UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE CIENCIAS

DISEÑO E IMPLEMENTACION DE UN MACROENSAMBLADOR PARA MICROCOMPUTADORA M6800

T E S I S

QUE PARA OBTENER EL TITULO DE:

A C T U A R I O

P R E S E N T A:

CAROLINA RUIZ LOPEZ

MEXICO, D. F. MARZO 1982

DISEÑO E IMPLEMENTACION DE UN MACROENSAMBLADOR PARA MICROCOMPUTADORA M6800.

INTRODUCCION

En los últimos años ha surgido un nuevo tipo de computadoras:- las microcomputadoras. Estas computadoras poseen entre otras características las de ser baratas, pequeñas y de fácil manejo.

Las microcomputadoras - como toda computadora - solo entienden el lenguaje máquina sin embargo este lenguaje es difícil de programar.

Cuando se trabaja con computadoras se espera no solo poder ejecutar rápidamente un programa sino poder prepararlo y corregirlo - fácilmente, en consecuencia las mismas computadoras proporcionan - diferentes lenguajes de programación - que facilitan la comunica-- ción entre el hombre y la máquina - y programas traductores que -- transforman estos lenguajes a lenguaje de máquina.

Entre los lenguajes más antiguos se encuentran los lenguajes - ensambladores.

El programa que realiza la traducción del lenguaje ensamblador al lenguaje máquina recibe el nombre de ensamblador.

Los ensambladores además de ser traductores pueden proporcio-- nar al usuario otro tipo de facilidades. Una extensión del algoritmo básico del ensamblador es el macroprocesador; cuando esta aso-- ciación es muy fuerte el sistema se conoce como macroensamblador, por lo tanto el macroensamblador es la combinación de 2 programas distintos.

El macroensamblador se considera como una extención del repertorio de instrucciones de una computadora.

El macroensamblador que se presenta está diseñado para el lenguaje ensamblador de una microcomputadora M6800. Se pretende que -

los usuarios de la M6800 puedan trabajar con el lenguaje ensamblador de esa máquina utilizando la minicomputadora PDP - 11 / 34. Ambas computadoras pertenecen al laboratorio de computación de la -- Facultad de Ciencias de la UNAM.

Uno de los objetivos de implementar el macroensamblador para - la microcomputadora M6800 es demostrar que los usuario pueden desarrollar parte de su sistema operativo obteniendo como resultado un beneficio económico y suprimiendo un poco la dependencia con los - fabricantes de computadoras que generalmente son de procedencia extranjera.

Además se pretende que la tesis sirva como guía a quién desee entender el funcionamiento de un ensamblador y de un macroprocesador o para quien desee implementarlo.

Para poder diseñar un ensamblador es necesario conocer el funcionamiento y el lenguaje ensamblador de la máquina a la que va dirigido, es por ello que el capítulo I describe el hardware de las microcomputadoras mencionando las características principales de - la M6800 y el capítulo II es esencialmente un manual del lenguaje ensamblador para esta máquina.

En el capítulo III se presenta lo que es un ensamblador y una macro tratándolos en forma general y sin enfocarlos al lenguaje ensamblador de una máquina específica. Se podría decir que en este - capítulo se especifica el problema de la tesis mientras que en el capítulo IV se resuelve el problema al diseñar el ensamblador y un macroprocesador para la microcomputadora M6800. Los diagramas de - flujo se encuentran en este capítulo.

El apéndice A muestra una lista de las instrucciones de la mi-

cro M6800 junto con los modos que admite. Al analizar este apéndice surgieron las principales ideas para diseñar el ensamblador.

El apéndice B es un resumen de la sintaxis para el lenguaje ensamblador de la microcomputadora M6800.

El apéndice C contiene el formato de salida para cualquier sentencia ensamblada.

En el apéndice D se proporciona una lista de los errores de -- sintaxis para que el usuario pueda corregir sus errores y sepa la razón por la cual se produjeron.

En el apéndice E se presentan varios ejemplos de programas ensamblados.

Finalmente el apéndice F contiene el listado del ensamblador. El programa está escrito en el lenguaje ensamblador Macro - 11.

CAPITULO I

La evolución de la tecnología electrónica en los pasados años, tuvo como consecuencia una nueva generación de computadoras: las - microcomputadoras.

Uno de los aspectos fundamentales que debe entenderse al utilizar computadoras es su sistema físico (hardware); por lo cual la finalidad de este capítulo es para que se adquiera un conocimiento básico del hardware de las microcomputadoras y la importancia de - estas máquinas.

Además de presentar una breve historia de su desarrollo se describe la función de cada una de las componentes de una microcomputadora para al final dar las características de una microcomputadora M6800.

1.1 Generalidades de las microcomputadoras.

1.1.1 Definición.

Las microcomputadoras son verdaderas computadoras.

El prefijo micro se refiere al tamaño, estructura y técnica de fabricación, nunca al funcionamiento o versatilidad de la máquina.

Una micro es realmente la microminiaturización del sistema de una computadora; todo el sistema puede implementarse en una o dos placas de silicón con circuitos integrados (chips).

1.1.2 Antecedentes históricos.

La Universidad de Pennsylvania desarrolló en 1945 la primera - computadora electrónica que operaba con bulbos, la ENIAC.

En la primera generación de computadoras, el costo del hardware, software, la investigación y el desarrollo, eran caros.

Las máquinas eran difíciles de programar, operar y mantener. - Tenían la capacidad de controlar muchos procesos y procedimientos, por lo que requerían mecanismos de diversos tipos.

Los sistemas de la computadora eran complicados: encontrar y - cambiar un bulbo deficiente tomaba, a un equipo de diez personas,

de 24 a 30 horas. Hoy en día el cambiar un circuito defectuoso toma unos minutos a una sola persona.

No debe admirar, en consecuencia, que estas máquinas grandes, poco prácticas y caras, tuvieran usuarios limitados: ingenieros militares, científicos del espacio, matemáticos teóricos, etc.

Los esfuerzos para miniaturizar los componentes electrónicos no fueron motivados por su complejidad, sino porque los departamentos militares y del espacio de los Estados Unidos, requerían que estos componentes fueran confiables, ligeros y consumieran poca energía.

Posteriormente surge el transistor para formar parte de satélites y proyectiles.

En 1959, la compañía Fairchild Semiconductor desarrolló los circuitos integrados como se conciben actualmente. Para ello, logró la separación e interconexión de transistores y otros elementos, eléctricamente, en vez de físicamente.

Para 1961 se encontraban en el mercado circuitos comerciales de este tipo. Se inició así la época de las minicomputadoras.

La invención de la microcomputadora se debe a una fábrica japonesa de calculadoras pequeñas que requerían una pieza pequeña e inteligente para ejecutar funciones aritméticas y de otro tipo, con el requisito de ser lo suficientemente barata como para hacer de su producto, un producto de consumo masivo.

Esto lo logró en 1971, Intel Corporation de Santa Clara California, al elaborar un semiconductor a base de silicón, diminuto y potente, que podía hacer más que simples operaciones de cálculo. Nació así la primera microcomputadora de 4 bits y con ella una nue

va tecnología. En 1972 aparece la siguiente generación de micros - con palabras de 8 bits.

Una aportación que contribuyó a la microminiaturización fue la fotolitografía, proceso mediante el cual todas las plantillas que componen un circuito son, en su esencia, transferidas a la superficie del silicón. Otra causa fue el desarrollo del método de difusión del estado sólido, por medio del cual se introducen en el silicón impurezas que crean las regiones positivas y negativas.

Las primeras microcomputadoras se volvieron muy comerciales y populares de 1973 a 1974. De 1975 a 1976, una segunda y tercera ola de diseños de sistema suplantaron rápidamente a los primeros aparatos. Ejemplo de ello son:

El Intel 8085 que en 1976 efectuaba 770,000 sumas por segundo y tenía 20,000 transistores en cada placa.

Los micros de la tercera generación como el de la Motorola -- 680,000 que tienen 68,000 transistores y 16 bits.

La revolución microelectrónica está lejos de haber recorrido - todo su camino. Cada día se aprende más acerca de los circuitos integrados, se desarrollan nuevas teorías y se diseñan nuevos circuitos.

Los circuitos integrados que se fabrican actualmente no podrían existir sin la ayuda de aquellos que se hicieron en el pasado, ya que las computadoras intervienen en su diseño, supervisan los - procesos de fabricación, intervienen para probar si son defectuosas etc.

1.1.3 Ventajas.

La reducción en el tamaño de los circuitos no solo reduce el -

costo de las micro sino que mejora su funcionamiento.

Un circuito se vuelve más rápido en la medida en que se reduzca su tamaño. Similarmente la energía se reduce al disminuir el área en el circuito.

Los usuarios necesitan menos espacio en una habitación, menos energía operativa y menos o ningún aire acondicionado para el equipo.

Las micros son usadas por una amplia gama de personas: arquitectos, contadores, físicos, matemáticos, estudiantes y, en general, cualquier persona con suficiente inteligencia para formular procedimientos paso por paso.

1.1.4 Aplicaciones.

Actualmente las microcomputadoras las encontramos en:

Hornos de microondas

Instrumentos de laboratorio

Cajas registradoras

Juegos eléctricos para T.V.

Controlando señales de tránsito

Bancos, etc.

Las micro intervienen controlando procesos en las industrias - química, del petróleo, automotriz, del aeroespacio, etc.

El único límite para su aplicación, es el límite de la imaginación humana.

1.2 Elementos básicos de un sistema de microcomputadora. (ver fig. 1.1)

Son fundamentalmente:

1.2.1 La unidad central de proceso (CPU)

Sirve principalmente para realizar:

a) Las operaciones lógicas y aritméticas.

b) Funciones de control de E/S.

En una microcomputadora, estas funciones las lleva a cabo un - microprocesador (también designado como MPU).

Un microprocesador es un circuito digital integrado de menos - de 1/4 de pulgada cuadrada. De ahí se genera el término microprocesador.

El MPU es el responsable de ejecutar una tarea siguiendo las - instrucciones dadas por un programador.

El microprocesador contiene:

a) Circuitos para accesar las localidades apropiadas de memo-- ria y para interpretar los resultados de las instrucciones.

b) La unidad lógica y aritmética (ALU). Es una combinación - de redes que ejecutan las operaciones lógicas y aritméticas sobre los datos.

c) Una sección de control que dirige las operaciones de la computadora. Manda señales de control a otras unidades.

d) Varios registros de datos para almacenamiento temporal y manipulación de datos e instrucciones.

Se clasifican en tres tipos de registros:

1) Los que mantienen al CPU informado de lo que está haciendo en cualquier momento.

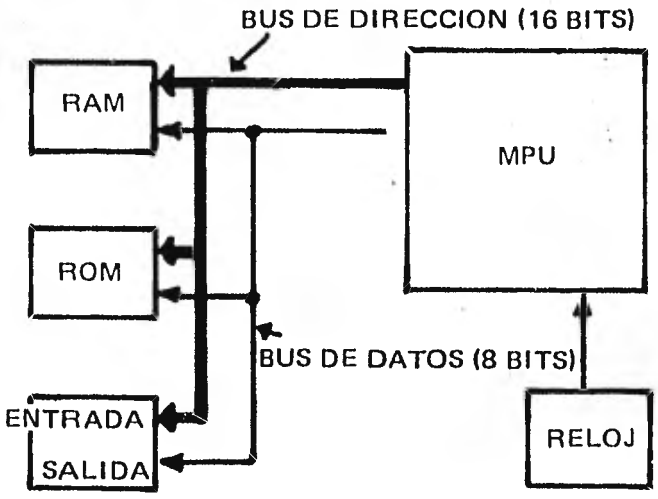


FIG. 1.1 SISTEMA DE UNA MICROCOMPUTADORA

ii) El registro contador de programa (PC). Registro que -- lleva cuenta de la ejecución del programa.
Durante la ejecución de la instrucción actual, el contenido del PC es actualizado para corresponder a la dirección de la siguiente instrucción que será ejecutada.
Se acostumbra decir que el PC señala a la instrucción -- que sera alimentada desde la memoria.

iii) Acumuladores. Facilitan un lugar para llevar a cabo las operaciones pedidas por la instrucción.

1.2.2 La memoria de acceso directo (RAM).

Esta es un área donde los datos pueden ser almacenados y borrados en cualquier momento para ser sustituidos por otros datos. Este tipo de memoria facilita y proporciona el acceso a cualquier localidad de memoria, por ello se encuentra generalmente organizada en arreglos de renglones y columnas.

La memoria de acceso directo lee o escribe operaciones en un período mínimo de tiempo, conocido como tiempo de ciclo.

1.2.3 La memoria de sólo lectura (ROM).

Para algunas de las aplicaciones de los microprocesadores se requiere de una memoria que almacene permanentemente información o que raramente se pueda alterar esa información.

Este tipo de almacenamiento - que puede leerse pero no destruirse - es proporcionado por la memoria de solo lectura (ROM) -- que es una memoria no volátil.

Las memorias de solo lectura pueden clasificarse en las programadas por:

1) Los fabricantes. La información se coloca durante su fabrica--

ción. Su contenido es fijado por los fabricantes que deciden el conjunto de instrucciones para la computadora.

2) Los usuarios. Las memorias programables por los usuarios (PROM) son de 2 tipos: las que después de ser programadas una vez no pueden cambiar y las que el usuario puede borrar y reprogramar.

El microprocesador opera bajo las instrucciones de un programa que se encuentra almacenado en localidades consecutivas del ROM.

1.2.4 Estructuras de E/S.

Siempre debe existir un medio de comunicación para mandar información dentro y fuera del sistema de microcomputadora. Para ello contamos con varios "buses".

Un bus consiste en un conjunto de líneas (una para cada bit que será transferido) que transfieren información de uno o varios dispositivos fuentes a uno o varios destinos.

En una microcomputadora se encuentra:

a) El bus de datos. Los datos se transmiten al interior y al exterior del microprocesador - generalmente a uno de los acumuladores - sobre el bus de datos que es un grupo de 8 líneas que conectan al procesador con la memoria y con otros dispositivos periféricos. Como los datos se transmiten en dos direcciones se dice que este bus es bidireccional.

b) El bus de dirección. Para accesar datos es necesario formular su dirección, así sabremos su localidad en memoria.

El microprocesador manda a través de este bus la dirección en memoria, de la localidad, por lo que cuenta con 16 líneas.

c) El bus de control. Hay un grupo de señales que entran y salen del microprocesador; algunas de ellas llevan señales de con---

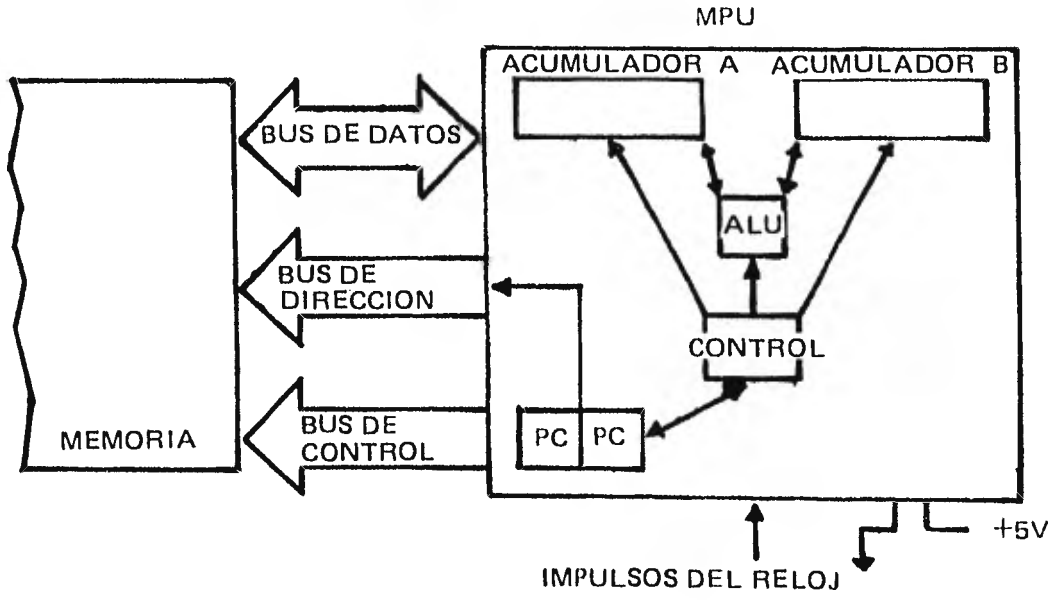


FIG. 1.2 SISTEMA DE UNA MICROCOMPUTADORA

trol de una parte a otra, entre el microprocesador, el ROM, el RAM y otros dispositivos. Estas señales se agrupan juntas y forman el bus de control. (ver figura 1.2)

1.2.5 El reloj.

Todo sistema de microcomputadora es regulado por señales de reloj para determinar cuándo deben ocurrir los acontecimientos.

1.3 Funcionamiento de una Microcomputadora

Para entender el funcionamiento de una micro, se seguirán las ejecuciones de una palabra que contenga:

- bits que formen una instrucción y
- bits que formen un dato.

Una microcomputadora funciona sincronizada por el reloj.

Se requiere de un número de ciclos para llevar a cabo la tarea especificada por una instrucción. (Ver fig. 1.3)

1) El contenido del contador de programa (PC) se coloca en el registro de direcciones de memoria (MAR) al principiar el ciclo.

2) El contenido del MAR se transfiere a la memoria y se decifra para determinar la palabra asignada.

3) La instrucción se lee desde la memoria, a través del bus de datos, hasta el registro de datos de memoria (MDR).

4) La instrucción se coloca en el registro de instrucción (IR) contenido en el MPU.

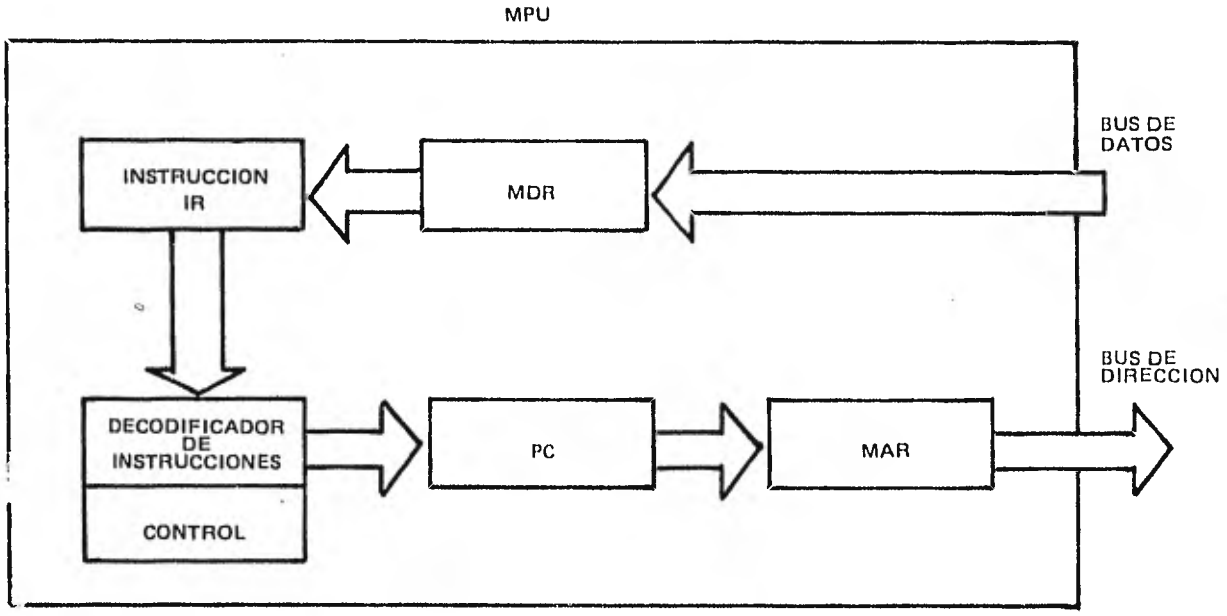
5) La instrucción se descifra por el decodificador de instrucciones.

6) La instrucción se ejecuta.

7) El contador de programa (PC) se incrementa de acuerdo a la instrucción que fué ejecutada.

La ejecución de una instrucción requiere frecuentemente accesar datos.

El dato puede entrar desde memoria o de un dispositivo de E/S. En muchas microcomputadoras la entrada del dato al MPU debe ser a través del acumulador. El acumulador también funciona como el



1.3 RECORRIDO DE UNA PALABRA CON INSTRUCCION

destino de todas las operaciones ejecutadas por la unidad lógica y aritmética (ALU). Después de que las operaciones finalizaron, -- las palabras con datos pasan a la memoria o a un dispositivo de -- E/S, utilizando el bus de E/S.

Todas las operaciones son controladas por la sección de con--- trol.

Las operaciones de palabras con datos tienen lugar en un ciclo de ejecución.

(ver figura 1.4)

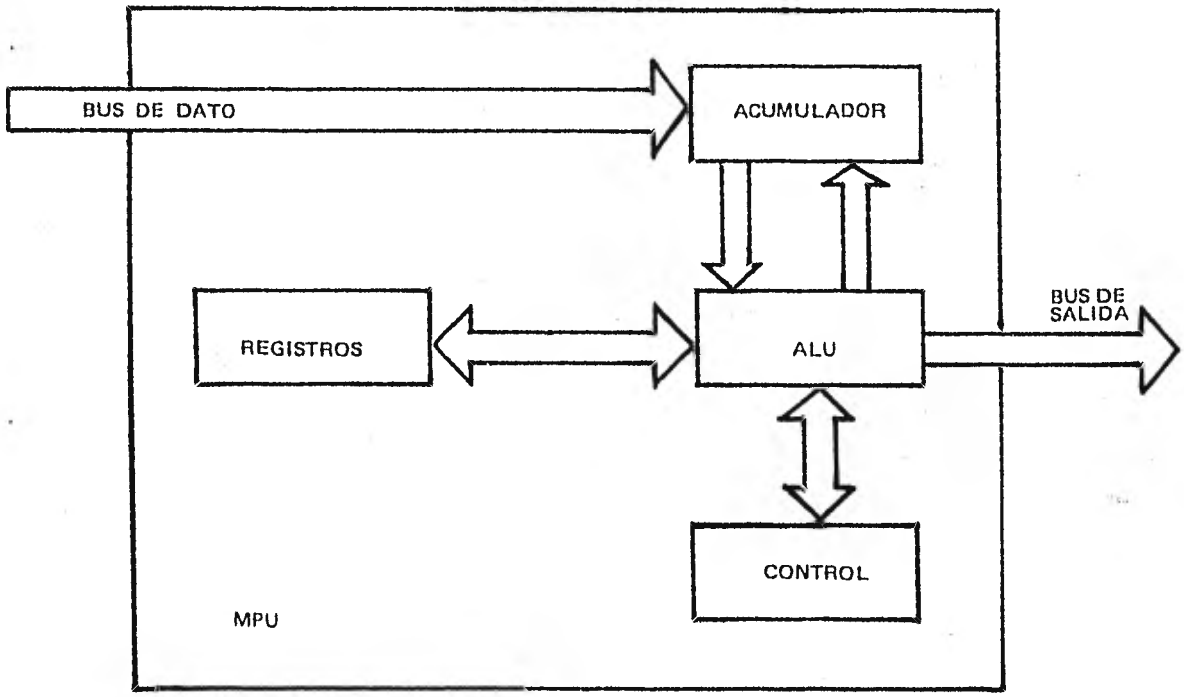


FIG. 1.4 RECORRIDO DE UNA PALABRA CON DATO

1.4 Características de la Microcomputadora M6800.

La familia de dispositivos M6800 permite el diseño de un sistema de microcomputadora con un mínimo de tiempo y esfuerzo. El núcleo de la familia M6800 es la unidad del microprocesador MC6800. (ver fig. 1.5)

1.4.1 Unidad del microprocesador MC6800.

Este MPU se introdujo en el mercado en 1974. Su precio varía - de 200 a 1000 dólares dependiendo de lo simple o complicado que se desee el sistema de microcomputadora.

Algunas de las características más importantes del MPU M6800 - que contribuye a su fácil uso son:

- Proceso paralelo de 8 bits.
- Bus de datos bidireccional.
- Bus de direccionamiento de 16 bits con 64 K bytes de direc-- cionamiento.
- 72 instrucciones de longitud variable.
- 7 modos de direccionamiento: directo, relativo, inmediato, - indexado, extendido, implicado y acumulador.
- 6 registros internos: dos acumuladores, registro índice, contador de programa, apuntador del stack y registro de condi-- ciones de código.
- Longitud variable del stack.
- Acceso directo de memoria (DMA) y procesador de capacidad múltiple.
- Reloj con tasas de operación mayores de $1MH_z$.
- Interconexión simple con TTL (Transistor-Transistor-Lógico).
- Interruptor separado sin máscara. Se usa para señalar cuán-

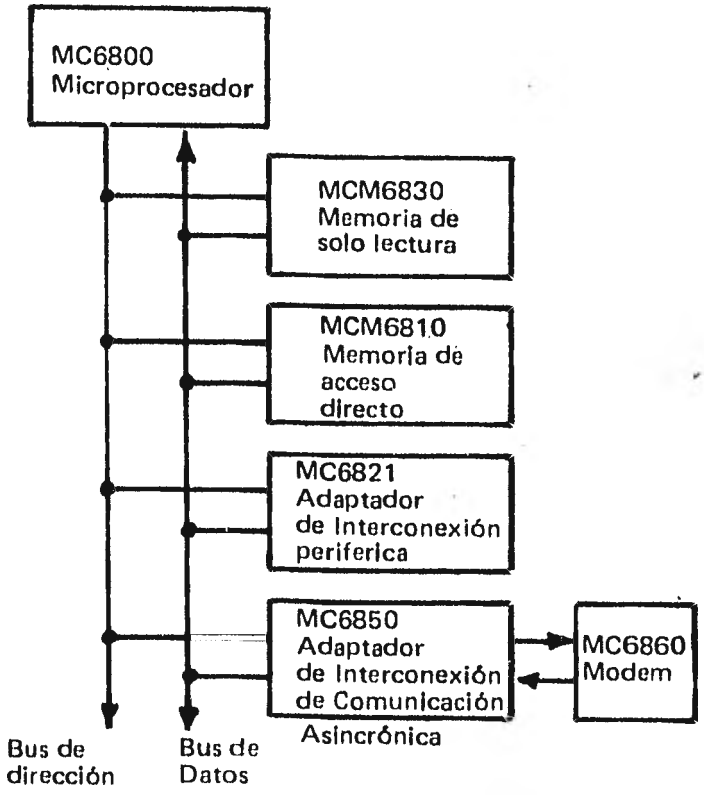


FIG. 1.5 SISTEMA PARA UNA MICROCOMPUTADORA M6800

do se deben ejecutar operaciones independientes del programa.- El bit interruptor de máscara en el registro de condición de código, no tiene efecto en este interruptor.

- Interruptor de vectores.

1.4.2 En apoyo del MPU tenemos dispositivos de memoria e interconexiones de E/S.

a) El MCM6810, memoria de acceso directo (RAM) de 128 bytes, sirve para almacenar datos y para 2 interconexiones de E/S.

b) El MCM6830 memoria de solo lectura (ROM) con 1024 bytes - que se utilizan para almacenar las instrucciones y las tablas permanentes de datos.

c) El MC6821 adaptador de interconexión periférica (PIA), facilita un medio efectivo para interconectar el equipo periférico - al MPU.

d) El MC6850, adaptador de interconexión de comunicación asincrónica (ACIA). Uno de sus usos más importantes es cambiar los - datos de 8 bits en paralelo a corrientes seriadas de bits. (En una línea manda cada vez un bit).

Las operaciones de (ACIA) pueden ser reguladas a través del registro del control de programa (PC).

El MPU interconecta todos estos dispositivos a través del bus de direccionamiento de 16 bits y la transmisión de datos se efectúa sobre el bus de datos de 8 bits. Además utiliza la técnica de tratar todos los periféricos como memoria.

Un dispositivo, tras adquirir su dirección, debe saber si recibe o envía datos; la línea de lectura/escritura (R/W) es la encargada de resolver este conflicto, controlando el flujo de datos

entre los dispositivos del sistema.

El sistema funciona con una potencia de 5 volts.

1.5 Unidad de registros del microprocesador.

El MPU tiene disponibles 3 registros de 16 bits y 3 registros de 8 bits para el uso del programador.

(ver figura 1.6)

1.5.1 Acumuladores.

El MPU contiene 2 acumuladores designados como acumulador A y B que se usan para retener temporalmente los resultados de las operaciones ejecutadas por la unidad lógica y aritmética (ALU). Cada uno es de 8 bits.

1.5.2 Registro índice (X).

El registro índice, contiene 16 bits, se usa principalmente para modificar direcciones.

El contenido del registro puede ser:

a) cargado desde memoria,

b) almacenado en memoria,

c) decrementarse,

d) incrementarse,

e) compararse con el contenido de una localidad en memoria.

1.5.3 Contador del Programa (PC).

Es un registro de 16 bits; contiene la dirección del siguiente byte de la instrucción que se va a traer desde la memoria.

Cuando el valor actual del PC se coloca en el bus de direccionamiento el PC se incrementará automáticamente.

1.5.4 Apuntador del stack o pila (SP).

Es un registro de 16 bits, contiene la dirección de la siguiente localidad disponible en una pila.

Esta pila es un acceso al azar de escritura/lectura para la me

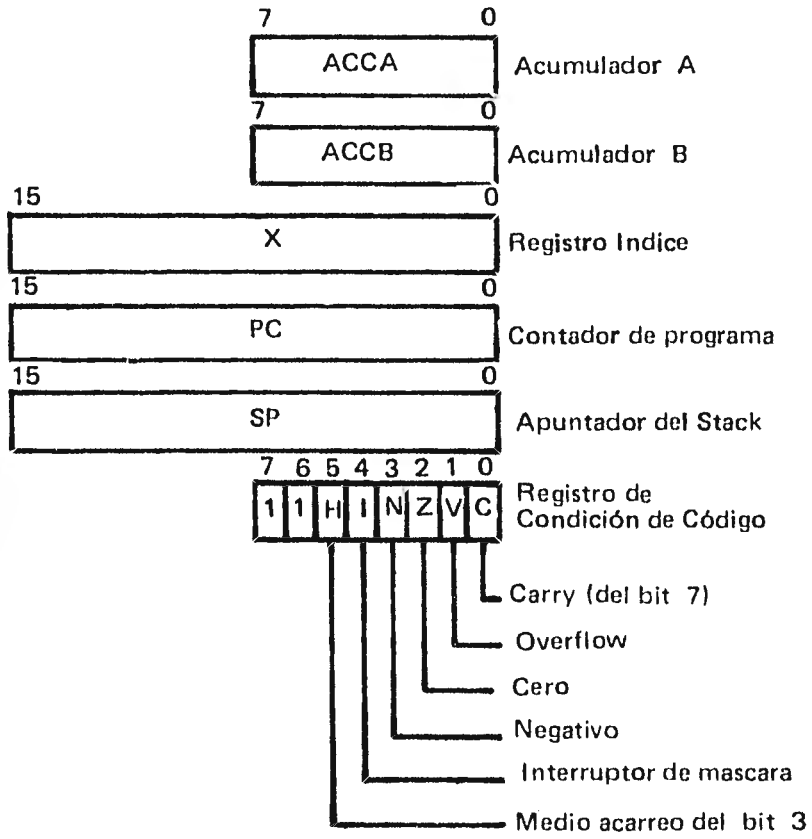


FIG. 1.6 REGISTROS DEL MPU

moria y ahí se guarda cualquier dirección.

1.5.5 Registro de condición de código, (CC).

Es un registro de 8 bits; el resultado de una operación efectuada por la unidad lógica y aritmética, así como algunas instrucciones, afectan a bits específicos de este registro. Los bits se denominan: en orden ascendente:

0) Carry - acarreo del bit 7 - (C)

1) Overflow - exceso - (V)

2) Cero (Z)

3) Negativo (N)

4) Interruptor de máscara (I)

5) Medio - acarreo del bit 3 - (H)

Estos bits generalmente se usan para efectuar instrucciones - de salto, las cuales ocurrirán de acuerdo al status (posición relativa) de los bits específicos del registro.

Los bits 6 y 7 no se usan y siempre están fijos con un " 1 ".

CAPITULO II

LENGUAJE PARA MOTOROLA M6800

El propósito del presente capítulo es el de discutir el lenguaje de máquina y el lenguaje ensamblador.

Debido a que el macroensamblador está diseñado para la microcomputadora M6800 el tercer inciso del capítulo es un manual para programar en el lenguaje ensamblador de esta computadora.

Si se conoce el lenguaje ensamblador de alguna otra máquina no será necesario profundizar en el lenguaje ensamblador para la micro M6800, bastará remitirse al apéndice B.

2.0 Programa.

Un programa es una lista de instrucciones que sigue una computadora para llegar a la solución de un problema.

El programador, para elaborar esta lista, generalmente se basa en un algoritmo o en un diagrama de flujo.

Después de cargar un programa en la máquina, las instrucciones quedarán almacenadas en la memoria de la computadora, específicamente en el ROM.

La computadora para llevar a cabo un programa, toma de memoria, una instrucción y ejecuta una operación, así una y otra vez hasta alcanzar el fin del programa.

2.1 Lenguaje de máquina.

Las instrucciones en forma binaria son las únicas que entiende la computadora. Toda computadora posee un conjunto de instrucciones en código binario que sólo ella comprende. Para cada instrucción la computadora tiene circuitos que obedecen la operación dictada por la instrucción.

Halstead (15) designa como código de máquina la preparación directa del código número de la computadora escribiendolo en la representación absoluta octal, hexadecimal, decimal o binaria de la máquina involucrada. Para él este es el lenguaje de máquina.

Quien haya utilizado el lenguaje de máquina conoce cuán laborioso es. Las mayores desventajas son:

- Las instrucciones formadas por números, ya sea en forma binaria, octal, o hexadecimal, son difíciles de recordar e interpretar.
- Es fácil equivocarse cuando se escriben números: especialmente si son cadenas de ceros y unos.
- Las instrucciones de transferencia (branch) son especialmente difíciles de codificar.
- Revisar un programa es complicado.
- Generalmente, cuando ocurren errores que requieren borrar o insertar líneas, habrá que revisar nuevamente el programa y corregir otras instrucciones.
- El programador debe tener un conocimiento exacto de todas las localidades de memoria usadas y para qué propósito las guarda.

2.2 Lenguaje ensamblador.

Para simplificar el proceso de escribir, eliminar errores y - leer el programa, se inventó el código simbólico o mnemónico, que es proporcionado por los fabricantes de las computadoras: consiste en tres o cuatro letras que describen la función de cada ins-- trucción. Como los códigos se escogen para ser fácilmente recordados, se denominan mnemónicos.

Cada mnemónico tiene asociado su equivalente en código de máquina, escrito en hexadecimal u octal. Para cada instrucción se - tendrá uno y sólo un código mnemónico.

El programador puede escribir su programa usando códigos simbólicos; sin embargo hay que recordar que la computadora sólo entiende el lenguaje de máquina. El programador, entonces, debe -- transformar el código mnemónico a lenguaje de máquina, usando para ello tablas con el conjunto de instrucciones. No obstante este procedimiento implicaría efectuar varias operaciones de cálculo. Ya que la transformación y los cálculos son operaciones mecánicas, se pueden efectuar en una computadora utilizando un programa ---- designado con el nombre de ensamblador.

El lenguaje que emplea el ensamblador lo designamos como lenguaje ensamblador; es característico de cada computadora y sigue un conjunto de reglas de sintaxis.

2.2.1 Ventajas y desventajas del lenguaje ensamblador.

El lenguaje ensamblador es esencialmente equivalente al len-- guaje máquina, con las siguientes excepciones:

- El código de las instrucciones o código de operación se escribe mnemónicamente.

- Las direcciones de las localidades de memoria no necesitan - especificarse numéricamente, sino que pueden escribirse simbólicamente, dando más flexibilidad al programador pues no - son absolutas.
- Es más fácil introducir datos en el programa.
- El usuario puede estipular constantes en forma binaria, octal, decimal, o hexadecimal.
- Se pueden entremezclar comentarios en el programa.
- La lectura del programa es más fácil.
- Los lenguajes ensambladores no solamente poseen mnemónicos - para los códigos de instrucción; también tienen instrucciones que definen operaciones para el ensamblador. Este tipo - de instrucciones las conocemos con el nombre de pseudo-instrucciones o directivas.

  Algunas directivas no generan código de máquina.

La desventaja del lenguaje ensamblador con respecto al de máquina es que se requiere un ensamblador.

Los lenguajes de alto nivel son adecuados para muchos problemas, facilitan al usuario la escritura de programas complejos, sin trabas, sin entretenerse en detalles de operaciones rudimentarias. Comparados con los lenguajes ensambladores son fáciles de escribir, comprender y de eliminar errores.

Los lenguajes de alto nivel no se diseñan para una computadora particular por lo tanto, rara vez pueden tomar ventaja suficiente de la organización de una computadora (14). En consecuencia tales programas no son siempre muy eficientes.

Los lenguajes ensambladores generán menos código de máquina -

que los lenguajes de alto nivel, en consecuencia se ejecutan en menos tiempo y ayudan a optimizar el espacio de memoria.

Se recomienda usar el lenguaje ensamblador en las siguientes - situaciones:

- Si se desea tener un control directo de las localidades de - memoria o de los registros internos de la máquina.
- Cuando la velocidad es de primera importancia (como en mu-- chos programas de servicio).
- Cuando se desea trabajar a un nivel muy detallado, donde cada instrucción representa una única instrucción primitiva.
- Cuando la computadora no está equipada con compiladores para lenguajes de alto nivel o no pueda soportarlos.

El principal uso de los lenguajes ensamblados se encuentra en la escritura de programas para el sistema operativo y para siste-- mas similares del software.

2.3 Lenguaje ensamblador para Motorola 6800.

Todo programa escrito en lenguaje ensamblador para la Motorola 6800 consiste en una secuencia de sentencias fuentes.

Una sentencia fuente incluye de uno a cuatro campos.

Los cuatro campos son de izquierda a derecha:
(1) etiqueta, (2) operador (mnemónico), (3) operando, (4) comentario.

Cada sentencia debe tener al menos el campo del operador.

2.3.1 Conjunto de caracteres.

Los siguientes caracteres son los que reconoce el ensamblador:

1. Las letras de la A a la Z.
2. Los enteros del 0 al 9.
3. Cuatro operaciones aritméticas:

   + - * /

4. Caracteres usados como prefijos especiales:

   # (signo de gato) indica el modo de direccionamiento inmediato.

   $ (signo de pesos) indica un número hexadecimal.

   @ (arroba) indica un número octal.

   % (porcentaje) indica un número binario.

   ! (signo de admiración) indica un número decimal.

   ' (apóstrofe) indica un caracter literal ASCII.

5. Caracteres usados como sufijos especiales:

   B (letra B) indica un número binario.

   H (letra H) indica un número hexadecimal.

   O (letra O) indica un número octal.

   Q (letra Q) indica un número octal.

6. Cuatro caracteres de separación.

   ESPACIO

   TAB horizontal

   CR (retorno del carro).

   , (coma).

   El uso del TAB horizontal es opcional y puede ser reemplazado por ESPACIO.

7. Un comentario puede incluir cualquier caracter ASCII.

8. Además, el ensamblador tiene la capacidad de leer cadenas de caracteres y de asignar los 7 bits correspondientes del código ASCII a localidades de memoria.

   Esto se realiza con la directiva FCC.

   Este proceso también se puede efectuar para un único caracter ASCII usando el modo de direccionamiento inmediato.

2.3.2 Campo de etiqueta.

El campo de etiqueta es el primer campo de una línea fuente. Este campo puede tomar una de las siguientes formas:

(1) Un asterisco (*) como primer caracter indica que el resto de la línea fuente es un comentario.

(2) Un blanco (b) o TAB como primer caracter indica que el campo de etiqueta está vacio (la línea no es un comentario y no tiene etiqueta).

(3) Un símbolo.

Los atributos del símbolo son:

- Consiste de 1 a 6 caracteres.
- Los caracteres válidos de un símbolo son las letras A a la Z y los dígitos 0 a 9.

. El primer caracter de un símbolo debe ser alfabético.

. Un símbolo debe aparecer solo una vez en el campo de - etiqueta.

A una etiqueta (un símbolo en el campo de etiqueta) generalmente se le asigna el valor del contador del programa (PC) en esa línea. En general, las etiquetas pueden corresponder a localidades de memoria o a valores numéricos.

Las etiquetas se usan en una sentencia en los siguientes casos:

. Cuando se requiere que una línea sea el destino de una instrucción de salto (jump) o de desplazamiento (branch). - Las instrucciones de jump y branch tendrán en el campo del operando un símbolo que deberá ser idéntico a la etiqueta del destino.

. Siempre que se use la directiva EQU, a la etiqueta se le asignará el valor de la expresión del campo del operando.

. En cualquier instrucción que produzca un código de máquina (instrucción ejecutable). Las directivas FCC, FCB, FDB, y -- RMB pueden llevar etiquetas, las demás directivas no admiten etiquetas.

2.3.3 Campo del operador.

Este campo ocurre directamente después del campo de etiqueta y consta de 3 o 4 caracteres.

Se clasifica en dos grupos:

. Instrucciones ejecutables. Son códigos mnemonicos representativos de las instrucciones que puede ejecutar la máquina --- 6800.

. Directivas del ensamblador o pseudo-instrucciones. Son códi-

gos de operación especial, conocidas sólo por el ensamblador, que controlan el proceso de ensamblaje antes de ser traducidas directamente al lenguaje de máquina.

2.3.4 Campo del operando.

La interpretación del campo del operando depende del campo del operador.

Para las instrucciones de la máquina M6800, el campo del operando debe especificar el modo de direccionamiento.

Los formatos del operando y sus correspondientes modos de direccionamiento son los siguientes:

Formato del operando	Modo de direccionamiento
Ningún operando	Inherente y acumulador
Expresión	Directo y extendido
# Expresión	Inmediato
Expresión ,X	Indexado

Para las directivas el campo del operando toma otras formas -- que se verán más adelante.

2.3.4.1 Expresiones.

Una expresión es una combinación de los símbolos y/o números - separados por uno de los operadores aritméticos (+, -,* , o /).

El ensamblador evalúa las expresiones algebraicas de izquierda a derecha sin usar grupos de paréntesis. No hay jerarquía de prioridad entre los operadores aritméticos. Si al evaluar, una expresión se obtiene un resultado fraccional se le reducirá a un valor entero.

2.3.4.2 Números o constantes.

Se pueden escribir de las maneras siguientes:

Decimal	! <número>	o	<número>
Hexadecimal	$ <número>	o	<número> H

(El primer dígito en el último caso debe ser 0 - 9)

Octal	<número>	o	<número> O o <número> Q
Binario	% <número>	o	<número> B

Donde número es elemento de los enteros positivos.

2.3.4.3 Literales ASCII.

Su formato es:

' caracter (Apóstrofe seguido de un caracter ASCII).

El resultado es el valor numérico del caracter ASCII.

2.3.4.4 Símbolos.

Un símbolo en una expresión es similar a un símbolo en el campo de etiqueta excepto porque en lugar de definirlo, sólo se le -- menciona.

El ensamblador reconoce como símbolo especial al asterisco --- " * " y representa el valor del contador del programa (PC). A cada símbolo se le asocia un valor entero de 16 bits. Este valor se usa en lugar del símbolo durante la evaluación de la expresión.

2.3.5 Comentario.

El último campo de una línea fuente es el campo de comentario. Es un campo opcional y puede incluir cualquier caracter ASCII.

El ensamblador ignora este campo excepto para listarlo. El campo de comentario está separado del campo del operando (o del campo del operador si no existe operando) por uno o más blancos. Este campo es un auxiliar en la documentación del programa.

2.3.6 Instrucciones.

La máquina M6800 tiene 72 instrucciones ejecutables; sin embargo, reconoce y actúa con 197 de las 256 posibilidades que pueden ocurrir al usar una palabra de 8 bits de longitud. Este variado número de instrucciones se produce porque muchas de las instrucciones ejecutables tienen más de un modo de direccionamiento.

El ensamblado de una instrucción produce de 1 a 3 bytes de código de máquina.

Existen varios tipos de instrucción, las dirigidas a:

1) Operaciones para los registros A y B.

2) Saltos (JUMP) y control de transferencia (BRANCH).

3) Controlar el registro índice (X) y el apuntador del stack.

4) Controlar el registro de condición de código.

La lista completa de instrucciones se puede consultar en el apéndice A.

2.3.7 Modos de direccionamiento.

El modo de direccionamiento se refiere a la manera en que el programa afecta al microprocesador MPU para obtener sus instrucciones y datos.

Los modos de direccionamiento son las formas en que el microprocesador calcula la dirección del dato, a partir del campo del operando de la instrucción (23).

A continuación se presenta una lista de los modos de direccionamiento y el número de bytes de código de máquina que genera cada instrucción dependiendo del modo.

Modo de direccionamiento	Número de bytes del código de máquina
Inherente	1
Acumulador	1
Relativo	2
Directo	2
Indexado	2
Inmediato:	
1. Todas las instrucciones excepto CPX, LDS y LDX	2
2. Las instrucciones CPX, LDS y LDX	3
Extendido	3

2.3.7.1 Modo de direccionamiento acumulador.

En este modo se asigna la dirección a través del acumulador A o el acumulador B.

Son instrucciones de un solo byte y necesitan solamente el campo del operador.

Operador	Comentario
INCA	INCREMENTA EN 1 ACUMULADOR A.
INCB	INCREMENTA EN 1 ACUMULADOR B.

(El flujo de programa para esta última instrucción se ilustra en la fig. 2.1)

2.3.7.2 Modo de direccionamiento inherente o implicado.

Para algunas instrucciones toda la información que se requiere para el direccionamiento, está contenida en el mnemónico del opera

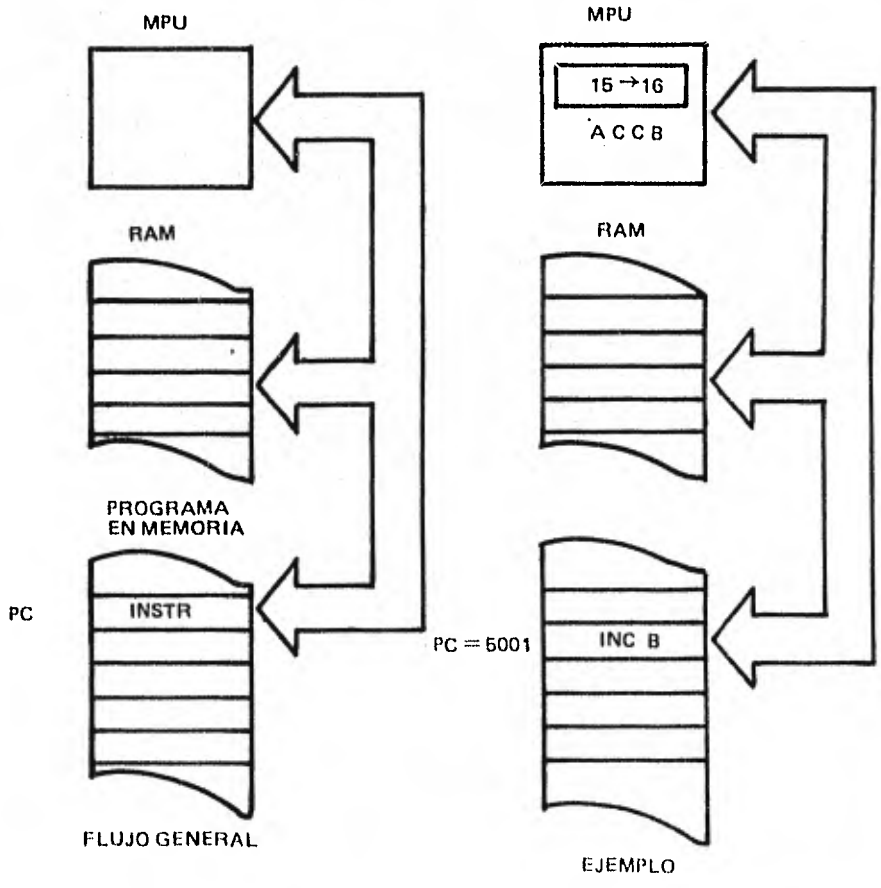


FIGURA 2.1 MODO DE DIRECCIONAMIENTO ACUMULADOR

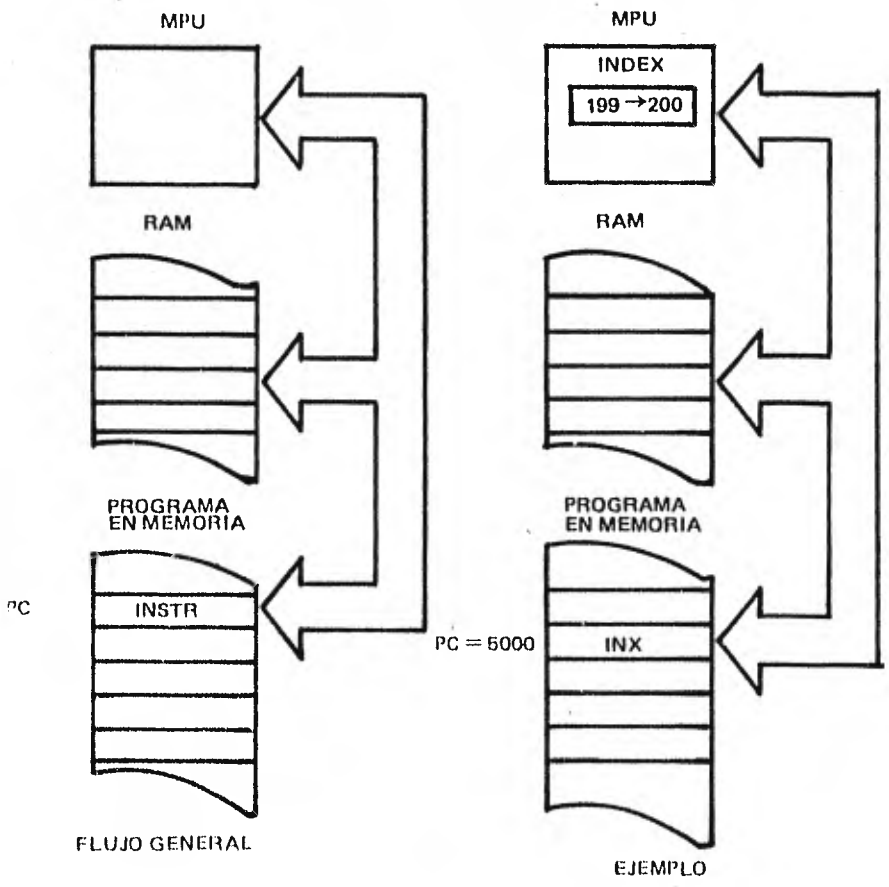


FIG. 2.2 MODO DE DIRECCIONAMIENTO INHERENTE

dor por lo que en la sentencia fuente, el campo del operando no se usa. Estas instrucciones solo requieren un byte. Existen 25 de estas instrucciones. (Como se puede ver en el apéndice A).

Ej.

Operador	Comentario
INX	Incrementa en 1 registro índice.

(El flujo del programa para esta instrucción se ilustra en la figura 2.2).

2.3.7.3 Modo de direccionamiento inmediato.

Se caracteriza por tener el signo #, precediendo al campo del operando. La expresión que sigue a este signo requiere uno o dos - bytes, dependiendo de la instrucción.

Este modo simplifica el uso de constantes y permite al usuario incluirlas en la lista del programa.

El operando puede escribirse con uno de los siguientes formatos:

Número

Símbolo

Expresión

'C

El último formato instruye al ensamblador a traducir el caracter siguiente del apóstrofe, en sus correspondientes 7 bits del código ASCII.

Ej. de este modo:

Operador	Operando	Comentario
LDAA	#25	CARGA 25 EN ACUMULADOR A

(El flujo del programa para este modo se ilustra en la fig. 2.3.)

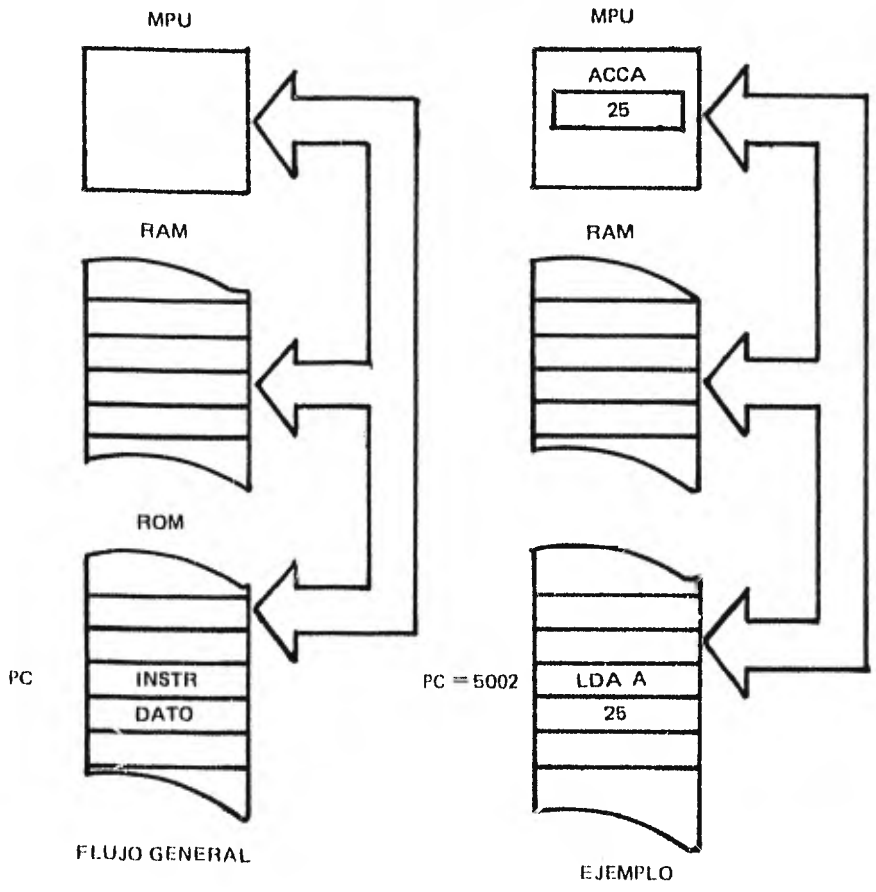


FIG. 2.3 MODO DE DIRECCIONAMIENTO INMEDIATO

2.3.7.4 Modo de direccionamiento indexado.

El modo indexado de direccionamiento emplea 2 bytes.

La localidad posterior de la instrucción contiene un número (conocido como desplazamiento u offset) que se suman al contenido del registro índice para formar una dirección que contendrá el dato.

Podemos expresar lo anterior con la siguiente fórmula:

D = Valor númerico + X

donde X = contenido del registro índice

D = dirección numérica

En este caso la dirección numérica es variable y no es predeterminada por el ensamblador como en los otros modos de direccionamiento.

El campo del operando - en la sentencia fuente - puede contener un símbolo o una expresión que el ensamblador deberá remplazar por el valor que será sumado al contenido del registro índice.

El offset debe tomar valores del intervalo 0 - 255.

Este modo de direccionamiento se caracteriza por tener después del campo del operando los símbolos " ,X ". Existen casos especiales en que no aparece el operando y solo encontramos " ,X " o simplemente X , estos dos casos significan lo mismo que 0,X.

El formato será de este modo:

X

,X

Símbolo ,X

Expresión ,X

Ej.

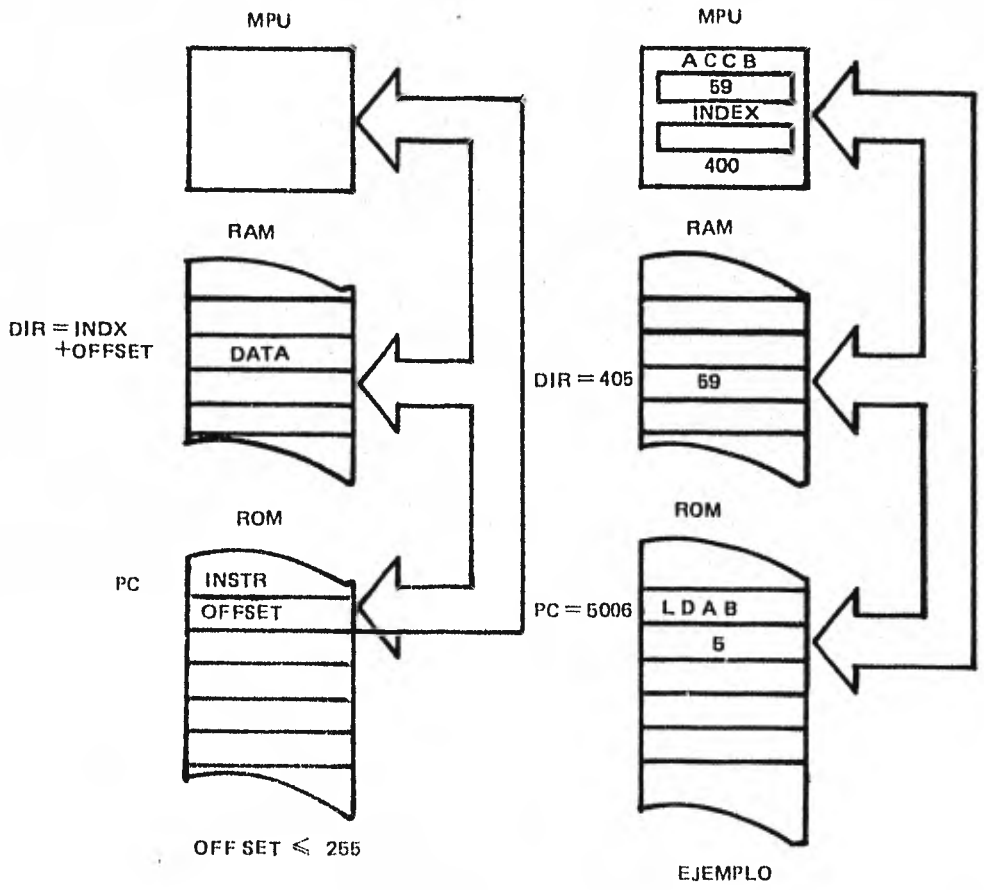


FIG. 2.4 MODO DE DIRECCIONAMIENTO INDEXADO

Operador	Operando	Comentario
LDAB	5 ,X	COLOCA EN B EL CONTENIDO DE UNA DIRECCION DADA POR (X) + 5

(El flujo del programa para esta instrucción se ilustra en la figura 2.4)

2.3.7.5 Modo de direccionamiento relativo.

Las instrucciones de transferencia permiten al usuario dirigir el MPU de un punto del programa a otro.

Para lograr esta transferencia, el usuario depende del estado en que se encuentre el registro de condición de código.

El direccionamiento relativo se emplea en la ejecución de las instrucciones de transferencia. El ensamblador traduce estas instrucciones en 2 bytes de código de máquina, uno para el código de operación de la instrucción y otro para la dirección relativa.

Para poder efectuar transferencias a cualquier dirección el byte de dirección se interpreta de la manera siguiente: el octavo bit se emplea como bit de signo y los restantes 7 bits contienen el valor numérico de la dirección (en complemento a dos en forma binaria).

Los valores que el modo relativo puede tomar son los localizados entre -127 y 127, con respecto a la localidad de la misma instrucción de transferencia. Sin embargo el valor de transferencia se evalúa con respecto a la siguiente instrucción que se ejecutaría si las condiciones de transferencia no se satisfacen. Ya que se generan 2 bytes, la siguiente instrucción es calculada como PC + 2.

Sea

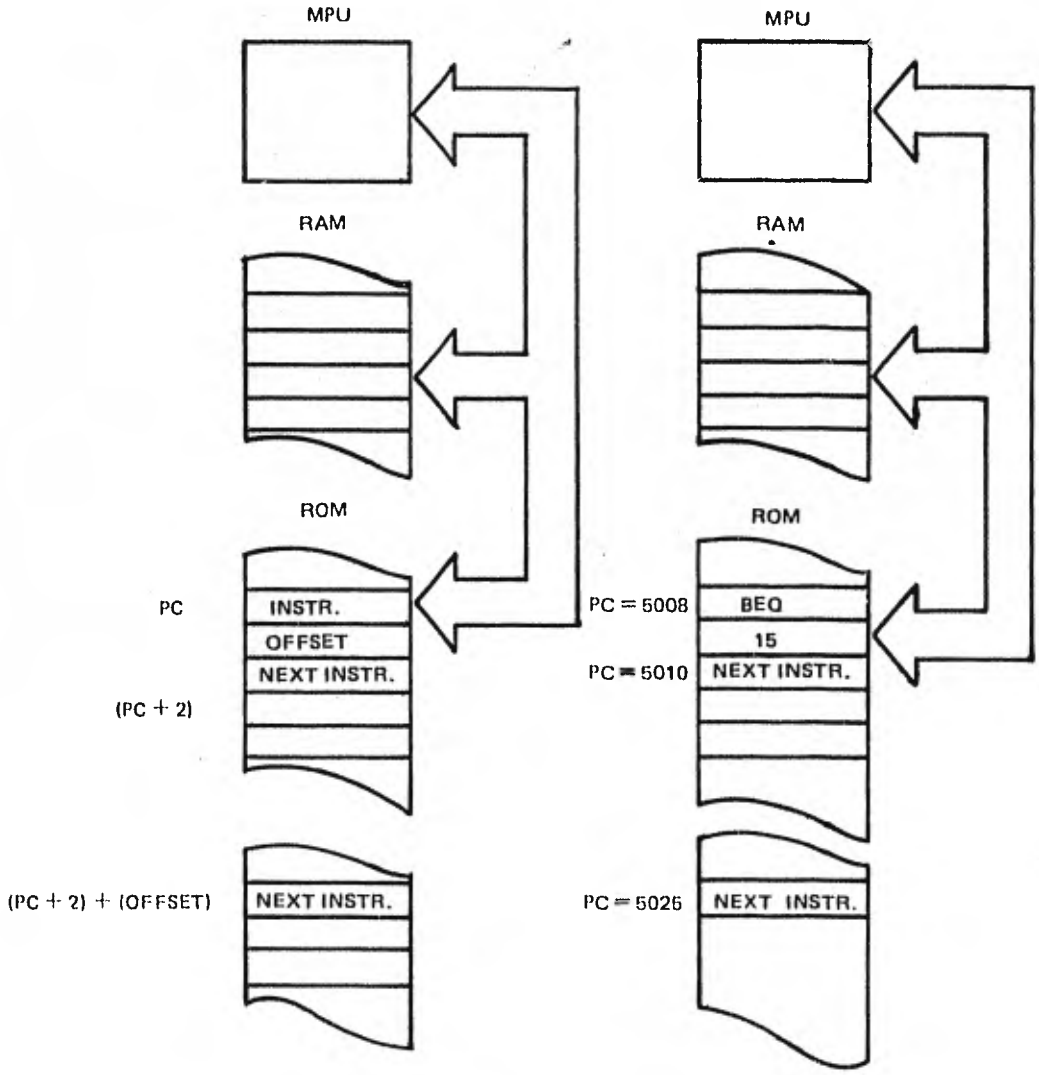


FIG. 2.5 MODO DE DIRECCIONAMIENTO RELATIVO

D = dirección del destino de la instrucción de transferencia.

PC = dirección del primer byte de la instrucción de transferencia.

Entonces

$$(PC + 2) - 127 \leq D \leq (PC + 2) + 127$$

o

$$PC - 125 \leq D \leq PC + 129$$

Por lo tanto el destino de una instrucción de transferencia debe estár entre - 125 a 129 localidades de memoria con respecto a - la misma instrucción de transferencia.

(Ver la figura 2.5)

2.3.7.6 Modos de direccionamiento directo y extendido.

La instrucción fuente, se transforma en 2 bytes de código de - máquina para el modo directo. El segundo byte contiene la direc--- ción en donde el dato puede encontrarse (dirección numérica absoluta). Este modo de direccionamiento se limita a los primeros 256 bytes de la memoria (0 a 255).

Para el modo extendido, la instrucción fuente se transforma en 3 bytes de código de máquina. El segundo de estos bytes contiene - los 8 bits más altos de la dirección y el tercero los 8 bits más - bajos.

Existen algunas instrucciones en las que se permite el modo extendido, pero no el modo directo.

Algunas instrucciones usan ambos modos. El ensamblador - para estas instrucciones - deberá evaluar el operando. Si el valor de - la dirección se encuentra entre 0 y 255 (decimal) se tiene el modo directo; si excede estos valores se tiene el modo extendido.

