ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

MACROCOMPILADOR DE BASIC A ASSEMBLER DEL MICROPROCESADOR M6800

POR

JORGIO VICENTE TORO RIOS

TESIS PREVIA A LA OBTENCION DEL TITULO DE INGENIERIO EN LA ESPECIALIZACION DE INGENIERIA ELECTRONICA Y TELECOMUNICACIONES EN LA ESCUELA POLITECNICA NACIONAL

QUITO

NOVIEMBRE DE 1987

AGRADECIMIENTO

Al Ing. César Esquetini, director del presente trabajo;
al Ing. Edgar Torres Proaño,- quien propuso el tema;
y a todas las personas que con su aporte hicieron posible la realización de esta tesis.

MACROCOMPILADOR DE BASIC A ASSEMBLER DEL MICROPROCESADOR M6800.

lógica del problema a resolverse, antes que a la forma de trabajo del computador.

Un computador no entiende ni el lenguaje ensamblador ni el de alto nivel, sino únicamente el de máquina. Por lo tanto es imprescindible que se disponga de algún medio que, teniendo como entrada un programa escrito en uno de estos dos lenguajes (ensamblador o de alto nivel), entregue al computador un programa equivalente, pero escrito en lenguaje de máquina. Este medio en general se lo denomina procesador de lenguaje. En un procesador de lenguaje el programa de entrada se lo llama programa fuente, y el de salida programa objeto.

Los procesadores de lenguajes se los clasifica en: traductores e intérpretes.

"Un traductor es un programa que traduce un programa fuente en un programa objeto equivalente. Un intérprete para un lenguaje fuente, acepta un programa fuente escrito en dicho lenguaje y lo ejecuta. La diferencia entre un traductor y un intérprete es que el intérprete no produce un programa objeto a ejecutar, sino que ejecuta el mismo programa fuente" (1)

---

(1) GRIES David, Construcción de compiladores, Paraninfo, Madrid 1975, p. 16.

TRADUCTOR CROSS-ASSEMBLER PARA EL MICROPROCESADOR M6800, Hernandez A. Miriam, y

SIMULADOR DEL SISTEMA MICROPROCESADOR M6800 USANDO COMO BASE EL CROSS-ASSEMBLER DEL MISMO, Zúñiga H. Edison.

Estas dos tesis estan disponibles en el computador DM-250 GENERAL AUTOMATION en el área de Procesamiento Automático del I.E.TEL. Región 1.

Se escogió el BASIC como el lenguaje de alto nivel para escribir los programas fuente, principalmente por dos razones: el BASIC es lenguaje más fácil de aprender, y es el lenguaje que más usuarios tiene en la actualidad. Esto permitirá que más personas se beneficien de este trabajo.

Como un aporte didáctico para las personas que se interesen en poder observar más de cerca el proceso de compilación, se prevee generar un código intermedio totalmente independiente de la arquitectura del microprocesador a cuyas instrucciones se traducirá finalmente el programa fuente. Esto generaliza el proceso de compilación a más de generar un programa traducido a código intermedio que es más fácil de entender que el programa objeto mismo.

Para desarrollar el compilador se usará el lenguaje PASCAL del computador IBM de la ESCUELA POLITECNICA NACIONAL. Este lenguaje dispone de facilidades empleadas con frecuencia en este tipo de trabajo, comp por ejemplo se pueden definir arreglos cuyos registros sean variantes, es decir registros que son diferentes y que sin embargo pertenecen a la misma estructura de datos. Ademas el PASCAL es un lenguaje estructurado que facilita la comprensión de los programas escritos con él; esto será una gran ayuda para quienes deseen analizar los programas que conforman el compilador.

El uso combinado del presente trabajo con las dos tesis anteriormente mencionadas constituirá un recurso valioso para el desarrollo de sistemas basados en el Microprocesador M6800. La utilidad del conjunto se centra en el hecho de que un programador con conocimientos fundamentales del BASIC, podrá realizar gran cantidad de aplicaciones con el Microprocesador M6800 sin conocer necesariamente ni el lenguaje ensamblador ni el lenguaje de máquina del mismo.

Contenido del trabajo.-

Para escoger una solución conveniente al compilador planteado en el presente trabajo, se analiza la teoría de

TEORIA BASICA PARA EL DISEÑO DE TRADUCTORES

Este capítulo contiene en primer lugar conceptos sobre los diferentes tipos de lenguajes de programación (lenguaje de máquina, ensamblador y de alto nivel), sus principales características, diferencias, ventajas y desventajas. Como segundo punto se trata conceptos sobre los traductores, y en la tercera parte se analiza la estructura general de un traductor y sus respectivas partes. A medida que se desarrolle este último punto se irá escogiendo la estructura particular que tendrá el compilador del presente trabajo.

2.1.- LENGUAJES DE PROGRAMACION

"En general un lenguaje de programación es una notación mediante la cual las personas pueden, por medio de un programa, comunicar algoritmos a una computadora para que esta los procese, entienda y ejecute" (1)

Los lenguajes de programación pueden clasificarse en tres grupos: lenguajes de máquina, lenguajes ensamblado-

---

(1) HERNADEZ A. Miriam, Traductor Cross-Assembler para el Microprocesador M6800, Escuela Politécnica Nacional (Tesis de grado) 1982, p.5.

2.1.2.- Lenguaje ensamblador.-

"El lenguaje ensamblador elimina los porblemas de tipo a) al utilizar códigos nemotécnicos en lugar de códigos binarios y direcciones simbólicas de memoria en lugar de direcciones numéricas. Los símbolos de los códigos de operación son fijos para cada lenguaje y las direcciones simbólicas las puede elegir el programador dentro de unas ciertas reglas" (1)

2.1.3.- Lenguaje de alto nivel.-

"Un lenguaje se lo considera de alto nivel si permite describir programas en una manera directa más cercana a la lógica de los procedimientos que se van a realizar, antes que orientada a la forma de trabajo del computador" (2)

Las principales ventajas de utilizar lenguajes de alto nivel son:

---

(1) Enciclopedia INFORMATICA, Editorial CINCO (IV tomos), Bogotá 1984, I p.50.

(2) HERNANDEZ A. Miriam, Traductor Cross-Assembler para el Microprocesador M6800, Escuela Politécnica Nacional (Tesis de grado) 1982, p. 6.

a) Un programa escrito en lenguaje de alto nivel puede ser utilizado en distintos computadores, y

b) Una persona puede aprender este tipo de lenguajes, realizar programas y corregir estos, en un tiempo relativamente corto.

Sus desventajas principales son:

a) Se incrementa la ocupación de memoria interna, tanto por parte del programa traductor como por el propio programa objeto resultante, y

b) No se aprovechan las posibles ventajas de la arquitectura interna del sistema.

2.2.- TRADUCTORES, COMPILADORES

"Un traductor es un programa que toma como entrada un programa escrito en un lenguaje de programación (el lenguaje fuente) y produce como salida un programa equivalente en otro lenguaje. Si el lenguaje fuente es un lenguaje de alto nivel tal como el FORTRAN, PL/I, COBOL, etc, y el lenguaje objeto es un lenguaje de bajo nivel tal como el lenguaje ensamblador o lenguaje de máquina, entonces dicho traductor se denomina compilador" (1)

---

(1) AHO Alfred and Ullman Jeffrey, Principles of Compiler Design, Addison-Wesley, Reading Mass. 1977, p.1.

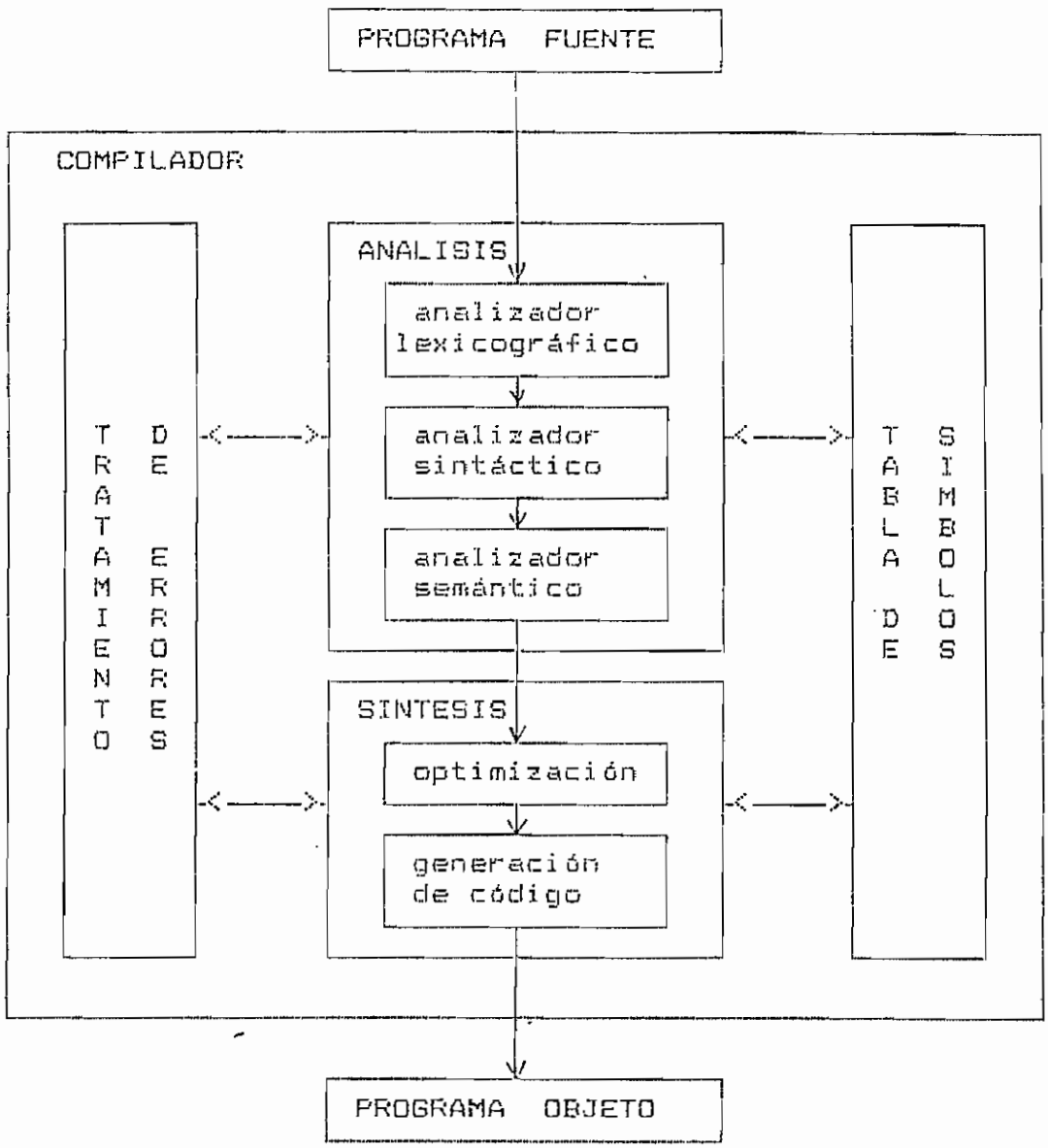


fig.2.1.- Estructura de un compilador.

Para que el trabajo del compilador se facilite, se asigna un código a cada tipo de símbolo para su representación interna. Este código puede ser por ejemplo un número o una palabra diferente para cada tipo de símbolo. En el presente trabajo se usará una palabra diferente para cada tipo de símbolo como se indica en la tabla 2.1.

TIPO de SIMBOLO	CODIGO INTERNO
indefinido	INDEF
identificador de variable	VRBLE
identificador de función	FUNCN
entero	ENTER
palabras reservadas	PRESV
delimitadores	DELIM

Tabla 2.1.- Representación interna de los tipos de símbolos.

Por ejemplo en la siguiente proposición:

50 LET A = 5 + (B * C) / 18

el explorador analiza los símbolos de izquierda a derecha y entregará a su salida los resultados que se indican en la tabla 2.2.

# de SIMBOLO EXTRAIDO	SALIDA	SIGNIFICADO	
		CLASE	SIMBOLO
1	ENTER, 50	entero	50
2	PRESV, LET	palabra reservada	LET
3	VRBLE, A	identif. de variable	A
4	DELIM, =	delimitador	=
5	ENTER, 5	entero	5
6	DELIM, +	delimitador	+
7	DELIM, (	delimitador	(
8	VRBLE, B	identif. de variable	B
9	DELIM, *	delimitador	*
10	VRBLE, C	identif. de variable	C
11	DELIM,)	delimitador	)
12	DELIM, /	delimitador	/
13	ENTER, 18	entero	18

Tabla 2.2.- Ejemplo de resultados entregados por el explorador

Un explorador se puede programar como un conjunto separado que realice un análisis completo del programa fuente y que entregue al analizador sintáctico una tabla conteniendo el programa fuente en su representación interna. Otra alternativa es la de un subprograma llamado

por el analizador sintáctico cuando necesite un nuevo símbolo. Cuando se le llama, el subprograma reconoce el siguiente símbolo del programa fuente y lo entrega al analizador sintáctico.

En el primer caso el explorador constituirá una fase independiente del compilador, y en el segundo caso sería dependiente del analizador sintáctico. En el presente trabajo se escoge la segunda alternativa como solución ya que se propuso tener una fase que hiciera la parte de análisis, de la cual también forma parte el explorador.

2.3.2.- ANALIZADOR SINTACTICO

"El analizador sintáctico identifica las estructuras mayores del programa (oraciones, declaraciones, expresiones, etc), usando los símbolos entregados por el analizador lexicográfico" (1)

El analizador sintáctico tiene dos funciones principalmente:

a) Chequear que los símbolos que aparecen en su entrada sean patrones sintácticamente válidos de acuerdo a

---

(1) PRATT Terrence , Programming Languages, Design and Implementation, Prentice-Hall, Englewood 1975, p.296

lisis semántico es el puente entre las partes de análisis y síntesis de la traducción". (1)

El analizador semántico se encarga principalmente de analizar el significado de las estructuras. Por ejemplo si se tiene una asignación de la forma:

variable = expresión

el analizador semántico comprobará que la variable y la expresión sean de tipo compatible.

Cuando el lenguaje fuente o los objetivos del proyecto son bastante complicados, puede interesarnos que el compilador genere, como resultado del análisis, un programa fuente escrito en código intermedio, con el cual se trabajará en la síntesis para obtener el programa en código objeto.

Generación de código intermedio

El generador de código intermedio usa la estructura producida por el analizador sintáctico y semántico para producir un conjunto de instrucciones simples. Hay va -

---

(1) PRATT Terrence, Programming Languages, Design and Implementation, Prentice-Hall, Englewood 1975, p.296

rios estilos de códigos intermedios, tres de los pricipales son:

a) Notación Polaca.-

"La notacion Polaca se emplea para representar expresiones aritméticas o lógicas de forma que especifica simple y exactamente el orden de cálculo de los operadores; además, no son necesarios los paréntesis. En esta notación los operadores van inmediatamente después de los operandos". (1)

En la tabla 2.3 se da ejemplos de la forma de escribir expresiones aritméticas en la notación polaca.

expres. aritmética	notación polaca
X+Y	XY+
(X-Y)+Z	XY-Z+
X-(Y+Z)	XYZ+-
X*(Y+Z)*W	XYZ+*W*

Tabla 2.3.- Ejemplo de expresiones aritméticas en notación Polaca.

(1) GRIES David, Construcción de Compiladores, Paraninfo, Madrid 1975, p.291.

Una bifurcaión incondicional del tipo:

GO TO 50

se escribirá como:

50 BRL

donde: - 50 es una etiqueta, y

- BRL significa bifurcar a una etiqueta.

Una proposición condicional tal como:

IF expresión THEN propos.1 ELSE propos.2

tendría la forma:

expresión OP1 BZ propos.1 OP2 BR propos.2

admitiendo que 0=FALSE y TRUE<>0 para cualquier expresión los otros términos significan lo siguiente:

- OP1, OP2 son constantes que nos indican el comienzo de la proposición 1 y proposición 2 respectivamente,
- BZ es un operador con dos operandos: expresión y OP1. Significa que si (y solo si) la expresión es cero,entonces loa siguientes símbolos a calcularse serán los que se inician en OP1.
- BR es un operador con un operando: OP2. Su significado es cambiar el orden en que se evalúan los símbolos empezando por el que nos indica OP2.

OPERADOR	OPERANDOS	SIGNIFICADO DEL CUARTETO
BR	i	bifurcar al cuarteto i
BZ (BP,BN)	i,p	bifurcar al cuarteto i si el valor descrito por p es cero (positivo o negativo).
BG (BL,BE)	i,p1,p2	bifurcar al cuarteto i si el valor descrito por p1 es mayor (menor o igual) que el valor descrito por p2

Tabla 2.4.- Ejemplos de operadores que se usan para generar cuartetos.

c) Tercetos.-

"Una de las desventajas de los cuartetos es el número de variables temporales que debemos describir; con el empleo de tercetos se elimina este problema. El terceto tiene la forma:

operador operando1 operando2

No hay campo de resultado; en vez de esto, cualquier operando que sea el resultado de esta operación se refiere directamente a ella. Así A+B*C se representaría

[1] * B, C

[2] + A, [1]

con esta notación, [1] se refiere al resultado del terceto uno, no es pues la constante 1". (1)

Para generar saltos se usa operadores como los que se describió en la tabla 2.4.

d) Código intermedio en el presente trabajo.-

El código intermedio proyectado para el presente trabajo está basado en un grupo de instrucciones simples independientes de la arquitectura del microprocesador con cuyas instrucciones se generará el programa objeto. Este código intermedio seguirá la notación Polaca (llamada a veces postfija). El formato de estas instrucciones se da a continuación:

# de INSTRUCCION	OPERADOR	OPERANDO

fig 2.2.- Formato de instrucciones en codigo intermedio.

Donde:

- # de instrucción se refiere al número de la instrucción generada en el programa en código intermedio.
- operador indica la acción que realizará la instrucción. Los operadores usados y su significado se muestran en la tabla 2.3.

---

(1) GRIES David, Construcción de Compiladores, Paraninfo, Madrid 1975, p. 299

- el operando puede ser una dirección o un número literal.

	OPDOR	SIGNIFICADO
1	CNU	Carga de un número en una loc. de memoria
2	CVA	Carga de una variable en una loc. de memor.
3	ALM	Almacena un valor en la dirección de una variable (corresponde a una asignación)
4	LLS	Activar una subrutina (corresponde a una instrucción de llamado de subrutina)
5	RTS	Fin de subrutina (corresponde a un retorno de subrutina).
6	INS	Asigna espacio en memoria a base de incrementar el contador de localidades de memor.
7	SAI	Salto incondicional de control
8	SAC	Salto condicional de control
9	END	Fin de programa
10	OPR	Conjunto de operadores aritméticos, de relación, lógicos y de concatenación

Tabla 2.5 .- Código intermedio: operadores y su significado.

OPERADOR	CODIGO
<	1
<=	2
=	3
<>	4
>=	5
>	6
-	7
+	8
-	9
OR	10
*	11
/	12
AND	13
|	14
NOT	15

Nota.-El operador "-" (codigo 7), es el menos unario

Tabla 2.6.- Operadores OPR: código y su significado.

Como se puede apreciar, el grupo de las instrucciones usadas para generar el presente código intermedio son sólo diez. Este número reducido hace que el código sea comprensible muy fácilmente para el lector. El operador 14 sirve para concatenar strings.

contador de código	código intermedio	
	operador	operando
...	...	...
11	CVA	5
12	CNU	1
13	OPR	3
14	SAC	19
15	CNU	58
16	CVA	5
17	OPR	8
18	ALM	6
19	END	

Tabla 2.7.- Código intermedio generado.

como se aprecia, la expresión aritmética 58+A (en lenguaje fuente), se ha trnasformado en:

```
CNU  58    = cargue el número 58
CVA  5     = cargue la variable de la localidad 5
OPR  8     = realize la operación suma
```

que bien podría expresarse como:

```
58 A +
```

2.3.4.- OPTIMIZACION

"En la optimización se reordenan y cambian las operaciones en el programa que se está traduciendo a fin de hacerlo más eficiente. Para cada programa fuente hay muchos programas objeto que realizan un trabajo equivalente,unos programas objeto ocupan un menor tamaño y son más rápidos. Con el proceso de optimización se intenta producir este código objeto más eficiente" (1)

El presente trabajo se ha proyectado para programas fuentes pequeños, por lo tanto los programas objeto que se obtengan no seran exesivamente grandes; es decir que si se implementara una optimización, en poco se disminuiría el tamaño del programa objeto, como poca sería la disminución del tiempo de ejecución. Por estas razones no se incluirá el proceso de optimización en el presente trabajo. Sin embargo se describirá los casos más comunes en donde se aplica la optimización. Estos casos son:

- Reducción,

---

(1) ANALES DE LAS JORNADAS DE INGENIERIA ELECTRICA Y-ELECTRONICA, Cross-Assemblres y Simuladores en el Desarrollo de Sistemas Basados en Microprocesadores. Estudio, Implementación y Ejemplos de Aplicación para el Z80/Z80A, Torres Proaño, Vol.5 (1984), p.72 - 82.

- Eliminación de operaciones redundantes, y
- Optimización en lazos.

a) Reducción

"La reducción es el proceso de realizar, en tiempo de compilación, operaciones del programa fuente cuyos valores de los operandos son conocidos, por lo que no es necesario realizarlos en tiempo de ejecución. La reducción se aplica principalmente a los operadores aritméticos ya que aparecen frecuentemente en programas fuentes". (1)

b) Eliminación de operaciones redundantes.-

"una operación i es redundante si existe una operación j y si ninguna de las variables de las que depende la operación se cambia por una tercera operación que está entre la i y la j." (2)

---

(1) GRIES David, Construcción de compiladores, Editorial Paraninfo, Madrid 1975, p.435.

(1) Idem (1), p.437

c) Optimización en lazos

Una operación es invariante en un lazo si ninguno de sus operandos cambia mientras se ejecuta el lazo. Una optimización efectiva es el sacar la operación invariante fuera del lazo. Por ejemplo, si en tiempo de compilación sacamos una multiplicación fuera de un lazo que se repite mil veces, se ha evitado la ejecución de 999 multiplicaciones.

2.3.5.- GENERACION DE CODIGO

Es el último proceso de la traducción. Aquí el programa traducido código intermedio ha sido optimizado y debe ser formado en oraciones en lenguaje ensamblador o lenguaje de máquina que será el resultado del proceso de la traducción (1).

En el presente trabajo el código a generarse será en lenguaje ensamblador del Microprocesador M6800. Para explicar este punto se usará el mismo ejemplo que se usó en

---

(1) PRATT, Terrence; Programming Languajes, Design and Implementacion, Prentice-Hall, Englewood Cliffs N.J. 1975, p.300.

contador d código ensamb.	contador de memoria	código ensamblador		instrucción correspondiente en cod. interm.
		operador	operando	
...	...			
11		NAM	PRUEB	
12		ORG	1000	
13	130	LDAA	5	CVA 5
14	133	STAA	8	
15	136	LDAA	#1	CNU 1
16	138	STAA	9	
17	141	LDAA	9	OPR 3
18	144	CMPA	8	
19	147	BEQ	5	
20	149	CLR	8	
21	152	BRA	5	
22	154	LDAA	#1	
23	156	STAA	8	
24	159	TST	8	SAC 19
25	162	BEQ	27	
26	164	LDAA	#58	CNU 58
27	167	STAA	8	
28	170	LDAA	5	CVA 5
29	173	STAA	9	

continúa....

30	176	LDAA	9	OPR 8
31	179	ADDA	8	
32	182	STAA	8	
33	185	LDAA	8	ALM 6
34	188	STAA	6	
35		END		

Tabla 2.8.- Código en lenguaje ensamblador.

2.3.6.- TABLAS DE SIMBOLOS

"Un compilador necesita coleccionar información acerca de todos los identificadores que aparecen en el progrma fuente. Por ejeplo un compilador necesita conocer si una variable representa a un número entero o real, que longitud tiene un arreglo, cuántos argumentos espera una función, y asi sucesivamente.

La información acerca de estos objetos es recogida al inicio de la compilacion (en el análisis lexicográfico, sintáctico y semántico) e ingresada en la tabla de símbolos". (1)

---

(1) AHO, Alfred and ULLMAN, Jeffrey; Principles of Compiler Design, Addisson-Wesley, Reading Mass 1977, p.20.

A esta tabla de símbolos deben tener acceso todas las partes del compilador porque ya sea tendrán que ingresar o consultar información de la misma.

Organización de una tabla de símbolos.-

La forma general de las tablas es:

	argumento	valor
entrada 1		
entrada 2		
...		
entrada n		

Tabla 2.9.- Forma general de una tabla de símbolos.

donde los argumentos serán los nombres de los identificadores, y los valores sus atributos (tales como si el identificador es de una variable subindicada, cuántos subíndices tiene, etc).

La parte valor de un identificador de la tabla se lo llama también descriptor, ya que describe al identificador. El descriptor puede necesitar la siguiente información en el caso de variables o procedimientos:

3.1.- DEFINICION DE UN LENGUAJE DE PROGRAMACION: SINTAXIS Y SEMANTICA.

Definición.-

"Todo lenguaje está basado en un vocabulario. Sus elementos se llaman, normalmente, palabras; en el campo de los lenguajes formales, sin embargo se llaman símbolos (básicos). Es una característica de los lenguajes que algunas secuencias de palabras sean consideradas frases correctas, y otras sean consideradas incorrectas o mal formadas. La gramática, sintaxis o estructura del lenguaje determina que una secuencia de palabras sea una frase correcta o no ". (1)

3.1.1.- Sintaxis y Semántica.

"Sintaxis es un conjunto de reglas formales que especifican la composición de los programas a base de letras, dígitos y otros caracteres. Por ejemplo, las reglas de la sintaxis especifican en PASCAL que dos sentencias deben ir separadas por un ";" o que la declaración de tipos ha de

---

(1) WIRTH, Niklaus; Algoritmos+Estructuras de Datos= Programas, Univ. Politécnica de Madrid, Madrid, 3ra impresión, p. 297.

la de variables.

La semántica es el conjunto de reglas que especifican el significado de cualquier sentencia (frase) sintácticamente correcta y escrita en un determinado lenguaje. Dicho significado puede expresarse mediante la correspondencia entre cada construcción del lenguaje en un dominio cuya semántica es conocida." (1)

Para definir la sintaxis de un lenguaje se utilizan ya sea el metalenguaje BNF (Backus-Naur Form) o los grafos sintácticos.

3.1.2.- Notación BNF. (2)

Algunas de las definiciones que utiliza esta notación son las siguientes:

Terminal.- Se dice que un símbolo es terminal cuando tiene entidad propia y se describe por si mismo, es decir, no requiere ninguna explicación.

---

(1) SANCHEZ, Gonzalo y VALVERDE, Juan; Compiladores e Intérpretes, Diaz de Santos, Madrid, 1ra edición-1984, p. 18.

(2) Idem (1), p. 23.

No terminal.- Se dice que un símbolo es no terminal cuando requiere una explicación mediante una regla o producción.

Metasímbolos.- Son aquellos símbolos de la notación utilizados para distinguir los elementos y propiedades de una regla. Por ejem. "|".

En la notación BNF se utiliza una serie de reglas o producciones cuyo objetivo es la descripción de unidades sintácticas o símbolos no terminales. Los símbolos no terminales van encerrados entre los paréntesis regulares "<" y ">", y en cada regla van siempre a la izquierda del metasímbolo "::=", es decir, se considera no-terminal a algo que debe aparecer más adelante como parte izquierda de una regla. La derecha de una regla es una combinación cualquiera de símbolos terminales y no terminales. Por ejemplo:

<SENTWHILE> ::= **while** <EXPRESION> **do** <SENTENCIA>

donde **while** y **do** son terminales y SENTWHILE, EXPRESION y SENTENCIA son no terminales.

Si hay varias reglas con un no terminal común a la izquierda, se pueden abreviar utilizando el metasímbolo "|" denominado alternativa, por ejemplo:

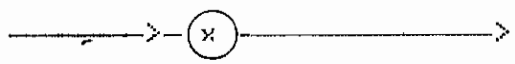
Reglas para construir un grafo (1)

1) Para todo símbolo no terminal que tenga un conjunto de producciones:

$$\langle A \rangle ::= \langle E_1 \rangle \mid \langle E_2 \rangle \mid \ldots \mid \langle E_n \rangle$$

se construye un grafo sintáctico <A> cuya estructura viene determinada por la parte derecha de su conjunto de producciones según las reglas 2) a 6).

2) Cada aparición de un símbolo terminal x en un $\langle E_i \rangle$ corresponde a una instrucción que reconozca este símbolo seguido del avance de posición de lectura al símbolo siguiente de una secuencia de entrada. Esto se representa en un grafo por un arco etiquetado con x dentro de un círculo:

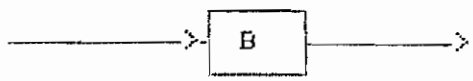


3) Cada aparición de un símbolo no terminal <B> en un $\langle E_i \rangle$ corresponde a una activación del reconocedor <B>. Esto se representa en el grafo con un arco

---

(1) WIRTH, Niklaus; Algoritmos + Estructuras de Datos = Programas, Univ. Politécnica de Madrid, Madrid, 3ª impresión 1984, p. 307.

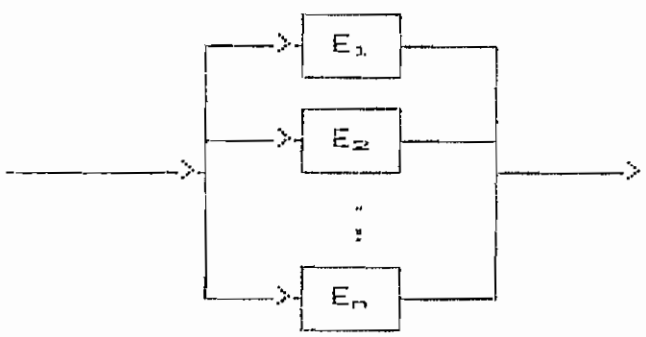
etiquetado con B dentro de un cuadrado:



4) Para la producción de la forma:

$$\langle A \rangle ::= \langle E_1 \rangle \mid \langle E_2 \rangle \mid \ldots \mid \langle E_n \rangle$$

se construye el grafo:



donde cada $\langle E_i \rangle$ se obtiene aplicando las reglas 2) a 6) a $\langle E_i \rangle$.

5) Para una <E> de la forma :

$$\langle E \rangle ::= \alpha_1 \ \alpha_2 \ \ldots \ \alpha_m$$

se construye el grafo:

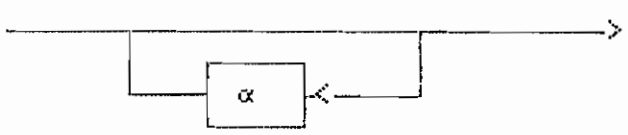
donde cada α_i se obtiene aplicando las reglas 2) a 6)

a α_1.

6) Para un <E> de la forma:

<E> ::= (α)

se construye un grafo:



donde α se obtiene aplicando las reglas 2) a 6) a α.

A los grafos construidos en base a estas reglas se los llama deterministas, siempre y cuando cumplan dos restricciones:

- En cada nudo de ramificación, el camino a seguir debe poder seleccionarse en base a los primeros símbolos de las ramas. Esto implica que distintas ramas deben empezar con distintos símbolos, y
- Si hay un grafo <A> que puede ser recorrido sin leer ningún símbolo de entrada, hay que etiquetar esta rama nula con todos los símbolos que puedan seguir a <A> (ver regla 6).

Por las ventajas que presenta el grafo sintáctico con respecto a la notación BNF, en el presente trabajo se op

tará por utilizar los grafos para describir la sintaxis del lenguaje fuente.

3.1.4.- Traducción de grafos en programas.

Teniendo un grafo sintáctico, se puede traducir este a un programa de alto nivel siguiendo algunas reglas. Estas reglas se aplican suponiendo que se tiene un programa principal que contiene una rutina para evanzar al siguiente símbolo, y los procedimientos correspondientes a los distintos objetivos parciales. Se supondrá también que la frase a tratar se la representa por un fichero INPUT, la variable CH contiene siempre el siguiente símbolo a tratar, y T(s) representa la instrucción que se obtiene al traducir el grafo s.

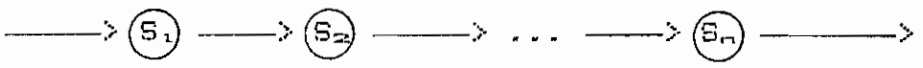
Se usará el lenguaje PASCAL para escribir estos programas, pues este lenguaje es el que se usará para implementar el compilador como se explicó anteriormente. El programa principal está formado por la instrucción inicial que lee el primer caracter, seguida de una instrucción que activa el procedimiento del objetivo principal del análisis. Las rutinas individuales correspondientes a los objetivos parciales se obtienen aplicando las siguientes reglas:

Reglas para traducir grafos en programas. (1)

1) Por medio de sustituciones apropiadas, reducir el sistema de grafos a un número de grafos individuales lo más pequeño posible.

2) Traducir cada grafo resultante en una declaración de procedimiento, según las reglas 3) a 7).

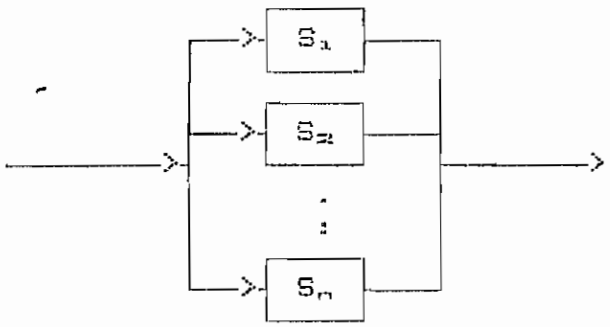
3) En una secuencia de elementos:



se traduce por la instrucción compuesta:

```
BEGIN
T(S1); T(S2); ... ; T(Sn)
END
```

4) Una bifurcación de elementos:



---

(1) WIRTH, Niklaus; Algoritmos + Estructuras de Datos = Programas, Univ. Politécnica de Madrid, Madrid, 3ra impresión 1984, p. 310.

Fig.3.1.- Grafos separados.

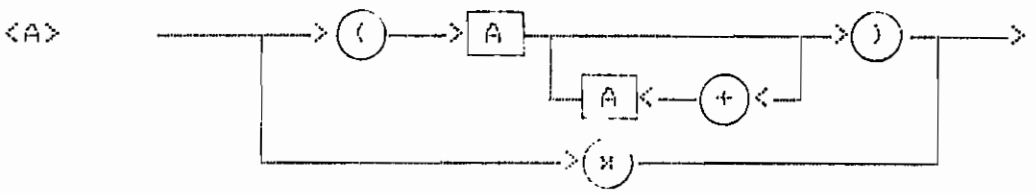


Fig. 3.2.- Grafo unificado.

Aplicando las siete reglas para traducir grafos a programas, al programa unificado, sse obtiene lo siguiente:

```
PROGRAM ANALIZAR (INPUT, OUTPUT);
     VAR  CH : CHAR;
     PROCEDURE A ;
          BEGIN
          IF  CH = 'x'   THEN READ (CH)  ELSE
          IF  CH = '('   THEN
               BEGIN
               READ (CH) ;
               A ;
               WHILE  CH = '+'  DO
                    BEGIN  READ (CH); A  END
```

OPERADOR	SIMBOLO	SIGNIFICADO	JERARQUIA
	**, ↑	exponenciación	1
	*	multiplicación	2
aritmético	/	división	2
	+	suma	3
	-	resta	3
	<, .LT.	menor que	4
	<=, .LE.	menor o igual que	4
de	=, .EQ.	igual a	4
relación	<>, .NE.	no igual a	4
	>=, .GE.	mayor o igual que	4
	>, .GT.	mayor que	4
	¬, NOT	negación	5
lógicos	^, AND	conjunción	6
	v, OR	unión	7

Tabla 3.1.- Operadores más comunes en un lenguaje de programación.

Asignación.- La asignación está estrechamente relacionada con la evaluación de expresiones y es una de las operaciones más importantes en computación. A continuación se da la representación de la asignación en algunos lenguajes.

LENGUAJE	SIMBOLO DE ASIGNACION
ALGOL	A := B
APL	A <- B
BASIC	LET A = B
COBOL	MOVE B TO A
FORTRAN	A = B

Tabla 3.2.- Símbolos de asignación.

Realizar la asignación significa poner el valor de B en la localidad denotada por A. Aquí A representa el nombre de la variable y B el valor de la variable A. Al aplicar la operación de asignación a expresiones se forman las proposiciones de asignación. Las proposiciones o sentencias en general son la base de la escritura de programas.

Estructuras de datos.-

Una estructura de datos es una colección organizada de los mismos. Se pueden distinguir dos tipos: estructuras fundamentales y estructuras avanzadas.

"Las variables de una estructura fundamental cambian de valor, pero nunca su estructura ni el conjunto de valores que pueden tomar. Como consecuencia, el tamaño del espacio que ocupan permanece constante. Las estructuras avanzadas sin embargo, se caracterizan por cambiar de valor y estructura durante la ejecución de un programa. Se necesitan, por ello, técnicas más sofisticadas para su implantación." (1)

De entre las estructuras fundamentales, la más difundida es sin lugar a dudas, el arreglo:

"Un arreglo es una estructura homogénea; está constituída por componentes, todos ellos del mismo tipo, llamado tipo base. El arreglo se denomina también estructura de acceso aleatorio, todos sus componentes pueden seleccionarse arbitrariamente y son igualmente accesibles." (2)

Las características principales de un arreglo son:

- el tipo de elementos que contiene,
- su dimensión o número de subíndices, y
- el número de elementos en cada dimensión.

---

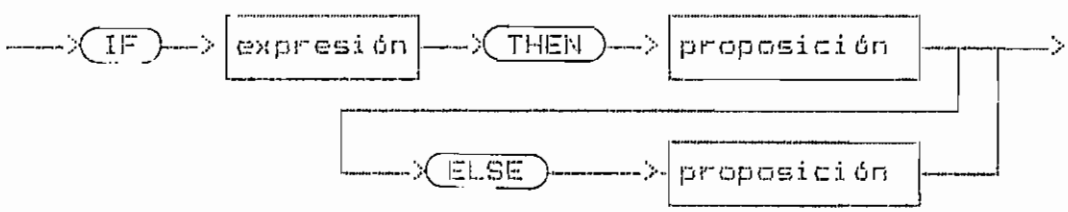
(1) WIRTH, Niklaus; Algoritmos + Estructuras de Datos = Programas, Univ. Politécnica de Madrid, Madrid, 3ra impresión 1984, p. xiii.

(2) Idem (1), p. 12.

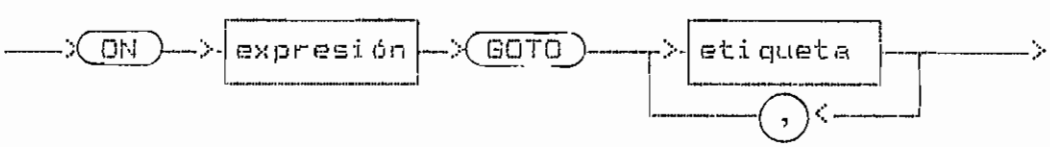
3.2.3.- La decisión.

Al escribir un programa, muchas veces es necesario especificar dos o más formas de acción, y permitir que el proceso que está ejecutando el programa seleccione una de ellas durante la ejecución. La estructura que permite esta acción es la decisión. La decisión es una estructura que hace posible que la ejecución del programa sea influida por los datos. La sintaxis de las proposiciones más comunes para implementar la decisión son:

a) proposición IF:



b) proposición ON-GOTO:



c) proposición CASE:

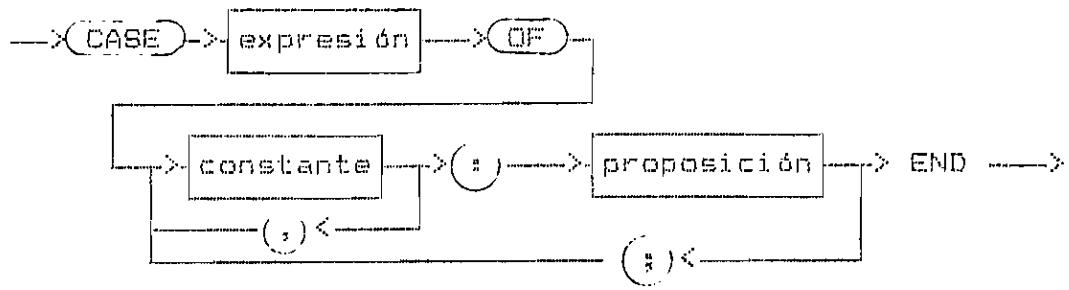


Fig.3.4.- Sintaxis de las proposiciones IF, ON-GOTO y CASE

3.2.4.- La estructura repetitiva a ciclo.

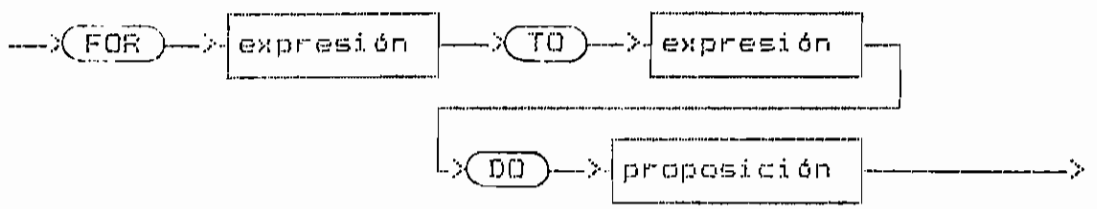
"La estructura repetitiva o ciclo se usa para ejecutar una instrucción o una secuencia de instrucciones varias veces. Aunque las instrucciones son las mismas cada vez que se efectúa el ciclo, los datos sobre los cuales opera este, no lo son. " (1)

La sintaxis de las proposiciones más usadas para implementar los ciclos, son:

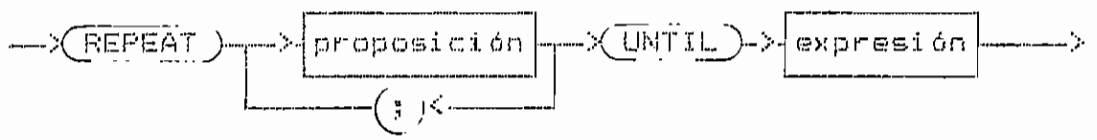
---

(1) GROGONO, Peter; Programación en Pascal, Fondo Educativo Interamericano, México 1984, p.4.

a) proposición FOR



b) proposición REPEAT



c) proposición WHILE

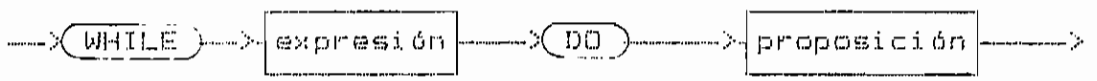


Fig. 3.4.- Sintaxis de las proposiciones FOR, REPEAT y WHILE.

3.2.5.- La rutina.

"La rutina hace posible que podamos reemplazar a un grupo de instrucciones con una sola instrucción. El uso de rutinas en la programación de computadores no solo hace los programas más cortos y más fáciles de leer y escribir,

C1	C$
X5	T$

Ejemplo 3.3.- Identificadores de variables numéricas y string.

Operadores.- En el siguiente cuadro se indican los operadores del BASIC en orden jerárquico.

OPERADOR	SIMBOLO
exponenciac.	↑
aritmético	*, /
	+, -
de relación	<, <=, =, <>, >=, >

Tabla 3.3.- Operadores BASIC en orden jerárquico.

Estructuras de datos.- El BASIC posee la estructura de datos que hemos denominado "arreglo" (estructura fundamental). Estos arreglos de hasta dos dimensiones, y sus datos pueden ser de tipo numérico o string. Un identificador de arreglo numérico se forma de una letra, uno de tipo char con una letra y el signo dólar.

Para referirse a un elemento de un arreglo se la hace

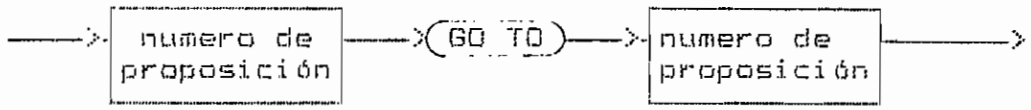
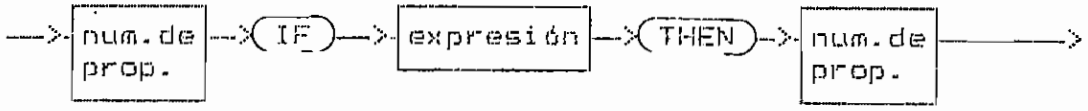


Fig.3.5.- Sintaxis de la proposición GO TO en BASIC.

3.3.3.- La decisión.

