	
	*		
146D 03		FCB 3	NOT (f1 -- f2)
146E 4E 4F 54		FCC 'NOT'	coloca el complemento a 1'sd de
1471 14 5E		FDB HERE-6	f1. no es identico a 0=
1473 BE 32	NOT3	LDX SP	
1475 D6 07 00		LDA SP0,X	
147B 43		COMA	
1479 D7 07 00		STA SP0,X	
147C 5C		INCX	
147D D6 07 00		LDA SP0,X	
1480 43		COMA	
1481 D7 07 00		STA SP0,X	
1484 CC 04 1E		JMP NEXT	
	*		
1487 02		FCB 2	1+ (n -- n+1)
1488 31 2B 20		FCC '1+ '	suma 1 al valor en la cima en la pila
148B 14 6D		FDB NOT3-6	
148D BE 32	ONEP	LDX SP	
148F 5C		INCX	
1490 A6 01		LDA #1	
1492 DB 07 00	COLOCA	ADD SP0,X	
1495 D7 07 00		STA SP0,X	
149B 5A		DECX	
1499 4F		CLRA	
149A D9 07 00		ADC SP0,X	
149D D7 07 00		STA SP0,X	
14A0 CC 04 1E		JMP NEXT	
	*		
14A3 03		FCB 3	HLD (-- dir)
14A4 48 4C 44		FCC 'HLD'	variable del usuario que guarda la
14A7 14 87		FDB ONEP-6	direccion del ultimo caracter de
14A9 A6 0A	HLD3	LDA #HLD	texto durante la conversion numerica
	*		de la salida.
14AB AB 34	DOUSE	ADD #USER	
14AD BE 32		LDX SP	
14AF 5A		DECX	
14B0 D7 07 00		STA SP0,X	
14B3 4F		CLRA	
14B4 5A		DECX	
14B5 D7 07 00		STA SP0,X	
14B8 BF 32		STX SP	
14BA CC 04 1E		JMP NEXT	
	*		
14BD 05		FCB 5	STATE (-- dir)
14BE 53 54 41		FCC 'STA'	deja la direccion de la variable
14C1 14 A3		FDB HLD3-6	que contiene el estado de compilacion

14C3 A6 02	STA5	LDA #STATE	un valor diferente de cero indica que
14C5 20 E4		BRA DOUSE	esta compilando, pero su valor puede
	*		ser dependiente de la instalacion.
14C7 07		FCB 7	
14C8 43 4F 4E		FCC 'CON'	CONTEXT (-- dir)
14CB 14 BD		FDB STA5-6	indica el vocabulario donde se haran
14CD A6 06	CON7	LDA #CONTEXT	las busquedas durante la interpretacion
14CF 20 DA		BRA DOUSE	de la entrada
	*		
14D1 07		FCB 7	CURRENT (-- dir)
14D2 43 55 52		FCC 'CUR'	deja la direccion de la variable que
14D5 14 C7		FDB CON7-6	especifica el vocabulario en el cual
14D7 A6 08	CUR7	LDA #CURRENT	se haran nuevas definiciones
14D9 20 D0		BRA DOUSE	
	*		
14DB 05		FCB 5	FORTH (--)
14DC 46 4F 52		FCC 'FOR'	Vocabulario principal. su ejecucion lo
14DF 14 D1		FDB CUR7-6	coloca como vocabulario context
14E1 A6 04	FOR5	LDA #FORTH	
14E3 20 C6		BRA DOUSE	
	*		
14E5 01		FCB 1	I (n dir --)
14E6 21 20 20		FCC 'I '	guarda n en la direccion dir
14E9 14 DB		FDB FOR5-6	
14EB BE 32	STO	LDX SP	
14ED D6 07 00		LDA SPO,X	
14F0 5C		INCX	
14F1 C7 04 68		STA CARGA+1	
14F4 D6 07 00		LDA SPO,X	
14F7 5C		INCX	
14F8 C7 04 69		STA CARGA+2	
14FB D6 07 00		LDA SPO,X	
14FE 5C		INCX	
14FF BF 32		STX SP	
1501 5F		CLR X	
1502 CD 04 5D		JSR CARGA0	
1505 BE 32		LDX SP	
1507 D6 07 00		LDA SPO,X	
150A 5C		INCX	
150B BF 32		STX SP	
150D AE 01		LDX #1	
150F CD 04 5D		JSR CARGA0	
1512 CC 04 1E		JMP NEXT	
	*		
1515 02		FCB 2	CI (n dir --)
1516 43 21 20		FCC 'CI '	guarda solo el byte menos significativo
1519 14 E5		FDB STO-6	de n en dir
151B BE 32	CSTO	LDX SP	
151D D6 07 00		LDA SPO,X	
1520 5C		INCX	
1521 C7 04 68		STA CARGA+1	


```

1524 D6 07 00      LDA SP0,X
1527 5C             INCX
1528 C7 04 69      STA CARGA+2
152B 5C             INCX
152C D6 07 00      LDA SP0,X
152F 5C             INCX
1530 BF 32         STX SP
1532 5F            CLRX
1533 CD 04 5D      JSR CARGAO
1536 CC 04 1E      JMP NEXT