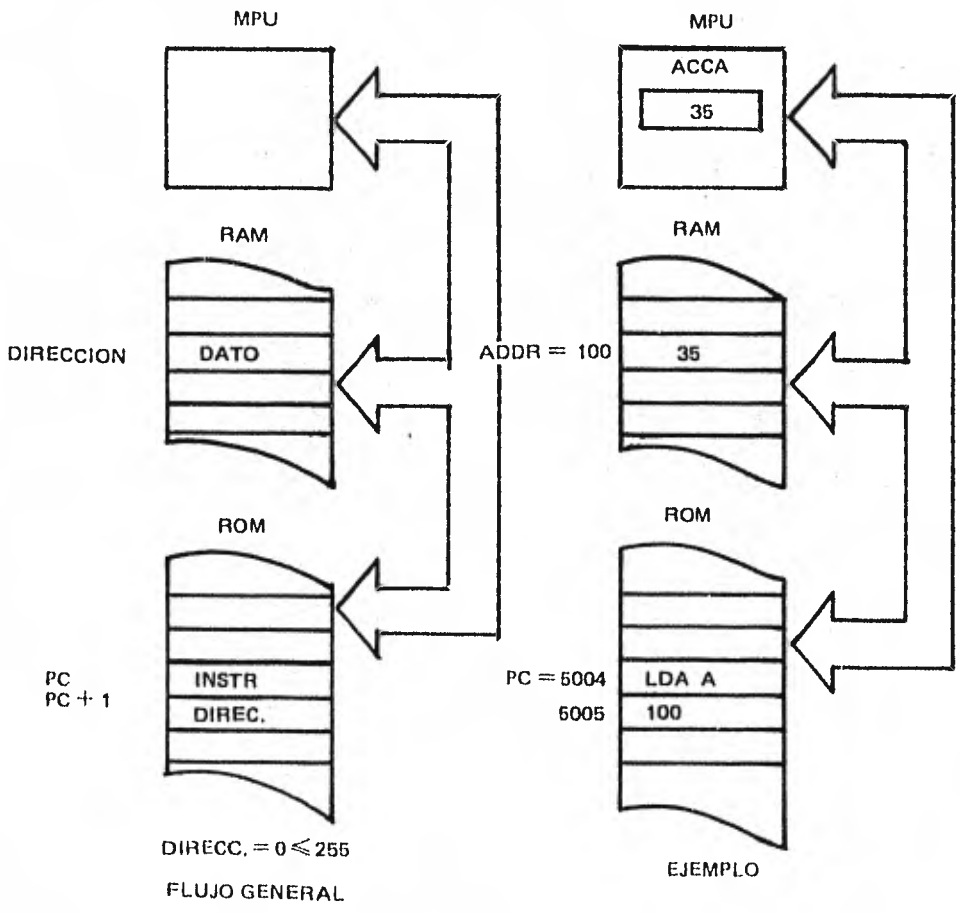


FIG. 2.6 MODO DIRECTO DE DIRECCIONAMIENTO

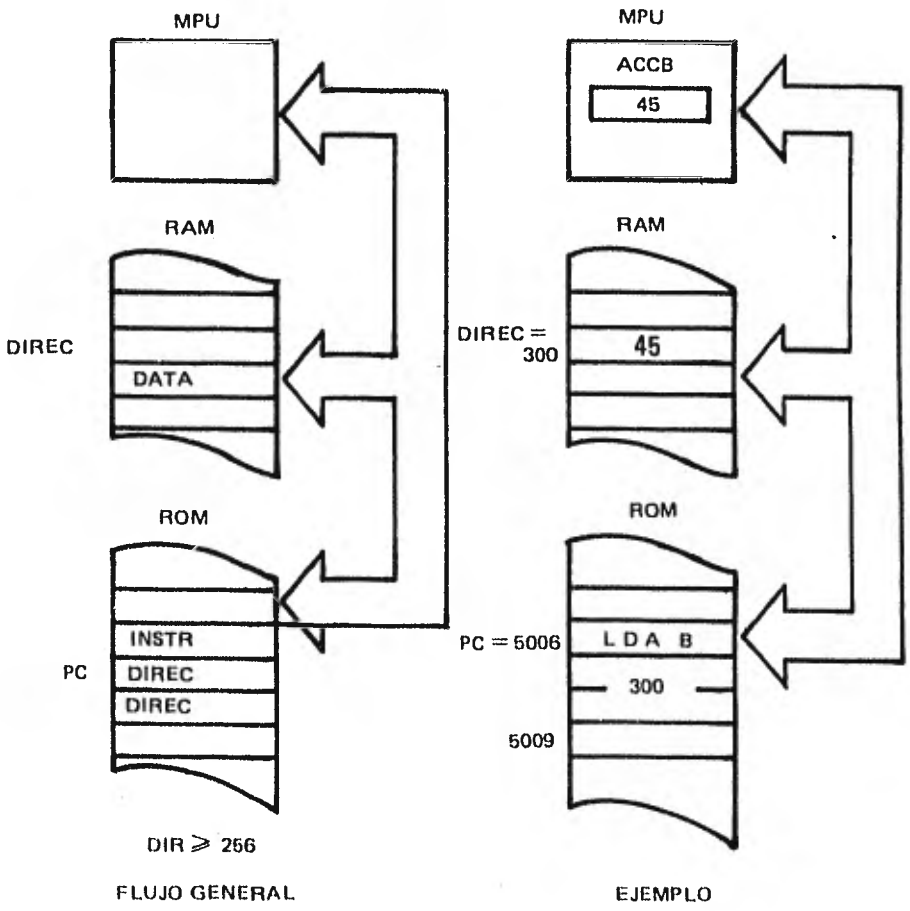


FIG. 2.7 MODO DE DIRECCIONAMIENTO EXTENDIDO

El formato del operando puede ser:

Número

Símbolo o

Expresión

(Para el modo directo ver figura 2.6 y para el extendido figura 2.7)

Se recomienda usar de preferencia el direccionamiento directo. Si en este modo se hace referencia a una variable, que no ha sido definida aun en el programa fuente, resultara un error de fase (- phasing). Para evitar este problema las variables que se empleen en el modo directo deberán ser siempre definidas antes de cual---- quier referencia a esta variable.

2.3.8 Pseudo-instrucciones.

Las directivas o pseudo-instrucciones son instrucciones para - el ensamblador más bien que instrucciones para ser transformadas - en código objeto.

Una lista y clasificación de las directivas se presenta a continuación:

DIRECTIVA	FUNCION
CONTROL DEL ENSAMBLADO	
NAM	Nombre del programa.
ORG	Origen.
END	Fin del programa.

DIRECTIVA	FUNCION
DEFINICION DE DATOS Y ASIGNACION DE LOCALIDADES	
FCC	Cadena de caracteres de datos.
FCB	Dato de un byte.
FDB	Dato de doble byte.
RMB	Reserva de bytes en la memoria.
DEFINICION DE SIMBOLO	
EQU	Asigna un valor permanente.

2.3.8.1 NAM - Nombre del programa.

FORMATO: NAM <nombre del programa> [<comentario>]

DESCRIPCION:

La directiva NAM debe ser la primera sentencia del programa. La directiva no permite una etiqueta pero requiere un operando de 1 a 8 caracteres.

2.3.8.2 ORG - Origen

FORMATO: ORG <expresión> [<comentario>]

DESCRIPCION:

La directiva de ORG cambia el valor del contador de programa - (PC) por el valor especificado en la expresión de su campo de operación.

A las sentencias subsiguientes se les asigna localidades de memoria que comienzan con el nuevo valor del contador de programa.

La directiva no admite etiqueta.

2.3.8.3 END - Fin del programa.

FORMATO: END

DESCRIPCION:

La directiva END indica al ensamblador que el programa termino. Si hay más líneas el ensamblador las ignora.

3.8.4 FCC - Forma constante de carácter.

FORMATO: [<etiqueta>] FCC

{ d <cadena ASCII> d
  <número decimal> , <cadena ASCII> }

[<comentario>] .

NOTA:

1. " d " es cualquier carácter no número
   (usado como delimitador)
2. La cadena ASCII no debe incluir un retorno de carro.

DESCRIPCION:

La directiva transforma cadenas de caracteres en sus correspondientes codigos ASCII de 7 bits.

Cualquiera de los caracteres que correspondan a los códigos - ASCII hexadecimal pueden ser procesados por esta directiva.

Existen realmente 2 formatos:

1. Contador, coma, texto. Donde el contador indica cuántos caracteres ASCII se van a generar. El texto empieza siguiendo la primera coma del operando. Cuando el contador sea mayor que el número de caracteres ASCII encontrados después de la coma, se introducirán espacios hasta alcanzar al contador. El número máximo del contador será de 255.
2. El texto encerrado entre delimitadores idénticos. Los delimitadores pueden ser cualquier carácter. Si los delimitadores no son números, el texto no debe empezar con una coma.

2.3.8.5 FCB - Forma constante de byte.

FORMATO: [<etiqueta>] FCB

{ { <expr> , [<expr> ,] 00 [<expr>]
{ <nulo> , [,] 0

<expr>

[<comentario>]

DESCRIPCION:

La directiva asigna valores de constantes a localidades consecutivas de memoria.

La directiva puede tener uno o más operandos separados por comas. Si hay más de un operando entonces se almacenarán en bytes sucesivos.

Una directiva FCB con uno o más operandos nulos separados por comas, almacenará ceros para esos operandos.

Un número binario sin signo de 8 bits se asigna como valor a cada operando.

2.3.8.6 FDB - Doble forma constante de byte.

FORMATO: [<etiqueta>] FDB

{ { <expr> , [<expr> ,] 00 [<expr>]
{ <nulo> , [,] 0

<expr>

[<comentario>]

DESCRIPCION:

Esta directiva trabaja igual que la directiva FCB excepto porque un número binario sin signo de 16 bits se asigna como valor para cada operando.

2.3.8.7 RMB - Reserva bytes de memoria.

FORMATO: [< etiqueta >] RMB <expresión > [<comentario>]

DESCRIPCION:

A contador de programa (PC) la directiva RMB lo incrementa con el valor del campo del operando. Así, se reserva un bloque de memoria con longitud igual al valor de la expresión.

La expresión no debe contener símbolos que sean definidos posteriormente en el programa.

2.3.8.8 EQU

FORMATO: < etiqueta > EQU <expresión > [<comentario>]

DESCRIPCION:

La directiva EQU asigna a la etiqueta el valor de la expresión que aparece en el campo del operando.

La expresión en el campo del operando no puede incluír un símbolo indefinido o que no haya sido anteriormente definido.

La expresión después de evaluarse debe ser positiva.

ENSAMBLADOR Y MACROS

CAPITULO III

Al ser el ensamblador un programa tiene que cumplir ciertos objetivos. En este capítulo se presentan cuales son esos objetivos, la manera en que se pueden llevar a cabo, los métodos y técnicas que existen para implementarlo y los problemas a los que se enfrentará quien lo diseñe.

También se introduce al lector en el funcionamiento de las macroinstrucciónes (macros) y el programa encargado de manejarlas: el macroprocesador. Después de leer este punto se verá lo que es un macroensamblador.

Debido a la importancia que tienen las macros y como Motorola no las define para la M6800 se define la sintaxis de las macros que se implementarón para esta microcomputadora.

En todo el capítulo se ha procurado tratar los conceptos en una forma general sin enfocarlos a una computadora específica.

3.1 Definición de ensamblador.

Un ensamblador es un programa cuya entrada es el conjunto de caracteres que forman un programa en lenguaje ensamblador y cuya salida es un conjunto de instrucciones de máquina.

El ensamblador por ser un programa traductor forma parte del sistema operativo de una computadora.

P. Wegner (38) sugiere que un ensamblador puede representarse por medio de una función.

El dominio de la función sería el conjunto de programas escritos en lenguaje simbólico o mnemónico y cuyo rango sería el conjunto de programas en lenguaje máquina.

E: <lenguaje simbólico> ⟶ <lenguaje de máquina>

La acción de un ensamblador E sobre un programa simbólico S produce un programa en lenguaje máquina M tal que:

$$M = E (S)$$

En un ensamblador se da generalmente una correspondencia uno a uno, entre las sentencias escritas en lenguaje ensamblador y las

sentencias escritas en el lenguaje máquina. Sin embargo, esto no a contece con algunas pseudo-instrucciones y especialmente con las - macros - que más adelante se explicarán-.

Cuando un ensamblador permite macros se le designa como macro-ensamblador.

El ensamblador como cualquier otro programa puede encontrarse almacenado en la memoria de la máquina o en una memoria secundaria, en este caso recibe el nombre de ensamblador corresidente.

Cuando los sistemas son pequeños, como en el caso de las micro, se pueden presentar dificultades para acomodar el programa ensam-- blador, tales como: insuficiente memoria y soporte.

La solución para este problema es almacenar el ensamblador en una computadora más grande y realizar en ella las operaciones de - programación. El programa generado en el lenguaje de máquina por - la computadora más grande deberá copiarse en la más pequeña.

La principal desventaja de este sistema es la dependencia, aun que esta disminuye cuando el propietario de cada una de las máqui- nas es la misma persona. Por su manera de actuar el ensamblador se llama en este caso ensamblador cruzado.

Para las micro y las minicomputadoras, el ensamblador es un -- sistema que facilita la interconección entre el usuario y el hard- ware de la máquina ya que permite escribir instrucciones que refle jan la estructura interna de la máquina.

3.2 Funciones del ensamblador.

Son funciones del ensamblador:

i) Proporcionar una forma simple de comunicación entre el programador y la máquina. Esto se logra al usar el lenguaje ensamblador. Esta es la función primordial del ensamblador, para ello se tiene que:

A) Generar instrucciones. Hay que evaluar cada uno de los campos de una sentencia y producir su correspondiente código de máquina.

B) Procesar pseudo-instrucciones.

ii) Facilitar la tarea de análisis del programa. Por ello se proporciona una lista del programa en el lenguaje ensamblador junto con su respectivo código de máquina. En el formato de salida deberán existir columnas especiales para el código de máquina y para cada uno de los campos que forman una sentencia.

iii) Detectar los errores de tipo sintáctico. El ensamblador deberá proporcionar una lista de los errores sintácticos detectados en un programa. En algunos ensambladores se acostumbra designar a los errores con un único caracter alfabético relacionado con el error, sin embargo existen otros ensambladores en que se les designa con un número entero. La lista de errores dependerá de la sintaxis del lenguaje ensamblador empleado. Un ejemplo de una lista de errores en que se puede incurrir es la siguiente:

A.- Error en el modo de direccionamiento.

D.- Símbolo doblemente definido.

E.- No existe End.

I.- Carácter ilegal.

O.- Código de operación desconocido.

U.- Símbolo indefinido.

T.- Transferencia fuera de rango.

iv) Realizar automáticamente ciertas tareas: establecer localidades en donde se almacenan los datos, manejar direcciones simbólicas.

v) Proporcionar al cargador cierta información como: referencias externas, símbolos relocalizables, direcciones absolutas, etc.

Se puede proporcionar una lista de símbolos. El ensamblador puede además permitir macros y directivas condicionales de ensamble. Estas últimas directivas facilitan al programador incluir o excluir segmentos de código de máquina, dependiendo de ciertas condiciones.

3.3 Proceso de ensamble y sus estructuras.

El proceso de traducir el código mnemónico al lenguaje de máquina es el que se designa como proceso de ensamble.

3.3.1 Tablas de símbolos.

Como la mayoría del trabajo del ensamblador consiste en convertir representaciones simbólicas a formas binarias, esta conversión se basa en una o más tablas de símbolos.

Una tabla de símbolos es esencialmente una secuencia de vectores (k_i, t_{1i},....., t_{ni}) con $n \geq 1$ - que en conjunto forman una matriz - con una única primera componente k_i llamada llave, donde k_i = código fuente y t_i es el valor o valores asociados a k_i.

Podemos definir una tabla de símbolos como una función cuyo dominio es el conjunto S de todas las llaves k_i y cuyo rango es el conjunto T de las t_i.

Hay que considerar varios problemas con respecto a la construcción de una tabla.

i) El mecanismo de como organizar la tabla para consultarla, dependerá de la manera de buscar cada llave (k_i).

ii) Como estructurar cada vector de la tabla.

La estructura dependerá del lenguaje ensamblador y aquí solo se considerarán los aspectos fundamentales.

La única regla que se da es que la estructura de la tabla debe contener toda la información que se requiera en un mínimo de espacio.

Lo primero que se deberá hacer al implementar un ensamblador es diseñar la estructura de las tablas que se van a utilizar. A partir de esta estructura es cuando realmente empezará el trabajo

de programación del ensamblador.

2.3.2 Proceso.

Se recuerda que una sentencia en lenguaje ensamblador generalmente está compuesta por los campos de:

(1) etiqueta (2) operador (3) operando y (4) comentario

El último campo es ignorado por el ensamblador, solo se debe imprimir en el listado.

Cualquier persona que desee entender el proceso de ensamble debe pensar que el programa ensamblador debe actuar simulando en muchos procesos, que es realmente una máquina.

Para cada sentencia el ensamblador debe:

1. Efectuar una búsqueda del grupo de caracteres que forman un campo.

   La manera para buscar los campos dependerá de que el lenguaje ensamblador tenga un formato fijo o un formato libre.

   En un formato fijo cada uno de los campos debe ocupar columnas específicas. Un ejemplo de este formato sería el que asignara al campo de etiqueta las columnas 1 a 6, al campo del operador las columnas 8 a 11 y las columnas 13 a 32 al operando.

   Para aislar cada campo, los formatos libres presentan mayores dificultades que los formatos fijos.

2. Transformar los campos a su correspondiente código de máquina.

   Para ello se necesita tener una tabla con la equivalencia entre el código de máquina y el código mnemónico, además de una técnica para su búsqueda.

   Si en el operador aparece una pseudo-instrucción, se deberá tener una tabla de pseudo-instrucciones.

Para el campo del operando deberá haber un procedimiento para evaluar símbolos, números y expresiones.

3. Llevar cuenta de las instrucciones y donde residen en memoria.

Para efectuar este proceso el ensamblador lleva un contador -- llamado: contador de localidades de programa (PLC).

El PLC equivaldría al registro PC que posee la máquina y por - lo tanto el PLC debe actuar para el ensamblador como actúa el PC para la máquina.

La función del PLC es señalar donde se va a alojar en memoria la siguiente sentencia.

El PLC se va incrementando a medida que se van ocupando las localidades de memoria y esto depende de la longitud de la ins-- trucción o de los modos de direccionamiento.

Despúes de que la sentencia es traducida al código de máquina, el PLC se actualiza para reflejar el hecho de que la localidad de ese momento ya se utilizó.

El ensamblador al principio del programa asigna al PLC el va-- lor cero, aunque el lenguaje ensamblador puede proporcionar alguna pseudo-instrucción para alterar el PLC en cualquier parte del programa.

Cuando termina el trabajo del ensamblador se tiene por fin el programa objeto, que deberá ser cargado a la memoria de la máquina.

Si el PLC se inicializa en cero, se deberán ocupar las localidades de memoria a partir de la dirección cero, pero esto muchas veces no es posible y habrá que relocalizar el programa, esto es, cambiar la dirección base donde se carga el programa,

en otras palabras alterar el PLC.

4. Llevar cuenta de los símbolos usados y de sus valores.

Un símbolo se define al aparecer en el campo de etiqueta de una sentencia, pero en el campo del operador puede aparecer una instrucción o una pseudo-instrucción, en el primer caso se seguirá llamando símbolo, en el segundo caso se llamará literal para señalar que la etiqueta almacena un dato.

Se recuerda que los símbolos son direcciones simbólicas.

El programador no conoce en que localidad de memoria quedará - almacenada una sentencia, y si quiere referirse a ella deberá hacerlo como se designa a un lugar conocido pero del que no se sabe su dirección, por ejemplo: Cine Latino.

Sin embargo el ensamblador proporcionará una tabla que actuará como un directorio y nos facilitará la dirección del Cine Latino.

El ensamblador constuye una tabla de símbolos donde se almacenan los símbolos que aparecen en el campo de etiqueta junto -- con el valor de PLC que tiene la instrucción en ese momento.

Otro de los datos que contendrá la tabla de símbolos, será que el símbolo es relocalizable (o relativo) por depender del valor del PLC.

Cuando existen múltiples definiciones del PLC en el transcurso de un programa,el símbolo deberá indicar a cual PLC es relativo.

Existen dos tipos de direcciones: direcciones relocalizables y direcciones absolutas. En su forma más rudimentaria las direcciones absolutas son número o carácteres que aparecen en el -- campo del operando de una sentencia.

Para las literales se construye una tabla de literales donde - se almacena la etiqueta, el valor del campo del operando y se indica si es absoluta o relativa. Este último caso ocurre cuando en el campo del operando aparece un símbolo definido ante-- riormente como relocalizable.

3.3.3 Base de datos del ensamblador.

Queda claro que la base de datos estará formada por:

1.- Un programa simbolico proporcionado por el usuario.

2.- El contador de localidades del programa (PLC).

3.- Una tabla de mnemónicos.

Cada vector contendrá:

a) Un mnemónico.

b) El código de máquina.

c) La longitud de la instrucción ó
una clave que señale los modos de direccionamiento,permitidos.

Esto permite transformar una instrucción a su respectivo código de máquina.

4.- Una tabla de pseudo-instrucciónes.

Como las pseudo-instrucciones son instrucciones que debe - ejecutar el ensamblador cada vector contendrá:

a) El mnemónico de la pseudo-instrucción.

b) El tipo de pseudo-instrucción.

c) Una clave que diga el procedimiento que se ejecutará, para procesar la pseudo-instrucción. La clave puede ser una etiqueta o una dirección para efectuar un salto y comenzar un procedimiento.

5.- Una tabla de símbolos.

Cada vector contendrá:

a) La etiqueta.

b) El valor del PLC.

c) Una clave que indique que es relocalizable.

6.- Una tabla de literales.

Cada vector contendrá:

a) La etiqueta.

b) El valor del operando.

c) Una clave que indique si es relocalizable o absoluta.

Las tablas se pueden combinar en una o varias tablas.

Por ejemplo la tabla de símbolos y la de literales son muy semejantes debido a su origen y se podrían combinar en una sola que se denominará tabla de símbolos.

Las tablas de mnemónicos y de pseudo-instrucciones reciben el nombre de tablas fijas por estar preestablecidas, es decir nunca - se insertan nuevos vectores, son tablas solo de consulta.

Las tablas de símbolos y de literales se denominan tablas li-- bres y se permite tanto la inserción como la consulta.

3.4 Ensamblador de dos pasos.

Los ensambladores de dos pasos examinan dos veces el programa simbólico antes de producir el código de máquina. En cada paso se lee el programa simbólico línea por línea.

La causa principal es la siguiente: el programador puede usar símbolos o literales en el campo del operando, antes de definirlos en el campo de etiqueta, sin embargo el ensamblador no puede referirse a ellos si no conoce la localidad en que quedan almacenados o el valor que almacenan.

El objetivo de cada uno de los pasos es el siguiente:

Primer paso.- Definir símbolos y literales.

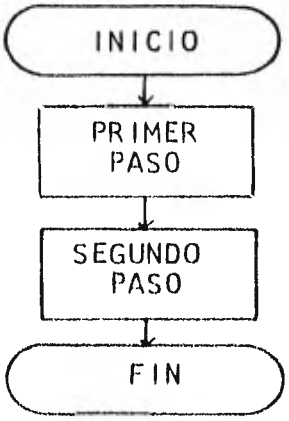
El ensamblador busca las etiquetas y las coloca en la tabla de símbolos junto con su correspondiente dirección de memoria o el valor que almacenan en caso de las literales.

Segundo paso.- Generar el código de máquina.

El ensamblador utilizando la tabla de símbolos y la tabla de mnemónicos genera el código de máquina que corresponde a cada instrucción.

El ensamblador como todo programa bien estructurado debe implementarse por módulos y comenzar a diseñarse de arriba hacia abajo.

El primer diseño será elemental:



3.4.1 Primer paso.

El diagrama de flujo del primer paso se muestra en la figura - 3.1. Es un diagrama bastante simple pero a partir del cual se puede ir profundizando en el diseño del ensamblador.

Para llevar a cabo el procedimiento " ACTUALIZAR PLC ", es necesario examinar el campo del operando para determinar que modo de direccionamiento se tiene y de acuerdo a él, incrementar el PLC.

Hay lenguajes ensambladores que tienen una longitud fija para cada instrucción, en ese caso solo habrá de obtenerse la longitud de la instrucción de la tabla de mnemónicos e incrementar el PLC.

A continuación aclararemos el PROCEDIMIENTO ESPECIAL PARA --- PSEUDO-INSTRUCCIONES.

i) Si se encontró pseudo-instrucción del tipo EQU, evaluar el operando y colocarlo junto con la etiqueta en la tabla de - símbolos indicando si es relocalizable o absoluto.

ii) Si se encontró una pseudo-instrucción que reserva bloques - de memoria, se evalúa el operando y se modifica el PLC.
PLC ⟵ PLC + N donde N es el valor del operando. Si existe etiqueta colocarla en tabla antes de incrementar el PLC.

iii) Para pseudo-instrucciones que almacenan datos será necesario contar cuantos datos son y multiplicarlos por el número de localidades que ocupa cada operando.
PLC ⟵ PLC + N * L donde L = número que ocupa cada localidad
N = número de operandos.

iv) Si la pseudo-instrucción altera el PLC, evaluar el operando y asignarle el nuevo valor del PLC.

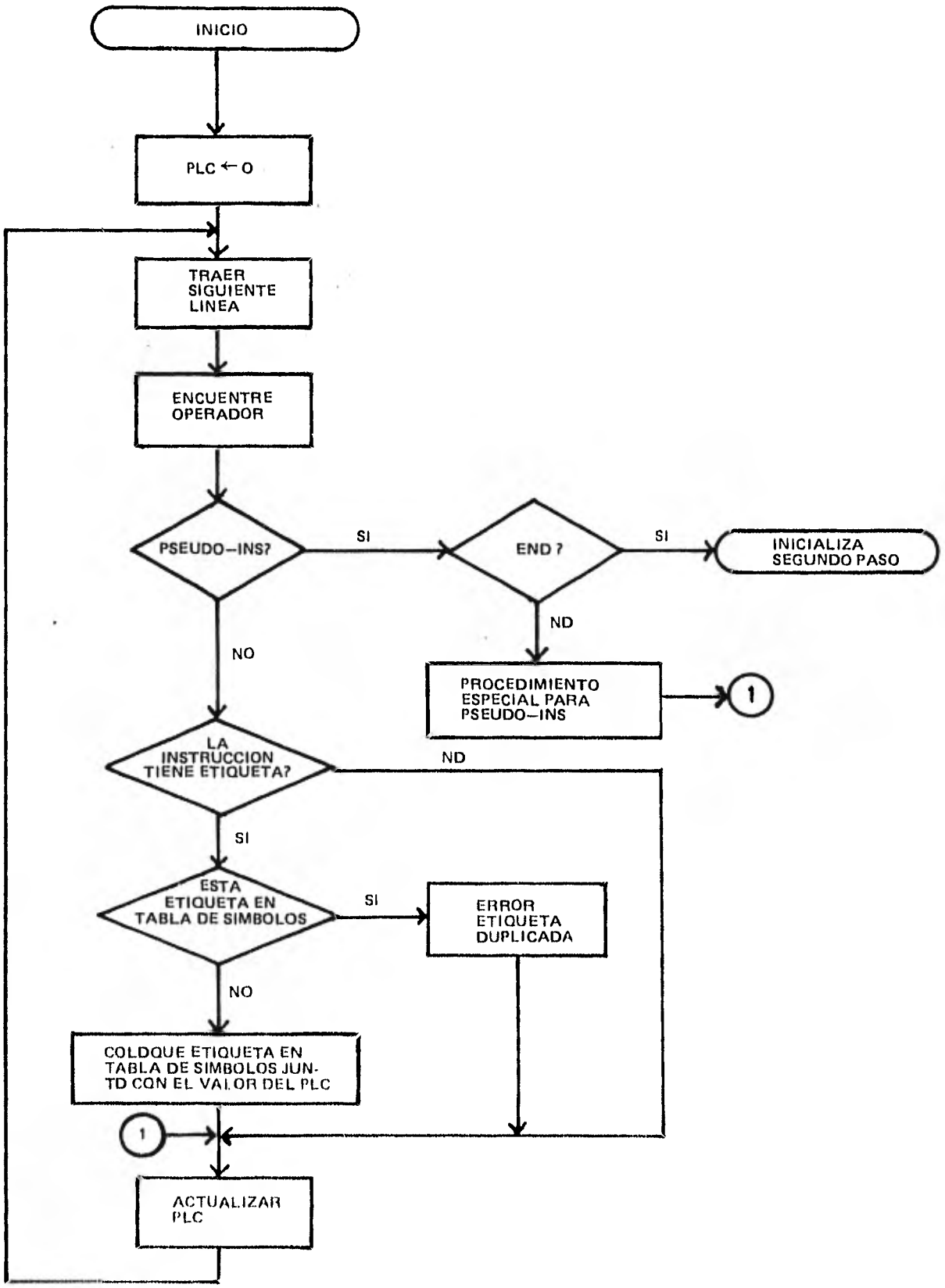


FIG. 3.1 PRIMER PASO DEL ENSAMBLADOR

PLC ⟵ N N = valor del operando.

Tanto para las instrucciones como para las pseudo-instrucciones debe quedar claro que actualizar el PLC dependerá del campo del operando.

Cuando se hace referencia a evaluar el operando aunque parezca un procedimiento simple, puede convertirse en uno de los más laboriosos del ensamblador. Todo el problema se reduce a evaluar expresiones, ya que éstas implican símbolos, literales y constantes. Antes de evaluar se deberá revisar la sintaxis.

Muchos ensambladores permiten referirse a localidades de memoria por medio de expresiones algebraicas. Generalmente las operaciones clásicas permitidas son: +, -, * y /. La división es entera, si hay residuo se desecha.

Las expresiones por ser direcciones simbólicas pueden clasificarse en relocalizables y absolutas.

Ullman (36) dice que una expresión es relocalizable si la dirección simbólica puede ser remplazada por $\alpha + \Delta$ donde Δ es una constante fija que se aplica a todas las direcciones simbólicas.

Las expresiones relocalizables toman generalmente la forma:

$\alpha \pm c$ donde c = constante y

α = dirección simbólica

Demostración:

sustituyendo α por $\alpha + \Delta$ tenemos que:

$$(\alpha + \Delta) \pm c = (\alpha \pm c) + \Delta$$

y esta última expresión es relocalizable.

Ejemplo de expresión relocalizable sería: 2 * A - B

Si sumamos a: A y B - direcciones simbólicas - Δ tenemos que:

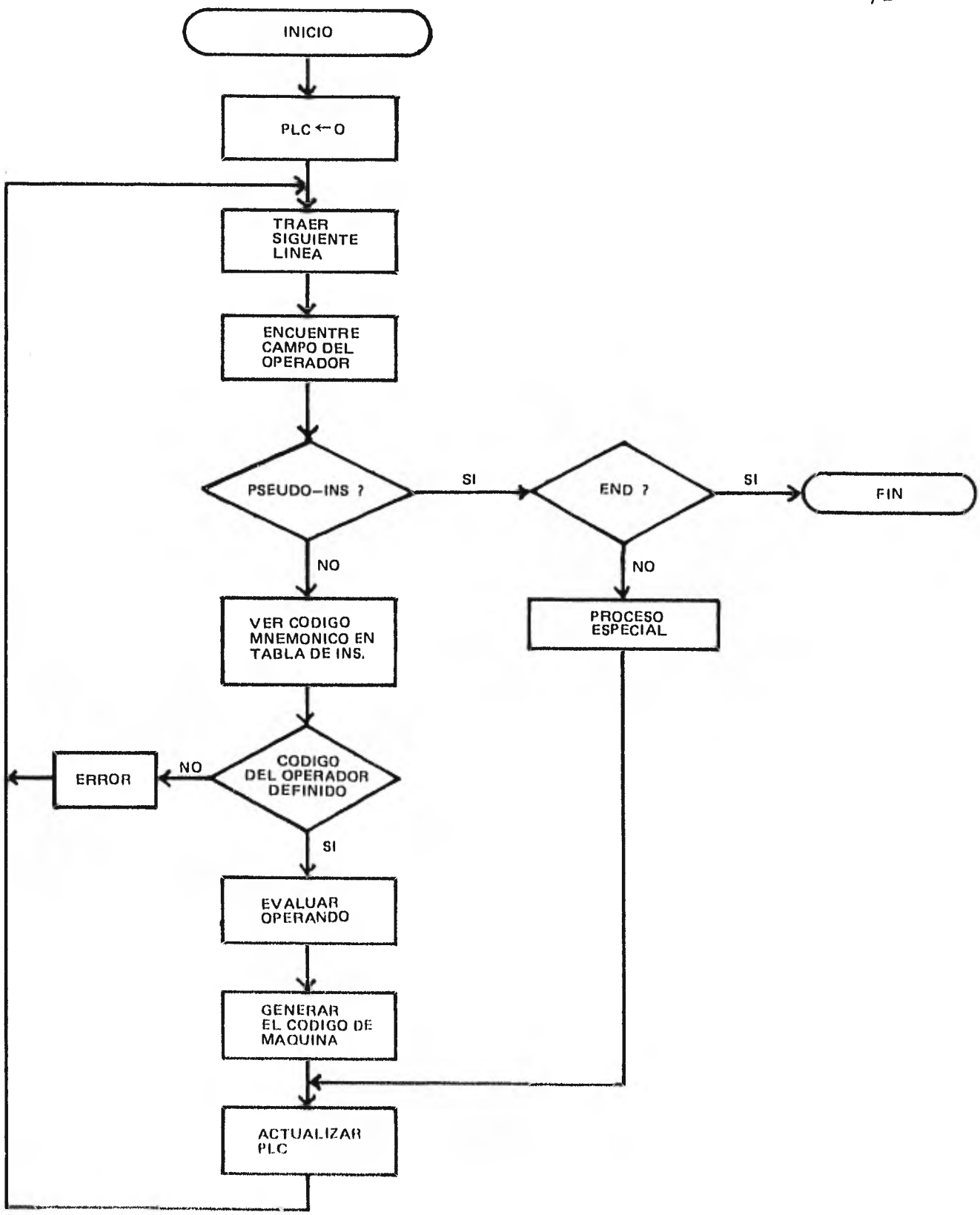


FIG. 3.2 SEGUNDO PASO DEL ENSAMBLADOR

$2 * (A + \Delta) - (B + \Delta) = 2A + 2\Delta - B - \Delta = (2*A-B) + \Delta$

Las expresiones absolutas son aquellas en que intervienen solo constantes, un símbolo con valor absoluto, la substracción entre - símbolos relativos.

Ejemplo de expresión absoluta sería A - B porque:

$(A + \Delta) - (B + \Delta) = A + \Delta - B - \Delta = A - B$

Nótese que la expresión original y la ultima son iguales.

Existen tambien expresiones que se clasifican como indefinidas por no ser ni relocalizables ni absolutas.

Ejemplo de expresión indefinida sería A * B ya que:

$(A + \Delta) * (B + \Delta) \neq A * B + \Delta$

$(A + \Delta) * (B + \Delta) \neq A * B$

Existe también un símbolo generalmente denotado por un asterisco " * " que designa al PLC. La diferencia entre este símbolo y el signo de multiplicación es que siempre aparece al principio de una expresión. Este símbolo permite a una instrucción referirse a si - misma o a instrucciones cercanas a ella, generalmente para propósitos de transferencia (BRANCH). El asterisco puede utilizarse para cualquier conveniencia del programador. Es evidente que este -- símbolo es relocalizable.

3.4.2 Segundo paso.

El diagrama del segundo paso se muestra en la figura 3.2. En - este paso no se examina el campo de etiqueta.

Para cada sentencia se examina el campo del operando para de-- terminar si tenemos una instrucción o una pseudo-instrucción. Si - se encontró una instrucción se irá a la tabla de mnemónicos y se - obtendrá el código de máquina. En las mini y las micro el código -

de máquina de cada instrucción dependerá tanto del campo del operador como del campo del operando. Hay que tener esto muy presente - cuando se diseñe la tabla de mnemónicos.

EVALUAR OPERANDO significa que en las instrucciones de transferencia (BRANCH) será el momento de calcular su desplazamiento (offset) y para aquellas que contengan expresiones efectuarlas.

Para las pseudo-instrucciones habrá que efectuar ciertas operaciones:

i) Aquellas que almacenan caracteres habrá que convertir cada caracter ASCII al código octal o hexadecimal.

ii) Las que almacenan constantes las deberán evaluar y convertir al código octal o hexadecimal.

Hay que tener presente que las pseudo-instrucciones pueden producir más de una línea.

Al finalizar este paso se imprimirá una lista de ensamble que contendrá la sentencia original y el equivalente de los bytes generados escritos en hexadecimal u octal.

Para este segundo paso se contará con rutinas para formatear - cada tipo de instrucción y de pseudo-instrucción, así que se deberá pensar de antemano cuales columnas ocupará cada campo.

Los diagramas del primer paso y del segundo dependerán siempre de las características de cada lenguaje ensamblador.

Quien diseñe un ensamblador deberá adaptarlo a su lenguaje. Habrá procedimientos que le convenga realizar desde el primer paso y deberá esperar efectuar otros hasta el segundo paso. Se recuerda - que realizar un ensamblador no es seguir una serie de reglas sino llevar a cabo un próposito: producir el lenguaje máquina.

3.5 Ensamblador de un paso.

El ensamblador de dos pasos no es el único que existe, también hay ensambladores de un solo paso o de varios pasos, sin embargo - por sus caracteristicas el ensamblador de dos pasos es el clásico.

Los ensambladores de un solo paso emplean la estrategia de --- transformar completamente a lenguaje máquina toda la sentencia si todos los símbolos que aparecen se encuentran definidos. Si uno o más símbolos están indefinidos se produce solo la parte del lenguaje de máquina que sea posible generar, pero precedida por una marca adecuada.

El segundo paso consiste en buscar las marcas producidas en el primer paso y terminar de transformar estas sentencias. Como la mayoría del trabajo se efectua en el primer paso se evitan duplica-- ciones inecesarias de análisis.

El proceso de ensamble puede verse como aquel en que se esta-- blece una unión entre objetos simbólicos (por ej: instrucciones - simbólicas) y sus valores (la versión binaria). Visto de esta - forma el ensamblador de 2 pasos es un sistema retardado de esta unión mientras que el ensamblador de " un paso y medio " hace este enlace pronto.

El ensamblador de un paso lleva cuenta de la parte incompleta de una sentencia en una tabla de fix-up (arreglo). Esta tabla -- consiste de:

- una serie de listas ligadas, una para cada símbolo que se -- usó pero no se definió.
- se almacena en la tabla de símbolos un apuntador a la cabeza de la lista en lugar del PLC.

Cada llave en la tabla de fix-up tiene asociados 4 valores:

i) un apuntador a la siguiente llave o una marca de fin.

ii) el PLC de la instrucción que falta de terminar de transformar.

iii) la dirección donde se encuentra almacenada la instrucción que falta de transformar.

iv) un código para indicar la acción requerida. Por ej:

- almacena el valor de un símbolo en un campo específico
- suma (o resta) el valor del símbolo a un campo específico
- calcula el desplazamiento (offset) y almacenalo en un campo específico.

Cuando vuelve a aparecer un símbolo indefinido hay que añadirlo a la cadena apropiada. Cuando el símbolo se encuentra por fin definido habrá que seguir la cadena para efectuar las transformaciones requeridas.

La idea de el ensamblador de un paso y las tablas de fix-up fueron idea de Wheeler que hizo este ensamblador para la computadora EDSAC 2.

3.6 Organización de tablas y mecanismos de búsqueda.

Se ha mencionado como estructurar cada vector perteneciente a una tabla pero aún falta resolver como localizar cada llave k y -- por ende sus valores asociados.

A continuación se mencionan algunos métodos.

3.6.1 Busqueda lineal.

Suponer que la tabla está organizada de una manera arbitraria entonces la estrategia más simple para encontrar un símbolo sería ir examinando cada llave de la tabla hasta que se establezca una - igualdad. Si se llega al fin de la tabla y no ocurre esa igualdad significará que la llave no está en la tabla.

Algoritmo.

1. Establezcamos que: $i \leftarrow 1$
2. $N \leftarrow$ límite de la tabla con $N > 1$
3. Es $k = k_i$ si fin.
   no $i \leftarrow i + 1$
4. $i \leq N$? si ve a 2. De otra manera k no es elemento de la ta-- bla y terminamos.

Knuth (22) resume esto de la siguiente manera: " Empezar en el - principio y seguir hasta que se encuentre la llave correcta: entonces parar ".

Si la tabla contiene N vectores y todos los símbolos ocurren - con igual frecuencia entonces habrá que examinar un promedio de -- (N + 1) / 2 llaves hasta encontrar la que se requiere. En la - práctica las llaves no ocurren con igual frecuencia algunas solo - se mencionan una sola vez en el programa simbólico - como el END - y otras, ni se mencionarán.

Este método se emplea si se quiere minimizar el código del ensamblador. Es apropiado para tablas cortas pero puede resultar lento si la tabla es extensa.

Si se quiere usar este método se sugiere colocar las llaves que se usan con mayor frecuencia al comienzo de la tabla y colocar un vector más al final de la tabla junto con la llave buscada. Si al final del algoritmo $i \leq N$ entonces encontramos la llave. Si $i = N + 1$ fracasamos. Al modificar el algoritmo de búsqueda secuencial se transforma en un método más rápido por lo que recibe el nombre de búsqueda secuencial rápida.

3.6.2 Búsqueda binaria.

Los caracteres de una llave al almacenarse en una computadora pueden interpretarse como un número entero en forma binaria y por consiguiente se puede ordenar la tabla en orden creciente.

Para localizar arbitrariamente una llave k se efectúa primero una comparación con la llave que se encuentra en medio de la tabla para determinar si la llave se encuentra en la parte inferior o superior de la tabla. El proceso se repite con la parte de la tabla en que se puede encontrar la llave, así sucesivamente hasta hallar la llave. El método también indicará si la busqueda fracasa cuando la longitud del intervalo es 1 y no es la llave buscada. Es un método eficiente para tablas extensas y requiere del orden de $\log_2 N$ comparaciones para localizar una llave, siendo N el tamaño de la tabla.

Algoritmo

1. Sea $l \leftarrow 1$, $u \leftarrow N$.
2. Es $u \prec l$ $k \notin$ Tabla
   sino $i \leftarrow \left[(l + u) / 2 \right]$ encontramos el punto medio de la tabla.
3. Si $k \prec k_i$ ve a 4;
   Si $k \succ k_i$ ve a 5;
   Si $k = k_i$ terminamos
4. Sea $u \leftarrow i - 1$ ve a 2.
5. Sea $l \leftarrow i + 1$ ve a 2.

3.6.3 Arbol de búsqueda binaria.

Si se desea insertar nuevas llaves en tablas lineales manejadas por el método de búsqueda binaria habría que añadir apuntadores a la tabla y por consiguiente no se podría encontrar la mitad de la tabla. Sin embargo esta bisección puede realizarse almacenando cada llave de la tabla con una estructura de árbol tal que cada nodo represente una llave.

Las llaves menores a una llave dada se colocan por debajo del nodo en la rama izquierda del árbol y las llaves mayores se colocan en la rama derecha.

Para insertar una llave k se hacen comparaciones con las demás llaves k_i , empezando por la raíz del árbol y recorriendo ramas izquierdas si $k \prec k_i$ y derechas si $k \succ k_i$ hasta encontrar un nodo vacío.

3.6.4 Indice alfabético.

Este método propone que cada llave sea ordenada alfabéticamen-

te. Se construirá una tabla auxiliar con cada letra del alfa- - - beto y un apuntador al principio de la tabla principal, como se -- muestra en la figura 3.3.

La tabla principal puede ser remplazada por 26 subtablas. De - pués de determinar el principio de cada subtabla se puede buscar - la llave de diferentes maneras.

El método no es conveniente para algunas tablas - como la de - símbolos - en que el tamaño de la tabla no se conoce de antemano.

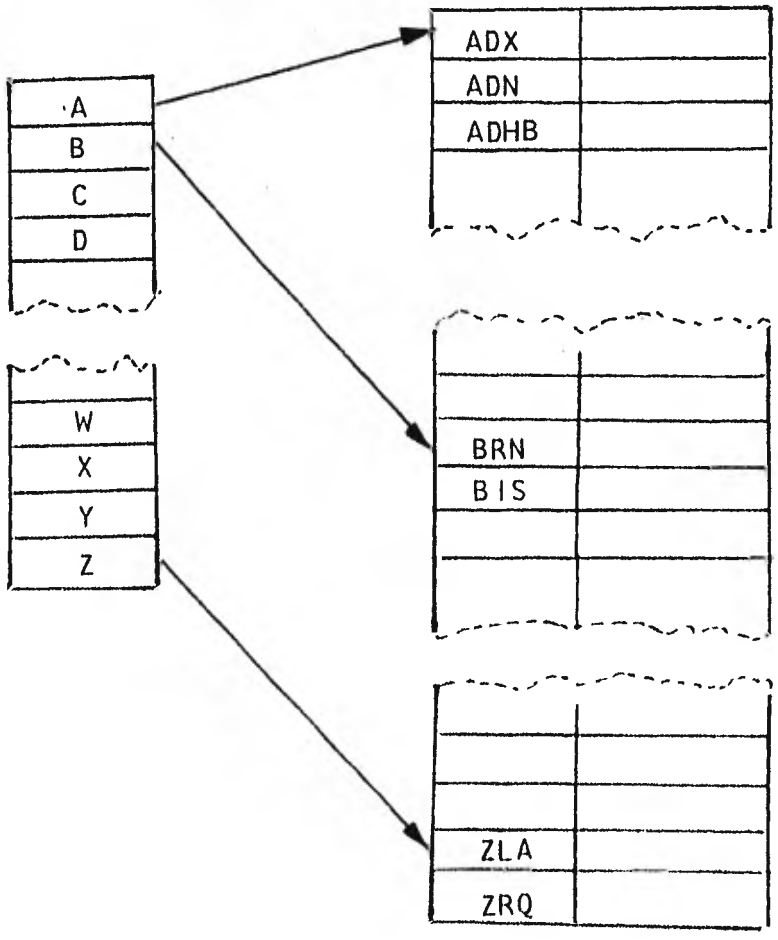


Fig. 3.3 Sección de una tabla usando índice alfabético.

3.6.5 Hash.

Los métodos discutidos anteriormente están relacionados con la manera en que una persona puede buscar una palabra en el diccionario.

Actualmente existe un método mejor para buscar una llave en una tabla grande.

Este método aprovecha la habilidad de la máquina para efectuar operaciones aritméticas o lógicas a altas velocidades.

El método de hash se basa en la idea de tratar cada letra de una llave como un número y de ahí parte para combinar esos números o desmenuzarlos hasta obtener un único número para cada llave. Ese número sera la dirección de hash de cada llave y dirá a la máquina donde localizar la llave dentro de la tabla.

Ya que más de una llave puede dar el mismo número se debe prever la posibilidad de colisión.

3.7 Conceptos básicos de macro.

Existen muchas situaciones en las cuales un programador debe repetir una serie de instrucciones en diferentes partes del programa. Esta situación se presenta con bastante frecuencia en programas escritos en lenguajes ensambladores, en consecuencia muchos de ellos proporcionán al usuario la facilidad de usar macroinstrucciones o - macros (abreviación de la palabra macroinstrucción).

Los macros no son exclusivos de los lenguajes ensambladores pueden ser proporcionados por cualquier lenguaje de programación. Sin embargo aquí solo se explicarán en el contexto de lenguajes ensam-- bladores.

En un principio las macroinstrucciones solo erán abreviaturas - de un sólo renglón que equivalían a un grupo de instrucciones. El - macroensamblador al encontrar una macroinstrucción, la substituirá por el conjunto de instrucciones que representa. Este mécanismo se puede interpretar como la inserción de texto - previamente definido - en una o varias partes del programa.

Se observa este mécanismo en la figura 3.4.

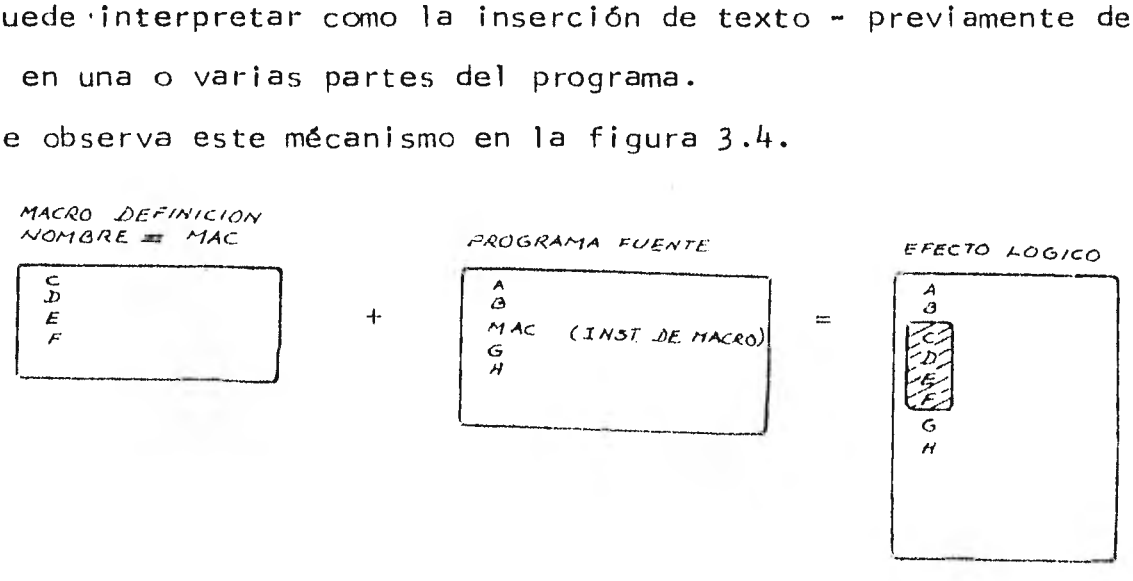


Fig. 3.4 MECANISMO ELEMENTAL DE MACRO.

Primero aparece una definición de la macro declarando que el - nombre del macro contiene una porción del texto del programa. Este nombre de macro puede ser usado en otra parte del texto del programa fuente, figura 3.4 (b). Cuando el macroensamblador reconoce - el nombre, copia el texto de la definición del macro en esa parte del programa - figura 3.4 (c) -, es decir al remplazar el nombre por una serie de instrucciones comúnes " expande " la macro.

El verdadero efecto de la macro es que mientras el programa es codificado en la forma de la figura 3.4 (b), el programa fuente que será ensamblado es realmente el que aparece en la figura 3.4 (c).

Hay otro tipo de macros: los que permiten parámetros. En este caso cuando se define a la macro se proporcionan cerca del nombre de la macro porciones de texto que se llamarán argumentos mudos - semejantes a los argumentos en una definición de subrutina -. Es-- tos argumentos permitirán posteriormente variaciones locales de la definición de macro. Ver la figura 3.5

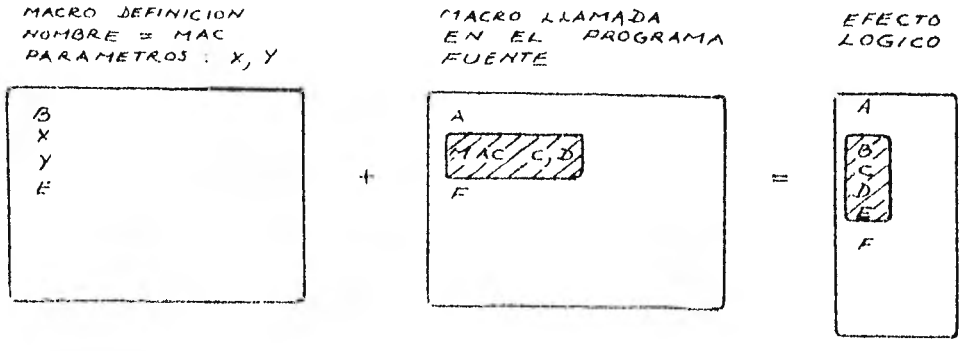


Figura 3.5 MACRO DEFINICION CON PARAMETROS.

Cuando la definición del macro es copiada en respuesta de la macro llamada, los parámetros X y Y son reemplazados por los valores C y D. Este proceso se asemeja a la definición de funciones matémati-- cas. Ver la figura 3.6.

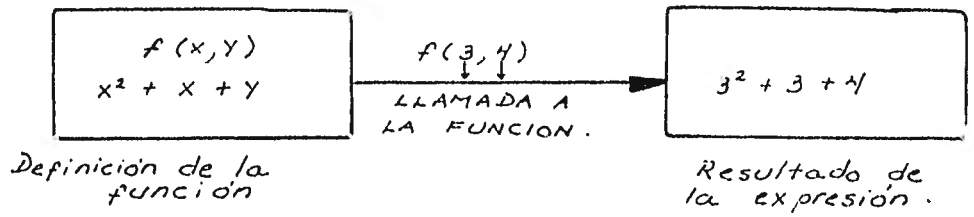


Figura 3.6 EL MACRO CON PARAMETROS SE COMPORTA COMO UNA FUNCION.

Se puede observar que la definición del macro actúa como un esqueleto o modelo al que se le pueden variar ciertas partes.

El macroensamblador puede proporcionar un ensamble condicional que permitirá la selección y reordenamiento de la definición de la macro.

Se verán algunas de estas situaciones:

i) En algunas circunstancias el programador querrá una macro - para generar una secuencia de instrucciones aritméticas y dejar el resultado ya sea en un registro, ya sea en alguna localidad de memoria.

ii) También querrá efectuar lazos (loops) alrededor de una macro para generar una secuencia de instrucciones similares a aquellas que se requieren para inicializar un número de variables con cero.

Ver la figura 3.7 para entender la acción del ensamble condicional.

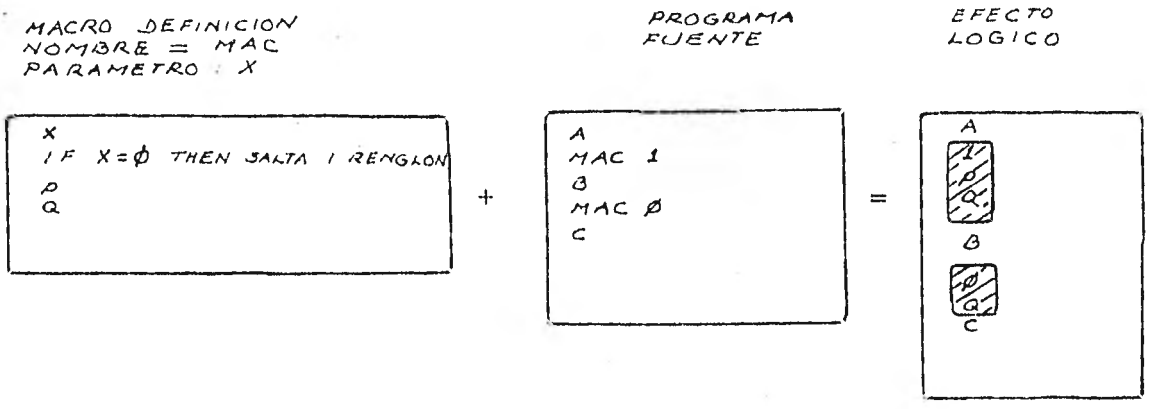
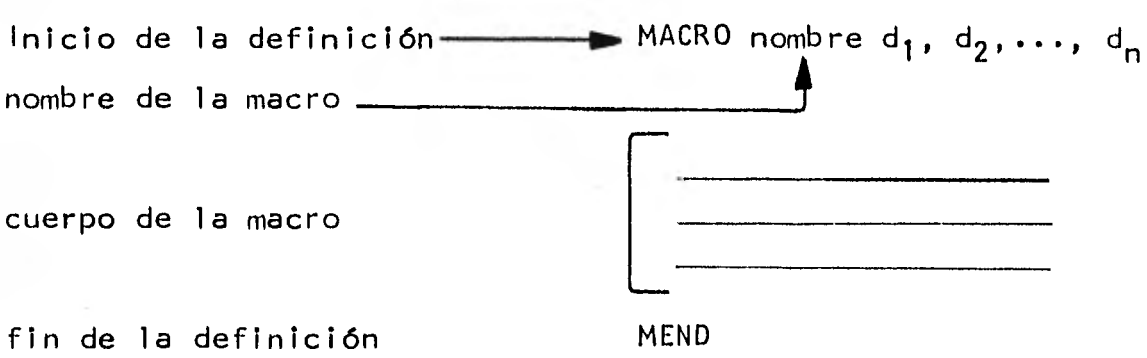


Figura 3.7 ENSAMBLE CONDICIONAL.

3.8 Definición de macro.

Después de entender el funcionamiento básico de las macroinstrucciones se darán algunas definiciones.

El formato general de una macrodefinición depende del diseñador del macro, pero básicamente estará formada de la siguiente manera:



La pseudo-instrucción MACRO inicia la línea de la definición. El nombre identifica el nombre de la macroinstrucción y se hará referencia a él como macronombre.

d_1, d_2,...., d_n se conocen como argumentos mudos. Los argumentos mudos pueden aparecer en cualquier campo de una sentencia del cuerpo de la macro. El cuerpo del macro es un conjunto de líneas.

La definición termina con la pseudo-instrucción MEND.

Se debe enfatizar que una macrodefinición no produce código de máquina simplemente es un modelo.

Un nombre de macro que aparezca como una instrucción en el programa es una macrollamada.

El formato general de una macrollamada es:

[etiqueta] nombre [a_1, a_2,...., a_n]

donde a_1, a_2,...., a_n son argumentos reales.

El proceso de reemplazar líneas e intercambiar parámetros mudos por reales cuando aparece una macrollamada lo denominamos expansión. Cada d_i será reemplazada por su correspondiente a_i. Debido a esta forma de aparearse los argumentos se las conoce como argumentos posicionales.

Una vez definida una macro puede llamarse cualquier número de veces.

Observese la figura 3.8. La macrodefinición se ensambla directamente con el programa fuente.

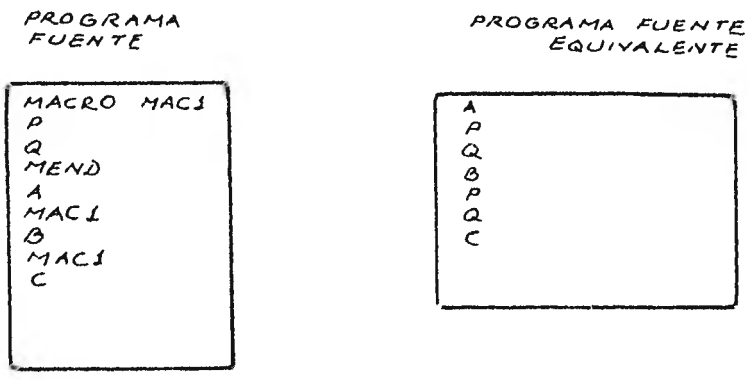


Figura 3.8 DEFINICION DE MACRO EN UN PROGRAMA FUENTE.

Algunos fabricantes proporcionan una librería de macros para ser utilizadas por los usuarios. El programador deberá recurrir a los manuales si quiere emplearlos.

3.8.1 Diferencias entre macro y subrutinas.

Observese la figura 3.9.

La subrutina cuando se define causa al compilador generar el código de máquina y cuando encuentra una llamada a esa subrutina se produce un salto a ese código.

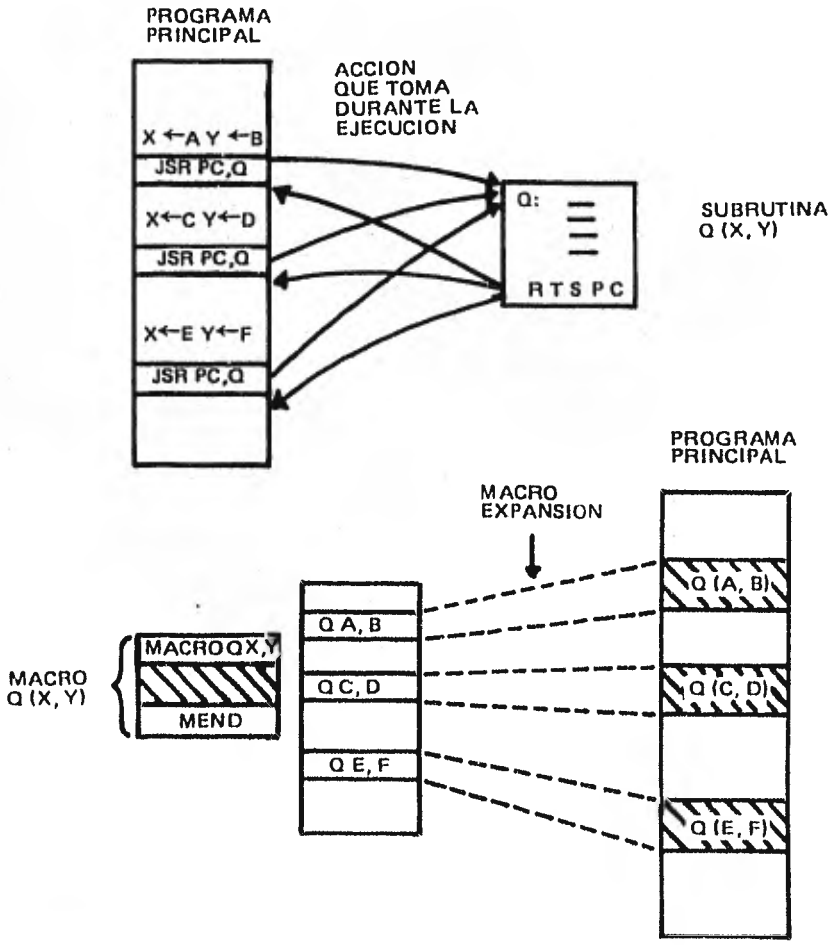


FIG. 3.9 COMPARACION DE SUBRUTINAS Y MACROS

La macrodefinición no produce código de máquina simplemente es un modelo. Cada vez que se encuentra una macrollamada se expande - el macro y el ensamblador entra en acción para producir el código de máquina. La macro por lo tanto no ahorra localidades de memoria pero ahorra tiempo al programador.

La ventaja de usar macros es que el programa se ejecuta más rápidamente ya que a diferencia de las subrutinas no se necesita efectuar saltos.

La ventaja de las subrutinas es que requieren menos espacio de memoria para el programa ya que el código de la subrutina aparece solo una vez.

Macros y subrutinas permiten al programador dividir un programa en módulos, los cuales son fáciles de entender y fácilitan estructurar un programa.

3.8.2 Beneficios dados por las macros.

El usar macros proporciona una variedad de beneficios:

- Se escribe menos código.
- Al generarse automáticamente el código se eliminan errores triviales.
- Se planea una sola vez la lógica a usar y no hay que volverla a pensar cada vez que se use.
- Si la logica es deficiente sólo habrá que reescribir en un solo lugar.
- Si se cambia de opinión sólo habrá de cambiarse la codificación en un solo lugar.
- Las macrollamadas al escribirse como mnemónicos esclarecen la lógica del programa.

- Mejor documentación en el programa y código más uniforme.
- Las macros son valiosas al producir software portable - progra-- mas que se transfieren de una máquina a otra sin necesidad de -- volverse a escribir lo cual implica efectuar menos trabajo.

3.9 Otros tipos de macros.

3.9.1 Macros interiores.

Se designan por macros interiores cuando en el cuerpo de la macrodefinición aparece una macrollamada. Ver la figura 3.10.

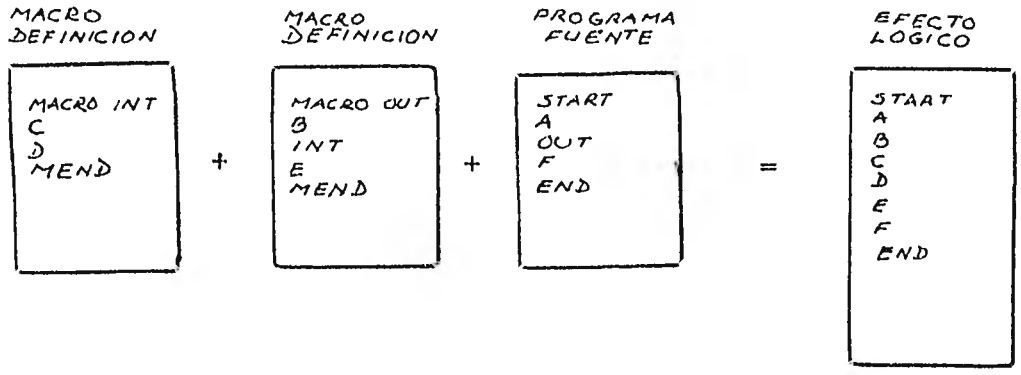


Figura 3.10 MACRO INTERIOR

3.9.2 Macros anidados.

Las macros anidados son macrodefiniciones dentro de macros por lo que algunas veces se llaman " macrodefiniciones dentro de macrodefinición ". Este tipo de macros sirve para definir una única macro que podría ser utilizada para facilitar el proceso de definir un grupo de macros similares. Ver la figura 3.11.

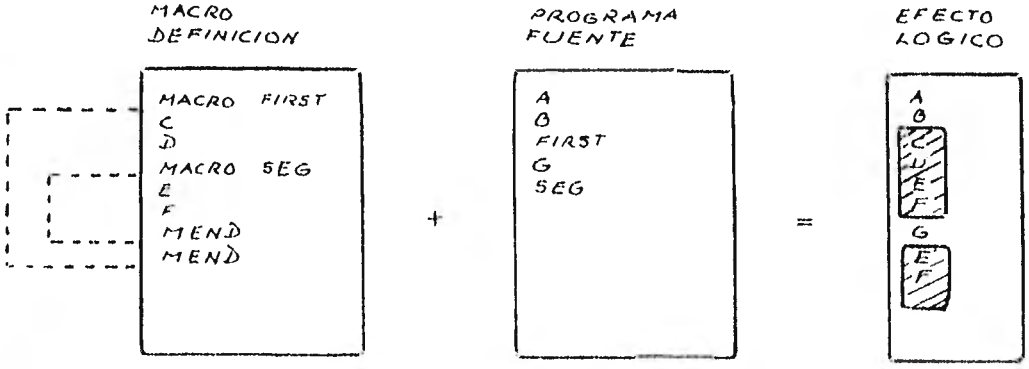


Figura 3.11 MACROS ANIDADOS.

3.9.3 Más facilidades en los macros.

Dependiendo del fabricante o de la persona que implemente el - macroensamblador, el programador encontrará diferentes facilidades que le proporcionan los macros: macros que permitán llamarse a si mismos (recursivos), aquellos que permiten concatenaciön, otros que permiten generar etiquetas locales etc.