La decisión se implementa con las proposiciones IF y con la ON-GOTO. Su sintaxis es:

* proposición IF :



* proposición ON-GOTO :

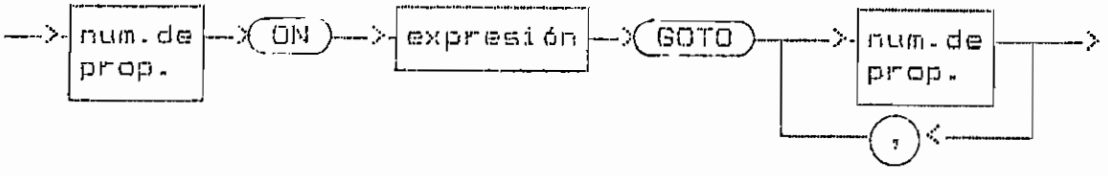


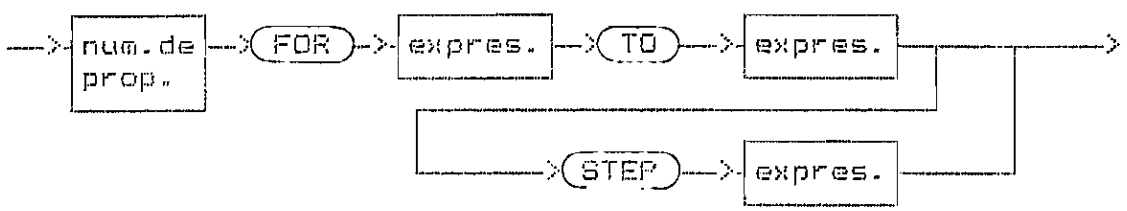
Fig.3.6.- Sintaxis de las proposiciones IF y ON-GOTO en BASIC.

3.3.4.- El ciclo.

El ciclo se forma con la proposición FOR, que tiene a siguiente sintaxis :

* proposición FOR :

* proposición FOR-TO :



* proposición NEXT :

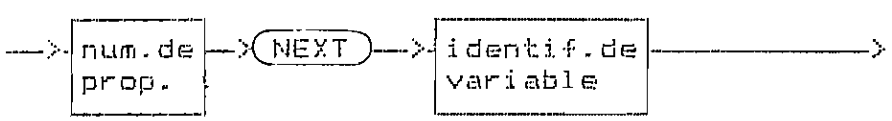
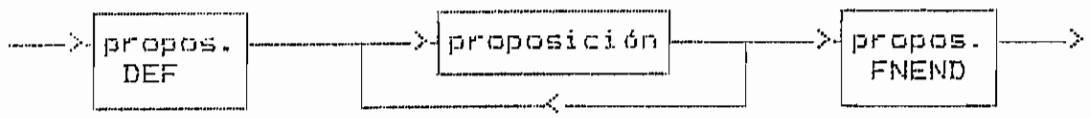


Fig.3.7.- Sintaxis de las proposición FOR en BASIC.

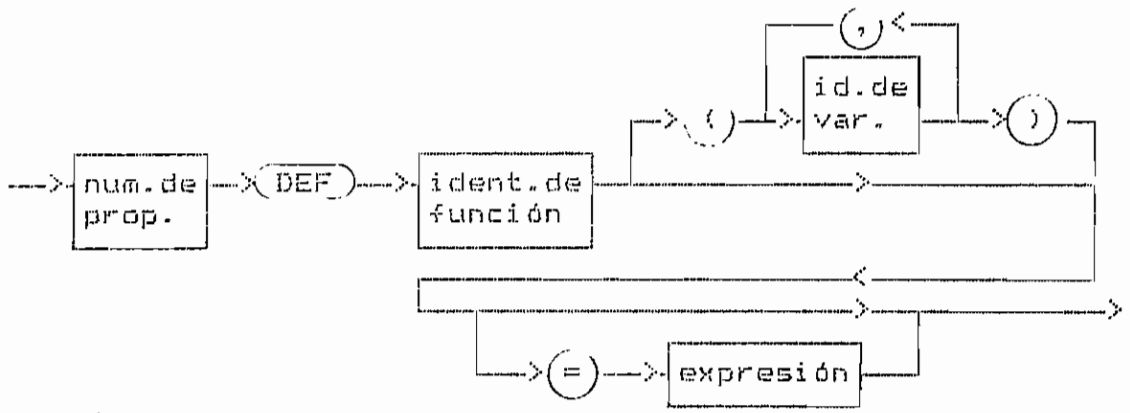
3.3.5.- La rutina.

En BASIC se definen funciones y subrutinas, como rutinas. Su sintaxis es la siguiente:

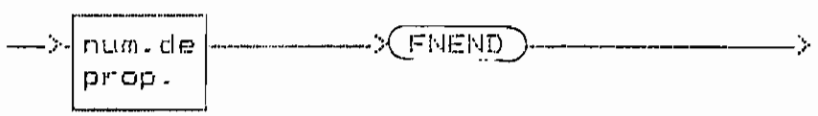
a) funciones:



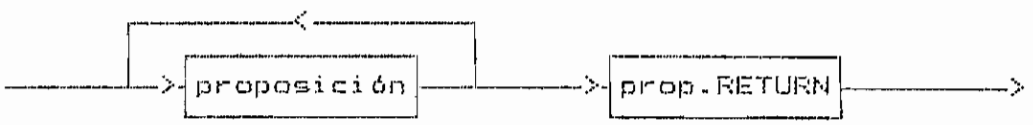
* proposición DEF :



* proposición FNEND :



a) subrutinas :



3.4 ALCANCE DEL LENGUAJE BASIC PARA EL COMPILADOR A IMPLEMENTARSE .

En este punto se explicará la estructura misma del lenguaje fuente que servirá para escribir los programas fuentes en el presente compilador. Las características principales de este lenguaje son:

a) es un lenguaje que tiene sus raíces en el lenguaje BASIC. Está basado en las características del lenguaje BASIC que se describe en el texto Programación BASIC del autor Byron S. Gottfried tratado en el punto anterior,

b) se le ha implementado una estructura especialmente en cuanto se refiere al orden en que se deberán escribir las proposiciones dentro de un programa, y en las proposiciones de decisión y ciclos, y

c) se ha incrementado proposiciones tales como la definición explícita de variables a usarse en el programa.

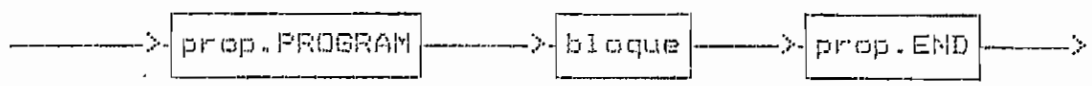
Siendo este un lenguaje que tiene sus deferencias respecto al tratado en el punto anterior (punto 3.3), en lo posterior se hablará de la Versión BASIC para referirnos al lenguaje fuente que usará el presente compilador.

3.4.1.- Estructura de un programa en VERSION BASIC.

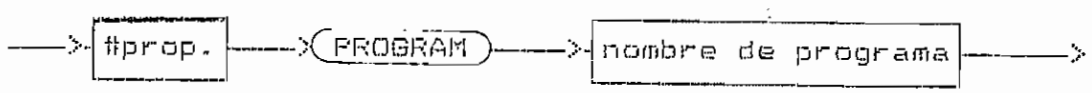
Un programa en VERSION BASIC será una serie de proposiciones colocadas en el orden en que se desea que se ejecuten, a menos que se indique explícitamente una bifurcación. Esto significa que la estructura secuencial (la secuencia) estará definida implícitamente.

La primera proposición del programa será la proposición PROGRAM, y la última la proposición END. Entre ellas se escribiran todas las proposiciones necesarias para completar el programa. A este último grupo de proposiciones se lo llamará bloque. Un comentario se lo puede escribir con la proposición REM y podrá incluírselo en cualquier lugar del programa. A continuación se muestra la sintaxis de un programa en VERSION BASIC.

programa



próp: PROGRAM



prop. END

bloque

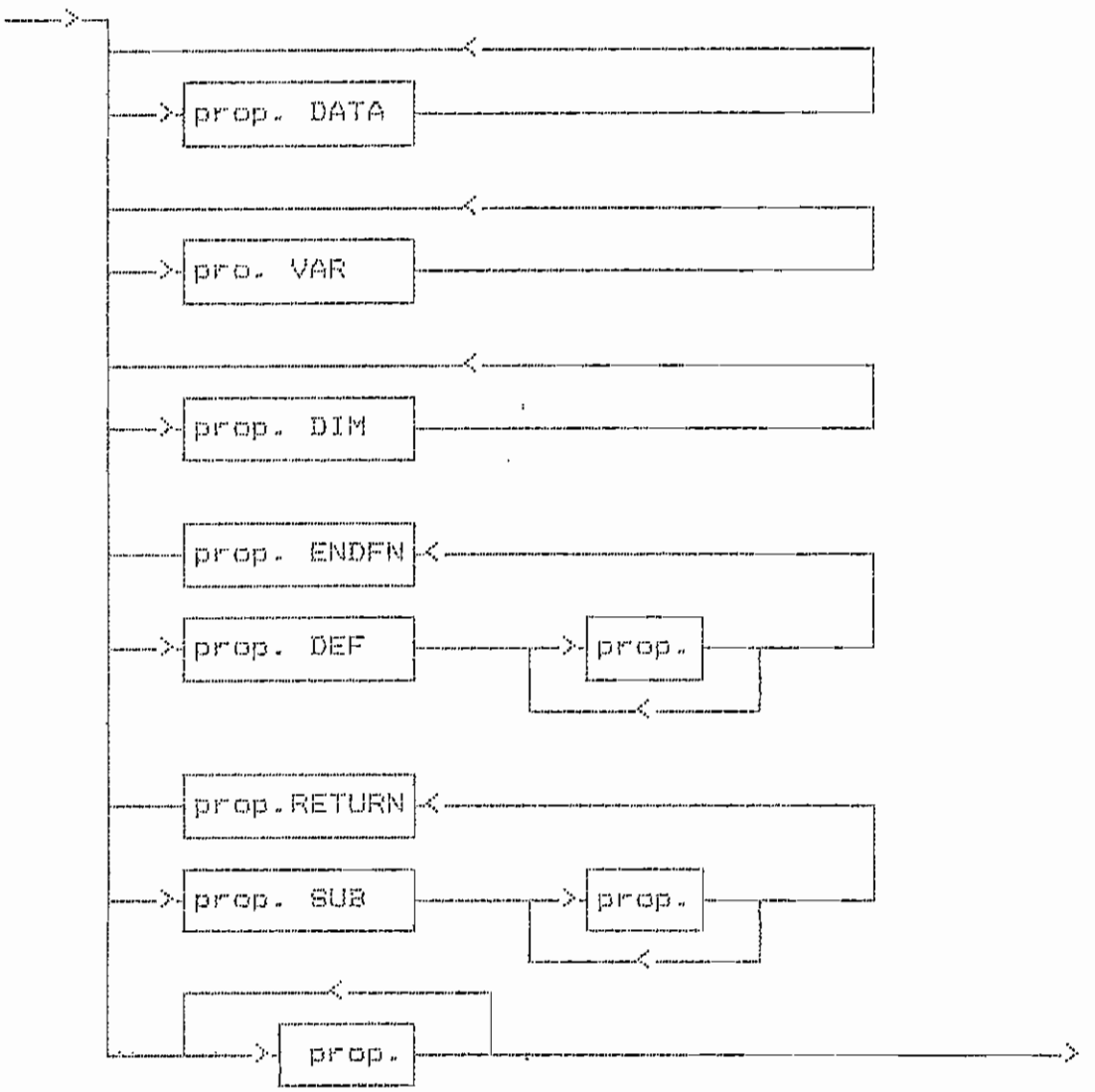
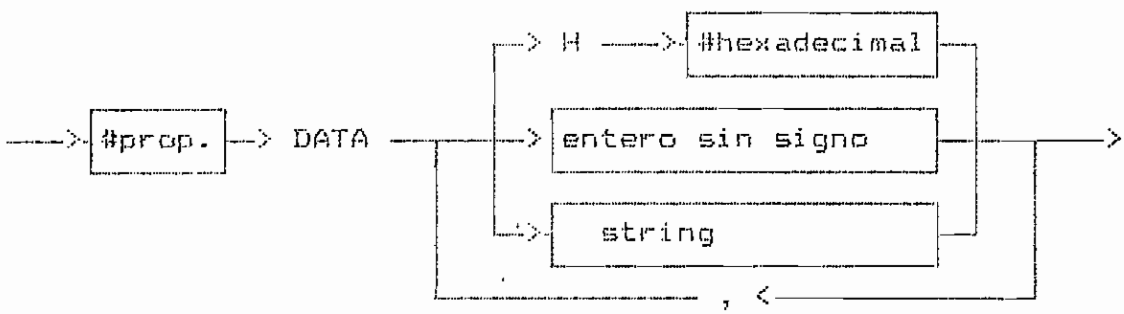


Fig. 3.9.- Estructura de un programa en VERSION BASIC

3.4.3.- Datos.

Se puede ingresar datos por medio de la proposición DATA. Los datos pueden ser numéricos o strigs (tiras de caracteres). Los datos numéricos pueden ser decimales enteros sin signo hasta de 9 dígitos o hexadecimales de 16 bits a los que se les antepone la letra H. Los datos pueden ser leídos por medio de la porposición READ. La sintaxis de las proposiciones DATA y READ, es la siguiente:

prop. DATA



prop. READ

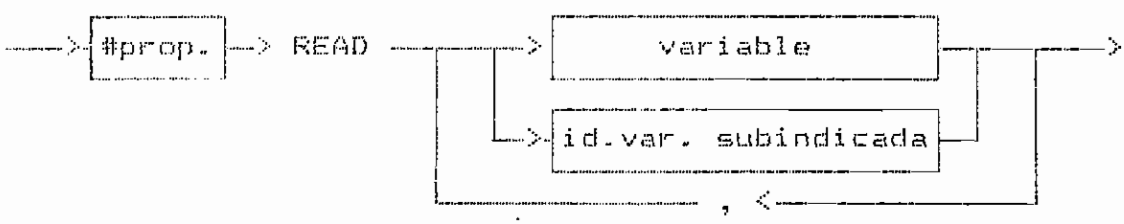
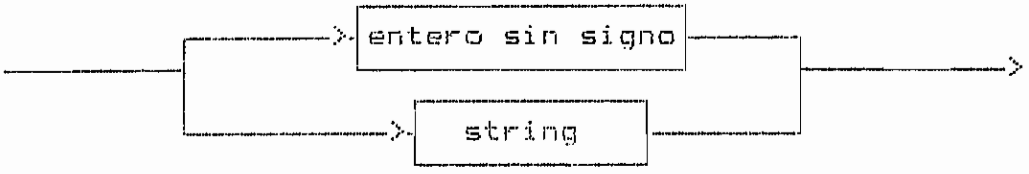


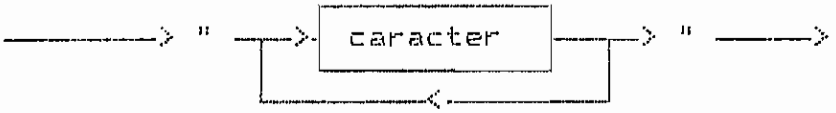
Fig. 3.10.- Sintaxis de las proposiciones DATA y READ

a) Constantes y variables.- Las constantes pueden ser números enteros (con o sin signo) o strings (tiras de caracteres) que iran encerrados entre comillas. Los identificadores de variables simples pueden ser numéricas o strings. A continuación se da la sintaxis de constantes y variables.

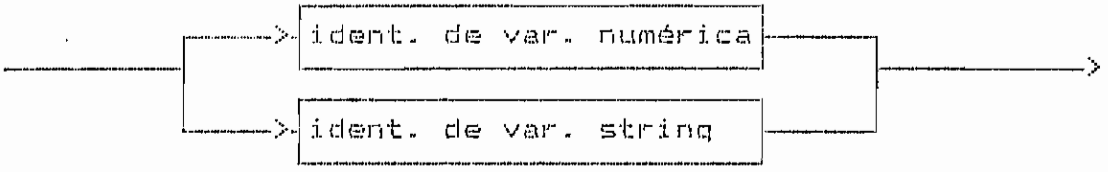
constante



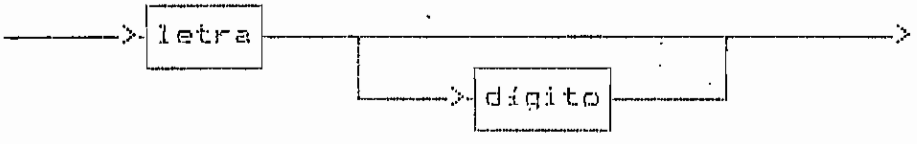
string



variable



id. de var. numérica



id. de var. string

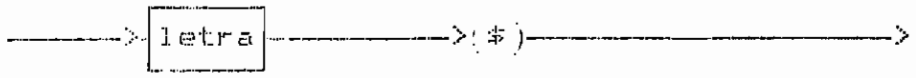


Fig. 3.11.- Sintaxis de constantes y variables.

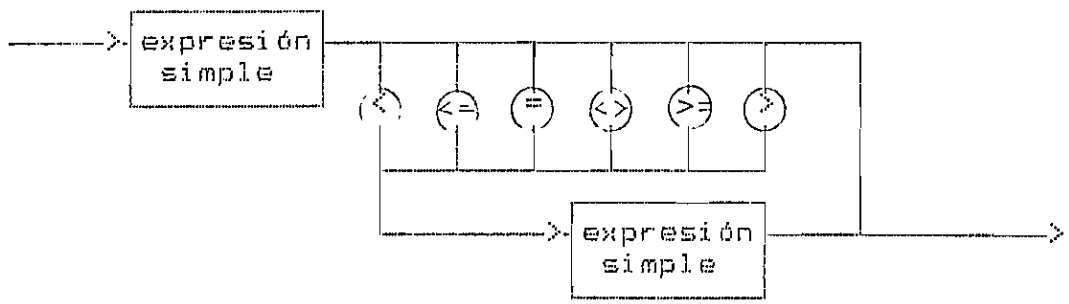
b) Operadores.- Los símbolos utilizados para los operadores, su significado y su jerarquía, se indican en la tabla 3.4.

operador	símbolo	significado	jerarquía
aritméticos	*, /, |	multiplicación, división, concatenación.	1
	+, -	suma, resta	2
de relación	<	menor que	3
	<=	menor o igual que	3
	=	igual que	3
	<>	no igual que	3
	>=	mayor o igual que	3
	>	mayor que	3
lógicos	NOT	negación	4
	AND	conjunción	5
	OR	unión	6

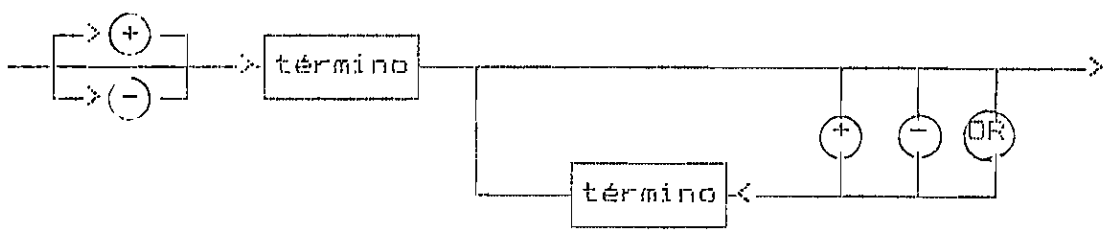
Tabla 3.4.- Operadores de la VERSION BASIC.

con lo que una expresión seguirá la siguiente sintaxis:

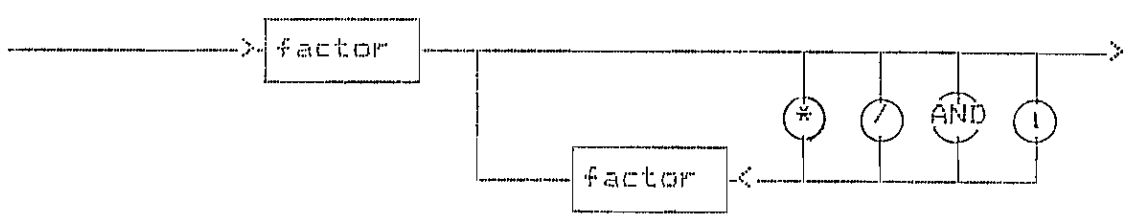
expresión



expresión simple



término



factor

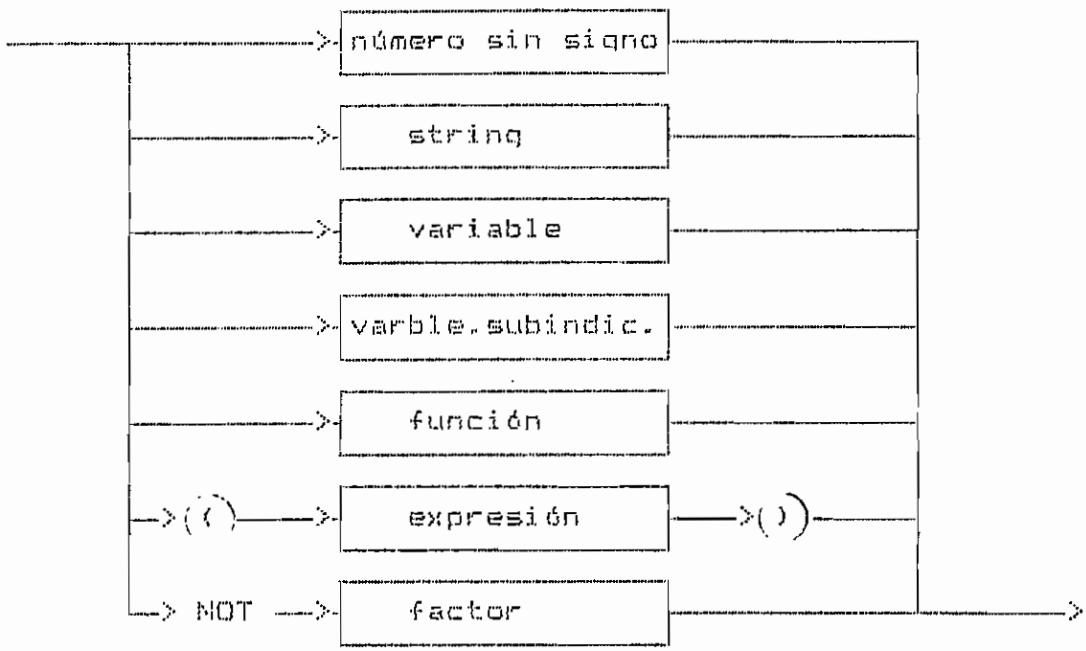
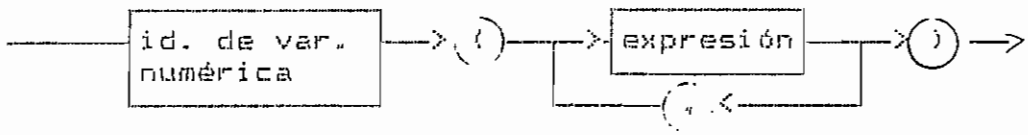


Fig. 3.12.- Sintaxis de una expresión en VERSION BASIC.

c) Estructuras de datos.- Las únicas estructuras de datos que se tendrá serán arreglos de hasta dos dimensiones. Estos arreglos pueden contener sólo datos numéricos. Una llamada a un elemento de un arreglo se la hace por medio de variables subindicadas. Los subíndices podrán ser números enteros o expresiones que tengan como resultado un número entero positivo.

Los arreglos deben declararse explícitamente con la proposición DIM. La sintaxis de una variable subindicada y de la oproposición DIM, se dan a continuación.

variable subindicada



prop. DIM

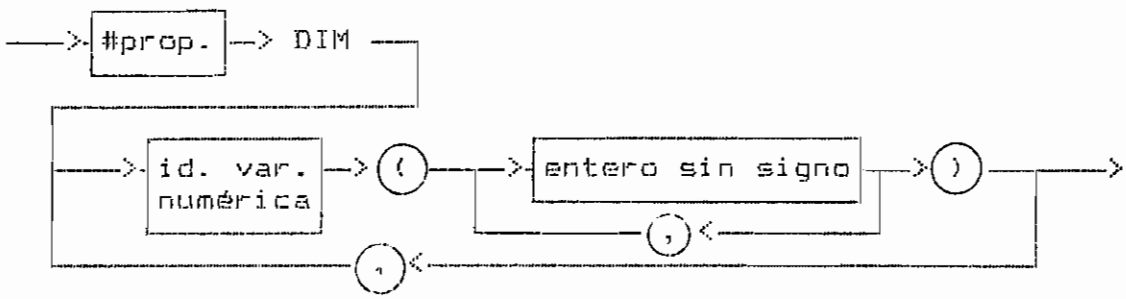


Fig. 3.13.- Sintaxis de variable subindicada y de proposición DIM, en VERSION BASIC.

3.4.4.- La transferencia incondicional de control.

La transferencia incondicional de control se la hace con la proposición GOTO que tiene la siguiente sintaxis:

prop. GOTO

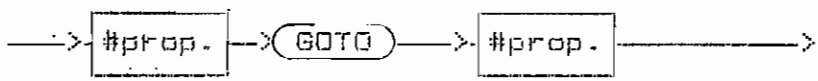
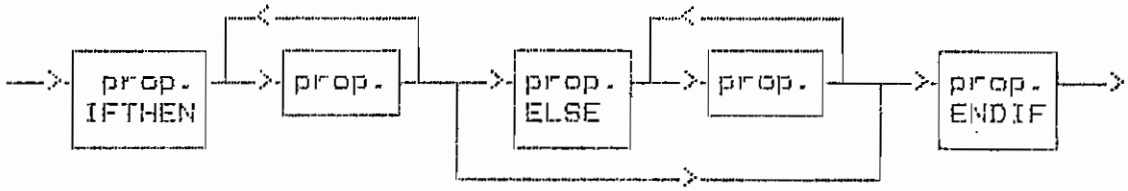


Fig. 3.14.- Sintaxis de prop. GOTO en VERSION BASIC.

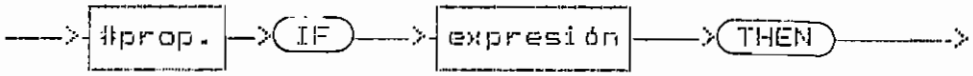
3.4.5.- La decisión.

La decisión se la construye con las proposiciones IF y CASE, con la siguiente sintaxis:

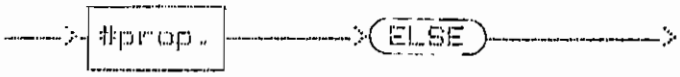
prop. IF



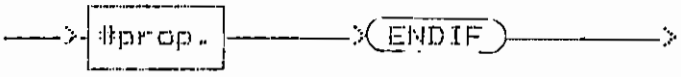
prop. IF-THEN



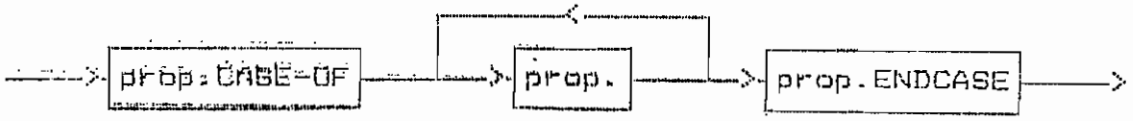
prop. ELSE



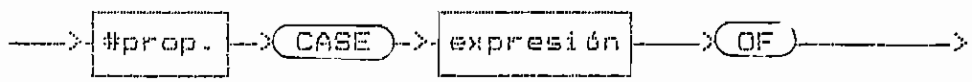
prop. ENDIF



prop. CASE



prop. CASE-OF



prop. ENDCASE

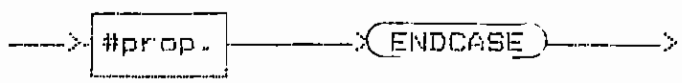


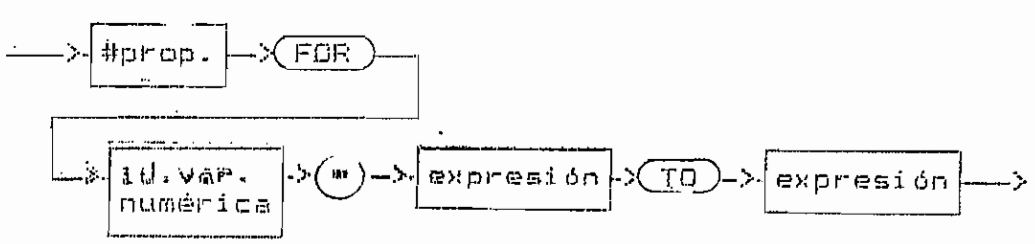
Fig. 3.15.- Sintaxis de la prop. IF y la prop. CASE en VERSION BASIC.

3.4.6.- El ciclo.

Se construye ya sea con la proposición FOR o con la proposición WHILE, y la siguiente sintaxis:

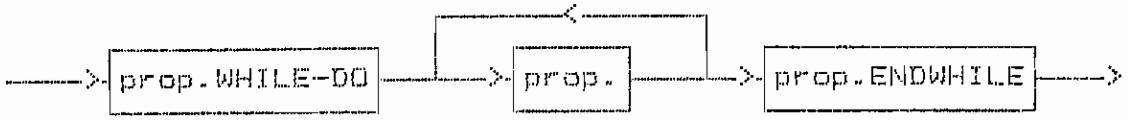
prop. FOR

prop. FOR-TO

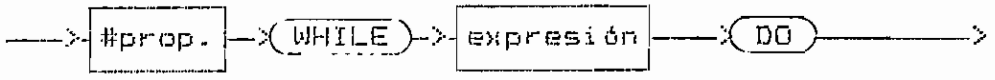


prop. NEXT

prop. WHILE



prop. WHILE-DO



prop. ENDWHILE

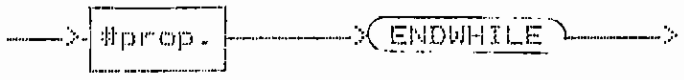
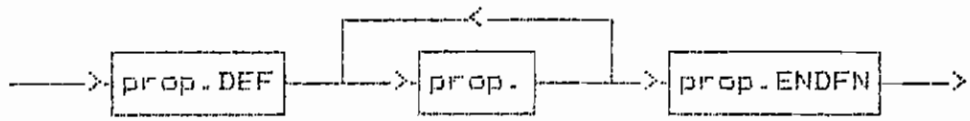


Fig. 3.16.- Sintaxis de la prop. FOR y la prop. WHILE en VERSION BASIC.

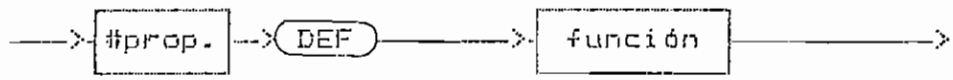
3.4.7.- La rutina.

Se pueden definir funciones y subrutinas. La definición de cada una de estas tiene la siguiente sintaxis:

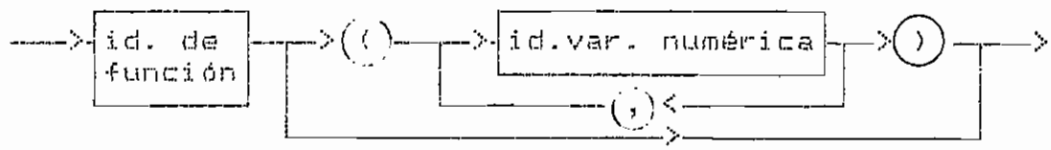
definición de funciones:



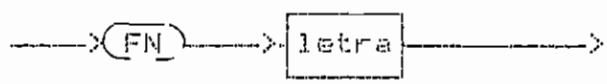
prop. DEF



función

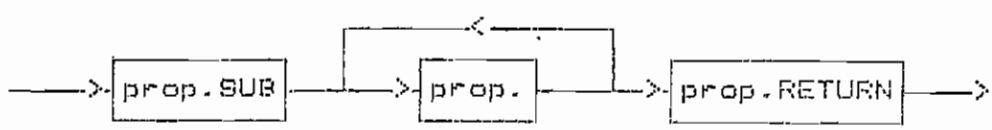


id. de función



prop. ENDFN

definición de subrutinas



3.4.8.- La asignación.

La asignación se la realiza con la proposición LET y la siguiente sintaxis:

prop. LET

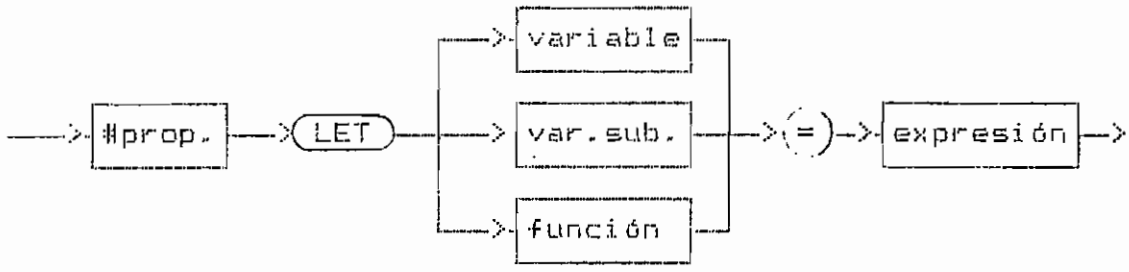


Fig. 3.19.- Sintaxis de la prop. LET en VERSION BASIC.

Además la VERSION BASIC tiene las proposiciones REM y RESTORE. La proposición REM permite escribir una línea de comentarios en cualquier lugar del programa. La proposición RESTORE permite reinicializar el contador de datos para que una proposición READ, posterior a la RESTORE, lea y asigne los datos encontrados en DATA a los identificadores encontrados en READ, empezando desde el primer valor en la proposición DATA. La sintaxis de estas dos proposiciones es la siguiente:

4.1 DIAGRAMA DE BLOQUES DEL COMPILADOR. DESCRIPCION.

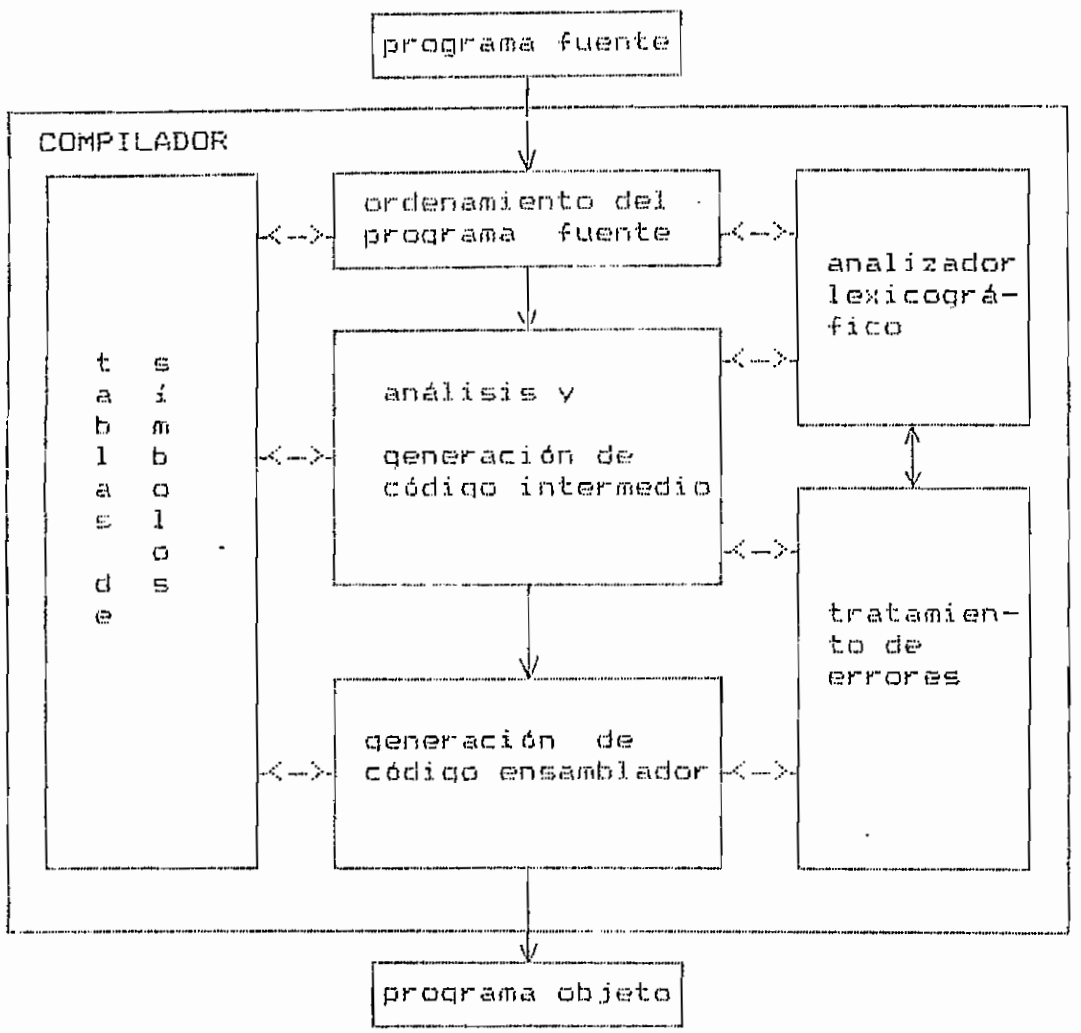


Fig. 4.2.- Diagrama de bloques del compilador implementado.

pos TEXT son sus respectivas compilaciones, utilizando los terminales del centro de cómputo. Además de este grupo de archivos se debe anotar que se tendrá llos archivos FILE FUENTE (para editar el programa fuente), FILE INPUT (almacena el programa reordenado), y FILE SALIDA (almacena los resultados). Información más detallada se da en el MANUAL DEL USUARIO.

4.3,4,5. DESRROLLO, DIAGRAMAS DE FLUJO Y DESCRIPCION DE PARAMETROS Y VARIABLES DEL PROGRAMA PRINCIPAL Y LAS RUTINAS.

4.3.1.- Programa COMPILADOR.

Es el programa principal. Declara e inicializa variables globales, llama a las rutinas para realizar la compilación y almacena en el archivo FILE SALIDA, los resultados.

constantes:

NMXID = número máximo de identificadores.

NMXDT = número máximo de datos.

NMXCD = número máximo de instrucciones en código intermedio.

NMXPR = número máximo de parámetros en una función.

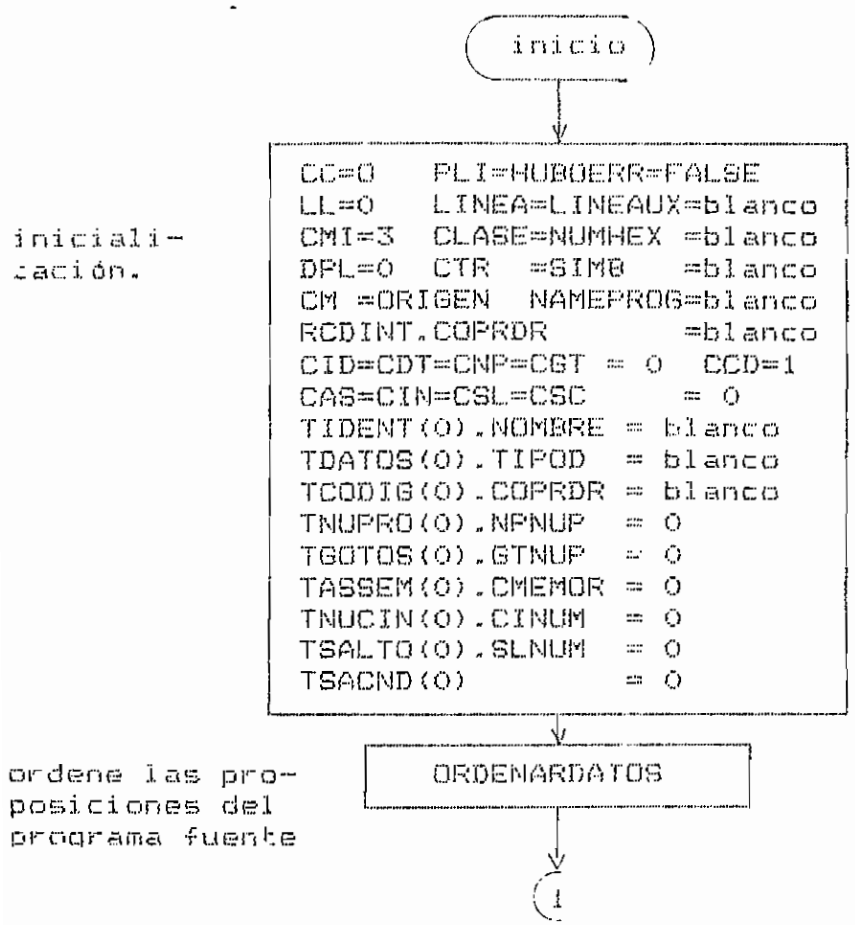
NMXNP = número máximo de proposiciones en el programa fuente.

CGT, TGOTOS = contador y tabla con información sobre saltos incondicionales del programa fuente.

CSL, TSALTO = contador y tabla con información sobre instrucciones que tienen saltos incondicionales en código intermedio.

CAS, TASSEM = contador y tabla de instrucciones en código ensamblador.

CSC, TSACND = contador y tabla con información sobre instrucciones que tienen saltos condicionales en código intermedio.



inicialización.

ordene las proposiciones del programa fuente

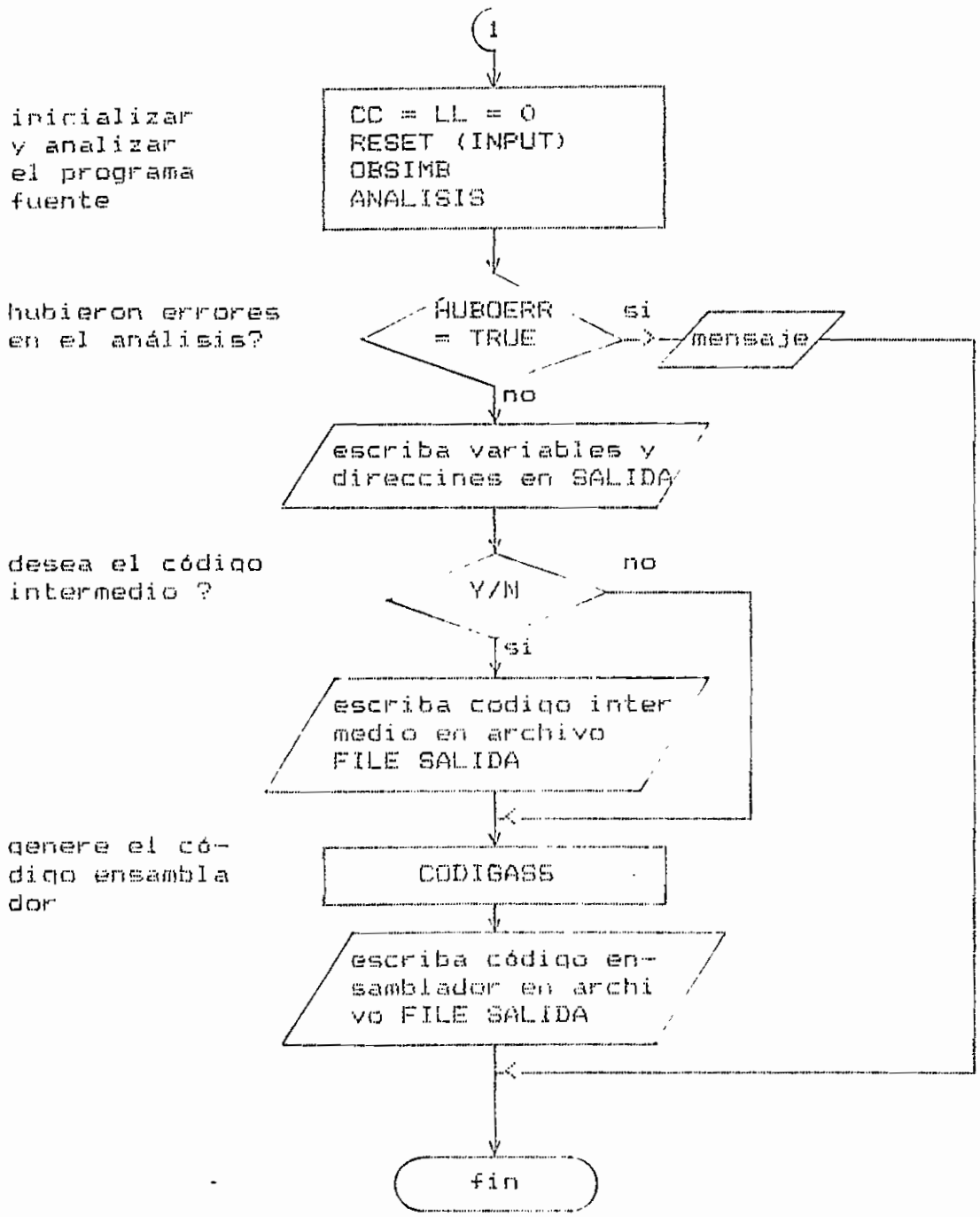


Fig. 4.3.- Diagrama de flujo del programa COMPILADOR.