```

★

```

1539 01             FCB 1
153A 2C 20 20      FCC ' , '
153D 15 15         FDB CSTO-6
153F B6 2B         LDA DP
1541 C7 04 68      STA CARGA+1
1544 B6 2C         LDA DP+1
1546 C7 04 69      STA CARGA+2
1549 BE 32         LDX SP
154B D6 07 00      LDA SP0,X
154E 5C             INCX
154F BF 32         STX SP
1551 5F            CLRX
1552 CD 04 5D      JSR CARGAO
1555 BE 32         LDX SP
1557 D6 07 00      LDA SP0,X
155A 5C             INCX
155B BF 32         STX SP
155D AE 01         LDX #1
155F CD 04 5D      JSR CARGAO
1562 A6 02         LDA #2
1564 BB 2C         ADD DP+1
1566 B7 2C         STA DP+1
1568 B6 2B         LDA DP
156A A9 00         ADC #0
156C B7 2B         STA DP
156E CC 04 1E      JMP NEXT

```

COMA

INCDP

★

```

1571 02             FCB 2
1572 43 2C 20      FCC 'C, '
1575 15 39         FDB COMA-6
1577 B6 2B         LDA DP
1579 C7 04 68      STA CARGA+1
157C B6 2C         LDA DP+1
157E C7 04 69      STA CARGA+2
1581 BE 32         LDX SP
1583 5C             INCX
1584 D6 07 00      LDA SP0,X
1587 5C             INCX
1588 BF 32         STX SP
158A 5F            CLRX

```

CCOMA

, (n --)

asigna dos bytes en el diccionario,
guardando n ahí

C, (n --)

guarda los 8 bits de orden mas bajo
de n en el siguiente byte en el diccionario
incrementando el apuntador del diccionario.

158B CD 04 5D		JSR CARGAO	
158E A6 01		LDA #1	
1590 20 D2		BRA INCDP	
	*		
1592 03		FCB 3	DUP (n -- n n)
1593 44 55 50		FCC 'DUP'	deja una copia del valor en la cima de
1596 15 71		FDB CCOMA-6	de la pila
1598 BE 32	DUP3	LDX SP	
159A D6 07 00		LDA SP0,X	
159D 5A		DECX	
159E 5A		DECX	
159F D7 07 00		STA SP0,X	
15A2 BE 32		LDX SP	
15A4 5C		INCX	
15A5 D6 07 00		LDA SP0,X	
15A8 5A		DECX	
15A9 5A		DECX	
15AA D7 07 00		STA SP0,X	
15AD 5A		DECX	
15AE BF 32		STX SP	
15B0 CC 04 1E		JMP NEXT	
	*		
15B3 02		FCB 2	+1 (n dir --)
15B4 2B 21 20		FCC '+1 '	suma n al valor de 16 bits contenido
15B7 15 92		FDB DUP3-6	en dir
15B9 BE 32	PLSTO	LDX SP	
15BB D6 07 00		LDA SP0,X	
15BE 5C		INCX	
15BF C7 04 68		STA CARGA+1	
15C2 C7 04 7A		STA OBT+1	
15C5 D6 07 00		LDA SP0,X	
15C8 5C		INCX	
15C9 C7 04 69		STA CARGA+2	
15CC C7 04 7B		STA OBT+2	
15CF BF 32		STX SP	
15D1 AE 01		LDX #1	
15D3 CD 04 73		JSR OBT0	
15D6 BE 32		LDX SP	
15D8 5C		INCX	
15D9 DB 07 00		ADD SP0,X	
15DC AE 01		LDX #1	
15DE CD 04 5D		JSR CARGAO	
15E1 5F		CLR X	
15E2 CD 04 73		JSR OBT0	
15E5 BE 32		LDX SP	
15E7 D9 07 00		ADC SP0,X	
15EA 5F		CLR X	
15EB CD 04 5D		JSR CARGAO	
15EE 3C 32		INC SP	
15F0 3C 32		INC SP	
15F2 CC 04 1E		JMP NEXT	