3.10 Macroprocesador y Macroensamblador.

Un macroprocesador se puede considerar como un traductor cuya entrada es una cadena de caracteres y cuya salida es otra cadena de caracteres. Cuando se emplea un macroprocesador se traduce un lenguaje A a un lenguaje B.

El macroprocesador permite una manipulación de caracteres.

Las macroinstrucciones son extensiones del lenguaje ensamblador, sin embargo hay una clara diferencia entre las operaciones de la macroexpresión y el resto del proceso de ensamble.

El proceso de ensamble sólo puede llevarse a cabo hasta que se efectúe la macroexpansión.

El macroensamblador es la combinación de dos programas distintos: el macroprocesador y el ensamblador. El macroprocesador es una extensión del algoritmo básico del ensamblador. El macroprocesador permite la definición y expansión de macros como una operación desligada del ensamblador.

El macroensamblador aumenta la capacidad de los ensambladores al permitir la definición de nuevas operaciones mnemónicas no incluidas en el repertorio de las instrucciones de la computadora y obtenidas al emplear una combinación de las mismas instrucciones mnemónicas.

Los macroensambladores proporcionan mucha más potencia que la disponible en un ensamblador simple ya que intentan reducir la dependencia con la máquina usada.

Algunos macroprocesadores son diseñados para un lenguaje particular, ejemplo de ello son los macroensambladores y macroprocesadores para ciertos lenguajes de alto nivel. Sorprendentemente muchos

de estos macroprocesadores aunque diseñados para un lenguaje particular son capaces de generar cualquier lenguaje. Esto se debe a -- que la mayoría del proceso del macroprocesador es independiente -- del lenguaje usado.

Por otra parte hay macroprocesadores que están restringidos a un solo lenguaje, ejemplo de ello son los macroensambladores en -- que el macroprocesador no es un paso previo al proceso de ensamble sino que se ejecuta al mismo tiempo que el ensamblador.

Hay tambien macroprocesadores que son diseñados expresamente - para generar cualquier salida de un lenguaje. Por ejemplo GPM (General purpose macrogenerator) que fue originalmente diseñado para ayudar a escribir un compilador para CPL. Cuando se diseñó el ma-- croprocesador GPM se empezaron a usar los macros para reducir el - número de líneas que se repetían. Más tarde se encontró que proporcionaban otros beneficios, así que lo que empezó como un método, - se convirtió en una política, al punto de que todas las secciones con código de máquina se incorporarón como macros, aunque esto implicara definir una macro especial que solo se utilizara una sola vez.

Los macroprocesadores pueden ser la base para generación telescópica de software portable. También pueden usarse como herramienta especializada para traducción. Por ejemplo: para reemplazar cada aparición de WRITE (6 por WRITE (60 o para cadenas escritas - de la forma " XYZ " a la forma 3HXYZ.

El macroprocesador PL/I es un intento para hacer un macrolen-- guaje que es esencialmente el mismo que un compilador de lenguaje de alto nivel.

3.10.1 Planteamiento del problema.

La función del macroprocesador es:

Reconocer y procesar: macrodefiniciones y macrollamadas.

Por lo tanto debe:

1. Reconocer las macroinstrucciones.

   En el primer paso al examinar cada sentencia se debe examinar - si aparece la pseudo-instrucción MACRO α d_1, d_2, d_3,... d_n y la pseudo-instrucción MEND.

   Se checará que para cada MACRO exista su correspondiente MEND.

2. Guardar el macronombre en una tabla para macronombres.

   Se rectificará que esté escrita de acuerdo a las reglas de sintaxis.

3. Conservar los parámetros mudos d_1, d_2, ... d_n en una lista - auxiliar. También habrá que rectificar su sintaxis.

4. Almacenar el cuerpo de la macro.

   La tabla para almacenar el cuerpo de la macro es una tabla a la que no se le puede asignar un número fijo de bits. Por ser de - la longitud variable, se recomendaría manejar un archivo donde se iría almacenando cada línea del cuerpo de la macro. Se tendría que llevar un contador de líneas que serviría para asociar a cada macronombre α el número de línea donde termina.

   Sería conveniente marcar los argumentos mudos que aparecen en el cuerpo de la macro, para facilitar posteriormente se sustitución por los argumentos reales.

   Esto se puede hacer si sustituimos cada argumento mudo d_i por: un carácter ^ (o cualquiera que no se use en el lenguaje ensamblador) seguido del valor i al que se le asignan dos localida-

des. Ver figura 3.12

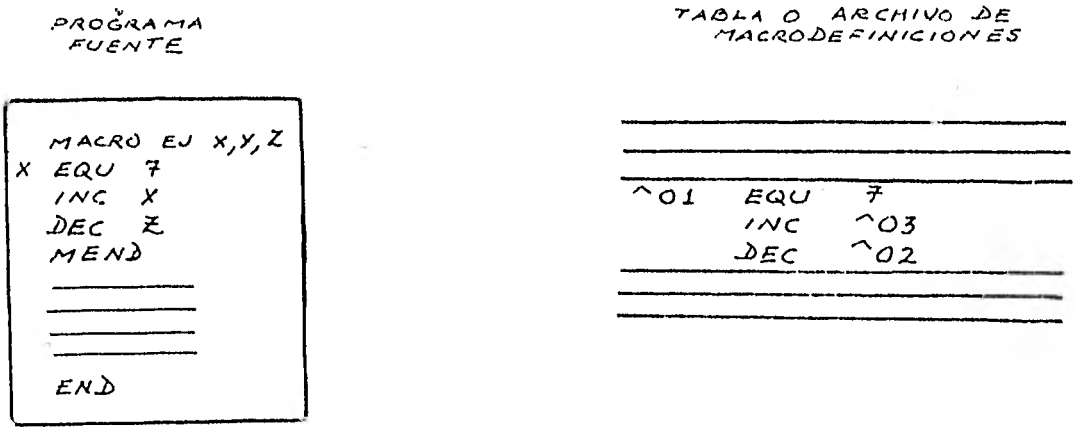


Figura 3.12

Para sustituir cada d_i por su correspondiente ^i necesitaremos rutinas para agrandar o achicar cada línea por ej. en la figura 3.12 X que ocupa un solo carácter que debe sustituirse por tres caracteres, por lo tanto deberá hacerles espacio a dos nuevos caracteres y conservar el resto de la línea invariable.

Gear (13) aconseja colocar una marca en las líneas en que aparecierón argumentos mudos, esto facilitaría la sustitución de los argumentos reales.

5. Reconocer las macrollamadas.

   El macroprocesador al encontrar en el campo del operador una macrollamada deberá reconocerla, y traerá de la tabla de macronombres apuntadores al archivo de macrodefiniciones.

6. Expandir macrollamadas y sustituir los argumentos.

   El macroprocesador deberá insertar en el programa fuente el cuerpo de la macrodefinición cambiando cada argumento mudo por uno real. Es decir cambiar cada ^I por su a_i.

   Para llevar a cabo esta sustitución se ocuparán rutinas que

reduzcan o agranden la línea.

El diseñador deberá resolver la acción a ejecutar:

a) Si hay menos argumentos reales que argumentos mudos.

b) Si hay más argumentos reales que argumentos mudos.

3.10.2 Diferentes criterios de implementación.

Los procesadores de macros pueden implementarse de diversas maneras:

a) Procesador independiente de 2 pasos.

El macroprocesador es implementado independiente del ensamblador. En el primer paso se reconocen las macrodefiniciones y se almacenan en su respectiva tabla. En el segundo paso reconoce las macrollamadas y las expande. No es necesario que las macrodefiniciones precedan a las macrollamadas.

b) Procesador de una pasada.

También es un procesador independiente del ensamblador.

El macroprocesador puede incorporarse al ensamblador y trabajar junto con él en el primer paso del ensamblador.

Esta implementación elimina archivos intermedios y puede mejorar tanto al ensamblador como al macroprocesador, ya que combina tablas y funciones similares.

3.11 Macro para Motorola 6800.

Las facilidades que da esta macro fueron diseñadas para extender el lenguaje ensamblador al incorporarle macros.

No se encontró en ningun manual de Motorola referencias que facilitarán macroinstrucciones, por lo tanto se diseñaron las macros para Motorola 6800 y se les dieron las facilidades que se creyeron convenientes.

El formato de una macroinstrucción es:

MACRO nombre $[d_1, d_2, \ldots . d_n]$

MEND

MACRO.- Pseudo-instrucción que indica el principio de la definición de la macro. No admite etiqueta.

nombre.- Sigue las mismas reglas de sintaxis que cualquier símbolo.

$d_1, \ldots, d_n$.- Argumentos mudos. Van separados por comas y al encontrar un blanco se sabe que acabó la lista. Pueden aparecer tantos como quepan en una línea.
Los argumentos mudos pueden aparecer en cualquier campo de una sentencia del cuerpo de la macrodefinición. El uso de estos argumentos es opcional. Cada d_i sigue las reglas de sintaxis de los símbolos aunque no importa que se repita un d_i en diferentes ma

cros porque se almacenán en tablas, de ahí su nom-- bre de argumentos " mudos ".

MEND.- Pseudo-instrucción que indica el fin de la defini-- ción de la macro.

El formato de una macrollamada será:

[etiqueta] nombre $[a_1,....,a_n]$

Y solo puede escribirse si se ha definido previamente su correspondiente macrodefinición, de otro modo causa un error.

nombre.- La macrollamada sigue las mismas reglas de sintaxis que las de un símbolo por corresponder al macronombre de una definición. Puede aparecer tantas veces como se quiera en un programa.

$a_1,...,a_n$.- Los argumentos reales son opcionales, van separados por comas, pueden ir tantos como quepan en la línea. Cada a_i es un conjunto cualquiera de caracteres. Si solo aparece una coma sin que le precedan caracte-- res entonces a_i se considera un blanco.

Si el número de argumentos reales es mayor que los argumentos mudos no se tomarán en cuenta los que sobren.

Si el número de argumentos reales es menor que el de los argumentos mudos se marcara error.

No se permiten macros anidados, macros interiores, ni macros - recursivos.

DISEÑO DEL MACROENSAMBLADOR.

CAPITULO IV

Ya se vió lo que es en general un macroensamblador, es decir - se planteo un problema. Ahora corresponde presentar su resolución para una máquina específica: la M6800.

A continuación se presenta la metodología que se llevó a cabo para diseñar el macroensamblador. Se presentan los diagramas de -- flujo del programa y finalmente se hace un comentario donde se justifican los objetivos que no se pudieron lograr.

4.0

El ensamblador diseñado para Motorola 6800 es de 2 pasos, se - implementó en una minicomputadora PDP - 11/34 por lo que es un ensamblador cruzado.

El macroprocesador está incorporado al primer paso del ensam-- blador.

El macroensamblador está escrito en el lenguaje ensamblador Macro - 11.

4.1 Acciones previas al primer paso.

Antes de empezar a diseñar el primer paso del ensamblador se necesitan:

a) Construir la tabla de mnemónicos, pseudo-instrucciones, símbolos, literales, macronombres y macrodefiniciones.

b) Tener un formato para el listado del programa ensamblado. Ver ápendice C.

c) Preparar una lista de los errores de sintaxis en que se puede incurrir. Ver ápendice D.

Para construir cada tabla debemos:

i) Formatear cada vector de la tabla.

ii) Elegir un método para buscar cada llave k.

iii) Organizar la tabla.

Después de llevar a cabo el inciso i) de todas las tablas y estudiar varios métodos de búsqueda se tomó la decisión de implementar en una sola tabla todas las tablas que se requerían, excepto la tabla de macrodefiniciones para la que se eligió un archivo de trabajo. Una persona que no tenga acceso a construir archivos podrá llevar un bloque de memoria o un arreglo que permita almacenar las líneas del cuerpo de la macro.

El contar con una sola tabla da las siguientes facilidades:

- Mayor rapidez en el proceso de búsqueda ya que solo se debe buscar la llave en una sola tabla. Esto es de gran utilidad sobre todo para localizar argumentos mudos.
- La elección de un sólo metodo de búsqueda y de inserción. Esto reduce además el número de instrucciones a ejecutar.
- Se aprovechan al máximo el espacio de la tabla.

. Se reduce el espacio de memoria que ocupa el ensamblador.

La desventaja - que realmente es mínima - es la de asociar a - cada llave una clave que indique su origen.

4.2 Radix - 50.

Las palabras de la computadora PDP - 11 / 34 son de 16 bits - lo que permite almacenar en ellas 2 caracteres en código ASCII.

Cada llave de la tabla tiene como máximo 6 caracteres alfanuméricos. Si almacenamos las llaves de la tabla utilizando el código ASCII tendriamos que reservar 3 palabras para cada llave.

Uno de los objetivos de un buen programador es el de ahorrar memoria al implementar un programa. Si se observa que las llaves - sólo utilizan de 36 a 40 caracteres (26 caracteres alfabéticos, 10 números y unos cuantos caracteres especiales), no resultara sorprendente que se puedan almacenar 3 caracteres en una palabra de 16 -- bits.

Macro - 11 facilita el formato radix - 50 que almacena 3 caracteres en cada palabra. Para implementar radix - 50 hay que basarse en la siguiente formula:

$$((C_1 * 50_8) + C_2) * 50_8 + C_3$$

donde: C_1, C_2 y C_3 son los 3 caracteres a convertir a radix - 50.

La relación entre radix - 50 y ASCII está dada en la siguiente tabla:

	ASCII	Radix - 50
Espacio	40	0
A - Z	101 - 132	1 - 32
$	44	33
.	56	34
No se usa		35
0 - 9	60 - 71	36 - 47

El valor máximo que puede tomar una palabra escrita en radix - 50 es:

$$47 * 50^2 + 47 * 50 + 47 = 174777$$

4.3 Formato de la tabla.

Se empezó por clasificar las llaves en grupos:

Grupo I : Pseudo-instrucciones:

a) generan código:

ORG, EQU, RMB, FCB, FDB, NAM, FCC.

b) no generan código:

END, MACRO, MEND.

Grupo II : Instrucciones que admiten:

a) varios modos.

b) modo inherente.

c) modo relativo.

Si se observa el ápendice A, se verá que hay instrucciones que admiten un solo modo. Esto sucede tanto para el modo inherente como para el modo relativo, sin embargo, se clasifican como diferentes casos porque para el modo relativo se deben efectuar una serie de procedimientos que no se ejecután en el modo inherente.

Grupo III: Etiquetas:

a) de símbolos.

b) de literales.

Grupo IV : Casos especiales:

a) macronombres

b) argumentos mudos.

Cada vector de la tabla ocupa 4 palabras. La llave ocupa las 2 primeras palabras. En la tercera palabra ocupando los bits 0 - 3 va una clave indicando el origen de la llave. La clave fue asignada de acuerdo a la acción que se debe ejecutar.

La tabla que se utilizó ocupó 1K de memoria, almacena 72 instrucciones y 9 pseudo-instrucciones y da la posibilidad de almacenar 175 llaves más.

A continuación y en el orden de la clave se muestran los diferentes formatos.

1. END

2. Macronombre

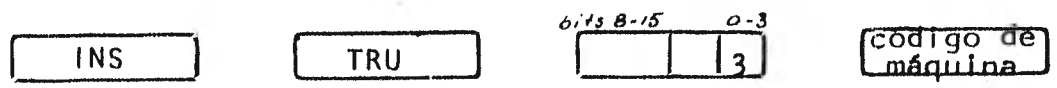
donde:

NOMBRE = es el nombre del macronombre

NFIRST = apuntador al archivo de macrodefiniciones. Indica el número de línea donde empieza el cuerpo de la macro. Ocupa los bits 4-15 de la tercera palabra.

NLAST = apuntador al archivo de macrodefiniciones. Indica la línea donde termina el cuerpo de la macrodefinición.

3. Instrucción que acepta varios modos.



INSTRU = es el mnemónico de la instrucción

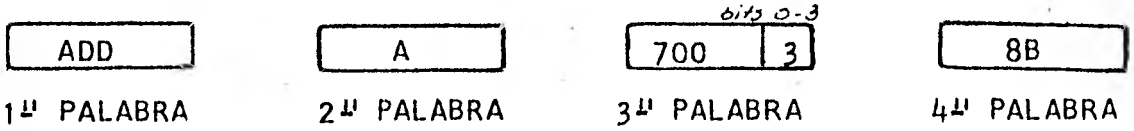
Código de máquina = son dos caracteres que forman un número en hexadecimal. Se coloca el código de máquina del primer modo que admita la instrucción de acuerdo al ápendice

A. Obsérvese que los códigos de los siguientes modos - aumentan en 1 la primera componente del código.

Por ej: la instrucción ADDA (ver ápendice A) tiene - como código de máquina para el modo:

- inmediato 8 B
- directo 9 B
- indexado A B
- extendido B B

Por lo que quedara almacenada como:



Asociado a cada instrucción debe existir una clave que indique los modos de direccionamiento permitidos.

Así que en la tercera palabra en el byte alto tendre-- mos que:

bit 8 encendido significa admite modo inmediato

bit 9 " " " " directo

bit 10 " " " " indexado

bit 11 " " " " extendido

bit 15 encendido significa que el modo inmediato se-incrementa en 3 bytes en lugar de 2.

El orden en que están encedidos los bits es el orden - en que aparecen los modos en el ápendice A.

En consecuencia la información que puede aparecer en - la tercera palabra es:

6003 Instrucción que acepta modo indexado y extendido.

7003 Instrucción que acepta modo directo, indexado y

extendido.

107403 o

7403 instrucción que acepta modo inmediato, directo, indexado y extendido.

Estos números servirán de auxiliar para obtener el verdadero código de máquina que le corresponda a una instrucción de acuerdo a su modo.

4. Instrucciones que solo admiten modo inherente.

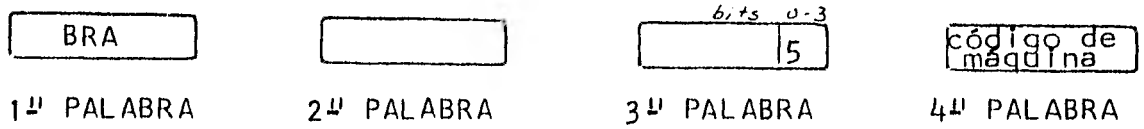
Por ejemplo:

donde:

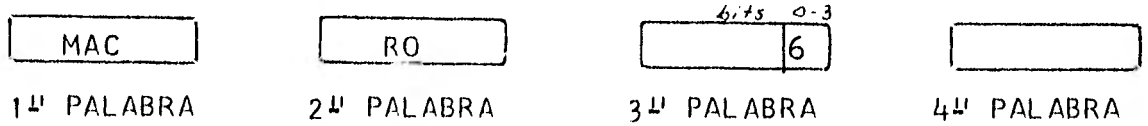
Código de máquina = 2 caracteres que forman un número en hexadecimal.

5. Instrucciones que solo admiten modo relativo.

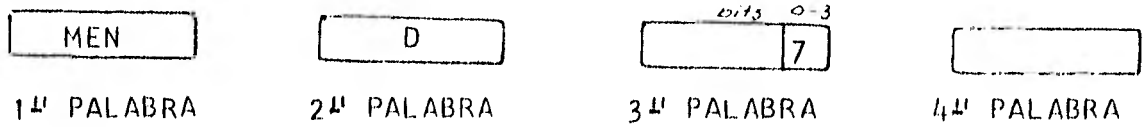
Por ejemplo:



6. Pseudo-instrucción MACRO.



7. Pseudo-instrucción MEND.



8. Pseudo-instrucciones: ORG, EQU, RMB, FCB, FDB, NAM y FCC.

1ª PALABRA 2ª PALABRA 3ª PALABRA 4ª PALABRA

donde:

PSEINS = mnemónico de la pseudo-instrucción

NUMERO = indica el tipo de pseudo-instrucción

/ Si

número	pseudo-instrucción
0	ORG
1	EQU
2	RMB
3	FCB
4	FDB
5	NAM
6	FCC

9. Etiquetas

a) Para símbolo

1ª PALABRA 2ª PALABRA 3ª PALABRA 4ª PALABRA

donde:

LABEL = es el nombre del símbolo

PLC = valor del PLC

b) Para literal

1ª PALABRA 2ª PALABRA 3ª PALABRA 4ª PALABRA

donde:

LITERA = nombre de la literal

NUMERO = es el valor asignado por una expresión

10. Argumentos mudos.

Son los argumentos mudos de las macroinstrucciones. Estos argumentos se borran de la tabla después de pasar al archivo de macrodefiniciones el cuerpo de la macro. Su formato es:

1ª PALABRA 2ª PALABRA 3ª PALABRA 4ª PALABRA

donde:

MUDO = nombre del argumento mudo

NUMERO = colocamos un número dependiendo de su orden de aparición. Si es el primer argumento mudo un 1, si es el segundo un 2 etc.

4.4 Método elegido para almacenar y buscar llaves: Hash.

El método que se recomienda para la construcción de tablas de símbolos tanto de ensambladores como de compiladores es el de hash, aunque para los ensambladores pueden emplearse otros métodos. Este método se puede aplicar a cualquier archivo o tabla en que los datos a almacenar entran en orden impredecible.

La principal ventaja es que los datos pueden identificarse a través de una llave que forma parte de los mismos datos.

La idea fundamental de este método se basa en que la llave asociada a cada vector después de alguna transformación produce una dirección a la tabla donde se localizan la llave y sus valores asociados.

Si k es una llave, existirá una función h tal que:
h (k) = A donde A es la dirección a la tabla.

Desafortunadamente es difícil desarrollar funciones que sean uno a uno. Cuando se presenta el caso en que dos diferentes llaves coinciden al producir la misma dirección, se está ante lo que se llama una colisión. A este tipo de llaves les llamamos sinónimas.

A la dificultad de encontrar funciones uno auno hay que añadirle la de restringir a un cierto intervalo el valor del dominio de la función. Lo anterior tiene su razón de ser si se piensa que para almacenar una llave con 6 caracteres, se necesitán 26^6 localidades de la tabla para que no se produzca una colisión, pero no es posible darse el lujo de tener tablas tan grandes en consecuencia la solución es restringir el tamaño de la tabla y por ende el de la función, de tal manera que:

$0 \leq h (k) \leq M$

donde M es la cota máxima del intervalo.

El diseñador del ensamblador deberá:

. Indicar el tamaño de la tabla y su arreglo.

. Elegir la función de hash.

. Elegir un método para resolver colisiones.

El tamaño de la tabla va relacionado con la función de hash y con el método para resolver colisiones, generalmente se escoge como tamaño de la tabla un número que sea una potencia de 2 o un número primo. Por ejemplo si la función de hash genera r bits como índice a la tabla, se necesitará un espacio de $N = 2^r$ localidades lo que permitirá un número al azar entre 0 y 2^{r-1}.

4.4.1 Arreglo de la tabla.

Al colocar un vector en la tabla de almacenamiento se ocupa más de una palabra. Al menos se necesita una palabra para la llave y otra para la información asociada a ella. Si un vector ocupa k palabras y el espacio de la tabla es de N localidades entonces se necesitán k * N palabras para almacenar toda la información de la tabla.

Se sugieren 2 caminos para resolver esta dificultad:

1. Colocar cada llave y sus valores asociados en palabras consecutivas de memoria y usar solo las direcciones de hash que apunten a la primera palabra. Esto se puede efectuar rápidamente multiplicando cada dirección de hash por k y usando el producto como índice de la tabla.

2. Dividir la tabla en k secciones cada una con N palabras. La dirección de hash se usará como índice en la primera tabla que contenga información asociada al vector; con este mismo in-

dice se buscará más información en la segunda tabla y así sucesivamente hasta llegar a la última tabla.

El programador puede elegir el método que más le convenga para programación ya que no existe diferencia entre ellos.

Para el ensamblador de Motorola se eligió el primer método.

El programador debe saber distinguir entre una localidad vacía y otra llena. Por ejemplo: si se excluye el número cero como llave, entonces al encontrar un cero en una localidad se indicará que está vacía. Al inicializarse una tabla deben marcarse las localidades vacías.

4.4.2 Función de hash.

Existen diferentes tipos de funciones de hash, generalmente se clasifican en métodos lógicos y metodos multiplicativos.

Una de las dificultades para escoger una buena función fue que las instrucciones de Motorola (Ver ápendice A) son muy parecidas por ej: tenemos llaves como DEC, DECA, DECB que facilmente pueden producir colisiones.

El método que se eligió fue una combinación de folding (plegadizo) que utiliza el OR exclusivo y el método de radix - 50.

El método de folding es un método muy rápido para obtener r -- bits de código de una llave de n bits para $r \leq n$.

Consiste en escoger algunos de los r bits del campo de una llave de n bits y sumarlos o aplicarles el OR exclusivo.

Por ejemplo, considerese una llave compuesta por cadenas de dígitos escritos en una tira de papel que se pliega como en la figura 4.1 (b). Todos lo dígitos que se encuentran frente a frente - (figura 4.1 (c)) se suman y producen una dirección. Este méto-

do se combina con otros cuando el tamaño de las llaves es grande.

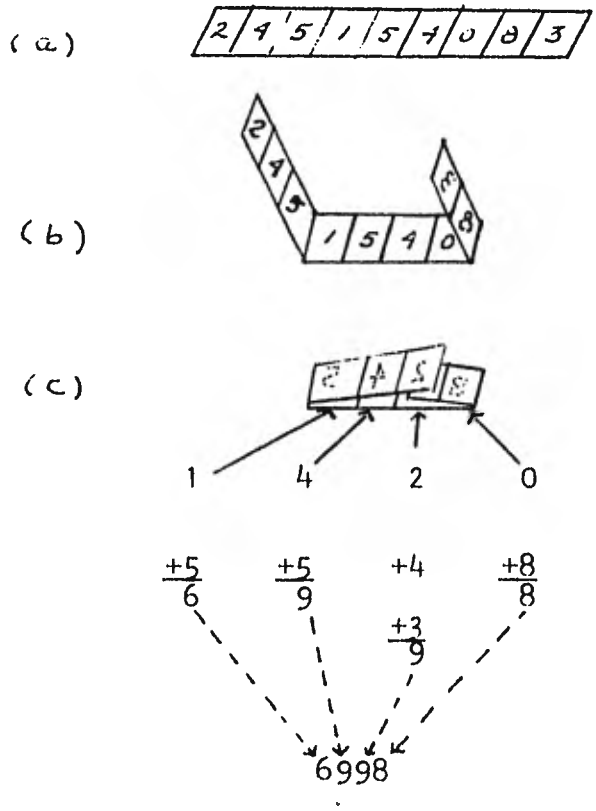


Figura 4.1

Los diagramas de flujo con la función de hash que se utilizó y la manera de buscar una llave e insertarla se muestran respectivamente en la página 137 con las subrutinas Hasaux y Haslab.

4.4.3 Método para resolver colisiones.

Es inevitable que no se produzcan colisiones y éstas aumentan en la medida en que se va llenando la tabla.

Se deberá escoger cuidadosamente el método para resolver colisiones, ya que afectará profundamente la eficiencia de la técnica de hash.

Entre los métodos que existen se eligió el método de prueba al azar (random probing) por la facilidad que proporciona para almacenar y recuperar una llave.

El algoritmo para calcular direcciones que resuelvan las colisiones es el siguiente:

1. Usar una función de hash para determinar una dirección d que actue como fndice a la tabla.
2. Si la localidad de la tabla está vacia o si se encontró la llave buscada, el algoritmo termina.
3. Si en la localidad hay otra llave, llamar al generador de números pseudo-random para producir un número entero ρ.
4. Hacer la siguiente prueba con $d + \rho$ e ir al paso 2.

El número pseudo-random ρ puede ser muy simple de calcular y ocupa pocas instrucciones de máquina. Debe generar una sola vez cada entero desde 1 hasta N-1 (donde N es el tamaño de la tabla). Cuando el número pseudo-random se sale de los números enteros la tabla esta llena y no podrán almacenarse más datos.

El siguiente algoritmo se usó para producir el número pseudo-random. Solo sirve para tablas cuyo tamaño N es una potencia de 2.

Algoritmo para generar ρ para tablas cuyo tamaño es $N = 2^n$.

1. Hacer R$\leftarrow$1 si se va a resolver por primera vez la colisión; si no ir directamente a 2.
2. Hacer R$\leftarrow$ R * 5
3. Solo dejar en R n + 2 bits, eliminando los bits superiores de la palabra.
4. Hacer ρ = R / 4 y regresar al algoritmo principal.

Parte de la serie de números pseudo-random que produce este mé

todo para una tabla de $N = 2^{10}$ (que es la que se uso) es:
1, 6, 37, 234, 1415, 1502, 113 ... etc.

Cuando se ha elegido la función de hash y el método para resolver colisiones se procederá a aplicarles la función de hash a cada una de las instrucciones y pseudo-instrucciones, posteriormente se procederá a resolver las colisiones. Evidentemente se utilizó un - programa para efectuar este proceso. Luego de correr el programa - se sabrá en que dirección queda almacenada cada llave y se procederá a construir la tabla, es decir se habrá de colocar en sus localidades respectivas la llave y los datos asociados a ella.

El diagrama de flujo para calcular ρ se puede consultar en la página 136 subrutina Full.

La mejor manera de explicar la eficiencia del método de prueba al azar, es a traves del número promedio de pruebas - llamemosle E - necesarias para recuperar una llave de la tabla.

Este número promedio E también es igual al número de pruebas - que se efectúan para almacenar una llave en su localidad.

E depende del factor de carga α es decir de la porción de la - tabla que esté ocupada.

Si N es el tamaño de la tabla y k llaves están almacenadas en ella entonces $\alpha = k / N$.

El número esperado de pruebas - llamemosle A - necesarias para almacenar la (k + 1) - ésima llave, incluyendo la prueba final - es:

$$A = 1 + \frac{k}{N} + \frac{k(k-1)}{N(N-1)} + \ldots + \frac{k(k-1)\ldots 1}{N(N-1)\ldots(N-k+1)} \quad \ldots (1)$$

donde el j-ésimo término en la suma es la probabilidad de que j o más pruebas sean necesarias para almacenar la llave.

Por inducción sobre k, (1) puede escribirse como:

$$A = 1 + \frac{k}{N - k + 1} = \frac{N + 1}{N - k + 1} = \frac{1}{1 - \frac{k}{N + 1}} \quad \ldots\ldots (2)$$

para $1 \leq k \leq N$

Para valores grandes de N se puede remplazar k / (N + 1) por α - en (2) y aproximadamente el valor de A será:

$$A = \frac{1}{1 - \alpha} = 1 + \alpha^2 + \alpha^3 + \ldots$$

cuya interpretación intuitiva sería: con probabilidad α se necesitán más de una prueba, con probabilidad α^2 se necesitán más de dos etc.

El número promedio de pruebas es igual al promedio de A para - valores que toma k del intervalo 0 a k - 1.

Se puede calcular este promedio por la integral:

$$E = \frac{1}{\alpha} \int_0^{\alpha} \frac{dx}{1 - X} = \frac{1}{\alpha} \left[\log_e (1 - x) \right]_0^{\alpha} = \frac{1}{\alpha} \log_e (1 - \alpha) = 1 + \frac{\alpha}{2} + \frac{\alpha^2}{3}$$

Algunos valores simples de E para varios valores de son:

Factor de carga α	E
.1	1.05
.5	1.39
.75	1.83
.9	2.58

PRIMER PASO

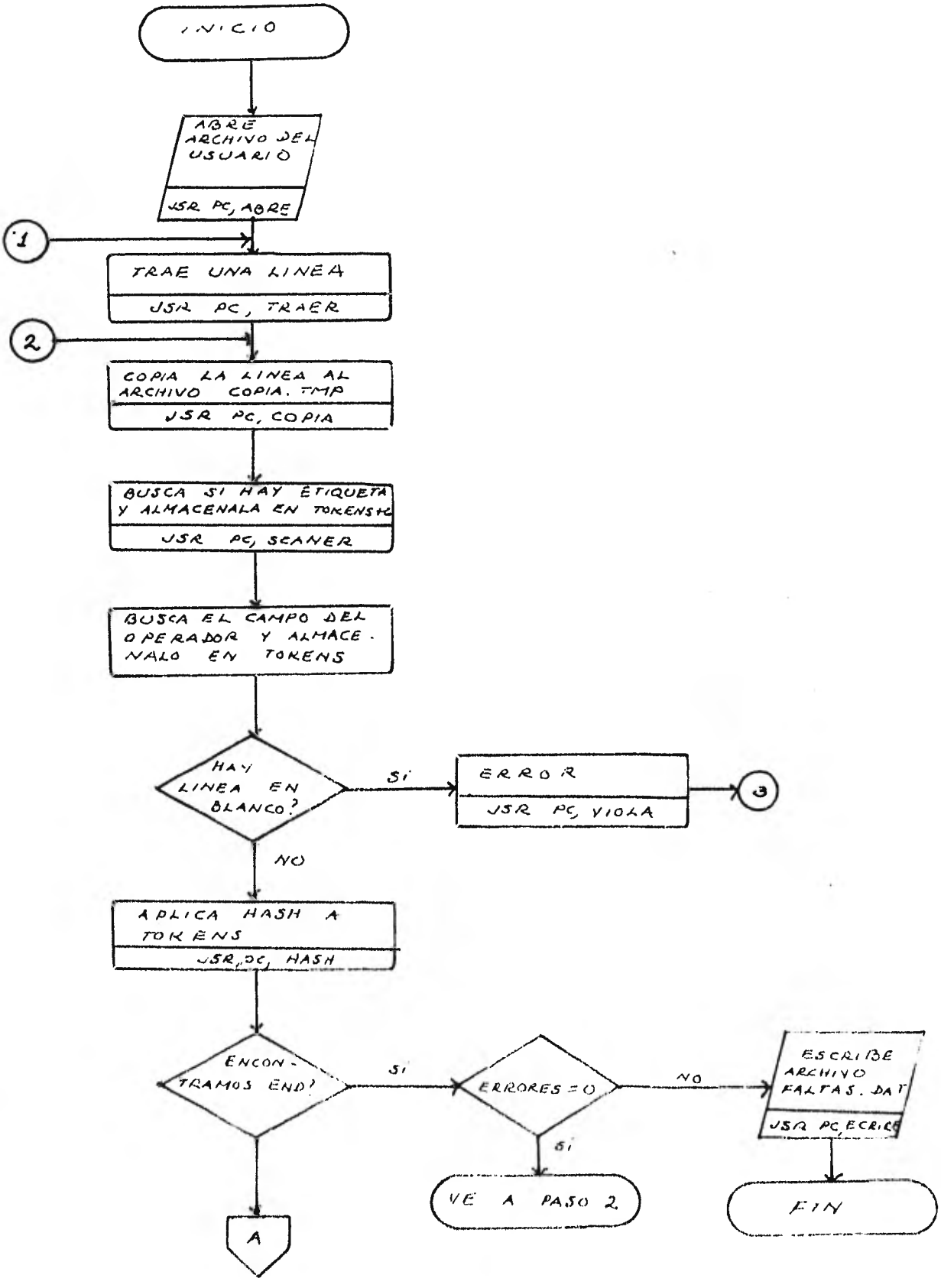


Fig. 4.2

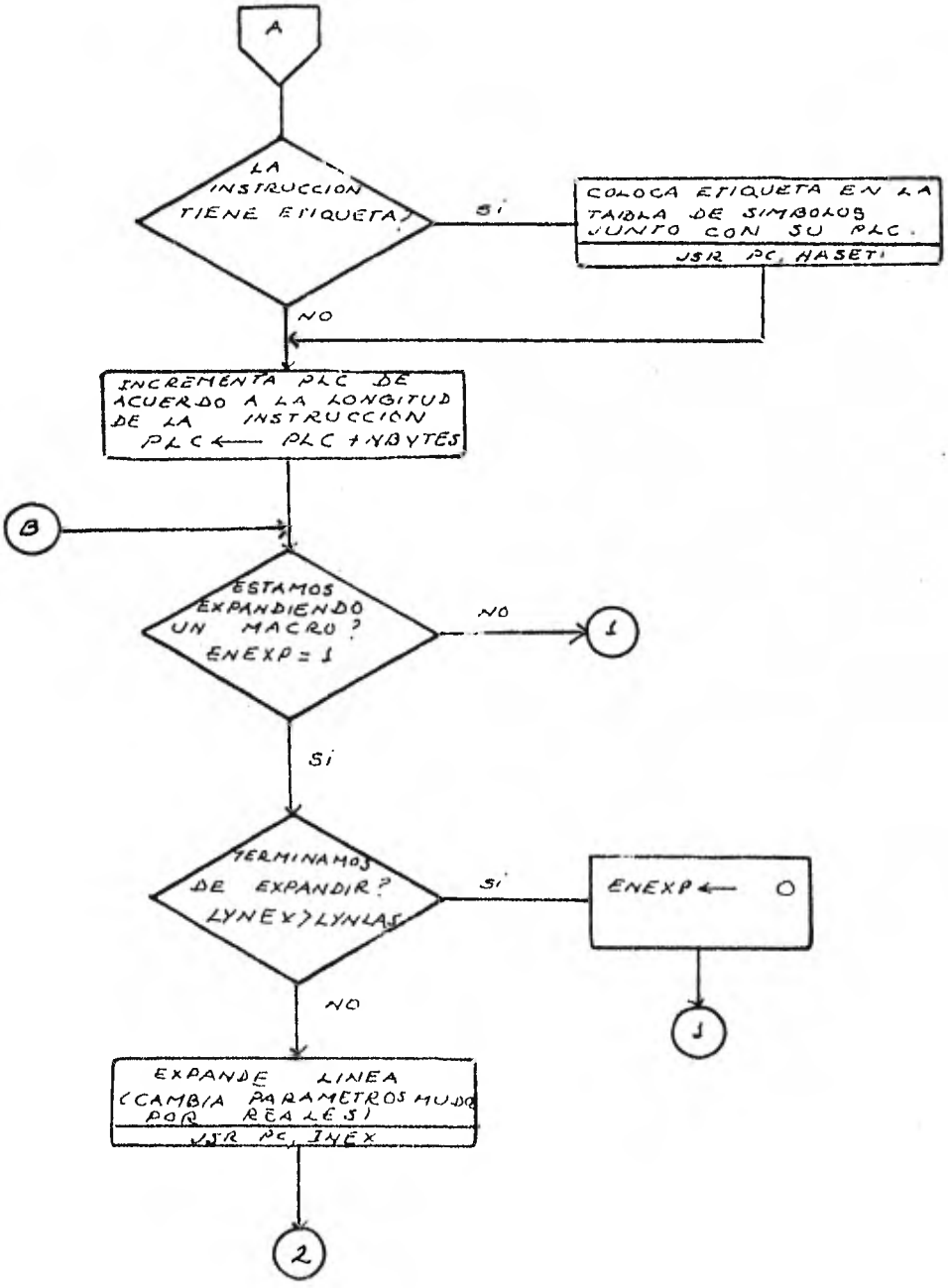

A
LA INSTRUCCION TIENE ETIQUETA?
SI
COLOCA ETIQUETA EN LA TABLA DE SIMBOLOS JUNTO CON SU PLC.
JSR PC, HASET1
NO
INCREMENTA PLC DE ACUERDO A LA LONGITUD DE LA INSTRUCCION
PLC ← PLC + NBYTES
B
ESTAMOS EXPANDIENDO UN MACRO ? ENEXP = 1
NO
1
SI
TERMINAMOS DE EXPANDIR? LYNEV > LYNLAS
SI
ENEXP ← 0
J
NO
EXPANDE LINEA (CAMBIA PARAMETROS MUDO POR REALES)
JSR PC, INEX
2

Fig. 4.2

SEGUNDO PASO

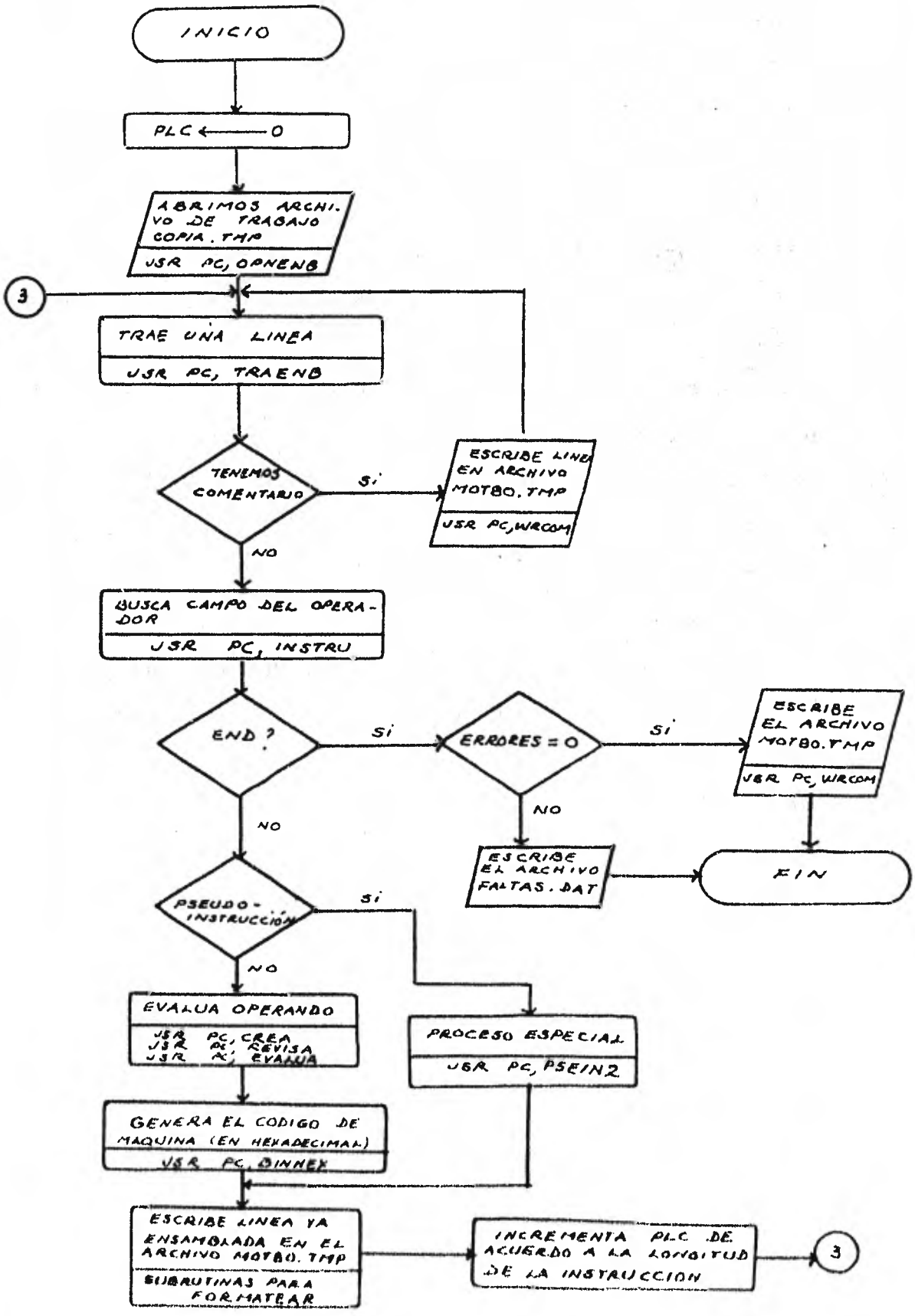


Fig. 4.3

4.5 Diagrama de flujo.

A continuación se presentan los diagramas de flujo del primero y segundo paso simplificados. En caso de querer seguir todo el diseño detenidamente se habrá de seguir los diagramas de flujo de cada subrutina.

Aunque las acciones del primero y segundo paso se encuentran en síntesis en los diagramas de flujo de las figuras 4.2 y 4.3 algunos procedimientos pueden resultar obscuros y por lo tanto se trata de aclararlos.

Al incorporar en el primer paso el macroprocesador existían 2 alternativas para cuando se debe expandir una macrollamada:

1. Actúan independientes el macroprocesador y el ensamblador.
   El macroprocesador para expandir una macrollamada trae todo el cuerpo de la macro, cambia los argumentos mudos por los reales y posteriormente permite actuar al ensamblador empezando con la primera línea de la expansión.
2. Actúan combinados el macroprocesador y el ensamblador.
   El macroprocesador al encontrar la macrollamada trae la primera línea del cuerpo de la macro, cambia argumentos mudos por reales, cuando termina este proceso comienza la acción del ensamblador, al terminar esta acción se continua con la segunda línea del cuerpo del macro y así sucesivamente hasta que se termina de analizar y transformar la última línea.

Ambos métodos requieren un espacio de trabajo mientras se efectua la acción del ensamblador, sin embargo el segundo método solo requiere almacenar una línea mientras que el primer método requiere espacio para varias líneas. La alternativa que se tomó fue la -

de utilizar el segundo método.

Para evitar que en el segundo paso se tengan que expandir las macrollamadas en el transcurso del primer paso se va elaborando un archivo que contenga cada una de las líneas del programa fuente así como la expansión de las macrollamadas.

Solo se podrá efectuar el segundo paso del ensamblador si en el primer paso no se encontraron errores. En caso de que el lenguaje no permitiera macros esta acción no se requeriría.

El máximo número de errores que detecta el ensamblador son 35. Al alcanzarse este número se da por terminada la acción del ensamblador.

El ensamblador trabaja con 5 archivos:

1. El archivo del usuario.
2. El archivo COPIA. TMP que se destruye después del segundo paso.
3. El archivo MACRO. TMP que almacena el cuerpo de las macrodefiniciones. Se destruye después del primer paso.
4. El archivo FALTAS. DAT que contiene los errores.
5. El archivo MOT80. TMP que contiene el programa ensamblado.

A continuación damos una lista de las subrutinas más importantes empleadas y la función que realizan. Se indica además si se emplean en el primero o en el segundo paso, en caso que no se diga es que se usan en ambos pasos.

ABRE. Primer paso.

Sirve para abrir el archivo del usuario y para crear un archivo de trabajo.

BARRE. Primer paso.

Elimina líneas en blanco o que tengan solo etiqueta.

CALPLC. Primer paso.

Calcula en cuánto debe incrementarse el PLC. Identifica el tipo de instrucción que es y la acción a ejecutar. Emplea otras subrutinas.

COPIA. Primer paso

Este procedimiento sirve para tener al final del primer paso un archivo que tiene las macrollamadas expandidas - (COPIA. TMP).

CREA. Construye un vector que contiene los operandos de una expresión en su forma binaria.

EVALUA. Evalúa una expresión.

FULL. Procedimiento para resolver colisiones. Calcula ρ .

INEX. Primer paso.

Rutina que sirve para expandir las macrollamadas. Cambia los argumentos mudos por los reales. Utiliza las subrutinas ACHICA, CAMBIA y DOBIG1 que sirven para agrandar o - reducir el buffer.

INSTRU. Rectifica la sintaxis de una instrucción y coloca los caracteres en el vector TOKENS.

LABELS. Rectifica la sintaxis de las etiquetas y las coloca en - el vector TOKENS + 6.

HASLAB. Primer paso.

Procedimiento para colocar un token en la tabla empleando una función de hash.

HASAUX. Procedimiento para buscar un token en la tabla.

PSEIN2. Segundo paso.

Rutina que se usa para procesar las pseudo-instrucciones

y escribir las líneas ensambladas en el archivo MOT80. - TMP.

REVISA. Revisa la sintaxis de las expresiones.

SBMAC. Primer paso.

Rutina que actúa en caso de encontrar la pseudo-instrucción MACRO. Almacena el cuerpo del macro en el archivo - MACRO. TMP y marca los parámetros mudos es decir coloca ^i en vez del argumento mudo d_i. Hace uso de las subrutinas ACHICA, CAMBIA y DOBIG1 para reducir o agrandar el - buffer.

SBMNAN. Primer paso.

Rutina que actúa en caso de encontrar una macrollamada. Nos dice cuantas líneas componen el cuerpo de la macro. Identifica cada argumento real y cuenta cuantos argumentos reales se tienen.

SBPSEU. Primer paso.

Procedimiento de las pseudo-instrucciones para incrementar el PLC.

SCANER. Primer paso.

Identifica el tipo de token y rectifica su sintaxis.

SEGUNDOP. Segundo paso.

Procedimiento que identifica el tipo de sentencia que se tiene. Incrementa el PLC y produce el lenguaje máquina. Llama a otras subrutinas para procesar pseudo-instrucciones y para escribir los formatos de salida.

STK. Procedimiento para cambiar cada uno de los operandos de una expresión a su forma binaria.

TRAE. Primer paso.

Siempre nos trae una línea del archivo del usuario.

Además de estas subrutinas se tienen subrutinas para formatear como: FORMAT, LASFRM, UTILFO, WRCOM.

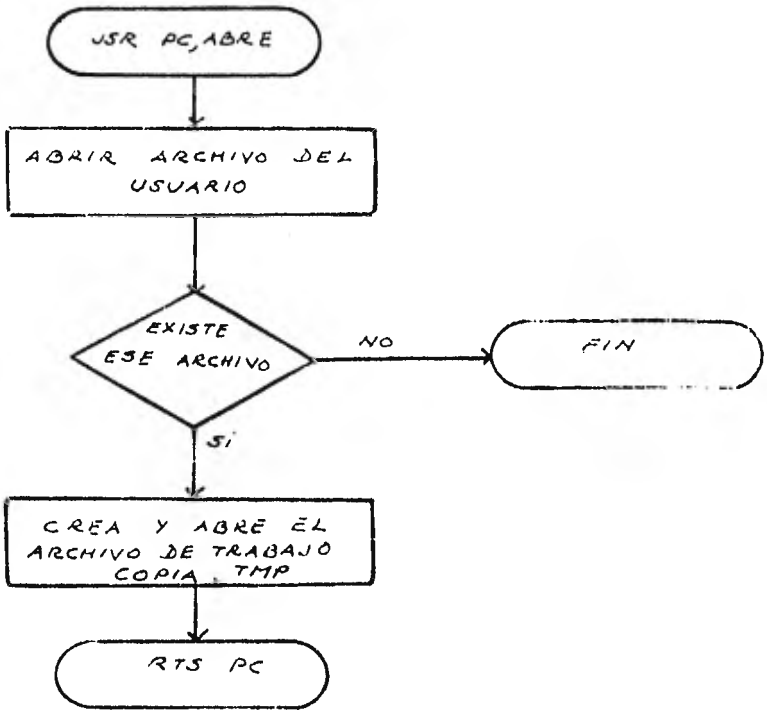
Otras subrutinas son:

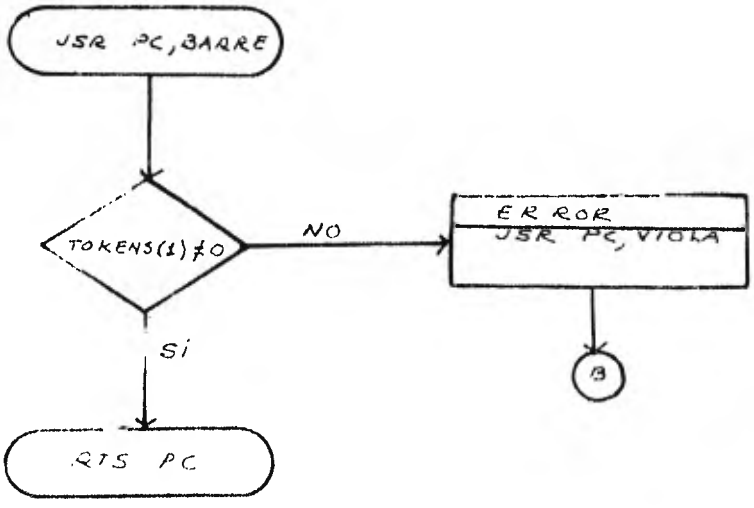
BINHEX. Sirve para convertir números de binario a hexadecimal.

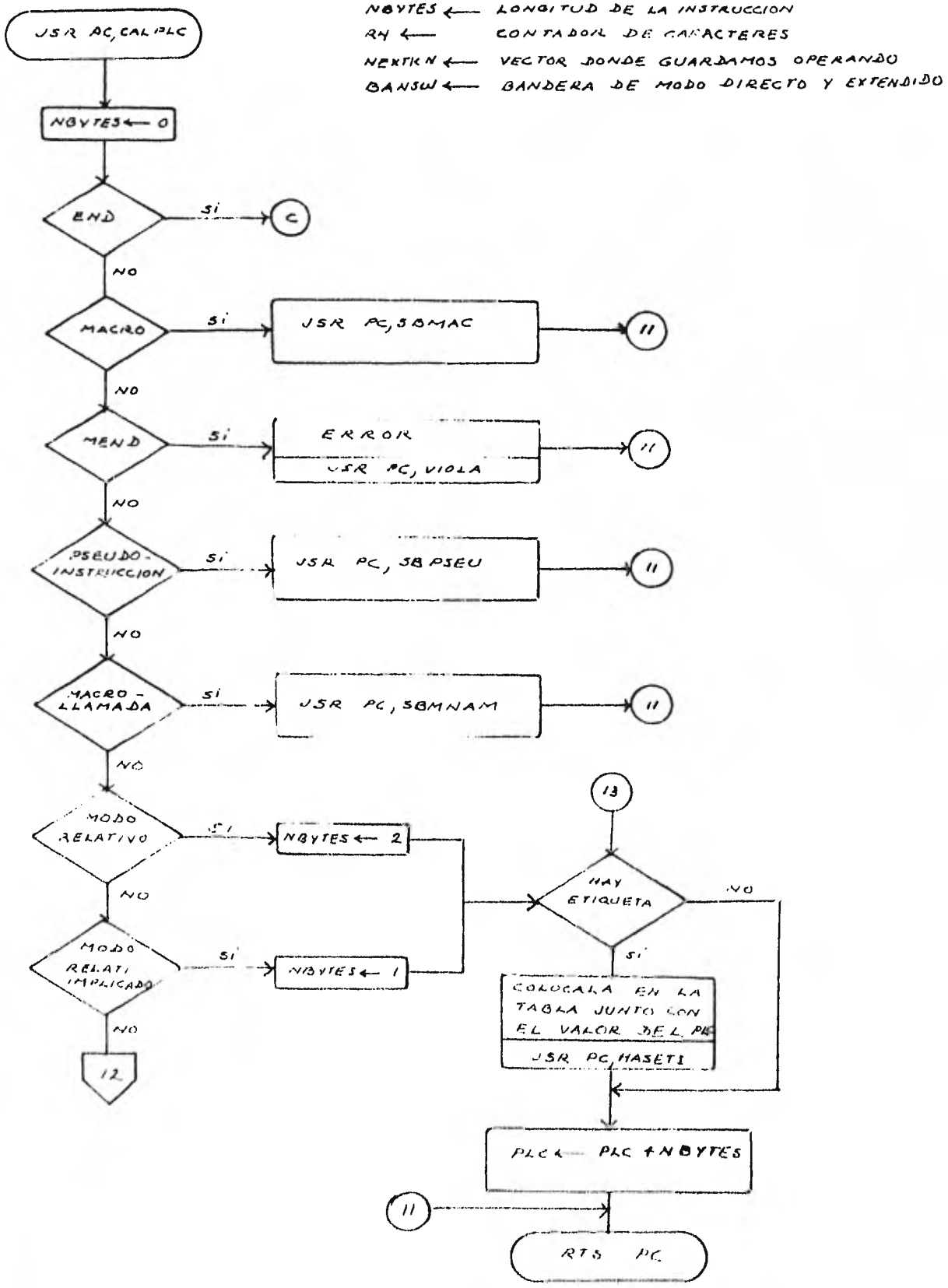
VIOLA. Crea un vector con el número de línea en que se produce un error y el tipo de error detectado.

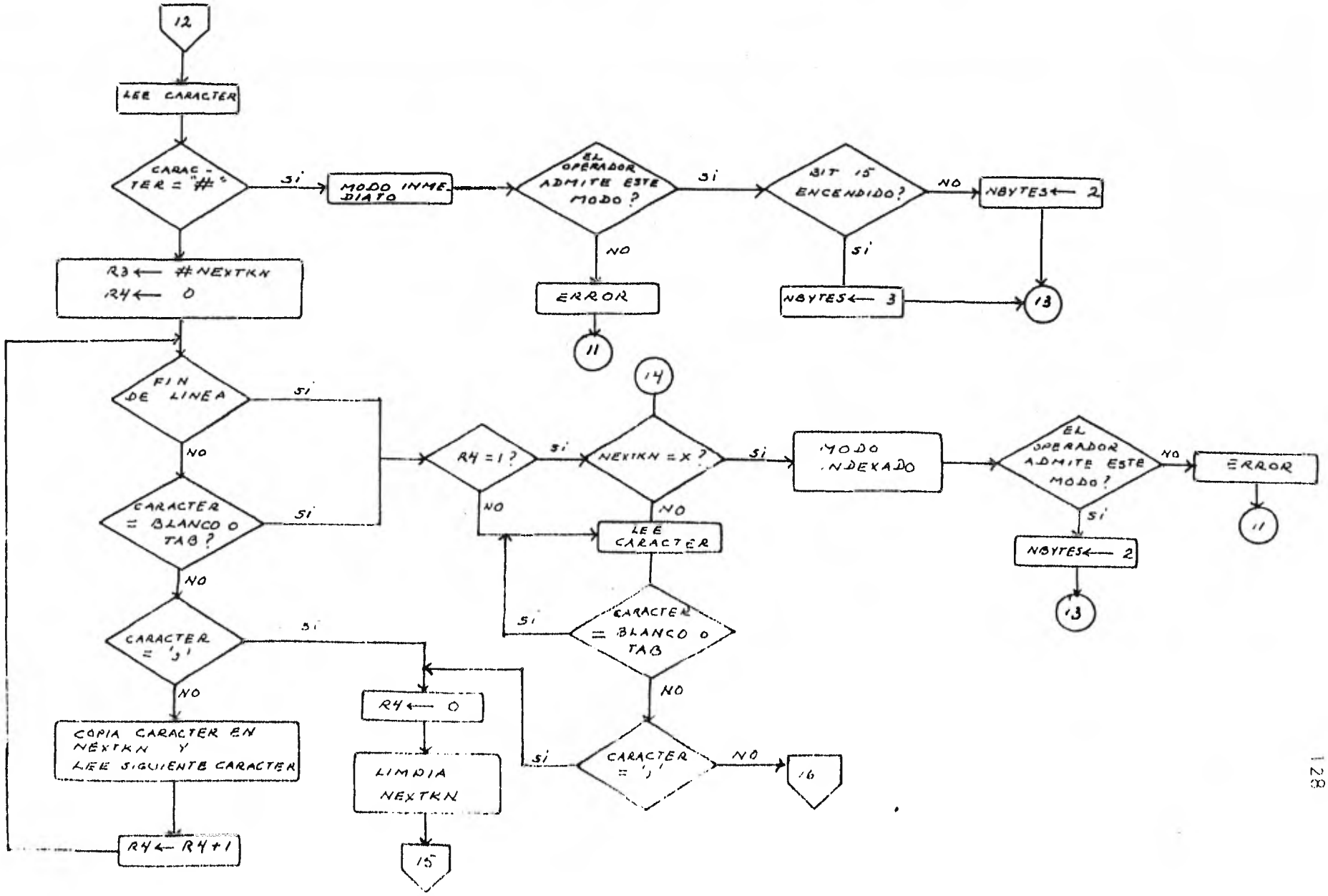
ECRIRE. Procedimiento que crea el archivo FALTAS. DAT que lista los errores producidos en el programa.

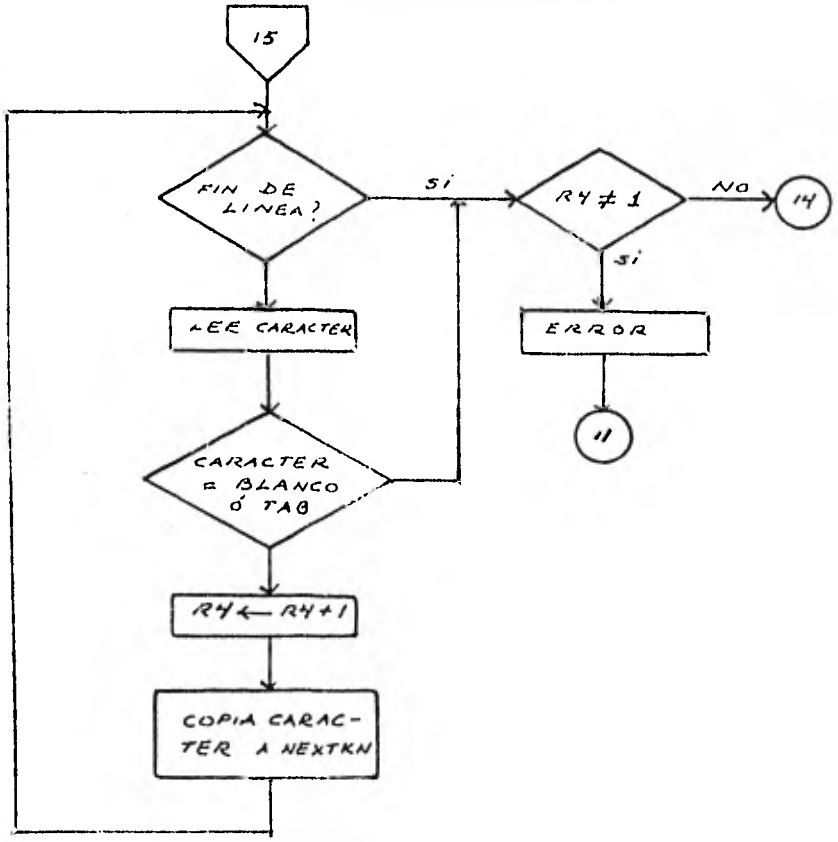
TSIMBOL. Lista los símbolos y literales almacenados en la tabla - junto con su valor.