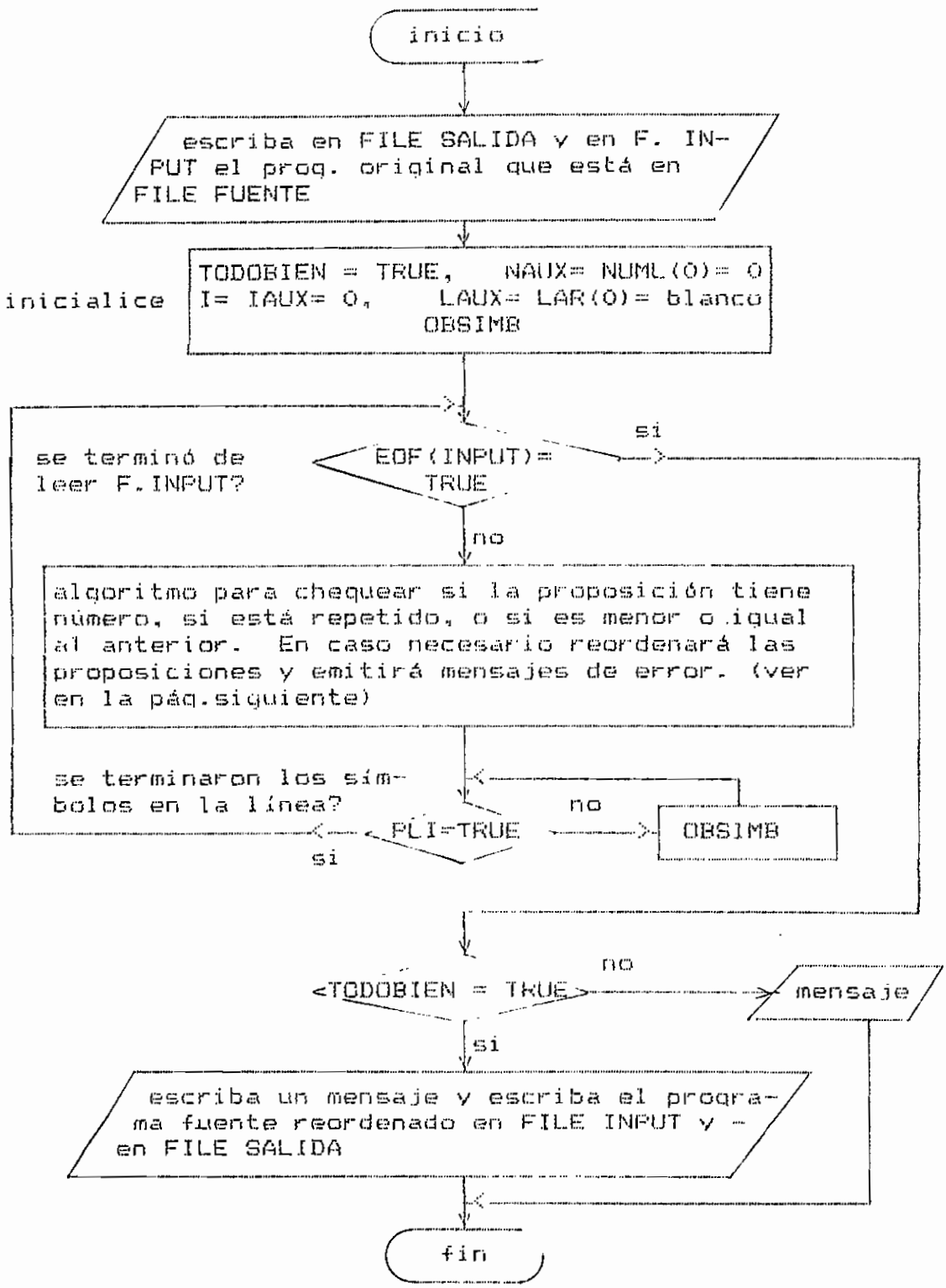


Fig. 4.4.- Diagrama de flujo de la rutina ORDENARDATOS.

Algoritmo para chequear si la proposición tiene número, si es²a repetido o si es menor o igual al anterior. En caso necesario reordenará las proposiciones y emitirá mensajes de error.

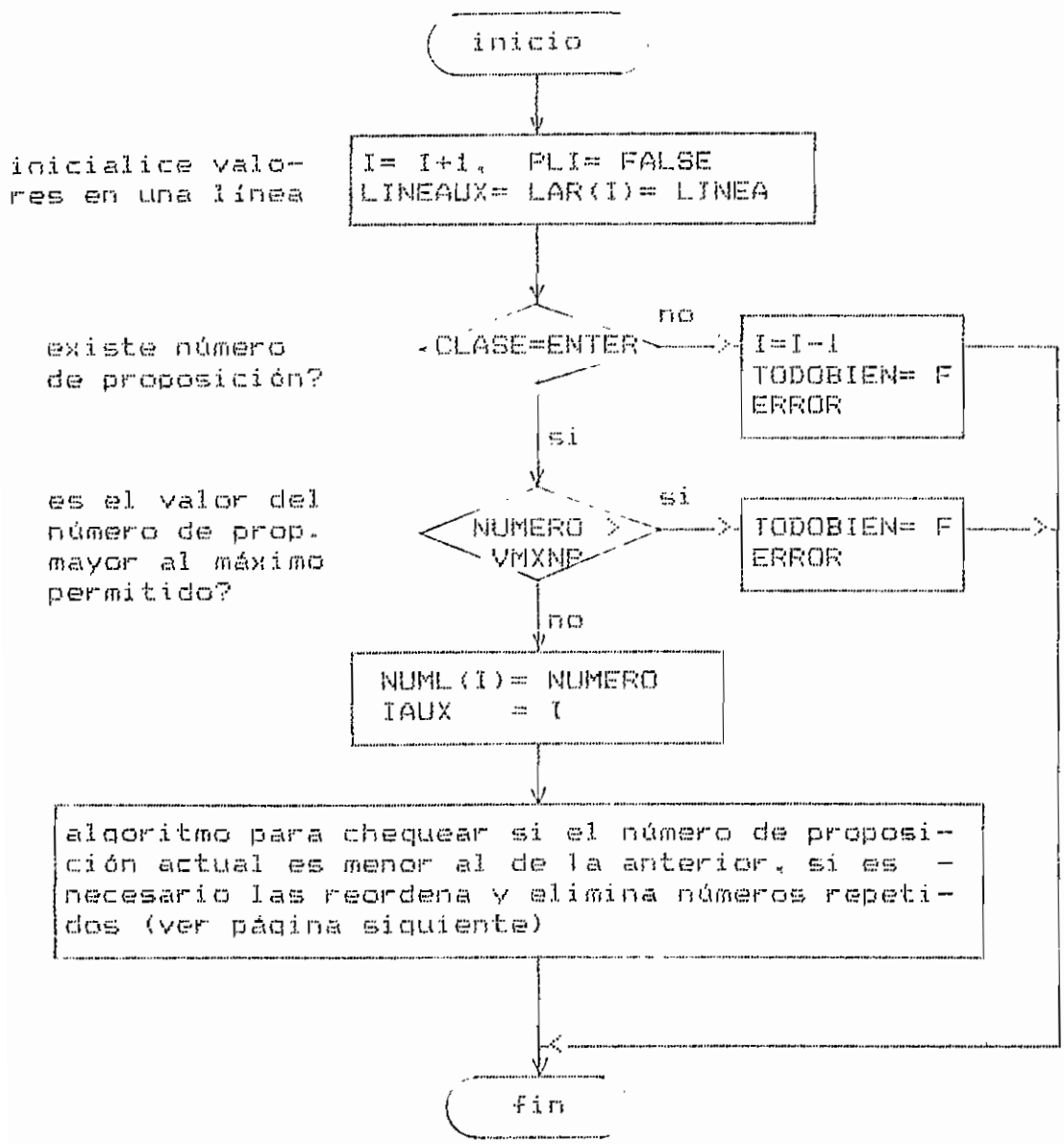


Fig. 4.4.a.- Algoritmo indicado en la página anterior.

Algotimo que chequea si el número de proposición actual es menor al de la anterior, en caso necesario las reordena y elimina números repetidos.

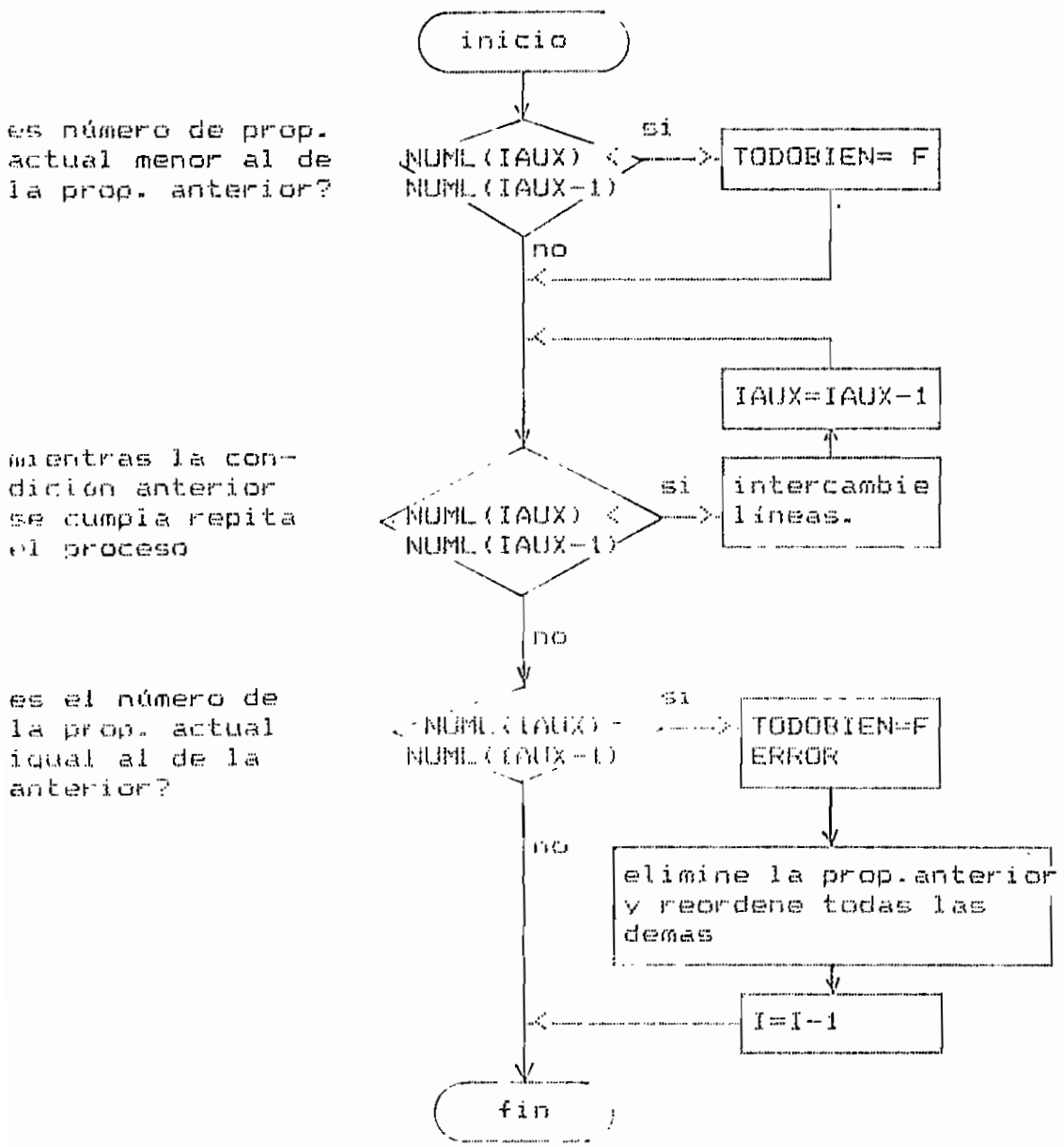


Fig. 4.4.b.- Algoritmo indicado en la página anterior.

4.3.5.- Rutina externa POSICION.

Ubica la posición, por su nombre, de un identificador de variable o de función dentro de la tabla de identificadores.

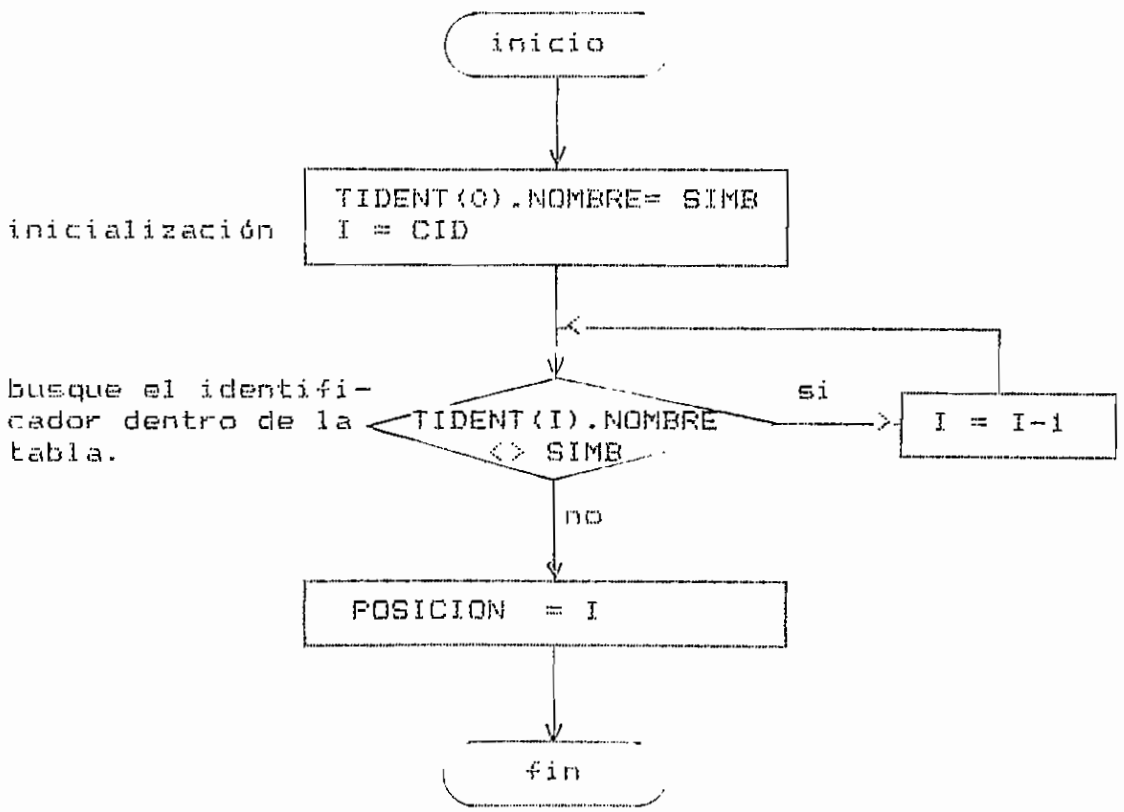


Fig. 4.7.- Diagrama de flujo de la rutina POSICION.

4.3.7.- Rutina externa GENCODIG.

Genera una instrucción en código intermedio y la ingresa a la tabla TCODIG.

parámetros:

GC1= operador en código intermedio.

GC2= tipo de variable (Simple, Arreglo o String).

GC3= tipo de operando decimal(N) o hexadecimal(H).

GC4= operando decimal.

GC5= dirección de un auxiliar que sirve para operar con el registro índice.

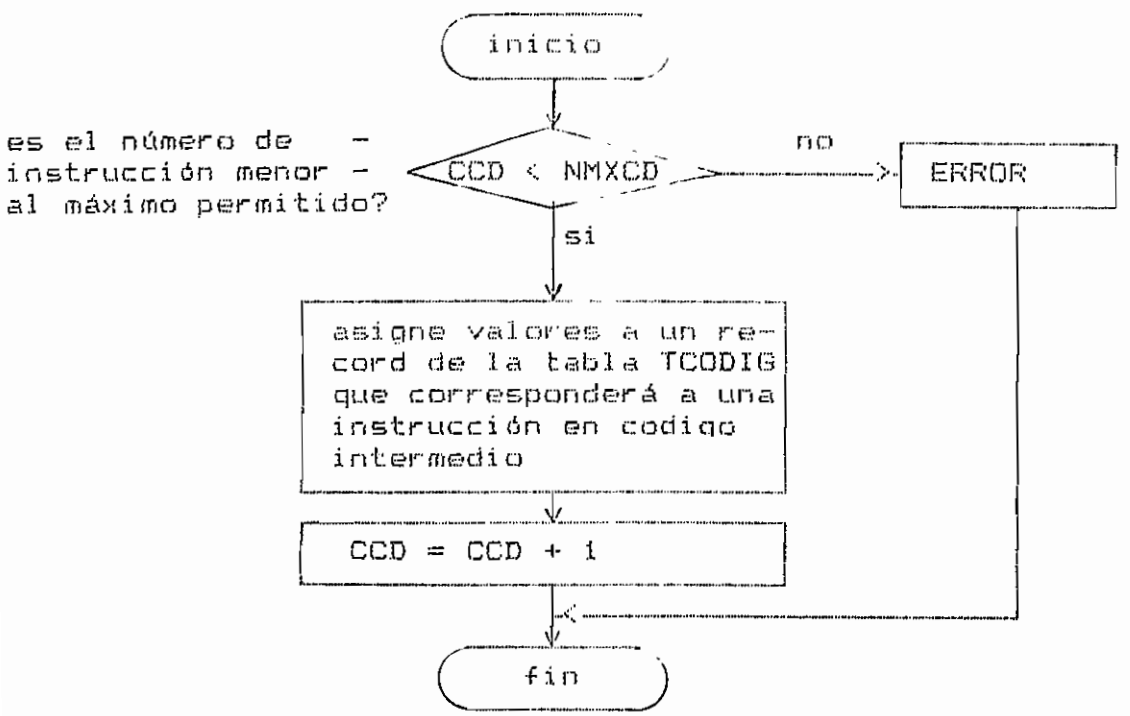


Fig. 4.9.- Diagrama de flujo de la rutina GENCODIG.

4.3.8.1.- Rutina C_STRING (interna a CONSTANTE).

Analiza una constante string.

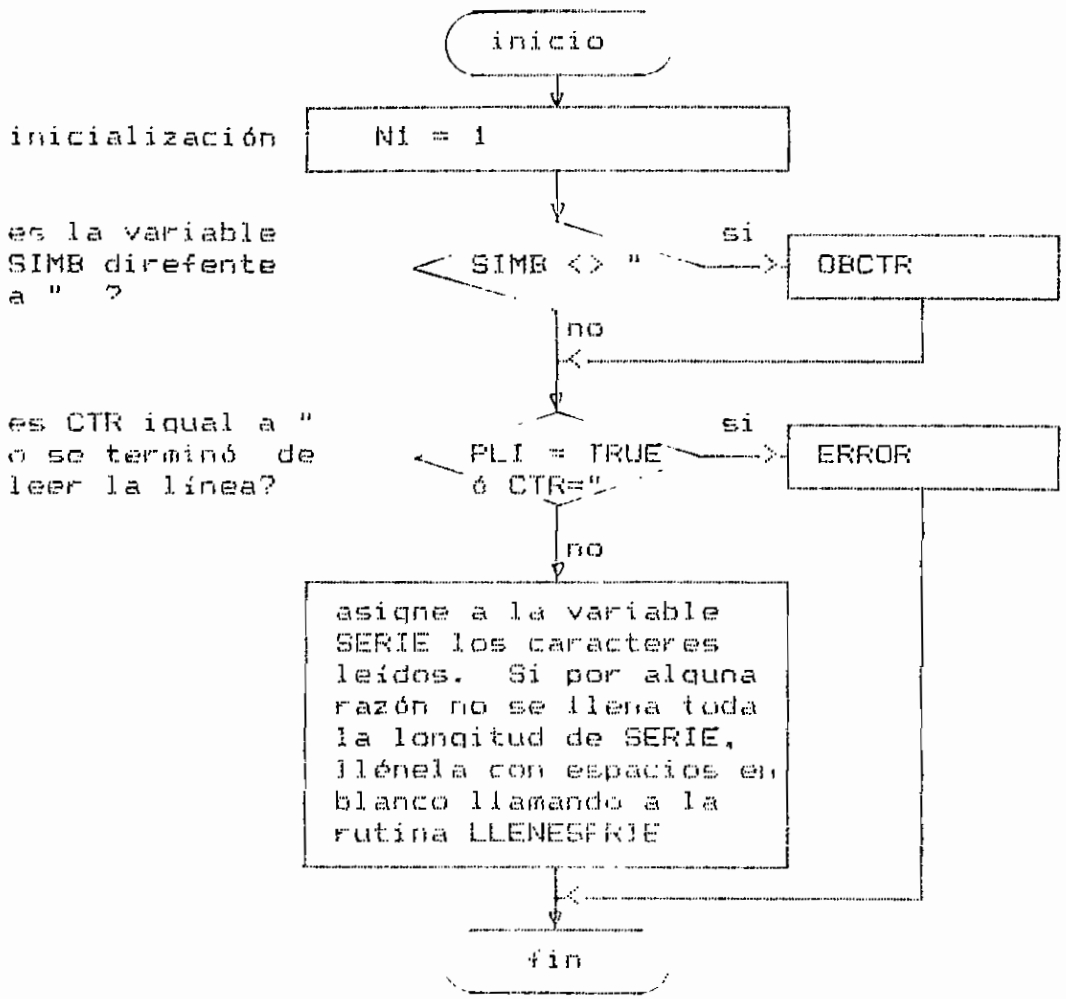


Fig. 4.11.- Diagrama de flujo de la rutina C_STRING.

4.3.8.1.1.- Rutina LLENESERIE (interna a C_STRING).

Asigna a la variable SERIE espacios en blanco hasta que llene su longitud máxima.

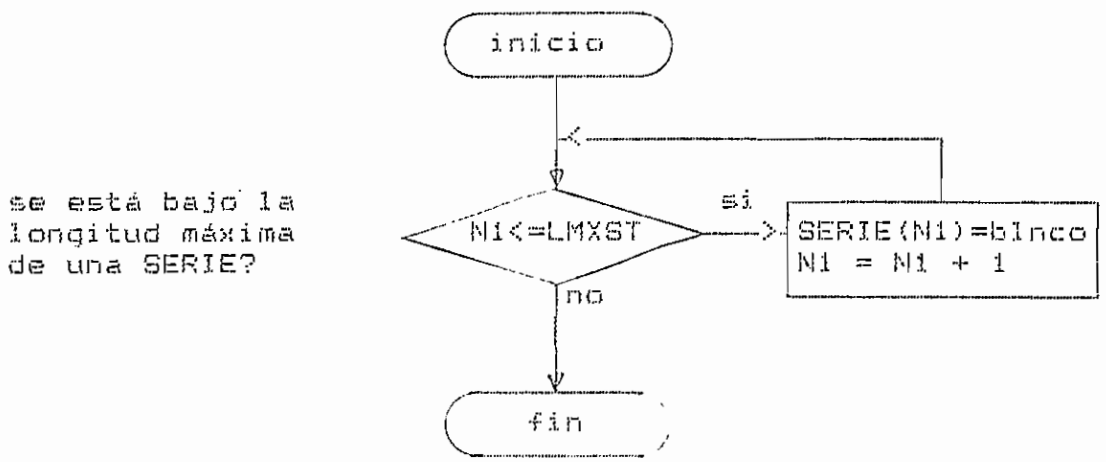


Fig. 4.12.- Diagrama de flujo de la rutina LLENESERIE.

4.3.8.2.- Rutina C_HEXADC (interna a CONSTANTE).

Analiza un número hexadecimal.

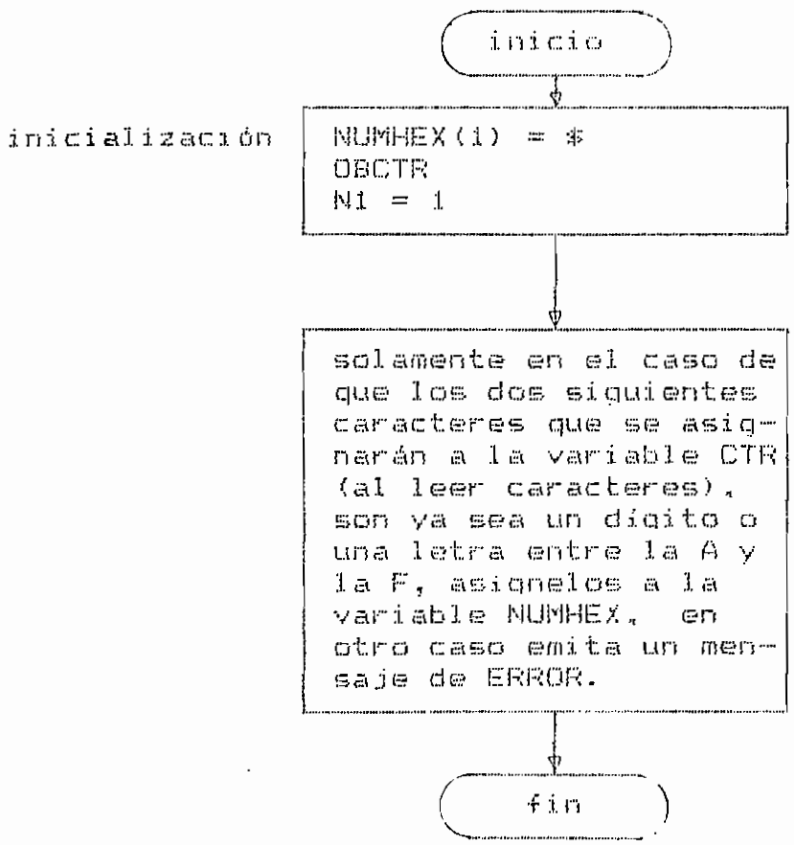


Fig. 4.13.- Diagrama de flujo de la rutina C_HEXADC.

4.3.9.- Rutina externa OBCTR.

Lee líneas del programa fuente desde el archivo FILE INPUT y asigna un caracter leído a la variable CTR.

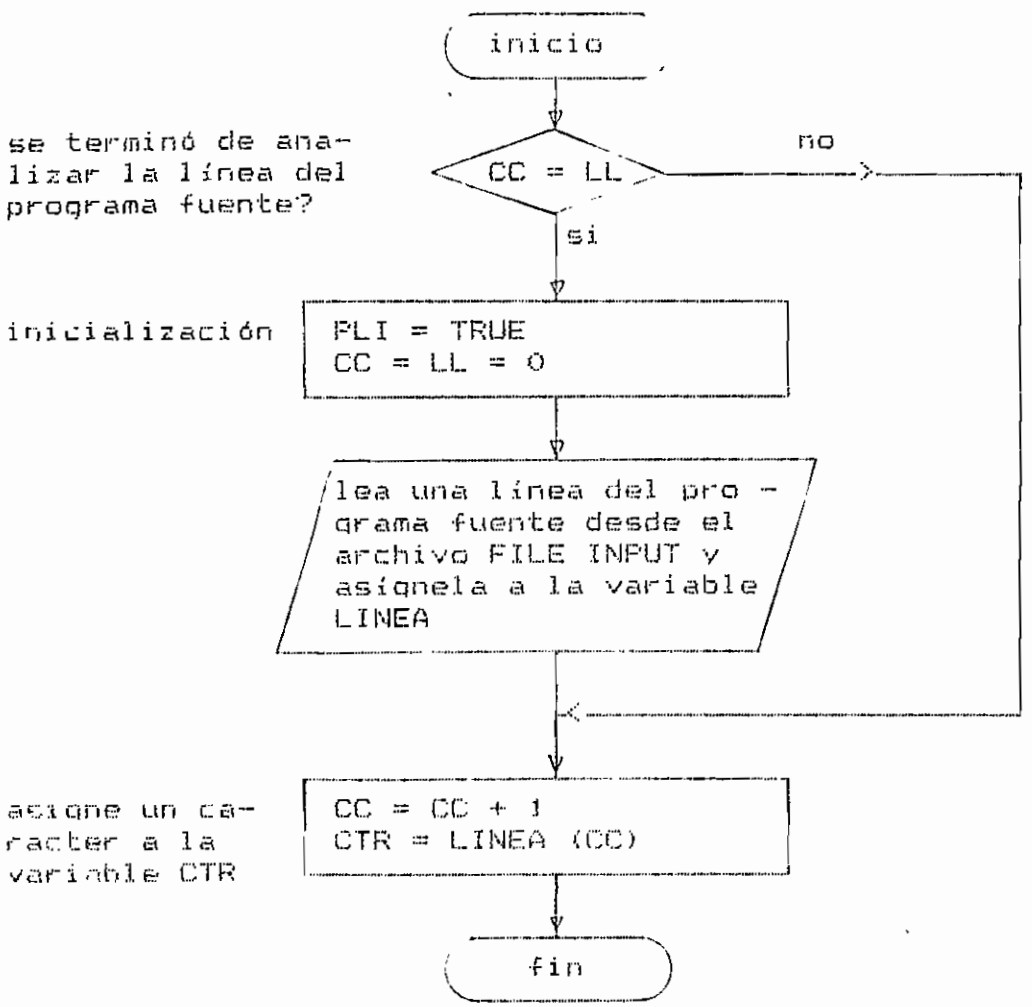


Fig. 4.14.- Diagrama de flujo de la rutina OBCTR.

4.3.10.- Rutina externa OBSIMB.

Lee un símbolo desde el programa fuente (através de OBCTR), asigna su valor a la variable SIMB y su clase a la variable CLASE.

variables locales:

I, J ,K ,FF, N1 = contadores.

FILA = vector donde se construye el símbolo leído.

PALABRA = arreglo donde se almacenan todas las palabras reservadas que tiene el lenguaje fuente.

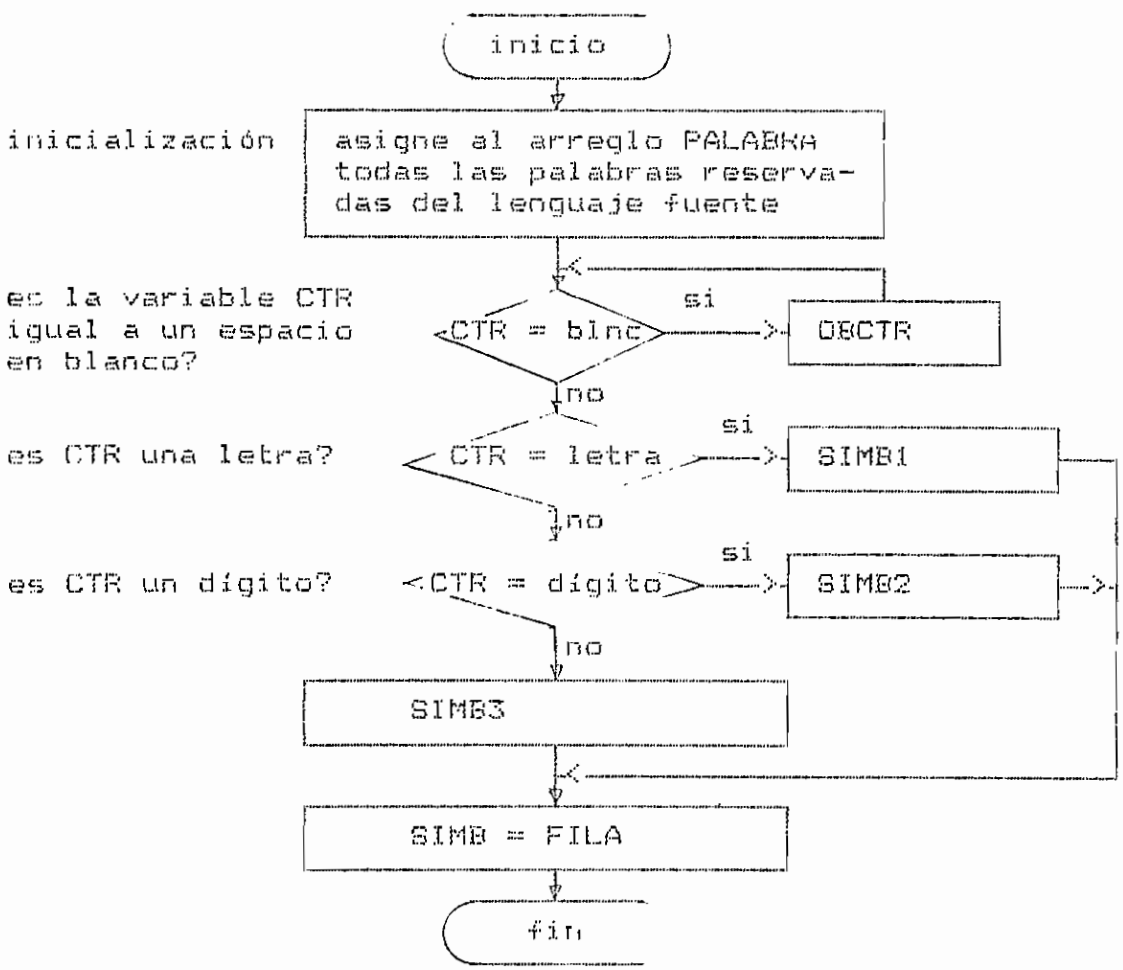


Fig. 4.15.- Diagrama de flujo de la rutina OBSIMB.

4.3.10.1.- Rutina SIMBI (interna a OBSIMB).

Construye palabras reservadas e identificadores de variables.

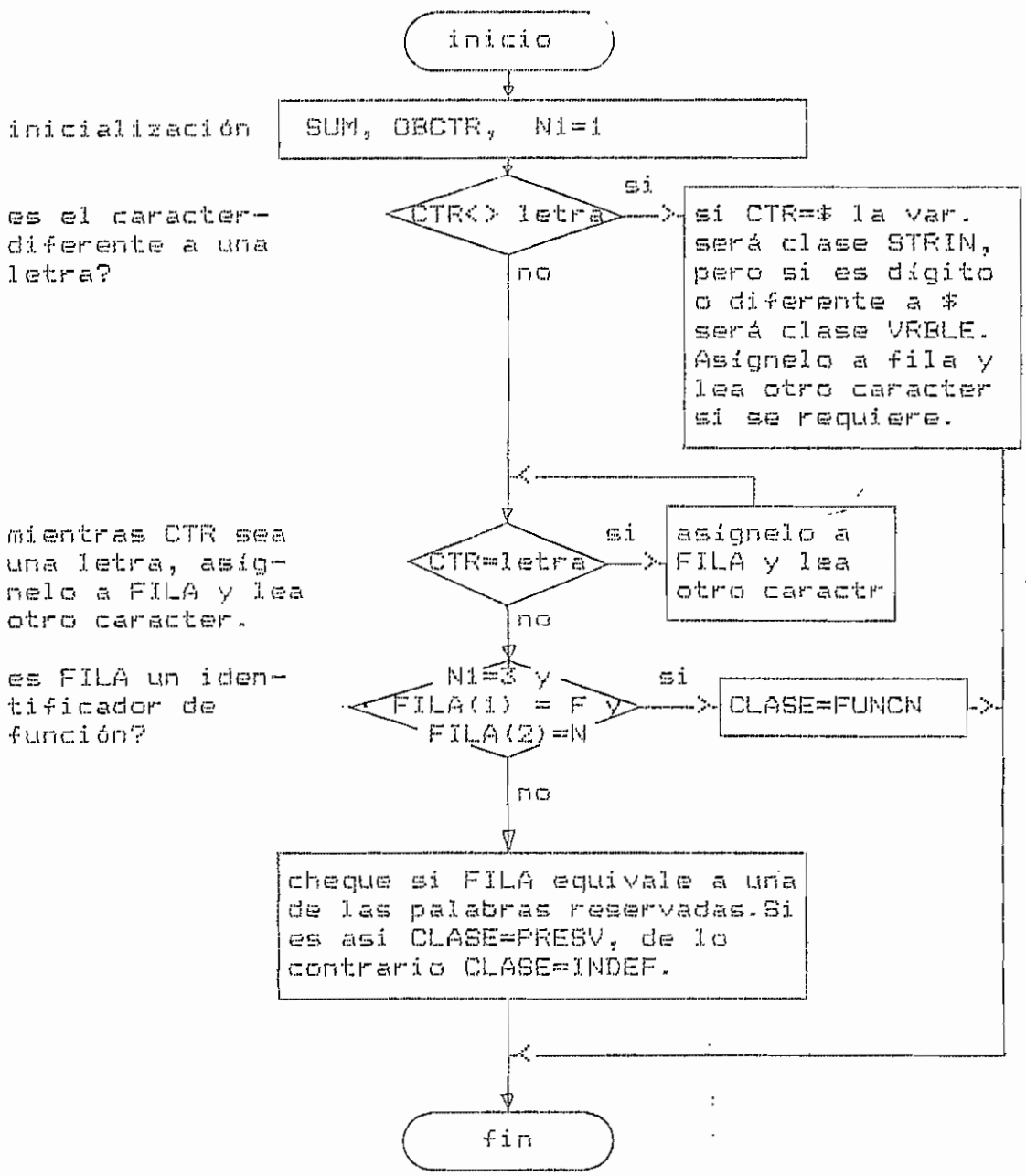


Fig. 4.16.- Diagrama de flujo de la rutina SIMBI.

4.3.10.2.- Rutina SIMB2 (interna a OBSIMB).

Construye números enteros y asigna su valor decimal a la variable NUMERO.

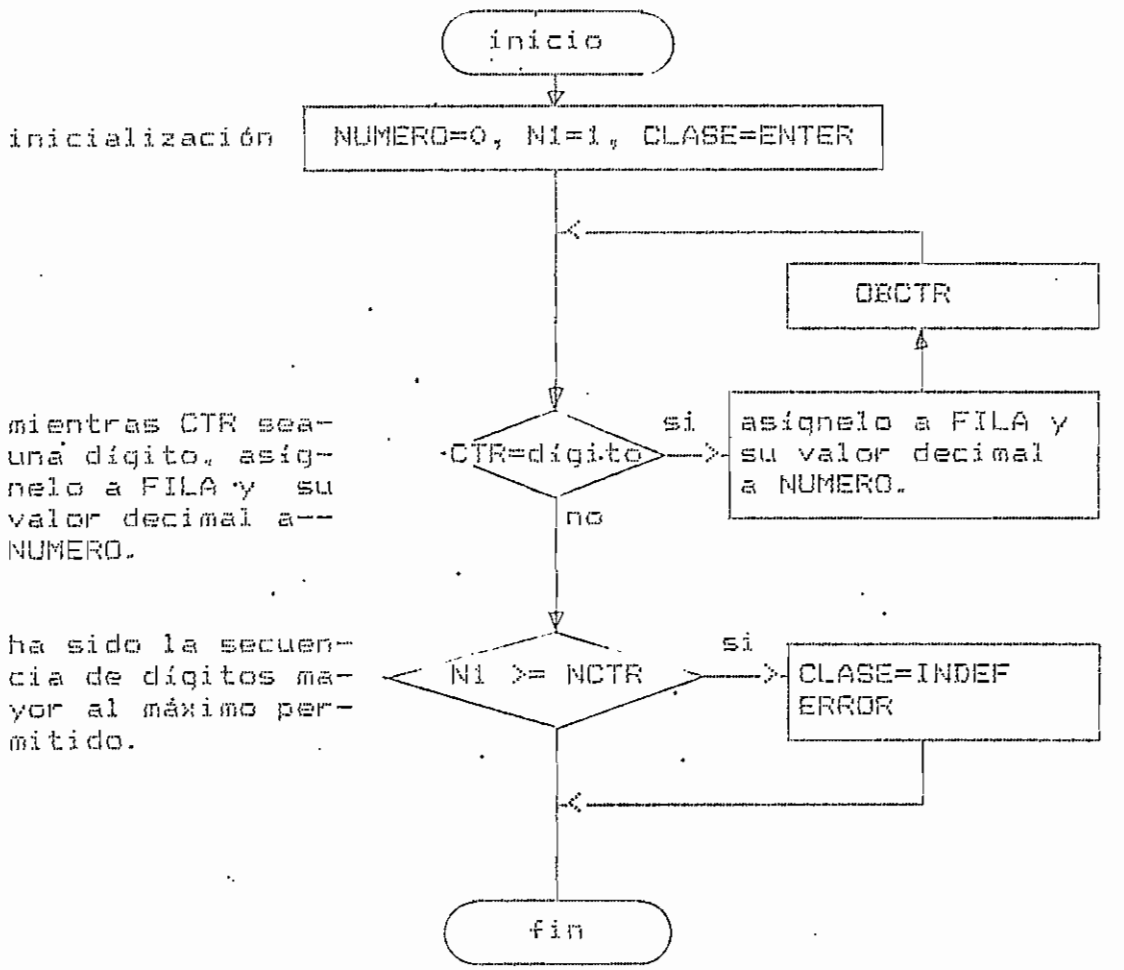


Fig. 4.17.- Diagrama de flujo de la rutina SIMB2.

4.3.10.3.- Rutina SIMB3 (interna a OBSIMB).

Construye delimitadores e indefinidos.

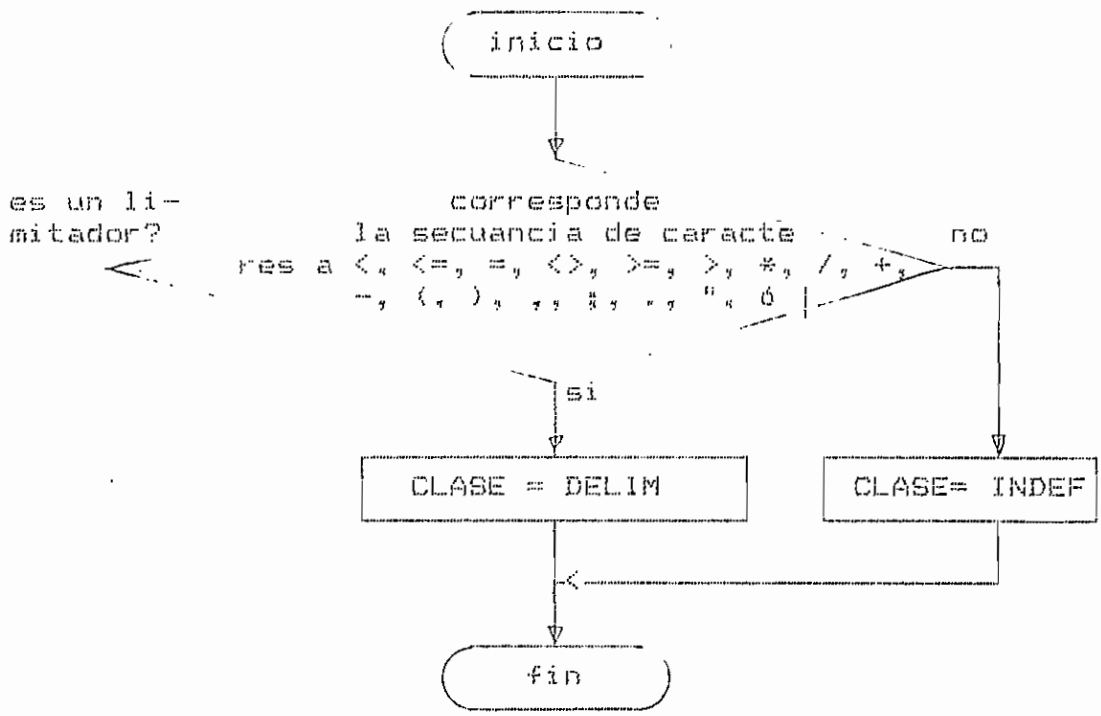


Fig. 4.18.- Diagrama de flujo de la rutina SIMB3.