15F5 06	*	FCB 6	LATEST (-- dir)
15F6 4C 41 54		FCC 'LAT'	deja la direccion del campo de nombre
15F9 15 B3		FDB PLSTO-6	de la palabra en la cima del vocabulario
15FB CC 04 00	LAT6	JMP DOCOL	actual
15FE 14 D7		FDB CUR7	
1600 14 1C		FDB FTCH	
1602 14 1C		FDB FTCH	
1604 12 09		FDB EXIT	
1606 05	*	FCB 5	ALLOT (n --)
1607 41 4C 4C		FCC 'ALL'	suma n bytes al campo de parametros de la
160A 15 F5		FDB LAT6-6	definicion mas reciente.
160C CC 04 00	ALL5	JMP DOCOL	
160F 14 4C		FDB DP2	
1611 15 B9		FDB PLSTO	
1613 12 09		FDB EXIT	
1615 03	*	FCB 3	LIT (-- n)
1616 4C 49 54		FCC 'LIT'	dentro de una definicion de dos puntos
1619 16 06		FDB ALL5-6	LIT se compilka automaticamente antes de
161B B6 2F	LIT3	LDA IP	cada literal numerica de 16 bits en el
161D C7 04 7A		STA OBT+1	texto de entrada.Su ejecucion hara que
1620 B6 30		LDA IP+1	el contenido de la siguiente direccion
1622 C7 04 7B		STA OBT+2	del diccionario se coloque en la pila
1625 AE 01		LDX #1	
1627 CD 04 73		JSR OBT0	
162A BE 32		LDX SP	
162C 5A		DECX	
162D D7 07 00		STA SPO,X	
1630 BF 32		STX SP	
1632 5F		CLR X	
1633 CD 04 73		JSR OBT0	
1636 BE 32		LDX SP	
1638 5A		DECX	
1639 D7 07 00		STA SPO,X	
163C BF 32		STX SP	
163E A6 02		LDA #2	
1640 BB 30		ADD IP+1	
1642 B7 30		STA IP+1	
1644 4F		CLRA	
1645 B9 2F		ADC IP	
1647 B7 2F		STA IP	
1649 CC 04 1E		JMP NEXT	
164C BE 32	*		
164E 5C	QIMM	LDX SP	
164F D6 07 00		INCX	
1652 4D		LDA SPO,X	
1653 2B 03		TSTA	
1655 4F		BMI QID	
		CLRA	

```

1656 20 02      BRA QSKIP
1658 A6 FF      QID   LDA #$FF
165A D7 07 00   QSKIP STA SP0,X
165D 5A          DECX
165E D7 07 00   STA SP0,X
1661 CC 04 1E   JMP NEXT

```

*

*

*

```

1664 04      LOK   FCB 4
1665 20 4F 4B OK   FCC 'OK'
1668 0D      FCB CR

```

El Mensaje de Forth

*

*

INTERPRETE EXTERNO

*

```

1669 08 68      EXTER FDB COU5
166B 04 86      FDB TYPE
166D 12 37      FDB INLINE
166F 17 E1      FDB DFND
1671 11 E2      FDB ZBRAN
1673 00 1E      FDB $001E
1675 16 4C      FDB QIMM
1677 14 73      FDB NOT3
1679 11 E2      FDB ZBRAN
167B 00 10      FDB $0010
167D 14 C3      FDB STA5
167F 14 1C      FDB FTCH
1681 11 E2      FDB ZBRAN
1683 00 08      FDB $0008
1685 15 3F      FDB COMA
1687 05 76      FDB BRAN
1689 FF E6      FDB $FFE6
168B 12 22      FDB EXE7
168D 05 76      FDB BRAN
168F FF E0      FDB $FFE0
1691 14 64      FDB HERE
1693 13 45      FDB NUM8
1695 11 E2      FDB ZBRAN
1697 00 14      FDB $0014
1699 14 C3      FDB STA5
169B 14 1C      FDB FTCH
169D 11 E2      FDB ZBRAN
169F FF D0      FDB $FFD0
16A1 0D 13      FDB COMP
16A3 16 1B      FDB LIT3
16A5 15 3F      FDB COMA
16A7 05 76      FDB BRAN
16A9 FF C6      FDB $FFC6
16AB 08 00      FDB QUES
16AD 05 76      FDB BRAN
16AF FF BA      FDB $FFBA

```

ENVIA EL MENSAJE
DE FORTH.

RECIBE CARACTERES DE ENTRADA
FORMA UNA UNIDAD DE LEXICO

SI NO ES UNA PALABRA,
SALTA 30 LOCALIDADES ADELANTE, POR SI ES NUMERO
SI ES PALABRA, ESTA EN MODO INMEDIATO?

INVIERTE LA RESPUESTA
SI ES INMEDIATA, SALTA
16 LOCALIDADES ADELANTE PARA EJECUTARLA.
DETERMINA EL ESTADO
DEL SISTEMA.

SI ES CERO (ESTADO DE EJECUCION)
SALTA 8 LOCALIDADES PARA EJECUTAR LA PALABRA
SINO, ASIGNA DOS LOCALIDADES PARA
COMPILARLA Y REGRESAR A
FORMAR OTRA PALABRA.
EJECUTA LA PALABRA CUYA DIRECCION ESTA EN
LA PILA Y REGRESA A
FORMAR OTRA PALABRA

ES UN NUMERO?

SI L.O ES, AVANZA
20 LOCALIDADES ADELANTE.

SI NO ESTA EN MODO
DE COMPILACION, COLOCA
NUMERO EN LA PILA Y
REGRESA A FORMAR OTRA PALABRA.

SI ESTA COMPILANDO, COLOCA
LIT Y EL VALOR NUMERICO
DENTRO DE LA DEFINICION.
REGRESA A FORMAR
OTRA UNIDAD DE LEXICO.