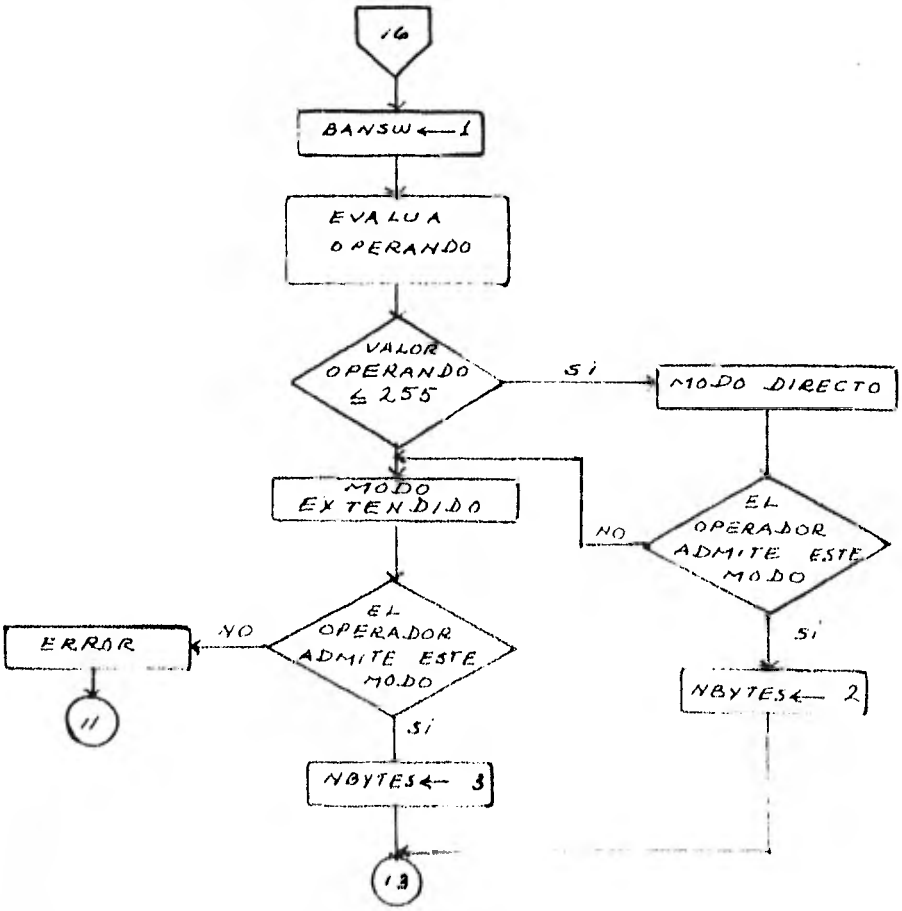

JSR PC,ABRE
ABRIR ARCHIVO DEL USUARIO
EXISTE ESE ARCHIVO
NO
FIN
SI
CREA Y ABRE EL ARCHIVO DE TRABAJO COPIA.TMP
RTS PC

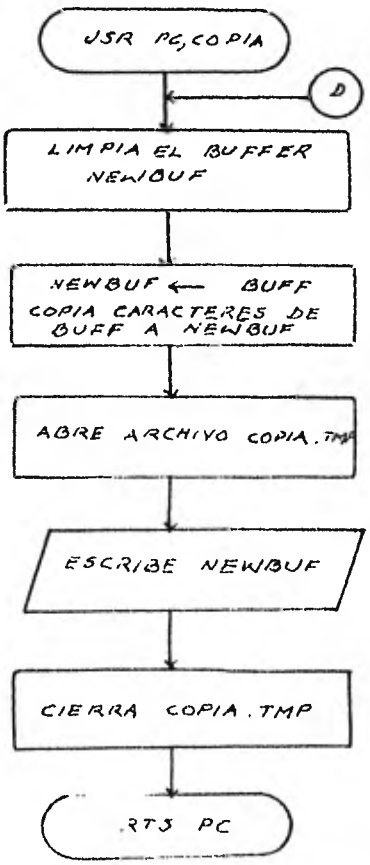

JSR PC,BARRE
TOKENS(1)≠0
NO
ERROR
JSR PC,VIOLA
B
SI
RTS PC

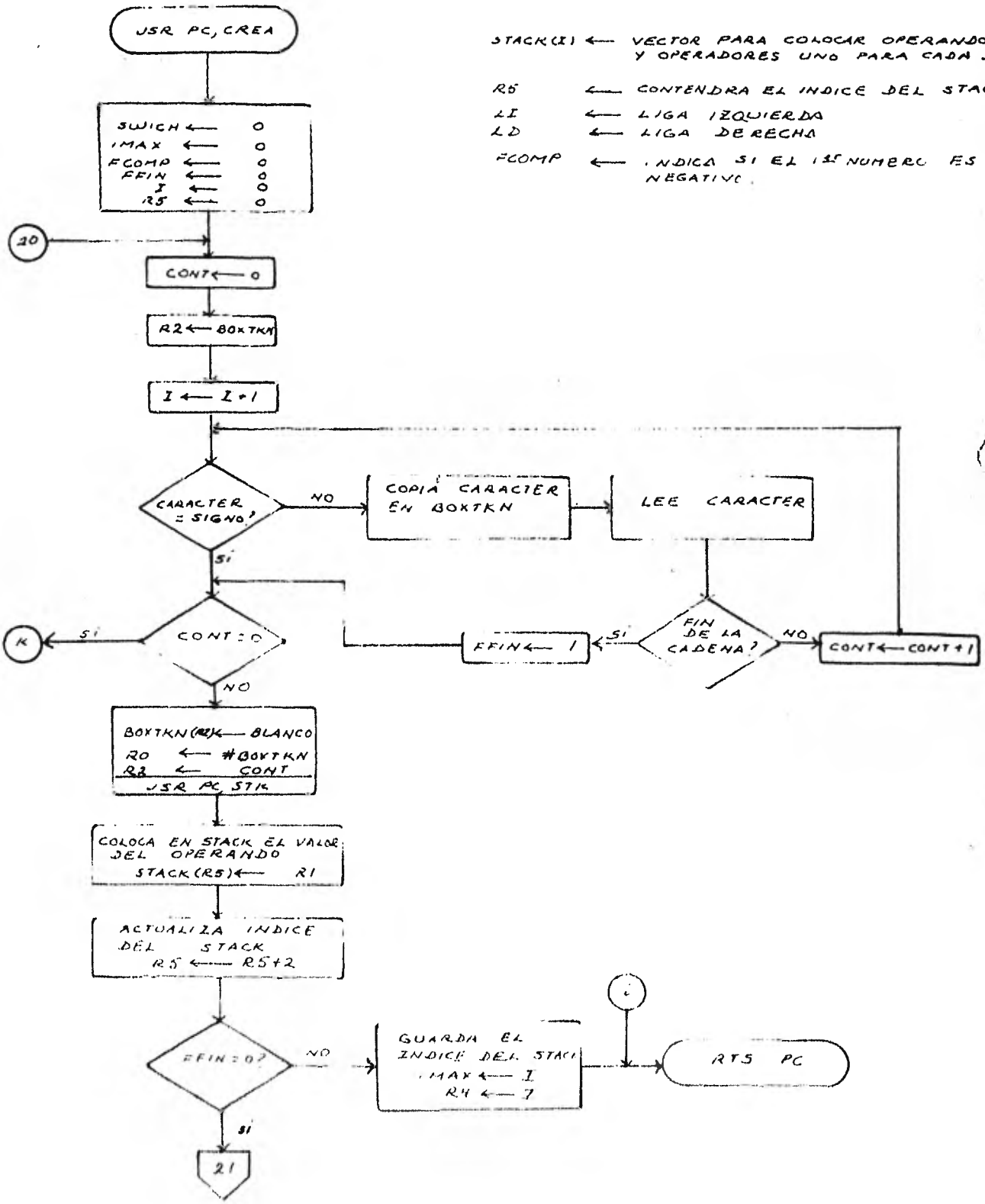

NBYTES ← LONGITUD DE LA INSTRUCCION
RH ← CONTADOR DE CARACTERES
NEXTKN ← VECTOR DONDE GUARDAMOS OPERANDO
BANSW ← BANDERA DE MODO DIRECTO Y EXTENDIDO
JSR AC,CALPLC
NBYTES ← 0
END
SI
C
NO
MACRO
SI
JSR AC,SBMAC
11
NO
MEND
SI
ERROR
JSR AC,VIOLA
11
NO
PSEUDO-INSTRUCCION
SI
JSR AC,SBPSEU
11
NO
MACRO-LLAMADA
SI
JSR AC,SBMNAM
11
NO
MODO RELATIVO
SI
NBYTES ← 2
NO
MODO RELATI IMPLICADO
SI
NBYTES ← 1
NO
12
13
HAY ETIQUETA
NO
SI
COLOCALA EN LA TABLA JUNTO CON EL VALOR DEL PC
JSR AC,HASETI
PLC ← PLC + NBYTES
11
RTS AC

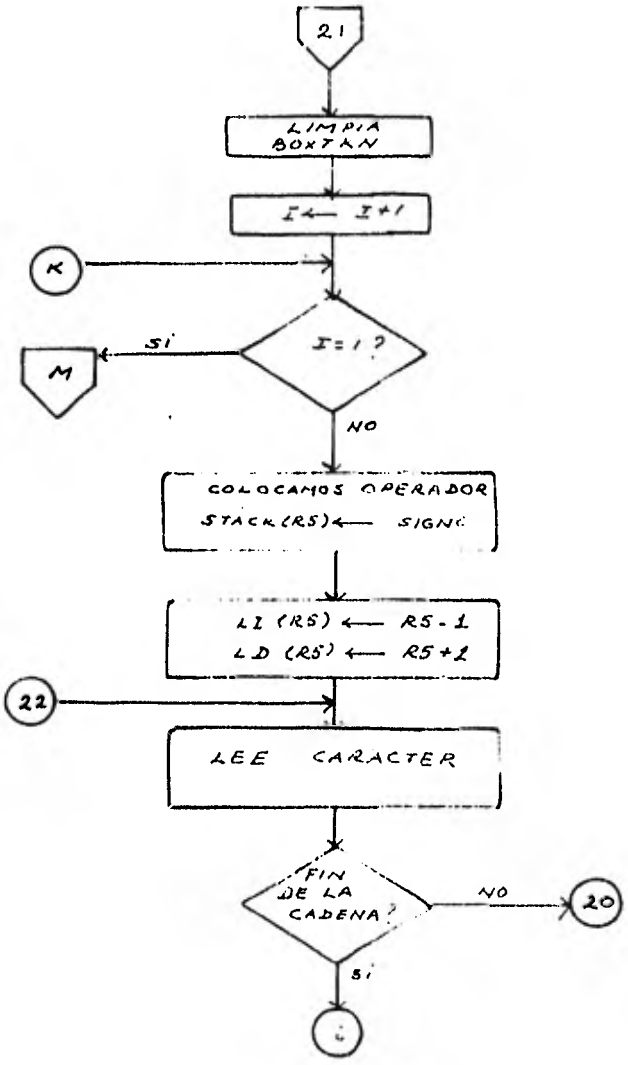

12
LEE CARACTER
CARAC - TER = "#"
SI
MODO INME- DIATO
EL OPERADOR ADMITE ESTE MODO ?
SI
BIT 15 ENCENDIDO?
NO
NBYTES ← 2
SI
NBYTES ← 3
13
NO
ERROR
11
R3 ← #NEXTKN
R4 ← 0
FIN DE LINEA
SI
NO
CARACTER = BLANCO O TAB ?
SI
NO
CARACTER = ','
SI
NO
COPIA CARACTER EN NEXTKN Y LEE SIGUIENTE CARACTER
R4 ← R4+1
14
R4 = 1 ?
SI
NO
NEXTKN = X ?
SI
NO
MODO INDEXADO
LEE CARACTER
CARACTER = BLANCO O TAB
SI
NO
CARACTER = ','
SI
NO
16
R4 ← 0
LIMPIA NEXTKN
15
EL OPERADOR ADMITE ESTE MODO ?
NO
ERROR
11
SI
NBYTES ← 2
13

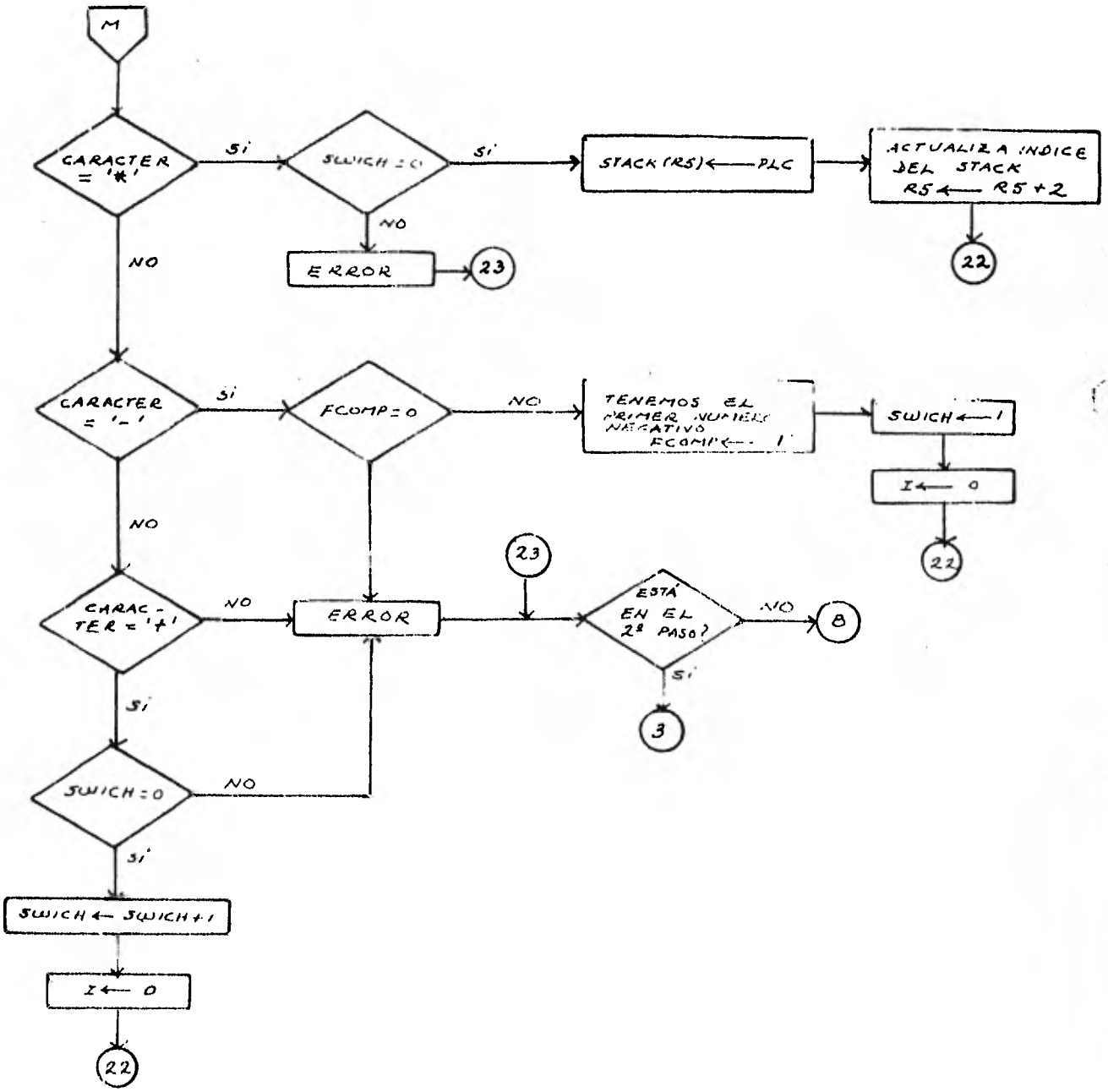

15
FIN DE LINEA?
SI
R4 ≠ 1
NO
14
SI
LEE CARACTER
ERROR
11
CARACTER = BLANCO Ó TAB
R4 ← R4+1
COPIA CARACTER A NEXTKN

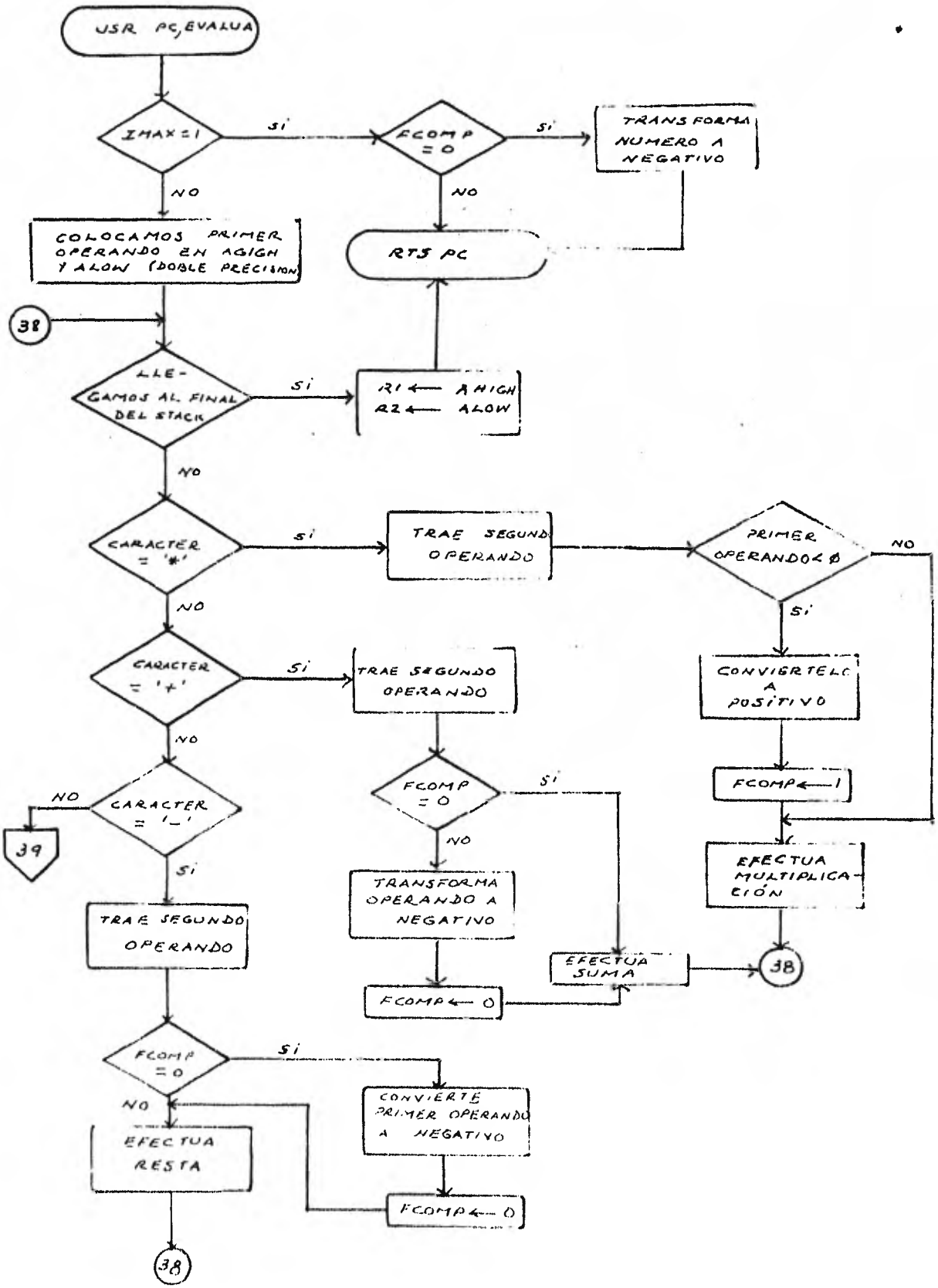

16
BANSW ← 1
EVALUA OPERANDO
VALOR OPERANDO ≤ 255
SI
MODO DIRECTO
MODO EXTENDIDO
NO
EL OPERADOR ADMITE ESTE MODO
SI
NBYTES ← 2
ERROR
NO
EL OPERADOR ADMITE ESTE MODO
11
SI
NBYTES ← 3
13

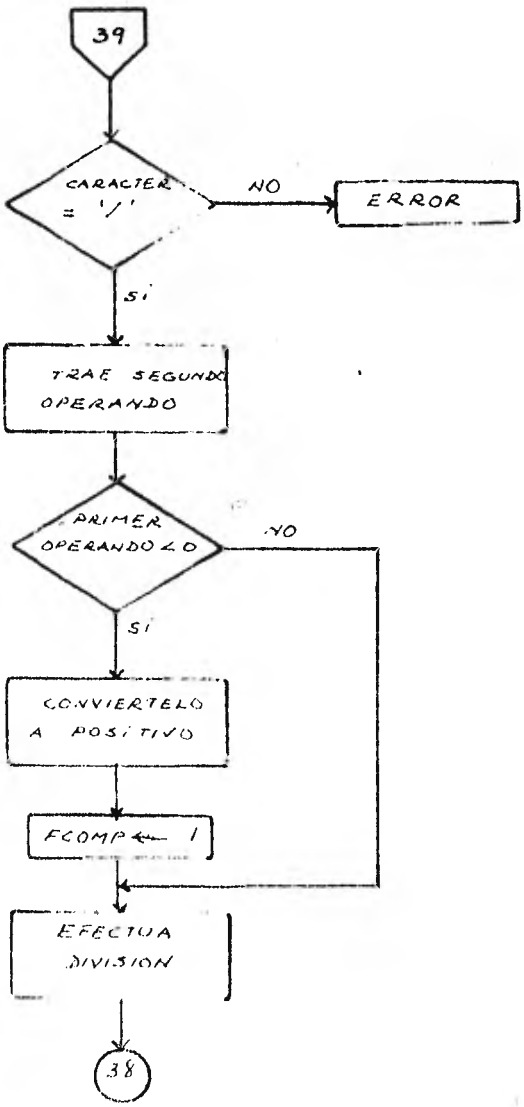

USR PC,COPIA
D
LIMPIA EL BUFFER NEWBUF
NEWBUF ← BUFF
COPIA CARACTERES DE BUFF A NEWBUF
ABRE ARCHIVO COPIA.TMP
ESCRIBE NEWBUF
CIERRA COPIA.TMP
.RTS PC

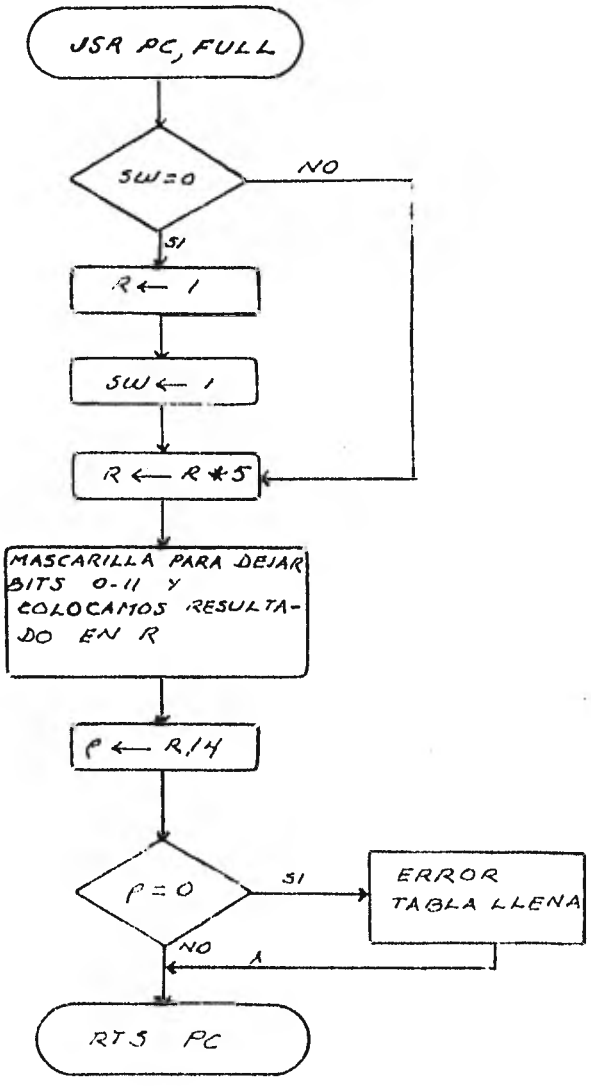

JSR PC, CREA
SWICH ← 0
IMAX ← 0
FCOMP ← 0
FFIN ← 0
I ← 0
R5 ← 0
20
CONT ← 0
R2 ← BOXTKN
I ← I + 1
CARACTER = SIGNO?
NO
COPIA CARACTER EN BOXTKN
LEE CARACTER
SI
K
SI
CONT = 0
NO
FFIN ← 1
SI
FIN DE LA CADENA?
NO
CONT ← CONT + 1
BOXTKN(R2) ← BLANCO
R0 ← #BOXTKN
R3 ← CONT
JSR PC, STK
COLOCA EN STACK EL VALOR DEL OPERANDO
STACK(R5) ← R1
ACTUALIZA INDICE DEL STACK
R5 ← R5+2
FFIN = 0?
NO
GUARDA EL INDICE DEL STACK
IMAX ← I
R4 ← 7
SI
21
RTS PC
STACK(I) ← VECTOR PARA COLOCAR OPERANDOS Y OPERADORES UNO PARA CADA I
R5 ← CONTENDRA EL INDICE DEL STAC
LI ← LIGA IZQUIERDA
LD ← LIGA DERECHA
FCOMP ← INDICA SI EL 1ER NUMERO ES NEGATIVO

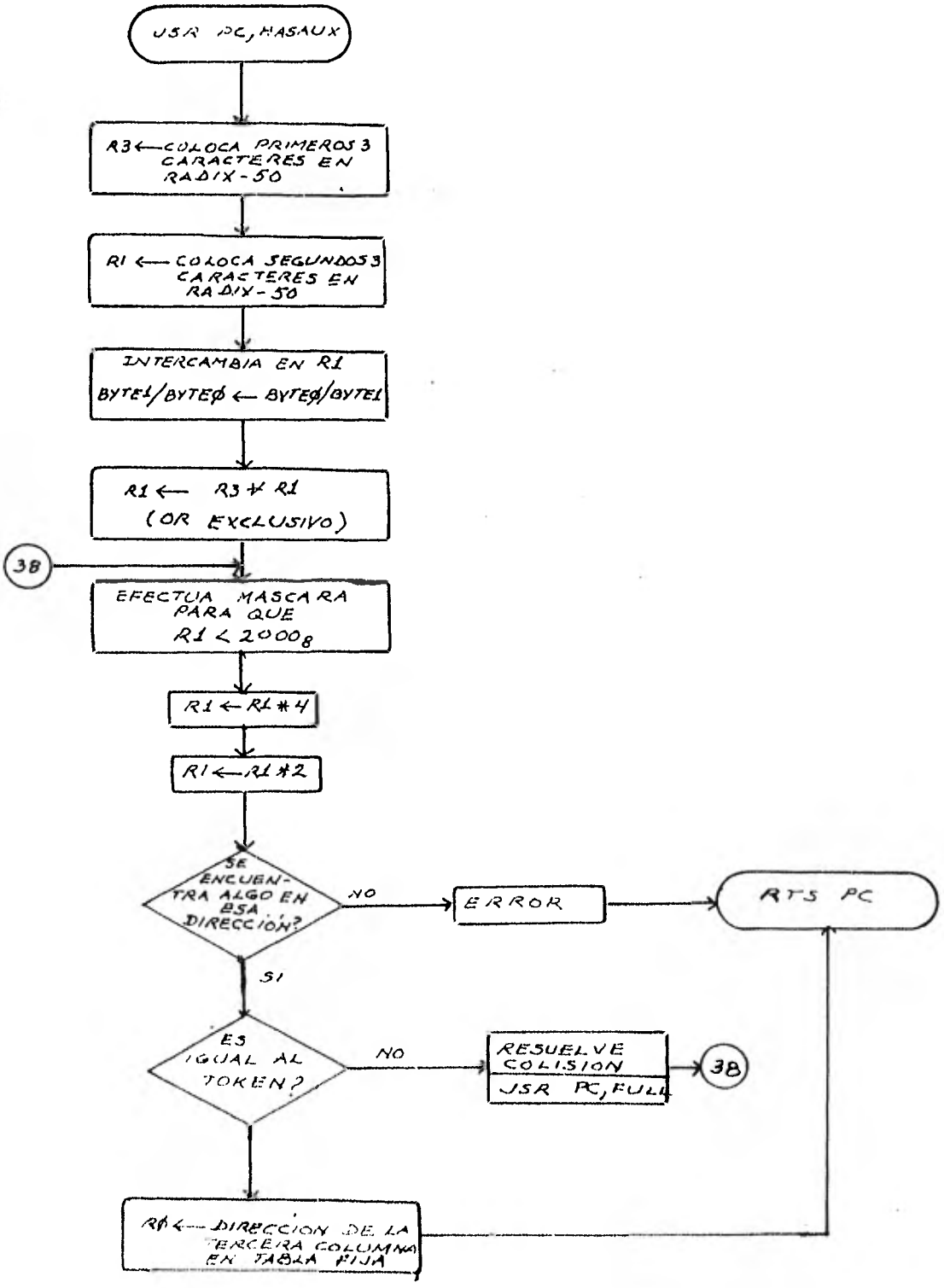

21
LIMPIA BOXTAN
I ← I+1
K
I=1 ?
SI
M
NO
COLOCAMOS OPERADOR
STACK(RS) ← SIGNO
LI (RS) ← RS-1
LD (RS) ← RS+1
22
LEE CARACTER
FIN DE LA CADENA ?
NO
20
SI
4

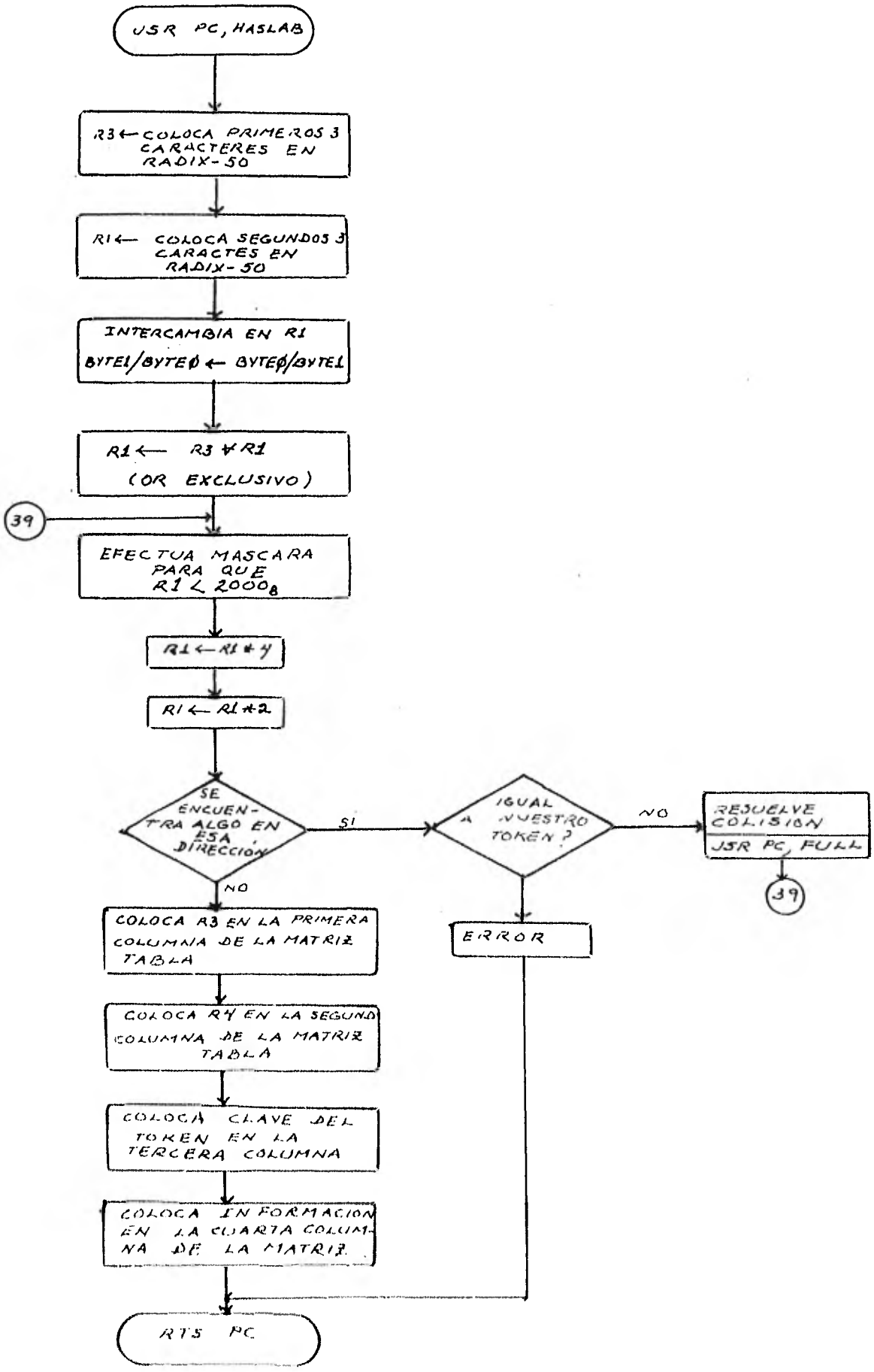

M
CARACTER = '*'
SI
SWICH = 0
SI
STACK (R5) ← PLC
ACTUALIZA INDICE DEL STACK R5 ← R5 + 2
22
NO
ERROR
23
NO
CARACTER = '-'
SI
FCOMP = 0
NO
TENEMOS EL PRIMER NUMERO NEGATIVO FCOMP ← 1
SWICH ← 1
I ← 0
22
NO
CARAC - TER = '+'
NO
ERROR
23
ESTÁ EN EL 2º PASO?
NO
8
SI
3
SI
SWICH = 0
NO
SI
SWICH ← SWICH + 1
I ← 0
22

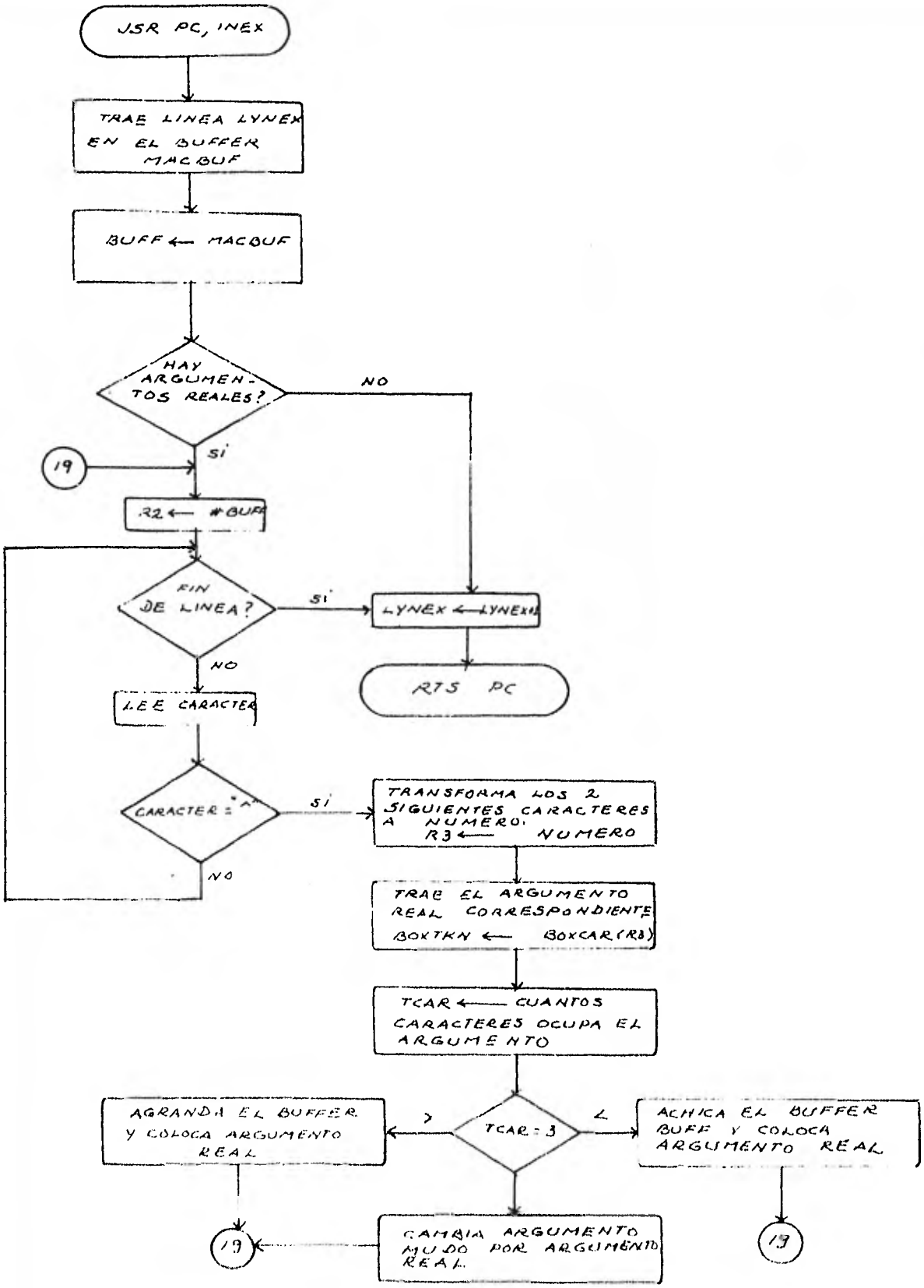

USR PC,EVALUA
IMAX=1
SI
FCOMP =0
SI
TRANSFORMA NUMERO A NEGATIVO
NO
NO
COLOCAMOS PRIMER OPERANDO EN AGIGH Y ALOW (DOBLE PRECISION
RTS PC
38
LLE-GAMOS AL FINAL DEL STACK
SI
R1 ← AHIGH
R2 ← ALOW
NO
CARACTER = '*'
SI
TRAE SEGUNDO OPERANDO
PRIMER OPERANDO< Ø
NO
SI
NO
CARACTER = '+'
SI
TRAE SEGUNDO OPERANDO
CONVIGRTELO A POSITIVO
NO
FCOMP =0
SI
FCOMP ← 1
NO
CARACTER = '-'
NO
39
SI
TRANSFORMA OPERANDO A NEGATIVO
EFECTUA MULTIPLICAEIÓN
TRAE SEGUNDO OPERANDO
EFECTUA SUMA
38
FCOMP ← 0
FCOMP =0
SI
CONVIERTE PRIMER OPERANDO A NEGATIVO
NO
EFECTUA RESTA
FCOMP ← 0
38

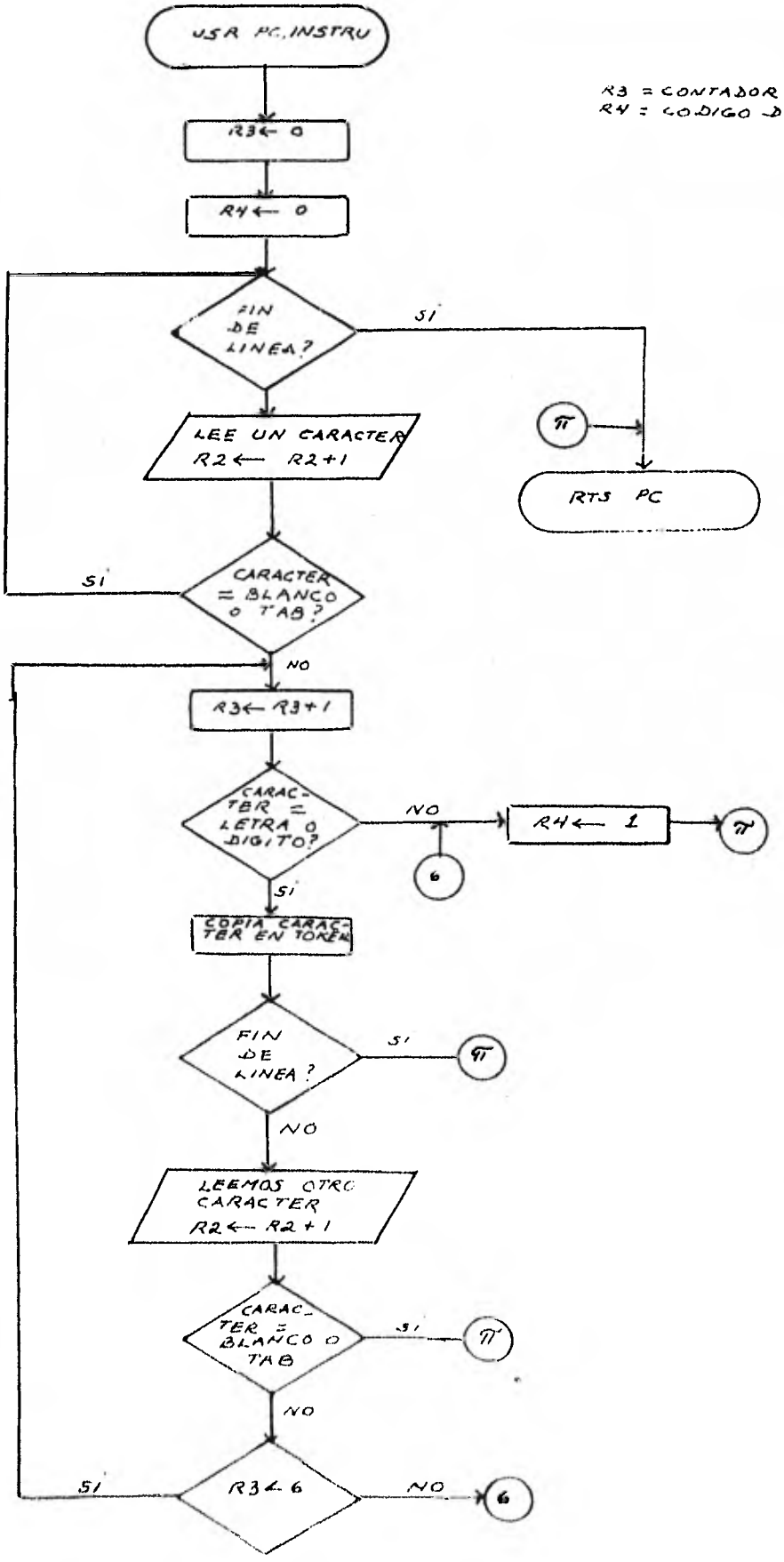

39
CARACTER = '/'
NO
ERROR
SI
TRAE SEGUNDO OPERANDO
PRIMER OPERANDO < 0
NO
SI
CONVIERTELO A POSITIVO
FCOMP ← 1
EFECTUA DIVISION
38

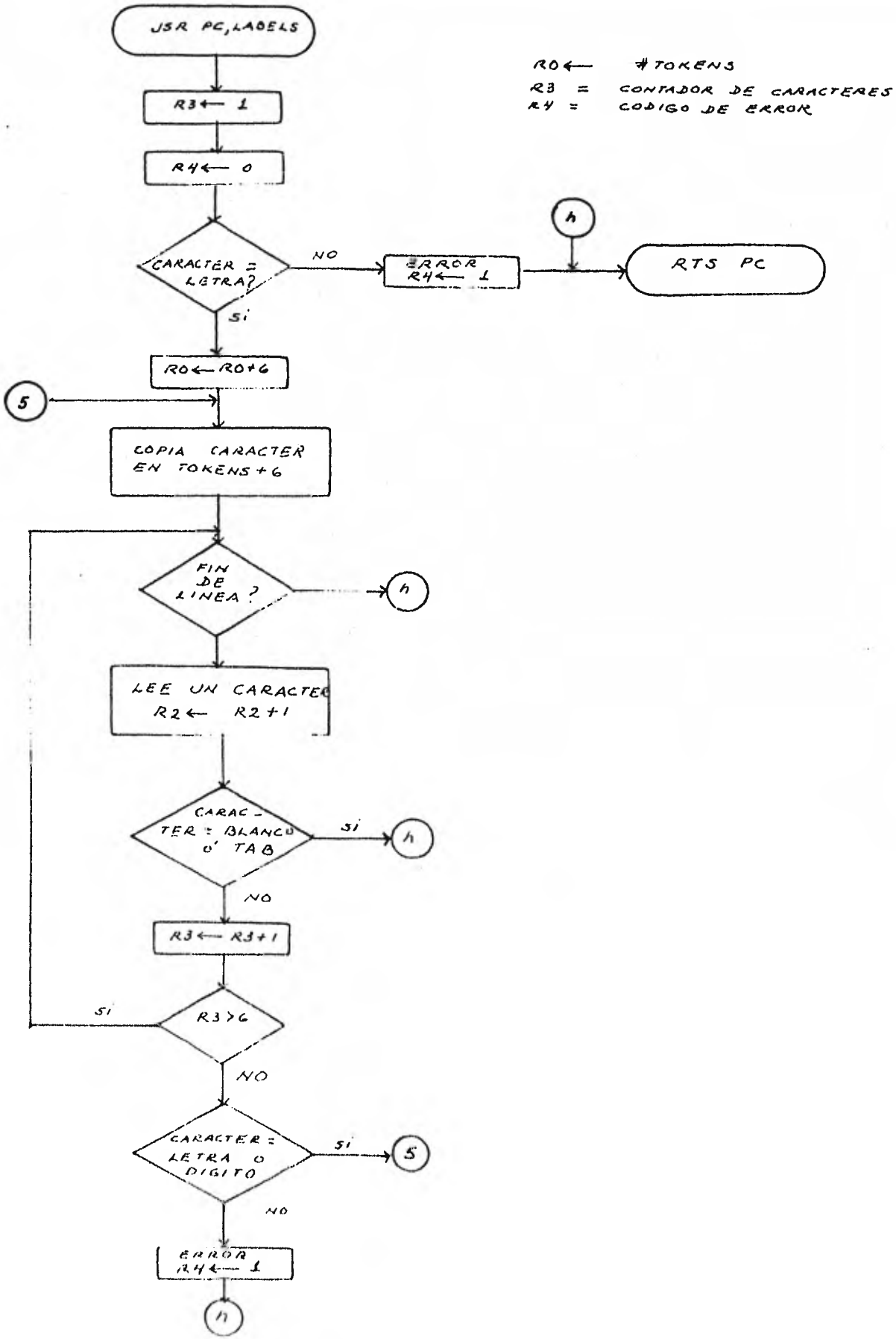

JSR PC, FULL
SW=0
NO
SI
R ← 1
SW ← 1
R ← R*5
MASCARILLA PARA DEJAR BITS 0-11 Y COLOCAMOS RESULTADO EN R
P ← R,14
P=0
SI
ERROR TABLA LLENA
NO
RTS PC

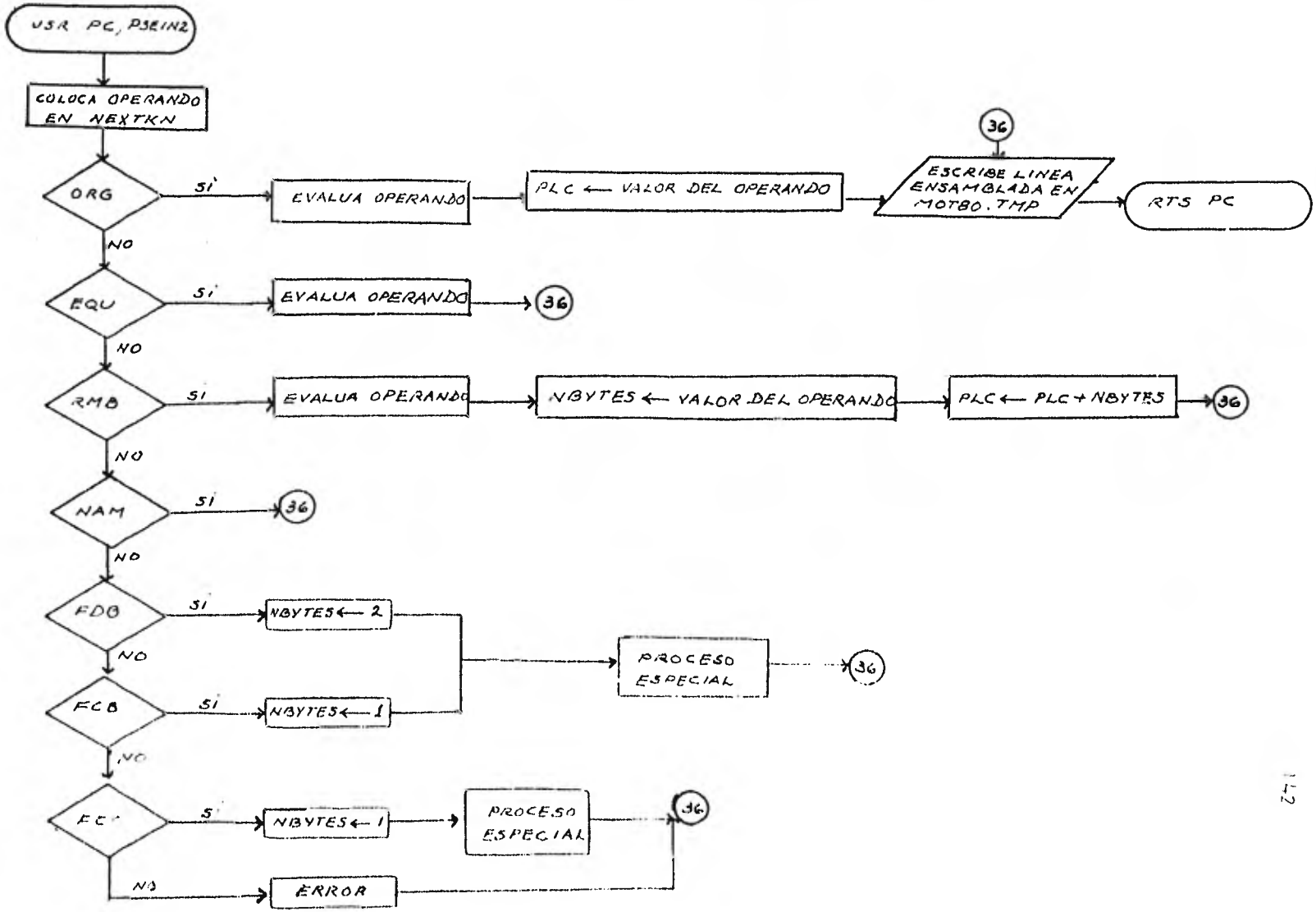

JSR PC,HASAUX
R3 ← COLOCA PRIMEROS 3 CARACTERES EN RADIX-50
R1 ← COLOCA SEGUNDOS 3 CARACTERES EN RADIX-50
INTERCAMBIA EN R1
BYTE1/BYTE0 ← BYTE0/BYTE1
R1 ← R3 ∀ R1
(OR EXCLUSIVO)
38
EFECTUA MASCARA PARA QUE
R1 < 2000₈
R1 ← R1 * 4
R1 ← R1 * 2
SE ENCUENTRA ALGO EN ESA DIRECCIÓN?
NO
ERROR
RTS PC
SI
ES IGUAL AL TOKEN?
NO
RESUELVE COLISION
JSR PC,FULL
38
R0 ← DIRECCION DE LA TERCERA COLUMNA EN TABLA FIJA

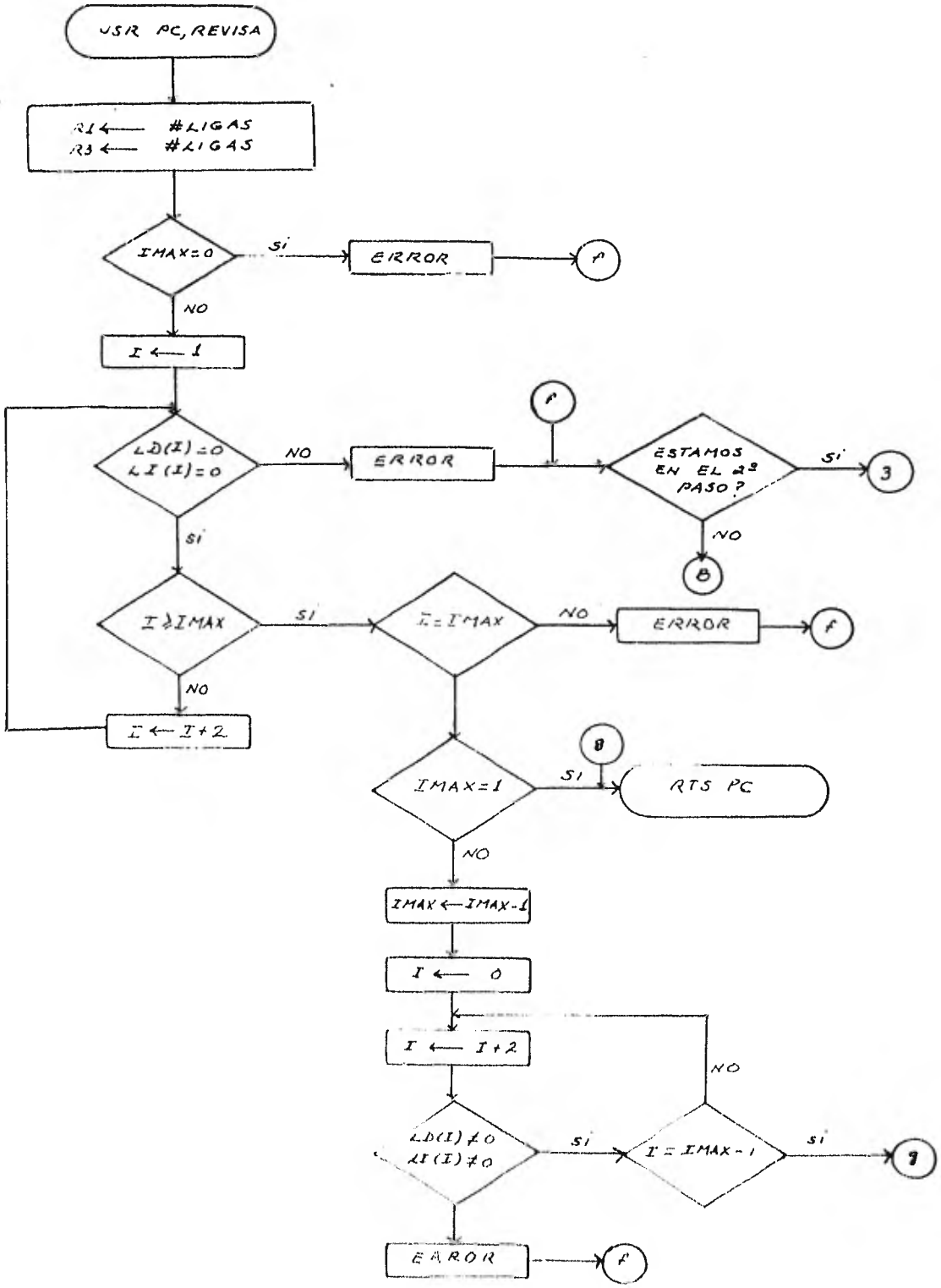

JSR PC, HASLAB
R3 ← COLOCA PRIMEROS 3 CARACTERES EN RADIX-50
R1 ← COLOCA SEGUNDOS 3 CARACTES EN RADIX-50
INTERCAMBIA EN R1
BYTE1/BYTE0 ← BYTE0/BYTE1
R1 ← R3 ∀ R1
(OR EXCLUSIVO)
39
EFECTUA MASCARA PARA QUE R1 < 2000₈
R1 ← R1 + 4
R1 ← R1 + 2
SE ENCUENTRA ALGO EN ESA DIRECCION
SI
IGUAL A NUESTRO TOKEN ?
NO
RESUELVE COLISION
JSR PC, FULL
39
NO
COLOCA R3 EN LA PRIMERA COLUMNA DE LA MATRIZ TABLA
ERROR
COLOCA R4 EN LA SEGUNDA COLUMNA DE LA MATRIZ TABLA
COLOCA CLAVE DEL TOKEN EN LA TERCERA COLUMNA
COLOCA INFORMACION EN LA CUARTA COLUMNA DE LA MATRIZ
RTS PC

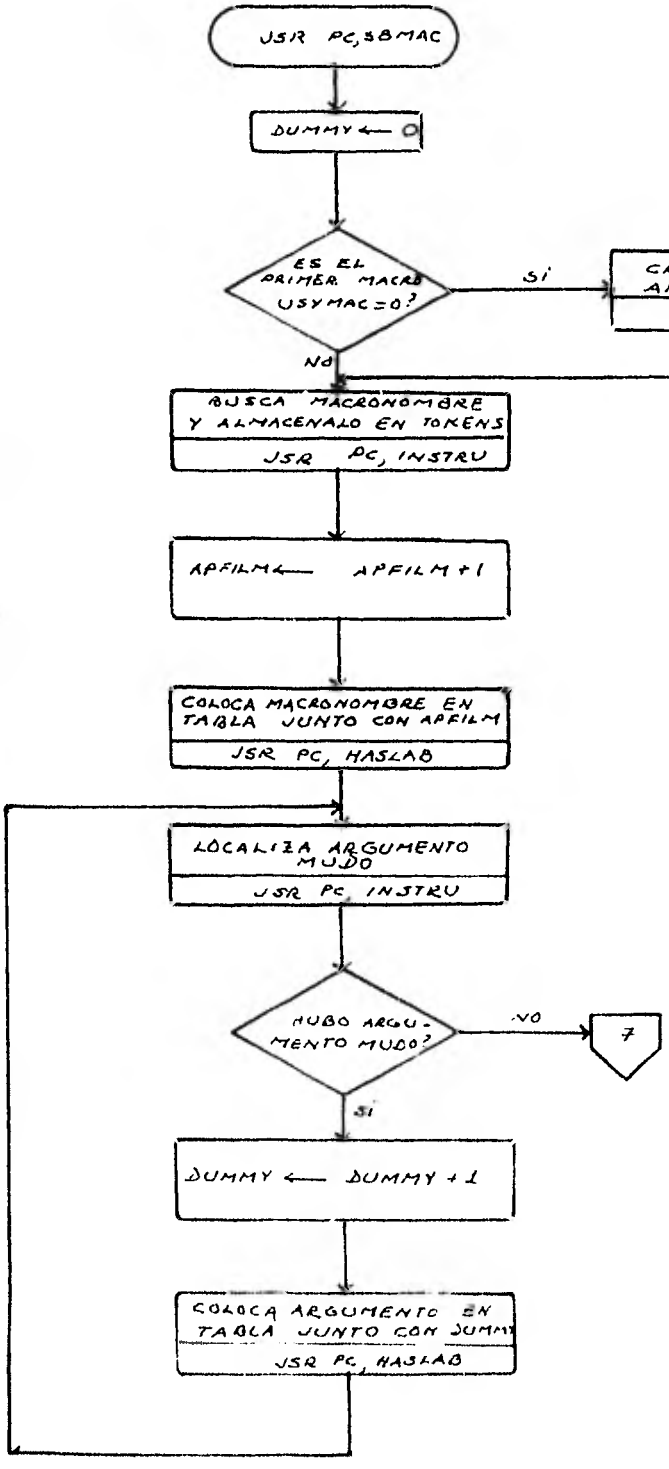

JSR PC, INEX
TRAE LINEA LYNEX EN EL BUFFER MACBUF
BUFF ← MACBUF
HAY ARGUMEN-TOS REALES?
NO
SI
19
R2 ← #BUF
FIN DE LINEA?
SI
LYNEX ← LYNEX+1
R7 ← PC
NO
LEE CARACTER
CARACTER = "A"
SI
TRANSFORMA LOS 2 SIGUIENTES CARACTERES A NUMERO: R3 ← NUMERO
NO
TRAE EL ARGUMENTO REAL CORRESPONDIENTE
BOXTKN ← BOXCAR(R8)
TCAR ← CUANTOS CARACTERES OCUPA EL ARGUMENTO
AGRANDA EL BUFFER Y COLOCA ARGUMENTO REAL
>
TCAR = 3
<
ACHICA EL BUFFER BUFF Y COLOCA ARGUMENTO REAL
19
CAMBIA ARGUMENTO MUDO POR ARGUMENTO REAL
19


USR PC,INSTRU
R3 = CONTADOR DE CARACTERES
R4 = CODIGO DE ERROR
R3← 0
R4← 0
FIN DE LINEA?
SI
LEE UN CARACTER
R2← R2+1
RTS PC
CARACTER = BLANCO O TAB?
SI
NO
R3← R3+1
CARAC- TER = LETRA O DIGITO?
NO
R4← 1
SI
COPIA CARAC- TER EN TOKEN
FIN DE LINEA?
SI
NO
LEEMOS OTRO CARACTER
R2← R2+1
CARAC- TER = BLANCO O TAB
SI
NO
R3 ≤ 6
SI
NO


JSR PC,LABELS
R3 ← 1
R4 ← 0
CARACTER = LETRA?
NO
ERROR R4 ← 1
SI
RTS PC
R0 ← R0+6
COPIA CARACTER EN TOKENS+6
FIN DE LINEA ?
LEE UN CARACTER R2 ← R2+1
CARACTER = BLANCO O TAB
SI
NO
R3 ← R3+1
R3 > 6
SI
NO
CARACTER = LETRA O DIGITO
SI
NO
ERROR R4 ← 1
R0 ← # TOKENS
R3 = CONTADOR DE CARACTERES
R4 = CODIGO DE ERROR

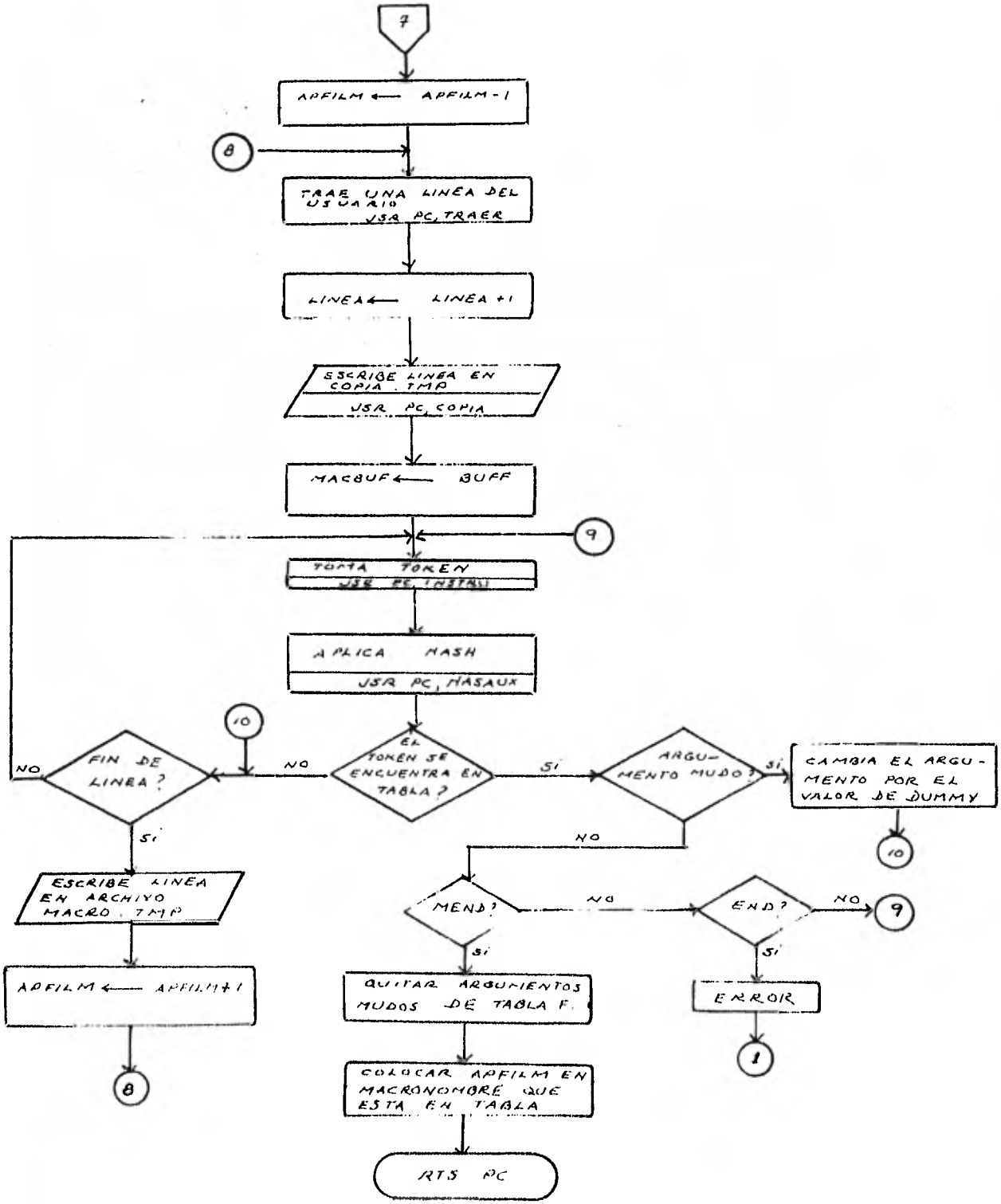

USR PC, PSEINZ
COLOCA OPERANDO EN NEXTKN
ORG
SI
EVALUA OPERANDO
PLC ← VALOR DEL OPERANDO
36
ESCRIBE LINEA ENSAMBLADA EN MOTBO.TMP
RTS PC
NO
EQU
SI
EVALUA OPERANDO
36
NO
RMB
SI
EVALUA OPERANDO
NBYTES ← VALOR DEL OPERANDO
PLC ← PLC + NBYTES
36
NO
NAM
SI
36
NO
FDB
SI
NBYTES ← 2
NO
FCB
SI
NBYTES ← 1
PROCESO ESPECIAL
36
NO
FCC
SI
NBYTES ← 1
PROCESO ESPECIAL
36
NO
ERROR


USR PC, REVISA
R1 ← #LIGAS
R3 ← #LIGAS
IMAX=0
SI
ERROR
NO
I ← 1
LD(I)=0
LI(I)=0
NO
ERROR
ESTAMOS EN EL 2º PASO?
SI
3
NO
SI
I≥IMAX
SI
I=IMAX
NO
ERROR
NO
I ← I+2
IMAX=1
SI
RTS PC
NO
IMAX ← IMAX-1
I ← 0
I ← I+2
NO
LD(I)≠0
LI(I)≠0
SI
I=IMAX-1
SI
EARROR