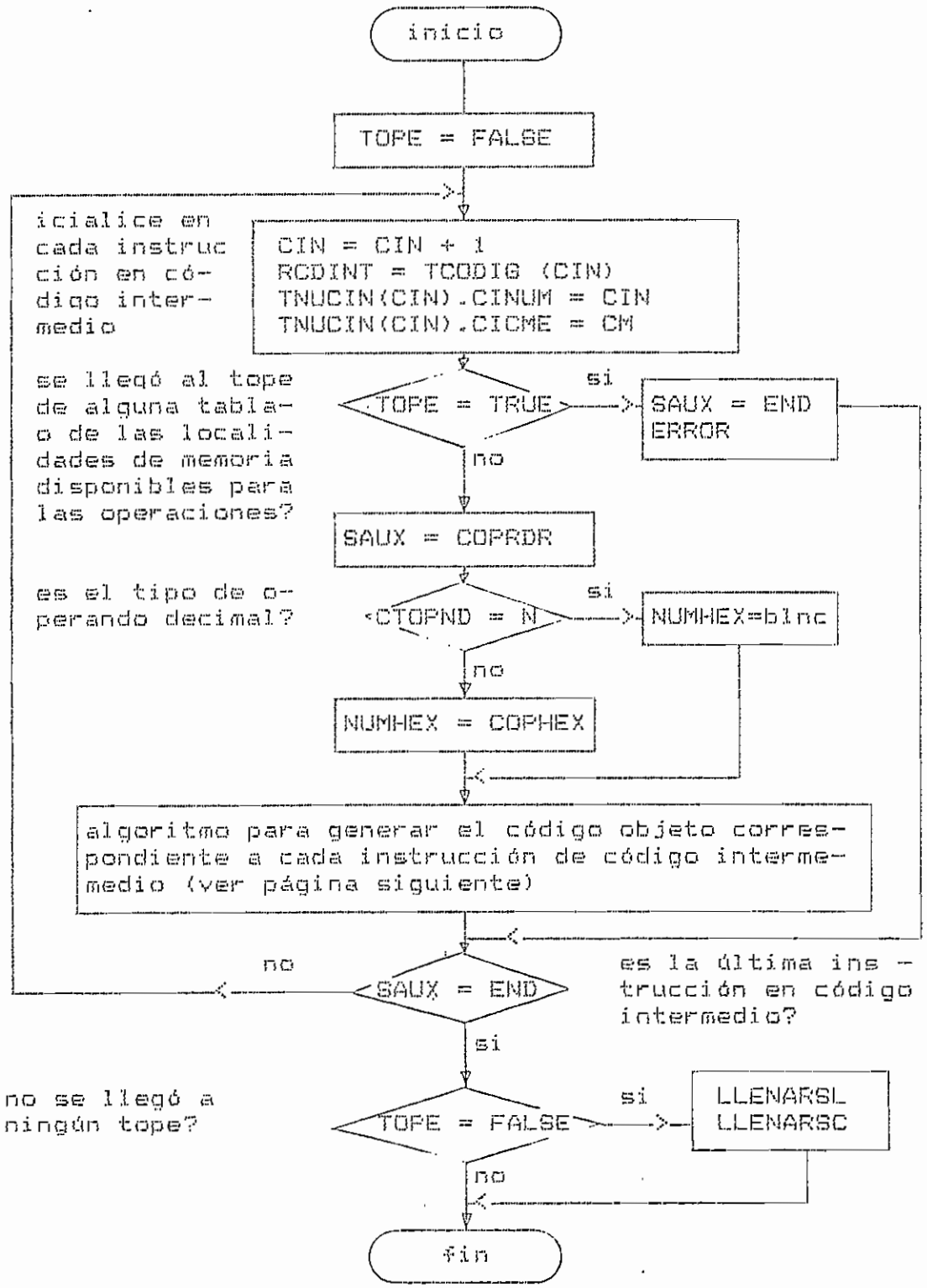
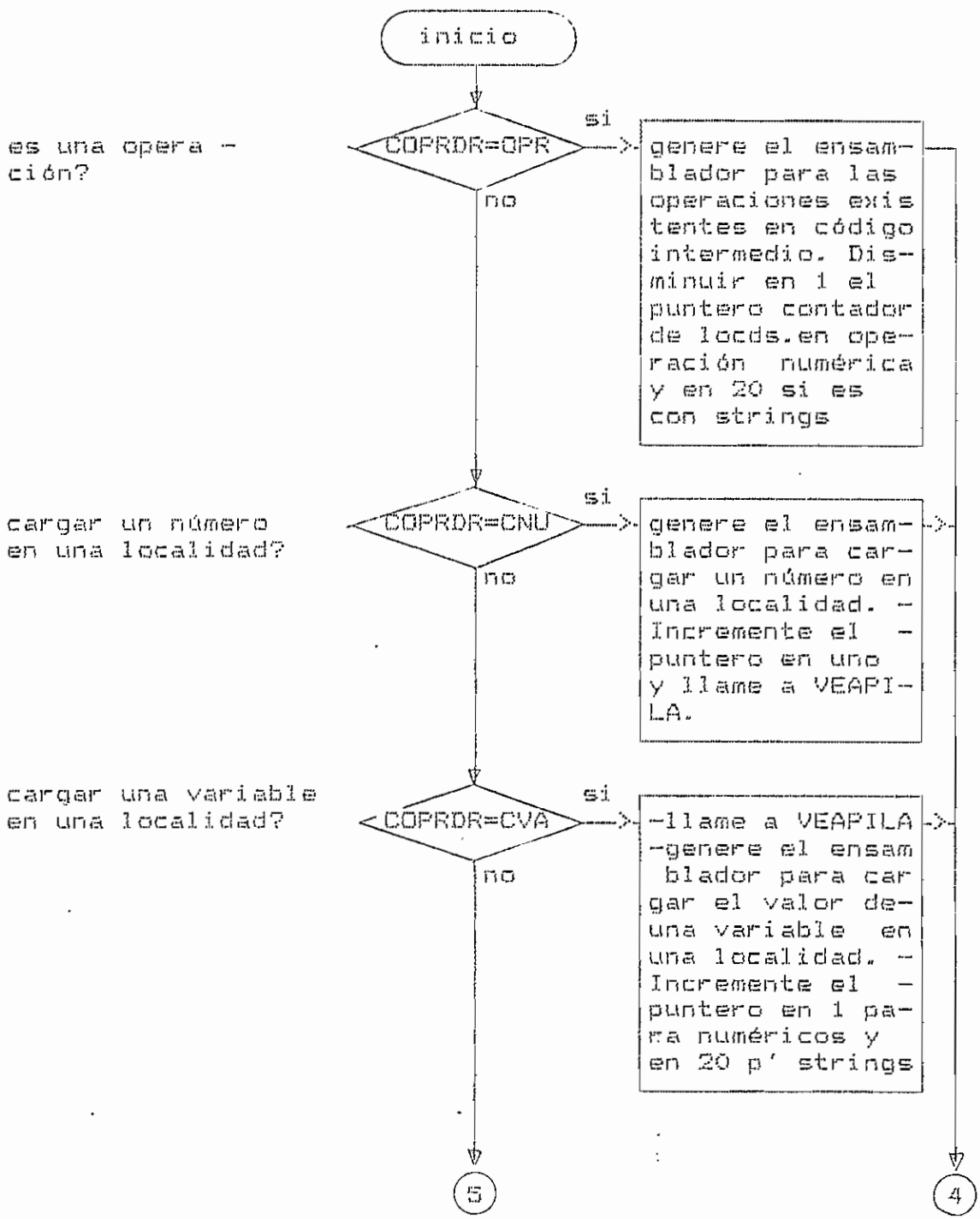


Fig. 4.20.- Diagrama de flujo de la rutina CODIGASS.

Algoritmo para generar el código objeto correspondiente a cada instrucción en código intermedio.



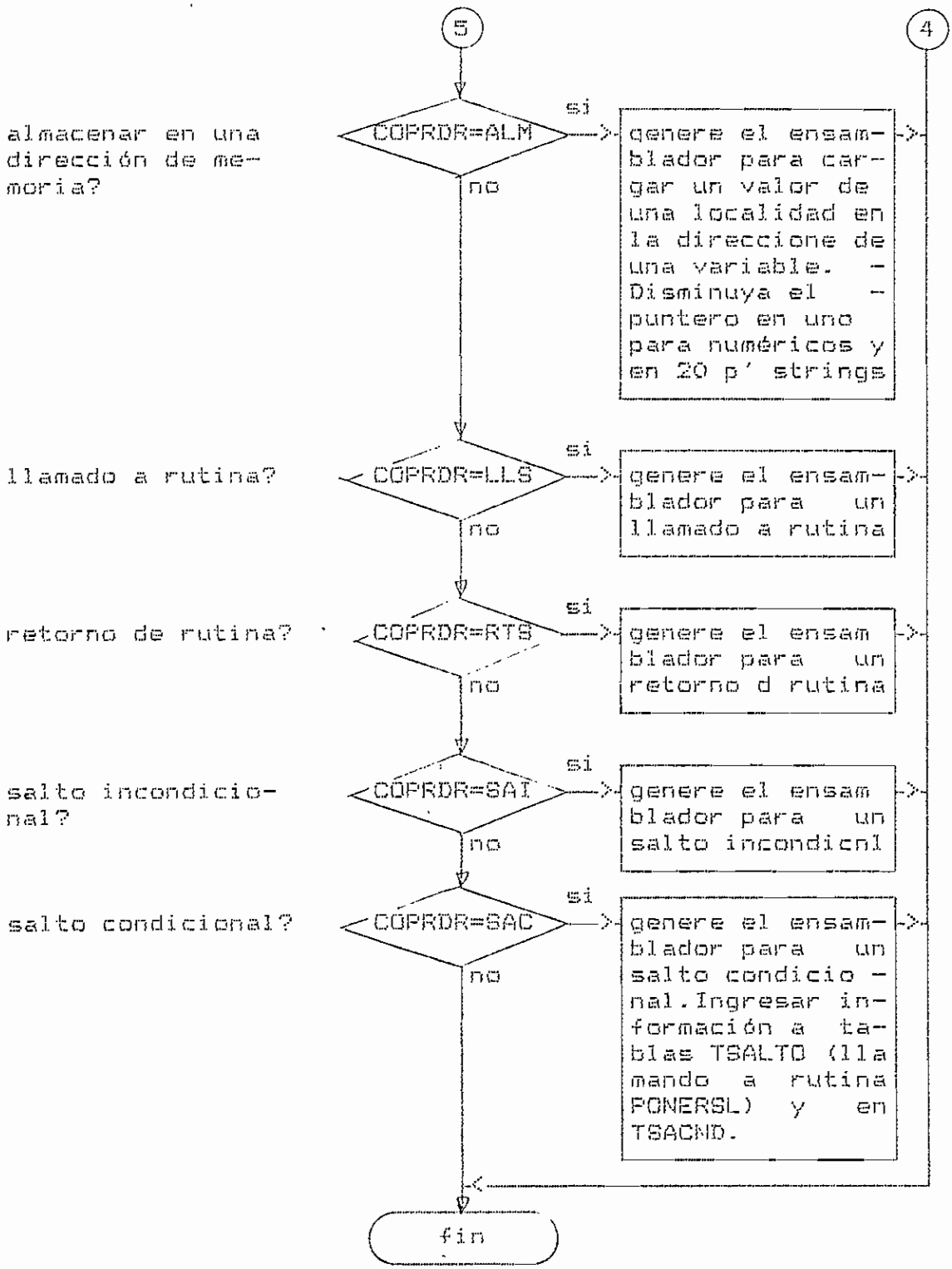


Fig. 4.20.a.- Algoritmo auxiliar de la Fig. 4.20.

4.3.11.1.- Rutina GENASS (interna a CODIGASS).

Genera una instrucción en código ensamblador y la ingresa a la tabla TASSEM.

parámetros:

GA1= indica la localidad de memoria donde empezará a escribirse la instrucción en ensamblador.

GA2= operador ensamblador del microprocesador M6800.

GA3= indica el tipo de direccionamiento de la instrucción.

GA4= indica el tipo de operando: decimal (N) ó hexadecimal (H).

GA5= operando numérico.

GA6= indica si se hará una operación con el registro índice.

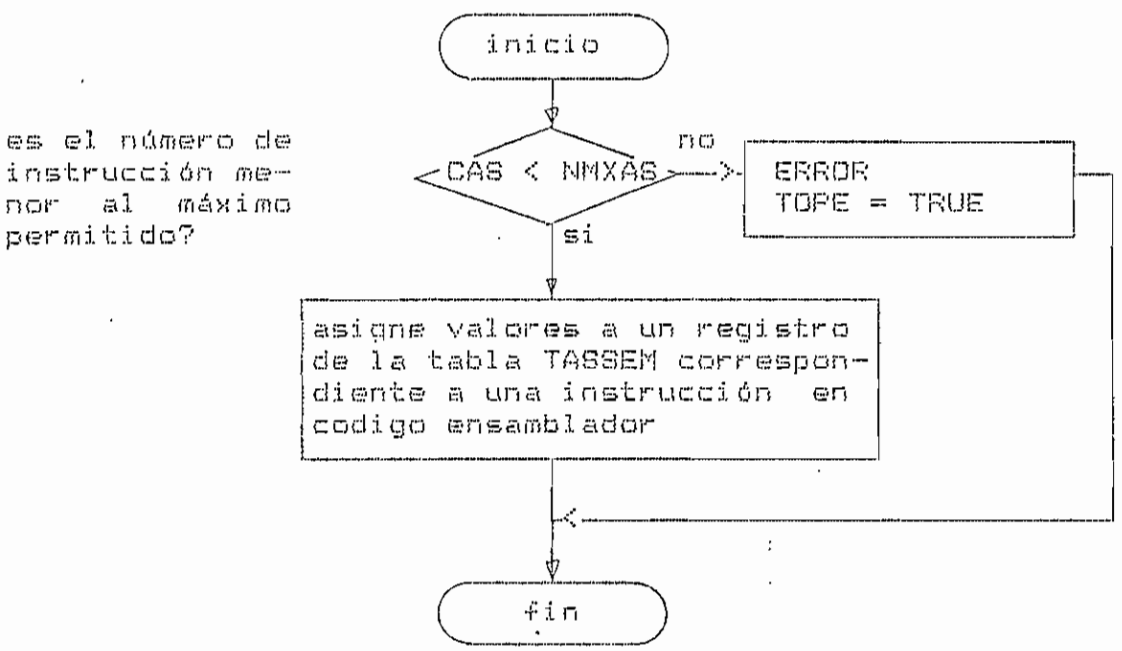


Fig. 4.21.- Diagrama de flujo de la rutina GENASS.

4.3.ii.2.- Rutina VEAPILA (interna a CODIGASS).

Chequea si se ha llegado al tope de las localidades de memoria disponibles para almacenar valores de variables y para realizar operaciones.

parámetros:

VP = valor actual del puntero de localidades de memoria.

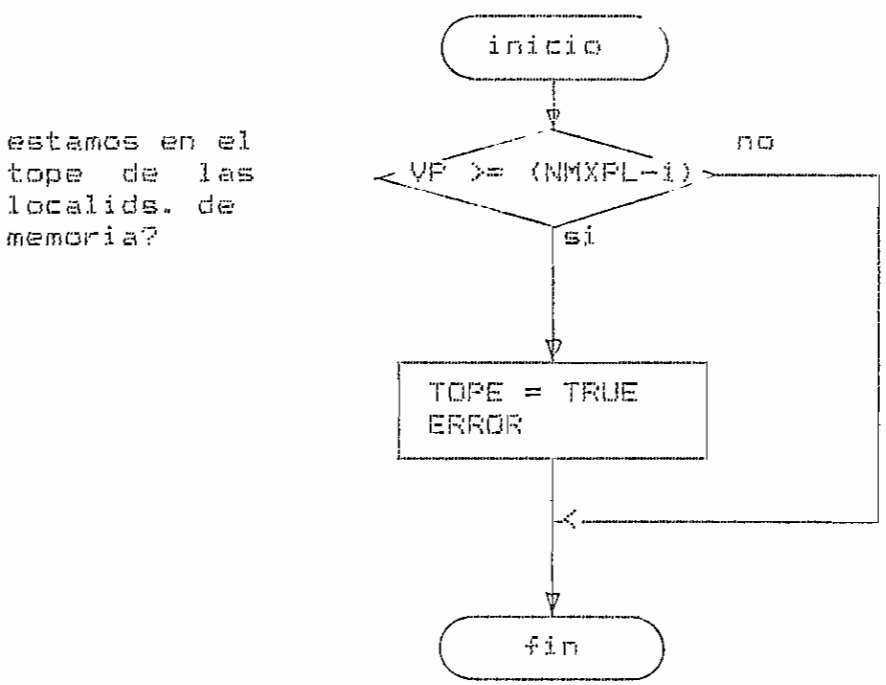


Fig. 4.22.- Diagrama de flujo de la rutina VEAPILA.

4.3.11.3.- Rutina LLENARSL (interna a CODIGASS).

Completa las instrucciones, en ensamblador, con información relacionada con saltos incondicionales en código intermedio.

variables locales:

I1, I2 = contadores.

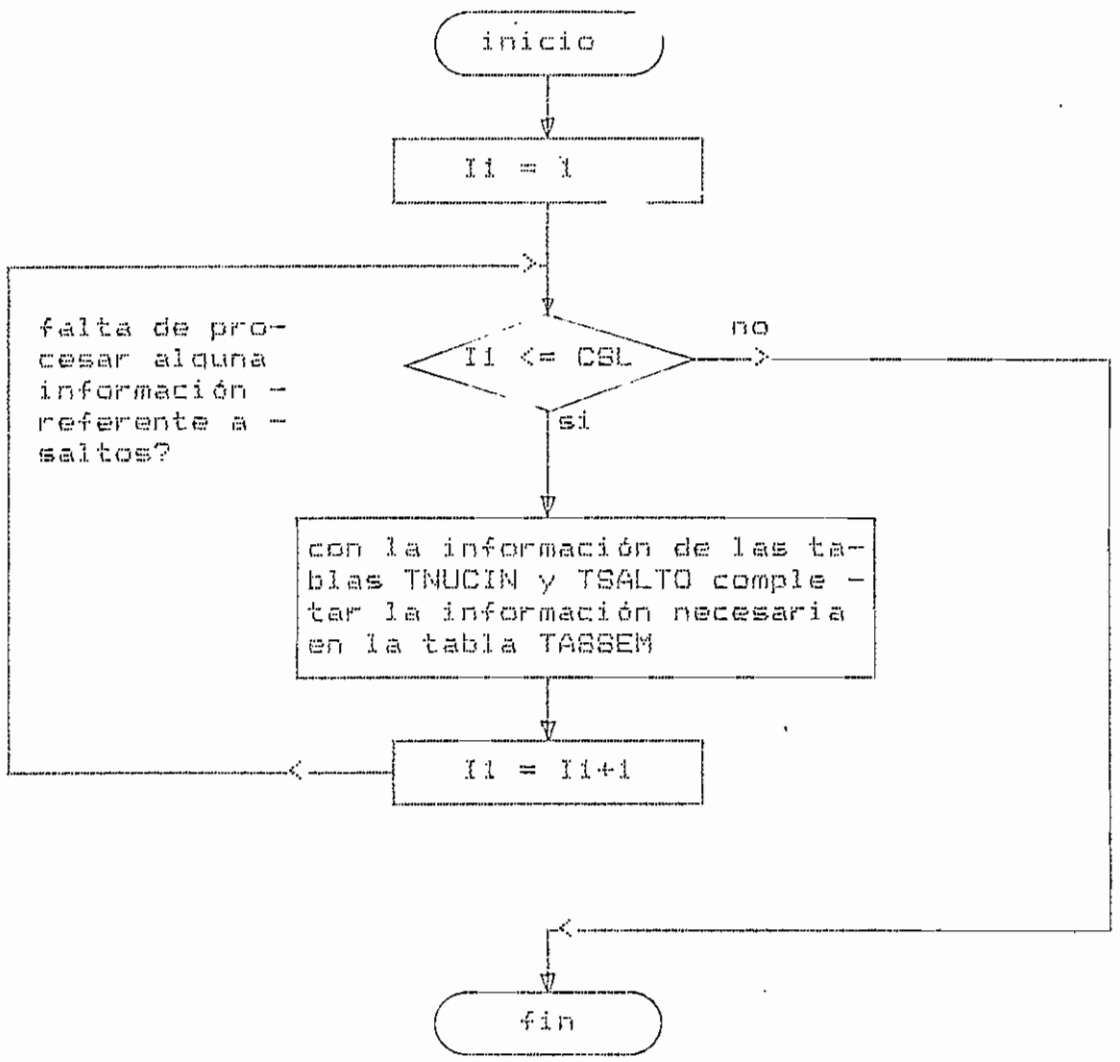


Fig. 4.23.- Diagrama de flujo de la rutina LLENARSL.

4.3.ii.4.- Rutina LLENARSC (interna a CODIGASS).

Completa las instrucciones, en ensamblador, con información relacionada con saltos condicionales en código intermedio.

variables locales:

I = contador.

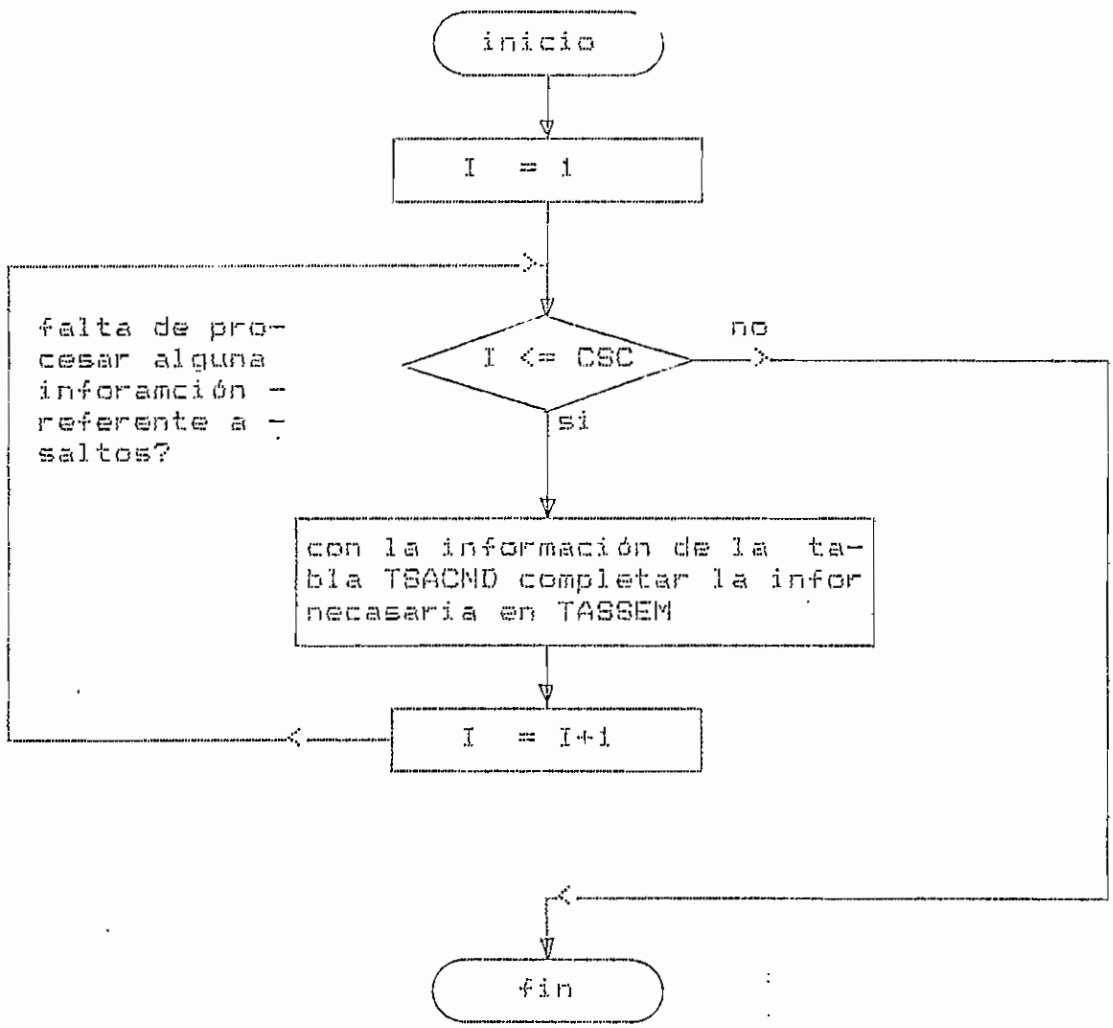


Fig. 4.24.- Diagrama de flujo de la rutina LLENARSC.

4.3.ii.5.- Rutina PONERSL (interna a CODIGASS).

Ingresa a la tabla TSALTO información necesaria sobre saltos en instrucciones en código intermedio.

parámetros:

SL = número de instrucción , en código intermedio, en la que existe un salto.

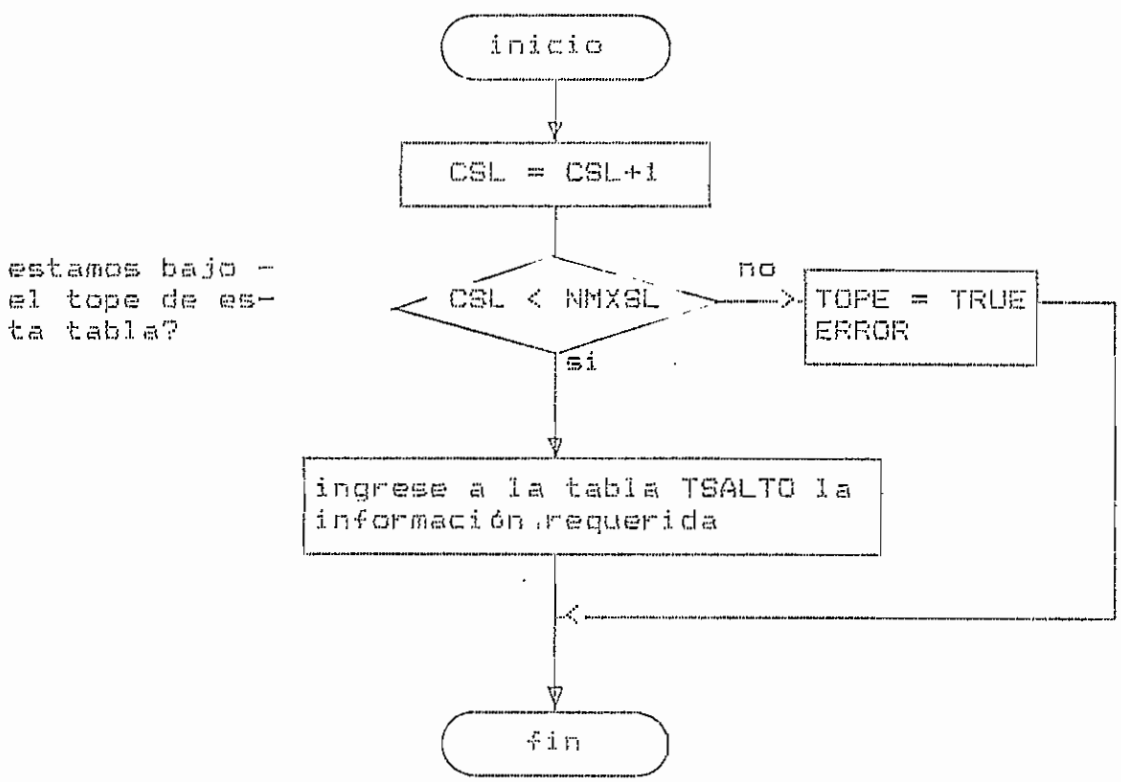


Fig. 4.25.- Diagrama de flujo de la rutina PONERSL.

4.3.ii.6.- Rutina CONCATENAR (interna a CODIGASS).

Realiza la concatenación de dos strings que estan almacenados en las 40 locadlidades de memoria superiores y pone el resultado en las 20 inferiores de estas 40.

variables locales:

N1, N2, N3 = contadores.

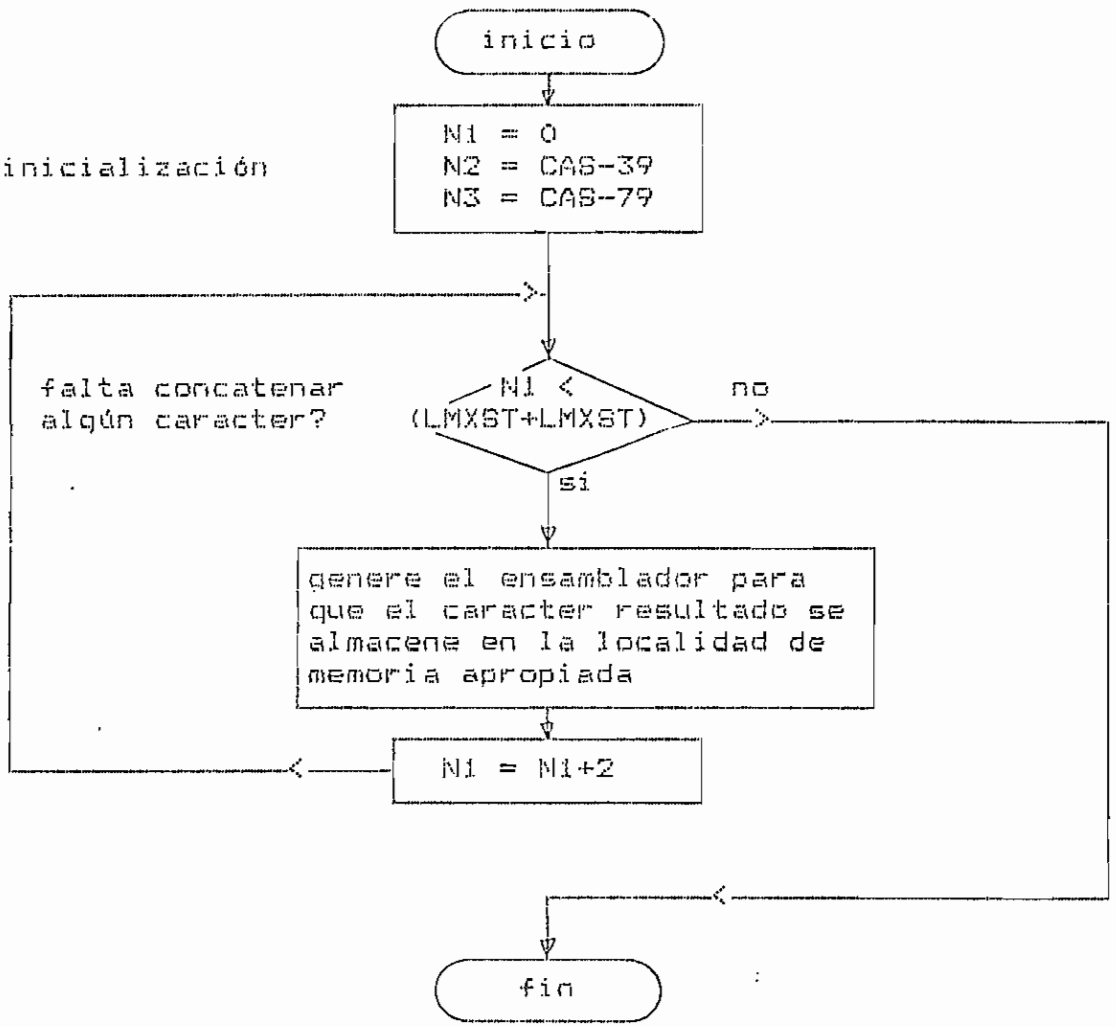


Fig. 4.26.- Diagrama de flujo de la rutina CONCATENAR.

4.3.11.7.- Rutina SIGNO (interna a CODIGASS).

Se la utiliza para el caso de operaciones de multiplicación o división. Chequea el signo de los números almacenados en las direcciones de localidades DPL y (DPL+1), y almacena el resultado en la localidad cero. El resultado puede ser: 1 si los dos números son de signo diferente y 0 si son de igual signo. Ambos números se almacenan en sus localidades respectivas pero con signo positivo siempre. La rutina tiene el siguiente diagrama:

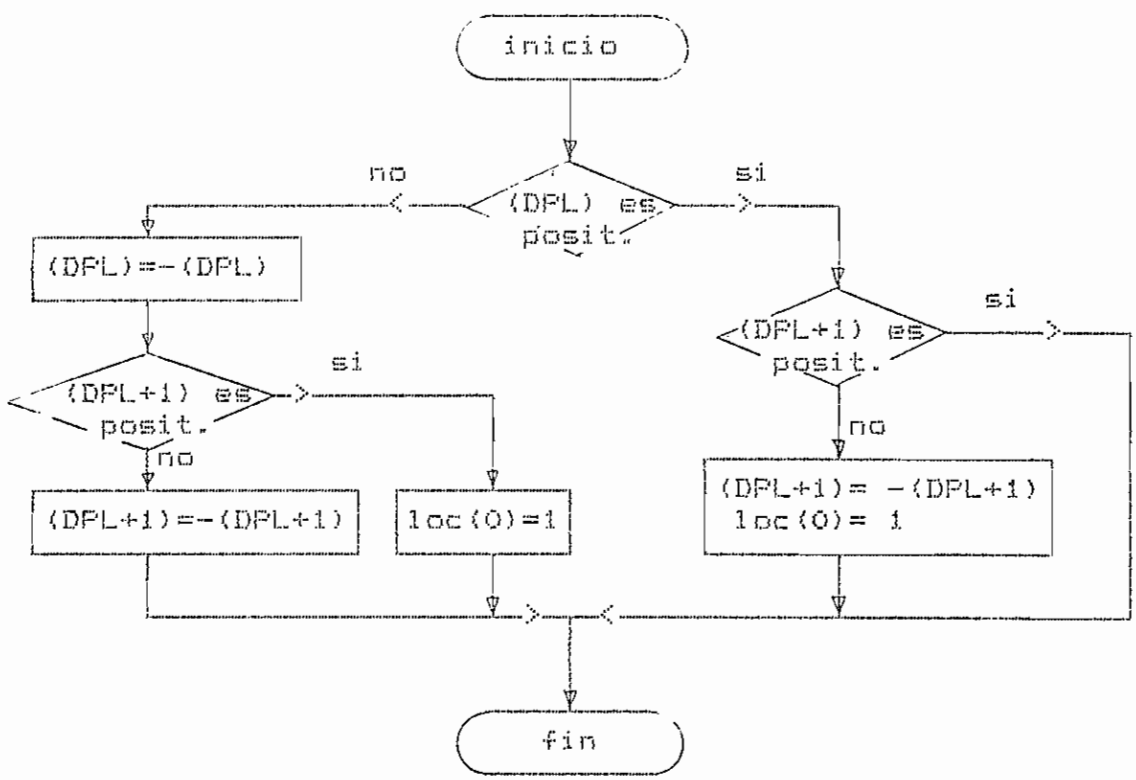


Fig. 4.27.- Diagrama de flujo de la rutina SIGNO.

4.3.11.8.- Rutina RESULT (interna a CODIGASS).

Se la utiliza para el caso de operaciones de multiplicación o división. Chequea si el signo de los dos números con que se hizo la operación, han sido iguales o diferentes. Si los signos han sido diferentes, se cambiará el signo del resultado que está en la localidad que indica el puntero de DPL. La rutina tiene el siguiente diagrama:

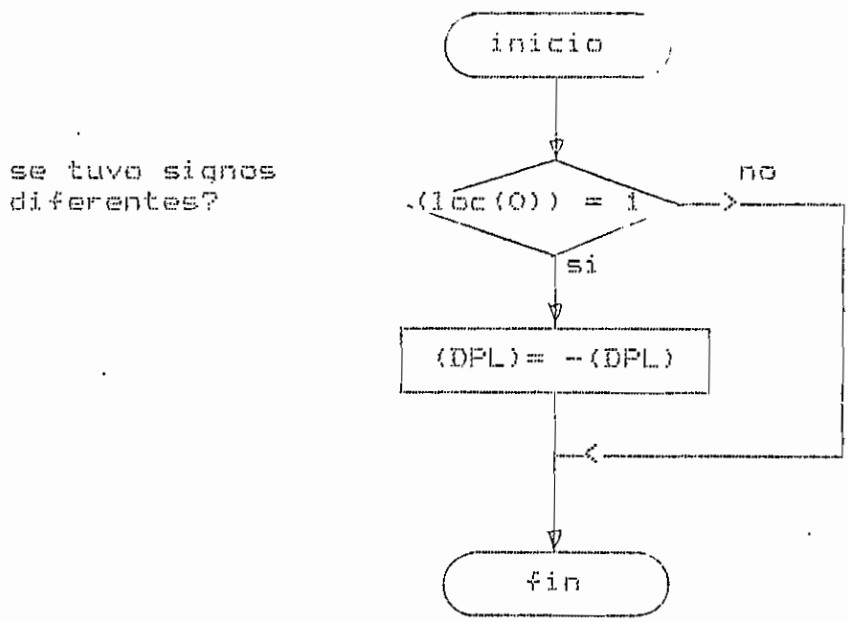


Fig. 4.28.- Diagrama de flujo de la rutina RESULT.

4.3.12.- Rutina externa EXPRESION.

Analiza y genera el código intermedio de una expresión en VERSION BASIC. Siempre almacena en la variable TPEXPRE el tipo de expresión analizada: numérica (N), de relación (lógica) (L) o string (S).

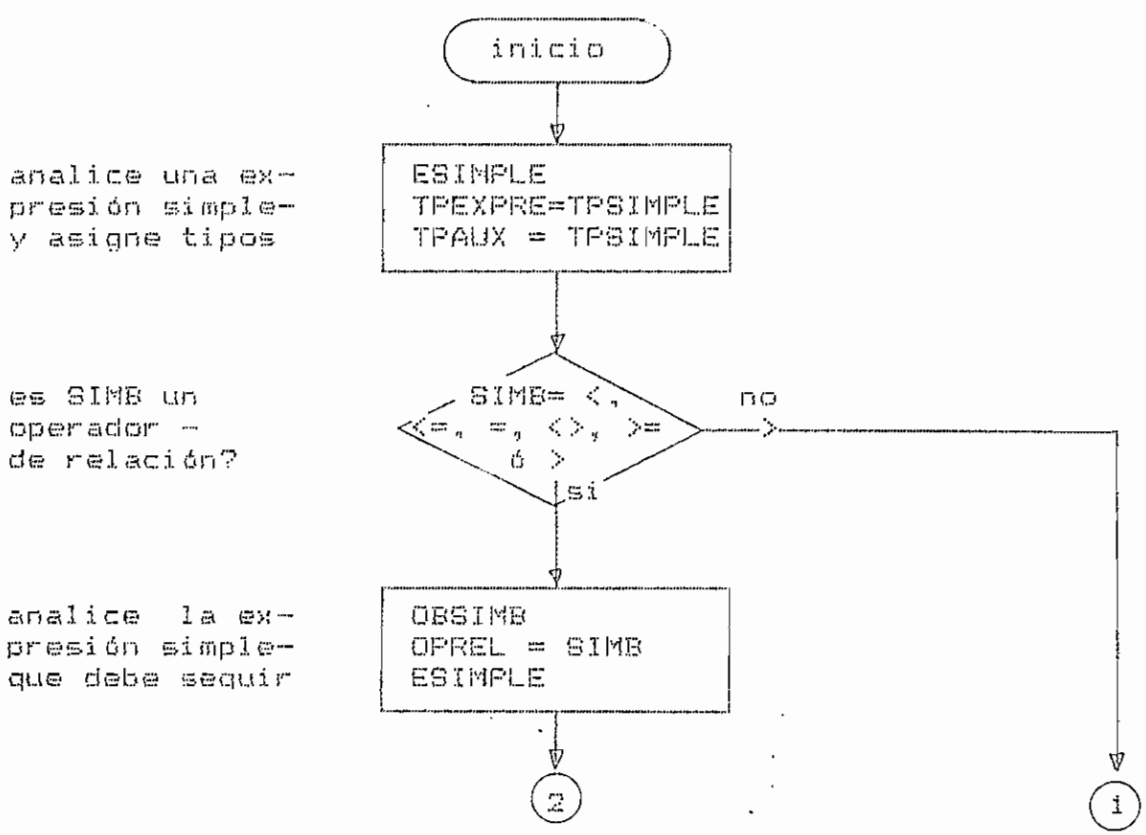
variables locales:

I = contador.

OPREL = auxiliar para almacenar el operador de relación.

TPAUX, TPFACTOR, TPTERMINO, TPSIMPLE = auxiliares para almacenar el tipo de factor, término y expresión simple.

CTRAUX = auxiliar.

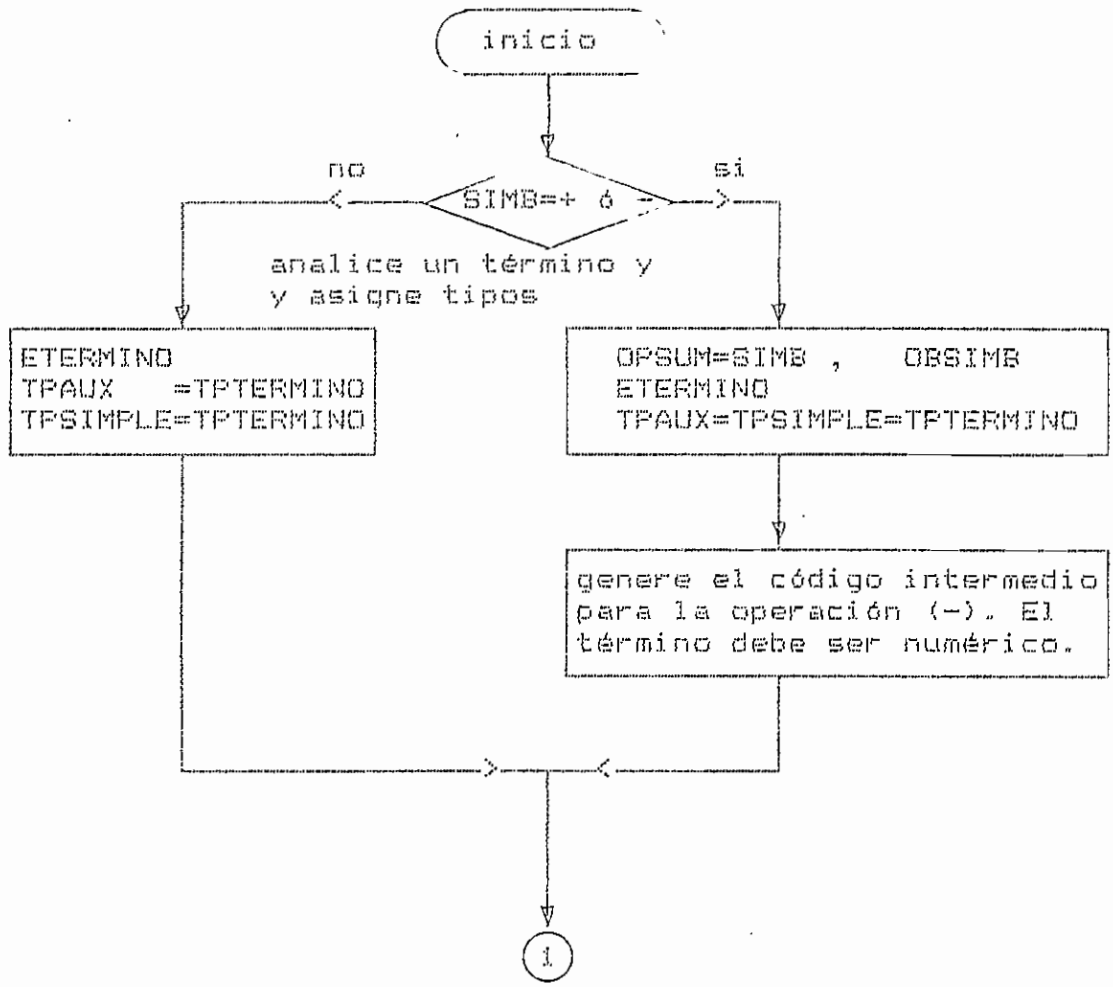


4.3.12.i.- Rutina ESIMPLE (interna a EXPRESION).

Analiza una expresión simple y genera el código intermedio respectivo. El tipo se almacena en la variable TPSIMPLE.

variables locales:

OPSUM = auxiliar para almacenar el operador.



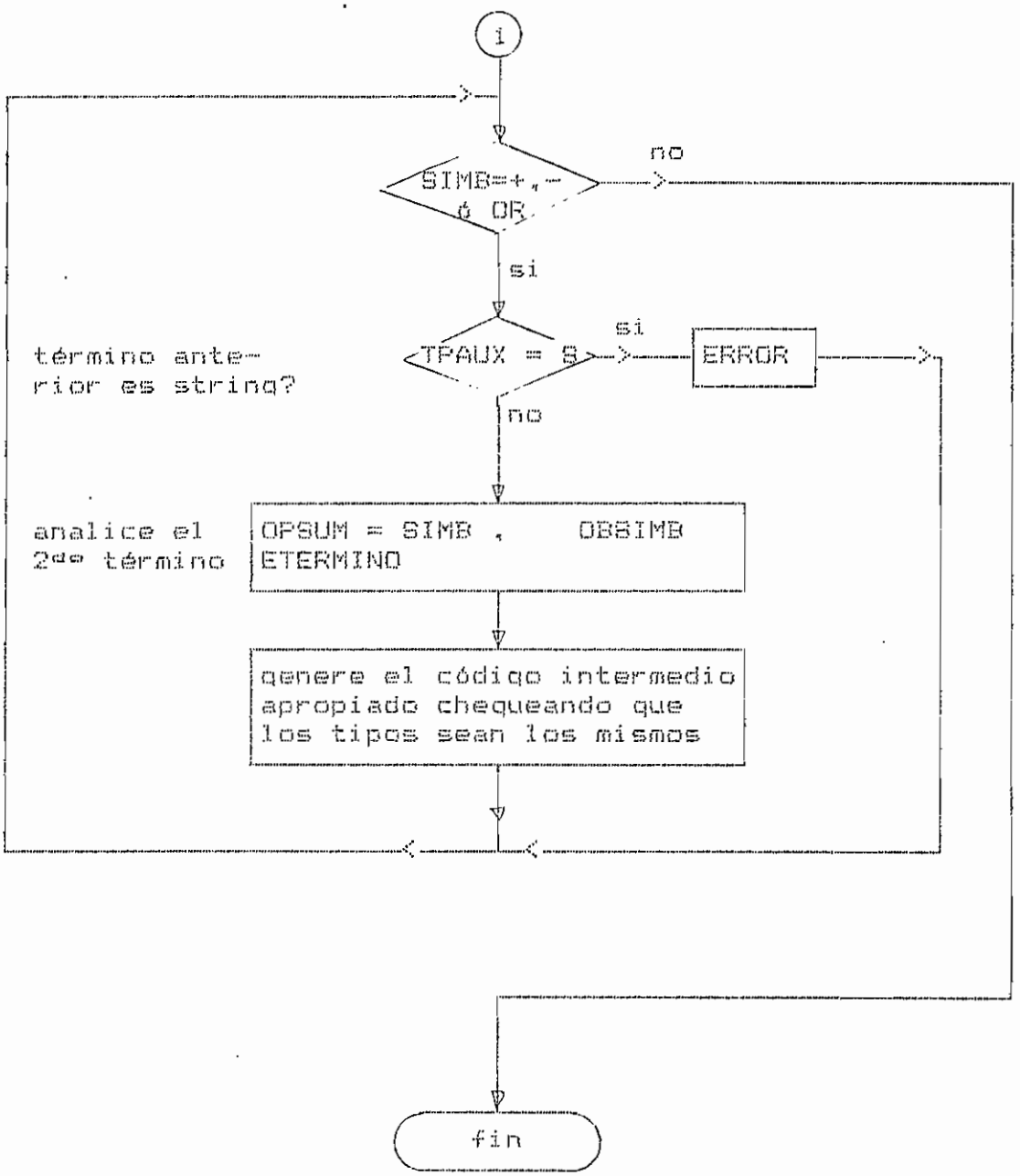


Fig. 4.30.- Diagrama de flujo de la rutina ESIMPLE.