EMITE UNIDAD DE LEXICO CON SIGNO DE
INTERROGACION (ERROR) Y REGRESA A
ACEPTAR NUEVA CUERDA DE ENTRADA.


```

*
*      RUTINA DE INICIALIZACION AL ENCENDER
*
16B1 04      FCB 4      COLD ( -- )
16B2 43 4F 4C      FCC 'COL'      ajusta el apuntador del diccionario al
16B5 16 15      FDB LIT3-6      estandard minimo y restablecerse via
16B7 16 05      COLD      BSET 3,$05      ABORT. puede llamarse desde la terminal
16B9 16 01      BSET 3,PUT      para remover un programa de aplicacion
16BB AE 17      LDX #$17      y volver a empezar
16BD D6 10 00      SDAT      LDA SEG2,X      hace la copia de segmentos modificables.
16C0 E7 28      STA $2B,X
16C2 5A      DECX
16C3 A3 FF      CPX #$FF
16C5 26 F6      BNE SDAT
16C7 5F      CLRX
16CB D6 10 18      SREPT      LDA SEG2+24,X
16CB D7 04 00      STA $400,X
16CE 5A      DECX
16CF 26 F7      BNE SREPT
16D1 5F      CLRX
16D2 D6 11 18      SREP2      LDA SEG2+$100+24,X
16D5 D7 05 00      STA $500,X
16D8 5A      DECX
16D9 26 F7      BNE SREP2

*
*      CALCULA LOS VALORES INFERIOR Y SUPERIOR DE EXTER
*
1600      HO      EQU EXTER/$100*$100
0069      LO      EQU EXTER-HO
0016      HO1     EQU EXTER/$100

*
16DB A6 69      LDA #LO
16DD B7 2E      STA INICIO+1
16DF A6 16      LDA #HO1
16E1 B7 2D      STA INICIO

*
*
*
0F00      HCR      EQU ULTIMO/$100*$100
00DC      LCR      EQU ULTIMO-HCR
000F      H1CR     EQU ULTIMO/$100

*
16E3 A6 0F      LDA #H1CR
16E5 B7 38      STA USER+FORTH
16E7 A6 DC      LDA #LCR
16E9 B7 39      STA USER+FORTH+1

*
*
*
1600      H      EQU LOK/$100*$100

```

0064	L	EQU LOK-H	
0016	H1	EQU LOK/\$100	
	*		
16EB 5F		CLR X	
16EC A6 64		LDA #L	coloca mensaje de inicio
16EE 5A		DECX	
16EF D7 07 00		STA SP0,X	
16F2 A6 16		LDA #H1	
16F4 5A		DECX	
16F5 D7 07 00		STA SP0,X	
16F8 BF 32		STX SP	inicializa el apuntador de pila
16FA A6 80	WARM	LDA #\$80	inicializa terminadores
16FC C7 05 FE		STA ICE+\$2E	
16FF C7 05 FF		STA ICE+\$2F	
1702 3F 36		CLR USER+STATE	coloca el sistema en estado de ejecucion
1704 3F 37		CLR USER+STATE+1	
1706 3F 31		CLR RP	inicializa apuntador de pila de retorno
1708 CD 17 62		JSR CRLF	
170B CC 08 23		JMP SIGUE	carga el IP y ve a ejecutar
	*		
	*		
170E 04		FCB 4	SWAP (n1 n2 -- n2 n1)
170F 53 57 41		FCC 'SWA'	intercambia los dos valores en la
1712 16 B1		FDB COLD-6	cima de la pila de datos
1714 BE 32	SWAP	LDX SP	
1716 D6 07 00		LDA SP0,X	
1719 5C		INCX	
171A B7 20		STA PH	
171C D6 07 00		LDA SP0,X	
171F 5C		INCX	
1720 B7 21		STA PL	
1722 D6 07 00		LDA SP0,X	
1725 5C		INCX	
1726 B7 24		STA QH	
1728 D6 07 00		LDA SP0,X	
172B B7 25		STA QL	
172D B6 21		LDA PL	
172F D7 07 00		STA SP0,X	
1732 B6 20		LDA PH	
1734 5A		DECX	
1735 D7 07 00		STA SP0,X	
1738 B6 25		LDA QL	
173A 5A		DECX	
173B D7 07 00		STA SP0,X	
173E B6 24		LDA QH	
1740 5A		DECX	
1741 D7 07 00		STA SP0,X	
1744 CC 04 1E		JMP NEXT	
	*		
1747 03		FCB 3	SPI (--)
1748 53 50 21		FCC 'SPI'	inicializa el apuntador de pila