APFILM ← CONTADOR DE LINEAS DEL ARCHIVO MACRO.TMP

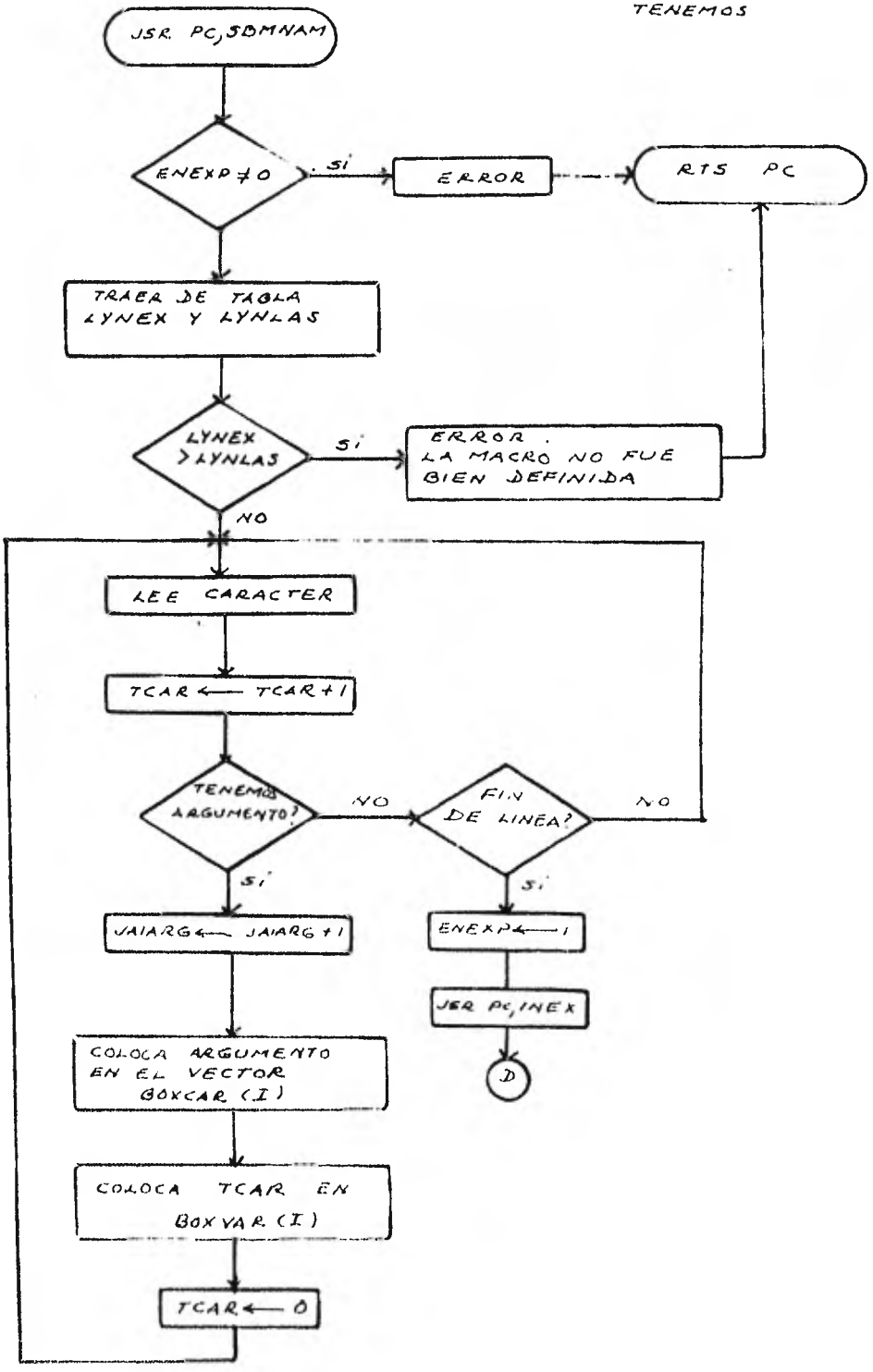
DUMMY ← CONTADOR DE ARGUMENTOS MUDOS

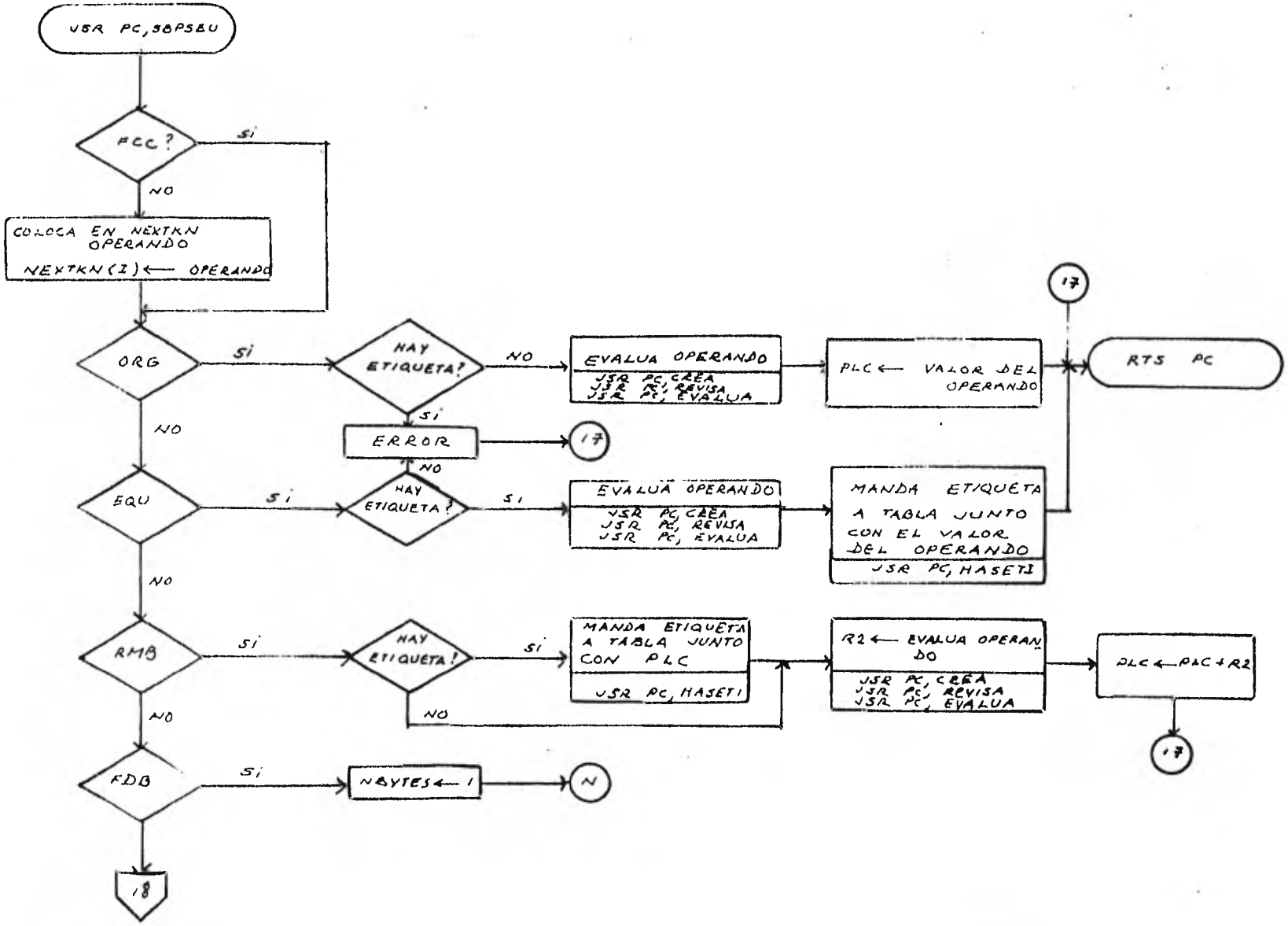

7
APFILM ← APFILM - 1
8
TRAE UNA LINEA DEL USUARIO
JSR PC, TRAER
LINEA ← LINEA + 1
ESCRIBE LINEA EN COPIA . TMP
JSR PC, COPIA
MACBUF ← BUFF
9
TOMA TOKEN
JSR PC, INSTALL
APLICA HASH
JSR PC, HASAUX
10
NO
FIN DE LINEA ?
NO
EL TOKEN SE ENCUENTRA EN TABLA ?
SI
ARGUMENTO MUDO ?
SI
CAMBIA EL ARGUMENTO POR EL VALOR DE DUMMY
10
SI
ESCRIBE LINEA EN ARCHIVO MACRO . TMP
APFILM ← APFILM + 1
8
NO
MEND ?
NO
END ?
NO
9
SI
QUITAR ARGUMENTOS MUDOS DE TABLA F.
COLOCAR APFILM EN MACRONOMBRE QUE ESTA EN TABLA
RTS PC
SI
ERROR
1

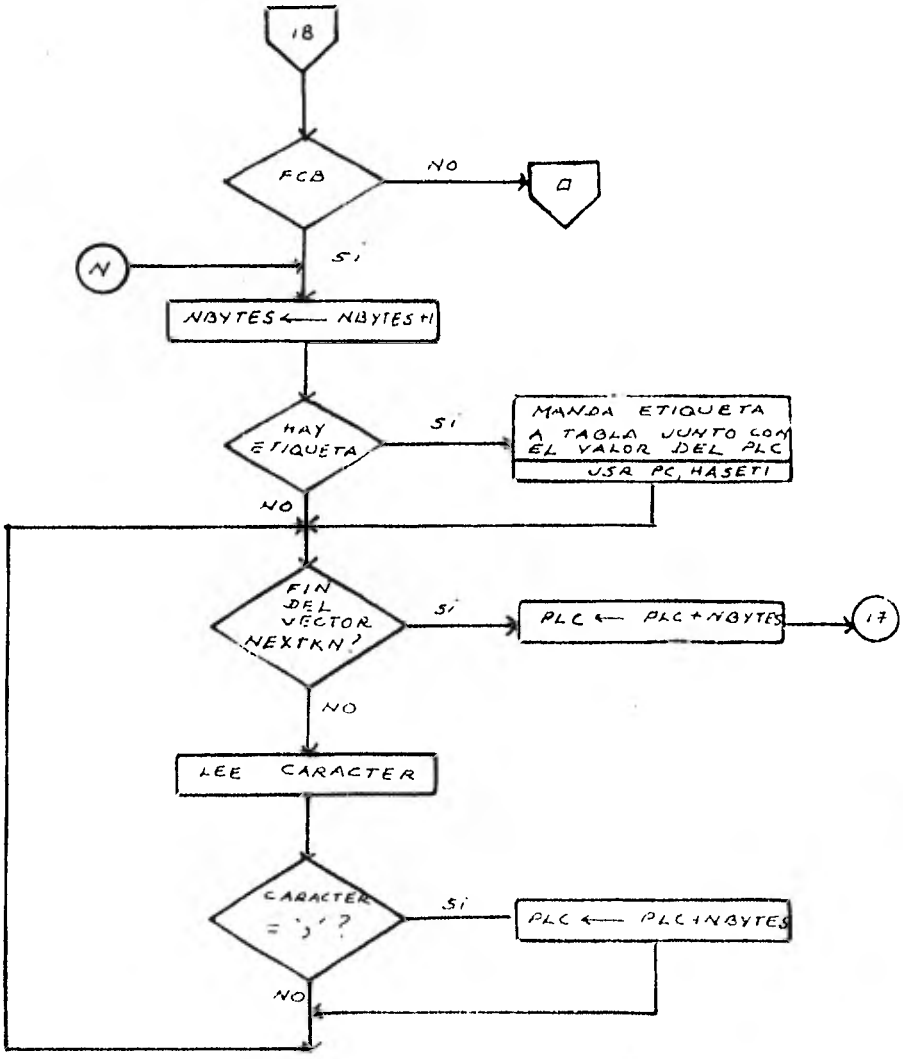
LYNEXP ← PRIMERA LINEA A EXPANDIR QUE SE LOCALIZA EN EL ARCHIVO MACRO.TMP

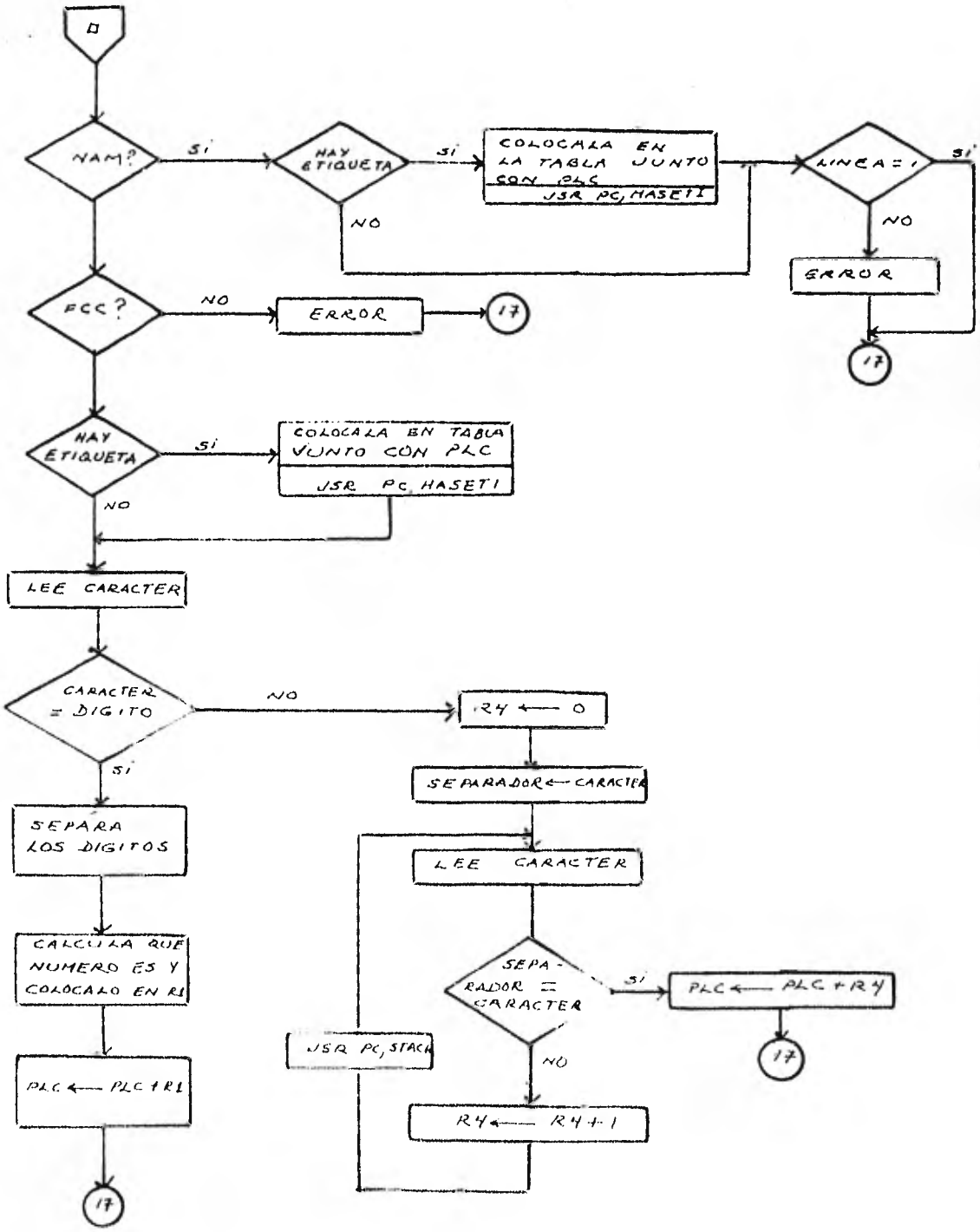
LYNLAS ← ULTIMA LINEA A EXPANDIR DEL ARCHIVO MACRO.TMP

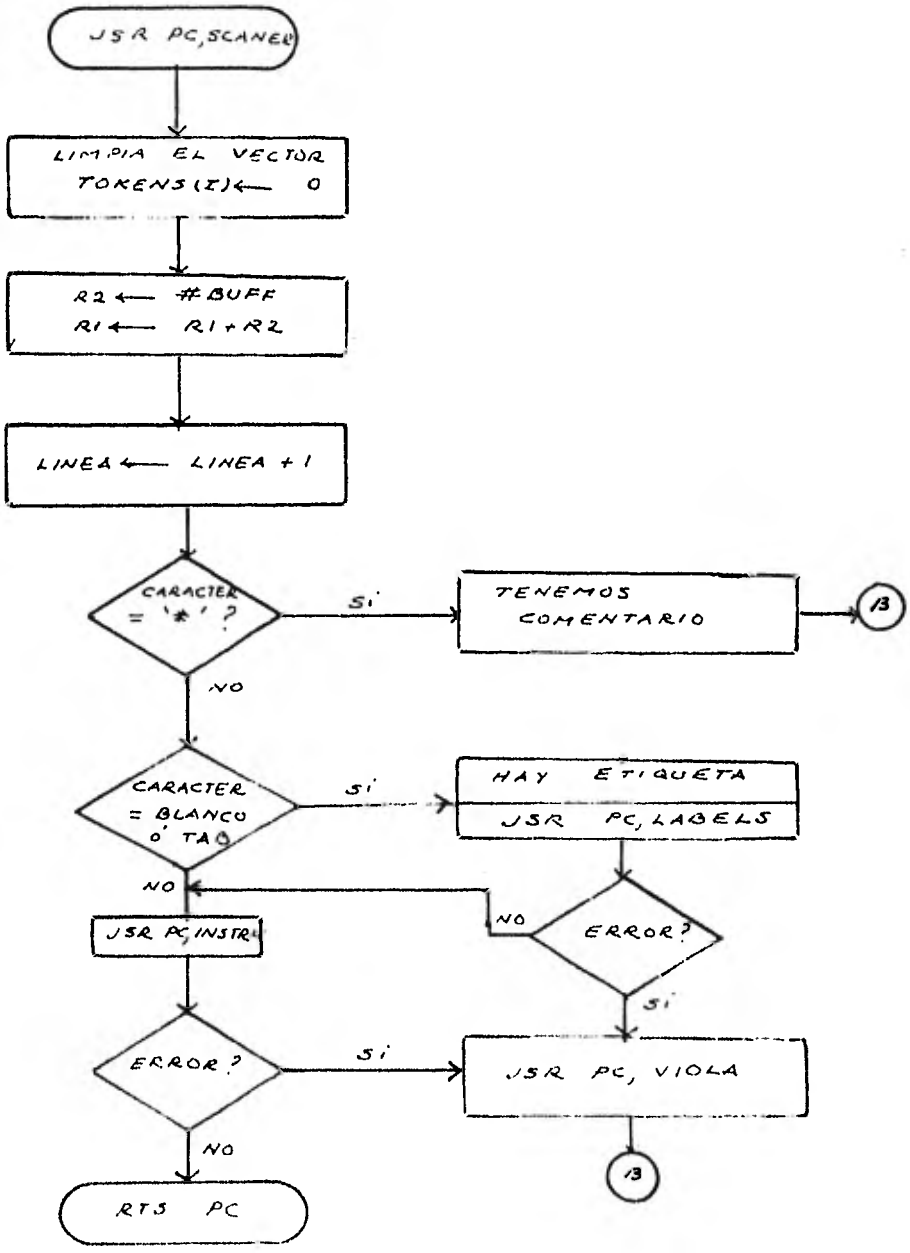
JAIARG ← CUANTOS ARGUMENTOS REALES TENEMOS

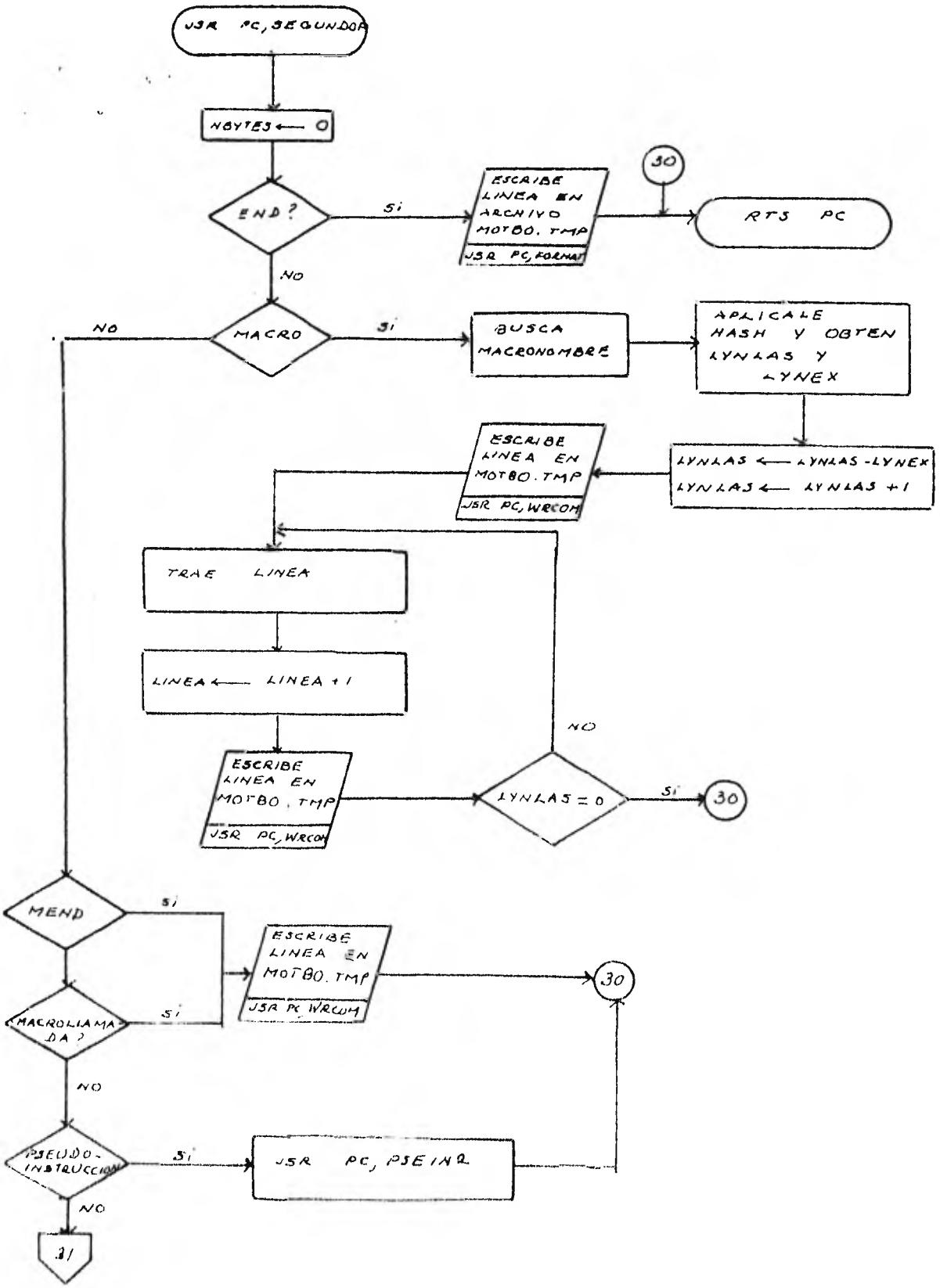


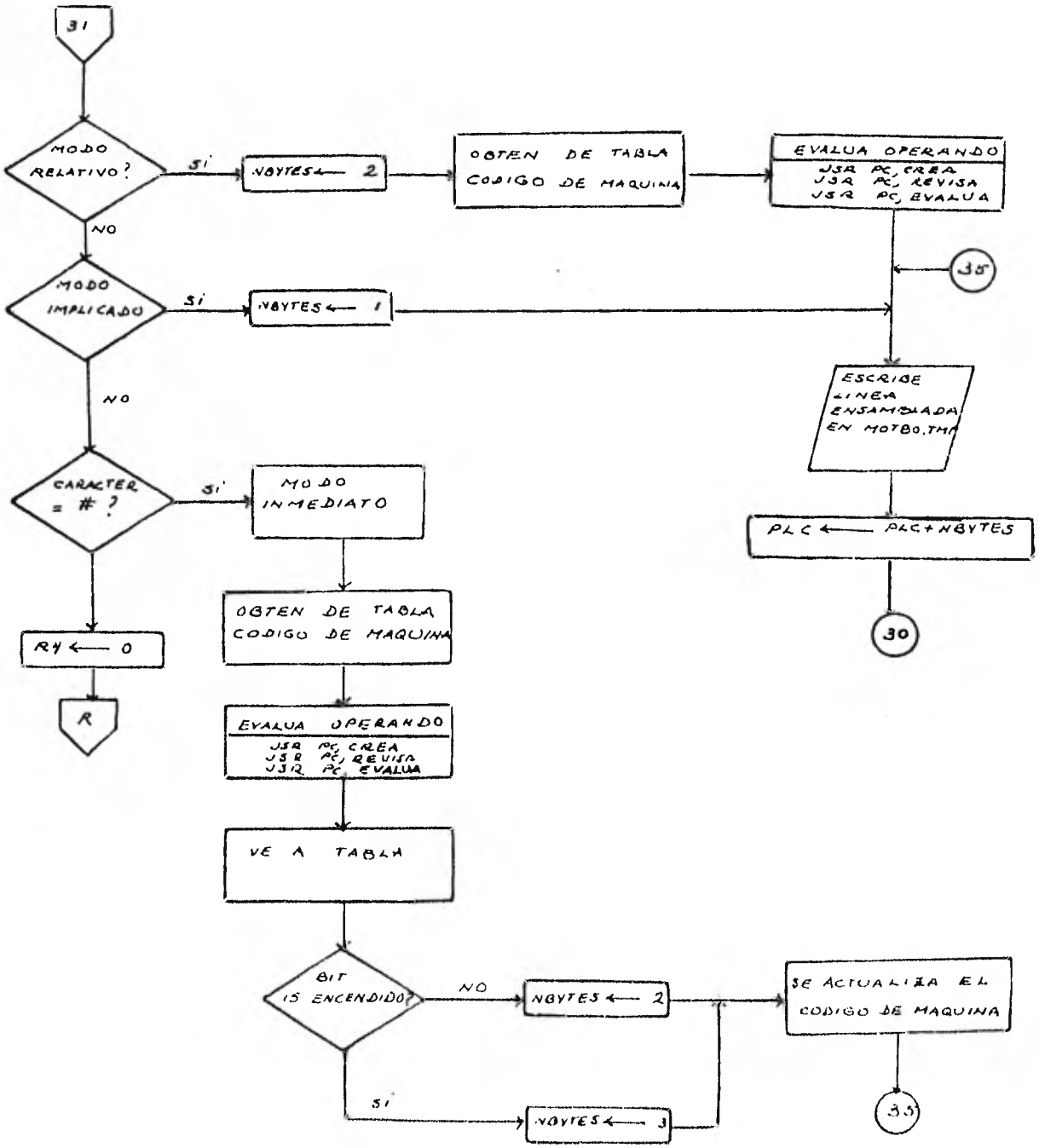

JSR PC,SOPSBU
PCC ?
SI
NO
COLOCA EN NEXTKN OPERANDO
NEXTKN(I) ← OPERANDO
ORG
SI
HAY ETIQUETA?
NO
EVALUA OPERANDO
JSR PC, CREA
JSR PC, REVISA
JSR PC, EVALUA
PLC ← VALOR DEL OPERANDO
17
RTS PC
SI
ERROR
17
NO
NO
EQU
SI
HAY ETIQUETA?
SI
EVALUA OPERANDO
JSR PC, CREA
JSR PC, REVISA
JSR PC, EVALUA
MANDA ETIQUETA A TABLA JUNTO CON EL VALOR DEL OPERANDO
JSR PC, HASETI
NO
RMB
SI
HAY ETIQUETA?
SI
MANDA ETIQUETA A TABLA JUNTO CON PLC
JSR PC, HASETI
R2 ← EVALUA OPERANDO
JSR PC, CREA
JSR PC, REVISA
JSR PC, EVALUA
PLC ← PLC + R2
17
NO
NO
FDB
SI
NBYTES ← 1
N
18

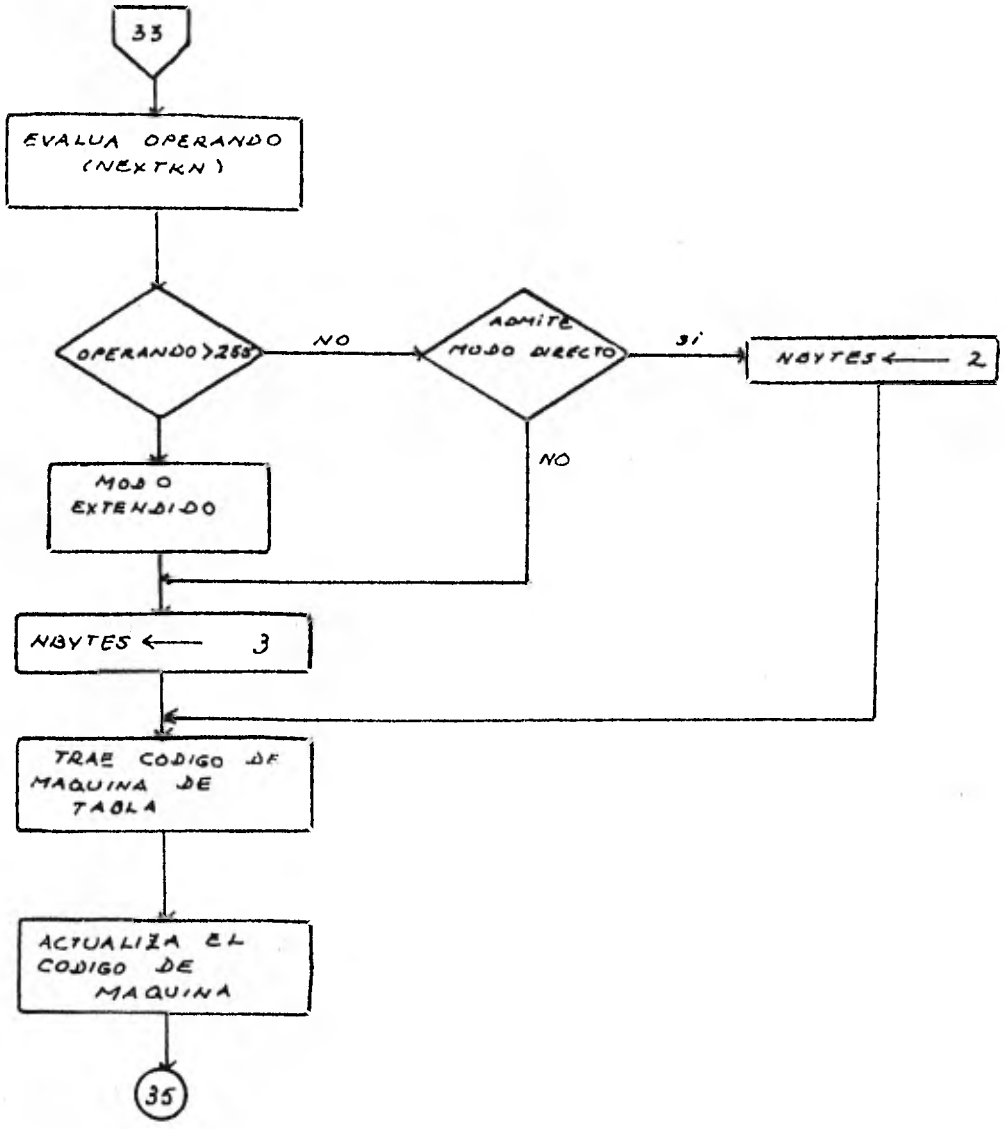

18
FCB
NO
D
N
SI
NBYTES ← NBYTES+1
HAY ETIQUETA
SI
MANDA ETIQUETA A TABLA JUNTO CON EL VALOR DEL PLC
JSR PC, HASETI
NO
FIN DEL VECTOR NEXTKN?
SI
PLC ← PLC+NBYTES
17
NO
LEE CARACTER
CARACTER = ';' ?
SI
PLC ← PLC+NBYTES
NO

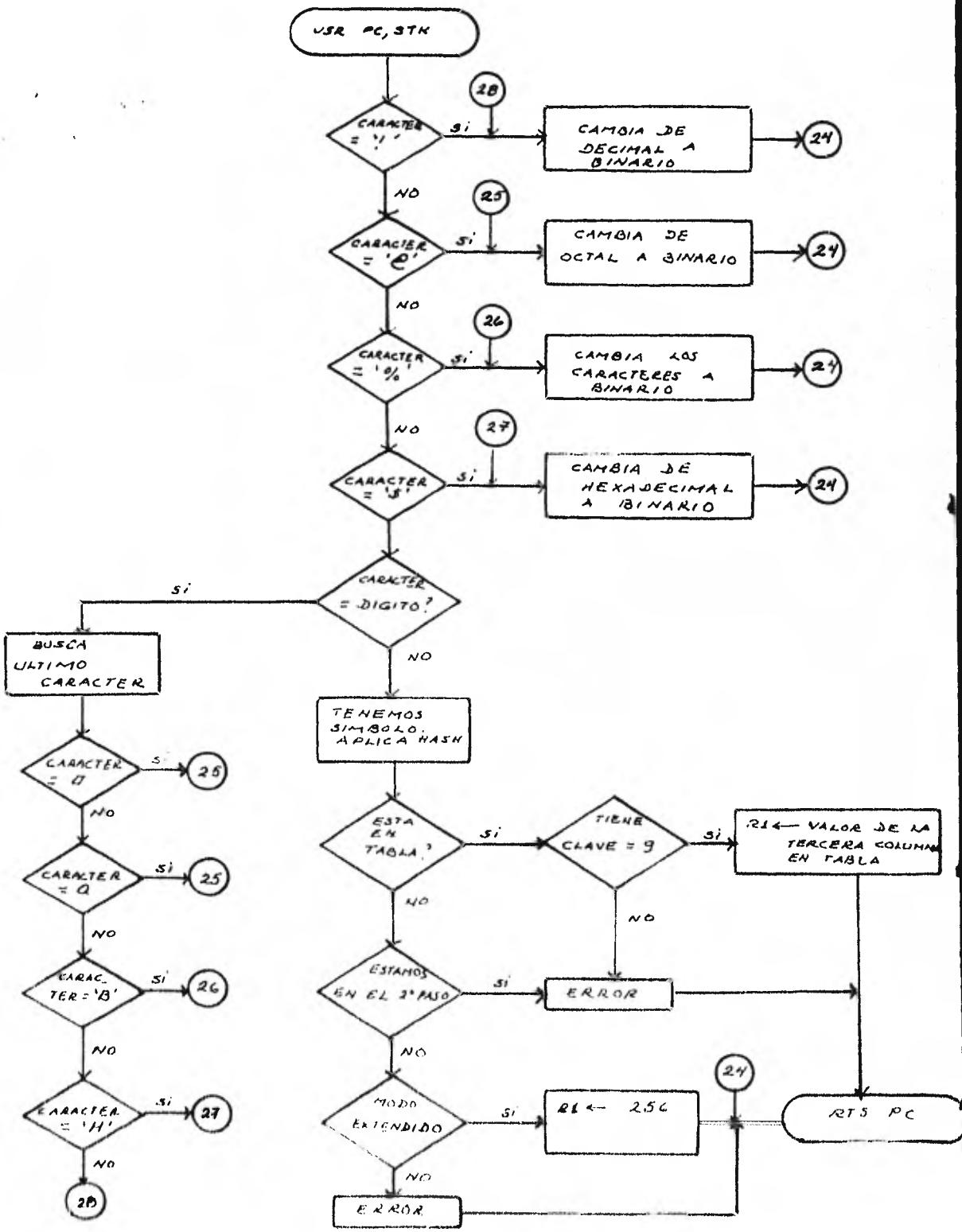

0
NAM?
SI
HAY ETIQUETA
SI
COLOCALA EN LA TABLA JUNTO CON PLC
JSR PC, HASETI
LINEA = 1
SI
NO
ERROR
17
NO
FCC?
NO
ERROR
17
HAY ETIQUETA
SI
COLOCALA EN TABLA JUNTO CON PLC
JSR PC, HASETI
NO
LEE CARACTER
CARACTER = DIGITO
NO
SI
R4 ← 0
SEPARADOR ← CARACTER
SEPARA LOS DIGITOS
LEE CARACTER
CALCULA QUE NUMERO ES Y COLOCALO EN R1
SEPARADOR = CARACTER
SI
PLC ← PLC + R4
17
JSR PC, STACK
NO
PLC ← PLC + R1
R4 ← R4 + 1
17

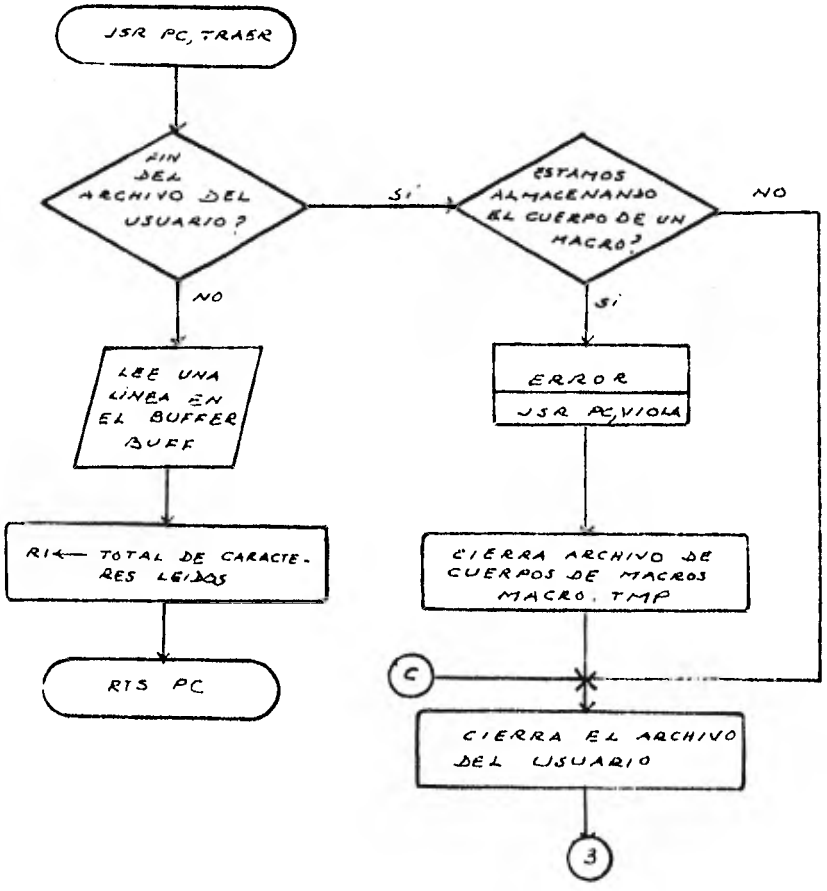

JSR PC,SCANER
LIMPIA EL VECTOR
TOKENS(I)← 0
R2 ← #BUFF
R1 ← R1+R2
LINEA ← LINEA + 1
CARACTER = '*' ?
SI
TENEMOS COMENTARIO
13
NO
CARACTER = BLANCO O' TAB
SI
HAY ETIQUETA
JSR PC, LABELS
NO
JSR PC,INSTR
NO
ERROR?
SI
ERROR?
SI
JSR PC, VIOLA
NO
13
RTS PC


JSR PC, SEGUNDOP
NBYTES ← 0
END ?
SI
ESCRIBE LINEA EN ARCHIVO MOTBO.TMP
JSR PC, FORMAT
30
RTS PC
NO
MACRO
NO
SI
BUSCA MACRONOMBRE
APLICALE HASH Y OBTEN LYNLAS Y LYNEX
LYNLAS ← LYNLAS - LYNEX
LYNLAS ← LYNLAS + 1
ESCRIBE LINEA EN MOTBO.TMP
JSR PC, WRCOM
TRAE LINEA
LINEA ← LINEA + 1
ESCRIBE LINEA EN MOTBO.TMP
JSR PC, WRCOM
LYNLAS = 0
NO
SI
30
MEND
SI
ESCRIBE LINEA EN MOTBO.TMP
JSR PC, WRCOM
30
MACROLLAMADA ?
SI
NO
PSEUDO-INSTRUCCION
SI
JSR PC, PSEINS
NO
31


31
MODO RELATIVO?
SI
NBYTES ← 2
OBTEN DE TABLA CODIGO DE MAQUINA
EVALUA OPERANDO
JSR PC, CREA
JSR PC, REVISA
JSR PC, EVALUA
NO
MODO IMPLICADO
SI
NBYTES ← 1
35
ESCRIBE LINEA ENSAMBLADA EN MOTBO.TMP
NO
CARACTER = # ?
SI
MODO INMEDIATO
PLC ← PLC+NBYTES
30
OBTEN DE TABLA CODIGO DE MAQUINA
R4 ← 0
R
EVALUA OPERANDO
JSR PC, CREA
JSR PC, REVISA
JSR PC, EVALUA
VE A TABLA
BIT IS ENCENDIDO?
NO
NBYTES ← 2
SE ACTUALIZA EL CODIGO DE MAQUINA
SI
NBYTES ← 3
35


33
EVALUA OPERANDO (NEXTKN)
OPERANDO > 255
NO
ADMITE MODO DIRECTO
SI
NBYTES ← 2
MODO EXTENDIDO
NO
NBYTES ← 3
TRAE CODIGO DE MAQUINA DE TABLA
ACTUALIZA EL CODIGO DE MAQUINA
35


USR PC, STK
CARACTER = '!'
SI
28
CAMBIA DE DECIMAL A BINARIO
24
NO
CARACTER = '@'
SI
25
CAMBIA DE OCTAL A BINARIO
24
NO
CARACTER = '%'
SI
26
CAMBIA LOS CARACTERES A BINARIO
24
NO
CARACTER = '$'
SI
27
CAMBIA DE HEXADECIMAL A BINARIO
24
CARACTER = DIGITO?
SI
BUSCA ULTIMO CARACTER
CARACTER = D
SI
25
NO
CARACTER = O
SI
25
NO
CARAC. TER = 'B'
SI
26
NO
CARACTER = 'H'
SI
27
NO
28
NO
TENEMOS SIMBOLO. APLICA HASH
ESTA EN TABLA?
SI
TIENE CLAVE = 9
SI
R14 ← VALOR DE LA TERCERA COLUMNA EN TABLA
NO
NO
ESTAMOS EN EL 2º PASO
SI
ERROR
NO
MODO EXTENDIDO
SI
R1 ← 256
24
RTS PC
NO
ERROR


JSR PC, TRABR
¿FIN DEL ARCHIVO DEL USUARIO?
SI
ESTAMOS ALMACENANDO EL CUERPO DE UN MACRO?
NO
NO
SI
LEE UNA LINEA EN EL BUFFER BUFF
ERROR
JSR PC, VIOLA
R1 ← TOTAL DE CARACTERES LEIDOS
CIERRA ARCHIVO DE CUERPOS DE MACROS MACRO.TMP
RTS PC
C
CIERRA EL ARCHIVO DEL USUARIO
3

4.6 Comentarios generales.

El diseño del ensamblador no toma en cuenta la relocalización del programa debido a los siguientes problemas que se presentan:

i) Al utilizarse la pseudo instrucción ORG se definen multiples - PLC por lo tanto los símbolos y literales relocalizables deberán indicar respecto a cual PLC son relocalizables.

ii) Las expresiones deberán también clasificarse en: relocalizables, absolutas e indefinidas. Hay algunos lenguajes ensambladores que restringen el uso de expresiones aritméticas en el - campo del operando, pero el problema persiste para seguir clasificando estas expresiones ya que el usuario se puede equivocar al escribir una expresión.

Existen computadoras con características que facilitan la relocalización de un programa. En estas computadoras todas las direcciones se generán relativas a un registro base. Cuando el programa se ejecuta el valor del registro base se toma como offset para generar direcciones para el resto de los modos de direccionamiento. La relocalización se lleva a cabo simplemente cargando la dirección inicial en el registro base y relocalizando solo las constantes de dirección aunque se pueden implementar de otra manera que - no necesita relocalización.

La complejidad de las direcciones manejadas por el ensamblador y el cargador se reduce grandemente dependiendo de las caracterfsticas de la organización de una computadora (16). Desafortunadamente la M6800 no permite esta solución.

Los lenguajes ensambladores pueden permitir el uso de referencias externas - a través de la pseudo-instrucción EXTRN - que son

simplemente símbolos mencionados en un programa llamemosle A aun-- que fueron definidos - a través de la pseudo-instrucción ENTRY - en otro programa llamemosle B traducido independientemente del programa A.

La dificultad para implementar referencias externas en el lenguaje ensamblador de la M6800 se debe a que para saber si un modo es directo o extendido se debe evaluar el operando: si éste contiene una referencia externa no se sabrá en cuanto incrementar el PLC y aunque se optará por tomarle como modo extendido podría ocurrir un error si al resolver las referencias externas el valor es menor a 255. Otro problema sería el contar con un área suficientemente - grande que permita las ligas de las variables externas.

Cuando la M6800 tiene un ensamblador corresidente cuenta ---- con la opción de borrar o no la tabla de símbolos, lo que permite manejar referencias externas sin necesidad de usar la pseudo-ins-- trucciones ENTRY y EXTRN bastará definir en el primer programa las variables externas que se van a utilizar.

CONCLUSIONES

Al profundizar en el estudio de los ensambladores se llega a la conclusión de que las referencias que se pueden consultar solo son una guía para quien diseñe un ensamblador. El programador deberá en muchos casos aplicar su criterio para resolver los problemas que se le presenten.

Se deberá tener presente que no existen un conjunto de reglas que se deberán seguir sino un propósito: producir el lenguaje máquina.

A lo largo de la tesis se ha procurado escribir las dificultades encontradas y la manera de resolverlas. Muchas de estas dificultades no las mencionan los autores consultados o no plantean la forma de resolverlas. La complejidad para resolver muchos problemas dependerá de la organización de la computadora involucrada. La microcomputadora M6800 es una máquina pequeña y por ello es difícil producir código relocalizable.

Debido a la importancia de las macroinstrucciones se implementaron macros para la M6800. Esto podría desarrollar software portables para obtener un ensamblador corresidente. Como se puede ver - la tesis puede ser un punto de partida para seguir generando software para la M6800.

La estrategia que se recomienda a quien diseñe un macroensamblador es:

- conocer el hardware y el lenguaje ensamblador de su máquina.
- entender que es un ensamblador, una macro y un macroprocesador.

. diseñar.

. programar y por último realizar un periodo de prueba.

BIBLIOGRAFIA

1.- Aho, V. Alfred y Ullman, D. Jeffrey.
Principles of Compiler Design.
Reading, Massachussetts.
Addison - Wesley, 1977. 604 pp.

2.- Albarrán Mario y Olvera Arturo.
Técnicas de programación para la computadora DEC - PDP - 11/10 con el sistema RT11. 28 pp.
Comunicaciones Internas No. 48, 1978.
Departamento de Matemáticas.
Facultad de Ciencias, UNAM.

3.- Bajar, R. Victoria.
Introducción al sistema operativo RSX - 11M para digital PDP-11
México, Limusa, 1980. 390 pp.

4.- Barrow, D. W.
Assemblers and loaders.
3a. edición. London, Macdonald and Jane's, 1978. 96 pp.

5.- Bennett, S. William y Exert, F. Carl.
What every engineer should know about microcomputers.
Hardaware an software design a step - by - step example.
New York, Marcel Dekker, 1980. 175 pp.
(Motorola series in solid - state electronics).

6.- Bishop, Ron.
Basic microprocessors and the 6800.
Rochelle Park, New Jersey, Hayden Book Company, 1979. 253 pp.
(Motorola series in solid - state electronics).

7.- Brown, P. J.
The ML / I Macro Procesor.
Communications ACM.
Vol. 10, No. 10, October 1967.

8.- Brown, P.J.
Macro processor and techniques for portable software..
Wiley, London, 1974.

9.- Donovan, John.
Programación de sistemas.
Argentina, El ateneo, 1977. (C 1972). 443 pp.

10.- Eckhoouse, H. Richard.
Minicomputer Systems: Organization and programming (PDP - 11).
Englewood Clifs, N. J., Prentice - Hall, 1975. 343 pp.

11.- Fletcher, John. G.
A Program to Solve the Pentomine Problem by the Recursive Use of Macros Communications of the ACM.
Volume 8, No. 10, October, 1965. pag. 621 - 623.

12.- Garland, Harry.
Introduction to microprocessor system design.
Tokyo, Japón, McGraw - Hill Kogakusha, 1979. 192 pp.

13.- Gear, Charles William.
Computer organization and programming.
3a. edición.
Tokyo, Japón, McGraw - Hill Kogakusha, 1980. 442 pp.

14.- Gill, Arthur.
Machine and assembly language programming of the PDP - 11
Englewood Cliffs, N. J., Prentice - Hall, 1978. 191 pp.

15.- Halstead, Maurice Howard.
A laboratory manual for compiler and operating system implementation.
New York, American Elsevier Publishing Co., Inc., 1974. 108 pp.

16.- Hamacher, V.C y Otros.
Computer Organization.
Tokyo, Japón, McGraw - Hill Kogakusha, 1978.

17.- Hassey, A. Alejandro.
Diseño de un simulador y ensamblador para una computadora digital.
México, 1974. 115 pp.
TESIS - (Ingeniero mécanico electricista en control, comunicaciones y electronica).
Universidad Anáhuac.

18.- Hilburn, L. John. y Julich, M. Paul.
Microcomputers / microprocessors: hardware, software and applications.
Englewood Cliffs, N. J., Prentice - Hall, 1976. 365 pp.

19.- Kent, William.
Assembler - Language macroprogramming: a tutorial oriented toward the IBM 360.
Computing Surveys.
Vol. 1, No. 4, 1969. pag. 183 - 196.

20.- Kernighan, Brian. W. y Plauger, P. J.
Software Tools.
Reading, Massachussetts, Addison - Wesley, 1976. 338 pp.

21.- Knuth, E. Donald.
Algorithms.
Scientific American.
Vol. 236, No. 4, April 1977, pag. 63 - 80.

22.- Knuth, E. Donald.
The Art of Computer programing.
Vol. 3 Sorting and Searching.
2a. edición. Reading, Massachussetts, Addison - Wesley, 1975.
(c 1973) 723 pp.

23.- Mandado, E. y Tassis, E.
Diseño de sistemas digitales con microprocesadores.
Barcelona, España, Marcombo Boixareu Editores, 1980. 256 pp.

24.- Maurer, W. D.
An Improved Hash Code for Scatter Storage.
Communications ACM.
Vol. 11, No. 1, January 1968, Pag. 35 - 38.

25.- Maurer, W. D. y Lewis, T. G.
Hash table methods.
Communications ACM.
Vol. 7, No. 1, March 1975. pags. 5 - 19.

26.- Moore, Alvin y otros.
Microprocessor Aplications Manual.
New York, McGraw - Hill, 1975. (pag. varía)
(Motorola serie in solid - state electronics).

27.- Morris, Robert.
Scatter Storage Techniques.
Communications ACM.
Vol. 11, No. 1, January 1968. pags. 38 - 44.

28.- Osborne, Adam.
An introduction to microcomputers.
Volume 0. The beginner's book.
2a. edición. Berkeley,California, 1979. (c 1977) (pag. varía)

29.- Poe, Elmer.
Using the 6800 Microprocessor.
Indianapolis, Indiana, Howard W. Sams, 1978. 177 pp.

30.- Presser, Leon y White, R. John.
Linkers and Loaders.
Computing Surveys.
Vol. 4, No. 3, September 1972. pags. 149 - 167.

31.- Sippl, J. Charles.
Microcomputer handbook.
New York, Van Nostrand Reinhold Company, 1977. 454 pp.

32.- Soucer, Branko.
Microprocessors and microcomputers.
New York, Wiley - Interscience, 1976. 603 pp.

33.- Southern, Bob.
Programming the 6800 microprocessor.
Canada, Mototrola Semiconductor Products Inc, 1977. (pag varía).

34.- Stone, S. Harold y Siewiorek, P. Daniel.
Introduction to computer organization and data structures,
PDP - 11 edition.
New York, McGraw - Hill, 1975. 368 pp.

35.- Strachey, C.
A general purpose macrogenerator.
Computer Journal, October, 1965. pags. 225 - 241.

36.- Ullman, Jeffy. D.
Fundamental concepts of programing systems.
Reading, Massachussetts, Addison - Wesley, 1976. 328 pp.

37.- Waite, William. M.
A language independient Macro Processor.
Communications of the ACM.
Vol. 10, No. 7, July 1967. pags. 433 - 440.

38.- Wegner, Peter.
Programming Languages, information structures and machine organization.
New York, McGraw - Hill, 1968. 401 pp.

39.- Microelectronics.
Scientif American.
Vol. 237, No. 3, September 1977. pags. 1 - 262.

40.- CONFERENCIA SOBRE MICROPROCESADORES Y MICROCOMPUTADORES.
MEXICO, 1980.
Panorama de los microprocesadores de 16 bits.
Portilla Marcial.
Fundación Arturo Rosenbluet.

MANUALES

. IAS / RSX - 11 RSX - 11 I / O Operations Reference Manual.
Digital Equipement Corporation.
Maynard, Mass., 1978.

. IAS / RSX - 11 Macro - 11 Reference Manual.
Digital Equipment Corporation.
Maynard, Mass., 1978.

. IAS / RSX - 11 System Library Routines Reference Manual.
Digital Equipment Corporation.
Maynard, Mass., 1978.

. M6800 Co-resident assembler reference manual
USA, Motorola Inc, 1976.

. M6800 Microprocessor programming manual.
USA, Motorola Inc, 1976.

. PDP - 11 04 / 05 / 10 / 35 / 40 / 45 processor handbook
Digital Equipment Corporation, 1975.

. RSX - 11M Beginner's Guide.
Digital Equipment Corporation.
Maynard, Mass., 1977.

. RSX - 11M Operating System Course Handouts.
Digital Equipment Corporation.
Maynard, Mass., 1977.

APENDICE A

CONJUNTO DE INSTRUCCIONES PARA MOTOROLA M6800.

ACCUMULATOR AND MEMORY OPERATIONS	MNEMONIC	ADDRESSING MODES: IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	INHER OP	INHER ~	INHER #	BOOLEAN/ARITHMETIC OPERATION (All register labels refer to contents)	COND. CODE REG. 5 H	4 I	3 N	2 Z	1 V	0 C
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3				A + M → A	↕	•	↕	↕	↕	↕
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3				B + M → B	↕	•	↕	↕	↕	↕
Add Acmltrs	ABA													1B	2	1	A + B → A	↕	•	↕	↕	↕	↕
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3				A + M + C → A	↕	•	↕	↕	↕	↕
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3				B + M + C → B	↕	•	↕	↕	↕	↕
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3				A • M → A	•	•	↕	↕	R	•
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3				B • M → B	•	•	↕	↕	R	•
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3				A • M	•	•	↕	↕	R	•
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3				B • M	•	•	↕	↕	R	•
Clear	CLR							6F	7	2	7F	6	3				00 → M	•	•	R	S	R	R
	CLRA													4F	2	1	00 → A	•	•	R	S	R	R
	CLRB													5F	2	1	00 → B	•	•	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3				A − M	•	•	↕	↕	↕	↕
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3				B − M	•	•	↕	↕	↕	↕
Compare Acmltrs	CBA													11	2	1	A − B	•	•	↕	↕	↕	↕
Complement, 1's	COM							63	7	2	73	6	3				$\overline{M}$ → M	•	•	↕	↕	R	S
	COMA													43	2	1	$\overline{A}$ → A	•	•	↕	↕	R	S
	COMB													53	2	1	$\overline{B}$ → B	•	•	↕	↕	R	S
Complement, 2's (Negate)	NEG							60	7	2	70	6	3				00 − M → M	•	•	↕	↕	①	⑦
	NEGA													40	2	1	00 − A → A	•	•	↕	↕	①	②
	NEGB													50	2	1	00 − B → B	•	•	↕	↕	①	②
Decimal Adjust, A	DAA													19	2	1	Converts Binary Add. of BCD Characters into BCD Format	•	•	↕	↕	↕	③
Decrement	DEC							6A	7	2	7A	6	3				M − 1 → M	•	•	↕	↕	④	•
	DECA													4A	2	1	A − 1 → A	•	•	↕	↕	④	•
	DECB													5A	2	1	B − 1 → B	•	•	↕	↕	④	•
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3				A ⊕ M → A	•	•	↕	↕	R	•
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3				B ⊕ M → B	•	•	↕	↕	R	•
Increment	INC							6C	7	2	7C	6	3				M + 1 → M	•	•	↕	↕	⑤	•
	INCA													4C	2	1	A + 1 → A	•	•	↕	↕	⑤	•
	INCB													5C	2	1	B + 1 → B	•	•	↕	↕	⑤	•
Load Acmltr	LDAA	86	2	2	96	3	2	A6	5	2	B6	4	3				M → A	•	•	↕	↕	R	•
	LDAB	C6	2	2	D6	3	2	E6	5	2	F6	4	3				M → B	•	•	↕	↕	R	•
Or, Inclusive	ORAA	8A	2	2	9A	3	2	AA	5	2	BA	4	3				A + M → A	•	•	↕	↕	R	•
	ORAB	CA	2	2	DA	3	2	EA	5	2	FA	4	3				B + M → B	•	•	↕	↕	R	•
Push Data	PSHA																						

	PSHB													37	4	1	B → M_{SP}, SP – 1 → SP	•	•	•	•	•	•
Pull Data	PULA													32	4	1	SP + 1 → SP, M_{SP} → A	•	•	•	•	•	•
	PULB													33	4	1	SP + 1 → SP, M_{SP} → B	•	•	•	•	•	•
Rotate Left	ROL							69	7	2	79	6	3				M	•	•	↕	↕	⑤	↕
	ROLA													49	2	1	A	•	•	↕	↕	⑤	↕
	ROLB													59	2	1	B	•	•	↕	↕	⑤	↕
Rotate Right	ROR							66	7	2	76	6	3				M	•	•	↕	↕	⑤	↕
	RORA													46	2	1	A	•	•	↕	↕	⑤	↕
	RORB													56	2	1	B	•	•	↕	↕	⑤	↕
Shift Left, Arithmetic	ASL							68	7	2	78	6	3				M	•	•	↕	↕	⑤	↕
	ASLA													48	2	1	A	•	•	↕	↕	⑤	↕
	ASLB													58	2	1	B	•	•	↕	↕	⑤	↕
Shift Right, Arithmetic	ASR							67	7	2	77	6	3				M	•	•	↕	↕	⑤	↕
	ASRA													47	2	1	A	•	•	↕	↕	⑤	↕
	ASRB													57	2	1	B	•	•	↕	↕	⑤	↕
Shift Right, Logic	LSR							64	7	2	74	6	3				M	•	•	R	↕	⑤	↕
	LSRA													44	2	1	A	•	•	R	↕	⑤	↕
	LSRB													54	2	1	B	•	•	R	↕	⑤	↕
Store Acmltr.	STAA				97	4	2	A7	6	2	B7	5	3				A → M	•	•	↕	↕	R	•
	STAB				D7	4	2	E7	6	2	F7	5	3				B → M	•	•	↕	↕	R	•
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3				A – M → A	•	•	↕	↕	↕	↕
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3				B – M → B	•	•	↕	↕	↕	↕
Subract Acmltrs.	SBA													10	2	1	A – B → A	•	•	↕	↕	↕	↕
Subtr. with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3				A – M – C → A	•	•	↕	↕	↕	↕
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3				B – M – C → B	•	•	↕	↕	↕	↕
Transfer Acmltrs	TAB													16	2	1	A → B	•	•	↕	↕	R	•
	TBA													17	2	1	B → A	•	•	↕	↕	R	•
Test, Zero or Minus	TST							6D	7	2	7D	6	3				M – 00	•	•	↕	↕	R	R
	TSTA													4D	2	1	A – 00	•	•	↕	↕	R	R
	TSTB													5D	2	1	B – 00	•	•	↕	↕	R	R

FIGURE 1-3-1 MC6800 Instruction Set

170

INDEX REGISTER AND STACK POINTER OPERATIONS	MNEMONIC	IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	INHER OP	INHER ~	INHER #	BOOLEAN/ARITHMETIC OPERATION	5 H	4 I	3 N	2 Z	1 V	0 C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	BC	5	3				$(X_H/X_L) - (M/M+1)$	•	•	⑦	↕	⑧	•
Decrement Index Reg	DEX													09	4	1	$X - 1 \rightarrow X$	•	•	•	↕	•	•
Decrement Stack Pntr	DES													34	4	1	$SP - 1 \rightarrow SP$	•	•	•	•	•	•
Increment Index Reg	INX													08	4	1	$X + 1 \rightarrow X$	•	•	•	↕	•	•
Increment Stack Pntr	INS													31	4	1	$SP + 1 \rightarrow SP$	•	•	•	•	•	•
Load Index Reg	LDX	CE	3	3	DE	4	2	EE	6	2	FE	5	3				$M \rightarrow X_H, (M+1) \rightarrow X_L$	•	•	⑨	↕	R	•
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	6	2	BE	5	3				$M \rightarrow SP_H, (M+1) \rightarrow SP_L$	•	•	⑨	↕	R	•
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3				$X_H \rightarrow M, X_L \rightarrow (M+1)$	•	•	⑨	↕	R	•
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M, SP_L \rightarrow (M+1)$	•	•	⑨	↕	R	•
Indx Reg → Stack Pntr	TXS													35	4	1	$X - 1 \rightarrow SP$	•	•	•	•	•	•
Stack Pntr → Indx Reg	TSX													30	4	1	$SP + 1 \rightarrow X$	•	•	•	•	•	•

JUMP AND BRANCH OPERATIONS	MNEMONIC	RELATIVE OP	RELATIVE ~	RELATIVE #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	INHER OP	INHER ~	INHER #	BRANCH TEST	5 H	4 I	3 N	2 Z	1 V	0 C
Branch Always	BRA	20	4	2										None	•	•	•	•	•	•
Branch If Carry Clear	BCC	24	4	2										C = 0	•	•	•	•	•	•
Branch If Carry Set	BCS	25	4	2										C = 1	•	•	•	•	•	•
Branch If = Zero	BEQ	27	4	2										Z = 1	•	•	•	•	•	•
Branch If ≥ Zero	BGE	2C	4	2										$N \oplus V = 0$	•	•	•	•	•	•
Branch If > Zero	BGT	2E	4	2										$Z + (N \oplus V) = 0$	•	•	•	•	•	•
Branch If Higher	BHI	22	4	2										$C + Z = 0$	•	•	•	•	•	•
Branch If ≤ Zero	BLE	2F	4	2										$Z + (N \oplus V) = 1$	•	•	•	•	•	•
Branch If Lower Or Same	BLS	23	4	2										$C + Z = 1$	•	•	•	•	•	•
Branch If < Zero	BLT	2D	4	2										$N \oplus V = 1$	•	•	•	•	•	•
Branch If Minus	BMI	2B	4	2										N = 1	•	•	•	•	•	•
Branch If Not Equal Zero	BNE	26	4	2										Z = 0	•	•	•	•	•	•
Branch If Overflow Clear	BVC	28	4	2										V = 0	•	•	•	•	•	•
Branch If Overflow Set	BVS	29	4	2										V = 1	•	•	•	•	•	•
Branch If Plus	BPL	2A	4	2										N = 0	•	•	•	•	•	•
Branch To Subroutine	BSR	8D	8	2											•	•	•	•	•	•
Jump	JMP				6E	4	2	7E	3	3				See Special Operations	•	•	•	•	•	•
Jump To Subroutine	JSR				AD	8	2	BD	9	3					•	•	•	•	•	•

No Operation	NOP																		
Return From Interrupt	RTI																		
Return From Subroutine	RTS																		
Software Interrupt	SWI																		
Wait for Interrupt	WAI																		

3B	10	1	See special Operations																
39	5	1																	
3F	12	1																	
3E	9	1																	

CONDITIONS CODE REGISTER		INHER			BOOLEAN	5	4	3	2	1	0
OPERATIONS	MNEMONIC	OP	~	=	OPERATION	H	I	N	Z	V	C
Clear Carry	CLC	0C	2	1	0 → C	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	2	1	0 → I	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	0 → V	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	1 → C	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	1 → I	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	1 → V	•	•	•	•	S	•
Acmltr A → CCR	TAP	06	2	1	A → CCR	——— (12) ———					
CCR → Acmltr A	TPA	07	2	1	CCR → A	•	•	•	•	•	•

CONDITION CODE REGISTER NOTES:

(Bit set if test is true and cleared otherwise)

① (Bit V) Test: Result = 10000000?

② (Bit C) Test: Result = 00000000?

③ (Bit C) Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)

④ (Bit V) Test: Operand = 10000000 prior to execution?

⑤ (Bit V) Test: Operand = 01111111 prior to execution?

⑥ (Bit V) Test: Set equal to result of N ⊕ C after shift has occurred

⑦ (Bit N) Test: Sign bit of most significant (MS) byte of result = 1?

⑧ (Bit V) Test: 2's complement overflow from subtraction of LS bytes?

⑨ (Bit N) Test: Result less than zero? (Bit 15 = 1)

⑩ (All) Load Condition Code Register from Stack. (See Special Operations)

⑪ (Bit I) Set when interrupt occurs. If previously set, a Non-Maskable Interrupt is required to exit the wait state.

⑫ (ALL) Set according to the contents of Accumulator A.

LEGEND:

OP	Operation Code (Hexadecimal);	00	Byte = Zero;
~	Number of MPU Cycles;	H	Half-carry from bit 3;
=	Number of Program Bytes;	I	Interrupt mask
+	Arithmetic Plus;	N	Negative (sign bit)
-	Arithmetic Minus;	Z	Zero (byte)
•	Boolean AND;	V	Overflow, 2's complement
M_{SP}	Contents of memory location pointed to be Stack Pointer;	C	Carry from bit 7
		R	Reset Always
		S	Set Always
+	Boolean Inclusive OR;	↕	Test and set if true, cleared otherwise
⊕	Boolean Exclusive OR;	•	Not Affected
$\bar{M}$	Complement of M;	CCR	Condition Code Register
→	Transfer Into;	LS	Least Significant
0	Bit = Zero;	MS	Most Significant

FIGURE 1-3-1 (continued)

APENDICE B

NOTACION SINTACTICA BNF PARA EL LENGUAJE ENSAMBLADOR DE MOTOROLA 6800.

< programa > :: = < sentencia > | < programa > < sentencia >

< sentencia > :: = < primera componente > < operador > < operando >

{ < espacio > / TAB } [< comentario >] < CR >

< separador > :: = < espacio > | CR | , | TAB

< primera componente > :: = < espacio > | < etiqueta >

< espacio > :: = < único espacio > | < espacio > < único espacio >

< único espacio > :: = { un blanco }

< CR > :: = { teclear (CR) }

< etiqueta > :: = < letra > | < letra > < etiqueta > |

< etiqueta > < digito >

El total del caracter debe ser de 1 a 6

< letra > :: = A | B | C | .... | Z |

< digito > :: = 0 | 1 | 2 | .... | 9 |

< operando > :: = < números > | < símbolos > | < expresión >

< operando > < etiqueta > |

< operando > < expresión > |

< operando > < número >

< número > :: = < digito > |
< prefijo > < digito > |
< digito > < sufijo >

< prefijo > :: = # | $ | @ | % | ¡ | ' (apostrofe)

< sufijos > :: = B | H | O | Q

< símbolo > :: = < etiqueta > | < contador de programa >

< contador de programa > :: = { " * " }

< expresión > :: = < símbolo > < operador aritmético > |

< número > < operador aritmético >

< operadores aritmético > :: = + | - | * | /

< comentario > :: = < Asterisco > < ASCII > | < ASCII >

< Asterisco > :: = { * en la primera posición de caracter de una línea }

< operador > :: = ABA | ADC | ADD | ..... | TXS | WAI

APENDICE C

FORMATO DE LA LISTA DEL PROGRAMA ENSAMBLADO.

La lista del programa ensamblado incluye el programa fuente y el lenguaje de máquina generado por el ensamblador.

La mayoría de las líneas corresponde a la sentencia fuente.

Las líneas que no corresponden a la sentencia fuente directa-- mente son:

. Las líneas de la expansión para las pseudo-instrucciónes: FCC, FDB, FCB.

. La expansión de macrollamadas.

La mayoría de las líneas sigue el modelo del formato siguiente: (los casos especiales no usan exactamente este formato)

COLUMNA	CONTENIDO
1 - 5	Número de línea. Se escriben 5 digitos en forma decimal. El ensamblador lleva cuenta de las líneas.
7 - 10	Valor del PLC escrito en hex.
12 - 13	Código de máquina del operador (hex)
15 - 16	Primer byte del operando (hex).
17 - 18	Segundo byte del operando (si es que hay).
20 - 25	Campo de etiqueta.
27 - 31	Campo del operador.
34 - 41	Campo del operando.
43 - 80	Campo del comentario.

APENDICE D

LISTA DE LOS ERRORES PRODUCIDOS Y SU SIGNIFICADO.