4.3.12.2.- Rutina ETERMINO (interna a EXPRESION).

Analiza un término en una expresión.

variables locales:

OPMUL = Auxiliar para almacenar el operador entre términos.

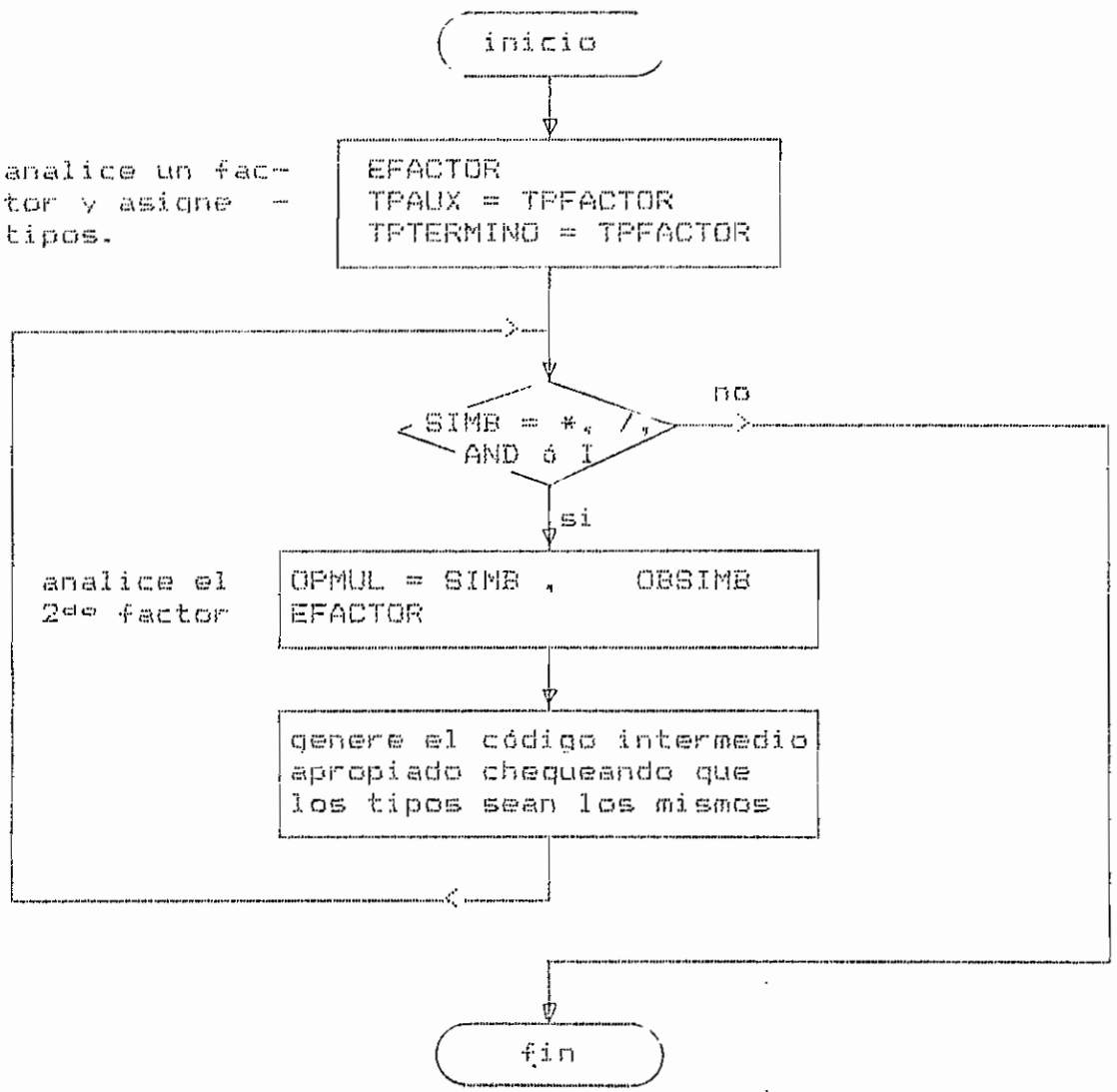


Fig. 4.3i.- Diagrama de flujo de la rutina ETERMINO.

4.3.12.3.- Rutina EFACTOR (interna a EXPRESION).

Analiza un factor en una expresión.

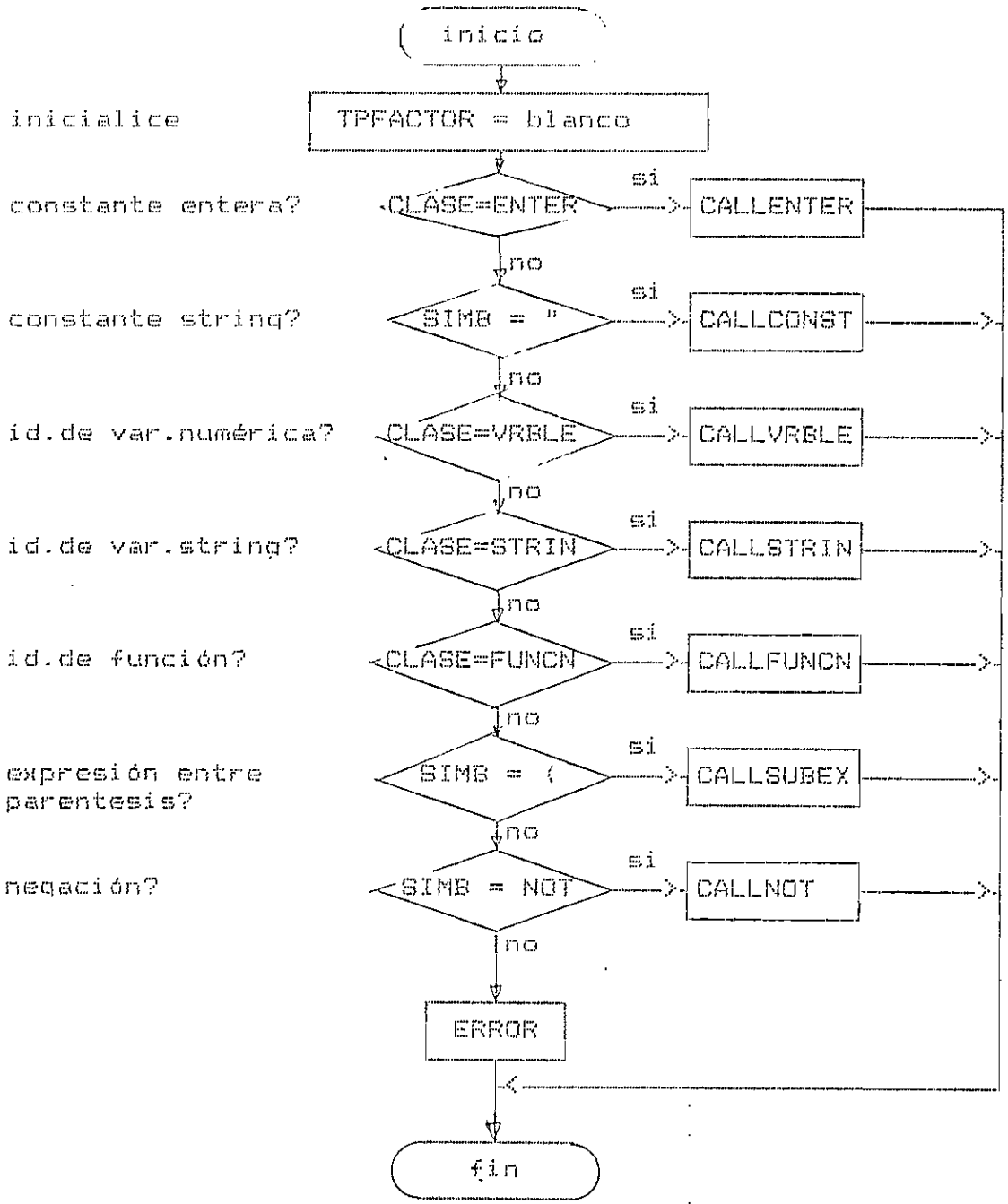


Fig. 4.32.- Diagrama de flujo de la rutina EFACTOR.

4.3.i2.3.2.- Rutina CALLCONST (interna a EFACTOR).

Analiza una constante string.

variables locales:

Ni = contador.

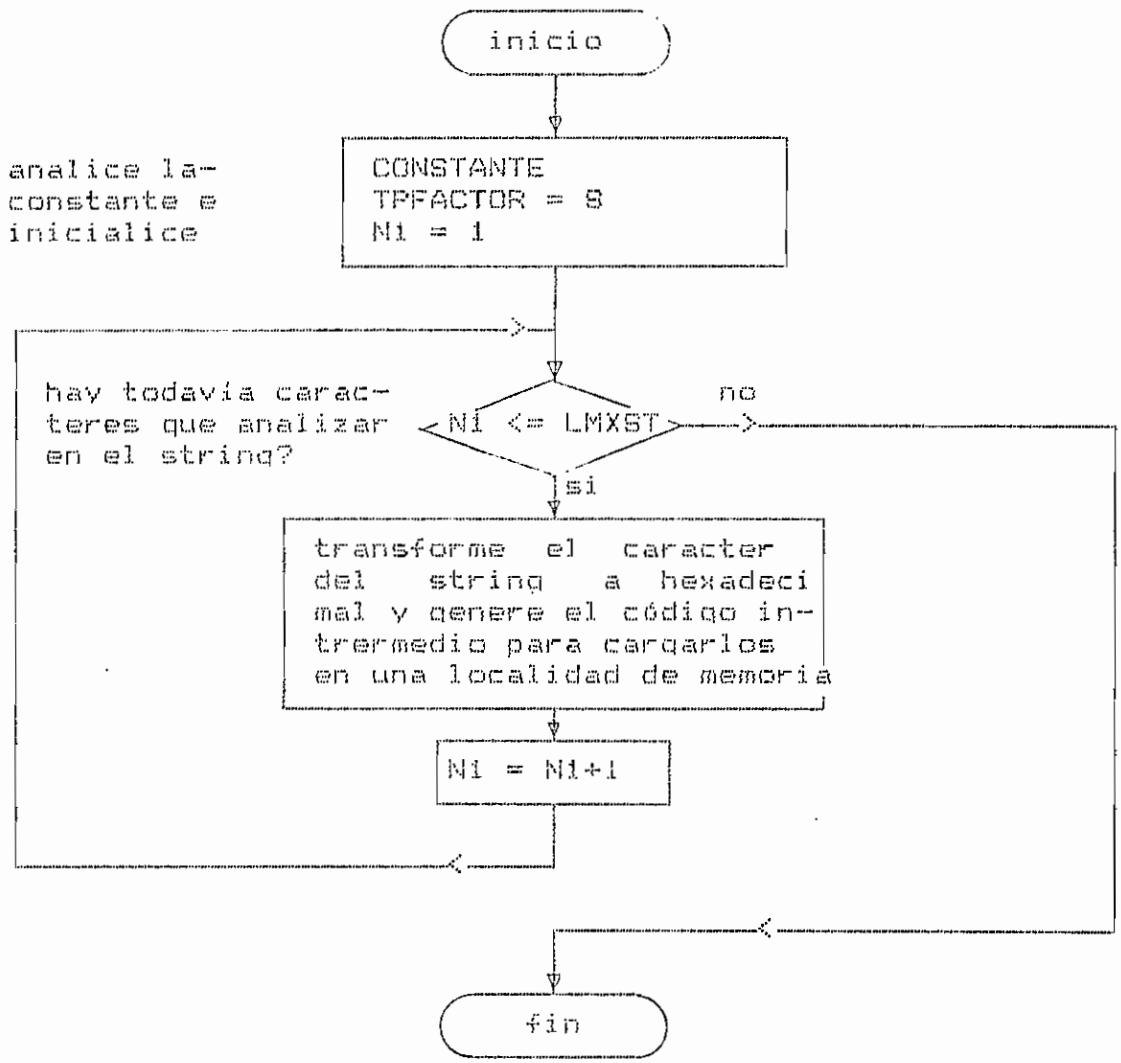


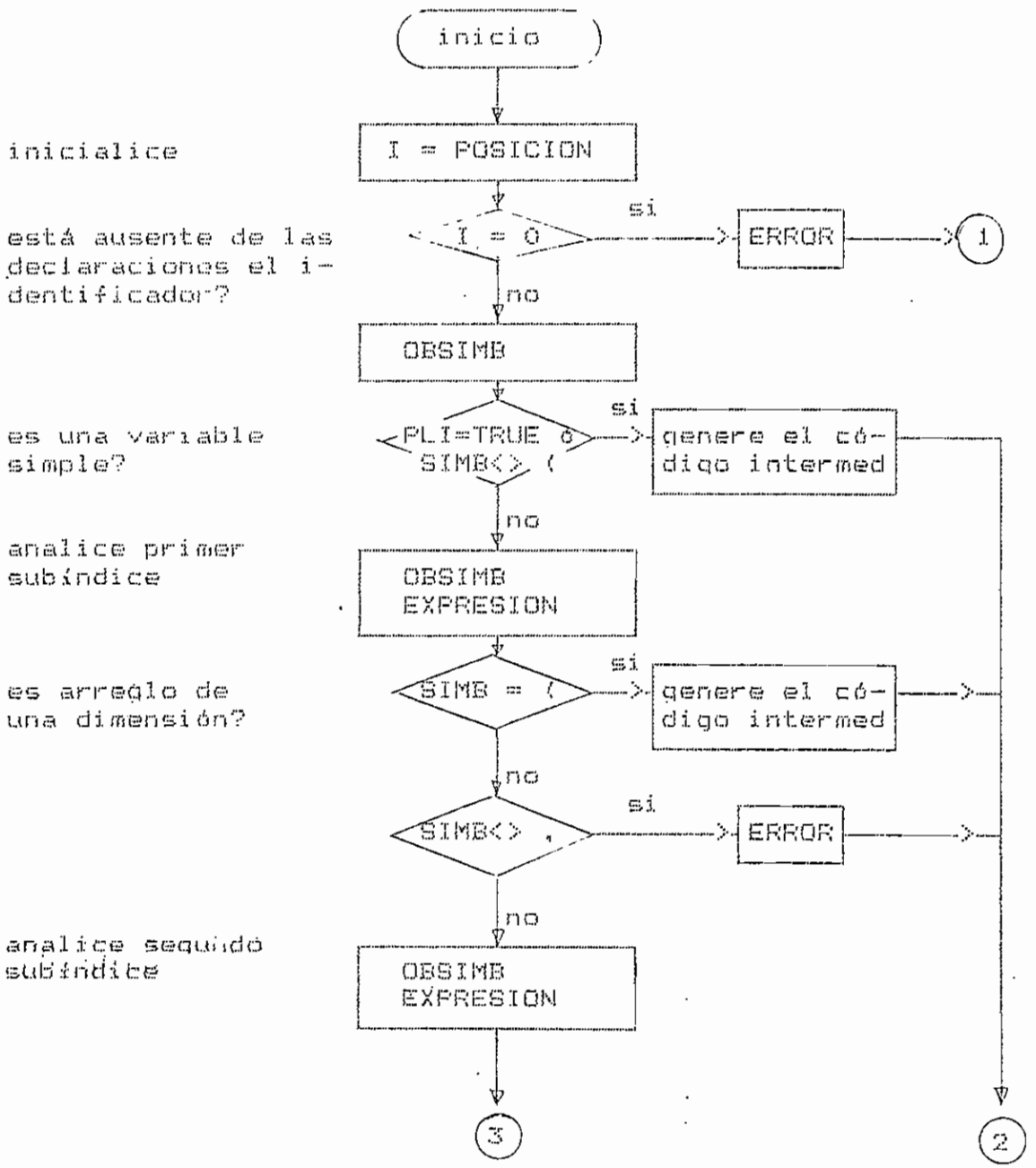
Fig. 4.34.- Diagrama de flujo de la rutina CALLCONST.

4.3.12.3.3.- Rutina CALLVRBLE (interna a EFACTOR).

Analiza una variable numérica simple o de un arreglo

variables locales:

I = contador.



4.3.i2.3.4.- Rutina CALLSTRIN (interna a EFACTOR).

Analiza una variable string.

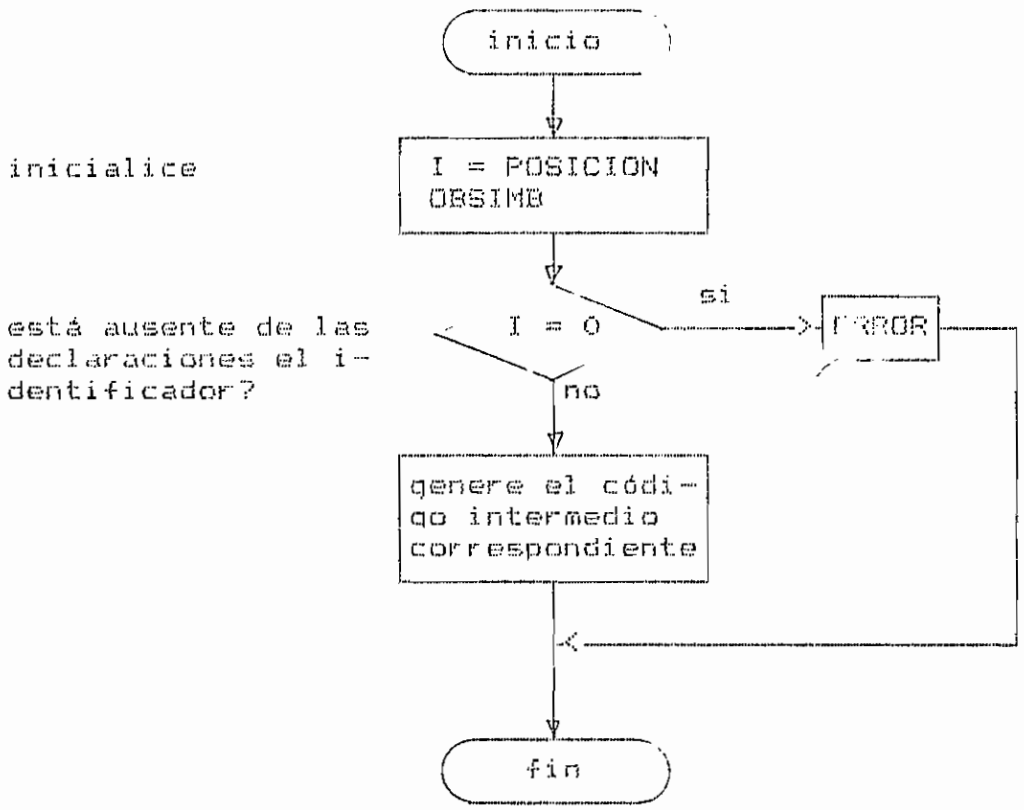


Fig. 4.36.- Diagrama de flujo de la rutina CALLSTRIN.

4.3.12.3.5.- Rutina CALLFUNCN (interna a EFACTOR).

Analiza una llamada a función.

variables locales:

I1 = contador.

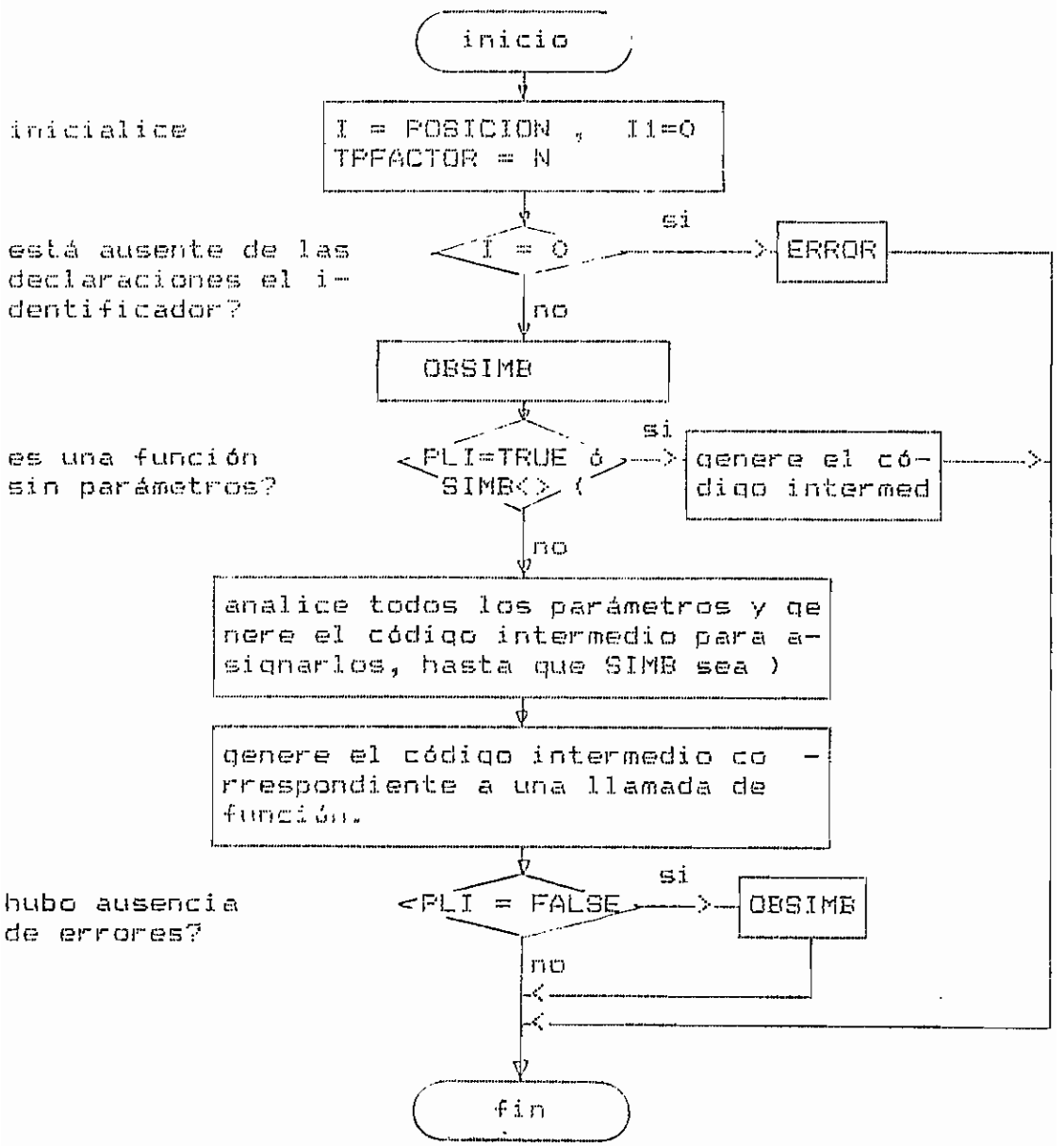


Fig. 4.37.- Diagrama de flujo de la rutina CALLFUNCN.

4.3.12.3.6.- Rutina CALLSUBEX (interna a EFACTOR).

Analiza una expresión encerrada entre paréntesis.

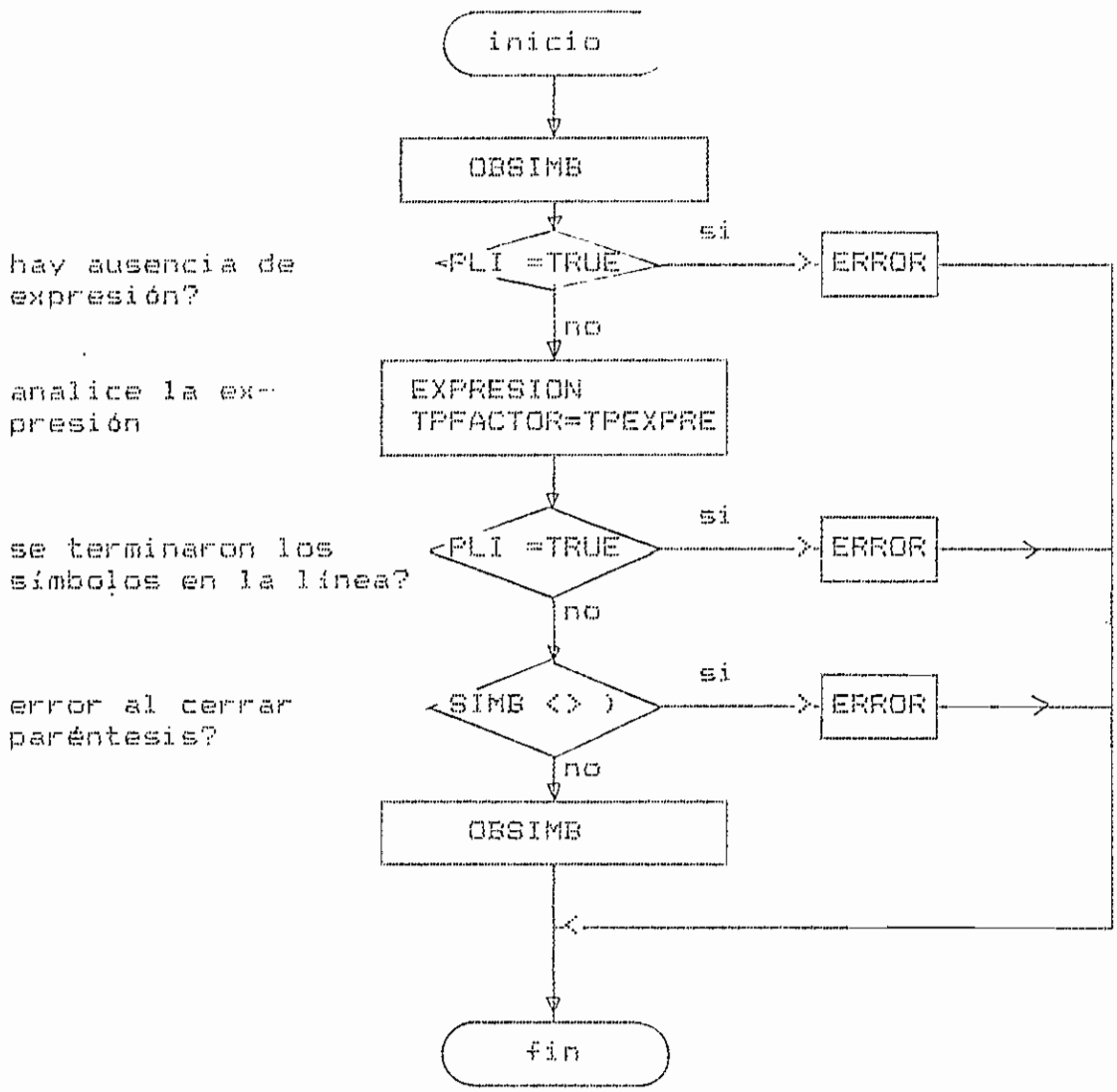


Fig. 4.38.- Diagrama de flujo de la rutina CALLSUBEX.

4.3.i3.- Rutina ANALISIS.

Realiza el análisis de un programa en VERSION BASIC.

variables locales:

CDTAUX = contador auxiliar de datos.

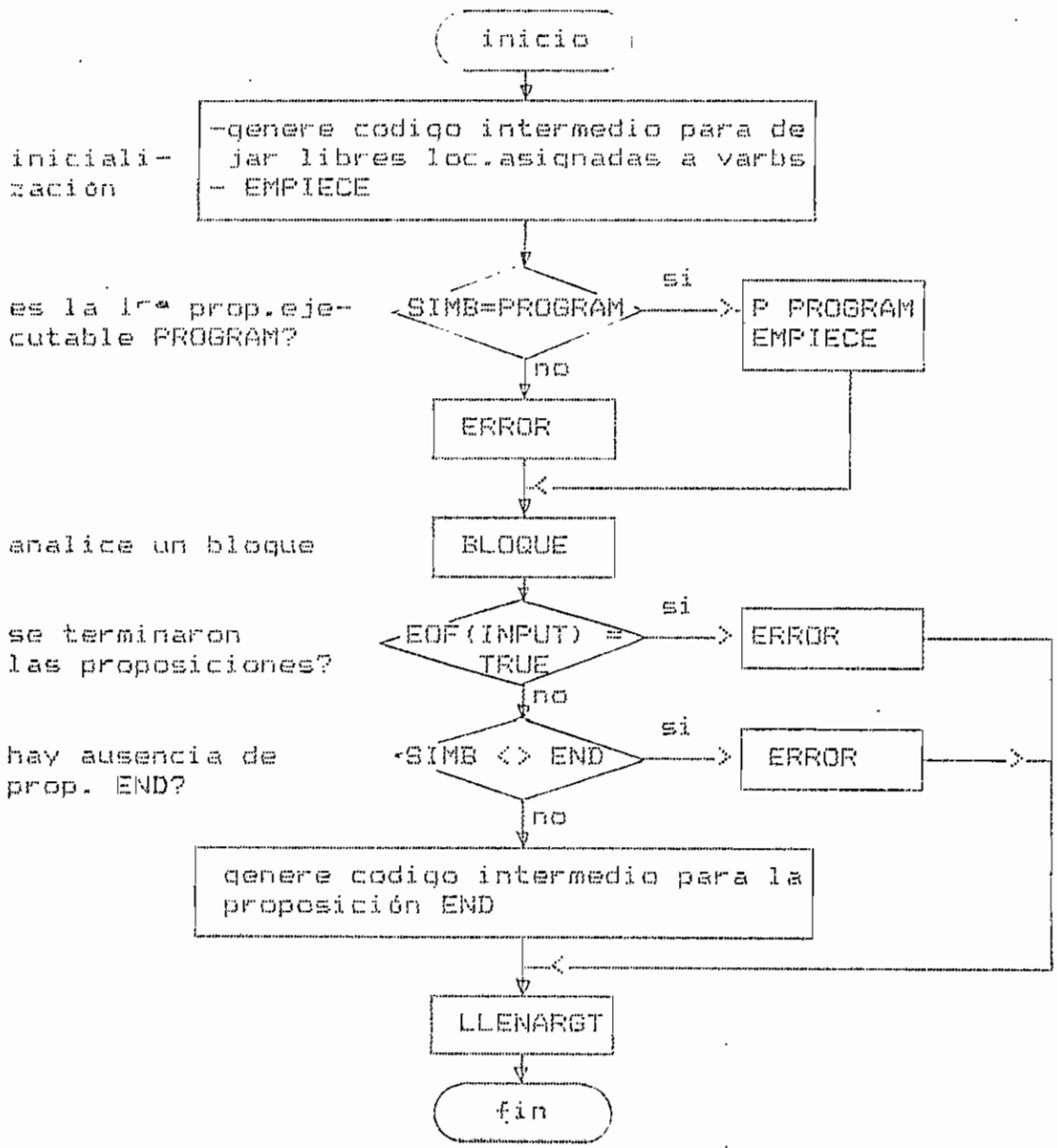
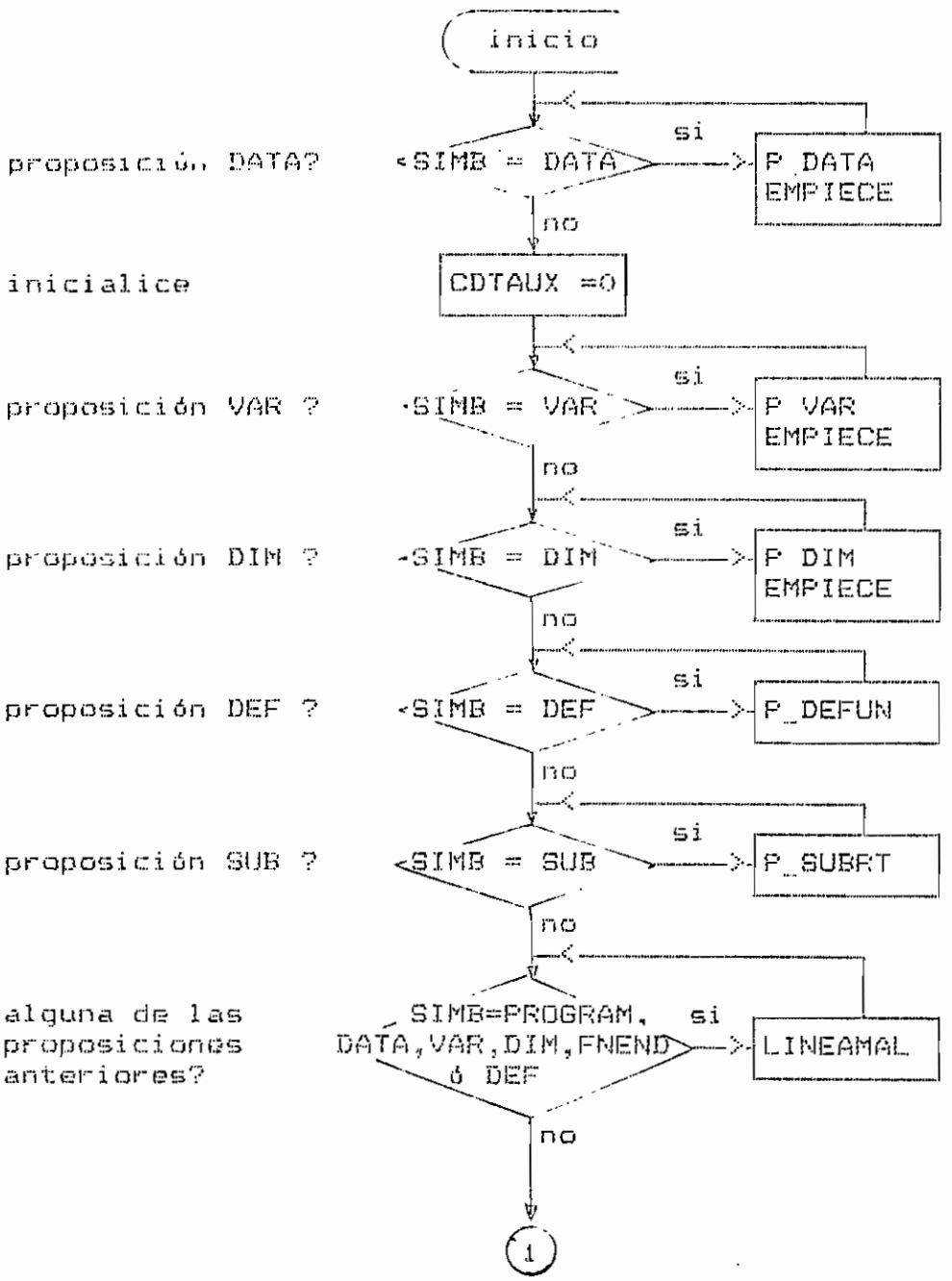


Fig. 4.40.- Diagrama de flujo de la rutina ANALISIS.

4.3.13.1.- Rutina BLOQUE (interna a ANALISIS).

Realiza el análisis de un bloque en VERSION BASIC.



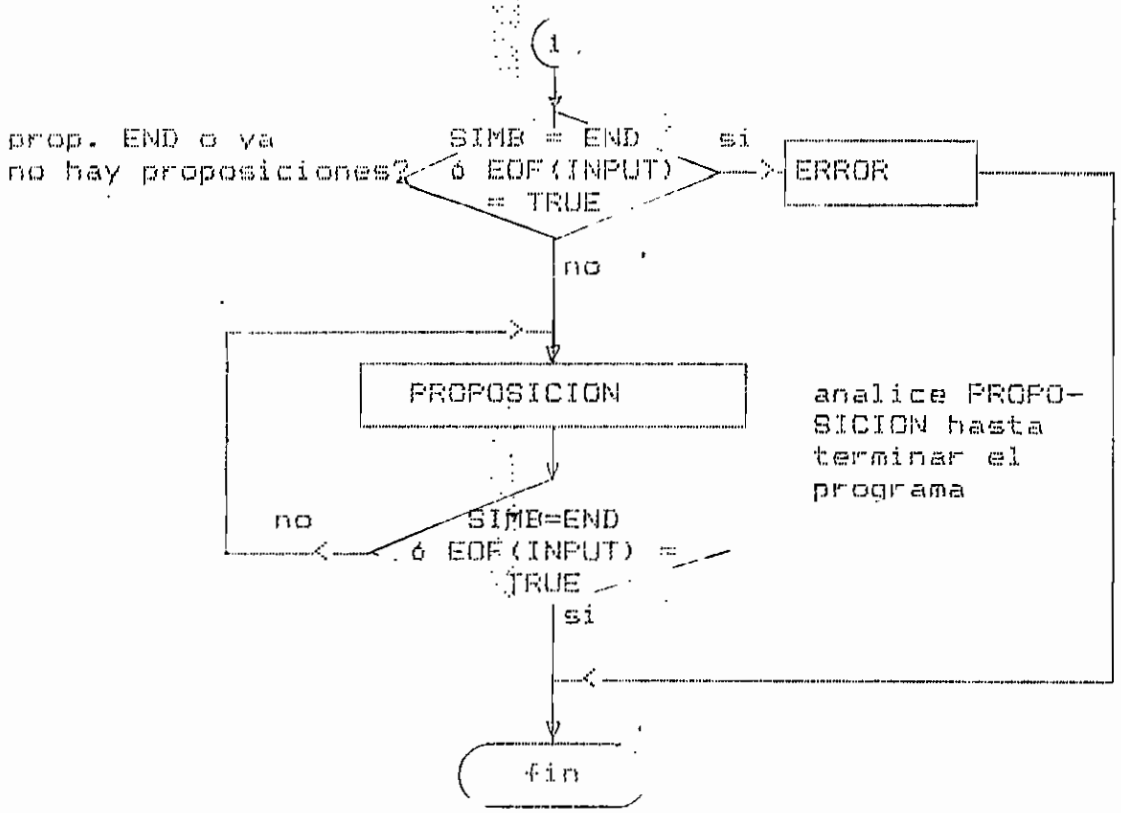


Fig. 4.41.- Diagrama de flujo de la rutina BLOQUE.

4.3.13.2.- Rutina P DATA (interna a ANALISIS).

Analiza la proposición DATA.

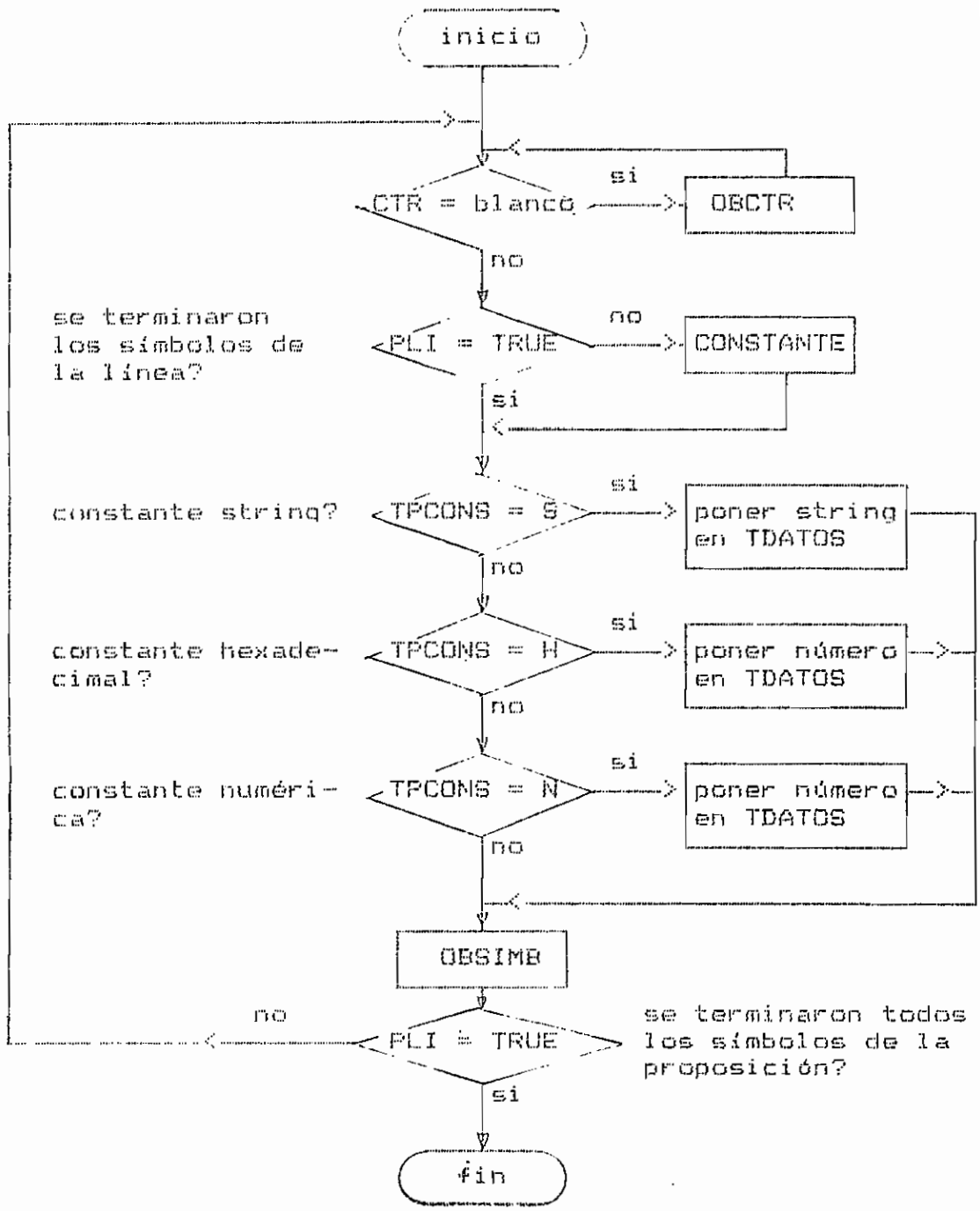


Fig. 4.42.- Diagrama de flujo de la rutina P DATA.

4.3.13.3.- Rutina P_VAR (interna a ANALISIS).

Analiza la proposición VAR.

variables locales:

I = contador

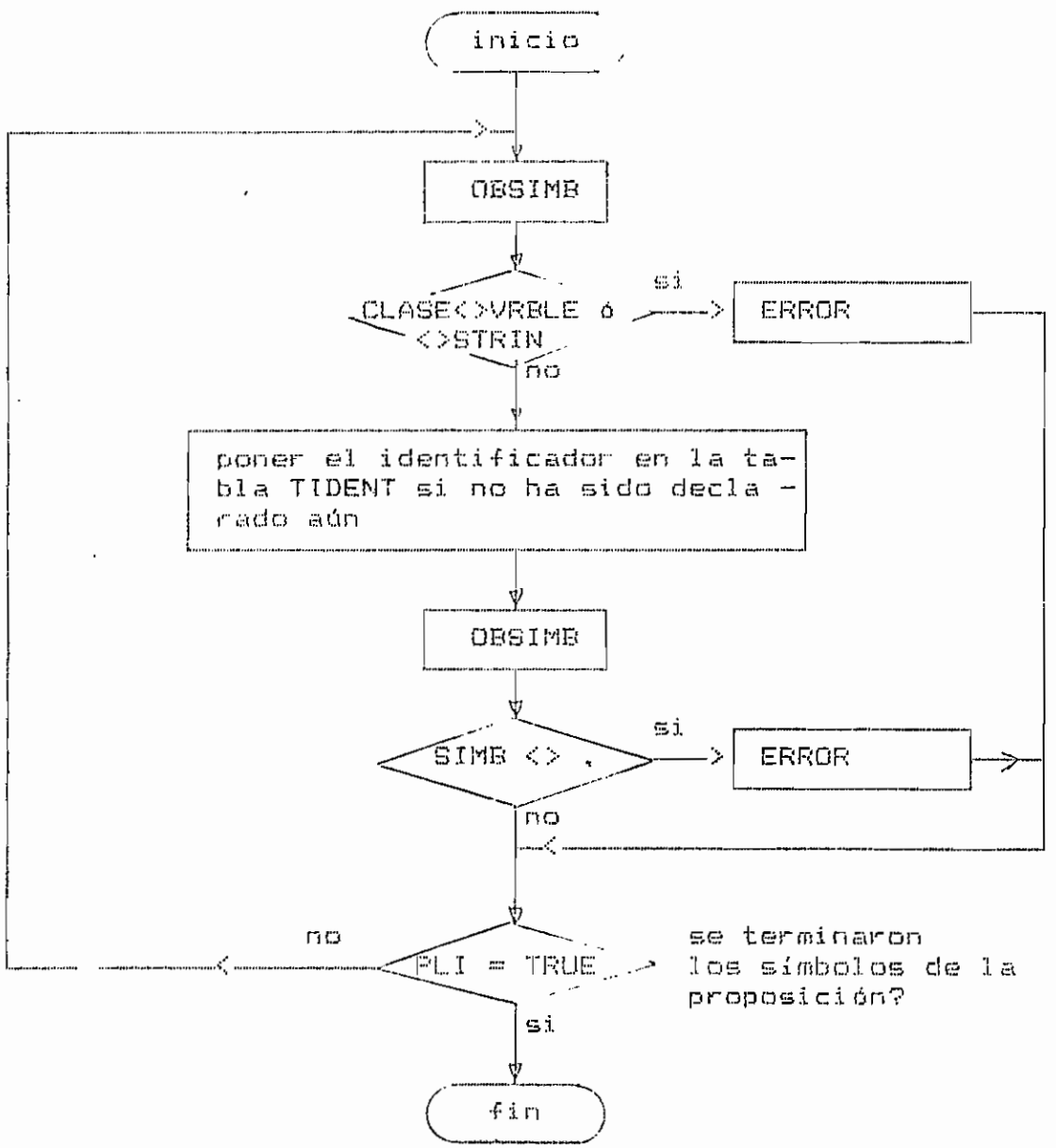


Fig. 4.45.- Diagrama de flujo de la rutina P_VAR.