174B 17 0E	FDB SWAP-6	desde S0
174D 3F 32	SPSTO CLR SP	
174F CC 04 1E	JMP NEXT	
*		
* RUTINAS DE ENTRADA/SALIDA PUERTO SERIE		
*		
1752 BF 14	OBTC STX XTEMP	
1752	OBTCARA EQU OBTC	
1754 CD 1F 31	JSR TRAE	
1757 BE 14	LDX XTEMP	
1759 81	RTS	
*		
175A BF 14	PUTC STX XTEMP	
175A	SALCARA EQU PUTC	
175C CD 1F 65	JSR ENVIA	
175F BE 14	LDX XTEMP	
1761 81	RTS	
*		
1762 A6 0D	CRLF LDA #CR	
1764 CD 17 5A	JSR SALCARA	
1767 A6 0A	LDA #LF	
1769 CD 17 5A	JSR SALCARA	
176C 81	RTS	
*		
176D 02	FCB 2	CR (--)
176E 43 52 20	FCC 'CR '	provoca cambio de linea
1771 17 47	FDB SPSTO-6	
1773 AD ED	CR2 BSR CRLF	
1775 CC 04 1E	JMP NEXT	
*		
1778 06	FCB 6	CREATE (--)
1779 43 52 45	FCC 'CRE'	permite crear una entrada al diccionario
177C 17 6D	FDB CR2-6	para <name> sin asignar ningun campo de
177E CC 04 00	CRE6 JMP DOCOL	parametros en memoria.
1781 12 A4	FDB BL2	
1783 12 BC	FDB WORD	
1785 16 1B	FDB LIT3	
1787 00 04	FDB 04	
1789 16 0C	FDB ALL5	
178B 15 FB	FDB LAT6	
178D 15 3F	FDB COMA	
178F 14 D7	FDB CUR7	
1791 14 1C	FDB FTCH	
1793 14 EB	FDB STO	
1795 12 09	FDB EXIT	
*		
1797 06	FCB 6	TOGGLE (dir b --)
1798 54 4F 47	FCC 'TOG'	Complementa el contenido de dir con el
179B 17 7B	FDB CRE6-6	patron b.
179D BE 32	TOG6 LDX SP	
179F 5C	INCX	

17A0 D6 07 00		LDA SPO,X	
17A3 5C		INCX	
17A4 B7 15		STA ATEMP	
17A6 D6 07 00		LDA SPO,X	
17A9 5C		INCX	
17AA C7 04 68		STA CARGA+1	
17AD C7 04 7A		STA OBT+1	
17B0 D6 07 00		LDA SPO,X	
17B3 5C		INCX	
17B4 BF 32		STX SP	
17B6 C7 04 69		STA CARGA+2	
17B9 C7 04 7B		STA OBT+2	
17BC 5F		CLR X	
17BD CD 04 73		JSR OBT0	
17C0 B8 15		EOR ATEMP	
17C2 CD 04 5D		JSR CARGAO	
17C5 CC 04 1E		JMP NEXT	
*			
17C8 09		FCB 9	IMMEDIATE (--)
17C9 49 4D 4D		FCC 'IMM'	marca la entrada mas reciente al diccionario
17CC 17 97		FDB TOG6-6	como inmediata; es decir, ser ejecutada en
17CE CC 04 00	IMM9	JMP DOCOL	lugar de compilarse.
17D1 15 FB		FDB LAT6	
17D3 16 1B		FDB LIT3	
17D5 00 80		FDB \$80	
17D7 17 9D		FDB TOG6	
17D9 12 09		FDB EXIT	
*			
17DB 05		FCB 5	-FIND
17DC 2D 46 49		FCC '-FI'	Busca una palabra dentro del diccionario.
17DF 17 C8		FDB IMM9-6	
17E1 CC 04 00	DFND	JMP DOCOL	
17E4 12 A4		FDB BL2	
17E6 12 BC		FDB WORD	
17E8 14 CD		FDB CON7	
17EA 14 1C		FDB FTCH	
17EC 14 1C		FDB FTCH	
17EE 04 C7		FDB FIN6	
17F0 12 09		FDB EXIT	
*			

Errors: 0

APENDICE C: PROGRAMA DE CONVERSION DE FORMATOS

```

/*****
*
* Programa para convertir un archivo en formato S1-S9 de
* MOTOROLA al formato HEX de INTEL.
* El programa puede generarse como S1S9_HEX.c utilizando
* TURBOC u otro compilador de C. Debe obtener un ejecuta-
* ble con la extensión .EXE; este programa generará un
* archivo en formato Hex a partir de un archivo en forma-
* to S1-S9 generado por el ensamblador cruzado AS5 bajo
* el nombre de M.out. Si lo desea, puede renombrar este
* archivo.
* El nombre del archivo fuente y del archivo destino de-
* deben darse junto al nombre del programa ejecutable:
*
*                               S1S9_HEX  fuente destino
*
*                               ENSENADA, BAJA CALIFORNIA      ABRIL 1988
*
*****/

```

```
#include <stdio.h>
```

```

main(argc,argv) int argc; char **argv;
{
    FILE *fuente, *destino;
    char a,b,c;
    int longitud,dir1,dir2,numero,i;
    unsigned int suma;

    if (argc !=3)
    {
        printf("faltan nombres de archivos !\n");
        exit(1);
    }
    fuente=fopen(argv[1],"r");
    destino=fopen(argv[2],"w");
    if (fuente==NULL)
    {
        printf("no existe archivo fuente\n");
        exit(1);
    }
    fscanf(fuente,"%c%c",&a,&b);
    printf("principia conversion...");

    while ( a=='S' && b=='1')
    {
        fscanf(fuente,"%2x%2x%2x",&longitud,&dir1,&dir2);
        suma=longitud - 3 + dir1 + dir2;
    }
}