201 ERROR 201

ERROR EN DIRECTIVA NAM.

Significado: la directiva NAM no es la primera sentencia fuente u ocurre más de una vez en el mismo programa fuente.

202 ERROR 202

ETIQUETA O COD. OPERACION.

Significado: la etiqueta o el símbolo en el campo del operador o el macronombre no empiezan con un caracter alfabetico.

203 ERROR 203

ERROR EN LA SENTENCIA.

Significado: La sentencia está en blanco o solo contiene la etiqueta.

204 ERROR 204

ERROR DE SINTAXIS.

Significado: La sentencia es sintacticamente incorrecta.

205 ERROR 205

ERROR EN ETIQUETA.

Significado: La sentencia no permite una etiqueta o la etiqueta es sintacticamente incorrecta.

206 ERROR 206

SIMBOLO REDEFINIDO.

Significado: El símbolo ha sido previamente definido.

207 ERROR 207

COD. DE OPERACION INDEFINIDO.

Significado: El símbolo en el campo del operador no es valido ni como instrucción ni como directiva.

208 ERROR 208

ERROR DE BRANCH.

Significado: El contador del branch (transferencia) se ex-- tiende más allá del intervalo:

$(* + 2) - 128 \leq D \leq (* + 2) + 127$

donde D = dirección del destino de la instrucción de branch.

* = dirección del primer byte de la instrucción de -- branch.

209 ERROR 209

MODO DE DIRECCIONAMIENTO ILEGAL.

Significado: El modo de direccionamiento no se permite con el código de operación.

210 ERROR 210

BYTE OVERFLOW.

Significado: Una expresión que debe ocupar un byte fue convertida a un valor mayor que 255_{10}. Este error también ocurre -- cuando se usan operandos negativos en el modo de direcciona-- miento inmediato. Ejemplo.

```
LDAA # - 5 : CAUSA ERROR 210
```

Este error puede evitarse usando el complemento a 2 en forma octal del número. Ejemplo:

```
LDAA #$FD  : ENSAMBLA BIEN
```

211 ERROR 211

SIMBOLO INDEFINIDO

Significado: El símbolo no aparece en el campo de etiqueta

212 ERROR 212

ERROR EN EL OPERANDO DE UNA DIRECTIVA.

Significado: Error de sintaxis en el campo del operando de -- una directiva.

213 ERROR 213

ERROR EN DIRECTIVA EQU

Significado: La directiva EQU requiere una etiqueta.

214 ERROR 214

ERROR EN MACRODEFINICION

Significado: No existe macrononbre en la definición de la macro.

215 ERROR 215

ERROR EN ARGUMENTO MUDO

Significado: Error de sintaxis en algun argumento mudo de una macrodefinición.

216 ERROR 216

MACRO INTERIOR

Significado: No se permiten macrollamadas en el cuerpo de una macroinstrucción.

217 ERROR 217

MACRO ANIDADO

Significado: No se permiten macros anidados. El mensaje se -- produce cuando se ha encontrado dos veces seguidas la directiva MACRO y no se encontro ningun MEND.

218 ERROR 218

WORD OVERFLOW

Significado: El PLC excede a los valores que se le permite to

mar c es un número negativo.

219 ERROR 219

FALTAN ARGUMENTOS REALES

Significado: El número de argumentos reales en una macrollamada es menor que el número de argumentos mudos de una macrodefinición.

220 ERROR 220

NO APARECE MEND

Significado: El usuario se olvido de escribir la directiva -- MEND en la definición de una macro.

221 ERROR 221

TABLA DE SIMBOLOS LLENA

Significado: La tabla de símbolos no puede almacenar más etiquetas, se ha excedido su capacidad.

222 ERROR 222

ERROR DE FASE

Significado: El valor del PLC durante el primer paso y el segundo paso fue diferente. El error se produce si el usuario utiliza símbolos o literales en el modo directo antes de que sean definidas.

223 ERROR 223

LA DIRECTIVA NO ADMITE ETIQUETA

Significado: La directiva no admite tener una etiqueta. El -- campo de etiqueta debe estar vacio.

224 ERROR 224

MAS DE 35 ERRORES

Significado: La acción del ensamblador acaba al cometerse 35

errores de sintaxis.

225 ERROR 225

ERROR EN MACROLLAMADA

Significado: Este error sucede cuando hay un error en la ma-- crodefinición, por lo tanto la macrollamada también estará -- mal.

226 ERROR 226

NO EXISTE END

Significado: El usuario debe poner un END al final de su pro- grama

APENDICE E

EJEMPLOS DE PROGRAMAS CORRIDOS

```
>PIP TI:=MOT80.TMP
```

```
                        FECHA:  16-FEB-82     HORA:  12:27:56
                        .TITULO ORG
00001                           NAM     ORG
00002                   *
00003                   *       PROGRAMA PARA ILUSTRAR EL USO DE LA DIRECTIVA ORIGEN
00004                   *
00005 0000 0001         BILL    RMB     1           EL PLC SE INICIALIZA EN CERO
00006      0001         JOHN    EQU     *
00007 0020                      ORG     $20         ALTERA PLC A 20 EN HEXADECIMAL
00008 0020 000A                 RMB     10
00009 0001                      ORG     JOHN        ALTERA PLC AL VALOR DE JOHN
00010 0001 000A                 RMB     10
00011                           END
JOHN    0001 BILL    0000
>
```

```
>PIP TI:=MOT80.TMP^
```

```
                        FECHA:  16-FEB-82     HORA:  12:32:34
                        .TITULO FDB
00001                           NAM     FDB
00002                   *
00003                   *       PROGRAMA PARA ILUSTRAR EL USO DE LA DIRECTIVA
00004                   *       FDB (DOBLE FORMA CONSTANTE DE BYTE)
00005                   *
00006 0000 0002                 FDB     2
00007 0002 0000         LABEL   FDB     ,$F,$FF,$FFF,,$FFFF
      0004 000F
      0006 00FF
      0008 0FFF
      000A 0000
      000C FFFF
00008 000E 000C                 FDB     LABEL+10,LABEL+5,LABEL
      0010 0007
      0012 0002
00009                           END
LABEL   0002
>
```

```
>PIP TI:=MOT80.TMP
```



```
                    FECHA:  25-FEB-82     HORA:  17:56:06
                    .TITULO RMB
00001                       NAM     RMB
00002                   * PROGRAMA PARA ILUSTRAR EL USO DE LA RESERVA
00003                   * DE MEMORIA POR BYTE
00004 0000 0001     CLAB1   RMB     1
00005 0001 0002     CLAB2   RMB     2
00006 0003 0003             RMB     *-CLAB1
00007 0006 0002             RMB     %10
00008 0008 0008             RMB     10Q
00009 0010 0026             RMB     %100+CLAB2+$F-1*2
00010                       END
CLAB1   0000 CLAB2   0001
>
```

```
PIP TI:=MOT80.TMP
```

```
                    FECHA:  25-FEB-82    HORA:  17:53:00
                    .TITULO PGM
00001                       NAM     PGM
00002                   *
00003                   * PROGRAMA SIMPLE PARA EJEMPLIFICAR EL USO DE LOS MODOS
00004                   * DE DIRECCIONAMIENTO DE MOTOROLA 6800
00005                   *
00006 0200                  ORG     $200        PLC<=2000 EN HEXADECIMAL
00007       0003    COUNT   EQU     @3          @ INDICA OCTAL
00008 0200 8E 0232 START   LDS     #STACK      INICIALIZA EL APUNTADOR DEL STACK
00009 0203 FE 0237         LDX     ADDR
00010 0206 C6 03           LDA B   #COUNT      MODO INMEDIATO
00011 0208 96 0A    BACK   LDA A   10          MODO DIRECTO
00012 020A A1 02           CMP A   2,X         MODO INDEXADO
00013 020C 27 05           BEQ     FOUND       MODO RELATIVO
00014 020E 09              DEX                 MODO IMPLICADO
00015 020F 5A              DEC B               MODO ACUMULADOR
00016 0210 26 F6           BNE     BACK
00017 0212 3E              WAI                 ESPERA LA INTERRUPCION
00018 0213 BD 0219 FOUND   JSR     SUBRTN      SALTA A LA SUBRUTINA
00019 0216 7E 0200         JMP     START       MODO EXTENDIDO
00020                   * COMENTARIO PARA INDICAR COMO SE TRUNCA LA LINEA 1234567890
00021 0219 16       SUBRTN TAB
00022 021A BA 0233         ORA A   BYTE        FIJA EL BIT MAS SIGNIFICATIVO
00023 021D 39              RTS                 REGRESA DE SUBRUTINA
00024                   *
00025 021E 0014            RMB     20          AREA PARA SCRATCH DEL STACK
00026 0232 0001     STACK  RMB     1
00027 0233 80       BYTE   FCB     $80         FORMA CONSTANTE DE BYTE
00028 0234 10              FCB     $10,$4      $ INDICA HEXADECIMAL
      0235 04
00029 0236 05              FCB     %101        % INDICA BINARIO
00030 0237 0239     ADDR   FDB     DATA        FORMA CONSTANTE DE DOBLE BYTE
00031 0239 53       DATA   FCC     /SET/       FORMA CONSTANTE DE DATOS DE CADENA ASC
      023A 45
      023B 54
00032                      END
BACK   0208 BYTE   0233 COUNT  0003 SUBRTN 0219 DATA   0239
START  0200 STACK  0232 ADDR   0237 FOUND  0213
```

```
                    FECHA:  25-FEB-82     HORA:  18:09:19
                    .TITULO ITEM2
00001                       NAM     ITEM2
00002               *  ---          SUMA DE DOS NUMEROS DE DOBLE PRECISION
00003               *  ---          EN CODIGO DECIMAL
00004         0008  NB        EQU     8
00005               *  -----   EMPIEZA SUBRUTINA
00006 1000                  ORG     $1000
00007 1000 C6 08    BCD     LDA B   #NB
00008 1002 FE 0100          LDX     ADDR        CARGA DIRECCION DE DATOS
00009 1005 0C               CLC
00010 1006 A6 07    NEXT    LDA A   NB-1,X      INICIA LOOP
00011 1008 A9 0F            ADC A   2*NB-1,X
00012 100A 19               DAA
00013 100B A7 17            STA A   3*NB-1,X
00014 100D 09               DEX
00015 100E 5A               DEC B
00016 100F 26 F5            BNE     NEXT
00017 1011 39               RTS                 FIN DE LA SUBRUTINA BCD
00018               *  ---          EMPIEZA PROGRAMA PRINCIPAL ......
00019               *  ---          PRUEBA LA SUBRUTINA "BCD"
00020 1100                  ORG     $1100
00021 1100 8E 013F          LDS     #$13F       INICIALIZA EL APUNTADOR DEL STACK
00022 1103 CE 0102          LDX     #P          CARGA LA DIRECCION DE P
00023 1106 FF 0100          STX     ADDR
00024 1109 BD 1000          JSR     BCD         HAZ LA SUMA BCD
00025 110C 01               NOP
00026 110D 20 FE            BRA     *           FIN DEL PROGRAMA PRINCIPAL
00027               *  ---- RESERVA UNA AREA EN LA MEMORIA DE ESCRITURA-LECTURA
00028 0100                  ORG     $0100
00029               *  -----        (1) PARA LA SUBRUTINA
00030 0100 0002     ADDR    RMB     2
00031               *  ----         (2) PARA EL PROGRAMA PRINCIPAL
00032 0102 0008     P       RMB     NB
00033 010A 0008     Q       RMB     NB
00034 0112 0008     RES     RMB     NB
00035                       END
P       0102 NEXT   1006 BCD    1000 RES    0112 Q      010A
ADDR    0100 NB     0008
>
```

```
>PIP TI:=MOTBO.TMP
```

```
                        FECHA:  25-FEB-82     HORA:   18:24:19
                        .TITULO MACROS
00001                           NAM     MACROS
00002                   *------- ESTE PROGRAMA EJEMPLIFICA EL USO DE MACROS
00003                   *------- MUESTRA COMO SE EXPANDEN
00004 0000 0001         PAUL    RMB     1
00005                   * ---
00006                   * ---           PRIMERA MACRODEFINICION: TEST
00007                   * ---
00008                           MACRO TEST A,B1,C33,D12345,B2
00009                           CLR D12345
00010                   A       RMB 1
00011                   B1      CMPA X
00012                   C33     EQU *
00013                   B2      EQU 1
00014                           MEND
00015                   * ---  SEGUNDA MACRODEFINICION: HELP
00016                           MACRO HELP A
00017                           CMPA A
00018                           MEND
00019                   * ---  TERCERA MACRODEFINICION
00020                           MACRO PRUEBA
00021                           NOP
00022                           NOP
00023                           NOP
00024                           MEND
00025                   * ---  PROGRAMA PRINCIPAL QUE CONTIENE MACROLLAMADAS
00026                           TEST L1,L2,L3,PAUL+10,JOHN          MACROLLAMADA TEST
00027 0001 7F 000A                CLR     PAUL+10
00028 0004 0001         L1        RMB     1
00029 0005 A1 00        L2        CMP A   X
00030      0007         L3        EQU     *
00031      0001         JOHN      EQU     1
00032                           PRUEBA                              MACROLLAMADA PRUEBA
00033 0007 01                     NOP
00034 0008 01                     NOP
00035 0009 01                     NOP
00036                           HELP X                              MACROLLAMADA HELP
00037 000A A1 00                  CMP A   X
00038 000C 01                     NOP
00039                             END
L2      0005 PAUL    0000 L3      0007 JOHN    0001 L1      0004
>
```

```
                        FECHA:  25-FEB-82     HORA:  18:12:31
                        .TITULO LPTDVR
00001                           NAM     LPTDVR
00002 EAB0                      ORG     $EAB0
00003                   * EN EL ENSAMBLADOR EDOS FIJAR LAS SIGUIENTES LOCALIDADES
00004                   *     (PDATA)=JMP PDATA
00005                   *     (PDATA1)=JMP PDATA1
00006       EC11        CNTRL   EQU     $EC11     DIRECCION PIA
00007       EC10        DATA    EQU     $EC10     DIRECCION PIA
00008       EAB0        LIST    EQU     *
00009 EAB0 36                       PSH A
00010 EAB1 7F EC11                  CLR     CNTRL
00011 EAB4 86 FF                    LDA A   #$FF
00012 EAB6 B7 EC10                  STA A   DATA
00013 EAB9 86 3E                    LDA A   #$3E
00014 EABB B7 EC11                  STA A   CNTRL
00015 EABE 32                       PUL A
00016 EABF B7 EC10                  STA A   DATA
00017 EAC2 86 36                    LDA A   #$36
00018 EAC4 B7 EC11                  STA A   CNTRL
00019 EAC7 86 3E                    LDA A   #$3E
00020 EAC9 B7 EC11                  STA A   CNTRL
00021 EACC B6 EC11  LIST1           LDA A   CNTRL
00022 EACF 2A FB                    BPL     LIST1
00023 EAD1 B6 EC10                  LDA A   DATA
00024 EAD4 39       LIST2           RTS
00025       EAD5    PDATA   EQU     *
00026 EAD5 86 0D                    LDA A   #$D
00027 EAD7 8D D7                    BSR     LIST
00028 EAD9 86 0A                    LDA A   #$A
00029 EADB 8D D3                    BSR     LIST
00030 EADD A6 00    PDATA1  LDA A   X
00031 EADF 81 04                    CMP A   #4
00032 EAE1 27 F1                    BEQ     LIST2
00033 EAE3 8D CB                    BSR     LIST
00034 EAE5 08                       INX
00035 EAE6 20 F5                    BRA     PDATA1
00036                               END
LIST    EAB0 PDATA  EAD5 LIST2  EAD4 LIST1  EACC CNTRL  EC11
DATA    EC10 PDATA1 EADD
>
```

```
>PIP TI:=A9.DAT
        NAM ERRORES
*
*  ESTE PROGRAMA MUESTRA ALGUNOS DE LOS ERRORES EN QUE PUEDE INCURRIR
*  UN USUARIO.
*
        NAM     FALTAS
LABEL   ORG     - 1
        ORG
        RMB     $FFFF+1
JOHN    EQU     1
JOHN    EQU     4
        EQU     NO
E       EQU     Y
        ORG     $FFFF+2
        ORG     MA
MA      FCC     /ES UN ERROR?
LABL4   FCC     300,1234
        ORG     $FF+-
        CMPA    3,U
        CLR
        COM     1
        CMPA    #8,X
LOOP:   DEX
        END
>
```

```
>PIP TI:=FALTAS.DAT


     EN LINEA NO.6 ERRROR NO.201
          NAM       FALTAS
     201 ERROR EN DIRECTIVA NAM
     EN LINEA NO.7 ERRROR NO.223
LABEL     ORG       1
     223 LA DIRECTIVA NO ADMITE ETIQUETA
     EN LINEA NO.8 ERRROR NO.212
          ORG
     212 ERROR EN EL OPERANDO DE UNA DIRECTIVA
     EN LINEA NO.9 ERRROR NO.210
          RMB       $FFFF+1
     210 BYTE OVERFLOW
     EN LINEA NO.11 ERRROR NO.206
JOHN      EQU       4
     206 SIMBOLO REDEFINIDO
     EN LINEA NO.12 ERRROR NO.213
          EQU       NO
     213 ERROR EN DIRECTIVA EQU
     EN LINEA NO.13 ERRROR NO.211
E         EQU       Y
     211 SIMBOLO INDEFINIDO
     EN LINEA NO.14 ERRROR NO.218
          ORG       $FFFF+2
     218 WORD OVERFLOW
     EN LINEA NO.15 ERRROR NO.211
          ORG       MA
     211 SIMBOLO INDEFINIDO
     EN LINEA NO.16 ERRROR NO.212
MA        FCC       /ES UN ERROR?
     212 ERROR EN EL OPERANDO DE UNA DIRECTIVA
     EN LINEA NO.17 ERRROR NO.212
LABL4     FCC       300,1234
     212 ERROR EN EL OPERANDO DE UNA DIRECTIVA
     EN LINEA NO.18 ERRROR NO.204
          ORG       $FF+-
     204 ERROR DE SINTAXIS
     EN LINEA NO.19 ERRROR NO.209
          CMPA      3,Q
     209 MODO DE DIRECCIONAMIENTO ILEGAL
     EN LINEA NO.20 ERRROR NO.209
          CLR
     209 MODO DE DIRECCIONAMIENTO ILEGAL
     EN LINEA NO.23 ERRROR NO.205
LOOP!     DEX
     205 ERROR EN ETIQUETA
>
```

```
>PIP TI:=A10.DAT
        NAM     FALTAS
*
*  EXISTEN ERRORES QUE SE DETECTAN EN EL SEGUNDO PASO.
*  ESTE LISTADO ES UNA MUESTRA DE ELLO
        DAA
        DECB
        BNE     LOOP
        NOP
        NOP
        JMP     LOOP
        WAI     FALTA END
>
```

```
>PIP TI:=FALTAS.DAT
    EN LINEA NO.7 ERRROR NO.211
        BNE     LOOP
    211 SIMBOLO INDEFINIDO
    EN LINEA NO.10 ERRROR NO.211
        JMP     LOOP
    211 SIMBOLO INDEFINIDO
    EN LINEA NO.11 ERRROR NO.226
        WAI     FALTA END
    226 NO EXISTE END
>
```

```
                        FECHA:  25-FEB-82     HORA:  17:42:59
                        .TITULO ASMPATCH
00001                           NAM     ASMPATCH
00002                   * ESTE PROGRAMA REMIENDA EL ENSAMBLADOR RESIDENTE
00003                   * PARA PEDIR AL USUARIO QUE SELECCIONE SI LA SALIDA ES POR S
00004                   * A LA UNIDAD DE LA IMPRESORA USANDO EL COMANDO PROM
00005        0000       PASS    EQU     0
00006        FF8A       XSTACK  EQU     $FF8A
00007        FF53       AECHO   EQU     $FF53
00008        011E       XDATA   EQU     $11E
00009        011B       XCIE    EQU     $11B
00010        0133       XHEAD   EQU     $133
00011        0136       XLINE   EQU     $136
00012        EAD5       LDATA   EQU     $EAD5
00013        EADD       LDATA1  EQU     $EADD
00014        0100       ASMB    EQU     $100
00015  0020                  ORG     $20
00016  0020  8E FF8A         LDS     #XSTACK
00017  0023  96 00           LDA A   PASS
00018  0025  81 09           CMP A   #$9         SOLO OBJETO?
00019  0027  27 1A           BEQ     ASM         SI
00020  0029  CE 0054  TOP    LDX     #MSG
00021  002C  BD 011E         JSR     XDATA
00022  002F  CE 0000         LDX     #0
00023  0032  09       DELAY  DEX
00024  0033  26 FD           BNE     DELAY
00025  0035  7F FF53         CLR     AECHO
00026  0038  BD 011B         JSR     XCIE
00027  003B  81 59           CMP A   #'Y
00028  003D  27 07           BEQ     PRNTR
00029  003F  81 4E           CMP A   #'N
Q0030  0041  26 E6           BNE     TOP
00031  0043  7E 0100  ASM    JMP     ASMB
00032  0046  CE EAD5  PRNTR  LDX     #LDATA
00033  0049  FF 0134         STX     XHEAD+1
00034  004C  CE EADD         LDX     #LDATA1
00035  004F  FF 0137         STX     XLINE+1
00036  0052  20 EF           BRA     ASM
00037  0054  50       MSG    FCC     /PRINTERT/
       0055  52
       0056  49
       0057  4E
       0058  54
       0059  45
       005A  52
       005B  3F
00038  005C  04               FCB     4
00039                          END
PASS    0000 MSG     0054 XSTACK FF8A AECHO  FF53 LDATA1 EADD
ASMB    0100 ASM     0043 DELAY  0032 XCIE   011B XDATA  011E
PRNTR   0046 XHEAD   0133 TOP    0029 LDATA  EAD5 XLINE  0136
>PIP TI:=MOTBO.TMP
```