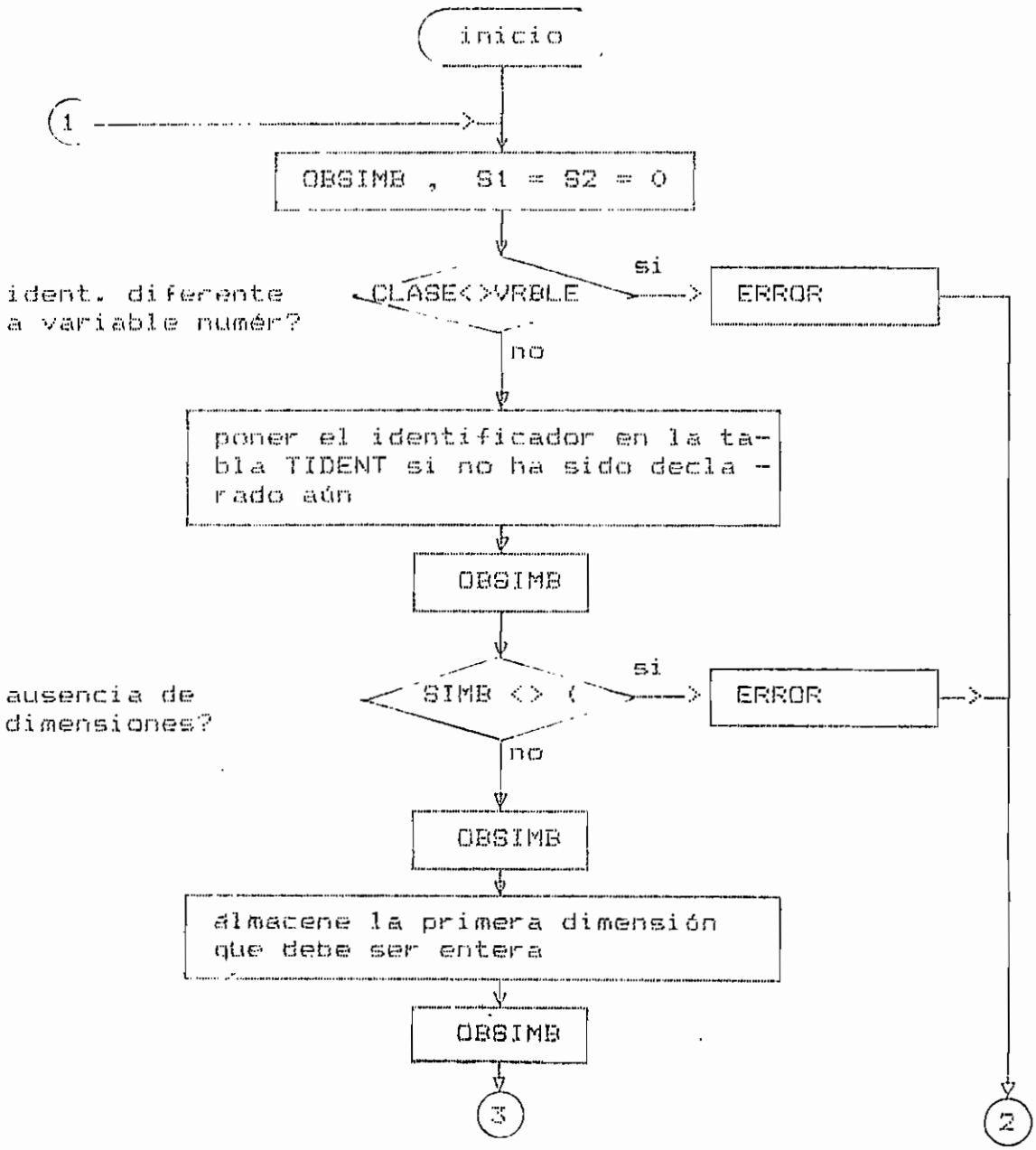
4.3.13.4.- Rutina P_DIM (interna a ANALISIS).

Analiza la proposición DIM.

variables locales:

I = contador

S1, S2 = auxiliares enteros.



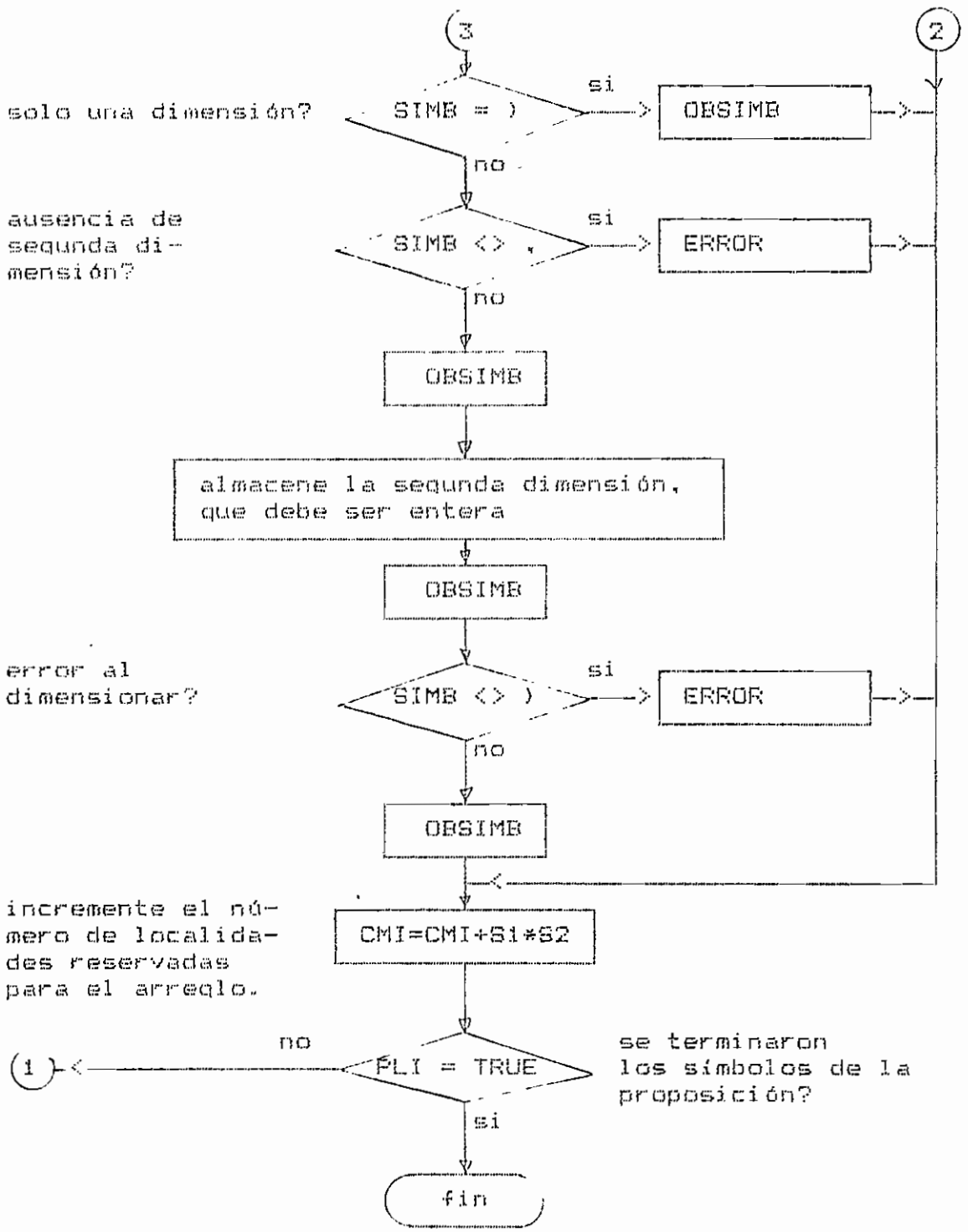


Fig. 4.44.- Diagrama de flujo de la rutina P_DIM.

4.3.13.5.- Rutina PROPOSICION (interna a ANALISIS).

Escoge una de las proposiciones siguientes: FOR, WHILE, CASE, GOTO, GOSUB, RESTORE, LET, o READ.

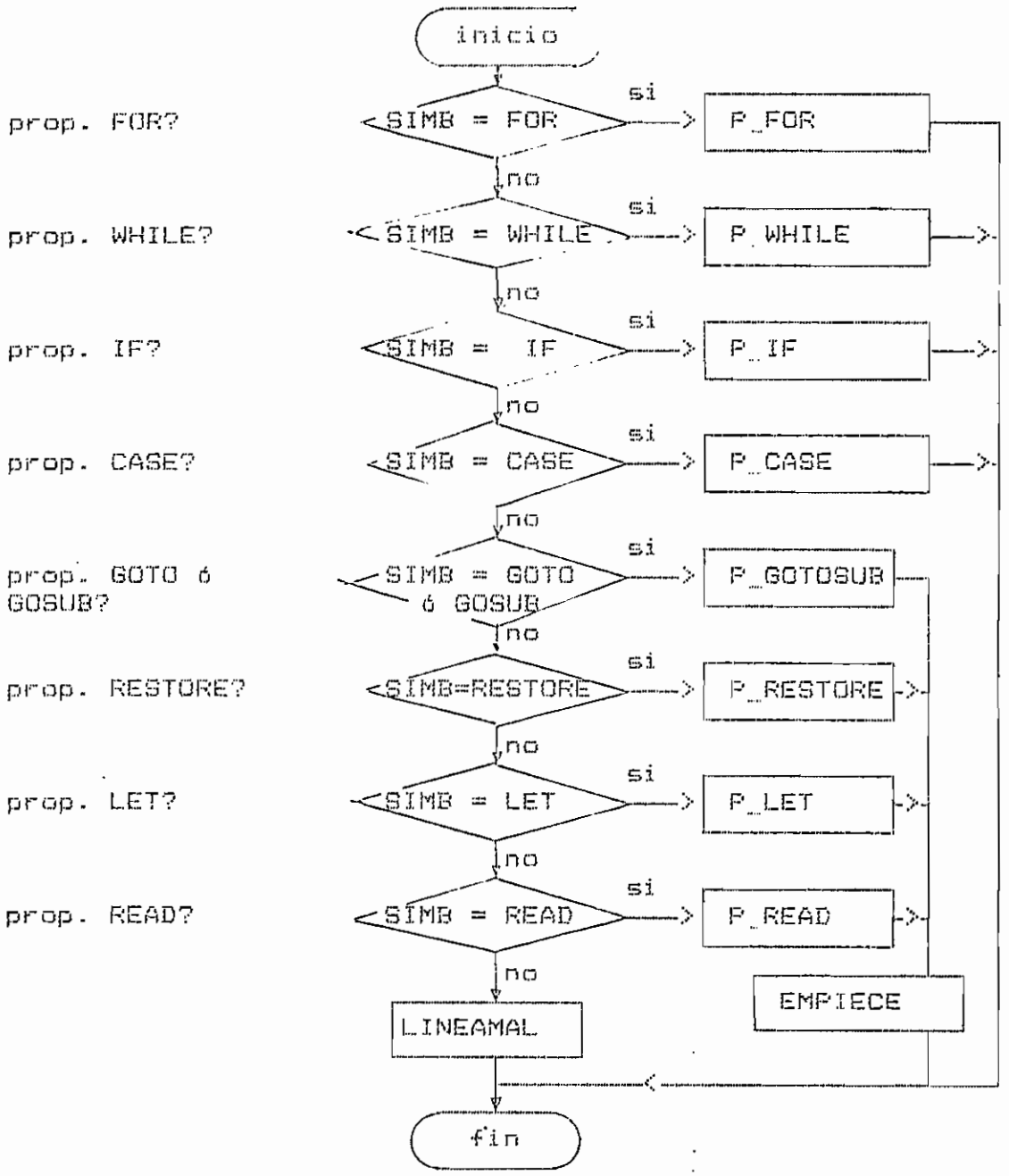


Fig. 4.45.- Diagrama de flujo de la rutina PROPOSICION.

4.3.13.6.- Rutina P_FOR (interna a ANALISIS).

Analiza la proposición estructurada de ciclo FOR.

variables locales:

I = contador.

CCD1, CCD2 = auxiliares para almacenar entero.

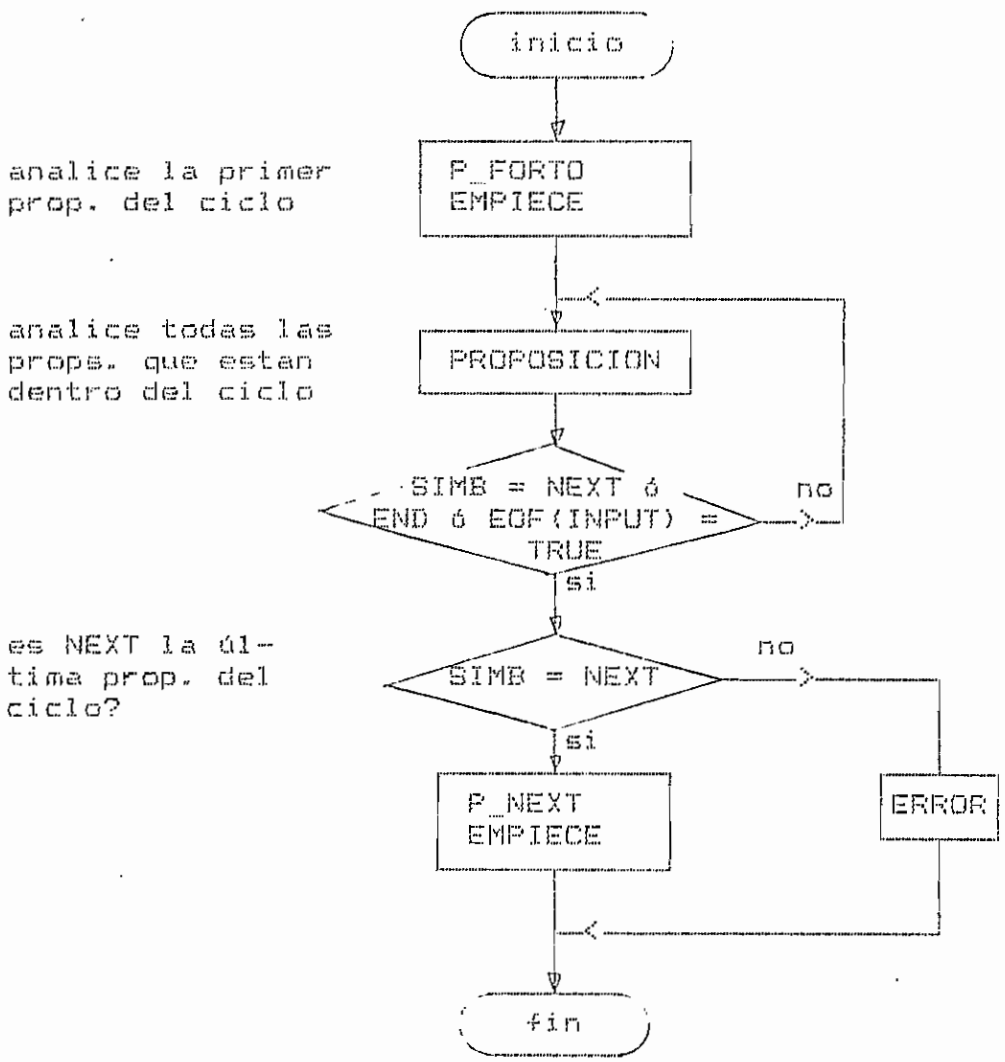


Fig. 4.46.- Diagrama de flujo de la rutina P_FOR.

4.3.i3.6.i.- Rutina P_FORTO (interna a P_FOR).

Analiza la primer proposición de un ciclo FOR.

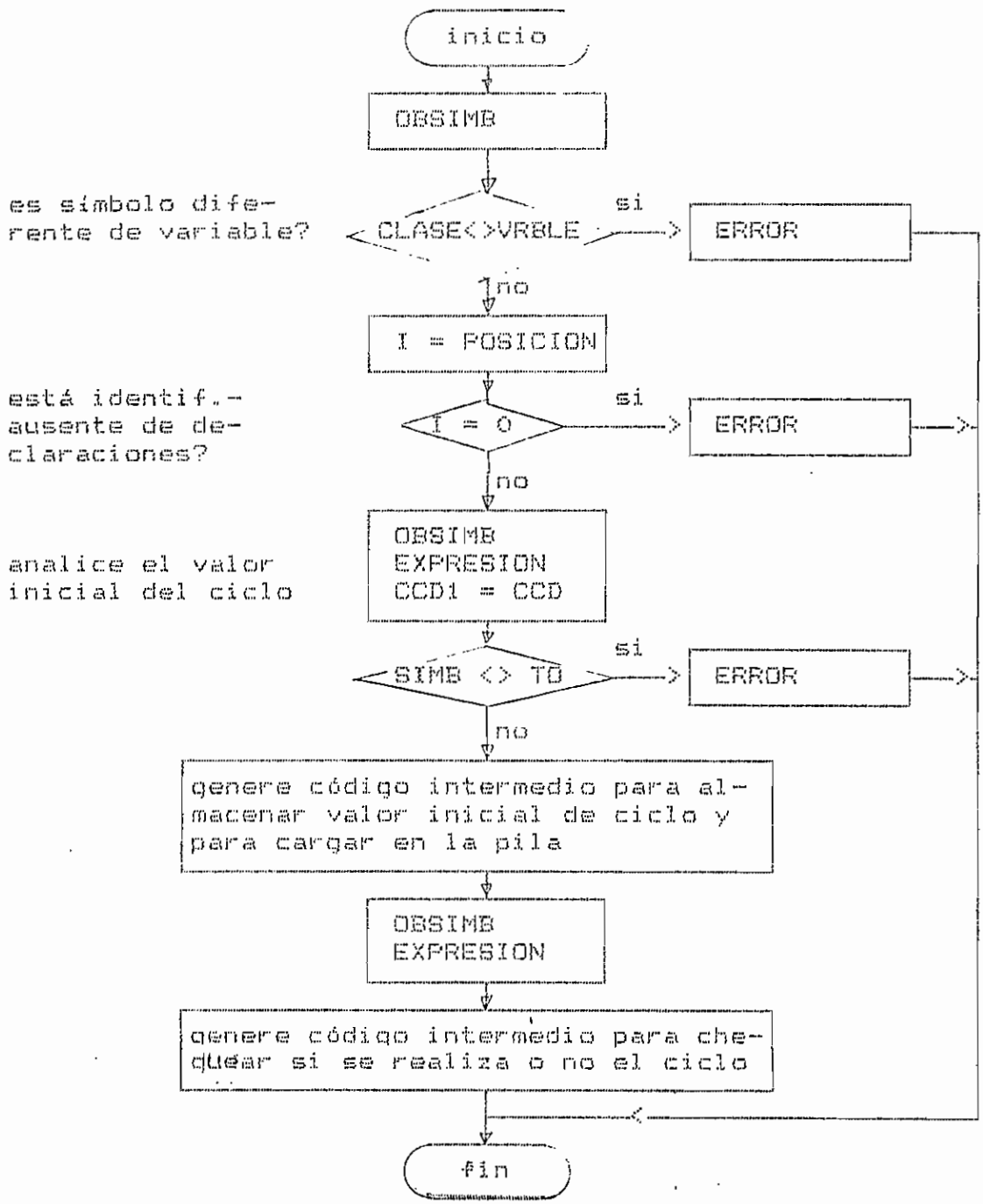


Fig. 4.47.- Diagrama de flujo de la rutina P_FORTO.

4.3.13.6.2.- Rutina P_NEXT (interna a P_FOR).

Analiza la última proposición de un ciclo FOR.

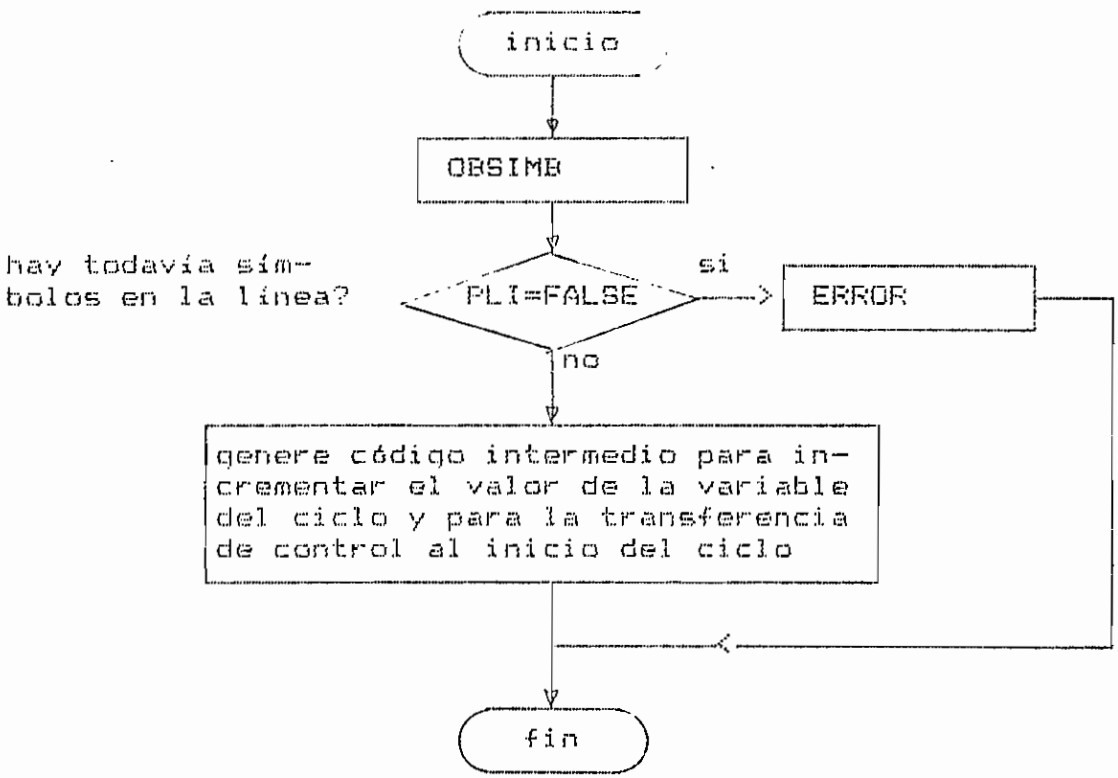


Fig. 4.48.- Diagrama de flujo de la rutina P_NEXT.

4.3.13.7.- Rutina P_WHILE (interna a ANALISIS).

Analiza la proposición estructurada de ciclo WHILE.

variables locales:

CCD1, CCD2 = auxiliares para almacenar entero.

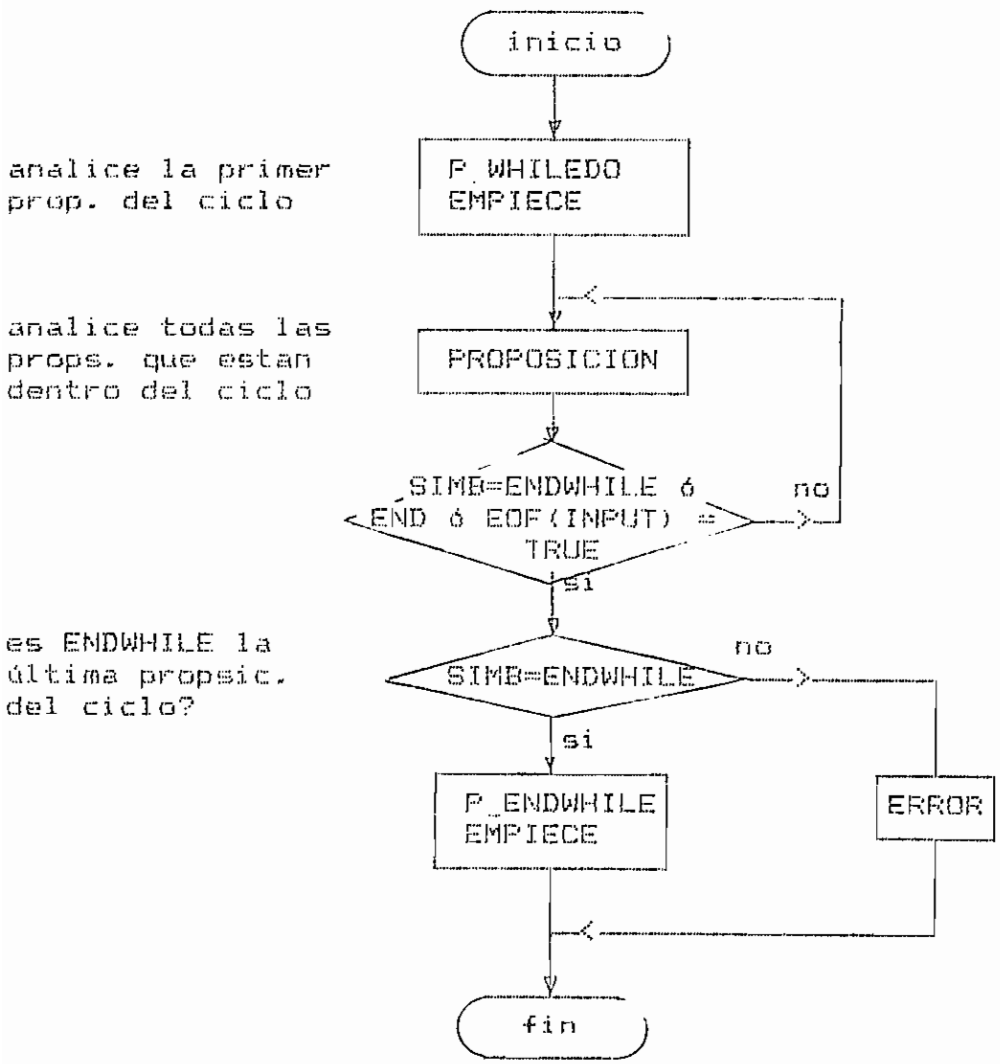


Fig. 4.49.- Diagrama de flujo de la rutina P_WHILE.

4.3.13.7.1.- Rutina P_WHILEDO (interna a P_WHILE).

Analiza la primer proposición de un ciclo WHILE.

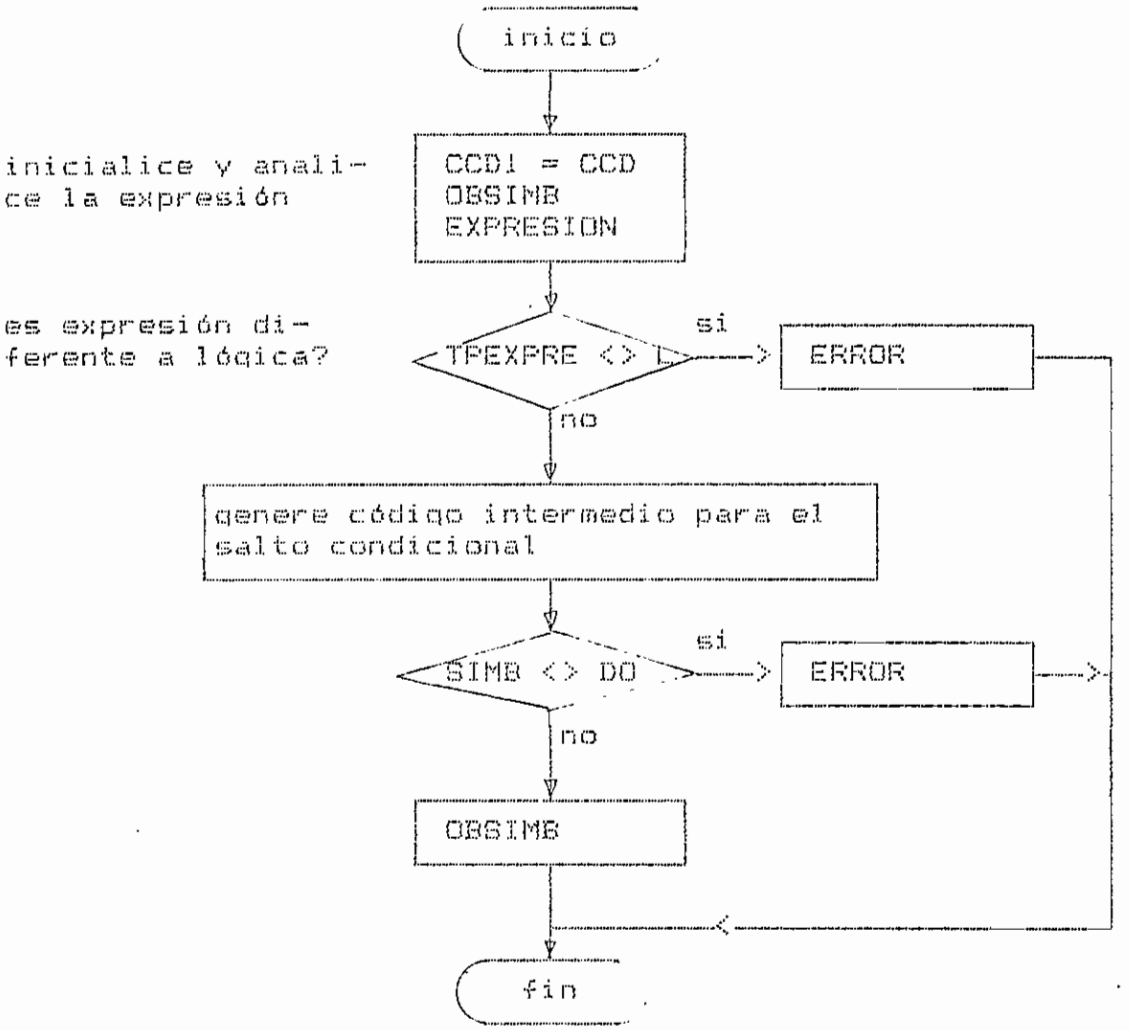


Fig. 4.50.- Diagrama de flujo de la rutina P_WHILEDO.

4.3.13.7.2.- Rutina P_ENDWHILE (interna a P_WHILE).

Analiza la última proposición de un ciclo WHILE.

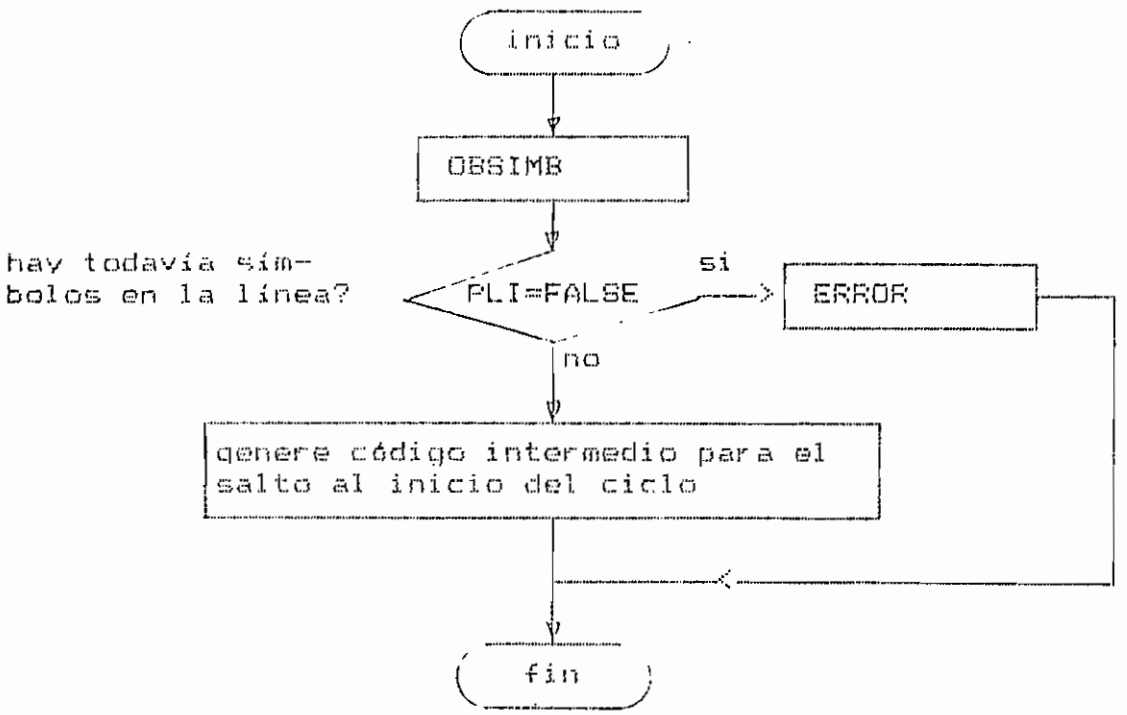


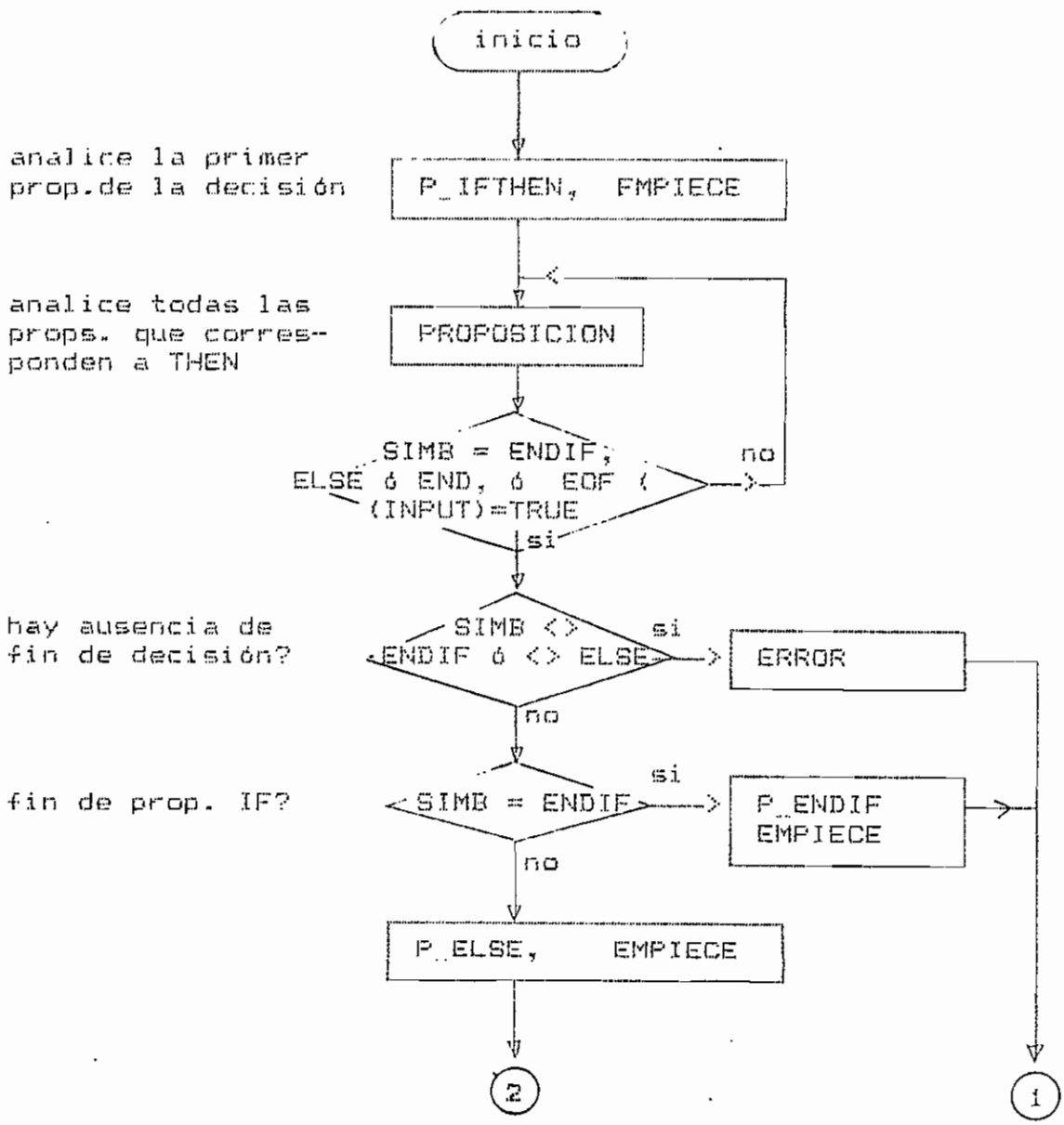
Fig. 4.51.- Diagrama de flujo de la rutina P_ENDWHILE.

4.3.13.8.- Rutina P_IF (interna a ANALISIS).

Analiza la proposición estructurada de decisión IF.

variables locales:

CCD1, CCD2 = auxiliares para almacenar entero.



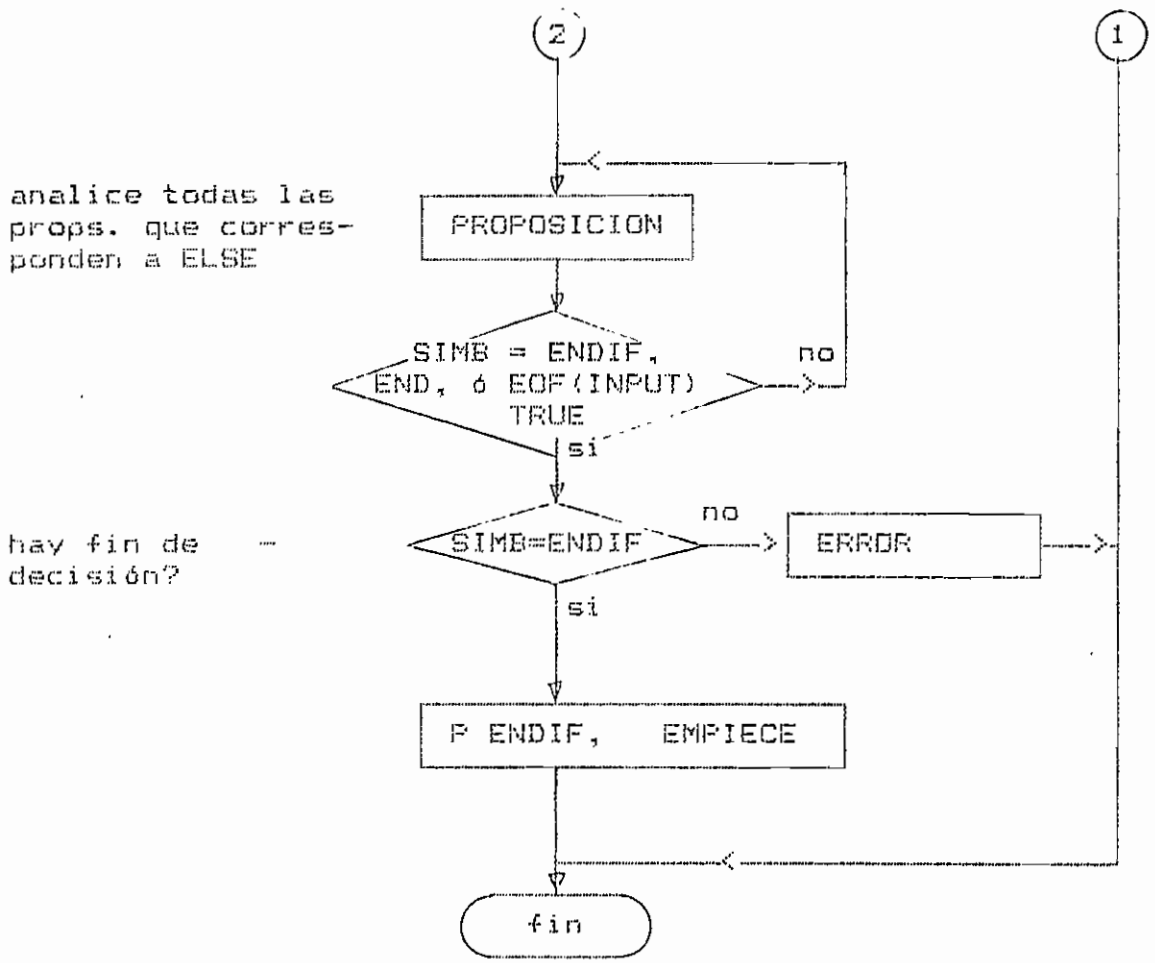


Fig. 4.52.- Diagrama de flujo de la rutina P_IF.

4.3.13.8.i.- Rutina P_IFTHEN (interna a P_IF).

Analiza la primer proposición de la decisión IF.

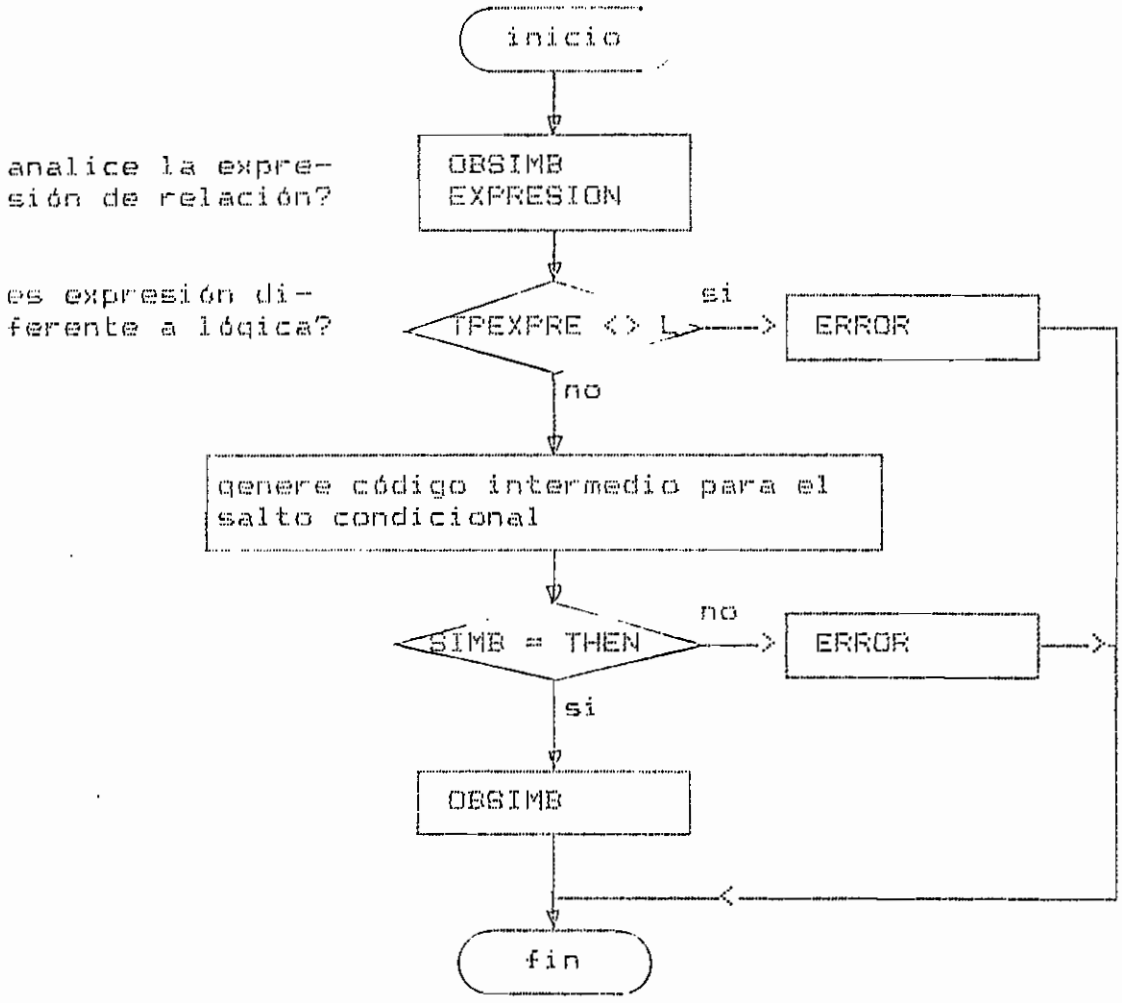


Fig. 4.53.- Diagrama de flujo de la rutina P_IFTHEN.