```

```

fprintf(destino, ":%02X%02X%02X00",
        longitud-3, dir1, dir2);
for (i=0; i<(longitud-3); i++)
{
    fscanf(fuente, "%2x", &numero);
    fprintf(destino, "%02X", numero);
    suma+=numero;
} /* del for */

fscanf(fuente, "%2x", &numero);
suma=~suma;
suma++;
suma &= 0x00FF;
fprintf(destino, "%02X\n", suma);
fscanf(fuente, "%c%c%c", &c, &a, &b);
) /* del while */

fscanf(fuente, "%2x%2x%2x", &longitud, &dir1, &dir2);
suma=longitud - 3 + dir1 + dir2;
fscanf(fuente, "%2x", &numero);
suma=~suma;
suma &= 0x00FF;
fprintf(destino, ":%02X%02X%02X0100", longitud-3, dir1, dir2);
fclose(fuente);
fclose(destino);
printf("Conversion concluida\n");
)

```


APENDICE D: PRIMITIVAS DEL DIALECTO FORTH/05

La descripción de una primitiva de FORTH/05 especifica la forma en que debe ser escrita para ser invocada, los valores que espera encontrar en la pila de datos y los valores que colocará después de su ejecución. Estos valores se encuentran entre paréntesis: los primeros valores antes de "---" representan los valores en la pila antes de la ejecución de la palabra, los valores siguientes a "---" son los valores que dejará en la pila de datos. En ambos casos, el orden de los valores dentro de la pila corresponden al orden en que aparecen enlistado, siendo el último el valor en la cima. Si no aparece ningún valor antes o después, o en ambos, no se afecta el contenido de la pila.

Los valores se han abreviado como sigue:

dir	es una dirección de dos octetos
n	es un número de 16 bits
car	es un caracter (8 bits)
d	es un número de 32 bits
byte	es un número de 8 bits
un	es un número de 16 bits sin signo
ud	es un número de 32 bits sin signo

ENTRADA/SALIDA DE TERMINAL

TYPE (dir n --)

Imprime cuerda de n caracteres que se encuentra a partir de la dirección dir , si n>0.

EMIT (car --)

Imprime el caracter ASCII correspondiente al valor en la cima de la pila.

WORD (car -- dir)

Lee la siguiente palabra de la corriente de entrada usando car como delimitador o hasta encontrar el nulo. Deja en dir la dirección del byte de longitud.

CR (--)

Provoca un retorno de carro y un cambio de línea.

MANEJO DE PILA

DROP (n --)

Desecha el valor en la cima de la pila.

DUP (n -- n n)

Duplica la cima de la pila.

SWAP (n1 n2 -- n2 n1)

Intercambia los dos valores en la cima de la pila.

OVER (n1 n2 -- n1 n2 n1)

Hace una copia del segundo elemento en la cima de la pila.

>R (n --)

Mueve el elemento en la cima de la pila a la pila de retorno para almacenamiento temporal.

R> (-- n)

Regresa el elemento en la cima de la pila de retorno a la pila de datos.

R@ (-- n)

Copia la cima de la pila de retorno a la pila de datos.

ROT (n1 n2 n3 -- n2 n3 n1)

Rota el tercer elemento de la pila

DDUP (d -- d d)

Duplica el número de doble precisión en la pila.

MEMORIA

c@ (dir -- byte)

Busca y trae el byte menos significativo contenido en la localidad dir.

@ (dir -- n)

Busca y trae a la pila el valor contenido en dir.

! (n dir --)

Almacena n en la localidad dir.

c! (n dir --)

Almacena el byte menos significativo de n en la localidad dir.

+! (n dir --)

Suma n al número en la localidad dir.

COMPARACION

NOT (n -- n')

Complemento a 1's del valor en la cima

0< (n -- bandera)

Bandera verdadera si el valor en la cima es negativo.

0= (n -- bandera)

Bandera verdadera si el valor en la cima es cero.

< (n1 n2 -- bandera)

Bandera verdadera si n1 es menor que n2.

> (n1 n2 -- bandera)

Bandera verdadera si n1 es mayor que n2.

= (n1 n2 -- bandera)

Bandera verdadera si los números son iguales.

ARITMETICAS/LOGICAS

1+ (n -- n+1)

suma 1 al valor en la cima.

0 (-- 0)

Coloca 0 en la pila.

1	(-- 1)	Coloca 1 en la pila.
2	(-- 2)	Coloca 2 en la pila.
2+	(n -- n+2)	Suma 2 al valor en la cima.
+	(n1 n2 -- suma)	Suma los dos valores en la cima de la pila.
-	(n1 n2 -- diferencia)	Resta n2 de n1.
U*	(un1 un2 -- ud)	Multiplica dos números de precisión sencilla sin signo, dejando un resultado de precisión doble sin signo.
U/MOD	(ud un -- ures ucoc)	Divide un número de precisión doble sin signo entre uno de precisión sencilla sin signo, dejando el residuo y el cociente sin signo.
M/MOD	(ud1 u2 -- u3 ud4)	Divide un número doble sin signo entre uno de precisión sencilla sin signo, dejando el residuo de precisión sencilla y el cociente de precisión doble, ambos sin signo.
ABS	(n -- n)	Valor absoluto de la cima de la pila.
NEGATE	(n -- -n)	Complemento a 2's del valor en la cima.