APENDICE F

LISTADO DEL MACROENSAMBLADOR

```
>
>PIP TI:=WRIT1.MAC
        .TITLE  PRINCIPAL
;
; ************************************************************************
; MACRO ENSAMBLADOR PARA MOTOROLA 6800
; ************************************************************************
;
; AUTOR: CAROLINA RUIZ LOPEZ
; FECHA: CIUDAD UNIVERSITARIA A 15 DE ENERO DE 1982.
;
; OBSERVACIONES:
; EL PROGRAMA SE IMPLEMENTO EN LA COMPUTADORA PDP-11/34 QUE PERTENECE
; AL LABORATORIO DE COMPUTACION DE LA FACULTAD DE CIENCIAS.
;
; ENTRADAS:
; PIDE LA CUENTA DEL USUARIO (SI EL MACROENSAMBLADOR NO ESTA EN ESA
; CUENTA) Y EL NOMBRE DEL ARCHIVO. POR EJ: TECLEAR [300,1]EJEMPLO.TMP
; SALIDAS:
; SI NO HUBO ERRORES EL PROGRAMA ENSAMBLADO SE ENCUENTRA EN EL ARCHIVO
; FALTAS.DAT
; SALIDAS:
; SI NO HUBO ERRORES EL PROGRAMA ENSAMBLADO SE ENCUENTRA EN EL ARCHIVO
; MOT80.TMP
;
;
        .MCALL EXIT$S,QIOW$C,DELET$
         .ENABL GBL
; TABLA FIJA DONDE SE ENCUENTRAN CARGADAS LAS INSTRUCCIONES Y PSEUDO-INS
; TRUCCIONES EN LAS LOCALIDADES QUE LE ASIGNO LA FUNCION DE HASH Y EL ME
; TODO PARA RESOLVER COLISIONES.
X::      .BLKW 12.
         .RAD50 /BVS/
         .BLKW 1
         .WORD 5
         .ASCII/28/
         .BLKW 30
         .RAD50/TSTB/
         .WORD 4
         .ASCII/4A/
         .BLKW 4
         .RAD50/TST/
         .BLKW 1
         .WORD 6003
         .ASCII/6D/
         .BLKW 10
         .RAD50/TSTA/
         .WORD 4
         .ASCII/4D/
         .RAD50/TSX/
         .BLKW 1
         .WORD 4
         .ASCII/30/
         .BLKW 340
         .RAD50/NEGB/
         .WORD 4
         .ASCII/50/
         .BLKW 10
         .RAD50/NEGA/
         .WORD 4
         .ASCII/40/
         .BLKW 4
         .RAD50/SUBA/
         .WORD 7403
         .ASCII/80/
         .RAD50/NEG/
```

```
.BLKW 1
.WORD 6003
.ASCII/60/
.BLKW 104
.RAD50/SUBB/
.WORD 7403
.ASCII/C0/
.BLKW 100
.RAD50/ANDB/
.WORD 7403
.ASCII/C4/
.BLKW 20
.RAD50/ANDA/
.WORD 7403
.ASCII/B4/
.BLKW 104
.RAD50/SBA/
.BLKW 1
.WORD 4
.ASCII/10/
.BLKW 24
.RAD50/BMI/
.BLKW 1
.WORD 5
.ASCII/2B/
.BLKW 10
.RAD50/ORAA/
.WORD 7403
.ASCII/BA/
.BLKW 20
.RAD50/ORAB/
.WORD 7403
.ASCII/CA/
.RAD50/JMP/
.BLKW 1
.WORD 6003
.ASCII/6E/
.BLKW 50
.RAD50/CLC/
.BLKW 1
.WORD 4
.ASCII/0C/
.BLKW 24
.RAD50/CLI/
.BLKW 1
.WORD 4
.ASCII/0E/
.BLKW 34
.RAD50/CLRA/
.WORD 4
.ASCII/4F/
.RAD50/CLR/
.BLKW 1
.WORD 6003
.ASCII/6F/
.BLKW 4
.RAD50/CLRB/
.WORD 4
.ASCII/5F/
.BLKW 4
.RAD50/CLV/
.BLKW 1
.WORD 4
.ASCII/0A/
.BLKW 10
.RAD50/ORG/
```

```
.BLKW 1
.WORD 10
.BLKW 1
.BLKW 144
.RAD50/TXS/
.BLKW 1
.WORD 4
.ASCII/35/
.BLKW 34
.RAD50/CMPA/
.WORD 7403
.ASCII/81/
.BLKW 10
.RAD50/CMPB/
.WORD 7403
.ASCII/C1/
.BLKW 160
.RAD50/BCC/
.BLKW 1
.WORD 5
.ASCII/24/
.BLKW 74
.RAD50/BCS/
.BLKW 1
.WORD 5
.ASCII/25/
.BLKW 134
.RAD50/COMB/
.WORD 4
.ASCII/53/
.BLKW 4
.RAD50/COM/
.BLKW 1
.WORD 6003
.ASCII/63/
.RAD50/COMA/
.WORD 4
.ASCII/43/
.BLKW 10
.RAD50/DAA/
.BLKW 1
.WORD 4
.ASCII/19/
.RAD50/TAB/
.BLKW 1
.WORD 4
.ASCII/16/
.BLKW 64
.RAD50/TAP/
.BLKW 1
.WORD 4
.ASCII/06/
.BLKW 44
.RAD50/ASLD/
.WORD 4
.ASCII/58/
.BLKW 4
.RAD50/ASL/
.BLKW 1
.WORD 6003
.ASCII/68/
.BLKW 10
.RAD50/ASLA/
.WORD 4
.ASCII/48/
.BLKW 4
```

```
.RAD50/ASRA/
.WORD 4
.ASCII/47/
.RAD50/ASR/
.BLKW 1
.WORD 6003
.ASCII/67/
.BLKW 4
.RAD50/ASRB/
.WORD 4
.ASCII/57/
.BLKW 20
.RAD50/TBA/
.BLKW 1
.WORD 4
.ASCII/17/
.RAD50/MACRO/
.WORD 6
.BLKW 1
.BLKW 24
.RAD50/CFX/
.BLKW 1
.WORD 107403
.ASCII/8C/
.RAD50/BEQ/
.BLKW 1
.WORD 5
.ASCII/27/
.BLKW 34
.RAD50/BGT/
.BLKW 1
.WORD 5
.ASCII/2E/
.BLKW 110
.RAD50/END/
.BLKW 1
.WORD 1
.BLKW 1
.BLKW 124
.RAD50/JSR/
.BLKW 1
.WORD 6003
.ASCII/AD/
.BLKW 70
.RAD50/RTS/
.BLKW 1
.WORD 4
.ASCII/39/
.BLKW 14
.RAD50/BGE/
.BLKW 1
.WORD 5
.ASCII/2C/
.BLKW 54
.RAD50/EORA/
.WORD 7403
.ASCII/88/
.BLKW 10
.RAD50/EORB/
.WORD 7403
.ASCII/C8/
.BLKW 154
.RAD50/DECA/
.WORD 4
.ASCII/4A/
.RAD50/BHI/
```

```
.BLKW 1
.WORD 5
.ASCII/22/
.BLKW 4
.RAD50/DEC/
.BLKW 1
.WORD 6003
.ASCII/6A/
.BLKW 4
.RAD50/DECB/
.WORD 4
.ASCII/5A/
.BLKW 64
.RAD50/DES/
.BLKW 1
.WORD 4
.ASCII/34/
.BLKW 20
.RAD50/DEX/
.BLKW 1
.WORD 4
.ASCII/09/
.BLKW 4
.RAD50/STAA/
.WORD 7003
.ASCII/97/
.BLKW 20
.RAD50/STAB/
.WORD 7003
.ASCII/D7/
.RAD50/NOP/
.BLKW 1
.WORD 4
.ASCII/01/
.BLKW 50
.RAD50/STS/
.BLKW 1
.WORD 7003
.ASCII/9F/
.BLKW 20
.RAD50/STX/
.BLKW 1
.WORD 7003
.ASCII/DF/
.BLKW 4
.RAD50/BITB/
.WORD 7403
.ASCII/C5/
.RAD50/FCC/
.BLKW 1
.WORD 10
.WORD 6
.BLKW 4
.RAD50/EQU/
.BLKW 1
.WORD 10
.WORD 1
.BLKW 4
.RAD50/BITA/
.WORD 7403
.ASCII/85/
.BLKW 44
.RAD50/LSRA/
.WORD 4
.ASCII/44/
.RAD50/LSR/
```

```
.BLKW 1
.WORD 6003
.ASCII/64/
.BLKW 4
.RAD50 /LSRB/
.WORD 4
.ASCII/54/
.BLKW 64
.RAD50/MEND/
.WORD 7
.BLKW 1
.BLKW 34
.RAD50/FDB/
.BLKW 1
.WORD 10
.WORD 4
.BLKW 370
.RAD50/SWI/
.BLKW 1
.WORD 4
.ASCII/3F/
.BLKW 14
.RAD50/BLE/
.BLKW 1
.WORD 5
.ASCII/2F/
.BLKW 50
.RAD50/INCA/
.WORD 4
.ASCII/4C/
.BLKW 10
.RAD50/INC/
.BLKW 1
.WORD 6003
.ASCII/6C/
.RAD50/BLT/
.BLKW 1
.WORD 5
.ASCII/2D/
.RAD50/INCB/
.WORD 4
.ASCII/5C/
.BLKW 64
.RAD50/INS/
.BLKW 1
.WORD 4
.ASCII/31/
.BLKW 20
.RAD50/INX/
.BLKW 1
.WORD 4
.ASCII/0B/
.BLKW 4
.RAD50/RMB/
.BLKW 1
.WORD 10
.WORD 2
.BLKW 30
.RAD50/ABA/
.BLKW 1
.WORD 4
.ASCII/1B/
.BLKW 214
.RAD50/BNE/
.BLKW 1
.WORD 5
```

```
.ASCII/26/
.BLKW 250
.RAD50/ADCA/
.WORD 7403
.ASCII/89/
.BLKW 4
.RAD50/ADDB/
.WORD 7403
.ASCII/CB/
.BLKW 4
.RAD50/ROL/
.BLKW 1
.WORD 6003
.ASCII/69/
.RAD50/ADCB/
.WORD 7403
.ASCII/C9/
.BLKW 4
.RAD50/ADDA/
.WORD 7403
.ASCII/8B/
.BLKW 4
.RAD50/RORA/
.WORD 4
.ASCII/46/
.RAD50/ROR/
.BLKW 1
.WORD 6003
.ASCII/66/
.BLKW 4
.RAD50/RORB/
.WORD 4
.ASCII/56/
.BLKW 130
.RAD50/PSHA/
.WORD 4
.ASCII/36/
.BLKW 10
.RAD50/PSHB/
.WORD 4
.ASCII/37/
.BLKW 24
.RAD50/BPL/
.BLKW 1
.WORD 5
.ASCII/2A/
.BLKW 14
.RAD50/SBCA/
.WORD 7403
.ASCII/82/
.RAD50/CBA/
.BLKW 1
.WORD 4
.ASCII/11/
.BLKW 14
.RAD50/SBCB/
.WORD 7403
.ASCII/C2/
.RAD50/ROLB/
.WORD 4
.ASCII/59/
.BLKW 210
.RAD50/ROLA/
.WORD 4
.ASCII/49/
.BLKW 134
```

```
.RAD50/BRA/
.BLKW 1
.WORD 5
.ASCII/20/
.RAD50/PULB/
.WORD 4
.ASCII/33/
.BLKW 20
.RAD50/PULA/
.WORD 4
.ASCII/32/
.BLKW 244
.RAD50/TPA/
.BLKW 1
.WORD 4
.ASCII/07/
.BLKW 40
.RAD50/BSR/
.BLKW 1
.WORD 5
.ASCII/8D/
.RAD50/SEC/
.BLKW 1
.WORD 4
.ASCII/0D/
.BLKW 24
.RAD50/SEI/
.BLKW 1
.WORD 4
.ASCII/0F/
.BLKW 34
.RAD50/BLS/
.BLKW 1
.WORD 5
.ASCII/23/
.BLKW 20
.RAD50/SEV/
.BLKW 1
.WORD 4
.ASCII/0B/
.BLKW 14
.RAD50/LDAA/
.WORD 7403
.ASCII/86/
.BLKW 20
.RAD50/LDAB/
.WORD 7403
.ASCII/C6/
.BLKW 4
.RAD50/RTI/
.BLKW 1
.WORD 4
.ASCII/3B/
.BLKW 44
.RAD50/LDS/
.BLKW 1
.WORD 107403
.ASCII/8E/
.BLKW 4
.RAD50/NAM/
.BLKW 1
.WORD 10
.WORD 5
.BLKW 10
.RAD50/LDX/
.BLKW 1
```

```
          .WORD 10/403
          .ASCII/CE/
          .BLKW 140
          .RAD50/FCB/
          .BLKW 1
          .WORD 10
          .WORD 3
          .BLKW 174
          .RAD50/WAI/
          .BLKW 1
          .WORD 4
          .ASCII/3E/
          .BLKW 4
          .RAD50/BVC/
          .BLKW 1
          .WORD 5
          .ASCII/28/
          .BLKW 60
BLANCO==40                          ;DESIGNAMOS EL CARACTER ESPACIO COMO
                                    ;"BLANCO"
TAB==11                             ;DESIGNAMOS EL CARACTER TAB COMO TAB
PAS2SW::        .WORD   0           ;BANDERA QUE INDICA SI ESTAMOS EN  EL
                                    ;SEGUNDO PASO.
TOKENS::        .BLKW   6           ;VECTOR PARA GUARDAR TOKENS
FLAGER::        .BLKW   1           ;MARCA EL ERROR QUE SE COMETE
LINEA  ::       .WORD   0           ;CONTADOR DE LINEA
PLC    ::       .WORD   0           ;CONTADOR DE LOCALIDADES DEL PRO
                                    ;GRAMA
TEMPLC::        .WORD   0           ;GUARDA TEMPORALMENTE EL PLC
BUFF   ::       .BLKB   80.         ;BUFFER DEL ARCHIVO DEL USUARIO
GUARD  ::       .WORD   0           ;GUARDA LA DIRECCION DEL ARCHIVO DEL
                                    ;USUARIO
PLCPRI::        .BLKW   1           ;GUARDA EL VALOR DEL PLC AL FINAL DEL
                                    ;PRIMER PASO
;
;
; ***************************************************************************
; EMPIEZA PRIMER PASO
; ***************************************************************************
;
START:
          JSR       PC,ABRE         ;ABRE EL ARCHIVO DEL USUARIO
          BR        AYUDA
BEGIN::   MOV       GUARD,R0
AYUDA:    JSR       PC,TRAER        ;TRAE UNA LINEA A EXAMINAR
          MOV       R0,GUARD
COPY::    JSR       PC,COPIA        ;COPIA LA LINEA EN EL ARCHIVO COPIA.TMP
          JSR       PC,SCANER       ;IDENTIFICA TOKENS
          JSR       PC,BARRE        ;QUITA LINEAS EN BLANCO
          MOV       R1,-(SP)        ;SALVA FIN DE LINEA
          MOV       R2,-(SP)        ;SALVA APUNTADOR DEL BUFFER
          MOV       #211.,FLAGER    ;FLAGER<=SIMBOLO INDEFINIDO
          JSR       PC,HASH         ;RECTIFICAMOS QUE EL OPERADOR ESTE EN
                                    ;TABLA
          MOV       (SP)+,R2        ;R2<= APUNTADOR DEL BUFFER
          MOV       (SP)+,R1        ;R1<=FIN DE LINEA
          JSR       PC,CALPLC       ;CALCULAMOS EN CUANTO SE INCREMENTA EL
                                    ;PLC Y COLOCAMOS ETIQUETA
CASFIN::  TST       ENEXP           ;ESTAMOS EXPANDIENDO UNA MACROLLAMADA?
          BEQ       BEGIN           ;NO VE A EXAMINAR LA LINEA
          CMP       LYNEX,LYNLAS    ;LLEGAMOS AL FINAL DEL CUERPO DE LA
                                    ;MACRO?
          BGT       FINEX           ;SI
          JSR       PC,INEX         ;EXPANDE LINEA
          BR        COPY            ;VE A EXAMINAR LINEA
FINEX:    CLR       ENEXP           ;APAGA BANDERA QUE INDICA EXPANSION DE
```

```
                                        ;MACROLLAMADA
        BR      BEGIN                   ;TRAE SIGUIENTE LINEA
; TERMINAMOS EL PRIMER PASO
FPAS1:: TST     USYMAC                  ;HUBO MACRODEFINICION?
        BEQ     $7                      ;NO
        DELET$  #FIBBMAC                ;BORRA ARCHIVO MACRO.TMP
$7:     TST     ERRORS
        BEQ     $10                     ;NO VE A SEGUNDO PASO
        JSR     PC,OPENER               ;CREA EL ARCHIVO FALTAS.DAT
        JSR     PC,ECRIRE               ;ESCRIBE ARCHIVO FALTAS.DAT
        DELET$  #FIBBNEW                ;BORRA EL ARCHIVO COPIA.TMP
        BR      FINAL
; ***************************************************************
; EMPIEZA EL SEGUNDO PASO
; ***************************************************************
$10:    MOV     PLC,PLCPRI              ;GUARDAR EL VALOR FINAL DEL PLC
        CLR     PLC                     ;LIMPIA PLC
        CLR     LINEA                   ;LIMPIA CONTADOR DE LINEA
        CLR     BANSW
        INC     PAS2SW                  ;PAS2SW<=1 ESTAMOS EN EL SEG. PASO
        JSR     PC,OPNEWB               ;ABRE EL ARCHIVO COPIA.TMP
PAS2::  JSR     PC,TRAENB               ;TRAE UNA LINEA
        INC     LINEA                   ;LINEA<=LINEA+1
        MOV     #NEWBUF,R2              ;R2<=APUNTADOR DEL BUFFER NEWBUF
        CMPB    (R2),#'*                ;TODA LA LINEA ES COMENTARIO
        BNE     $502                    ;NO
        JSR     PC,WRCOM                ;ESCRIBE LINEA ENSAMBLADA EN MOT80.TMP
        BR      PAS2                    ;CONTINUA CON EL SEGUNDO PASO.
; LIMPIA TOKENS
$502:   MOV     #TOKENS,R0
        CLR     (R0)
        CLR     2(R0)
        CLR     4(R0)
; ENCUENTRA OPERADOR
$503:   CMPB    (R2),#BLANCO            ;CARACTER=BLANCO?
        BEQ     $504
        CMPB    (R2),#TAB
        BEQ     $504
        INC     R2                      ;LEE CARACTER
        BR      $503
$504:   JSR     PC,INSTRU               ;TENEMOS OPERADOR
        MOV     R2,-(SP)
        MOV     R1,-(SP)
        MOV     #TOKENS,R0
        JSR     PC,HASAUX               ;LOCALIZA OPERADOR EN TABLA
        MOV     (SP)+,R1
        MOV     (SP)+,R2
        JSR     PC,SEGCPC               ;CALCULA PLC Y ESCRIBE LINEA ENSAMBLADA
        BR      PAS2
; FIN DEL ENSAMBLADOR
FINAL:: EXIT$S
        .END    START
```

```
        .TITLE  OPENER
;+
; PROCEDIMIENTO PARA CREAR EL ARCHIVO FALTAS.DAT
; QUE CONTIENE LA LISTA DE ERRORES COMETIDOS
; -
        .MCALL  FCSMC$,QIOW$C,PUT$S
        FCSMC$
        .ENABL  GBL
FON::   .BLKW   80.
        FSRSZ$  1
FDERR:: FDBDF$
        FDAT$A  R.SEQ,FD.CR,80.
        FDRC$A  FD.INS
        FDOP$A  3,,PONNAM
PONNAM: NMBLK$  FALTAS,DAT,1
;
;
;
OPENER::
        OPEN$W  #FDERR,#3,,,#PON,#80.
        RTS PC
;
;
        .PAGE
        .SBTTL  VIOLA
;+
; PROCEDIMIENTO PARA LLEVAR CUENTA DE LAS LINEAS CON ERROR Y EL TIPO DE
; ERROR.
; CREA EL VECTOR NOREAD.
; NOREAD(I)=NO. DE LINEA CON ERROR
; NOREAD(I+1)=TIPO DE ERROR DETECTADO
; -
ERRORS::.WORD   0
NOREAD::.BLKW   72.
TAIL:   .BLKW   1
;
;
;
VIOLA::
        INC     ERRORS
        MOV     TAIL,R3         ;R3<=APUNTA AL FINAL DE NOREAD(I)
        MOV     LINEA,NOREAD(R3);NOREAD(I)<=NO. DE LINEA CON ERROR
        ADD     #2,R3
        MOV     FLAGER,NOREAD(R3)  ;NOREAD(I+1)<=TIPO DE ERROR
        ADD     #2,R3
        MOV     R3,TAIL
        CMP     FLAGER,#221.    ;TABLA DE SIMBOLOS LLENA?
        BEQ     11$
        CMP     ERRORS,#35.     ;HAY 35 ERRORES?
        BNE     12$             ;NO VE A FIN
        INC     ERRORS
        MOV     LINEA,NOREAD(R3)
        ADD     #2,R3
        MOV     #224.,NOREAD(R3)
11$:    TST     PAS2SW
        BEQ     PARAD1
        JMP     FERMER
12$:    RTS     PC
PARAD1: JMP     FPAS1
;
;
        .SBTTL  ECRIRE
; +
```

```
; PROCEDIMIENTO QUE CREA EL ARCHIVO FALTAS.DAT QUE LISTA LOS ERRORES
; PRODUCIDOS EN EL PROGRAMA
; ENTRADAS:
; R0<=#PON
; R1<=ISTRING
; R2<=NOREAD
; SALIDAS:
; CREA EL ARCHIVO FALTAS.DAT
; -
        .MACRO HELPW N
LTR'N:  MOV     #MSG'N,R1
        JMP     $6
        .ENDM
; CADENA PARA MANDAR MENSAJE
ZX1:    .ASCII/   TUVISTE ERRORES/
        YX1=.-ZX1
        .EVEN
ZX2:    .ASCII/   LISTA EL ARCHIVO FALTAS.DAT/
        YX2=.-ZX2
        .EVEN
ISTRIN: .ASCIZ/%4SEN LINEA NO.%M ERRROR NO.%M/
        .EVEN
TEMPO:  .BLKW 1
;
;
;
ECRIRE::
        CLR     R5
        CLR     R3
        FDRC$R  #FDBNEW,#FD.RAN,#NEWBUF,#80.  ;DECLARA ARCHIVO WORK.TMP
        MOV     #NOREAD,R2      ;ESCRIBE
$1:     MOV     #PON,R0         ; UNA
        MOV     #ISTRIN,R1      ;  LINEA
        CALL    $EDMSG
2$:     PUT$S   #FDERR
        MOV     #PON,R4         ;LIMPIA
3$:     CLRB    (R4)+           ; EL VECTOR
        SOB     R1,3$           ;  PON
        CMP     -(R2),-(R2)     ;R2<=NO. DE LINEA CON ERROR
        OPEN$R  #FDBNEW         ;ABRE ARCHIVO WORK.TMP
        MOV     (R2),F.RCNM+2(R0)  ;INDICA LINEA DESEADA
        GET$    #FDBNEW         ;TRAE LINEA
        TST     (R2)+
        MOV     #80.,R1
        MOV     R1,TEMPO        ;SALVA R1
        CLOSE$  #FDBNEW         ;CERRRAMOS ARCHIVO WORK.TMP
        MOV     #NEWBUF,R4
        MOV     #PON,R0
4$:     MOVB    (R4)+,(R0)+     ;PON<=NEWBUF
        SOB     R1,4$
        PUT$S   #FDERR          ;ESCRIBE LINEA CON ERROR
        MOV     TEMPO,R1        ;REGRESA VALOR DE R1
        MOV     #PON,R0
        MOV     R0,R4
5$:     CLRB    (R4)+
        SOB     R1,5$
        MOV     (R2),R4         ;DE ACUERDO
        SUB     #201.,R4        ; AL TIPO
        ASL     R4              ;  DE ERROR
        JMP     @CALERR(R4)     ;   ESCRIBIMOS MENSAJE
CALERR: .WORD   LTR01,LTR02,LTR03,LTR04,LTR05
        .WORD   LTR06,LTR07,LTR08,LTR09,LTR10
        .WORD   LTR11,LTR12,LTR13,LTR14,LTR15
        .WORD   LTR16,LTR17,LTR18,LTR19,LTR20
        .WORD   LTR21,LTR22,LTR23,LTR24,LTR25
        .WORD   LTR26
```

```
; LISTA DE ERRORES
MSG01:  .ASCIZ/%4S201 ERROR EN DIRECTIVA NAM/
        .EVEN
        HELPW 01
MSG02:  .ASCIZ/%4S202 ETIQUETA O COD.OPERACION/
        .EVEN
        HELPW 02
MSG03:  .ASCIZ/%4S203 ERROR EN LA SENTENCIA/
        .EVEN
        HELPW 03
MSG04:  .ASCIZ/%4S204 ERROR DE SINTAXIS/
        .EVEN
        HELPW 04
MSG05:  .ASCIZ/%4S205 ERROR EN ETIQUETA/
        .EVEN
        HELPW 05
MSG06:  .ASCIZ/%4S206 SIMBOLO REDEFINIDO/
        .EVEN
        HELPW 06
MSG07:  .ASCIZ/%4S207 COD. DE OPERACION INDEFINIDO/
        .EVEN
        HELPW 07
MSG08:  .ASCIZ/%4S208 ERROR DE BRANCH/
        .EVEN
        HELPW 08
MSG09:  .ASCIZ/%4S209 MODO DE DIRECCIONAMIENTO ILEGAL/
        .EVEN
        HELPW 09
MSG10:  .ASCIZ/%4S210 BYTE OVERFLOW/
        .EVEN
        HELPW 10
MSG11:  .ASCIZ/%4S211 SIMBOLO INDEFINIDO/
        .EVEN
        HELPW 11
MSG12:  .ASCIZ/%4S212 ERROR EN EL OPERANDO DE UNA DIRECTIVA/
        .EVEN
        HELPW 12
MSG13:  .ASCIZ/%4S213 ERROR EN DIRECTIVA EQU/
        .EVEN
        HELPW 13
MSG14:  .ASCIZ/%4S214 ERRROR EN MACRODEFINICION/
        .EVEN
        HELPW 14
MSG15:  .ASCIZ/%4S215 ERROR EN ARGUMENTO MUDO/
        .EVEN
        HELPW 15
MSG16:  .ASCIZ/%4S216 MACRO INTERIOR/
        .EVEN
        HELPW 16
MSG17:  .ASCIZ/%4S217 MACRO ANIDADO/
        .EVEN
        HELPW 17
MSG18:  .ASCIZ/%4S218 WORD OVERFLOW/
        .EVEN
        HELPW 18
MSG19:  .ASCIZ/%4S219 FALTAN ARGUMENTOS REALES/
        .EVEN
        HELPW 19
MSG20:  .ASCIZ/%4S220 NO APARECE MEND/
        .EVEN
        HELPW 20
MSG21:  .ASCIZ/%4S221 TABLA DE SIMBOLOS LLENA/
        .EVEN
        HELPW 21
MSG22:  .ASCIZ/%4S222 ERROR DE FASE/
        .EVEN
```

```
        HELPW 22
MSG23:  .ASCIZ/%4S223 LA DIRECTIVA NO ADMITE ETIQUETA/
        .EVEN
        HELPW 23
MSG24:  .ASCIZ/%4S224 MAS DE 35 ERRORES/
        .EVEN
        HELPW 24
MSG25:   .ASCIZ/%4S225 ERROR EN MACROLLAMADA/
        .EVEN
        HELPW 25
MSG26:  .ASCIZ/%4S226 NO EXISTE END/
        .EVEN
        HELPW 26
$6:     CALL    $EDMSG          ;ESCRIBE
        PUT$S   #FDERR          ; MENSAJE
        INC     R5
        MOV     #PON,R4
7$:     CLRB    (R4)+
        SOB     R1,7$
        TST     (R2)+
        CMP     R5,ERRORS
        BEQ     10$
        JMP     $1
10$:    CLOSE$  #FDERR
        QIOW$C  IO.WLB,5,3,,,,<ZX1,YX1,40>
        QIOW$C  IO.WLB,5,4,,,,<ZX2,YX2,40>
        RTS     PC
        .END
PIP>TI:=WRIT3.MAC
```

```
        .TITLE  CREA
;+
;
; CONSTRUYE UN VECTOR QUE CONTIENE LOS OPERANDOS DE UNA EXPRESION EN SU
; FORMA BINARIA.
; ENTRADAS:
; R3<=DIRECCION DEL VECTOR CON LA CADENA A CONVERTIR
; R4<=TOTAL DE CARACTERES
; SALIDAS:
; STACK<=LLENA EL VECTOR STACK CON OPERANDOS Y OPERADORES
; IMAX<=INDICE MAXIMO DEL STACK
; SE ALTERAN R0-R5
; -
BOXTKN::.BLKB   18.             ;ESTO DA 16 CARACTERES POR SIMBOLO O
                                ;NUMERO
CONT:   .BLKW   1
FCOMP:: .BLKW   1
FFIN:   .BLKW   1
I::     .BLKW   1
IMAX::  .BLKW   1
LIGAS:: .BLKW   80.
STACK:: .BLKW   80.             ;VECTOR CON CAPACIDAD PARA ALMACENAR
                                ;NUMEROS Y SIMBOLOS
SWICH:  .WORD   0
        .ENABL  GBL
;
;
CREA::
; LIMPIAMOS VARIABLES A UTILIZAR
        MOV     #STACK,R0
        MOV     #LIGAS,R1
1$:     CLR     (R0)+
        CLR     (R1)+
        TST     (R0)
        BNE     1$
        CLR     SWICH
        CLR     IMAX
        CLR     FCOMP
        CLR     FFIN
        CLR     I
        MOV     #BOXTKN,R0
        MOV     #9.,R1
2$:     CLR     (R0)+
        SOB     R1,2$
        CLR     R5              ;R5<=INDICE DEL STACK
        ADD     R3,R4           ;R4<=FIN DEL VECTOR QUE CONTIENE LA
                                ;CADENA DE CARACTERES
WORK:   CLR     CONT            ;CONT<=CUENTA LOS CARACTERES DE UN TOKEN
        MOV     #BOXTKN,R2
        INC     I
        CMP     I,#75.          ;I>EL MAXIMO DEL STACK(I) ?
        BEQ     MARK
; IDENTIFICAMOS SI TENEMOS UNA OPERACION DE:
SEPARA: CMPB    (R3),#'*        ;MULTIPLICACION
        BEQ     GOOPER
        CMPB    (R3),#'+        ;SUMA
        BEQ     GOOPER
        CMPB    (R3),#'-        ;RESTA
        BEQ     GOOPER
        CMPB    (R3),#'/        ;DIVISION
        BEQ     GOOPER
; TENEMOS UN CARACTER
        MOVB    (R3)+,(R2)+     ;COPIA CARACTER A BOXTKN
```

```
        INC     CONT
        CMP     CONT,#10.
        BEQ     MARK
        CMP     R3,R4           ;HAY MAS CARACTERES EN NEXTKN?
        BLT     SEPARA
        INC     FFIN            ;ENCENDEMOS BANDERA PARA MARCAR FIN
GOOPER: TST     CONT            ;BOXTKN>0
        BEQ     GOSIGN
        MOVB    #BLANCO,(R2)    ;COLOCA BLANCO AL FINAL DE BOXTKN
        MOV     #BOXTKN,R0
        MOV     CONT,R2
        JSR     PC,STK          ;CONVIERTE EL TOKEN A BINARIO
        MOV     R1,STACK(R5)    ;Y LO COLOCA EN EL VECTOR STACK
        ADD     #2,R5           ;ACTUALIZA EL INDICE DEL STACK
        TST     FFIN            ;TERMINAMOS ?
        BNE     ALTO
; LIMPIAMOS BOXTKN
        MOV     #9.,R1
        MOV     #BOXTKN,R2
BOXCLR: TST     (R2)
        BEQ     4$
        CLR     (R2)+
        SOB     R1,BOXCLR
4$:     INC     I
GOSIGN: CMP     I,#1            ;I=1?
        BEQ     SPCIAL
        MOV     I,R2
        DEC     R2
        MOVB    (R3),STACK(R5)  ;COLOCA OPERADOR
        MOVB    R2,LIGAS(R5)
        INC     R5
        ADD     #2,R2
        MOVB    R2,LIGAS(R5)
        INC     R5
DEALTO: INC     R3              ;LEEMOS OTRO CARACTER
        CMP     R3,R4
        BLT     WORK
ALTO:   MOV     I,IMAX
        MOV     IMAX,R4
        RTS     PC
; PROCEDIMIENTO PARA DETECTAR EXPRESIONES QUE EMPIECEN CON EL SIGNO
; DEL PLC O CON EL SIGNO MENOS O MAS
SPCIAL: CMPB    (R3),#'*
        BNE     MENOS
        TST     SWICH           ;DETECTA EXPRESIONES COMO +* O -* ETC.
        BNE     MARK
        MOV     PLC,STACK(R5)
        ADD     #2,R5
        BR      DEALTO
MENOS:  CMPB    (R3),#'-        ;DETECTA EXPRESIONES COMO --1
        BNE     MAS
        TST     FCOMP
        BNE     MARK
        INC     FCOMP           ;FCOMP=1 INDICA PRIMER NUMERO NEGATIVO
        INC     SWICH
        CLR     I
        BR      DEALTO
;
MAS:    CMPB    (R3),#'+        ;DETECTA EXPRESIONES COMO ++,ETC.
        BNE     MARK
        TST     SWICH
        BNE     MARK
        INC     SWICH
        CLR     I
        BR      DEALTO
; SE DETECTO ERROR
```

```
MARK:   MOV     #204.,FLAGER
        JSR     PC,VIOLA
        TST     PAS2SW
        BNE     5$
        JMP     CASFIN
5$:     JMP     PAS2
;
;
;
        .PAGE
        .SBTTL  REVISA
;+
; REVISA LA SINTAXIS DE LAS EXPRESIONES
; ENTRADAS:
; R3<=#LIGAS
; R4<=IMAX
; -
REVISA::
        MOV     #LIGAS,R3
        TST     R4              ;SI IMAX=0 SE PRODUCE UN ERROR
        BEQ     BAD
        MOV     R3,R1           ;R1<=#LIGAS
; SE REVISAN LAS LOCALIDADES CON NUMEROS
        MOV     #1,R5           ;I<=1
10$:    TST     (R3)+           ;LIGAS=0?
        BNE     BAD             ;NO ERROR
        CMP     R5,R4
        BGE     20$
        ADD     #2,R5           ;I<=I+2
        ADD     #2,R3
        BR      10$
20$:    CMP     R5,R4           ;AL FINAL DEL STACK I=IMAX?
        BNE     BAD             ;NO ERROR
; SE REVISAN LAS LOCALIDADES CON SIGNO
        CMP     #1,R4           ;IMAX=1?
        BEQ     40$             ;SI ,TERMINAMOS
        MOV     R4,R2
        DEC     R2              ;IMAX<=IMAX-1
        CLR     R5              ;I<=0
30$:    ADD     #2,R1
        ADD     #2,R5           ;I<=I+2
        TST     (R1)+           ;SI LIGAS VACIAS ERROR
        BEQ     BAD
        CMP     R5,R2           ;I=IMAX-1
        BNE     30$
40$:    RTS     PC
; SE DETECTO ERROR
BAD:    MOV     #204.,FLAGER
        JSR     PC,VIOLA
        TST     PAS2SW
        BNE     50$
        JMP     CASFIN
50$:    JMP     PAS2
;
;
        .PAGE
        .SBTTL  EVALUA
; +
; EVALUA UNA EXPRESION
; ENTRADAS:
; R4<=IMAX
; R5<=#STACK
; SALIDAS:
; DA UN NUMERO EN DOBLE PRECISION
; R1<=PARTE ALTA DEL NUMERO
; R2<=PARTE BAJA DEL NUMERO
```

```
;
ALOW:   .BLKW   1
AHIG:   .BLKW   1
BLOW:   .BLKW   1
BHIG:   .BLKW   1
;
;
EVALUA::
        MOV     #STACK,R5
        CMP     R4,#1           ;IMAX=1?
        BEQ     $107
        ASL     R4              ;R4<=IMAX*2
        ADD     R5,R4           ;R4<=DIRECCION DONDE ACABA EL STACK
        CLR     AHIG
        MOV     (R5)+,ALOW      ;COLOCAMOS PRIMER OPERANDO CON DOBLE
                                ;PRECISION
$101:   CMP     R4,R5           ;SE LLEGO AL FINAL DEL STACK?
        BLE     $110            ;SI
        MOV     (R5)+,R3        ;MOVEMOS EL SIGNO DEL OPERADOR
        SUB     #52,R3
        ASL     R3
; DE ACUERDO AL SIGNO EFECTUAMOS OPERACIONES
        JMP     @FAIRE(R3)
FAIRE:  .WORD   MULTI,SUMA,FEO,RESTA,FEO,DIV
MULTI:  MOV     (R5)+,R0        ;MOVEMOS EL SIGNO DEL OPERADOR
        MOV     AHIG,R2
        MOV     ALOW,R3         ;MOVEMOS PRIMER OPERANDO
        BIT     #100000,R2      ;EXAMINAMOS SI HAY NUMERO NEGATIVO
        BEQ     102$            ;NO VE A 102$
        NEG     R2              ;CONVIERTE NUMERO NEGATIVO A POSITIVO
        NEG     R3
        SBC     R2
        MOV     #1,FCOMP        ;ENCIENDE BANDERA DE NUMERO NEGATIVO
102$:   CALL    $DMUL           ;EFECTUA MULTIPLICACION
        MOV     R0,AHIG
        MOV     R1,ALOW
        BR      $101
SUMA:   CLR     R2              ;HIGH<=0
        MOV     (R5)+,R3        ;TRAE SEGUNDO OPERANDO
        TST     FCOMP           ;FCOMP=0?
        BEQ     103$
        NEG     AHIG            ;TRANSFORMA PRIMER OPERANDO A NEGATIVO
        NEG     ALOW
        SBC     AHIG
        CLR     FCOMP
103$:   ADD     R3,ALOW         ;EFECTUA SUMA
        ADC     AHIG
        ADD     R2,AHIG
        BVS     FEO
        BR      $101
RESTA:  CLR     R2
        MOV     (R5)+,R3        ;TRAE SEGUNDO OPERANDO
        TST     FCOMP           ;FCOMP=0 ?
        BEQ     104$
        NEG     AHIG            ;TRANSFORMA PRIMER OPERANDO A NEGATIVO
        NEG     ALOW
        SBC     AHIG
        CLR     FCOMP
104$:   SUB     R3,ALOW         ;EFECTUA SUMA
        SBC     AHIG
        SUB     R2,AHIG
        BVS     FEO
        BR      $101
DIV:    MOV     (R5)+,R0        ;MUEVE DIVISOR
        MOV     AHIG,R1         ;MUEVE DIVIDENDO
        MOV     ALOW,R2
```

```
        BIT     #100000,R1      ;EXAMINA SI HAY NUMERO NEGATIVO
        BEQ     105$
        NEG     R1              ;CAMBIAMOS NUMERO NEGATIVO A POSITIVO
        NEG     R2                                                  211
        SBC     R1
        MOV     #1,FCOMP        ;FCOMP<=1
105$:   CALL    $DDIV           ;EFECTUA DIVISION
        MOV     R1,AHIG
        MOV     R2,ALOW
        BR      $101
; SE DETECTO ERROR
FEO:    MOV     #218.,FLAGER
        JSR     PC,VIOLA
        TST     PAS2SW          ;SEGUNDO PASO ?
        BNE     106$
        JMP     CASFIN
106$:   JMP     PAS2
$107:   CLR     R1              ;HIG
        MOV     (R5),R2         ;LOW
        BR      $111
$110:   MOV     AHIG,R1
        MOV     ALOW,R2
$111:   TST     FCOMP           ;NUMERO>0
        BEQ     112$            ;TRANSFORMA NUMERO A NEGATIVO
        NEG     R1
        NEG     R2
        SBC     R1
112$:   RTS     PC
;
;
        .PAGE
        .SBTTL STK
;+
; PROCEDIMIENTO PARA CAMBIAR CADA UNO DE LOS OPERANDOS DE UNA EXPRESION
; A SU FORMA BINARIA
; ENTRADAS:
; R0<=DIRECCION DEL VECTOR CON CADENA DE CARACTERES.
; AL FINAL DE LA CADENA VA UN BLANCO
; R2<=TOTAL DE CARCTERES
; SALIDAS:
; R1<=NUMERO EN FORMA BINARIA
;-
STK::
        JSR     R5,$SAVRG       ;SALVA R0-R5
        MOV     R0,R1
        DEC     R2
        MOV     #204.,FLAGER
        ADD     R2,R1           ;R1<=APUNTA AL FINAL DEL VECTOR
; EL NUMERO DE ACUERDO A SU PREFIJO ES:
        CMPB    (R0),#'!        ;DECIMAL
        BEQ     PREDEC
        CMPB    (R0),#'@        ;OCTAL
        BEQ     PREOCT
        CMPB    (R0),#'%        ;BINARIO
        BEQ     PREBI1
        CMPB    (R0),#'$        ;HEXADECIMAL
        BEQ     CALHEX
; ES SIMBOLO ?
        CMPB    (R0),#'A
        BLT     NUMBER
        CMPB    (R0),#'Z
        BGT     MAL1
; PROCEDIMIENTO PARA SIMBOLO
        MOV     R0,R3
        MOV     R0,R5
        ADD     #6,R5           ;R5<=TAMA$O MAXIMO QUE PERMITE UN
```

```
                                    ;UN SIMBOLO
        ADD     R2,R1
        CMP     R1,R5               ;TENEMOS MAS DE SEIS CARACTERES
        BLE     201$
        MOVB    #BLANCO,(R5)        ;COLOCA BLANCO AL FINAL DE LA CADENA
        MOV     #204.,FLAGER
201$:   ADD     #1,R3
        CMPB    (R3),#BLANCO        ;LLEGAMOS AL FINAL DE LA CADENA?
        BEQ     SALTA
        CMPB    (R3),#'Z
        BGT     MALO
        CMPB    (R3),#'0
        BLT     MALO
        CMPB    (R3),#'9
        BLE     201$
        CMPB    (R3),#'A
        BGE     201$
MAL1:   BR      MALO
PREBI1: JMP     PREBIN
CALHEX: JMP     PREHEX
SALTA:  MOV     #211.,FLAGER
        JSR     PC,HASAUX           ;APLICA HASH A SIMBOLO
        TST     R3                  ;MANDA MENSAJE DE ERROR
        BEQ     NOW                 ;SI R3=0
        CMP     X(R0),#9.           ;FUE UN SIMBOLO
        BNE     NOW
        ADD     #2,R0
        MOV     X(R0),R1            ;R1<=VALOR DEL NUMERO (PLC)
        JMP     $225
NOW:    TST     PAS2SW              ;PARA SABER SI LA INSTRUCCION
        BNE     MALO                ;ES MODO EXTENDIDO DIRECTO
        TST     BANSW
        BEQ     MALO
        MOV     #400,R1             ;COLOCAMOS UN NUMERO FICTICIO >255
        JMP     $225
; EL NUMERO DE ACUERDO A SU SUFIJO ES:
NUMBER:
        CMPB    (R1),#'D            ;OCTAL
        BEQ     SUFOCT
        CMPB    (R1),#'Q            ;OCTAL
        BEQ     SUFOCT
        CMPB    (R1),#'B            ;BINARIO
        BEQ     SUFBIN
        CMPB    (R1),#'H            ;HEXADECIMAL
        BEQ     SUFHEX
; PROCEDIMIENTO PARA NUMEROS DECIMALES
        INC     R2
        BR      CALDEC
PREDEC: ADD     #1,R0               ;POR TENER PREFIJO SE DESHECHA
CALDEC: TST     R2
        BEQ     MALO
        CMP     R2,#5
        BGE     AUXDEC
DEC:    CALL    $CDTB               ;CONVIERTE DE DECIMAL A BINARIO
        CMP     R2,#BLANCO          ;SINO ENCONTRO BLANCO, ERROR
        BNE     MALO
        JMP     $225
AUXDEC: CMPB    (R0),#'0            ;SI SON MAS DE CINCO CARACTERES
        BEQ     QUITAR              ;QUITAR CEROS
        CMP     R2,#5               ;OVERFLOW SI SON MAS DE CINCO CARACTERES
        BGT     MALO
        CMPB    (R0),#'6
        BLE     DEC
        BR      MALO
QUITAR: DEC     R2                  ;SE QUITAN LOS CEROS A LA IZQUIERDA
        ADD     #1,R0               ;DEL NUMERO
```

```
        BR      CALDEC
; CONVERTIMOS UN NUMERO DE OCTAL A BINARIO
SUFOCT: MOVB    #BLANCO,(R1)    ;POR TENER SUFIJO SE SUPRIME
        BR      $211
PREOCT: ADD     #1,R0           ;POR TENER PREFIJO SE SUPRIME
$211:   TST     R2              ;SE EVITA EL ERROR @  .
        BEQ     MALO
        CMP     R2,#6           ;LIMITE DE CARACTERES PERMITIDOS
        BGE     AUXOCT
OCT:    CALL    $COTB           ;CONVERTIMOS DE OCTAL A BINARIO
        CMP     R2,#BLANCO
        BNE     MALO
        JMP     $225
AUXOCT: CMPB    (R0),#'0        ;SUPRIME CEROS A LA IZQUIERDA
        BEQ     BORRA
        CMP     R2,#6           ;CARACTER>6
        BGT     MALO            ;SI OVERFLOW
        CMPB    (R0),#'1
        BEQ     OCT
; DETECCION DE ERRORES
MALO:   JSR     PC,VIOLA
        TST     PAS2SW
        BNE     212$
        JMP     CASFIN
212$:   JMP     PAS2
BORRA:  DEC     R2
        BR      PREOCT
; PROCEDIMIENTO PARA CONVERTIR LOS CARACTERES EN BINARIO A UN NUMERO
; EN BINARIO. SE ADMITEN HASTA 16 CARACTERES. HAY QUE CHECAR QUE SOLO
; SEAN CEROS Y UNOS.
SUFBIN: MOVB    #BLANCO,(R1)    ;POR TENER SUFIJO SE SUPRIME
        BR      $213
PREBIN: ADD     #1,R0           ;POR TENER PREFIJO SE SUPRIME
        TST     R2              ;EVITAMOS EL ERROR % .
        BEQ     MALO
$213:   CLR     R1
214$:   CLR     R3
        MOV     R2,R4           ;R4<=TOTAL DE CARACTERES
        MOVB    (R0)+,R3        ;PASA CARACTER LEIDO A R3
        CMP     #'0,R3          ;CHECA
        BGT     MALO            ; QUE
        CMP     #'1,R3          ;  SOLO SEAN
        BLT     MALO            ;    CEROS Y UNOS
        BIC     #177760,R3      ;PRENDEMOS BIT SI CAR=1
                                ;APAGAMOS BIT SI CAR=0
        TST     R3
        BEQ     217$            ;SI ES CERO NO HAY NADA QUE HACER
        DEC     R4
        TST     R4              ;YA ACABAMOS?
        BEQ     216$
215$:   ASL     R3              ;GENERAMOS
        SOB     R4,215$         ; EL
216$:   ADD     R3,R1           ;  NUMERO EN
217$:   SOB     R2,214$         ;    BINARIO
        JMP     $225
; CONVERTIMOS UN NUMERO HEXADECIMAL A BINARIO
SUFHEX: MOVB    #BLANCO,(R1)    ;SUPRIME SUFIJO
        BR      $220
PREHEX: ADD     #1,R0           ;SUPRIME PREFIJO
$220:!  TST     R2              ;EVITAMOS EL ERROR $
        BEQ     MALO
        CMP     R2,#4           ;SE ENCUENTRA EN EL LIMITE DE CARACTERES
        BGT     AUXHEX          ;PERMITIDOS ?
        CLR     R1
$221:   CLR     R3
        MOVB    (R0)+,R3        ;MUEVE CARACTER A R3
```

```
        MOV     R2,R5
        DEC     R5
HEXBIN::CMP     R3,#'0          ;REVISAMOS                      214
        BLT     MALO            ; QUE
        CMP     R3,#'F          ;  LOS
        BGT     MALO            ;   CARACTERES
        CMP     R3,#'9          ;    SEAN
        BLE     222$            ;     LOS
        CMP     R3,#'A          ;      PERMITIDOS
        BLT     MALO
        SUB     #7,R3           ;PARA CARACTERES A-F
222$:   BIC     #177760,R3
        TST     R5
        BEQ     223$
        ASL     R5
        ASL     R5              ;R5<=R5*4
        ASH     R5,R3           ;R3<=R3*R5*2
223$:   ADD     R3,R1           ;GENERAMOS EL NUMERO
        SOB     R2,$221         ;EN BINARIO
$225:   RTS     PC
AUXHEX: CMPB    (R0),#'0        ;SUPRIME CEROS A LA IZQUIERDA
        BNE     MALO
        DEC     R2
        BR      PREHEX
        .END
PIP>TI:=WRIT4.MAC
```

```
        .TITLE  OPMACRO
;+
; CREA EL ARCHIVO TEMPORAL MACRO.TMP DONDE SE GUARDA
; EL CUERPO DEL MACRO
;-
        .MCALL  FCSMC$
        FCSMC$                          ;LLAMA A TODOS LOS MACROS DEL FCS
        .ENABL  GBL
MACBUF::.BLKB   80.
; INICIALIZA FDB
FDBMAC::FDBDF$
        FDRC$A  ,MACBUF,80.             ;ACHIVO DE LONGITUD FIJA
        FDAT$A  R.FIX,,80.              ;CON 80 CARACTERES
        FDOP$A  3,,NAMMAC               ;ASOCIA EL ARCHIVO A LUN=3
NAMMAC: NMBLK$  MACRO,TMP,1             ;DECLARA EL NOMBRE DEL ARCHIVO
;
;
OPMAC:: INC     USYMAC                  ;INDICA QUE SE CREO EL ARCHIVO
        OPEN$W  #FDBMAC                 ;CREA EL ARCHIVO
        CLOSE$  #FDBMAC
        RTS     PC
;
;
        .PAGE
        .SBTTL  SBMAC
;+
; RUTINA QUE ACTUA EN CASO DE ENCONTRAR LA PSEUDO-INSTRUCCION
; MACRO. ALMACENA EL CUERPO DEL MACRO EN EL ARCHIVO MACRO.TMP
; Y MARCA LOS PARAMETROS MUDOS.
;-
APFILM::.BLKW   1                       ;APUNTADOR DE LINEAS DEL MACRONOMBRE
BINASC::.ASCII  /^/                     ;INDICA ARGUMENTO MUDO
        .BLKB   2
        .EVEN
USYMAC::.BLKW   1                       ;BANDERA PARA APERTURA DEL ARCHIVO
DUMMY:: .BLKW   1                       ;CONTADOR DE ARGUMENTOS MUDOS
INMAC:: .BLKW   1                       ;INDICA QUE ESTAMOS EN MACRODEFINICION
NOCOPY::.BLKW   1
;
;
SBMAC:: TST     USYMAC                  ;SI ES PRIMERA VEZ ABRIMOS EL ARCHIVO
        BEQ     3$
2$:     INC     INMAC                   ;ESTAMOS EN EL MACRO
        MOV     #TOKENS,R0              ;LIMPIEZA DE TOKENS
        CLR     (R0)
        CLR     2(R0)
        CLR     4(R0)
        JSR     PC,INSTRU               ;ESTE SCANER DEJA EN TOKENS MACRONOMBRE
        MOV     #214.,FLAGER            ;FLAGER<=NO ENCONTRE MACRONOMBRE
        TST     R3                      ;TCAR=0?
        BEQ     5$
        TST     R4                      ;HUBO ERROR AL ESCRIBIR MACRONOMBRE?
        BNE     5$
        MOV     #TOKENS,R0              ;HUBO ERROR EN EL PRIMER CARACTER DEL
                                        ;MACRONOMBRE ?
        CMPB    (R0),#'Z
        BGT     5$                      ;SI , ERROR.
        CMPB    (R0),#'A
        BLT     5$
        MOV     R2,-(SP)                ;SP<=APUNTADOR DEL BUFFER
        MOV     R1,-(SP)                ;SP<=FIN DEL BUFER
; PREPARATIVOS PARA MANDAR A TFIJA EL MACRONOMBRE
        INC     APFILM                  ;APFILM<=APFILM+1
```

```
                                ;INCREMENTAMOS APUNTADOR DEL ARCHIVO
                                ;MACBUF. APFILM<7777
        MOV     APFILM,FLAG     ;MANDAMOS A TABLA DONDE EMPIEZA EL
                                ;ARCHIVO DEL MACRO
        ASL     FLAG            ;DEJAMOS LIBRES
        ASL     FLAG            ; LOS ULTIMOS 4 BITS DE LA
        ASL     FLAG            ;   TERCERA PALABRA ASOCIADA
        ASL     FLAG            ;    AL MACRONOMBRE
        ADD     #2,FLAG         ;CODIGO DE MACRONOMBRE
        CLR     TEMPLC
        MOV     #214.,FLAGER    ;FLAGER<=SIMBOLO DE NOMBRE REDEFINIDO
        MOV     #TOKENS,R0
        JSR     PC,HASLAB       ;COLOCA MACRONOMBRE EN TABLA .
        MOV     R1,R0           ;R0<=DIRECCION DE HASH
        MOV     (SP)+,R1        ;APUNTADOR DEL BUFER
        MOV     (SP)+,R2        ;R2<=FIN DE LINEA
        TST     R3              ;INICADOR DE ERRORES
        BEQ     5$
; PREPARATIVOS PARA GUARDAR EN NEXTKN
; ARGUMENTOS MUDOS
        MOV     #NEXTKN,R5      ;LIMPIA NEXTKN
4$:     CLR     (R5)+
        TST     (R5)
        BNE     4$
        MOV     #NEXTKN,R5      ;GUARDA EN NEXTKN
        MOV     R0,(R5)+        ;LA DIRECCION DONDE ACABA MACRONOMBRE
        CLR     DUMMY
        BR      10$
3$:     JSR     PC,OPMAC        ;CREA EL ARCHIVO MACRO.TMP
        BR      2$
; PROCEDIMIENTO PREVIO A DETECCION DE ERRORES
5$:     TST     DUMMY
        BEQ     7$
        MOV     #NEXTKN,R5      ;BORRAMOS EL CONTENIDO DE NEXTKN
        ADD     #2,R5
6$:     TST     (R5)
        BEQ     7$
        MOV     (R5),R4         ;BORRAMOS ARGUMENTOS
        CLR     (R4)+           ;MUDOS CARGADOS EN TABLA
        CLR     (R4)+
        CLR     (R4)
        CLR     (R5)+
        BR      6$
; DETECTA ERROR
7$:     CLR     DUMMY
        INC     NOCOPY          ;NOCOPY<=BANDERA QUE ELIMINA EL CUERPO
                                ;DEL MACRO
        JSR     PC,VIOLA
        JMP     30$             ;REGRESA A EXAMINAR
; CARGAMOS ARGUMENTOS MUDOS EN TABLA Y COMO ESTOS NO DEBEN PERMANECER
; EN LA TABLA DESPUES DE ACABAR LA MACRODEFINICION HABRA QUE BORRARLOS
; POR  ELLO LA DIRECCION EN DONDE QUEDAN CARGADOS LA GUARDAMOS EN
; NEXTKN
10$:    MOV     #TOKENS,R0
        CLR     (R0)            ;LIMPIA TOKENS
        CLR     2(R0)
        CLR     4(R0)
        CLR     R4              ;CODIGO DE ERRORES
        JSR     PC,INSTRU       ;BUSCA ARGUMENTO MUDO
        TST     R3              ;SI R3=0 ES QUE NO HAY MAS CARACTERES O
                                ;PORQUE SE LLEGO A FIN DE LINEA
        BEQ     25$
        MOV     #TOKENS,R0
        MOV     #215.,FLAGER    ;ALGUN ARGUMENTO MUDO ESTA MAL ESCRITO
        CMPB    (R0),#'Z
        BGT     5$
```

```
        CMPB    (R0),#'A
        BLT     5$
        TST     R4
        BEQ     20$             ;SINO HUBO ERRORES ES QUE ENCONTRO EL
                                ; ULTIMO ARGUMENTO MUDO
        MOV     #215.,FLAGER
        CMPB    (R2),#',        ;EN ESTE CASO LA COMA ACTUA COMO
                                ;SEPARADOR
        BNE     5$
        DEC     R3              ;R3<=TCAR-1
        TST     R3              ;ERROR POR ESCRIBIR,,
        BEQ     5$
        INC     R2              ;LEEMOS UN CARACTER
        CMP     R2,R1           ;FIN DE LINEA?
        BGT     5$
        CMPB    (R2),#BLANCO
        BEQ     5$
        CMPB    (R2),#TAB
        BEQ     5$
        DEC     R2              ;VOLVEMOS R2 A SU ANTIGUO VALOR
; LOCALIZAMOS UN ARGUMENTO MUDO
20$:    INC     DUMMY
; PROTEJEMOS REGISTROS
        MOV     R1,-(SP)        ;R1<=FIN DEL BUFFER
        MOV     R2,-(SP)        ;R2<=APUNTADOR DEL CARCTER
        MOV     R4,-(SP)        ;R4=0 ,FUE EL ULTIMO ARGUMENTO
        MOV     R5,-(SP)        ;R5<=#DISPONIBLE EN NEXTKN
        MOV     #215.,FLAGER    ;EXISTE SIMBOLO IGUAL REDEFINIDO
        MOV     #12,FLAG        ;CODIGO DE ARGUMENTO MUDO
        MOV     DUMMY,TEMPLC
        MOV     #TOKENS,R0
        JSR     PC,HASLAB       ;COLOCA ARGUMENTO MUDO
        SUB     #6,R1           ;CALCULAMOS LA DIRECCION DE HASH
        ADD     #X,R1           ;R1<=DIRECCION DONDE ESTA ARG. MUDO
        MOV     (SP)+,R5
        MOV     R1,(R5)+        ;R5<=ALMACENA EN NEXTKN DONDE
                                ;ESTA ARG. MUDO .
; RESTAURA R4,R2 Y R1
        MOV     (SP)+,R4
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        TST     R3              ;R3<=CODIGO DE ERROR DE SUB. HASLAB
        BEQ     5$
        TST     R4              ;R4=0 INDICA ULTIMO ARG. MUDO
        BNE     10$
; TERMINAMOS DE EXAMINAR LINEA DE MACRODEF. MACRO NOMBRE MUDO1,MUDO2 ETC.
; TRABAJAMOS CON EL CUERPO DEL MACRO
25$:    DEC     APFILM          ;DECREMENTAMOS EL APUNTADOR DE LINEAS
                                ;DEL ARCHIVO MACRO.TMP
30$:    MOV     GUARD,R0
        JSR     PC,TRAER        ;TRAE UNA LINEA DEL USUARIO
        MOV     R0,GUARD
        JSR     PC,COPIA        ;COPIA LA LINEA EN EL VECTOR NEWFILE
        INC     LINEA           ;LINEA<=LINEA+1
        MOV     #MACBUF,R3      ;LIMPIAMOS MACBUF
        MOV     R3,R4
35$:    CLR     (R4)+
        TST     (R4)
        BNE     35$
        MOV     #BUFF,R2
        MOV     R1,R4           ;R1<=TCAR
40$:    MOVB    (R2)+,(R3)+     ;MACBUF<=BUFF
        SOB     R4,40$
        MOV     #MACBUF,R2      ;R2<=#MACBUF
        ADD     R2,R1
        DEC     R1              ;R1<=FIN DE LINEA
; BUSCAMOS MEND Y LOS CARACTERES MUDOS
```

```
;SI DUMMY=0 SOLO BUSCAMOS MEND
        DEC     R2
50$:    MOV     #TOKENS,R0      ;LIMPIA TOKENS                  218
        CLR     (R0)
        CLR     2(R0)
        CLR     4(R0)
        JSR     PC,INSTRU
        TST     R3              ;R3=0? FIN DE LINEA
        BEQ     70$
        TST     R4              ;ENCONTRO TOKEN CON DIFICULTAD
        BEQ     60$
        CLR     R4
        MOVB    (R2),R4
        CMP     R4,#'+          ;CARACTER =+,/,-,*?
        BEQ     55$
        CMP     R4,#52
        BLT     50$
        CMP     R4,#57          ;NO ES SEPARADOR ESPECIAL?
        BGT     50$
55$:    DEC     R3
        TST     R3
        BEQ     50$
; PARA CUANDO HAY TOKEN
60$:    MOV     R3,R4           ;R3<=TCAR
        MOV     R4,-(SP)        ;R4<=TCAR
        MOV     R2,-(SP)
        MOV     R1,-(SP)
        MOV     #TOKENS,R0
        JSR     PC,HASAUX       ;MANDAMOS A TABLA EL TOKEN
        MOV     (SP)+,R1
        MOV     (SP)+,R2
        MOV     (SP)+,R4
        TST     R3              ;EL TOKEN SE ENCUENTRA EN TABLA?
        BEQ     50$             ;NO
        MOV     X(R0),R5        ;TENEMOS MEND?
        CMP     R5,#7
        BEQ     100$
        TST     DUMMY           ;HAY ARG. MUDOS?
        BEQ     50$             ;NO
        BIC     #177760,R5
        MOV     #217.,FLAGER    ;FLAGER<=MACRO ANIDADO
        CMP     R5,#6           ;SE ENCONTRO DIRECTIVA MACRO?
        BNE     62$
        JMP     5$
62$:    MOV     #216.,FLAGER    ;FLAGER<=MACRO INTERIOR
        CMP     R5,#2           ;SE ENCONTRO MACRONOMBRE?
        BNE     63$             ;NO
        JMP     5$              ;SI
63$:    CMP     R5,#12          ;ES ARGUMENTO MUDO?
        BNE     50$             ;NO
        ADD     #2,R0
        MOV     X(R0),R5        ;R5<=NUMERO QUE INDICA EL ORDEN DEL
                                ;ARG. MUDO
; CONVERTIMOS EL NUMERO DE BINARIO A ASCII
        BIC     #177707,R5
        ADD     #60,R5          ;CONVERTIMOS LOS PRIMEROS 3 CARACTERES
        MOV     #BINASC,R3
        INC     R3
        MOVB    R5,(R3)+        ;(R3)<=MUEVE CARACTER ''NN
        MOV     X(R0),R5
        BIC     #177770,R5
        ADD     #60,R5
        MOVB    R5,(R3)
        MOV     #BINASC,R5
        CMP     R2,R1
        BNE     65$
```

```
        CMPB    (R2),#BLANCO
        BEQ     65$
        CMPB    (R2),#TAB
        BEQ     65$
        INC     R2
; PROCEDIMIENTO PARA INTERCAMBIAR ARGUMENTO MUDO POR ^NN
65$:    MOV     R4,TCAR         ;R4<=TOTAL DE CARACTERES
        CMP     TCAR,#3
        BEQ     66$
        BLT     67$
        SUB     TCAR,R2         ;TCAR>3
        MOV     #3,R4
        MOV     TCAR,R3
        SUB     #3,R3
        JSR     PC,ACHICA       ;ACHICA EL BUFFER
        JMP     50$
66$:    JSR     PC,CAMBIA       ;TCAR=3 ,INTERCAMBIA CARACTERES
        JMP     50$
67$:    MOV     #MACBUF,R4      ;TCAR<3
        MOV     #3,AUXCAR       ;AGRANDA EL BUFFER
        SUB     TCAR,AUXCAR
        MOV     R2,HELCAR
        SUB     TCAR,R2
        JSR     PC,DOBIG
        JMP     50$
; FIN DE LINEA
70$:    TST     NOCOPY          ;SUPRIME LA LINEA EN EL CUERPO DEL MACRO
        BNE     75$
        INC     APFILM          ;APFILM<=APFILM+1
                                ;UNA LINEA MAS EN EL CUERPO DEL MACRO
        MOV     #79.,R2         ;COPIA LA LINEA
; SE HACE LA LINEA DE LONGITUD FIJA
        ADD     #MACBUF,R2
        SUB     R1,R2
        BEQ     72$
        INC     R1
71$:    MOVB    #BLANCO,(R1)+
        SOB     R2,71$
72$:    OPEN$A  #FDBMAC         ;ABRE ARCHIVO MACRO.TMP
        PUT$    #FDBMAC         ;ESCRIBE LINEA
        CLOSE$  #FDBMAC         ;CIERRA ARCHIVO MACRO.TMP
75$:    JMP     30$
; PARA CUANDO TENEMOS MEND
100$:   CLR     INMAC           ;SIGNIFICA NO ESTAMOS EN MACRODEF.
        TST     NOCOPY
        BNE     104$
        MOV     #NEXTKN,R5
        MOV     (R5)+,R0
        MOV     APFILM,X(R0)    ;COLOCA APFILM EN MACRONOMBRE
                                ;QUE ESTA EN TABLA
104$:   CLR     NOCOPY
105$:   TST     DUMMY           ;HUBO ARGUMENTO MUDO?
        BEQ     110$            ;NO
; BORRA ARGUMENTOS MUDOS DE TABLA
        MOV     (R5),R4
        CLR     (R4)+           ;LIMPIA PALABRA 1 EN TABLA.
        CLR     (R4)+           ;LIMPIA PALABRA 2 EN TABLA.
        CLR     (R4)+           ;LIMPIA PALABRA 3 EN TABLA.
        CLR     (R4)            ;LIMPIA PALABRA 4 EN TABLA
        CLR     (R5)+           ;ACTUALIZA # DE NEXTKN
        DEC     DUMMY
        BR      105$
110$:   RTS     PC
;
;
        .PAGE
```

```
        .SBTTL  SBMNAM
;+
; RUTINA QUE ACTUA EN CASO DE ENCONTRAR UNA MACROLLAMADA      220
; NOS DICE DE CUANTAS LINEAS CONSTA EL CUERPO DEL MACRO.
; IDENTIFICA CADA ARG. REAL Y CUANTOS ARGUMENTOS REALES
; APARECEN.
;-
JAIARG: .BLKW   1               ;CUANTOS ARGUMENTOS REALES TENEMOS
ENEXP:: .BLKW   1               ;INDICA QUE ESTAMOS EXPANDIENDO
BOXCAR: .BLKB   80.             ;GUARDA CARACTERES
BOXVAR::.BLKW   42.             ;BOXVAR(I)<=CUANTOS CARACTERES HAY QUE
                                ;LEER ANTES DE TENER LOS DEL ARG. REAL
                                ;BOXVAR(I+1)<=DE CUANTOS CARACTERES ES
                                ;EL ARGUMENTO REAL
NOARG:  .BLKW   1
LYNEX:: .BLKW   1               ;LINEA A EXPANDIR DEL ARCHIVO MACRO.TMP
LYNLAS::.BLKW   1               ;ULTIMA LINEA A TRAER DEL ARCHIVO MACRO.
;
;
SBMNAM::MOV     #216.,FLAGER    ;FLAGER<=MACRO  INTERIOR
        TST     ENEXP           ;ESTAMOS EXPANDIENDO UNA MACROLLAMADA?
        BNE     $ERROR          ;SI ERROR
; QUITAMOS EL CODIGO DE MACRONOMBRE
        MOV     X(R0),LYNEX
        ASR     LYNEX
        ASR     LYNEX           ;TRAEMOS EL NUMERO DE LINEA DONDE
        ASR     LYNEX           ;EMPIEZA LA DEF DEL MACRO
        ASR     LYNEX
        ADD     #2,R0
        MOV     X(R0),LYNLAS    ;LYNLAS<=NO. DE LINEA DONDE TERMINA LA
                                ;DEFINICION DEL MACRO
        MOV     #225.,FLAGER    ;FLAGER<=ERROR EN MACROLLAMADA
        CMP     LYNEX,LYNLAS    ;LYNEX>LYNLAS ?
        BGT     $ERROR          ;SI, COMETIMOS ERROR
        MOV     #TOKENS+6,R0    ;HAY ETIQUETA EN LA MACROLLAMADA?
        TST     (R0)
        BEQ     277$            ;NO
        MOV     #206.,FLAGER    ;FLAGER<=SIMBOLO REDEFINIDO
        MOV     #9.,FLAG        ;FLAG<=CODIGO DE ETIQUETA
        MOV     R2,-(SP)
        MOV     R1,-(SP)
        JSR     PC,HASLAB       ;COLOCA ETIQUETA EN TABLA
        TST     R3              ;HUBO ERROR EN LA DEF DE ETIQUETA?
        BNE     $ERROR
        MOV     (SP)+,R1        ;R1<=FIN DE LINEA
        MOV     (SP)+,R2        ;R2<=APUNTADOR DEL BUFFER
277$:   INC     ENEXP           ;ENEXP<=1
; LIMPIA BOXVAR
300$:   MOV     #BOXVAR,R0
301$:   CLR     (R0)+
        TST     (R0)
        BNE     301$
; LIMPIA BOXCAR
        MOV     #BOXCAR,R5
302$:   CLR     (R5)+
        TST     (R5)
        BNE     302$
        CLR     R4              ;R4<=TOTAL DE ARG.
        CLR     JAIARG
        MOV     #BOXCAR,R5      ;R5<=#BOXCAR
        MOV     #BOXVAR,R0      ;R0<=#BOXVAR
        CMP     R2,R1           ;ES FIN DE LINEA?
        BGE     $325            ;SI, TERMINAMOS
        INC     R2              ;LEE CARACTER
        CMPB    (R2),#BLANCO    ;CARACTER=BLANCO?
        BEQ     $325
```

```
        CMPB    (R2),#TAB       ; O A TAB?
        BEQ     $325
        DEC     R2
        CLR     R3              ;R3<=TOTAL DE CARACT DE UN ARGUMENTO
303$:   INC     R4              ;R4<=TENEMOS UN ARG MAS
304$:   INC     R2              ;LEE CARACTER
        CMP     R2,R1           ;FIN DE LINEA
        BGT     315$
; TENEMOS' SEPARADOR
        CMPB    (R2),#BLANCO
        BEQ     320$
        CMPB    (R2),#TAB
        BEQ     320$
        CMPB    (R2),#',
        BEQ     310$            ;VE A CASO ESPECIAL
        INC     R3              ;R3<=TCAR+1
        MOVB    (R2),(R5)+      ;BOXCAR<=BUFF
        BR      304$
; PARA CUANDO EL SEPARADOR IGUAL A COMA
310$:   TST     R3              ;PARA EL PRIMER ARGUMENTO
        BNE     315$
        INC     R3              ;ARG. REAL + 1
        MOVB    #BLANCO,(R5)+   ;PON BLANCO EN ARG. REAL
315$:   TST     (R0)+           ;SALTA UNA PALABRA DE BOXVAR
        MOV     R3,(R0)+        ;BOXVAR(I+1)<=TOTAL DE CARACTERES
        CLR     R3              ;R3<=0
        CMP     R2,R1           ;FIN DE LINEA?
        BLE     303$            ;REGRESA A EXAMINAR EL RESTO DE LA LINEA
320$:   TST     R4              ;NO HAY ARGUMENTOS?
        BEQ     $325            ;VE A FIN
        TST     R3
        BNE     315$
        CLR     R3              ;R3<=LUGAR PARA EFECTUAR SUMA
        CLR     R5              ;R5<=CUENTA CUANTOS CARACTERES PRECEDEN
                                ;A CADA ARGUMENTO
        MOV     #BOXVAR,R2      ;LA PRIMERA PALABRA CONTIENE CEROS
        ADD     #2,R2           ;YA QUE ESTE ARG. NO PRECEDE A NINGUNO
; VAMOS A COLOCAR EL NO. DE CARACTERES QUE PRECEDEN A CADA ARGUMENTO EN
; LA PALABRA DEJADA EN CEROS. AQUI YA NO IMPORTA R2.
323$:   INC     R5
        CMP     R5,R4
        BEQ     $325
        ADD     (R2)+,R3
        MOV     R3,(R2)+
        BR      323$
; SE DETECTO ERROR
$ERROR: JSR     PC,VIOLA
        RTS     PC
; LIMPIAMOS BUFFER
$325:   MOV     #BUFF,R2
327$:   CLR     (R2)+
        CMP     R2,R1
        BLE     327$
        MOV     R4,JAIARG       ;JAIARG<=TOTAL DE ARGUMENTOS REALES
        JSR     PC,INEX         ;EXPANDE LA PRIMERA LINEA
        JMP     COPY            ;COPIA LA LINEA Y CONTINUA EL PRIMER PAS
;
;
        .PAGE
        .SBTTL  INEX
;+
; RUTINA QUE SIRVE PARA EXPANDIR LAS MACROLLAMADAS
; CAMBIA LOS ARGUMENTOS MUDOS POR LOS REALES
;
;-
BUFF:   .BLKB   80.
```

```
HEC1:   .WORD   3
;
;                                                              222
INEX::
; LIMPIA MACBUF Y BUFF
        MOV     #MACBUF,R5
        MOV     #BUFF,R2
        MOV     #80.,R3
330$:   CLRB    (R5)+
        CLRB    (R2)+
        SOB     R3,330$
        FDRC$R  #FDBMAC,#FD.RAN,#MACBUF,#80.    ;DECLARAMOS ARCHIVO
                                                ;RANDOM MACRO.TMP
        OPEN$R  #FDBMAC         ;ABRIMOS ARCHIVO MACRO.TMP
        MOV     LYNEX,F.RCNM+2(R0) ;PEDIMOS LA LINEA DESEADA
        GET$    #FDBMAC         ;TRAE LA LINEA
        MOV     F.NRBD(R0),R1   ;R1<=TCAR LEIDOS
        CLOSE$  #FDBMAC         ;CIERRA ARCHIVO MACRO.TMP
        MOV     #MACBUF,R3
; BUFF<=MACBUFF
        MOV     #BUFF,R2
        MOV     R1,R5
331$:   MOVB    (R3)+,(R2)+
        SOB     R5,331$
        MOV     #BUFF,R2        ;R2<=BUFF
        ADD     R2,R1
        DEC     R1              ;R1<=FIN DE LINEA
        TST     JAIARG          ;HAY ARGUMENTOS MUDOS?
        BEQ     $FIN            ;NO, VE A $FIN
$335:   CMPB    (R2)+,#'^       ;CARACTER=^ ?
        BEQ     $340            ;SI, LOCALIZA ARG. MUDO
        CMP     R2,R1           ;FIN DE LINEA?
        BLE     $335            ;NO, SIGUE BUSCANDO ARG. MUDO
$FIN:   INC     LYNEX           ;LYNEX<=UNA LINEA MAS EXPANDIDA
        INC     R1
        MOV     #BUFF,R2
        SUB     R2,R1           ;R1<=TCAR EN LA LINEA
        RTS     PC
; REMPLAZA ARGUMENTO MUDO POR ARGUMENTO REAL
$340:   MOV     R1,-(SP)        ;SALVA FIN DE LINEA
; LIMPIA BOXTKN,R0
        MOV     #BOXTKN,R0
        MOV     R0,R3
        MOV     #9.,R5
341$:   CLR     (R3)+
        SOB     R5,341$
        MOV     R0,R5           ;BOXTKN<=BUFF
        MOVB    (R2)+,(R5)+     ;SOLO
        MOVB    (R2)+,(R5)+     ; TRES CARACTERES
        MOVB    #BLANCO,(R5)    ;  AL FINAL VA BLANCO
        MOV     R2,-(SP)
        CALL    $COTB           ;CONVIERTE DE ASCII A BINARIO
        MOV     R1,R3           ;R3<=VALOR EN BINARIO
        MOV     (SP)+,R2
        MOV     (SP)+,R1
; TRAE EL ARGUMENTO REAL CORRESPONDIENTE
; CON R3 LOCALIZAMOS EN BOXVAR LA VARAIABLE
        DEC     R3
        ASL     R3
        ASL     R3              ;R3<=[I*4]
        MOV     BOXVAR(R3),R5   ;R5<=NUMERO DE CARACTERES A LEER EN
                                ;BOXCAR ANTES DE LLEGAR A LOS DEL
                                ;ACTUAL ARGUMENTO
        ADD     #2,R3
        MOV     BOXVAR(R3),R4   ;R4<=TCAR DEL ARGUMENTO
        MOV     #219.,FLAGER    ;FLAGER<=FALTAN ARGUMENTOS REALES
```

```
        TST     R4                  ;TCAR=0?
        BEQ     $ERR                ;SI, ERROR
;SALTAMOS CARACTERES QUE PRECEDEN A NUESTRO ARG. REAL          223
        MOV     #BOXCAR,R0
        TST     R5
        BEQ     345$
344$:   INC     R0
        SOB     R5,344$
; LIMPIA BOXTKN
345$:   MOV     #BOXTKN,R3
        MOV     #9.,R5
346$:   CLR     (R3)+
        SOB     R5,346$
; TRAE ARGUMENTO REAL CORRESPONDIENTE EN BOXTKN
        MOV     #BOXTKN,R3
        MOV     R4,TCAR
        MOV     R4,R5
347$:   MOVB    (R0)+,(R3)+
        SOB     R5,347$
; CUANTOS CARACTERES OCUPA EL ARGUMENTO REAL?
        CMP     TCAR,#3
        BEQ     350$
        BLT     355$
; TCAR>3
        MOV     #BOXTKN,R5
        MOV     #BUFF,R4
        MOV     TCAR,AUXCAR
        SUB     #3,AUXCAR
        MOV     R2,HELCAR
        SUB     #3,R2
        JSR     PC,DOBIG1           ;AGRANDA EL BUFFER
        MOV     #BUFF,R2
        JMP     $335
; TCAR=3
350$:   MOV     #BOXTKN,R5
        JSR     PC,CAMBIA           ;INTERCAMBIA CARACTERES
        MOV     #BUFF,R2
        JMP     $335
; TCAR<3
355$:   MOV     #BOXTKN,R5
        MOV     #3,R3
        SUB     #3,R2
        SUB     TCAR,R3
        JSR     PC,ACHICA           ;ACHICA EL BUFFER
        MOV     #BUFF,R2
        JMP     $335
; DETECTA ERROR
$ERR:   JSR     PC,VIOLA
        INC     R1
        MOV     #BUFF,R2
        SUB     R2,R1
        JSR     PC,COPIA            ;COPIA LINEA EN EL ARCHIVO COPIA.TMP
        INC     LYNEX               ;LYNEX<=LYNEX+1
        JMP     CASFIN              ;VE A PROGRAMA PRINCIPAL
;
;
        .PAGE
        .SBTTL  CAMBIA
;+
; SE UTILIZA PARA INTERCAMBIAR 3 CARACTERES UN BUFFER
;-
CAMBIA::SUB     TCAR,R2
$2000:  MOVB    (R5)+,(R2)+
        DEC     TCAR
        BNE     $2000
        RTS     PC
```

```
;
;
        .PAGE
        .SBTTL  DOBIG1
;+
; AGRANDA EL BUFFER
;-
TCAR::  .BLKW   1               ;NUMERO DE CARACTERES A INSETAR EN LA
                                ;CADENA
AUXCAR::.BLKW   1               ;TOTAL DE CARACTERES A LOS QUE HAY QUE
                                ;HACER ESPACIO
HELCAR::.BLKW   1               ;DIRECCION A PARTIR DE LA CUAL HAY QUE
                                ;EFECTUAR ROTACIONES
VAR1:   .BLKW   1
;
;
DOBIG::ADD      #79.,R4
       MOV      #3,TCAR
       BR       $2400
DOBIG1::
       ADD      #79.,R4         ;QUITAMOS
       MOV      R4,VAR1         ; BLANCOS DEL FINAL
2300$: CMPB     (R4),#BLANCO
       BNE      2350$
       CLRB     (R4)
       DEC      R4
       BR       2300$
2350$: MOV      R4,R1
       MOV      VAR1,R4
$2400: CMP      R1,R4
       BEQ      2710$
       DEC      HELCAR
       CMP      R1,HELCAR
       BEQ      2720$
       INC      HELCAR
2500$: INC      R1
       CMP      R4,R1
       BLE      2710$
       MOV      R1,R0
       MOV      R1,R3
2600$: MOVB     -(R3),(R0)
       DEC      R0
       CMP      R3,HELCAR
       BGT      2600$
       DEC      AUXCAR
       INC      HELCAR
       TST      AUXCAR
       BNE      2500$
2700$: MOVB     (R5)+,(R2)+
       DEC      TCAR
       TST      TCAR
       BNE      2700$
2710$: RTS      PC
2720$: INC      R1
       CMP      R1,R4
       BGT      2710$
       DEC      AUXCAR
       TST      AUXCAR
       BNE      2720$
       BR       2700$
;
;
        .PAGE
        .SBTTL  ACHICA
;+
; ACHICA EL BUFFER
```

```
; ENTRADAS:
; R1<=FIN DEL BUFFER
; R2<=APUNTADOR
; R3<=NO. DE CARACTERES QUE FALTAN POR QUITAR
; R4<=NUMERO DE CARACTERES A MOVER
; R5<=APUNTADOR DONDE VIENEN LOS CARACTERES A INTERCAMBIAR.
;-
ACHICA::MOVB    (R5)+,(R2)+
        SOB     R4,ACHICA
2150$:  CMPB    R2,R1
        BEQ     2250$
        MOV     R2,R5
2200$:  ADD     #2,R5
        MOVB    -(R5),-(R5)
        INC     R5
        CMP     R5,R1
        BNE     2200$
2250$:  CLRB    (R1)
        DEC     R1
        DEC     R3
        BNE     2150$
        RTS     PC
        .END
PIP>TI:=WRITS.MAC
```

```
>PIP TI:=WRITS.MAC



        .TITLE  OPNEWB
;+
; PROCEDIMIENTO PARA CREAR EL ARCHIVO MOT80.TMP
; QUE CONTIENE LA LISTA DEL PROGRAMA ENSAMBLADO
;-
        .MCALL  GTIM$C,FCSMC$,PUT$S,GET$S,QIOW$C
        FCSMC$                          ;LLAMA TODOS LOS MACROS DEL FCS
IBTIME: .ASCIZ/%F%20SFECHA:%2S%Y%4SHORA:%2S%3Z/
        .EVEN
PON2::  .BLKB   82.
ARG:
TIMBUF: .BLKW   8.
ZX4:    .ASCII/     ** ERRORES = 0 **/
        YX4=.-ZX4
        .EVEN
ZX5:    .ASCII/     ** LISTA EL ARCHIVO MOT80.TMP **/
        YX5=.-ZX5
        .EVEN
FDBENS::FDBDF$
        FDAT$A  R.SEQ,FD.CR,80. ;SECCION DE ATRIBUTOS
        FDRC$A  FD.INS,PON2,80. ;SECCION DE REGISTRO DE ACCESO
        FDOP$A  1,,ENSNAM       ;SECCION DE APERTURA DEL ARCHIVO
ENSNAM: NMBLK$  MOT80,TMP,1     ;NOMBRE DEL ARCHIVO
        .ENABL  GBL
;
;
OPNEWB::
        OPEN$R  #FDBNEW         ;ABRE EL ARCHIVO WORK.TMP (LEE)
        OPEN$W  #FDBENS         ;ABRE EL ARCHIVO MOT80.TMP (ESCRIBE)
        GTIM$C  TIMBUF          ;ESCRIBE
        MOV     #PON2,R0        ; LA
        MOV     #IBTIME,R1      ;   FECHA
        MOV     #ARG,R2         ;    Y
        CALL    $EDMSG          ;     LA
        PUT$S   #FDBENS         ;      HORA
        CLOSE$  #FDBENS         ;CIERRA EL ARCHIVO MOT80.TMP
        RTS     PC
;
;
        .PAGE
        .SBTTL  TRAENB
;+
; PROPORCIONA UNA LINEA DEL ARCHIVO COPIA.TMP
;-
TRAENB::
        MOV     #NEWBUF,R5      ;LIMPIA
        MOV     #40.,R3         ; EL
1$:     MOV     #20040,(R5)+    ;   BUFFER
        SOB     R3,1$           ;    NEWBUF
        GET$S   #FDBNEW,,,FERMER;TRAE UNA LINEA
        MOV     F.NRBD(R0),R1   ;R1<=TOTAL DE CARACTERES LEIDOS
        MOV     #NEWBUF,R2
        ADD     R2,R1
        DEC     R1              ;R1<=APUNTA AL FINAL DE NEWBUF
        RTS     PC
FERMER::CLOSE$  #FDBNEW         ;CIERRA COPIA.TMP
        TST     IHAVE
        BNE     $6
```

```
        MOV     #226.,FLAGER    ;FLAGER<=NO HAY ERROR
        BR      $5
$6:     MOV     #222.,FLAGER    ;FLAGER<=ERROR DE FASE          227
        CMP     PLC,PLCPRI
        BEQ     $4              ;CORRECTO
$5:     JSR     PC,VIOLA
$4:     TST     ERRORS          ;SI HAY ERRORES
        BEQ     2$              ; LOS
        JSR     PC,OPENER       ;  ESCRIBE
        JSR     PC,ECRIRE
        DELET$  #FDBENS         ;BORRA EL ARCHIVO MOT80.TMP
        BR      $3
2$:     JSR     PC,TSIMBO       ;ESCRIBE LA TABLA DE SIMBOLOS
        QIOW$C  IO.WLB,5,3,,,,<ZX4,YX4,40>
        QIOW$C  IO.WLB,5,4,,,,<ZX5,YX5,40>
$3:     DELET$  #FDBNEW         ;BORRA EL ARCHIVO COPIA.TMP
        JMP     FINAL           ;SE TERMINO EL SEGUNDO PASO
;
;
        .PAGE
        .SBTTL  UTILFO
;+
; FORMATOS DE ESCRITURA PARA COLUMNAS 1-19
;-
ISF2:   .ASCIZ/%6S%9A%64S/
        .EVEN
ISTINM: .ASCIZ/%80A/
        .EVEN
;
;
UTILFO::
        MOV     R1,-(SP)
        MOV     #ARGBL1,R0
        MOV     LINEA,R1        ;R1<=NO. DE LINEA
        MOV     #1,R2           ;PARA NO SUPRIMIR LOS CEROS
        MOV     #SALPLC,R3
        MOV     #WAEQU,R4
        CALL    $CBDMG          ;ESCRIBE:
        MOV     #NEWBUF,R2      ;COLUMNAS 1-5
        MOVB    #BLANCO,(R0)+   ;COLUMNA 6
        MOV     (R3)+,(R0)+     ;COL. 7-8
        MOV     (R3),(R0)+      ;COL. 9-10
        MOVB    #BLANCO,(R0)+   ;COL. 11
        MOVB    (R4)+,(R0)+     ;COL. 12
        MOVB    (R4)+,(R0)+     ;COL. 13
        MOVB    (R4)+,(R0)+     ;COL. 14
        MOVB    (R4),(R0)+      ;COL. 15
        MOVB    #BLANCO,(R0)+   ;COL. 16
        MOV     #20040,(R0)+    ;COL. 17-18
        MOVB    #BLANCO,(R0)+   ;COL 19
        MOV     (SP)+,R1
        JSR     PC,FOR2         ;COL. 20-41
        JSR     PC,LASFRM       ;COL.42-80
        RTS     PC
;
;


        .PAGE
        .SBTTL  UTILF2
;+
; FORMATOS DE SALIDA PARA CASOS EN QUE UNA INSTRUCCION PRODUCE MAS DE
; UNA LINEA.
;-
UTILF2::
        MOV     #ARGBL1,R0
        MOV     R0,ARGBLK
```

```
        MOV     #SALPLC ,R3
        MOV     #WAEQU,R4       ;ESCRIBE:
        MOV     (R3)+,(R0)+     ;COL. 7-8
        MOV     (R3),(R0)+      ;COL. 9-10
        MOVB    #BLANCO,(R0)+   ;COL. 11
        MOVB    (R4)+,(R0)+     ;COL. 12
        MOVB    (R4)+,(R0)+     ;COL. 13
        MOVB    (R4)+,(R0)+     ;COL. 14
        MOVB    (R4),(R0)       ;COL. 15
; ESCRIBE UNA LINEA EN EL ARCHIVO MOT80.TMP
        MOV     #PON2,R0
        MOV     #ISF2,R1
        MOV     #ARGBLK,R2
        CALL    $EDMSG
        OPEN$A  #FDBENS         ;ABRE ARCHIVO MOT80.TMP
        PUT$S   #FDBENS
        CLOSE$  #FDBENS         ;CIERRA ARCHIVO MOT80.TMP
        ADD     NBYTES,PLC      ;ACTUALIZA EL PLC
        RTS     PC
;
;       .PAGE
        .SBTTL LASFRM
;+
; FORMATO DE SALIDA PARA LAS COLUMNAS 42-80
; ENTRADAS:
; R0<=POINTER DE ARGBLK ACTUALIZADO
; R1<=FIN DE LINEA
; R2<=POINTER DE LECTURA
; R4<=AREA DE TRABAJO
;-
LASFRM::
        CLR     R3
        MOV     #8.,R4          ;R4<=CAMPO DE OPERACION