4.3.13.8.2.- Rutina P_ELSE (interna a P_IF).

Analiza la el fin de las proposiciónes que forman la parte THEN de una decisión y el inicio de las que forman la parte ELSE.

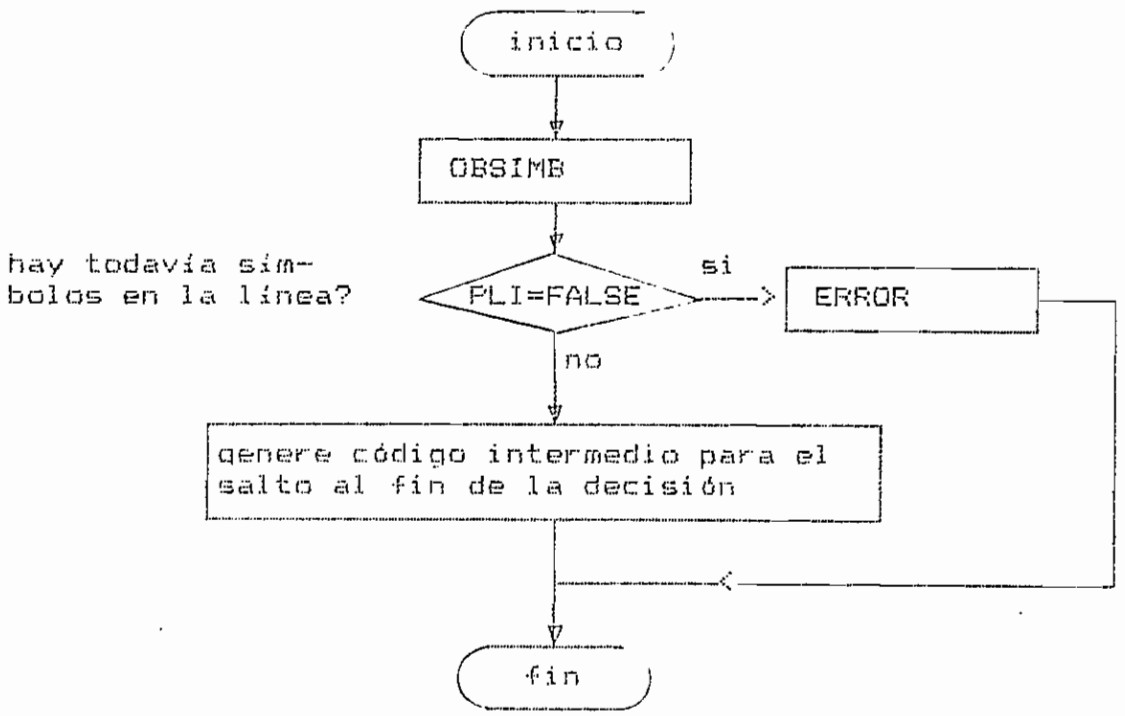


Fig. 4.54.- Diagrama de flujo de la rutina P_ELSE.

4.3.13.8.3.- Rutina P_ENDIF (interna a P_IF).

Analiza el fin de una decisión.

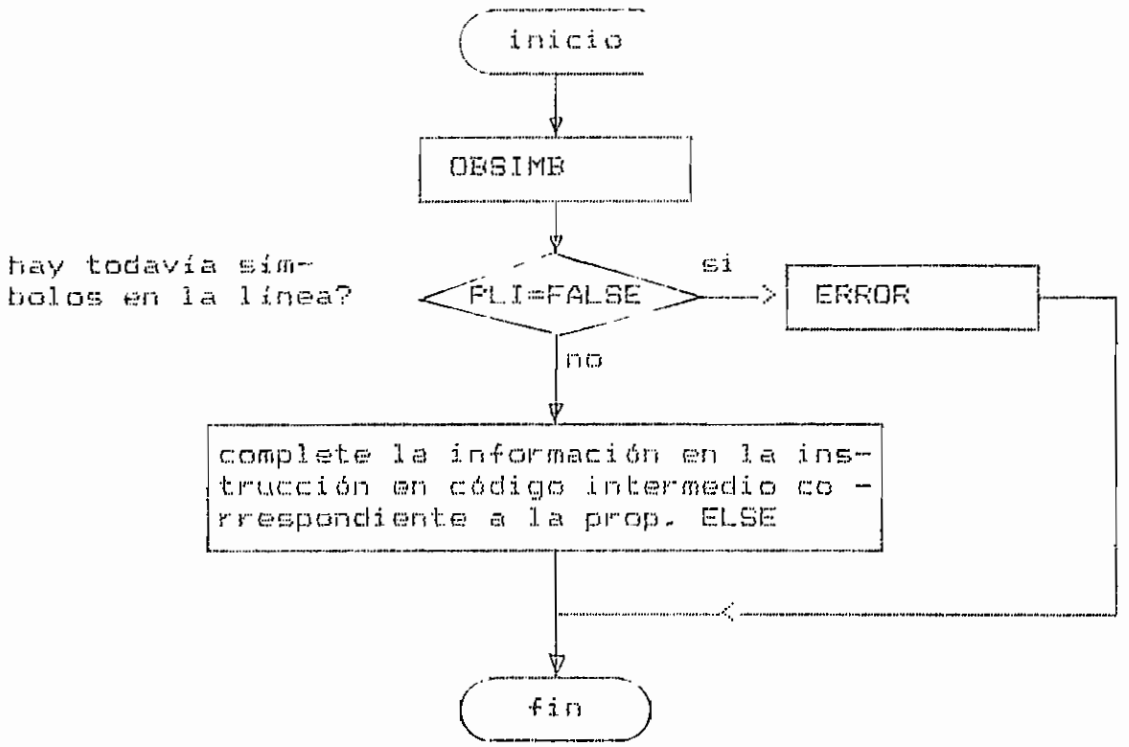


Fig. 4.55.- Diagrama de flujo de la rutina P_ENDIF.

4.3.13.9.- Rutina P_CASE (interna a ANALISIS).

Analiza la proposición de decisión múltiple CASE.

variables locales:

I = contador.

CCD1 = auxiliar para almacenar entero.

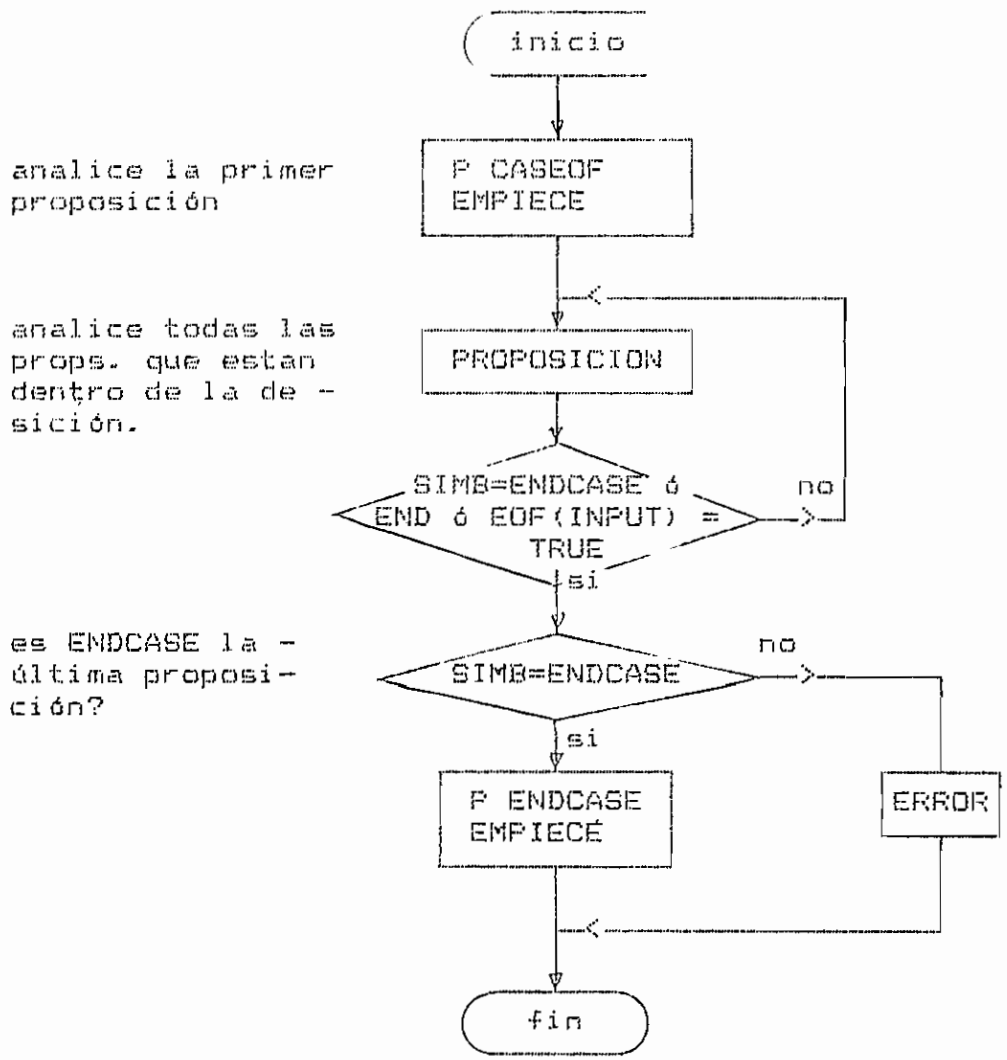


Fig. 4.56.- Diagrama de flujo de la rutina P_CASE.

4.3.13.9.1.- Rutina P_CASEOF (interna a P_CASE).

Analiza la primer proposición de una decisión múltiple CASE.

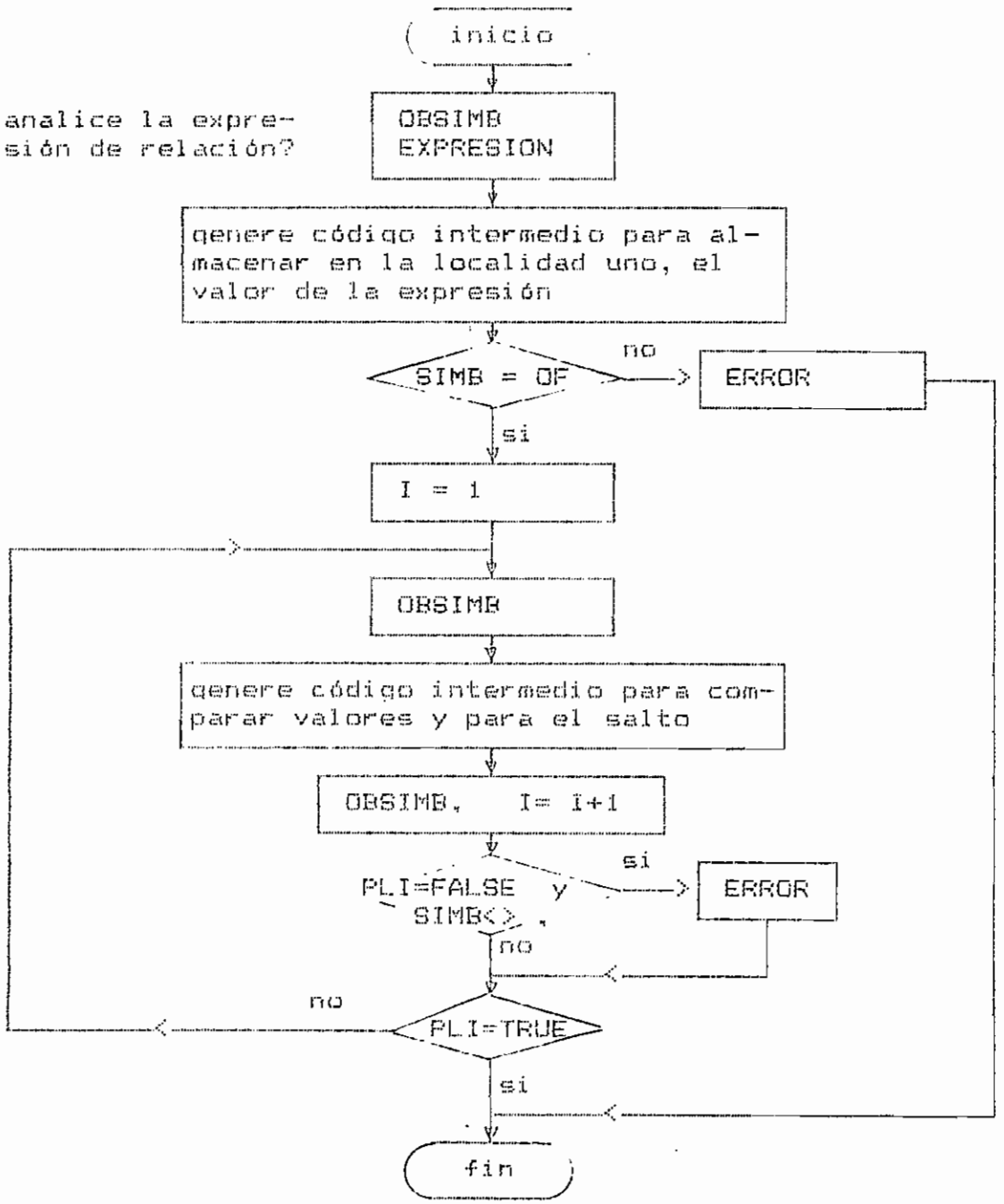


Fig. 4.57.- Diagrama de flujo de la rutina P_CASEOF.

4.3.13.10.- Rutina P_GOTOSUB (interna a ANALISIS).

Analiza la proposición GOTO y la GOSUB.

variables locales:

PR = auxiliar para almacenar un símbolo.

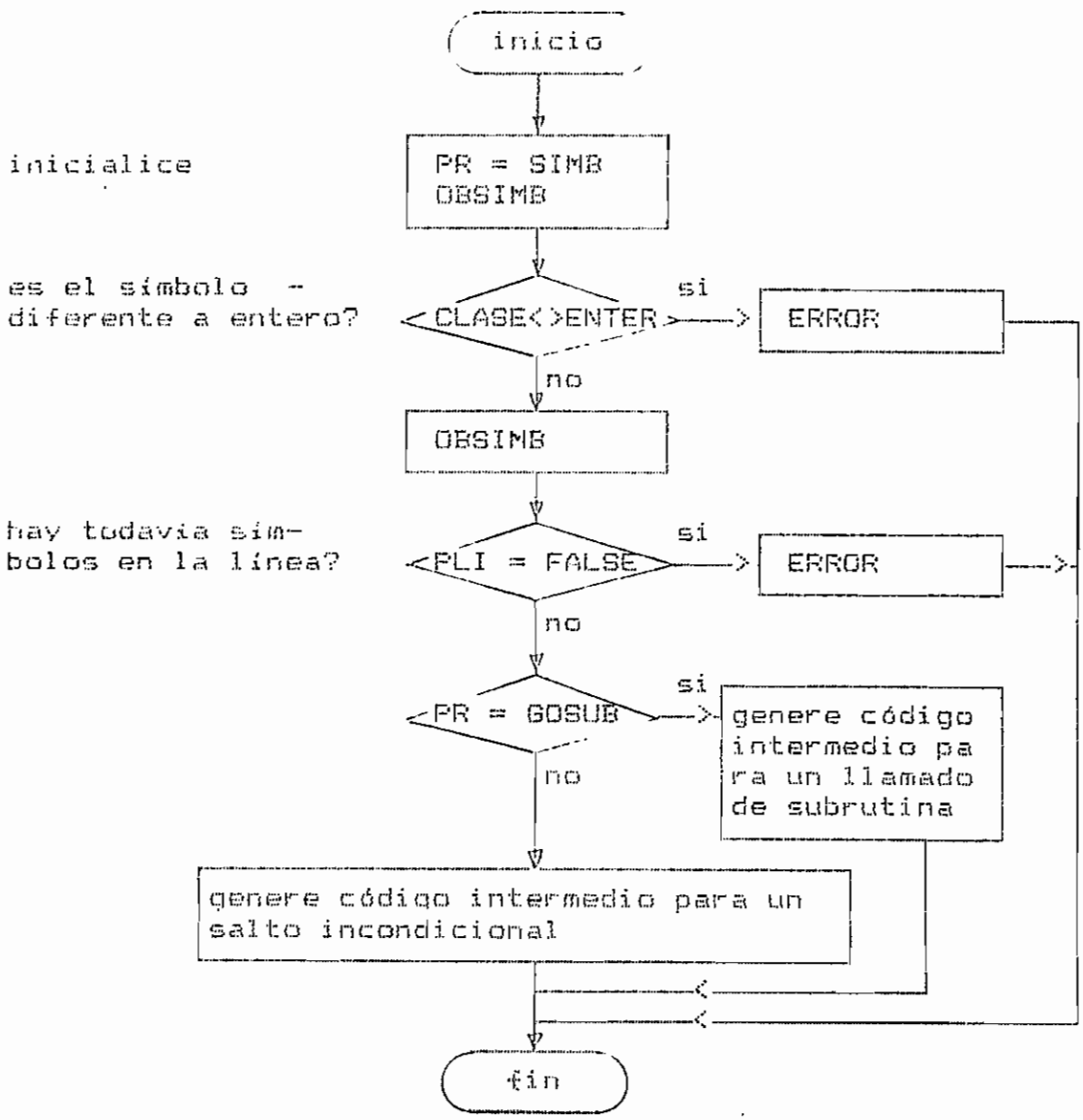


Fig. 4.59.- Diagrama de flujo de la rutina P_GOTOSUB.

4.3.13.11.- Rutina P_RESTORE (interna a ANALISIS).

Analiza la proposición RESTORE.

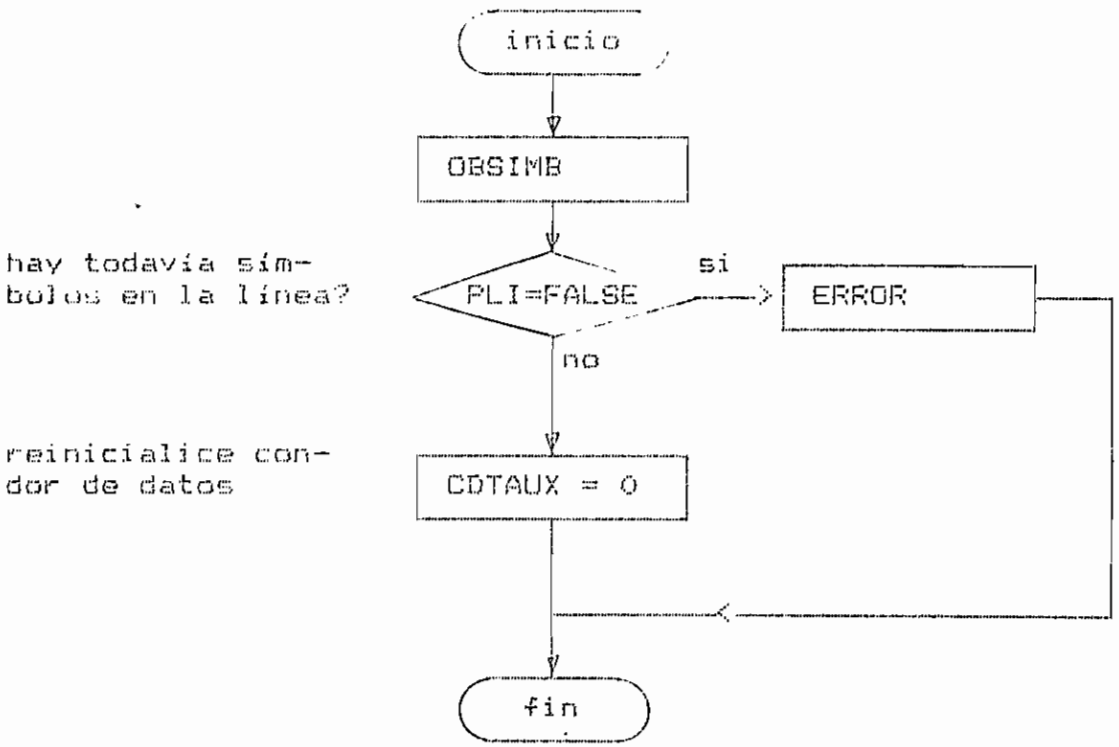


Fig. 4.58.- Diagrama de flujo de la rutina P_ENDCASE.

4.3.13.12.- Rutina P_LET (interna a ANALISIS).

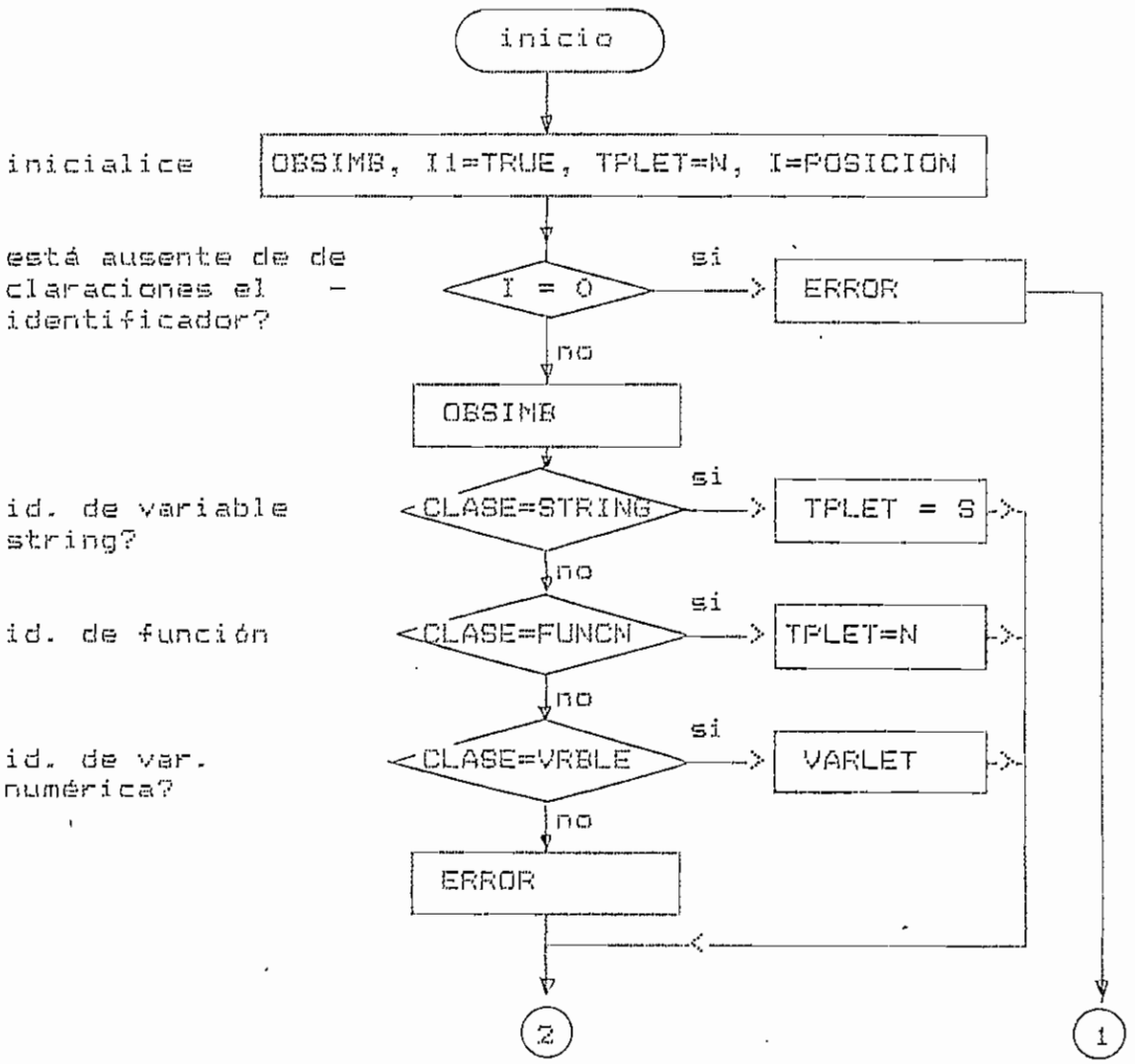
Analiza la proposición de asignación.

variables locales:

I = auxiliar para almacenar un entero.

I1 = auxiliar Booleano.

TP = auxiliar que almacena el tipo de variable a la que se asignará un valor: string(S) ó numérico(N).



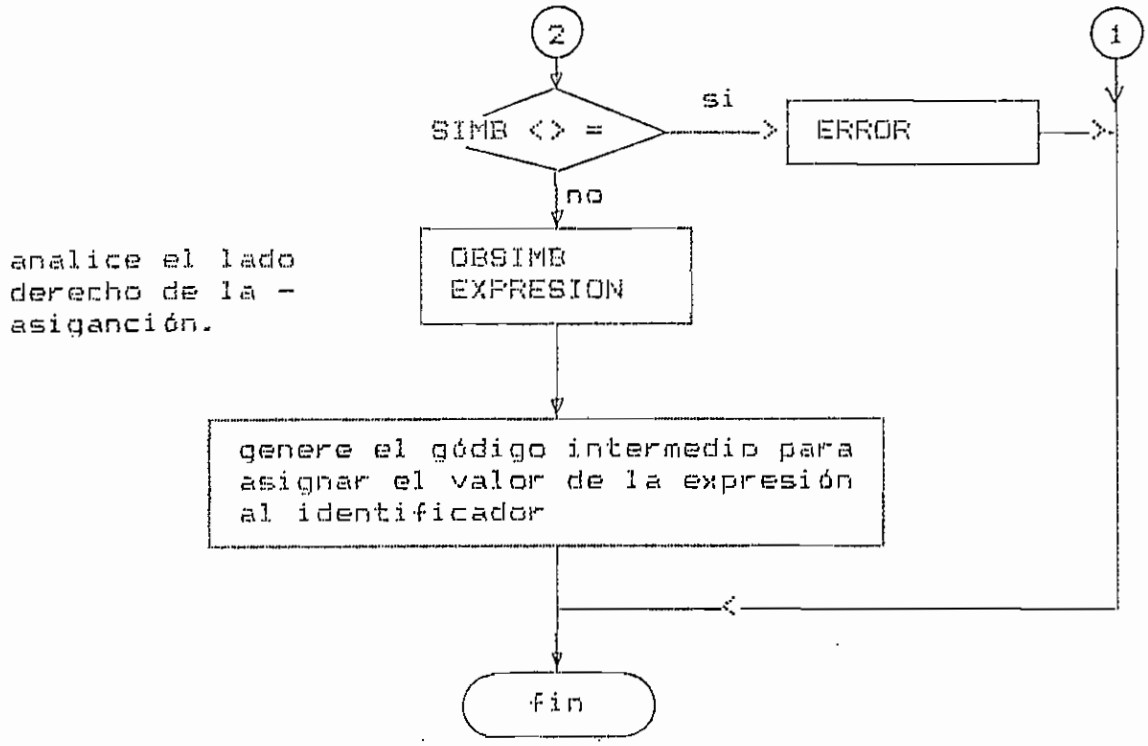


Fig. 4.61.- Diagrama de flujo de la rutina P_LET.

4.3.13.12.1.- Rutina VARLET (interna a P_LET).

Analiza y genera código para un identificador de variable numérica simple o de arreglo al que se le hará una asignación.

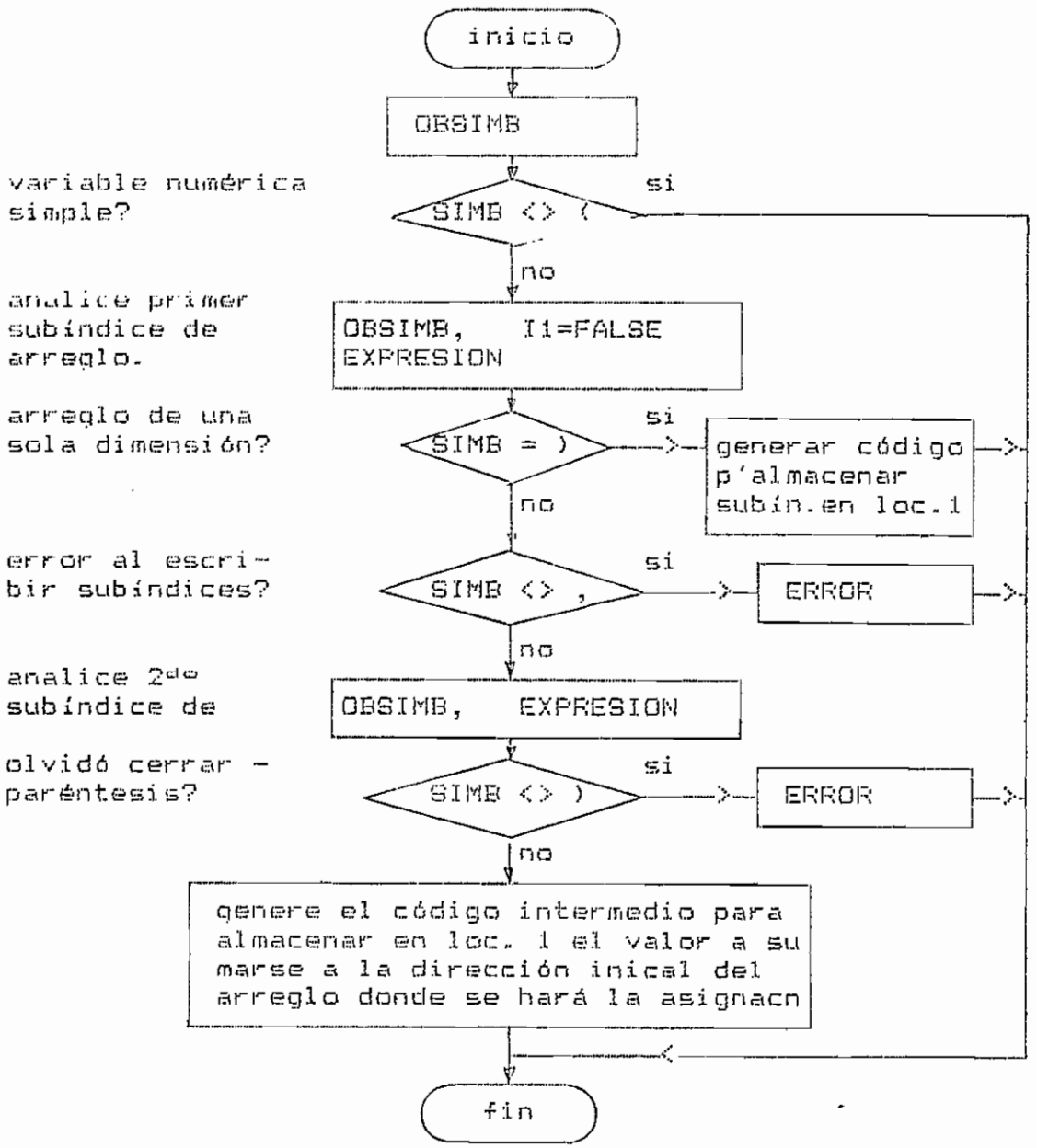


Fig. 4.62.- Diagrama de flujo de la rutina VARLET.

4.3.13.13.- Rutina P_READ (interna a ANALISIS).

Analiza la proposición READ.

variables locales:

I1 = auxiliar para almacenar un entero.

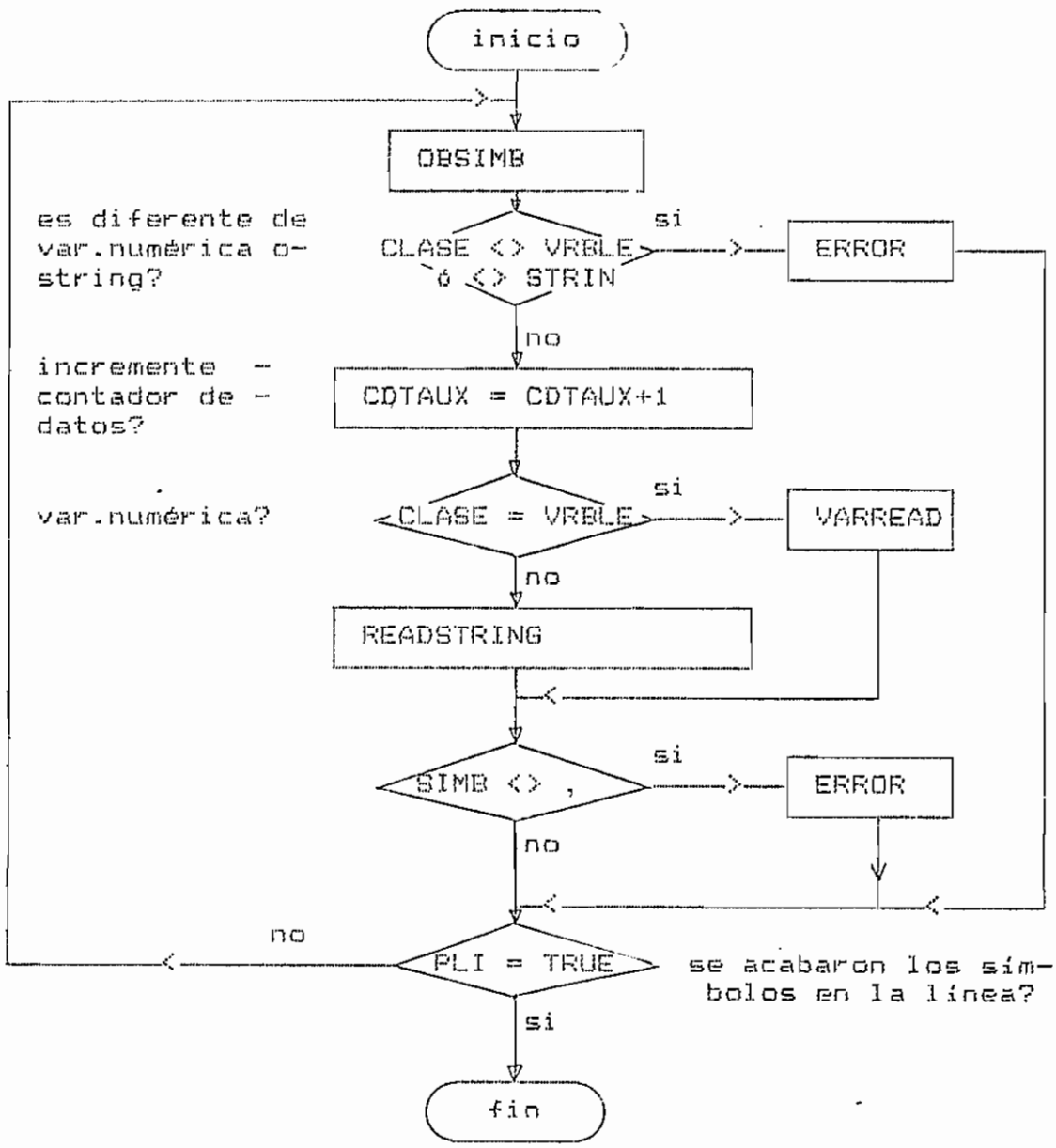
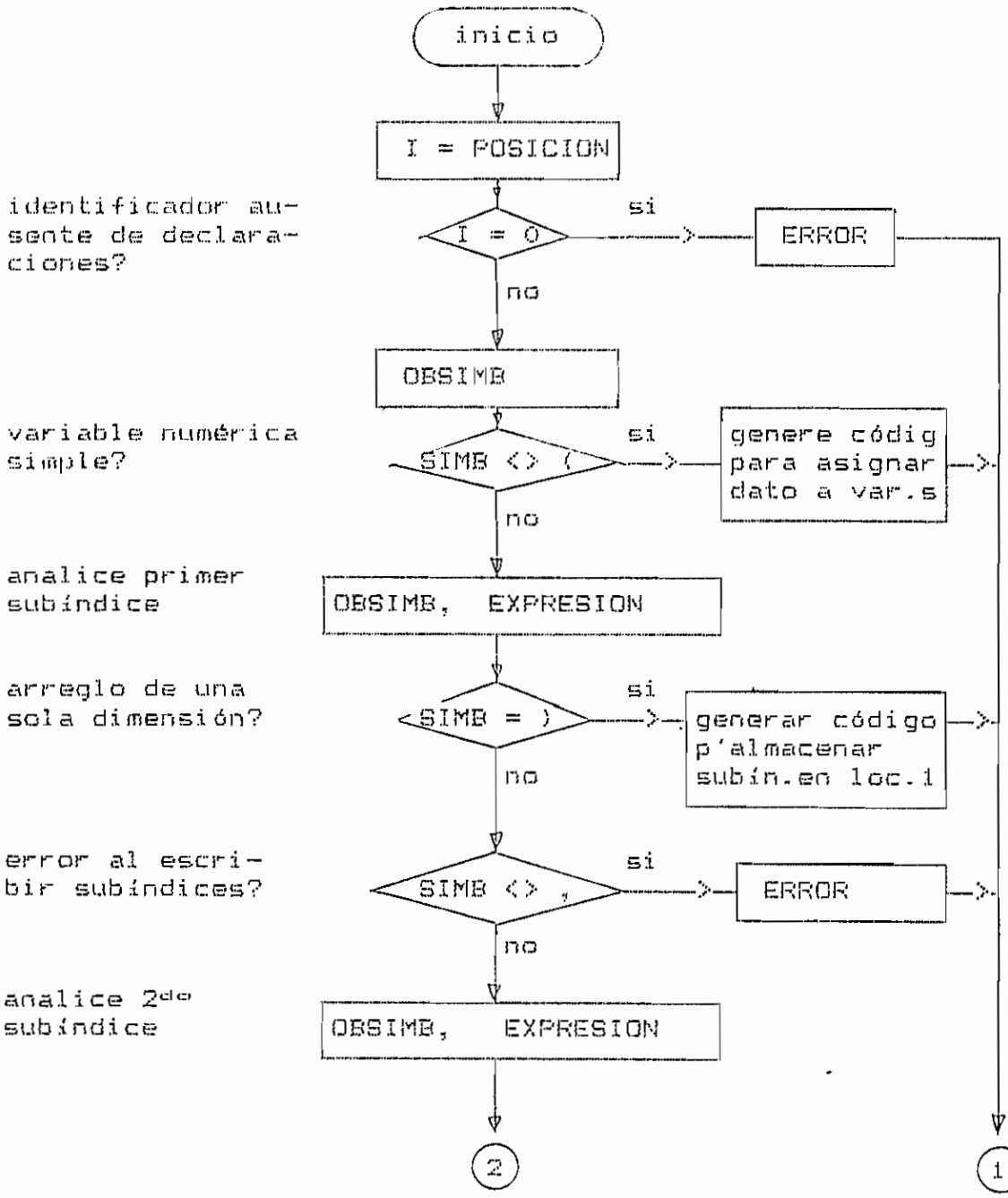


Fig. 4.63.- Diagrama de flujo de la rutina P_READ.

4.3.13.13.i.- Rutina VARREAD (interna a P_READ).

Analiza y genera código para asiganar un dato a una variable numérica simple o un arreglo.



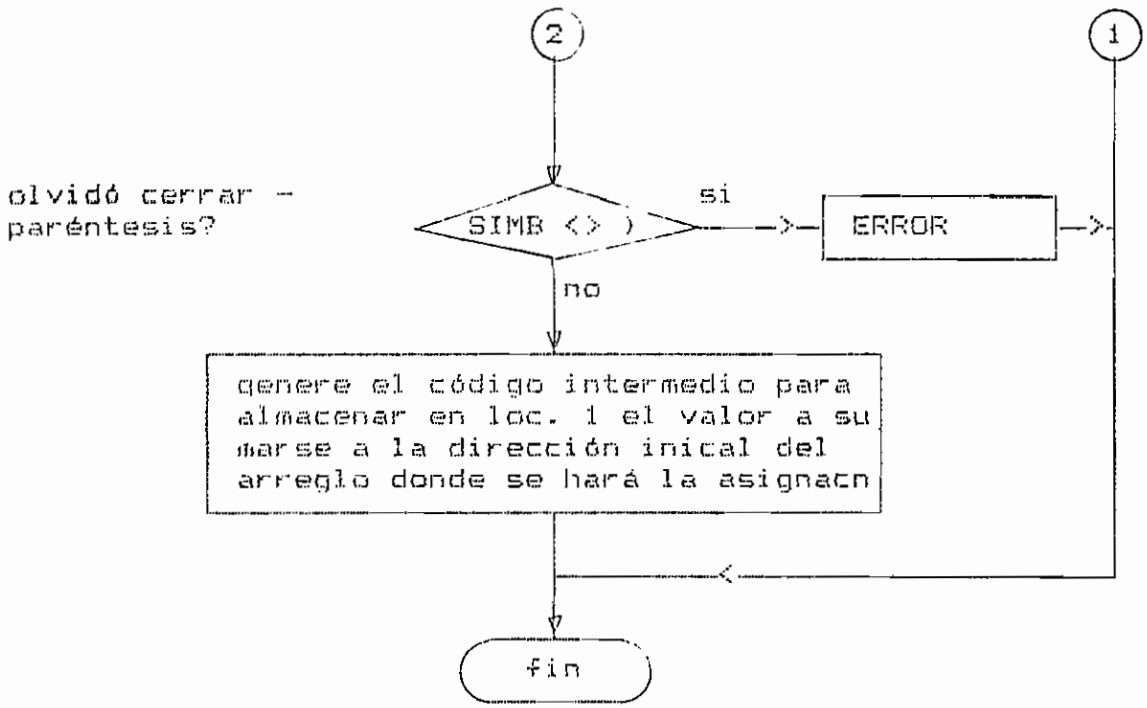


Fig. 4.64.- Diagrama de flujo de la rutina VARREAD.

4.3.13.13.2.- Rutina READSTRING (interna a P_READ).

Analiza y genera código para leer un dato string y asignarlo a una variable.

variables locales:

N1 = contador.

I = auxiliar entero.

CTRAUX = auxiliar para almacenar un caracter.

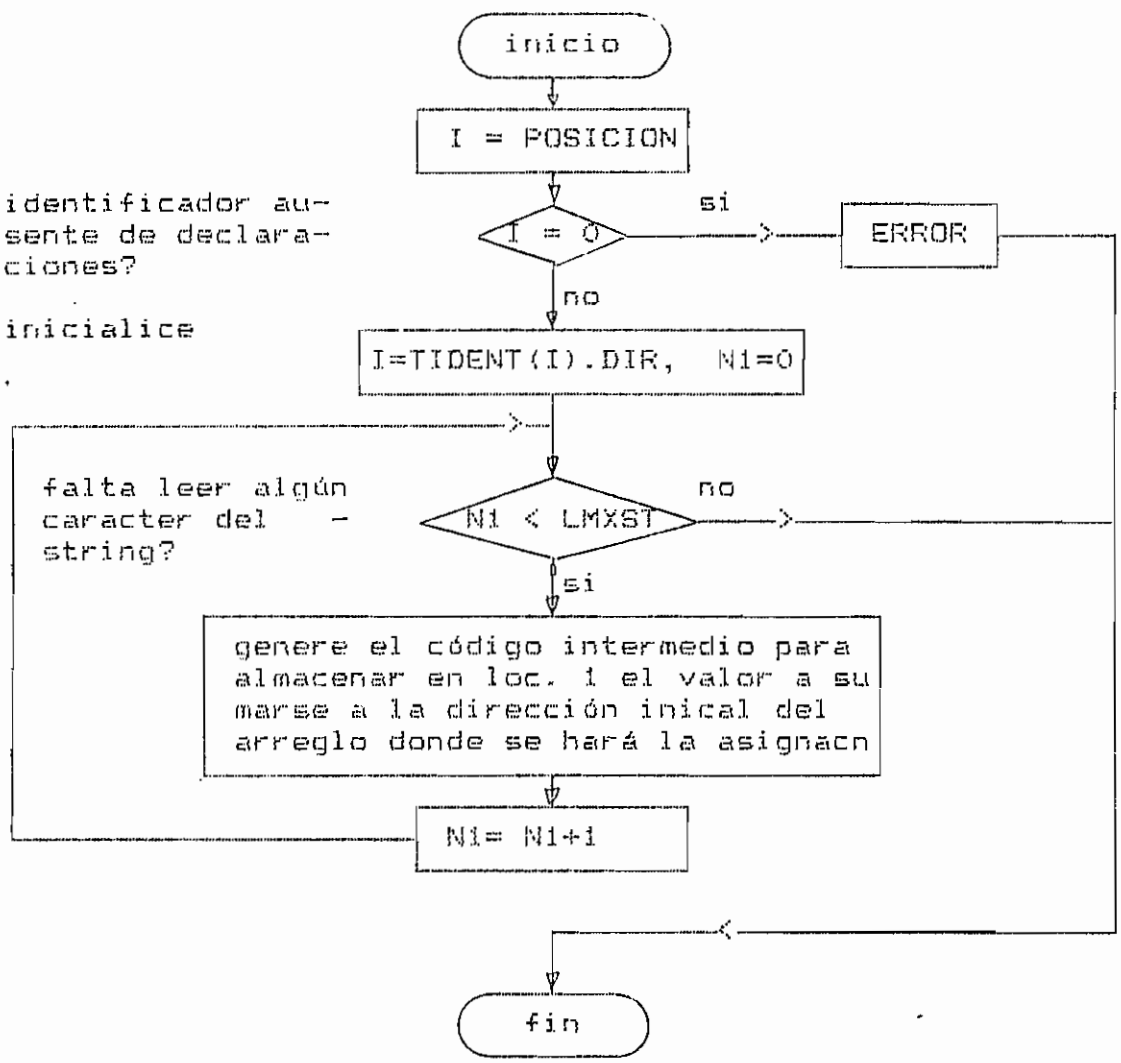


Fig. 4.65.- Diagrama de flujo de la rutina READSTRING.