OR (n1 n2 -- or)

O lógico de bits

AND (n1 n2 -- and)

Y lógico de bits

XOR (n1 n2 -- xor)

O exclusivo lógico de bits.

* (n1 n2 -- producto)

Multiplicación.

DNEGATE (d -- -d)

Complemento a 2's del número de doble precisión en la cima de la pila.

ESTRUCTURAS DE CONTROL

EXIT (--)

Termina la ejecución de una definición de dos puntos.

EXECUTE (dir --)

Ejecuta la palabra en el diccionario cuya dirección de compilación está en la pila.

COLD (--)

Ajusta el apuntador del diccionario al estándar mínimo. Se utiliza para remover un programa de aplicación y volver a la situación inicial del sistema.

QUIT (--)

Limpia la pila de retorno, se coloca en modo de ejecución y regresa el control a la terminal.

DO...LOOP (final+1 inicio --)

Inicializa un ciclo de repetición de acuerdo al rango marcado por los índices dados como inicio y final+1.

I (-- Indice)

Coloca el índice actual del ciclo en la pila de datos.

DO...+LOOP

DO (límite inicio --)
+LOOP (n --)

Similar a DO...LOOP, pero suma el valor en la cima de la pila al índice (en lugar de incrementarse siempre en 1). El ciclo termina cuando el índice es mayor o igual a límite ($n \geq 0$), o cuando el índice es menor que límite ($n < 0$).

<DO> (n1 n2 --)

Procedimiento al tiempo de corrida compilado por DO que envía los parámetros de control del ciclo a la pila de retorno.

<LOOP> (--)

Procedimiento al tiempo de corrida compilado por LOOP que incrementa el índice del ciclo y prueba por la culminación de éste.

<+LOOP> (n --)

Procedimiento al tiempo de corrida compilado por +LOOP, que incrementa el índice del ciclo y prueba por su terminación.

IF...xxx...THEN
(bandera --)

Si la cima de la pila es verdadera, ejecuta xxx.

IF...xxx...ELSE...yyy...THEN
 (bandera --)

Si la cima de la pila es verdadera, ejecuta xxx; si es falsa , ejecuta yyy.

BEGIN...bandera UNTIL
 UNTIL (bandera --)

Regresa el ciclo a BEGIN hasta que la bandera probada por UNTIL sea verdadera.

BEGIN...bandera WHILE...REPEAT
 WHILE (bandera --)

Cuando la bandera es verdadera, continua la ejecución hasta REPEAT, el cual regresa a la palabra después de BEGIN. Cuando la bandera es falsa, la ejecución se transfiere después de REPEAT, saliéndose de la estructura.

BEGIN...AGAIN

Efectua un salto incondicional al inicio de ciclo BEGIN.

CONVERSION

BASE (-- dir)

Variable del sistema que contiene el radix para la conversión numérica.

SIGN (n --)

Agrega el signo menos a la cuerda de salida si $n < 0$.

+- (n1 n2 -- n3)

Aplica el signo de n2 a n1, el cual se deja como n3.

(ud1 -- ud2)

Convierte el siguiente dígito de un número doble sin signo y agrega el carácter a la cuerda de salida.

S->D (n -- d)

Extiende el signo s un número de precisión sencilla al formar uno de doble precisión.

<# (--)

Inicia conversión numérica de salida.

#> (d -- dir n)

Descarta d y termina la conversión numérica de salida, dejando la dirección del inicio de la cuerda (dir) y el contador de caracteres de la cuerda formada, para que sea utilizada por TYPE.

HOLD (car --)

Agrega el carácter ASCII a la cuerda de salida.

#8 (ud -- 0 0)

Convierte todos los dígitos significativos de un número sin signo de precisión doble a una cuerda de salida.

. (n --)

Imprime número con un blanco antecediéndolo y signo, en caso que sea negativo.

<NUMBER> (dir -- n)

Convierte la cuerda de caracteres que empieza en dir en un entero con signo de 16 bits.

COMPILACION

STATE (-- dir)

Variable del sistema cuyo valor es diferente de cero cuando está en modo de compilación.

, (n --)

Compila un número en el diccionario.

C, (n --)

Guarda los ocho bits menos significativos de n en el siguiente byte en el diccionario, incrementando DP.

ALLOT (n --)

Agrega n bytes al campo de parámetros de la palabra definida mas recientemente.

LIT (-- n)

Dentro de una definición de dos puntos LIT se compila automáticamente antes de cada literal numérica encontrada en el texto de entrada. La posterior ejecución de LIT causa que el contenido de la siguiente dirección del diccionario se coloque en la pila.

[(--)

Termina la compilación del texto de entrada y empieza su ejecución.

] (--)

Termina la ejecución del texto de entrada y empieza su compilación.