7$:     INC     R2              ;LEEMOS CARACTER
        CMP     R2,R1           ;FIN DE LINEA?
        BGT     20$
; BUSCAMOS SEPARADOR
        CMPB    (R2),#BLANCO
        BEQ     7$
        CMPB    (R2),#TAB
        BEQ     7$
10$:    INC     R3              ;TCAR<=TCAR+1
        MOVB    (R2)+,(R0)+     ;ARGBLK<=NEWBUF
        CMP     R2,R1           ;FIN DE LINEA?
        BGT     20$
        CMPB    (R2),#BLANCO
        BEQ     20$
        CMPB    (R2),#TAB
        BEQ     20$
        BR      10$
20$:    INC     R2
        CMP     R4,R3           ;R4<=8-R3
        BLT     23$             ;MAS DE 8 CARACTERES?
        BEQ     22$             ;TCAR=8?
; CUANDO SON MENOS DE OCHO CARACTERES LLENAMOS CON BLANCO LO QUE FALTA
        SUB     R3,R4
21$:    MOVB    #BLANCO,(R0)+
        SOB     R4,21$
22$:    MOVB    #BLANCO,(R0)+   ;COL 49
        CLR     R3              ;PARA EL CASO R3=R4
23$:    MOV     #38.,R4
        TST     R3
        BEQ     24$
        SUB     #9.,R3
        TST     R3
        BEQ     24$
```

```
        SUB     R3,R4
        CLR     R3
24$:    CMP     R2,R1           ;FIN DE LINEA?                  229
        BGT     25$
        INC     R3              ;TCAR<=TCAR+1
        MOVB    (R2)+,(R0)+     ;ARGBL1(I)<=NEWBUF(I)
        CMP     R3,R4
        BEQ     27$
        BR      24$
25$:    TST     R4
        BLE     27$
        SUB     R3,R4
26$:    MOVB    #BLANCO,(R0)+   ;LLENA CON BLANCOS
        SOB     R4,26$
; ESCRIBE LINEA
27$:    MOV     #ARGBL1,ARGBLK
        MOV     #PON2,R0
        MOV     #ISTINM,R1
        MOV     #ARGBLK,R2
        CALL    $EDMSG
        OPEN$A  #FDBENS         ;ABRE ARCHIVO
        PUT$S   #FDBENS
        CLOSE$  #FDBENS         ;CIERRA ARCHIVO MOT80.TMP
        ADD     NBYTES,PLC      ;ACTUALIZA PLC
        RTS     PC
;
;
        .PAGE
        .SBTTL WRCOM
;+
; FORMATO PARA ESCRIBIR LINEAS CON COMENTARIOS
; ENTRADA:
; R1<=FIN DE LINEA
;-
ISTCOM::.ASCIZ/%U%14S%60A/
        .EVEN
ARGBL1::.BLKB   86.
;
;
WRCOM::
        MOV     #60.,R4         ;R4<=MAXIMO NO. DE CARACTERES A ESCRIBIR
        CLR     R3
        MOV     #NEWBUF,R2
        MOV     #ARGBLK,R0
        MOV     LINEA,(R0)+     ;ESCRIBE NUMERO DE LINEA
        MOV     #ARGBL1,(R0)
        MOV     #ARGBL1,R0
300$:   INC     R3              ;R3<=CUENTA CARACTERES
        MOVB    (R2)+,(R0)+     ;ARGBLK(I)<=NEWBUF(J)
        CMP     R2,R1           ;YA COPIAMOS TODO NEWBUF?
        BGT     301$
        CMP     R3,R4
        BNE     300$            ;TRUNCA POR HABER + DE 60 CARACTERES
301$:   SUB     R3,R4           ;R4<=60-TOTAL DE CARACTERES
        TST     R4              ;60 CARACTERES A ARGBLK?
        BEQ     303$            ;SI, TERMINAMOS
302$:   MOVB    #BLANCO,(R0)+   ;MUEVE BLANCOS HASTA COMPLETAR 60
        SOB     R4,302$         ;CARACTERES
; ESCRIBIMOS LINEA
303$:   MOV     #PON2,R0        ;BUFFER DE SALIDA
        MOV     #ISTCOM,R1
        MOV     #ARGBLK,R2
        CALL    $EDMSG
        OPEN$A  #FDBENS         ;ABRE ARCHIVO MOT80.TMP
        PUT$S   #FDBENS
        CLOSE$  #FDBENS         ;CIERRA ARCHIVO MOT80.TMP
```

```
        RTS     PC
;
;                                                                    230
        .PAGE
        .SBTTL  FORMAT
;+
; FORMATO DE ESCRITURA DE LAS COLUMNAS 1-33
; ENTRADAS:
; R1<=;INEA
; R3<=# DE SALPLPC
; R4<=# DE CODIGO
; R5<=PRIMERO Y SEGUNDO BYTE DEL OPERANDO
;-
FORMAT::
        MOV     R1,-(SP)        ;SALVA FIN DE LINEA
        MOV     #ARGBL1,R0      ;ESCRIBE
        MOV     LINEA,R1        ;COLUMNAS 1-5
        MOV     #NEWBUF,R2
        MOV     #SALPLC,R3
        MOV     #CODIGO,R4
        MOV     #FIASEC,R5
        CALL    $CBDMG          ;CAMBIA DE BINARIO A DECIMAL
        MOV     #NEWBUF,R2
        MOVB    #BLANCO,(R0)+   ;COLUMNA 6
        MOVB    (R3)+,(R0)+     ;COL. 7-10
        MOVB    (R3)+,(R0)+
        MOVB    (R3)+,(R0)+
        MOVB    (R3),(R0)+
        MOVB    #BLANCO,(R0)+   ;COL. 11
        MOVB    (R4)+,(R0)+     ;COL. 12
        MOVB    (R4),(R0)+      ;COL. 13
        MOVB    #BLANCO,(R0)+   ;COL 14
        MOVB    (R5)+,(R0)+     ;COL. 15-19
        MOVB    (R5)+,(R0)+
        MOVB    (R5)+,(R0)+
        MOVB    (R5)+,(R0)+
        MOVB    #BLANCO,(R0)+   ;COL. 19
        MOV     (SP)+,R1        ;R1<=FIN DE LINEA
FOR2::  CLR     R4              ;R4<=CONTADOR DE CARACTERES
        MOV     #NEWBUF,R2
        MOV     #6,R3
        CMPB    #BLANCO,(R2)    ;HAY ETIQUETA?
        BEQ     530$
        CMPB    #TAB,(R2)
        BEQ     530$
; PARA COLOCAR ETIQUETAS
510$:   INC     R4
        MOVB    (R2)+,(R0)+
        CMP     R4,#6
        BEQ     520$
        CMPB    #BLANCO,(R2)
        BEQ     530$
        CMPB    (R2),#TAB
        BEQ     530$
        BR      510$
; PARA CUANDO TCAR>6 BUSCAMOS EL LIMITADOR
520$:   CMPB    (R2),#BLANCO
        BEQ     530$
        CMPB    (R2),#TAB
        BEQ     530$
        INC     R2              ;LEEMOS CARACTER
        BR      520$
; PONEMOS EL NO. DE BLANCOS DEPENDIENDO DE CUANTOS CARACTERES CONSTA
; LA ETIQUETA. SINO HAY ETIQUETA 7 BLANCOS
530$:   SUB     R4,R3           ;R3<=6-R4
535$:   MOVB    #BLANCO,(R0)+   ;COL. 20-26
```

```
        TST     R3
        BEQ     540$
        DEC     R3
        BR      535$
; MOVEMOS LAS INSTRUCCIONES
540$:   INC     R2
        CMPB    (R2),#BLANCO
        BEQ     540$
        CMPB    (R2),#TAB
        BEQ     540$
; ESCRIBE EL CAMPO DE OPERACION
        MOVB    (R2)+,(R0)+     ;PRIMEROS 3 CARACTERES DE LA INSTRUCCION
        MOVB    (R2)+,(R0)+     ;COL 27-29
        MOVB    (R2)+,(R0)+
        MOVB    #BLANCO,(R0)+
; LA INSTRUCCION ES DE 3 O 4 CARACTERES?
        CMP     R2,R1
        BGT     550$
        CMPB    (R2),#BLANCO
        BEQ     550$
        CMPB    (R2),#TAB
        BEQ     550$
        MOVB    (R2)+,(R0)+     ;COL. 31
        BR      552$
550$:   MOVB    #BLANCO,(R0)+   ;COL. 31
552$:   MOVB    #BLANCO,(R0)+   ;COL. 32
        MOVB    #BLANCO,(R0)+   ;COL. 33
        RTS     PC
        .END
>
```

```
        .TITLE  CALPLC
;+
; SUBRUTINA PARA PRIMER PASO DEL ENSAMBLADOR
; CALCULA EN CUANTO DEBE INCREMENTARSE EL PLC. IDENTIFICA EL TIPO DE
; INSTRUCCION Y LA ACCION A EJECUTAR
; R4<=CONTADOR DE CARACTERES
;-
BANSW:: .WORD   0               ;BANDERA PARA INDICAR SI ESTAMOS EN MODO
                                ;DIRECTO O EXTENDIDO
FLAG::  .WORD                   ;MANDAMOS LA CLAVE DEL TOKEN ENCONTRADO
NBYTES::.WORD                   ;LONGITUD DE LA INSTRUCCION.
                                ;INDICA CUANTOS BYTES OCUPA
NEXTKN::.BLKB   80.             ;VECTOR DONDE GUARDAMOS OPERANDO
        .ENABL  GBL
;
;
CALPLC::
        CLR     NBYTES
        CLR     BANSW
        MOV     X(R0),R3        ;MOVEMOS CLAVE DEL TIPO DE INSTRUCCION
        BIC     #177760,R3      ;TRABAJAMOS CON LOS BITS 0-3
        MOV     #207.,FLAGER    ;FLAGER<=COD. DE OPERACION INDEFINIDO
        DEC     R3              ;DE ACUERDO AL
        ADD     R3,R3           ; TIPO DE INSTRUCCION
        JMP     @TABLE(R3)      ;  VAMOS A SU PROCEDIMIENTO
TABLE:: .WORD   FIN,MACNOM,INST
        .WORD   OP1,CLEAR,MACRO
        .WORD   MEND,PSEUDO,CALLER
INST:   MOV     #209.,FLAGER    ;FLAGER<=NO SE PUEDE IDENTIFICAR EL MODO
INST1:  CMP     R2,R1           ;SI NO HAY MAS CARACTERES SERA ERROR
        BGE     CALLER          ;SE GARANTIZA QUE HAY MAS CARACTERES
        ADD     #1,R2           ;LEEMOS CARACTER
        CMPB    (R2),#BLANCO    ;ES BLANCO?
        BEQ     INST1           ;SI
        CMPB    (R2),#TAB       ;ES TAB?
        BEQ     INST1           ;SI
        CMPB    (R2),#43        ;CARACTER=#?
        BNE     NEXT
; MODO INMEDIATO
INME:
        BIT     #100000,X(R0)   ;BIT 15 ENCENDIDO?
        BEQ     SALT            ;NO,NBYTES<=2
        INC     NBYTES          ;NBYTES<=3
SALT:   BIT     #400,X(R0)      ;EL OPERADOR ADMITE ESTE MODO?
        BEQ     CALLER          ;NO, ERROR
        JMP     CLEAR           ; VE A ACTULIZAR PLC
; LIMPIA NEXTKN
NEXT:   MOV     #NEXTKN,R4
        MOV     #40.,R3
2$:     CLR     (R4)+
        SOB     R3,2$
        MOV     #NEXTKN,R3      ;R3<=#NEXTKN
        CLR     R4              ;R4<=CUENTA CARACTERES
3$:     CMP     R2,R1           ;FIN DE LINEA?
        BGT     ASKINX          ;PREGUNTA SI ES MODO INDEXADO
        CMPB    (R2),#BLANCO    ;BUSCAMOS SEPARADORES
        BEQ     ASKINX          ;PREGUNTA SI ES MODO INDEXADO
        CMPB    (R2),#TAB       ;
        BEQ     ASKINX
        CMPB    (R2),#',
        BEQ     ASKDIR          ;PREGUNTA SI ES MODO DIRECTO
                                ; O EXTENDIDO
```

```
        MOVB    (R2)+,(R3)+     ;NEXTKN<=BUFF
        INC     R4              ;TCAR<=TCAR+1
        BR      3$
; IDENTIFICA SI ES MODO INDEXADO
ASKINX: CMP     R4,#1           ;TCAR=1?
        BNE     PREDOE
        CMPB    NEXTKN,#'X      ;TENEMOS X?
        BNE     PREDOE
; MODO INDEXADO
INDEX:  BIT     #2000,X(R0)     ;BIT 10 ENCENDIDO?
        BEQ     CALLER          ;NO ADMITIO EL MODO
        JMP     CLEAR           ;ACTUALIZA PLC
; LIMPIA NEXTKN
ASKDIR: MOV     #NEXTKN,R3
        MOV     #40.,R4
4$:     CLR     (R3)+
        SOB     R4,4$
        CLR     R4              ;R4<=0 ,CUENTA CARACTERES
        MOV     #NEXTKN,R3      ;R3<=#NEXTKN
5$:     CMP     R2,R1           ;FIN DE LINEA?
        BGE     6$              ;SI
        ADD     #1,R2           ;LEE CARACTER
        CMPB    (R2),#BLANCO    ;ES BLANCO?
        BEQ     6$              ;SI
        CMPB    (R2),#TAB       ;ES TAB?
        BEQ     6$              ;SI
        INC     R4              ;R4<=R4+1
        MOVB    (R2),(R3)+      ;COPIA CARACTER A NEXTKN
        BR      5$
6$:     CMP     R4,#1           ;R4=1?
        BNE     CALLER
        CMPB    NEXTKN,#'X      ;MODO INDEXADO
        BNE     CALLER          ;NO,ERROR
        BR      INDEX           ;TENEMOS MODO INDEXADO
; DETECCION DE ERRORES
CALLER: JSR     PC,VIOLA
        RTS     PC
PREDOE: INC     R2              ;LEE CARACTER
        CMP     R2,R1           ;FIN DE LINEA?
        BGT     DIROEX          ;SI,TENEMOS MODO DIRECTO O EXTENDIDO
        CMPB    (R2),#BLANCO    ;BLANCO?
        BEQ     PREDOE
        CMPB    (R2),#TAB       ;TAB?
        BEQ     PREDOE
        CMPB    (R2),#',        ;COMA?
        BEQ     ASKDIR
; IDENTIFICAMOS SI ES MODO DIRECTO O EXTENDIDO
DIROEX: MOV     R0,-(SP)
        INC     BANSW           ;SWICH PARA SUB. CREA
        MOV     #NEXTKN,R3      ;R3<=#NEXTKN
        JSR     PC,CREA         ;CONSTRUYE VECTOR CON OPERANDOS EN FORMA
                                ;BINARIA
        JSR     PC,REVISA       ;REVISA LA SINTAXIS DE UNA EXPRESION
        JSR     PC,EVALUA       ;EVALUA UNA EXPRESION
        MOV     (SP)+,R0
        MOV     #210.,FLAGER    ;FLAGER<=BYTE OVERFLOW
        TST     R1              ;NUMERO>65535
        BNE     CALLER
        MOV     #209.,FLAGER    ;FLAGER<=MODO DE DIRECCIONAMIENTO ILEGAL
        BIT     #177400,R2      ;R2>255
        BEQ     DIRECT          ;NO , ES MODO DIRECTO
; MODO EXTENDIDO
EXPAN:
        MOV     #210.,FLAGER    ;FLAGER<=BYTE OVERFLOW
```

```
        BIT     #4000,X(R0)     ;BIT ONCE ENCENDIDO?
        BEQ     CALLER          ;NO ADMITE ESE MODO
EXPA2:: MOV     #3,NBYTES       ;NBYTES<=3                          234
        BR      SIGUE
; MODO DIRECTO
DIRECT:
        BIT     #1000,X(R0)
        BEQ     EXPAN
        MOV     #2,NBYTES
        BR      SIGUE
; MODO INHERENTE Y ACUMULADOR
; COLOCA ETIQUETAS Y ACTUALIZA PLC
CLEAR:  INC     NBYTES          ;NBYTES<=NBYTES+1
OP1:    INC     NBYTES          ;NBYTES<=NBYTES+1
SIGUE:  MOV     PLC,TEMPLC
        MOV     #TOKENS+6,R0
        TST     (R0)            ;HAY ETIQUETA?
        BEQ     NOLABEL         ;NO VE A INCREMENTAR PLC
        MOV     #206.,FLAGER    ;FLAGER<=SIMBOLO REDEFINIDO
        MOV     #9.,FLAG        ;FLAG<=CLAVE DE ETIQUETA
        JSR     PC,HASETI       ;COLOCA ETIQUETA EN TABLA
; PARA CUANDO NO HAY ETIQUETA
NOLABEL:MOV     #218.,FLAGER    ;FLAGER<=WORD OVERFLOW
        ADD     NBYTES,TEMPLC   ;ACTUALIZA PLC
        BCS     CALAUX          ;SI PLC>65535 ERROR
        MOV     TEMPLC,PLC
        CLR     R5
        RTS     PC              ;TERMINAMOS DE ACTUALIZAR PLC
CALAUX: JMP     CALLER
FIN:    JMP     CIERRA
PSEUDO: JSR     PC,SBPSEU       ;VE A SUB. DE PSEUDO-INSTRUCCIONES
        RTS     PC
MACNOM: JSR     PC,SBMNAM       ;PROCESAR MACROLLAMADA
        RTS     PC
MACRO:  JSR     PC,SBMAC        ;PROCESA MACRODEFINICION
        RTS     PC
; SI ENCONTRO MEND, SERA ERROR
MEND:   MOV     #217.,FLAGER    ;FLAGER<=MACRO ANIDADO?
        JSR     PC,VIOLA        ;SI,ERROR
        JMP     CASFIN          ;NO, TERMINAMOS
;
;
        .PAGE
        .SBTTL  SBPSEU
;+
; PRIMER PASO.
; PROCEDIMIENTO PARA LAS PSEUDO-INSTRUCCIONES.
; SE INCREMENTA EL PLC
;-
SBPSEU::
        MOV     #212.,FLAGER    ;FLAGER<=ERROR EN EL OPERANDO DE UNA
                                ;DIRECTIVA
21$:    CMP     R2,R1           ;FIN DE LINEA?
        BGE     HELCHE
        ADD     #1,R2           ;LEEMOS CARACTER
        CMPB    (R2),#BLANCO    ;ES BLANCO?
        BEQ     21$
        CMPB    (R2),#TAB       ;ES TAB?
        BEQ     21$
; YA TENEMOS UN CARACTER?
        MOV     #NEXTKN,R5
        MOV     R5,R3
; LIMPIAMOS NEXTKN
22$:    CLR     (R5)+
        TST     (R5)
        BNE     22$
```

```
        CLR     R4              ;TCAR<=0
        ADD     #2,R0
        CMP     X(R0),#6        ;CLAVE DE FCC?
        BGE     $24             ;SI VE A 24
23$:    CMP     R2,R1           ;FIN DE LINEA?
        BGT     $24             ;SI VE A $24
        CMPB    (R2),#BLANCO    ;ES BLANCO
        BEQ     $24
        CMPB    (R2),#TAB       ;ES TAB?
        BEQ     $24
        MOVB    (R2)+,(R3)+     ;COLOCA EN NEXTKN OPERANDO
        INC     R4              ;TCAR<=TCAR+1
        BR      23$             ;SIGUE CREANDO NEXTKN
HELCHE: JMP     CHERER
$24:    MOV     X(R0),R3        ;R3<=CLAVE DE LA INSTRUCCION
        ASL     R3              ;DE ACUERDO
        MOV     #TOKENS+6,R0    ; AL TIPO DE DIRECTIVA
        JMP     @TABPSE(R3)     ;  REALIZA PROCEDIMIENTO
TABPSE: .WORD   ORG,EQU,RMB,FCB,FDB,NAM,FCC
; DIRECTIVA ORG
ORG:
        MOV     #223.,FLAGER    ;FLAGER<=LA DIRECTIVA NO ADMITE ETIQUETA
        TST     (R0)            ;HAY ETIQUETA?
        BNE     CHERER          ;SI,ERROR
        MOV     #NEXTKN,R3
        JSR     PC,CREA         ;EVALUA
        JSR     PC,REVISA       ; OPERANDO
        JSR     PC,EVALUA
        MOV     #218.,FLAGER    ;FLAGER<=WORD OVERFLOW
        TST     R1              ;PLC CORRECTO?
        BNE     CHERER          ;NO
        MOV     R2,PLC          ;MUEVE VALOR DE LA EXPRESION
        JMP     $50
; DIRECTIVA EQU
EQU:
        MOV     #213.,FLAGER    ;FLAGER<=ERROR EN DIRECTIVA EQU
        TST     (R0)            ;HAY ETIQUETA?
        BEQ     CHERER          ;NO,ERROR
        MOV     #NEXTKN,R3
        JSR     PC,CREA         ;EVALUA
        JSR     PC,REVISA       ; OPERANDO
        JSR     PC,EVALUA       ;
        MOV     #218.,FLAGER    ;FLAGER<=WORD OVERFLOW?
        TST     R1              ;EXPRESION<65535
        BNE     CHERER          ;NO ERROR
        MOV     R2,TEMPLC
        MOV     #TOKENS+6,R0    ;COLOCA
        MOV     #9.,FLAG        ; ETIQUETA

        JSR     PC,HASETI       ;  EN TABLA
        JMP     $50
; DIRECTIVA RMB
RMB:
        MOV     PLC,TEMPLC
        TST     (R0)            ;HAY ETIQUETA?
        BEQ     $26             ;NO
        MOV     R4,-(SP)        ;SALVA TCAR
        MOV     #206.,FLAGER    ;FLAGER<=SIMBOLO REDEFINIDO
        MOV     #9.,FLAG        ;FLAG<=CLAVE DE ETIQUETA
        JSR     PC,HASETI       ;COLOCA ETIQUETA EN TABLA JUNTO CON EL
                                ;VALOR DEL OPERANDO
        MOV     (SP)+,R4
$26:    MOV     #NEXTKN,R3
        JSR     PC,CREA         ;EVALUA
        JSR     PC,REVISA       ; OPERANDO
        JSR     PC,EVALUA       ;
```

```
        MOV     #210.,FLAGER    ;FLAGER<=BYTE OVERFLOW
        TST     R1              ;EXPRESION>255?
        BNE     CHERER          ;SI,ERROR
        ADD     R2,TEMPLC       ;ACTUALIZA
        BCS     CHERER          ; PL
        MOV     TEMPLC,PLC      ;PLC<=PLC+R2
        JMP     $50
; DETECTO ERROR
CHERER: JSR     PC,VIOLA
        JMP     CASFIN
; DIRECTIVA FDB
FDB:    MOV     #1,NBYTES
; DIRECTIVA FCB
FCB:    INC     NBYTES
        MOV     PLC,TEMPLC
        TST     (R0)            ;HAY ETIQUETA?
        BEQ     $30             ;NO
        MOV     R4,-(SP)        ;SALVA TCAR
        MOV     #206.,FLAGER    ;FLAGER<=SIMBOLO REDEFINIDO
        MOV     #9.,FLAG        ;COLOCA ETIQUETA EN TABLA
        JSR     PC,HASETI       ;JUNTO CON EL VALOR DEL PLC
        MOV     (SP)+,R4        ;RESTAURA TCAR
$30:    MOV     #218.,FLAGER    ;FLAGER<=WORD OVERFLOW
        MOV     #NEXTKN,R2
        ADD     R2,R4
        DEC     R4
$31:    CMP     R2,R4           ;FIN DEL VECTOR NEXTKN?
        BGT     $32             ;SI
        CMPB    (R2)+,#',       ;TENEMOS UN OPERANDO MAS?
        BNE     $31             ;NO
        ADD     NBYTES,TEMPLC   ;PLC<=PLC+NBYTES
        BCS     CHERER
        BR      $31
$32:    MOV     #218.,FLAGER    ;FLAGER<=WORD OVERFLOW
        ADD     NBYTES,TEMPLC   ;PLC>65535?
        BCS     CHERER          ;SI ERROR
        MOV     TEMPLC,PLC      ;PLC<=PLC+NBYTES
        JMP     $50
; DIRECTIVA NAM
NAM:
        MOV     #TOKENS+6,R0
        MOV     #223.,FLAGER    ;FLAGER<=LA DIRECTIVA NO ADMITE ETIQUETA
        TST     (R0)            ;TENEMOS ETIQUETA
        BNE     CHERER          ;SI,ERROR
        MOV     #201.,FLAGER    ;FLAGER<=ERROR EN LA DIRECTIVA NAM
        CMP     LINEA,#1        ;SI NO ES PRIMERA LINEA ERROR
        BNE     CHERER
        JMP     $50
; DIRECTIVA FCC
; R2<=APUNTADOR DEL BUFFER
; R1<=FIN DEL BUFFER
; R3<=#NEXTKN
FCC:
$40:    MOV     PLC,TEMPLC
        MOV     #TOKENS+6,R0
        TST     (R0)            ;HAY ETIQUETA?
        BEQ     $41             ;NO
        MOV     #9.,FLAG        ;FLAG<=CLAVE DE ETIQUETA
        MOV     #206.,FLAGER    ;FLAGER<=SIMBOLO REDEFINIDO
        MOV     R2,-(SP)
        MOV     R1,-(SP)
        JSR     PC,HASETI       ;COLOCA ETIQUETA EN TABLA JUNTO CON PLC
        MOV     (SP)+,R1
        MOV     (SP)+,R2
$41:    CMPB    (R2),#'9        ;DIRECTIVA EMPIEZA CON NUMERO
        BGT     $45             ;NO
```

```
        CMPB    (R2),#'0
        BLT     $45
; PARA CUANDO EMPIEZA CON NUMERO
        MOV     #9.,R3          ;LIMITE DE BOXTKN
        MOV     #BOXTKN,R5
        MOV     R5,R0           ;R0<=#BOXTKN
$42:    TST     (R5)            ;LIMPIAMOS BOXTKN
        BEQ     $43
        CLR     (R5)+
        SOB     R3,$42
$43:    MOV     R2,R5           ;R5<=APUNTADOR DEL BUFFER
        MOV     R1,R4           ;R4<=FIN DE LINEA
        CLR     R2              ;R2<=TCAR
        MOV     #212.,FLAGER    ;FLAGER<=ERROR EN EL OPERANDO DE UNA
                                ;DIRECTIVA
$44:    CMP     R5,R4           ;FIN DE LINEA?
        BGE     CHEKER
        MOVB    (R5)+,(R0)+     ;COPIA DE NEXTKN A BOXTKN
        INC     R2
        CMP     #18.,R2         ;MAS DE 18 CARACTERES?
        BEQ     CHEKER          ;SI ERROR
        CMPB    (R5),#',        ;DELIMITADOR DE NUMERO?
        BNE     $44             ;BUSCALO
        MOVB    #BLANCO,(R0)    ;INSERTAMOS BLANCO AL FINAL DE BOXTKN
        MOV     #BOXTKN,R0
        JSR     PC,STK
        MOV     #212.,FLAGER    ;FLAGER<=ERROR EN EL OPERANDO DE UNA
                                ;DIRECTIVA
        BIT     #177400,R1      ;R1>255
        BNE     CHEKER          ;SI ,ERROR
        MOV     #218.,FLAGER    ;FLAGER<=WORD OVERFLOW
        ADD     R1,TEMPLC
        BCS     CHEKER
        MOV     TEMPLC,PLC      ;ACTUALIZA PLC
        JMP     $50
; CUANDO EL USUARIO DA EL DELIMITADOR
$45:    CLR     R4              ;TCAR<=0
        MOV     #212.,FLAGER    ;ERROR EN EL OPERANDO DE UNA DIRECTIVA
        MOVB    (R2)+,R0        ;R0<=DELIMITADOR
$46:    CMP     R2,R1           ;FIN DE LINEA?
        BGT     CHEKER
        CMPB    (R2)+,R0        ;ENCONTRAMOS DELIMITADOR?
        BEQ     $47
        INC     R4              ;R4<=TCAR+1
        BR      $46
$47:    MOV     #218.,FLAGER    ;FLAGER<=WORD OVERFLOW
        ADD     R4,TEMPLC
        BCS     CHEKER
        MOV     TEMPLC,PLC      ;PLC<=PLC+R4
$50:    RTS     PC
; DETECTO ERROR
CHEKER: JSR     PC,VIOLA
        JMP     BEGIN
        .END
PIP>TI:=WRIT7.MAC
```

```
        .TITLE PSEIN2.MAC
;+
; RUTINA DEL SEGUNDO PASO DEL ENSAMBLADOR QUE SE USA PARA PROCESAR LAS
; PSEUDO-INSTRUCCIONES Y ESCRIBIR LAS LINEAS ENSAMBLADAS EN EL ARCHIVO
; MOT80.TMP.
;-
        .MCALL  OPEN$A,PUT$S,CLOSE$
WAEQU:: .BLKW   2
AUXBOX: .BLKB   80.
SWFDB:  .BLKW   1
LETNAM: .ASCIZ/%20S.TITULO %8A%44S/
        .EVEN
TMPFDB: .BLKW   1
SWFCC:  .BLKW   1
        .ENABL GBL
;
;
PSEIN2::
        INC     R2              ;LEE CARACTER
        CMPB    (R2),#BLANCO    ;QUITA BLANCOS
        BEQ     PSEIN2
        CMPB    (R2),#TAB       ;QUITA TAB
        BEQ     PSEIN2
        CLR     R4              ;R4<=TCAR
        ADD     #2,R0           ;MA DA
        CMP     X(R0),#6        ; ES DIRECTIVA FCC?
        BGE     $1004           ;SI, VE A $1004
; COLOCA OPERANDO EN NEXTKN
        MOV     #NEXTKN,R3
$1002:  CMP     R2,R1
        BGT     $1004
        CMPB    (R2),#BLANCO    ;ES BLANCO?
        BEQ     $1004
        CMPB    (R2),#TAB       ;ES TAB?
        BEQ     $1004
        MOVB    (R2)+,(R3)+     ;NEXTKN<=NEWBUF
        INC     R4              ;TCAR<=TCAR+1
        BR      $1002
$1004:  MOV     X(R0),R3        ;DE ACUERDO A LA CLAVE
        ASL     R3              ; DE LA PSEUDO-INSTRUCCION
        JMP     @ARRPSI(R3)     ;  REALIZAMOS SU PROCEDIMIENTO
ARRPSI: .WORD   ORGPS2,EQUPS2,RMBPS2,FCBPS2,FDBPS2,NAMPS2
        .WORD   FCCPS2
; DIRECTIVA ORG
ORGPS2: MOV     R1,-(SP)
        MOV     #NEXTKN,R3
        JSR     PC,CREA         ;EVALUA OPERANDO
        JSR     PC,EVALUA
        MOV     R2,PLC          ;PLC<=VALOR DEL OPERANDO
        MOV     R2,R4
; ESCRIBE LINEA ENSAMBLADA EN ARCHIVO MOT80.TMP
        MOV     #SALPLC,R3
        JSR     PC,BINHEX       ;CONVIERTE NUMERO DE BINARIO A HEXDECIM.
        MOV     #20040,CODIGO
        MOV     #20040,FIASEC   ;COLOCA BLANCOS
        MOV     #20040,FIASEC+2
        MOV     (SP)+,R1
        JSR     PC,FORMAT
        JSR     PC,LASFRM
        JMP     $1252
; DIRECTIVA EQU
```

```
EQUPS2: MOV     R1,-(SP)
        MOV     #NEXTKN,R3
        JSR     PC,CREA         ;EVALUA
        JSR     PC,EVALUA       ; OPERANDO
        MOV     (SP)+,R1
        MOV     R2,R4
; ESCRIBE LINEA ENSAMBLADA EN ARCHIVO MOT80.TMP
        MOV     #WAEQU,R3
        JSR     PC,BINHEX       ;CONVIERTE NUMERO DE BINARIO A HEXADECI.
        MOV     #20040,SALPLC   ;SALPLC<=BLANCOS
        MOV     #20040,SALPLC+2
        JSR     PC,UTILFO
        JMP     $1252
; DIRECTIVA RMB
RMBPS2: MOV     R1,-(SP)
        MOV     #NEXTKN,R3
        JSR     PC,CREA         ;EVALUA OPERANDO
        JSR     PC,EVALUA
        MOV     R2,NBYTES       ;NBYTES<=VALOR DEL OPERANDO
        MOV     R2,R4
; ESCRIBE LINEA ENSAMBLADA EN MOT80.TMP
        MOV     #WAEQU,R3
        JSR     PC,BINHEX       ;CONVIERTE NUMERO DE BINARIO A DECIMAL
        MOV     PLC,R4
        MOV     #SALPLC,R3
        JSR     PC,BINHEX
        MOV     (SP)+,R1
        JSR     PC,UTILFO       ;PLC<=PLC+NBYTES
        JMP     $1252
; DIRECTIVA FDB
FDBPS2: CLR     SWFDB
        MOV     #2,NBYTES       ;NBYTES<=2
        BR      $1010
; DIRECTIVA FCB
FCBPS2: CLR     SWFDB
        INC     SWFDB           ;ENCIENDE SWICH DE FCB
        INC     NBYTES          ;NBYTES<=1
$1010:  MOV     R1,TMPFDB
        MOV     #NEXTKN,R0
        MOV     R4,R5           ;R5<=TCAR EN NEXTKN
        CLR     R2              ;R2<=TCAR LEIDOS
        CLR     FLAG            ;FLAG<=INDICA EL FORMATO DE SALIDA 1 0 2
$1015:  CLR     R4              ;R4<=TCAR DE CADA TOKEN
; LIMPIA AUXBOX
        MOV     #AUXBOX,R3
$1020:  CLR     (R3)+
        TST     (R3)
        BNE     $1020
        MOV     #AUXBOX,R3      ;R3<=AUXBOX
; HEMOS LEIDO TODOS LOS CARACTERES EN NEXTKN?
$1025:  CMP     R2,R5
        BGE     $1030
        INC     R2              ;R2<=TCAR+1
        CMPB    (R0),#',        ;TENEMOS COMA?
        BEQ     $1030           ;SI
        MOVB    (R0)+,(R3)+     ;AUXBOX<=NEXTKN
        INC     R4
        BR      $1025
$1030:  MOV     R0,-(SP)
        MOV     R2,-(SP)        ;SALVA TCAR LEIDOS
        MOV     R5,-(SP)        ;SALVA TCAR DE NEXTKN
        TST     R4              ;R4=0?
        BEQ     $1035
        MOV     #AUXBOX,R3
        JSR     PC,CREA         ;EVALUA
        JSR     PC,REVISA       ;OPERANDO
```

```
        JSR     PC,EVALUA
        TST     R1              ;ERROR AL EVALUAR
        BNE     $1047           ;SI
        TST     SWFDB           ;VE A FORMATO FDB?
        BEQ     $1034           ;SI
; PARA DIRECTIVA FCD
        BIT     #177400,R2      ;R2<255?
        BNE     $1047           ;NO ERROR
$1034:  MOV     R2,R4
$1035:  MOV     #WAEQU,R3
        JSR     PC,BINHEX       ;CONVIERTE NO. DE BINARIO A HEXADECIMAL
        TST     SWFDB           ;ESTAMOS EN FDB?
        BEQ     $1036           ;SI
        MOV     #WAEQU,R3       ;RECORREMOS EL NUMERO
        MOV     2(R3),(R3)      ;E INSERTAMOS
        MOV     #20040,2(R3)    ;BLANCOS
$1036:  MOV     PLC,R4          ;R4<=PLC
        MOV     #SALPLC,R3
        JSR     PC,BINHEX       ;CONVIERTE NO. DE BINARIO A HEXADECIMAL
        TST     FLAG            ;FORMATO 1?
        BNE     $1040           ;NO
        INC     FLAG            ;FLAG<=1
        MOV     TMPFDB,R1
        JSR     PC,UTILF0       ;MANDA A ESCRIBIR LINEA
        BR      $1045
$1040:  JSR     PC,UTILF2       ;MANDA ESCRIBIR LINEA CON FORMATO 2
$1045:  MOV     (SP)+,R5
        MOV     (SP)+,R2
        MOV     (SP)+,R0
        INC     R0
        CMP     R2,R5           ;FIN DE LINEA?
        BLT     $1015           ;NO
        CMPB    -(R0),#',       ;TERMINA CON ,?
        BNE     $1043           ;NO
        CLR     R4              ;R4<=0
        MOV     #WAEQU,R3
        JSR     PC,BINHEX       ;CONVIERTE NO. DE BINARIO A HEXADECIMAL
        TST     SWFDB           ;FDB?
        BEQ     $1046           ;SI
        MOV     #WAEQU,R3       ;RECORRE NUMERO E
        MOV     2(R3),(R3)      ;  INSERTA
        MOV     #20040,2(R3)    ;    BLANCOS
$1046:  MOV     PLC,R4
        MOV     #SALPLC,R3
        JSR     PC,BINHEX       ;CONVIERTE NO. A HEXADECIMAL
        JSR     PC,UTILF2       ;ESCRIBE        LINEA ENSAMBLADA
$1043:  JMP     $1252
$1047:  MOV     #210.,FLAGER    ;FLAGER<=BYTE OVERFOLOW
        JSR     PC,VIOLA
        ADD     NBYTES,PLC
        JMP     $1252
; DIRECTIVA NAM
NAMPS2: MOV     R1,-(SP)
        MOV     #8.,R5          ;SEPARA EL NOMBRE
        SUB     R4,R5
        BLE     $1120
$1110:  MOVB    #BLANCO,(R3)+   ;SEPARA 8 LETRAS Y EL RESTO BLANCOS
        SOB     R5,$1110
$1120:  MOV     #NEXTKN,ARGBLK
        MOV     #PON2,R0
        MOV     #LETNAM,R1
        MOV     #ARGBLK,R2
        CALL    $EDMSG          ;ESCRIBE EL NOMBRE DEL PROGRAMA
        OPEN$A  #FDBENS         ;EN EL ARCHIVO
        PUT$S   #FDBENS
        CLOSE$  #FDBENS         ;MOTBO.TMP
```

```
; INSERTAMOS BLANCOS
        MOV     #20040,CODIGO
        MOV     #20040,SALFLC
        MOV     #20040,SALFLC+2
        MOV     #20040,FIASEC
        MOV     #20040;FIASEC+2
        MOV     (SP)+,R1
        JSR     PC,FORMAT       ;ESCRIBE LINEA ENSAMBLADA
        JSR     PC,LASFRM       ;EN ARCHIVO MOT80.TMP
        JMP     $1252
; DIRECTIVA FCC
FCCPS2: CLR     SWFCC           ;SWFCC<=BANDERA DE NUMERO
        CLR     FLAG            ;FLAG<=BANDERA DE DELIMITADOR
        INC     NBYTES          ;NBYTES<+NBYTES+1
        CMPB    (R2),#'9
        BGT     $1245
        CMPB    (R2),#'0
        BLT     $1245
; PARA CUANDO EMPIEZA CON NUMERO
        INC     SWFCC
        MOV     #9.,R3
        MOV     #BOXTKN,R5
        MOV     R5,R0           ;R0<=#BOXTKN
$1210:  TST     (R5)
        BEQ     $1215
        CLR     (R5)+
        SOB     R3,$1210
$1215:  MOV     R2,R5           ;R5<=POINTER DEL NEWBUF
        MOV     R1,R4           ;R4<=FIN DE LINEA
$1220:  MOVB    (R5)+,(R0)+     ;BOXTKN<=NEWBUF
        CMPB    (R5),#',        ;ENCONTRAMOS COMA?
        BNE     $1220
        MOVB    #BLANCO,(R0)    ;MANDA BLANCO AL FINAL DE BOXTKN
        CLR     R2              ;R2<=CONTADOR DE CARACTERES
        MOV     #BOXTKN,R0
$1225:  CMPB    (R0),#BLANCO
        BEQ     $1235
        CMPB    (R0),#TAB
        BEQ     $1230
        INC     R2
        INC     R0
        BR      $1225
$1230:  MOVB    #BLANCO,(R0)
$1235:  MOV     #BOXTKN,R0
        JSR     PC,STK          ;CONVIERTE EL NUMERO A BINARIO
        MOV     R1,R3           ;R3<=TCAR EN ASCII
        MOV     R4,R1           ;R1<=RESTAURAMOS FIN DE LINEA
        MOV     R5,R2           ;R2<=APUNTADOR DE FIN DE LINEA
        MOV     R1,R5           ;SALVAMOS FIN DE LINEA
        CLR     R0              ;R0<=TOTAL DE CARACTERES FORMATEADOS EN
                                ;ASCII
$1240:  INC     R2
        CMP     R2,R5           ;FIN DE LINEA?
        BGT     $1241
        CLR     R4
        MOVB    (R2),R4
        BR      $1242
$1241:  MOV     #BLANCO,R4
$1242:  CMP     R3,R0
        BEQ     $1252           ;HEMOS MANDADO TODOS LOS CARACTERES ASCI
        INC     R0
        MOV     R0,-(SP)
        MOV     R2,-(SP)
        MOV     R3,-(SP)
        MOV     R5,-(SP)
        BR      $1247
```

```
; RESTAURA APUNTADORES
$1243:  MOV     (SP)+,R5
        MOV     (SP)+,R3
        MOV     (SP)+,R2
        MOV     (SP)+,R0
        BR      $1240
; CUANDO EMPIEZA CON DELIMITADOR
$1245:  MOVB    (R2)+,R5        ;MUEVE DELIMITADOR DE LA CADENA
$1246:  CMPB    (R2),R5         ;ENCONTRASTE EL DELIMITADOR?
        BEQ     $1252           ;SI
        MOV     R2,-(SP)        ;PROTEGE APUNTADOR DE NEWBUF
        MOV     R5,-(SP)        ;PROTEGE SEPARADOR
        CLR     R4
        MOVB    (R2),R4         ;R4<=NUMERO A CONVERTIR A HEX.
$1247:  MOV     #WAEQU,R3       ;R3<=COLOCALO EN WAEQU
        JSR     PC,BINHEX       ;CONVIERTE DE BINARIO A HEX.
        MOV     PLC,R4          ;CONVIERTE PLC A HEX
        MOV     #SALPLC,R3
        JSR     PC,BINHEX
        TST     FLAG            ;MANDA FORMATO 1
        BNE     $1250
        INC     FLAG
        MOV     #20040,FIASEC
        MOV     #20040,FIASEC+2 ;MANDA BLANCOS
        MOV     WAEQU+2,CODIGO
        JSR     PC,FORMAT       ;ESCRIBE LINEA ENSAMBLADA
        JSR     PC,LASFRM       ;EN ARCHIVO MOT80.TMP
        TST     SWFCC           ;FCC EMPIEZA CON NUMERO?
        BNE     $1243
        MOV     (SP)+,R5
        MOV     (SP)+,R2
        INC     R2
        BR      $1246
; PARA FORMATO 2
$1250:  MOV     #WAEQU,R3
        MOV     2(R3),(R3)
        MOV     #20040,2(R3)
        JSR     PC,UTILF2
        TST     SWFCC
        BNE     $1243
        MOV     (SP)+,R5
        MOV     (SP)+,R2
        INC     R2
        BR      $1246
$1252:  RTS     PC
        .END
PIP>TI:=WRIT8.MAC
```

```
        .TITLE SEGUNDOP
;+
; PROCEDIMIENTO QUE INDICA EL TIPO DE SENTENCIA QUE SE TIENE,
; INCREMENTA EL PLC Y PRODUCE EL LENGUAJE MAQUINA
; -
        .MCALL  OPEN$A,PUT$S,CLOSE$
IHAVE:: .WORD   0
ISTROP1::.ASCIZ/%33A%VS%38A/
        .EVEN
ISTEND: .ASCIZ/%33A%47S/
        .EVEN
CODIGO::.BLKW   1
ARGBLK::.BLKB   84.
FIASEC::.BLKW   2.
        .ENABL  GBL
;
SEGCPC::
; LIMPIA NEXTKN
        MOV     #NEXTKN,R4
$10:    CLR     (R4)+
        TST     (R4)
        BNE     $10             ;LIMPIA EL BUFFER PON2
        MOV     #PON2,R4
        MOV     #40.,R5
$7:     MOV     #20040,(R4)+
        SOB     R5,$7
        MOV     #ARGBL1,R4      ;LIMPIA ARGBL1
        MOV     #40.,R5
$6:     CLR     (R4)+
        SOB     R5,$6
        CLR     NBYTES          ;NBYTES<=0
        MOV     X(R0),R3        ;TRAE CLAVE DE TABLA
        BIC     #177760,R3
        DEC     R3              ;DE ACUERDO
        ADD     R3,R3           ; A CLAVE IDENTIFICA EL TIPO
        JMP     @TABLP2(R3)     ;  DE INSTRUCCION Y VE A PROCESARLA
TABLP2: .WORD   FIN2
        .WORD   MNOM2
        .WORD   INSP2
        .WORD   OP1SEG
        .WORD   CLEAR2
        .WORD   MAC2
        .WORD   MEN2
        .WORD   PSEINS
; PARA DIRECTIVA END
FIN2:   INC     IHAVE           ;IHAVE<=INDICA QUE TERMINO EL PROGRAMA
; MANDA A ESCRIBIR LINEA ENSAMBLADA
        MOV     #20040,SALPLC
        MOV     #20040,SALPLC+2
        MOV     #20040,CODIGO
        MOV     #20040,FIASEC
        MOV     #20040,FIASEC+2
        JSR     PC,FORMAT
        MOV     #ARGBL1,ARGBLK
        MOV     #PON2,R0
        MOV     #ISTEND,R1
        MOV     #ARGBLK,R2
        CALL    $EDMSG
        OPEN$A  #FDBENS         ;ABRE EL ARCHIVO MOTBO.TMP
        PUT$S   #FDBENS
        CLOSE$  #FDBENS         ;CIERRA ARCHIVO MOTBO.TMP
```

```
        RTS PC
; PARA ESCRIBIR MACRODEFINICIONES
MAC2:   MOV     #TOKENS,R0      ;LIMPIAMOS TOKENS
        CLR     (R0)
        CLR     2(R0)
        CLR     4(R0)
        JSR     PC,INSTRU       ;DEJA EN TOKENS MACRONOMBRE
        MOV     R2,-(SP)
        MOV     R1,-(SP)
        MOV     #TOKENS,R0
        JSR     PC,HASAUX       ;APLICA HASH A MACRONOMBRE
; CALCULAMOS DE CUANTAS LINEAS CONSTA EL CUERPO DEL MACRO
        MOV     X(R0),LYNEX     ;LYNEX<=NUMERO DE LINEA DEL ARCHIVO
                                ;DONDE ALMACENO EL CUERPO DEL MACRO
        ASR     LYNEX
        ASR     LYNEX
        ASR     LYNEX
        ASR     LYNEX
        ADD     #2,R0
        MOV     X(R0),LYNLAS
        SUB     LYNEX,LYNLAS
        INC     LYNLAS
        MOV     (SP)+,R1
        MOV     (SP)+,R2
; MANDAMOS A ESCRIBIR LA LINEA
        JSR     PC,WRCOM
; MANDAMOS A ESCRIBIR EL CUERPO DEL MACRO
$200:   JSR     PC,TRAENB       ;TRAEMOS UNA LINEA MAS
        INC     LINEA           ;LINEA<=LINEA+1
        JSR     PC,WRCOM        ;ESCRIBR LINEA
        DEC     LYNLAS
        TST     LYNLAS
        BGT     $200
        RTS     PC
; DIRECTIVA MEND
MEN2:   JSR     PC,WRCOM        ;ESCRIBE LINEA ENSAMBLADA
        RTS     PC
; PARA ESCRIBIR MACROLLAMADA
MNOM2:  JSR     PC,WRCOM
        RTS     PC
; PROCESA PSEUDO-INSTRUCCIONES
PSEINS: JSR     PC,PSEIN2
        RTS     PC
; IDENTIFICA EL MODO
INSP2:  CLR     R4              ;R4<=0
        CLR     R5
        MOV     #NEXTKN,R3      ;R3<=#NEXTKN
$203:   INC     R2              ;LEE CARACTER
        CMPB    (R2),#BLANCO    ;ES BLANCO?
        BEQ     $203
        CMPB    (R2),#TAB       ;ES TAB?
        BEQ     $203
        CMPB    (R2),#43        ;CARACTER=#?
        BEQ     INMME           ;SI,TENEMOS MODO INMEDIATO
        JMP     NEXT2
; MODO INMEDIATO
INMME:: MOV     #204,,FLAGER    ;FLAGER<=ERROR DE SINTAXIS
        INC     R2              ;LEE CARACTER
        CMP     R2,R1           ;FIN DE LINEA
        BGT     $210            ;SI
        CMPB    (R2),#47        ;CARACTER=' ?
        BNE     $230            ;NO
        INC     R2              ;LEE CARACTER
        CMP     R2,R1           ;FIN DE LINEA?
        BGT     $210            ;SI
        MOVB    (R2)+,R5        ;R5<=VALOR DEL CARACTER EN ASCII
```

```
        CMP     R2,R1           ;FIN DE LINEA
        BGT     $220
        CMPB    (R2),#BLANCO    ;ES BLANCO?
        BEQ     $220
        CMPB    (R2),#TAB       ;ES TAB?
        BEQ     $220
; DETECTA ERROR
$210:   JSR     PC,VIOLA
        RTS     PC
;SALVA REGISTROS
$220:   MOV     R1,-(SP)
        MOV     R0,-(SP)
        MOV     R5,R2
        CLR     R1
        JMP     $240
; SEPARA EL OPERANDO
$230:   DEC     R2
$232:   CMP     R2,R1           ;FIN DE LINEA
        BGE     $236
        INC     R2              ;LEE CARACTER
        CMPB    (R2),#BLANCO    ;ES BLANCO?
        BEQ     $236
        CMPB    (R2),#TAB       ;ES TAB?
        BEQ     $236
        INC     R4              ;TCAR<=TCAR+1
        MOVB    (R2),(R3)+      ;NEXTKN<=NEWBUF
        BR      $232
$236:   TST     R4              ;TCAR=0?
        BEQ     $210
        MOV     R1,-(SP)        ;R1<=FIN DE LINEA
        MOV     R0,-(SP)        ;R0<=DIRECCION DE HASH
        MOV     #NEXTKN,R3
        JSR     PC,CREA
        JSR     PC,REVISA       ;EVALUA
        JSR     PC,EVALUA       ; OPERANDO
;R1<=HIGH R2<=LOW
$240:   MOV     (SP)+,R0
        MOV     #210.,FLAGER    ;FLAGER<=BYTE OVERFLOW
        BIT     #100000,X(R0)   ;BIT 15 ENCENDIDO?
        BEQ     $260            ;NO
; PARA CUANDO NBYTES=3
        MOV     #3,NBYTES
        ADD     #2,R0           ;MOVEMOS APUNTADOR A TABLA
        MOV     X(R0),CODIGO    ;TRAE CODIGO DE MAQUINA DE TABLA
        TST     R1              ;NO SOPORTA NUMEROS NEGATIVOS
        BNE     $210
        MOV     #FIASEC,R3
        MOV     R2,R4
        JSR     PC,BINHEX       ;CONVIERTE NUMERO DE BINARIO A HEX.
$245:   MOV     #SALPLC,R3
        MOV     PLC,R4
        JSR     PC,BINHEX       ;CONVIERTE PLC DE BINARIO A HEX.
        MOV     (SP)+,R1
        JSR     PC,FORMAT       ;ESCRIBE LINEA ENSAMBLADA
        JSR     PC,LASFRM       ;EN ARCHIVO MOT80.TMP
        RTS     PC
; PARA CUANDO NBYTES=2
$260:   MOV     #2,NBYTES
        ADD     #2,R0
        MOV     X(R0),CODIGO    ;TRAE EL CODIGO DE MAQUINA
        BIT     #100000,R1      ;EL NUMERO ES NEGATIVO?
        BMI     $277
        TST     R1
        BNE     $270
        BIT     #177400,R2      ;R2>255?
        BNE     $270            ;SI,ERROR
```

```
        MOV     #FIASEC,R3
        MOV     R2,R4
        JSR     PC,BINHEX       ;CONVIERTE NUMERO DE BINARIO A HEX.
        MOV     #FIASEC,R5                                  246
        MOV     2(R5),(R5)
        MOV     #20040,2(R5)
        JMP     $245
; DETECTO ERROR
$270:   JSR     PC,VIOLA
        ADD     NBYTES,PLC
        RTS     PC
$277:   JMP     $506
NEXT2:  MOV     #NEXTKN,R3
        CLR     R4              ;R4<=TCAR
        CLR     R5
$301:   CMP     R2,R1           ;FIN DE LINEA
        BGT     $303
        CMPB    (R2),#BLANCO    ;ES BLANCO?
        BEQ     $303
        CMPB    (R2),#TAB       ;ES TAB?
        BEQ     $303
        CMPB    (R2),#',        ;CARACTER=,?
        BEQ     $406
        MOVB    (R2)+,(R3)+     ;NEXTKN<=NEWBUF
        INC     R4              ;R4<=R4+1
        BR      $301
$303:   CMP     R4,#1           ;R4=1?
        BNE     $305
        CMP     NEXTKN,#'X      ;NEXTKN=X?
        BEQ     $410            ;SI
$305:   INC     R2              ;LEE CARACTER
        CMP     R2,R1           ;FIN DE LINEA
        BGT     $310
        CMPB    (R2),#BLANCO    ;ES BLANCO?
        BEQ     $305
        CMPB    (R2),#TAB       ;ES TAB?
        BEQ     $305
        CMPB    (R2),#',        ;CARACTER=,?
        BEQ     $406
$310:   MOV     R1,-(SP)
        MOV     R0,-(SP)
        MOV     #NEXTKN,R3
        JSR     PC,CREA
        JSR     PC,REVISA       ;EVALUA
        JSR     PC,EVALUA       ; OPERANDO
        MOV     #210.,FLAGER    ;FLAGER<=BYTE OVERFLOW
        TST     R1              ; NEGATIVO?
        BNE     $326            ; DETECTA ERROR
        BIT     #177400,R2      ;R2<255?
        BEQ     $330
; MODO EXTENDED
$311:   MOV     #3,NBYTES       ;NBYTES<=3
        MOV     R2,R4
        MOV     #FIASEC,R3
        JSR     PC,BINHEX       ;CONVIERTE NO. DE BINARIO A HEX.
        CLR     R4              ;R4<=CANTIDAD A SUMAR A LA PRIMERA
                                ;LETRA DEL CODIGO DE MAQUINA
        MOV     (SP)+,R0
        MOV     (SP)+,R1
        CMP     X(R0),#6003
        BEQ     $325
        CMP     X(R0),#7003
        BEQ     $320
        INC     R4
$320:   INC     R4
$325:   INC     R4
```

```
        BR      $422
; DETECTO ERROR
$326:   JSR     PC,VIOLA
        RTS     PC
; MODO DIRECTO
$330:   MOV     (SP)+,R0
        MOV     R0,-(SP)
        BIT     #1000,X(R0)     ;ADMITE MODO DIRECTO
        BEQ     $311            ;SOLO ADMITE MODO EXTENDIDO
        MOV     #2,NBYTES       ;NBYTES<=2
        MOV     R2,R4
        MOV     #FIASEC,R3
        JSR     PC,BINHEX       ;CONVIERTE OPERANDO A HEXADECIMAL
        MOV     #FIASEC,R5
        MOV     2(R5),(R5)
        MOV     #20040,2(R5)
        MOV     (SP)+,R0
        MOV     (SP)+,R1
; ACTUALIZA CODIGO DE MAQUINA
        CMP     X(R0),#7003
        BEQ     $335
        MOV     #1,R4
        BR      $422
$335:   ADD     #2,R0
        MOV     X(R0),CODIGO    ;TRAE CODIGO DE MAQUINA
        BR      $423
; MODO INDEXADO
$406:   TST     R4
        BEQ     $410
        MOV     R0,-(SP)        ;SALVA DIRRCCION DE CODIGO
        MOV     R1,-(SP)        ;SALVA FIN DELINEA
        MOV     #NEXTKN,R3
        JSR     PC,CREA
        JSR     PC,REVISA
        JSR     PC,EVALUA
; R1<=HIGH R2<=LOW
        MOV     #210.,FLAGER    ;FLAGER<=BYTE OVERFLOW
        TST     R1              ;NO. NEGATIVO
        BNE     $326
        BIT     #177400,R2      ;R2<255?
        BNE     $326            ;NO MARCA ERROR
        MOV     R2,R4
        MOV     (SP)+,R1
        BR      $415
; MODO INDEXADO
$410:   CLR     R4              ;VECTOR DEL OPERANDO<=0
        MOV     R0,-(SP)
$415:   MOV     #FIASEC,R3
        JSR     PC,BINHEX
        MOV     #FIASEC,R5
        MOV     2(R5),(R5)
        MOV     #20040,2(R5)
        MOV     #2,NBYTES
        MOV     (SP)+,R0
; PREPARATIVOS PARA ACTUALIZAR CODIGO DE MAQUINA
        CMP     X(R0),#6003
        BEQ     $425
        CLR     R4
        CMP     X(R0),#7003
        BEQ     $420
        INC     R4
$420:   INC     R4
$422:   ADD     #2,R0
        MOV     X(R0),CODIGO
        MOV     R1,-(SP)
        CLR     R1
```

```
        CLR     R3
        CLR     R5
        MOV     #1,R2
        MOVB    CODIGO,R3       ;TRAE CODIGO DE MAQUINA
        JSR     PC,HEXBIN       ;CONVIERTE DE HEXADECIMAL A BINARIO
        ADD     R1,R4
        MOV     #SALPLC,R3
        JSR     PC,BINHEX
        MOV     #SALPLC,R3
        MOV     #CODIGO,R4
        MOVB    3(R3),(R4)
        MOV     (SP)+,R1
$423:   MOV     PLC,R4
        MOV     #SALPLC,R3
        JSR     PC,BINHEX       ;CONVIERTE PLC A HEXADECIMAL
        JSR     PC,FORMAT
        JSR     PC,LASFRM
        RTS     PC
$425:   ADD     #2,R0
        MOV     X(R0),CODIGO
        BR      $423
; MODO IMPLICADO Y MODO ACUMULADOR
OP1SEG::
        INC     NBYTES          ;NBYTES<=NBYTES+1
        ADD     #2,R0
        MOV     X(R0),CODIGO    ;TRAE CODIGO DE MAQUINA
        MOV     PLC,R4
        MOV     #SALPLC,R3
        JSR     PC,BINHEX       ;CONVIERTE PLC DE BINARIO A HEX
        MOV     #FIASEC,R5
        MOV     #20040,(R5)
        MOV     #20040,2(R5)
        MOV     #ARGBL1,ARGBLK
        MOV     #9.,ARGBLK+2
; MANDA ESCRIBIR LINEA EN ARCHIVO MOT80.TMP
        JSR     PC,FORMAT
        CLR     R3
        TSTB    (R0)+
        MOV     R0,ARGBLK+4
        MOV     #38.,R4
$401:   INC     R2
        CMP     R2,R1           ;FIN DE LINEA
        BGT     $402
        INC     R3
        MOVB    (R2),(R0)+
        CMP     R3,R4           ;SOLO COPIAMOS 38 CARACTERES DE
                                ;COMENTARIO
        BNE     $401
$402:   SUB     R3,R4           ;R4<=38-R3TST   R4
        BEQ     $404
$403:   MOVB    #BLANCO,(R0)+
        SOB     R4,$403
$404:   ADD     NBYTES,PLC
        MOV     #PON2,R0
        MOV     #ISTROP1,R1
        MOV     #ARGBLK,R2
        CALL    $EDMSG
        OPEN$A  #FDBENS         ;ABRE ARCHIVO MOT80.TMP
        PUT$S   #FDBENS         ;CIERRA ARCHIVO MOT80.TMP
        CLOSE$  #FDBENS
        RTS     PC
; MODO RELATIVO
CLEAR2: ADD     #2,R0
        MOV     X(R0),CODIGO    ;TRAE CODIGO DE MAQUINA
        MOV     #2,NBYTES       ;NBYTES<=2
        CLR     R4
```

```
        MOV     #NEXTKN,R3
$500:   INC     R2                      ;LEEMOS CARACTER
        MOV     #204.,FLAGER            ;FLAGER<=EERROR DE SINTAXIS    249
        CMP     R2,R1                   ;FIN DE LINEA
        BGT     $507
        CMPB    (R2),#BLANCO            ;ES BLANCO?
        BEQ     $500
        CMPB    (R2),#TAB               ;ES TAB?
        BEQ     $500
$501:   MOVB    (R2)+,(R3)+             ;NEXTKN<=NEWBUF
        INC     R4                      ;R4<=TCAR
        CMP     R2,R1                   ;FIN DE LINEA/
        BGT     $503
        CMPB    (R2),#BLANCO            ;ES BLANCO?
        BEQ     $503
        CMPB    (R2),#TAB               ;ES TA?
        BEQ     $503
        BR      $501
$503:   MOV     R1,-(SP)
        MOV     #NEXTKN,R3
        JSR     PC,CREA                 ;EVALUA
        JSR     PC,REVISA
        JSR     PC,EVALUA               ; OPERANDO
;R1<=HIGH R2<=LOW
        CLR     R3
        MOV     PLC,R4
        MOV     #218.,FLAGER            ;FLAGER<=WORD OVERFLOW
        ADD     #2,R4
        BCS     $507
        SUB     R4,R2                   ;CALCULAMOS EL OFFSET
        SBC     R1
        SUB     R3,R1
        BVS     $507                    ;ERROR
        TST     R1                      ;R1<=ES NEGATIVO?
        BNE     $506                    ;SI VE A CALCULAR COMPLEMENTO
        MOV     #208.,FLAGER            ;FLAGER<=ERROR DE BRANCH
        BIT     #177740,R2              ;CUALQUIER BIT DEL 7-15 ENCENDIDO?
        BNE     $507                    ;NO ERROR
$505:   MOV     #FIASEC,R3
        MOV     R2,R4
        JSR     PC,BINHEX               ;CONVIERTE DE BINARIO A HEXADECIMAL
        MOV     #FIASEC,R5
        MOV     2(R5),(R5)
        MOV     #20040,2(R5)
        MOV     PLC,R4
        MOV     #SALPLC,R3
        JSR     PC,BINHEX               ;CONVIERTE PLC DE BINARIO A HEXADECIMAL.
        MOV     (SP)+,R1
        JSR     PC,FORMAT               ;ESCRIBE LINEA ENSAMBLADA
        JSR     PC,LASFRM               ;EN ARCHIVO MOTBO.TMP
        RTS     PC
; PARA NUMEROS NEGATIVOS
$506:   CMP     #177777,R1
        BNE     $507
        BIT     #200,R2
        BEQ     $507
        BIC     #177400,R2
        CMP     R2,#377                 ;NUMERO NEGATIVO DEBE ESTAR ENTRE 377
                                        ;Y 200
        BGT     $507
        BR      $505
; DETECTO ERROR
$507:   ADD     NBYTES,PLC
        JSR     PC,VIOLA
        RTS     PC
        .END
```

```
PIP>TI:=WRIT9.MAC
```



```
        .TITLE  ABRIR
;+
; SIRVE PARA ABRIR EL ARCHIVO DEL USUARIO Y PARA CREAR EL ARCHIVO DE
; TRABAJO COPIA.TMP
;-
;
; DECLARAMOS LOS MACROS QUE SE USAN EN EL PROGRAMA
        .MCALL  FDBDF$,FSRSZ$,FDRC$A,FDOP$A,NMBLK$,FDAT$A
        .MCALL  OPEN$R,CSI$,CSI$1,CSI$2,OPEN$W,CLOSE$,FDRC$R
        .MCALL  GET$S,DELET$,PUT$S,OPEN$A,EXIT$S,QIOW$C
;
NPROG:  .BLKB   30.             ;ESPACIO PARA MANDAR EL NOMBRE DEL
                                ;PROGRAMA A ENSAMBLAR
; AREA PARA LETREROS
MSG00:  .ASCII/    EL NOMBRE DEL ARCHIVO ES:/
        SIZ00=.-MSG00
        .EVEN
MSG0I:  .ASCII/    ERROR AL ABRIR EL ARCHIVO/
        SIZ0I=.-MSG0I
        .EVEN
NEWBUF::.BLKB   80.             ;BUFFER PARA EL ARCHIVO WORK.TMP
        .ENABL  GBL
        FSRSZ$  2               ;2 REGISTROS ABIERTOS PARA REGISTROS DE
                                ;E/S
; ASIGNA ESPACIO PAR EL ARCHIVO DESCRIPTOR DE USUARIO
FDBABR::FDBDF$
        FDRC$A                  ;SECCION DE ACCESO DE REGISTRO
        CSI$                    ;DEFINE CSI
        .EVEN
CSIBLK: .BLKB   C.SIZE          ;NOMBRA CSI CONTOL BLOK Y ASIGNA ALMACE-
                                ;NAMIENTO REQUERIDO
        FDOP$A  1,CSIBLK+C.DSDS,NAME    ;SECCION DE APERTURA DEL ARCHIVO
                                ;ASOCIA LUN=1. USA DEFAULT NAME BLOCK
NAME:   NMBLK$  PRUEBA,DAT      ;ESTABLECE EL NOMBRE DEL ARCHIVO .
; DECLARACIONES DEL FILE DESCRIPTOR BLOCK DEL ARCHIVO WORK.TMP
FDBNEW::FDBDF$
        FDAT$A  R.FIX,FD.CR,80. ;DE LONGITUD FIJA
        FDRC$A  FD.INS,NEWBUF,80.
        FDOP$A  2,,NEWNAM       ;LUN=2
NEWNAM: NMBLK$  WORK,TMP,1      ;DECLARA NOMBRE DEL ARCHIVO
; SE MANDA UN MENSAJE AL USUARIO PARA QUE MANDE EL NUMERO DE CUENTA EN
; DONDE SE ENCUENTRA EL PROGRAMA QUE SE QUIERE ENSAMBLAR Y EL NOMBRE
; DEL PROGRAMA. SI EL PROGRAMA ENSAMBLADOR SE ENCUENTRA EN LA CUENTA
; DEL USUARIO NO ES NECESARIO MANDAR EL NUMERO DE CUENTA.
ABRE::
; MANDA EL MENSAJE
        QIOW$C  IO.WLB,5,2,,,,<MSG00,SIZ00,40>
; RECIBE NUMERO DE CUENTA Y NOMBRE DEL PROGRAMA
        QIOW$C  IO.RLB,5,3,,,,<NPROG,30.>
        CSI$1   #CSIBLK,#NPROG,#30. ;COMANDO DEL ANALIZADOR SINTACTICO
        BCS     ABERR           ;ERROR VE A ABERR
        CSI$2   #CSIBLK,OUTPUT
        BCS     ABERR
; ABRIMOS ARCHIVO DEL USUARIO SOLO PARA LEER
        OPEN$R  #FDBABR,,,,#BUFF,#80.
        BCS     ABERR
; CREAMOS EL ARCHIVO WORK.TMP
        OPEN$W  #FDBNEW
        CLOSE$  #FDBNEW
        RTS     PC
; ERROR AL ABRIR EL ARCHIVO
ABERR:  QIOW$C  IO.WVB,5,1,,,,<MSG0I,SIZ0I,40>
```

```
        EXIT$S
;
;
        .PAGE
        .SBTTL  BARRE
;+
; AQUI QUITAMOS LINEAS EN BLANCO O QUE SOLO TENGAN ETIQUETAS
BARRE:: MOV     #TOKENS,R0      ;LINEA SIN CAMPO DE OPERACION?
        TST     (R0)
        BNE     $2              ;NO, VE A 2
; ERROR
        MOV     #203.,FLAGER    ;FLAGER<=ERROR EN LA SENTENCIA
        JSR     PC,VIOLA
        JMP     CASFIN
$2:     RTS     PC
;
;
        .PAGE
        .SBTTL  COPIA
;+
; PROCEDIMIENTO PARA TENER AL FINAL DEL PRIMER PASO UNA COPIA DEL
; ARCHIVO ORIGINAL SALVO QUE SE TIENEN LAS MACROLLAMADAS EXPANDIDAS
; R1<=TCAR LEIDOS
; R2<=DIRECCION DEL BUFFER QUE SE COPIA
; R3<=DIRECCION DEL BUFFER
;-
COPIA::
; LIMPIA EL BUFFER NEWBUF Y HAZLO DE LONGITUD FIJA
        MOV     #NEWBUF,R3
        MOV     #80.,R2
$70:    MOVB    #BLANCO,(R3)+
        SOB     R2,$70
; NEWBUF<=BUFF
        MOV     #BUFF,R2
        MOV     #NEWBUF,R3
        MOV     R1,R5
$100:   MOVB    (R2)+,(R3)+
        SOB     R5,$100
        OPEN$A  #FDBNEW         ;ABRE EL ARCHIVO COPIA.TMP
        PUT$S   #FDBNEW
        CLOSE$  #FDBNEW         ;CIERRA ARCHIVO COPIA.TMP
        RTS     PC
;
;
        .PAGE
        .SBTTL  TRAER
;+
; TRAE UNA LINEA DEL ARCHIVO DEL PROGRAMA
;-
TRAER::
; LIMPIA EL BUFFER
        MOV     #BUFF,R3
        MOV     #80.,R2
$12:    CLRB    (R3)+
        SOB     R2,$12
        GET$S   #FDBABR,,,CIERRA        ;LEE UNA LINEA
        MOV     F.NRBD(R0),R1   ;R1<=TAMANO DEL REGISTRO
        RTS     PC
CIERRA::CLOSE$  #FDBABR         ;CIERRA ARCHIVO DEL USUARIO
        TST     INMAC           ;ESTAMOS EN MACRODEFINICION?
        BEQ     $13
        MOV     #220.,FLAGER    ;FLAGER<=NO APARECE MEND
        JSR     PC,VIOLA
        CLOSE$  #FDBMAC
$13:    JMP     FPAS1
        .END
```

```
PIP>TI:=WRITIO.MAC
```



```
        .TITLE  SCANER
;+
; IDENTIFICA EL TIPO DE TOKEN Y RECTIFICA LA SINTAXIS
;-
        .ENABL  GBL
SCANER::
; LIMPIA VECTOR TOKENS
        MOV     #TOKENS,R0
        CLR     (R0)
        CLR     2(R0)
        CLR     4(R0)
        CLR     6(R0)
        CLR     10(R0)
        CLR     12(R0)
$1:     MOV     #BUFF,R2        ;R2<=APUNTADOR DEL BUFFER
        ADD     #BUFF,R1        ;R1<=FIN DE LINEA
        DEC     R1
        INC     LINEA           ;LEEMOS UNA LINEA MAS
        CMPB    (R2),#'*        ;TENEMOS LINEA CON COMENTARIO?
        BEQ     $31
        MOV     #202.,FLAGER    ;FLAGER<=ETIQUETA O COD. DE OPERACION
        CMPB    (R2),#BLANCO    ;ES UN BLANCO?
        BEQ     $20             ;NO HAY ETIQUETA
        CMPB    (R2),#TAB       ;ES TAB?
        BEQ     $20             ;NO HAY ETIQUETA
        JSR     PC,LABELS       ;REVIZA SI ESTA BIEN SIMBOLO
        TST     R4              ;R4=0?
        BNE     $30             ;NO,ERROR
$20:    MOV     #202.,FLAGER    ;FLAGER<=ETIQUETA O COD. DE OPERACION
        JSR     PC,INSTRU       ;REVISA QUE ESTE BIEN EL OPERADOR
        TST     R4              ;R4=0?
        BNE     $30             ;NO,ERROR
        RTS     PC
$30:    JSR     PC,VIOLA
$31:    JMP     CASFIN
;
; CUANDO LLEGAMOS AQUI ES POSIBLE QUE ENCONTREMOS ETIQUETA
;
LABELS::
        MOV     #1,R3
        CLR     R4              ;CODIGO DE ERROR
        CMPB    (R2),#'Z        ;CHECAMOS
        BGT     $4              ; SI EL PRIMER
        CMPB    (R2),#'A        ;  CARACTER ES UNA LETRA
        BLT     $4
        ADD     #6,R0           ;GUARDAMOS ETIQUETA APARTIR DE TOKENS+6
        MOV     #205.,FLAGER    ;FLAGER<=ERROR EN ETIQUETA
$2:     MOVB    (R2),(R0)+      ;MUEVE LO LEIDO AL BUFFER DE ESCRITURA
$3:     CMP     R2,R1           ;MAC CARACTERES?
        BGE     $5              ;AQUI SE ACABA LA LINEA Y HAY ETIQUETA
        INC     R2              ;TENEMOS UN NUEVO CARACTER
        CMPB    (R2),#BLANCO    ;CARCTER=BLANCO
        BEQ     $5
        CMPB    (R2),#TAB       ;CARACTER=TAB?
        BEQ     $5              ;HAY TOKEN
        INC     R3              ;TCAR<=TCAR+1
        CMP     R3,#6           ;TCAR>6?
        BGT     $3              ;ELIMINA LOS RESTANTES CARACTERES
        CMPB    (R2),#'Z        ;CHECAMOS
        BGT     $4              ; QUE CARACTER
```

```
        CMPB    (R2),#'0        ;  SEA DIGITO
        BLT     $4              ;    O LETRA
        CMPB    (R2),#'9
        BLE     $2
        CMPB    (R2),#'A
        BGE     $2
; MARCA ERROR
$4:     INC     R4
$5:     RTS     PC
;
;
INSTRU::
        MOV     #TOKENS,R0
        CLR     R3              ;TCAR<=0
        CLR     R4              ;R4=CODIGO DE ERROR
$6:     CMP     R2,R1
        BGE     $12
        INC     R2              ;LEE CARACTER
        CMPB    (R2),#BLANCO    ;ES BLANCO?
        BEQ     $6
        CMPB    (R2),#TAB
        BEQ     $6              ;VE A LEER OTRO CARACTER
        MOV     #207.,FLAGER    ;FLAGER<=COD. OPERACION INDEFINIDO
$7:     INC     R3              ;TCAR<=TCAR+1
        CMPB    (R2),#'Z        ;CARACTER ES NUMERO O LETRA?
        BGT     $11
        CMPB    (R2),#'0
        BLT     $11
        CMPB    (R2),#'9
        BLE     $10
        CMPB    (R2),#'A
        BLT     $11
$10:    MOVB    (R2),(R0)+      ;TOKENS<=BUFFER
        CMP     R2,R1           ;FIN DE LINEA?
        BGE     $12
        INC     R2              ;LEEMOS CARACTER
        CMPB    (R2),#BLANCO    ;BLANCO?
        BEQ     $12
        CMPB    (R2),#TAB
        BEQ     $12
        CMP     R3,#6           ;TOTAL DE CARACTERES<6?
        BLT     $7              ;SIGUE EXAMINANDO BUFFER
; MARCA ERROR
$11:    INC     R4
$12:    RTS     PC
        .END
PIP>TI:=WRIT11.MAC
```

```
        .TITLE BINHEX
;+
; ESTA SUBRUTINA CONVIERTE UN NUMERO DE BINARIO A HEXADECIMAL
;
; R3<=DIRECCION DONDE SE DEPOSITARA EL NUMERO EN HEXADECIMAL
; R4<=NUMERO A TRANSFORMAR
;
; SE ALTERAN R0,R3,R4 Y R5
;-
;
        .MCALL  OPEN$A,PUT$S,CLOSE$
SALPLC::        .BLKW   2
BINHEX::CLR     (R3)+           ;LIMPIAMOS EL AREA DONDE SE DEPOSITARA
        CLR     (R3)+           ;EL NUMERO
        MOV     #4,R0           ;INDICA LOS CARACTERES A TRANSFORMAR
$40:    MOV     R4,R5
        BIC     #177760,R5      ;TOMA 4 BITS PARA TRANSFORMAR A CARACTER
        CMP     R5,#9.          ;R5<9 O IGUAL
        BLE     $51
; PARA SACAR LAS LETRAS A-F. QUEDARAN EN ASCII.
        ADD     #67,R5
        BR      $55
; PARA SACAR LOS NUMEROS 1-9. QUEDARAN EN ASCII.
$51:    ADD     #60,R5
$55:    MOVB    R5,-(R3)        ;GUARDAMOS EL CARACTER
        DEC     R0
        TST     R0
        BEQ     $60
; TRAE LOS SIGUIENTES 4 BITS
        ASR     R4
        ASR     R4
        ASR     R4
        ASR     R4
        BR      $40
$60:    RTS     PC
        .SBTTL TSIMBOL
;+
; PARA TERMINAR EL LISTADO DEL ENSAMBLADOR ESCRIBIMOS LOS SIMBOLOS QUE DEFINIC
; EL USUARIO Y SU CONTENIDO. PARA ELLO EXAMINAMOS LA TABLA FIJA Y
; BUSCAMOS LOS SIMBOLOS CUYO CODIGO SEA IGUAL A 9.
;-
VAR:    .BLKB 6
SW:     .WORD 0
        .ENABL GBL
TSIMBO::MOV     #X,R3           ;R3<=DIRECCION DE LA TABLA FIJA
$1:     MOV     #PON2,R4        ;R4<=APUNTADOR DEL BUFFER DE SALIDA
$2:     ADD     #4,R3           ;R3<=X(I,3) .ESTAMOS EN LA TERCERA
                                ;COLUMNA DE LA TABLA FIJA
        CMP     R3,#X+17774     ;FIN DE LA TABLA FIJA ?
        BGT     $11
        CMP     (R3),#9.        ;TENEMOS ETIQUETA ?
        BNE     $2
        MOV     R3,R5           ;R5<=RENGLON ACTUAL DE LA TABLA FIJA
        SUB     #4,R5           ;R5<=X(I,1). PRIMERA COLUMNA DE LA TABLA
; PONEMOS 6 BLANCOS EN VAR
        MOV     #VAR,R0
        MOV     #3,R2
$3:     MOV     #20040,(R0)+
        SOB     R2,$3
; CONVERTIMOS LOS CARACTERES DEL SIMBOLO DE RADIX-50 A ASCII
```

```
        MOV     #VAR,R0
        MOV     (R5)+,R1
        CALL    $C5TA           ;CARACTERES 1-3
        MOV     (R5)+,R1
        CALL    $C5TA           ;CARACTERES 3-6
        MOV     R3,-(SP)        ;SALVAMOS RENGLON DE LA TABLA
        MOV     R4,-(SP)        ;SALVAMOS EL APUNTADOR DEL BUFFER DE SALIDA
; CONVERTIMOS DE BINARIO A HEXADECIMAL LO ALMACENADO EN EL SIMBOLO
        TST     (R5)+           ;TERCERA PALABRA DEL SIMBOLO
        MOV     (R5),R4         ;R4<=NUMERO A CONVERTIR DE HEXADECIMAL A BINAR
        MOV     #SALPLC,R3      ;R3<=DIRECCION DONDE SE ALMACENA EL NUMERO
        JSR     PC,BINHEX
        MOV     (SP)+,R4        ;R4<=REGRESAMOS EL APUNTADOR AL BUFFER DE SALI
; ESCRIBIMOS EN EL BUFFER DE SALIDA EL SIMBOLO Y SU CONTENIDO
        MOV     #VAR,R0
        MOV     #6,R2
$4:     MOVB    (R0)+,(R4)+
        SOB     R2,$4           ;SEIS CARACTERES DEL SIMBOLO
        MOVB    #BLANCO,(R4)+   ;UN SEPARADOR
        MOV     #4,R2
        MOV     #SALPLC,R3
$5:     MOVB    (R3)+,(R4)+     ;UN NUMERO EN HEXADECIMAL
        SOB     R2,$5           ;FORMADO POR CUATRO CARACTERES
        MOVB    #BLANCO,(R4)+   ;UN BLANCO
        INC     SW              ;TENEMOS ALGO POR ESCRIBIR
        MOV     (SP)+,R3        ;R3<=DIRECCION DE LA TABLA FIJA
        CMP     SW,#5           ;SOLO MANDAMOS EL BUFFER AL ARCHIVO DE ESCRITU
        BLT     $2
; ESCRIBIMOS EN EL ARCHIVO DE SALIDA.
; LLENAMOS CON BLANCOS LO QUE FALTA POR LLENAR DEL BUFFER.
$6:     MOV     #PON2,R2
        ADD     #79.,R2         ;R2<=DIRECCION DEL FIN DEL BUFFER
$7:     CMP     R2,R4
        BLT     $10
        MOVB    #BLANCO,(R4)+
        BR      $7
$10:    OPEN$A  #FDBENS         ;ESCRIBIMOS EN EL ARCHIVO DE SALIDA
        PUT$S   #FDBENS
        CLOSE$  #FDBENS
        CLR     SW              ;APAGAMOS EL SWITCH DE ESCRITURA
        BR      $1
$11:    TST     SW              ;FALTA ALGO POR ESCRIBIR ?
        BNE     $6
        RTS     PC
        .END
PIP>TI:=WRIT12.MAC
```

```
        .TITLE HASH
;+
; PROCEDIMIENTO PARA BUSCAR UN TOKEN (LLAVE) EN UN TABLA
; -
SW::    .WORD   0               ;INDICA SI ES PRIMERA VEZ QUE SE RESUEL-
                                ;VE UNA COLISION
        .ENABL  GBL
HASAUX::CLR     SW
        CALL    $CAT5           ;CONVIERTE 3 CARACTERES A RADIX-50
        MOV     R1,R3           ;R3<=3 PRIMEROS CARACTERES
        CALL    $CAT5
; CALCULA HASH
        MOV     R1,R4
        ASR     R1              ;ROTA R1
        SWAB    R1              ;BYTE1/BYTE0<=BYTE0/BYTE1
        XOR     R3,R1           ;R1<=R3 OR EXCLUSIVO R1
ARRAY:  BIC     #176000,R1      ;R1<2000 BASE 8.
        MUL     #4,R1           ;PARA COLOCARLO EN LA PRIMERA COLUMNA
        ASL     R1              ;DE LA TABLA
VACIO:  TST     X(R1)           ;SE ENCUENTRA ALGO EN ESA DIRECCION
        BEQ     HASER           ;NO, ERROR
; ES IGUAL A TOKEN?
        CMP     X(R1),R3
        BNE     LLENO           ;NO RESUELVE COLISION
        ADD     #2,R1
        CMP     X(R1),R4
        BNE     LLENO1          ;NO, RESUELVE COLISION
        ADD     #2,R1
        MOV     R1,R0           ;R0<=DIRECCION EN LA TERCERA COLUMNA DE
                                ;TABLA
        RTS     PC
LLENO1: TST     -(R1)
LLENO:  JSR     PC,FULL         ;RESUELVE COLISION
        BR      ARRAY           ;REGRESA A BUSCAR TOKEN
HASER:  CLR     R3              ;R3=0 CODIGODE ERROR
        RTS     PC
        .PAGE
        .SBTTL  FULL
;+
; PROCEDIMIENTO QUE RESUELVE COLISIONES. CALCULA RHO.
;-
FULL::
        TST     SW              ;PRIMERA VEZ QUE SE RESUELVE LA
                                ;COLISION
        BNE     RANDOM          ;NO
        MOV     #1,R5           ;R5<=1
        MOV     #1,SW           ;SW<=1
        MOV     R1,R2           ;R2<=DIRECCION ORIGINAL DE HASH
RANDOM: MUL     #5,R5           ;R5<=R5*5
        BIC     #170000,R5      ;ELIMINAMOS BITS 12-15
        MOV     R5,R0
        ASR     R0
        ASR     R0              ;RHO<=R0/4
        TST     R0              ;SE RECORRIO TODA LA SERIE RANDOM?
        BEQ     FINLOP          ;SI,LA TABLA SE SATURO
        ADD     R2,R0           ;L<=L+RHO
        MOV     R0,R1
        RTS     PC
FINLOP: MOV     #221,,FLAGER    ;FLAGER<=TABLA LLENA
        JMP     VIOLA
;
;
        .PAGE
        .SBTTL  HASLAB
..
```

```
;+
; PROCEDIMIENTO PARA COLOCAR UN TOKEN EN TABLA EMPLEANDO LA FUNCION
; DE HASH                                                    257
;-
        .ENABL  GBL
HASLAB::
        CLR     SW
        CALL    $CAT5           ;CONVIERTE 3 CARACTERES A RADIX-50
        MOV     R1,R3           ;R3<=3 PRIMEROS CARACTERES
        CALL    $CAT5           ;CONVIERTE 3 CARACTERES A RADIX-50
        MOV     R1,R4
        ASR     R1              ;ROTA R1
        SWAB    R1              ;BYTE1/BYTE0<=BYTE0/BYTE1
        XOR     R3,R1           ;EFECTUA OR EXCLUSIVO
FIJAT:  BIC     #176000,R1      ;R1<=COLOCA BIT 0-11
        MUL     #4,R1           ;PARA COLOCAR TOKEN EN LA PRIMERA
        ASL     R1              ;COLUMNA DE LA TABLA
COLOCA: TST     X(R1)           ;SE ENCUENTRA ALGO EN ESA LOCALIDAD
        BNE     PLEIN           ;VE A RESOLVER COLISION
; COLOCA TOKEN EN TABLA
        MOV     R3,X(R1)
        ADD     #2,R1
        MOV     R4,X(R1)
        ADD     #2,R1
        MOV     FLAG,X(R1)      ;COLOCA CLAVE DEL TOKEN
        ADD     #2,R1
        MOV     TEMPLC,X(R1)    ;COLOCA PLC U OTROS VALORES
        RTS     PC
PLEIN:  CMP     X(R1),R3
        BNE     SATURA          ;VE A RESOLVER COLISION
        ADD     #2,R1
        CMP     X(R1),R4
        BNE     SATUR1
        MOV     #206.,FLAGER    ;FLAGER<=SIMBOLO REDEFINIDO
        CLR     R3              ;R3<=0 CODIGO DE ERROR
        RTS     PC
SATUR1: TST     -(R1)
SATURA: JSR     PC,FULL
        BR      FIJAT
;
;
        .PAGE
        .SBTTL SUBHASH
HASH::
        JSR     PC,HASAUX       ;COLOCA TOKEN
        TST     R3
        BNE     100$
        JSR     PC,VIOLA
        JMP     CASFIN
100$:   RTS     PC
HASETI::
        JSR     PC,HASLAB
        TST     R3
        BNE     400$
        JMP     VIOLA
        JMP     CASFIN
400$:   RTS     PC
        .END
PIP>
```