COMPILE (--)

Compila la dirección de la siguiente palabra no inmediata dentro del diccionario.

[COMPILE] (--)

Compila la siguiente palabra, aún si es inmediata.

<."> (--)

Procedimiento al tiempo de corrida compilado por "." que transmite el siguiente texto en línea al dispositivo de salida.

VOCABULARIO

CONTEXT (-- dir)

Variable del sistema que apunta al vocabulario donde son buscados los nombres de las palabras.

CURRENT (-- dir)

Variable del sistema que devuelve la dirección del vocabulario donde serán puestas las nuevas definiciones.

FORTH (--)

Vocabulario principal.

LATEST (-- dir)

Deja la dirección del campo de nombre de la palabra en la cima del vocabulario actual.

DEFINITIONS (--)

Coloca el vocabulario CURRENT como CONTEXT.

' xxx (--)

Encuentra dirección de xxx en el diccionario. Si se usa en una definición, compila la dirección.

DEFINICION

CREATE (--)

Crea una entrada en el diccionario sin asignar memoria para el campo de parámetros.

IMMEDIATE (--)

Marca la entrada al diccionario mas recientemente hecha como una palabra que será ejecutada cuando se encuentre durante la compilación, en lugar de compilarse.

; (--)

Termina definición de dos puntos.

: xxx (--)

Inicia definición dos puntos de xxx.

VARIABLE xxx (--)

Crea una variable de dos bytes llamada xxx. Regresa la dirección cuando se ejecuta.

CONSTANT xxx (n --)

Crea una constante llamada xxx con valor n; regresa el valor cuando se ejecuta.

ADQUISICION DE DATOS

LEEADC (n -- dato)

Lee del canal n del convertidor analógico digital y coloca el valor leído en la pila.

DIVERSAS

BL (-- car)

Constante que deja el valor ASCII para el espacio en blanco en la pila.

DP (-- dir)

Variable del usuario, apuntador al diccionario, que contiene la dirección de la siguiente localidad de memoria disponible encima del diccionario.

HERE (-- dir)

Deja la dirección de la siguiente localidad disponible en el diccionario.

SP! (--)

Procedimiento para inicializar el apuntador de pila.

TOGGLE (dir b --)

Complementa el contenido de dir por el patrón de bits b.

-FIND (-- pfa b bandera_verdadera) o
(-- bandera_falsa)

Busca la palabra de la corriente de entrada en el diccionario. Si la encuentra, devuelve la dirección del campo de nombre, el byte de longitud y bandera verdadera; si no la encuentra, devuelve una bandera falsa.

COUNT (dir -- dir+1 n)

Deja la dirección dir+1 y el contador de caracteres de la cuerda al inicio de dir. El primer caracter de la cuerda debe ser el contador de caracteres.

SMUDGE (--)

Usada durante la definición de una palabra para acoplar el bit de marca en el campo de nombre de la definición. Esto previene que una definición incompleta sea encontrada durante la búsqueda del diccionario hasta que la compilación termine sin error.

PAD (-- dir)

Deja la dirección de un área temporal.

TIB (-- dir)

Constante del usuario que contiene la dirección del área de almacenamiento de la corriente de entrada.

>IN (-- dir)

Variable del sistema que contiene el desplazamiento a caracteres dentro del área de almacenamiento de la entrada de texto.

'STREAM (-- dir)

Regresa la dirección del siguiente carácter en la corriente de entrada.

APENDICE E: INSTALACION DE FORTH/05

Para instalar FORTH/05 necesita tener el programa objeto en formato HEX listo para cargarse o realizar el ensamblado utilizando el ensamblador cruzado AS5, editar direcciones, convertir al formato HEX empleando el programa S1S9_HEX.EXE. Después procederá a cargar al programa al ROMULATOR a través del programa ROMUL8.EXE. Es recomendable tenga en un disco flexible los siguientes programas:

- * Programa fuente de FORTH/05 (FORTH/05.ASM),
- * Ensamblador Cruzado AS5,
- * Programa de conversión de formatos (S1S9_HEX.EXE),
- * Cargador al emulador de ROMs ROMULATOR (ROMUL8.EXE),
- * Un editor de texto.

Por lo que respecta a partes mecánicas necesita una microcomputadora IBM-PC o compatible, el emulador de ROMs ROMULATOR, la computadora en una tarjeta CUTCMOS, la tarjeta de expansión de memoria MEM-64 y una terminal. Deberá conectar la microcomputadora IBM-PC al ROMULATOR a través del puerto serie RS-232C y el ROMULATOR al la tarjeta MEM-64 que está conectada a la CUTCMOS por el ducto C-44. El ROMULATOR debe colocar el programa en las direcciones 0000H a la 0FFFH, que corresponden a las localidades del banco 0 en la MEM-64, por lo que debe conectarse a este banco a través de su interfase.

La CUTCMOS debe conectarse a la terminal por medio de una interfase en el puerto serie (bits 7 y 8 del puerto A).

Para cargar FORTH/05, inicialize la CUTCMOS y entrará el monitor a trabajar. Por medio de éste, invoque a FORTH/05 a través del comando G 16B7.