4.3.13.14.- Rutina LLENARGT (interna a ANALISIS).

Completa la información en instrucciones en código intermedio correspondientes a transferencias incondicionales de control en proposiciones del programa fuente.

variables locales:

I1,I2 = contador.

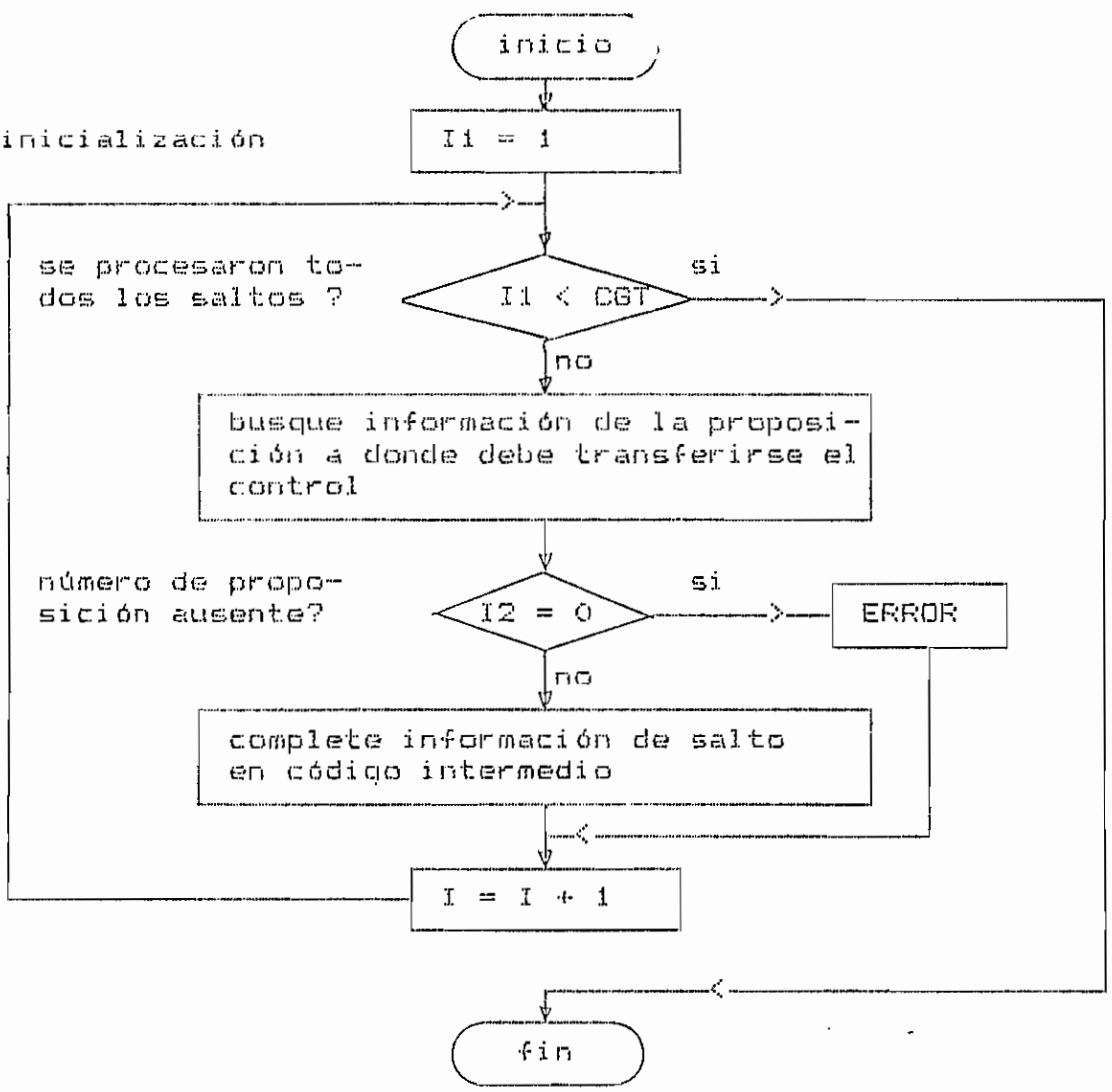


Fig. 4.66.- Diagrama de flujo de la rutina LLENARGT.

4.3.13.15.- Rutina PONERNP (interna a ANALISIS).

Almacena información en tabla TNUPRO.

parámetros:

NP = corresponde al número de prop. en lenguaje fuente.

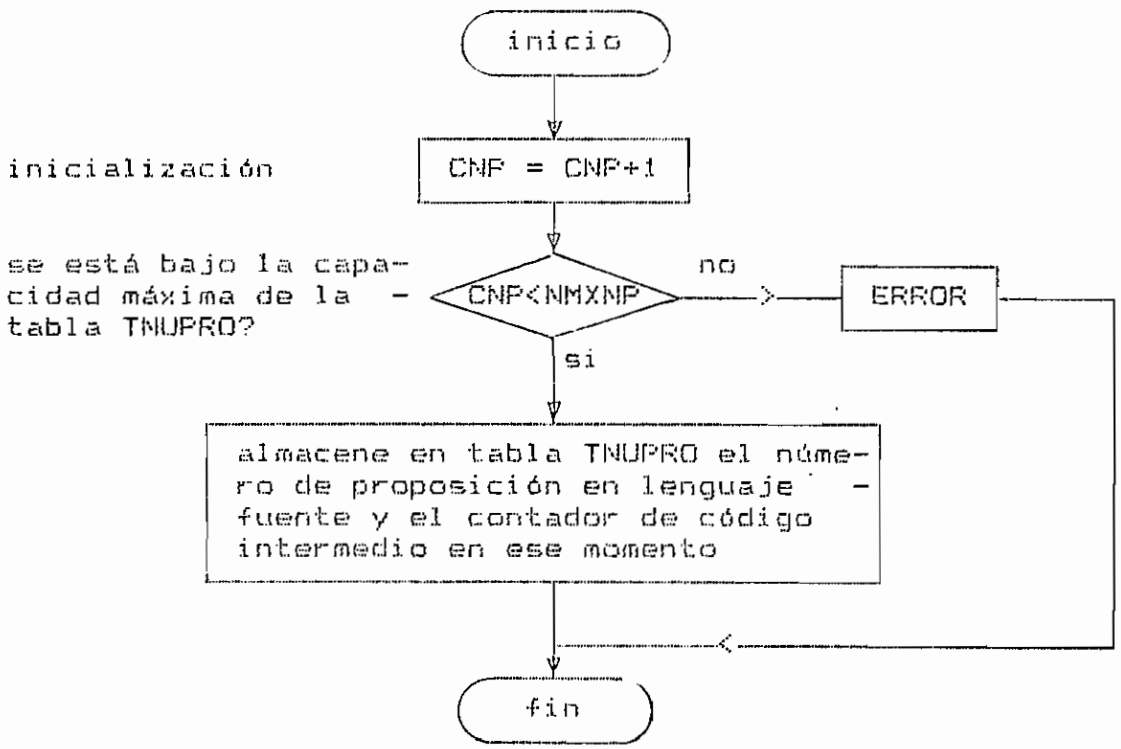


Fig. 4.67.- Diagrama de flujo de la rutina PONERNP.

4.3.13.16.- Rutina PONERGT (interna a ANALISIS).

Almacena información en tabla TGOTOS.

parámetros:

GT = número de proposición, en lenguaje fuente, que corresponde a una transferencia incondicional de control.

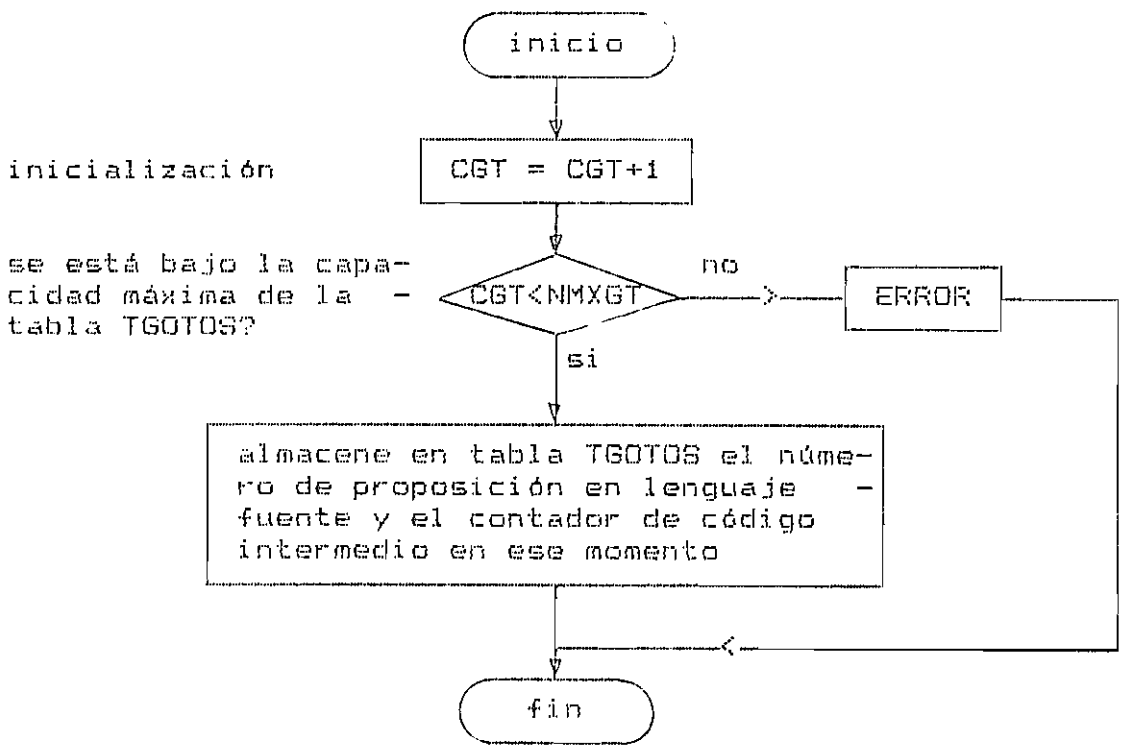


Fig. 4.68.- Diagrama de flujo de la rutina PONERGT.

4.3.13.17.- Rutina PONERID (interna a ANALISIS).

Almacena información en tabla TIDENT.

parámetros:

TP = indica el tipo de identificador: variable numérica simple (V), variable string (S), arreglo (A) o función (F).

variables locales:

I = contador.

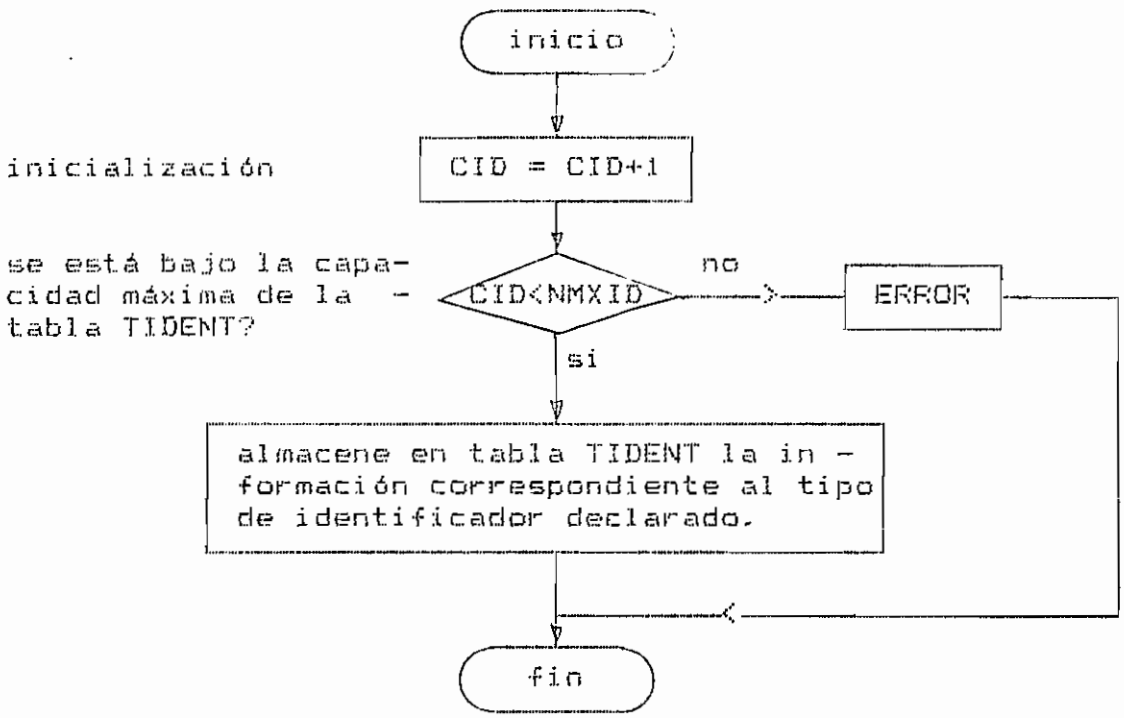


Fig. 4.69.- Diagrama de flujo de la rutina PONERID.

4.3.13.18.- Rutina PONERDT (interna a ANALISIS).

Almacena información en tabla TDATOS.

parámetros:

TD1= indica el tipo de dato: decimal (N), hexadecimal (H), string (S).

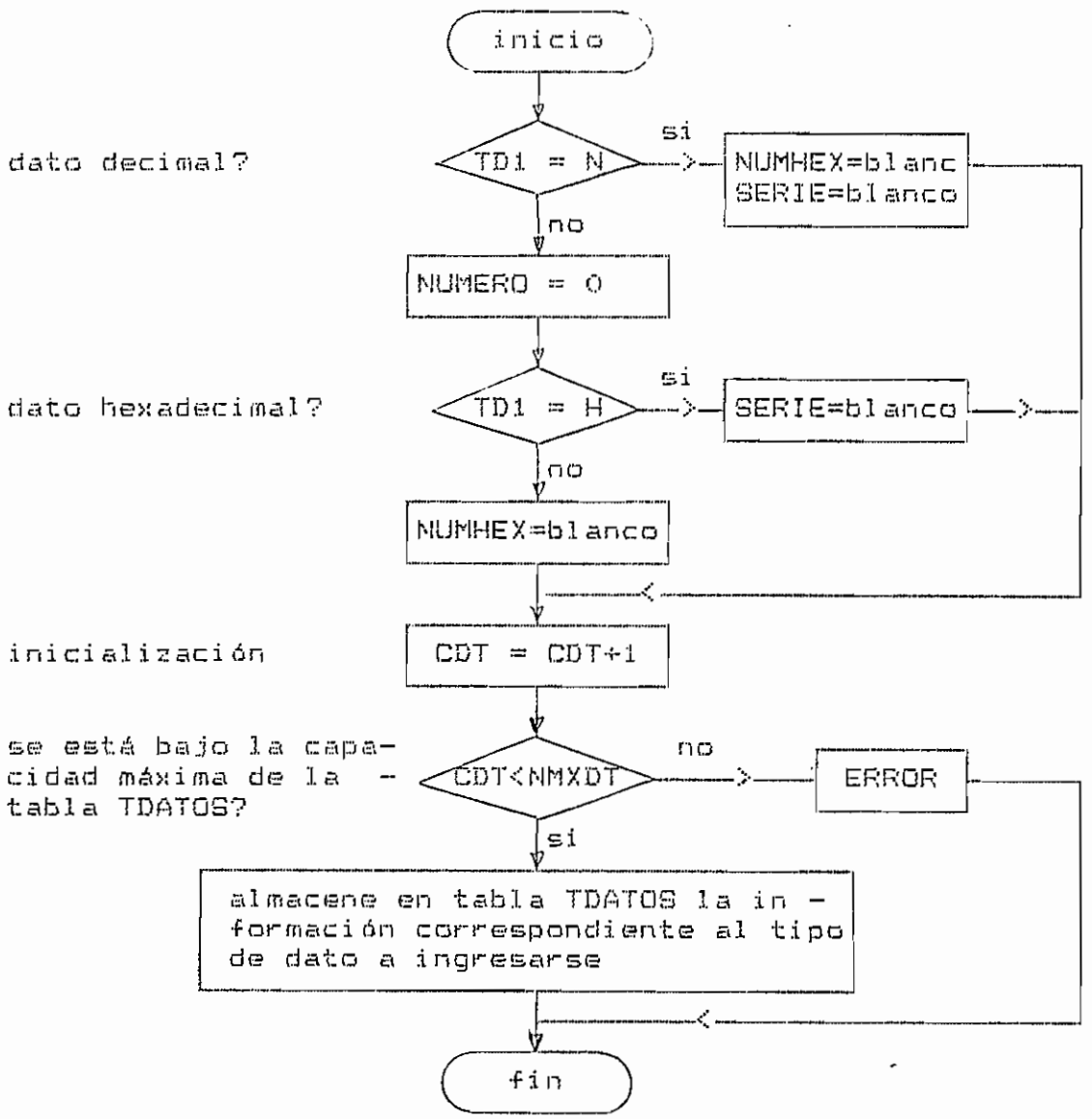


Fig. 4.70.- Diagrama de flujo de la rutina PONERDT.

4.3.13.19.- Rutina EMPIECE (interna a ANALISIS).

Almacena el inicio de una proposición en lenguaje fuente hasta encontrar una palabra reservada, y separa las proposiciones que corresponden a comentarios.

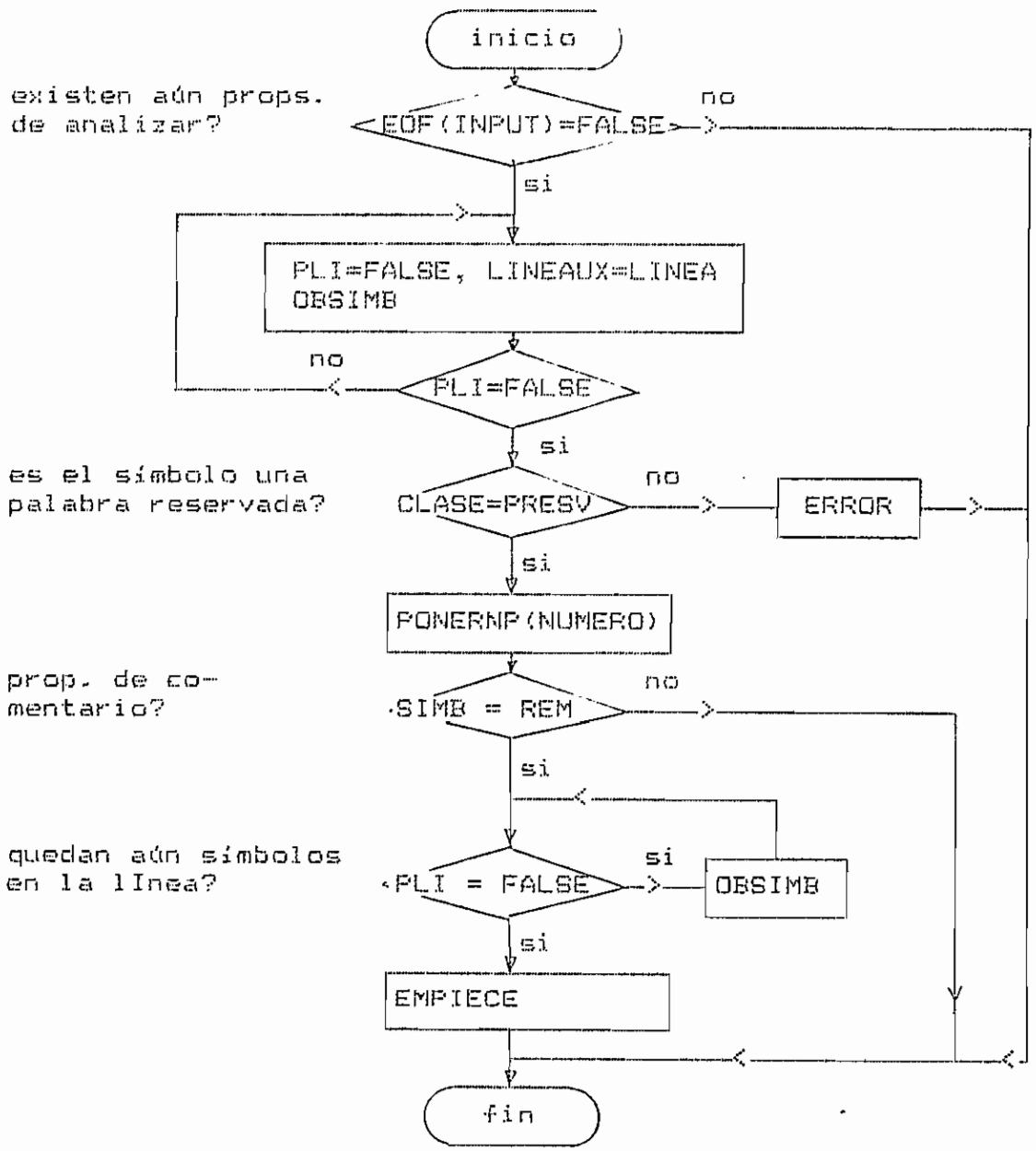


Fig. 4.7i.- Diagrama de flujo de la rutina EMPIECE.

4.3.13.2i.- Rutina P_PROGRAM (interna a ANALISIS).

Analiza la proposición PROGRAM.

variables locales:

I = contador.

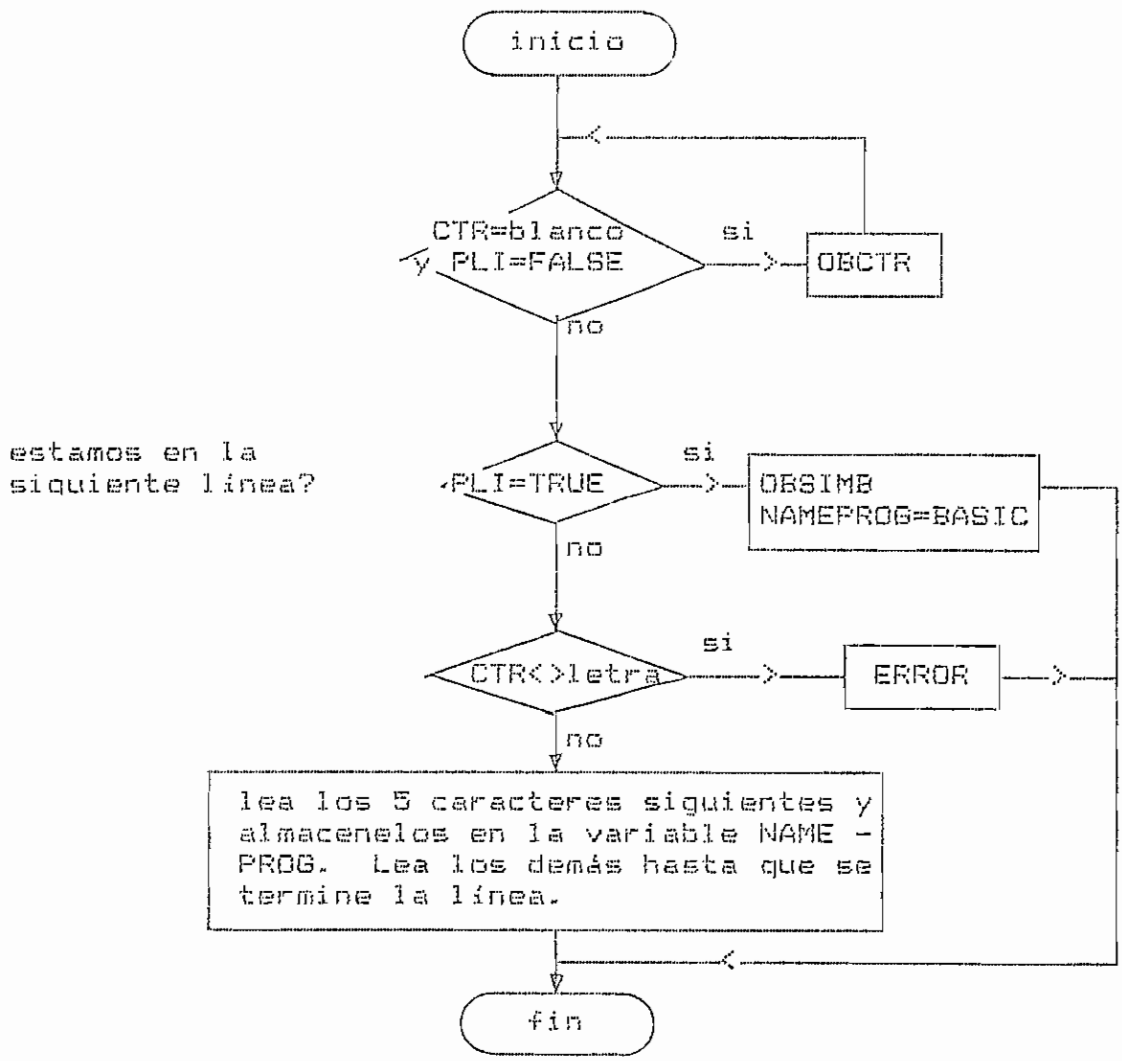


Fig. 4.73.- Diagrama de flujo de la rutina P_PROGRAM.

ESCUELA POLITECNICA NACIONAL
FACULTAD DE INGENIERIA ELECTRICA
ESPECIALIZACION: ELECTRONICA Y TELECOMUNICACIONES

TESIS DE GRADO: COMPILADOR DE BASIC A ASSEMBLER
DEL MICROPROCESADOR M6800

ALUMNO: JORGIO VICENTE TORO R.
DIRECTOR: ING. CESAR ESQUETINI
QUITO 1987

---> PROGRAMA FUENTE ORIGINAL

```
10   PROGRAM EJM1.
20   REM          INGRESO DE DATOS HEXADECIMALES
30   REM          Y OPERACIONES DE SUMA Y RESTA
40   DATA 12, H00A
50   VAR A,B,C
60   READ B,C
70   LET A = B+C-2
80   REM
90   END
```


---> EL PROGRAMA A COMPILARSE SERA EL
ORIGINALMENTE INGRESADO.

---> IDENTIFICADORES EN EL PROGRAMA FUENTE

NOMBRE	TIPO	DIRECCION
A	V	3
B	V	4
C	V	5

---> PROGRAMA FUENTE TRADUCIDO A CODIGO INTERMEDIO

CCD	OPERADOR	OPERANDO HEXADECIMAL	OPERANDO NUMERICO	REGISTRO INDICE
1	INS		6	0
2	CNU		12	0
3	ALM		4	0
4	CNU	$000A	0	0
5	ALM		5	0
6	CVA		4	0
7	CVA		5	0
8	OPR		8	0
9	CNU		2	0
10	OPR		9	0
11	ALM		3	0
12	END		0	0

---> PROGRAMA OBJETO EN LENGUAJE ENSAMBLADOR DEL MICROPROCESADOR M6800

CAS	CMEMOR	OPERADOR	OPERANDO
1		NAM	EJM1
2		ORG	1000
3	1000	LDAA	#12
4	1002	STAA	7
5	1005	LDAA	7
6	1008	STAA	4
7	1011	LDAA	#$000A
8	1013	STAA	7
9	1016	LDAA	7
10	1019	STAA	5
11	1022	LDAA	4
12	1025	STAA	7
13	1028	LDAA	5
14	1031	STAA	8
15	1034	LDAA	8
16	1037	ADDA	7
17	1040	STAA	7
18	1043	LDAA	#2
19	1045	STAA	8
20	1048	LDAA	8
21	1051	SUBA	7

```
                22    1054      NEGA     0
                23    1055      STAA     7
                24    1058      LDAA     7
                25    1061      STAA     3
                26    1064      SWI
                27    1065      END
```

---> LA COMPILACION FUE EXITOSA

ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

ESPECIALIZACION: ELECTRONICA Y TELECOMUNICACIONES

TESIS DE GRADO: COMPILADOR DE BASIC A ASSEMBLER DEL MICROPROCESADOR M6800

ALUMNO: JORGIO VICENTE TORO R.

DIRECTOR: ING. CESAR ESQUETINI

QUITO 1987

---> PROGRAMA FUENTE ORIGINAL

```
10   PROGRAM EJM2
20   REM          DIVISION Y APLICACION DE LA PROPOSICION RESTORE
40   DATA 13, 3
50   VAR A,B,R,R1
60   READ A,B
70   LET R = A/B
80   RESTORE
90   READ R1
100  REM
110  END
```

---> EL PROGRAMA A COMPILARSE SERA EL ORIGINALMENTE INGRESADO.

---> IDENTIFICADORES EN EL PROGRAMA FUENTE

NOMBRE	TIPO	DIRECCION
A	V	3
B	V	4
R	V	5
R1	V	6

---> PROGRAMA FUENTE TRADUCIDO A CODIGO INTERMEDIO

CCD	OPERADOR	OPERANDO HEXADECIMAL	OPERANDO NUMERICO	REGISTRO INDICE
1	INS		7	0
2	CNU		13	0
3	ALM		3	0
4	CNU		3	0
5	ALM		4	0
6	CVA		3	0
7	CVA		4	0
8	OPR		12	0
9	ALM		5	0
10	CNU		13	0
11	ALM		6	0
12	END		0	0

---> PROGRAMA OBJETO EN LENGUAJE ENSAMBLADOR DEL MICROPROCESADOR M6800

CAS	CMEMOR	OPERADOR	OPERANDO
1		NAM	EJM2
2		ORG	1000
3	1000	LDAA	#13
4	1002	STAA	8
5	1005	LDAA	8
6	1008	STAA	3
7	1011	LDAA	#3
8	1013	STAA	8
9	1016	LDAA	8
10	1019	STAA	4
11	1022	LDAA	3
12	1025	STAA	8
13	1028	LDAA	4
14	1031	STAA	9
15	1034	LDAA	#0
16	1036	STAA	0
17	1039	TST	9
18	1042	BNE	4
19	1044	LDAB	#70000

```
20    1046    BRA     68
21    1048    LDAB    #0
22    1050    TST     8
23    1053    BEQ     61
24    1055    LDAA    8
25    1058    CMPA    9
26    1061    BMI     53
27    1063    TST     8
28    1066    BPL     20
29    1068    NEG     8
30    1071    TST     9
31    1074    BPL     5
32    1076    NEG     9
33    1079    BRA     20
34    1081    LDAA    #1
35    1083    STAA    0
36    1086    BRA     13
37    1088    TST     9
38    1091    BPL     8
39    1093    LDAA    #1
40    1095    STAA    0
41    1098    NEG     9
42    1101    INCB    0
43    1102    LDAA    8
44    1105    SUBA    9
45    1108    STAA    8
46    1111    CMPA    9
47    1114    BPL     $F1
48    1116    STAB    8
49    1119    TST     0
50    1122    BEQ     3
51    1124    NEG     8
52    1127    LDAA    8
53    1130    STAA    5
54    1133    LDAA    #13
55    1135    STAA    8
56    1138    LDAA    8
57    1141    STAA    6
58    1144    SWI
59    1145    END
```

---> LA COMPILACION FUE EXITOSA

ESCUELA POLITECNICA NACIONAL
FACULTAD DE INGENIERIA ELECTRICA
ESPECIALIZACION: ELECTRONICA Y TELECOMUNICACIONES

TESIS DE GRADO: COMPILADOR DE BASIC A ASSENBLER
DEL MICROPROCESADOR M6800

ALUMNO: LORGIO VICENTE TORO R.
DIRECTOR: ING. CESAR ESQUETINI
QUITO 1987

---> PROGRAMA FUENTE ORIGINAL

```
10   PROGRAM EJM3
20   REM          APLICACION DE LAZOS
22   VAR A
30   DIM B(5)
40   FOR A=1 TO 5
50     LET B(A) = A+10
60   NEXT
100  REM
110  END
```


---> EL PROGRAMA A COMPILARSE SERA EL
ORIGINALMENTE INGRESADO.

---> IDENTIFICADORES EN EL PROGRAMA FUENTE

NOMBRE	TIPO	DIRECCION
A	V	3
B	A	4

---> PROGRAMA FUENTE TRADUCIDO A CODIGO INTERMEDIO

CCD	OPERADOR	OPERANDO HEXADECIMAL	OPERANDO NUMERICO	REGISTRO INDICE
1	INS		9	0
2	CNU		1	0
3	ALM		3	0
4	CVA		3	0
5	CNU		5	0
6	OPR		2	0
7	SAC		18	0
8	CVA		3	0
9	ALM		1	0
10	CVA		3	0
11	CNU		10	0
12	OPR		8	0
13	ALM		4	1
14	CVA		3	0
15	CNU		1	0
16	OPR		8	0
17	SAI		3	0
18	END		0	0

---> PROGRAMA OBJETO EN LENGUAJE ENSAMBLADOR DEL MICROPROCESADOR M6800

CAS	CMEMOR	OPERADOR	OPERANDO
1		NAM	EJM3
2		ORG	1000
3	1000	LDAA	#1
4	1002	STAA	10
5	1005	LDAA	10
6	1008	STAA	3
7	1011	LDAA	3
8	1014	STAA	10
9	1017	LDAA	#5
10	1019	STAA	11
11	1022	LDAA	11
12	1025	CMPA	10
13	1028	BGE	5
14	1030	CLR	10
15	1033	BRA	5
16	1035	LDAA	#1

```
17    1037    STAA    10
18    1040    TST     10
19    1043    BEQ     63
20    1045    LDAA    3
21    1048    STAA    10
22    1051    LDAA    10
23    1054    STAA    1
24    1057    LDAA    3
25    1060    STAA    10
26    1063    LDAA    #10
27    1065    STAA    11
28    1068    LDAA    11
29    1071    ADDA    10
30    1074    STAA    10
31    1077    LDAA    10
32    1080    LDX     #1
33    1083    STAA    4     ,X
34    1085    LDAA    3
35    1088    STAA    10
36    1091    LDAA    #1
37    1093    STAA    11
38    1096    LDAA    11
39    1099    ADDA    10
40    1102    STAA    10
41    1105    JMP     1005
42    1108    SWI
43    1109    END
```

---> LA COMPILACION FUE EXITOSA

ESCUELA POLITECNICA NACIONAL
FACULTAD DE INGENIERIA ELECTRICA
ESPECIALIZACION: ELECTRONICA Y TELECOMUNICACIONES

TESIS DE GRADO: COMPILADOR DE BASIC A ASSEMBLER
DEL MICROPROCESADOR M6800

ALUMNO: JORGE VICENTE TORO R.
DIRECTOR: ING. CESAR ESQUETINI
QUITO 1987

---> PROGRAMA FUENTE ORIGINAL

```
10   PROGRAM EJM4
20   REM          APLICACION DE LA PROPOSICION CASE
30   VAR A,B
40   LET A=1
50   CASE A OF 60,80
60     LET B= 10
70     GOTO 90
80     LET B= 20
90   ENDCASE
100  END
```

---> EL PROGRAMA A COMPILARSE SERA EL ORIGINALMENTE INGRESADO.

---> IDENTIFICADORES EN EL PROGRAMA FUENTE

NOMBRE	TIPO	DIRECCION
A	V	3
B	V	4

---> PROGRAMA FUENTE TRADUCIDO A CODIGO INTERMEDIO

CCD	OPERADOR	OPERANDO HEXADECIMAL	OPERANDO NUMERICO	REGISTRO INDICE
1	INS		5	0
2	CNU		1	0
3	ALM		3	0
4	CVA		3	0
5	ALM		1	0
6	CVA		1	0
7	CNU		1	0
8	OPR		3	0
9	SAC		11	0
10	SAI		17	0
11	CVA		1	0
12	CNU		2	0
13	OPR		3	0
14	SAC		16	0
15	SAI		20	0
16	SAI		22	0
17	CNU		10	0
18	ALM		4	0
19	SAI		22	0
20	CNU		20	0
21	ALM		4	0
22	END		0	0

---> PROGRAMA OBJETO EN LENGUAJE ENSAMBLADOR DEL MICROPROCESADOR M6800

CAS	CMEMOR	OPERADOR	OPERANDO
1		NAM	EJM4
2		ORG	1000
3	1000	LDAA	#1
4	1002	STAA	6
5	1005	LDAA	6
6	1008	STAA	3
7	1011	LDAA	3
8	1014	STAA	6
9	1017	LDAA	6
10	1020	STAA	1
11	1023	LDAA	1

```
12    1026    STAA    6
13    1029    LDAA    #1
14    1031    STAA    7
15    1034    LDAA    7
16    1037    CMPA    6
17    1040    BEQ     5
18    1042    CLR     6
19    1045    BRA     5
20    1047    LDAA    #1
21    1049    STAA    6
22    1052    TST     6
23    1055    BEQ     3
24    1057    JMP     1100
25    1060    LDAA    1
26    1063    STAA    6
27    1066    LDAA    #2
28    1068    STAA    7
29    1071    LDAA    7
30    1074    CMPA    6
31    1077    BEQ     5
32    1079    CLR     6
33    1082    BRA     5
34    1084    LDAA    #1
35    1086    STAA    6
36    1089    TST     6
37    1092    BEQ     3
38    1094    JMP     1114
39    1097    JMP     1125
40    1100    LDAA    #10
41    1102    STAA    6
42    1105    LDAA    6
43    1108    STAA    4
44    1111    JMP     1125
45    1114    LDAA    #20
46    1116    STAA    6
47    1119    LDAA    6
48    1122    STAA    4
49    1125    SWI
50    1126    END
```

---> LA COMPILACION FUE EXITOSA

ESCUELA POLITECNICA NACIONAL
FACULTAD DE INGENIERIA ELECTRICA
ESPECIALIZACION: ELECTRONICA Y TELECOMUNICACIONES

TESIS DE GRADO: COMPILADOR DE BASIC A ASSEMBLER
DEL MICROPROCESADOR M6800

ALUMNO: LORGIO VICENTE TORO R.
DIRECTOR: ING. CESAR ESQUETINI
QUITO 1987

---> PROGRAMA FUENTE ORIGINAL

```
10   PROGRAM EJM5
20   REM          APLICACION DE LA PROPOSICION WHILE
30   VAR A1
40   DIM B(5)
50   LET A1=1
60   WHILE A1<=5 DO
70     LET B(A1) = A1
80     LET A1= A1+1
90   ENDWHILE
100  END
```

---> EL PROGRAMA A COMPILARSE SERA EL ORIGINALMENTE INGRESADO.

---> IDENTIFICADORES EN EL PROGRAMA FUENTE

NOMBRE	TIPO	DIRECCION
A1	V	3
B	A	4

---> PROGRAMA FUENTE TRADUCIDO A CODIGO INTERMEDIO

CCD	OPERADOR	OPERANDO HEXADECIMAL	OPERANDO NUMERICO	REGISTRO INDICE
1	INS		9	0
2	CNU		1	0
3	ALM		3	0
4	CVA		3	0
5	CNU		5	0
6	OPR		2	0
7	SAC		17	0
8	CVA		3	0
9	ALM		1	0
10	CVA		3	0
11	ALM		4	1
12	CVA		3	0
13	CNU		1	0
14	OPR		8	0
15	ALM		3	0
16	SAI		4	0
17	END		0	

---> PROGRAMA OBJETO EN LENGUAJE ENSAMBLADOR DEL MICROPROCESADOR M6800

CAS	CMEMOR	OPERADOR	OPERANDO
1		NAM	EJM5
2		ORG	1000
3	1000	LDAA	#1
4	1002	STAA	10
5	1005	LDAA	10
6	1008	STAA	3
7	1011	LDAA	3
8	1014	STAA	10
9	1017	LDAA	#5
10	1019	STAA	11
11	1022	LDAA	11
12	1025	CMPA	10
13	1028	BGE	5
14	1030	CLR	10
15	1033	BRA	5
16	1035	LDAA	#1

```
17    1037    STAA    10
18    1040    TST     10
19    1043    BEQ     55
20    1045    LDAA    3
21    1048    STAA    10
22    1051    LDAA    10
23    1054    STAA    1
24    1057    LDAA    3
25    1060    STAA    10
26    1063    LDAA    10
27    1066    LDX     #1
28    1069    STAA    4     ,X
29    1071    LDAA    3
30    1074    STAA    10
31    1077    LDAA    #1
32    1079    STAA    11
33    1082    LDAA    11
34    1085    ADDA    10
35    1088    STAA    10
36    1091    LDAA    10
37    1094    STAA    3
38    1097    JMP     1011
39    1100    SWI
40    1101    END
```

---> LA COMPILACION FUE EXITOSA

ESCUELA POLITECNICA NACIONAL
FACULTAD DE INGENIERIA ELECTRICA
ESPECIALIZACION: ELECTRONICA Y TELECOMUNICACIONES

TESIS DE GRADO: COMPILADOR DE BASIC A ASSEMBLER
DEL MICROPROCESADOR M6800

ALUMNO: LORGIO VICENTE TORO R.
DIRECTOR: ING. CESAR ESQUETINI
QUITO 1987

---> PROGRAMA FUENTE ORIGINAL

```
10  PROGRAM EJM6
20    REM          APLICACION DE LA PROPOSICION IF
30    DATA 11
40    VAR A, B
50    READ A
60    IF  A<10  THEN
70      LET B = 999
80    ELSE
90      LET B = 888
100   ENDIF
110   END
```

---> EL PROGRAMA A COMPILARSE SERA EL ORIGINALMENTE INGRESADO.

---> IDENTIFICADORES EN EL PROGRAMA FUENTE

NOMBRE	TIPO	DIRECCION
A	V	3
B	V	4

---> PROGRAMA FUENTE TRADUCIDO A CODIGO INTERMEDIO

CCD	OPERADOR	OPERANDO HEXADECIMAL	OPERANDO NUMERICO	REGISTRO INDICE
1	INS		5	0
2	CNU		11	0
3	ALM		3	0
4	CVA		3	0
5	CNU		10	0
6	OPR		1	0
7	SAC		11	0
8	CNU		999	0
9	ALM		4	0
10	SAI		13	0
11	CNU		888	0
12	ALM		4	0
13	END		0	0

---> PROGRAMA OBJETO EN LENGUAJE ENSAMBLADOR DEL MICROPROCESADOR M6800

CAS	CMEMOR	OPERADOR	OPERANDO
1		NAM	EJM6
2		ORG	1000
3	1000	LDAA	#11
4	1002	STAA	6
5	1005	LDAA	6
6	1008	STAA	3
7	1011	LDAA	3
8	1014	STAA	6
9	1017	LDAA	#10
10	1019	STAA	7
11	1022	LDAA	7
12	1025	CMPA	6
13	1028	BGT	5
14	1030	CLR	6
15	1033	BRA	5
16	1035	LDAA	#1
17	1037	STAA	6
18	1040	TST	6
19	1043	BEQ	14

```
20   1045   LDAA   #999
21   1047   STAA   6
22   1050   LDAA   6
23   1053   STAA   4
24   1056   JMP    1070
25   1059   LDAA   #888
26   1061   STAA   6
27   1064   LDAA   6
28   1067   STAA   4
29   1070   SWI
30   1071   END
```

---> LA COMPILACION FUE EXITOSA

ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

ESPECIALIZACION: ELECTRONICA Y TELECOMUNICACIONES

TESIS DE GRADO: COMPILADOR DE BASIC A ASSEMBLER DEL MICROPROCESADOR M6800

ALUMNO: LORGIO VICENTE TORO R.

DIRECTOR: ING. CESAR ESQUETINI

QUITO 1987

---> PROGRAMA FUENTE ORIGINAL

```
10   PROGRAM EJM7
20   REM          CONTACTENACION DE STRINGS
30   DATA  "TESIS "
40   VAR A$
50   READ A$
60   LET  A$ = A$|"DE GRADO"
70   END
```

---> EL PROGRAMA A COMPILARSE SERA EL ORIGINALMENTE INGRESADO.

---> IDENTIFICADORES EN EL PROGRAMA FUENTE

NOMBRE	TIPO	DIRECCION
A$	S	3

---> PROGRAMA FUENTE TRADUCIDO A CODIGO INTERMEDIO

CCD	OPERADOR	OPERANDO HEXADECIMAL	OPERANDO NUMERICO	REGISTRO INDICE
1	INS		23	0
2	CNU	$54	0	0
3	ALM		3	0
4	CNU	$45	0	0
5	ALM		4	0
6	CNU	$53	0	0
7	ALM		5	0
8	CNU	$49	0	0
9	ALM		6	0
10	CNU	$53	0	0
11	ALM		7	0
12	CNU	$20	0	0
13	ALM		8	0
14	CNU	$22	0	0
15	ALM		9	0
16	CNU	$22	0	0
17	ALM		10	0
18	CNU	$22	0	0
19	ALM		11	0
20	CNU	$22	0	0
21	ALM		12	0
22	CNU	$22	0	0
23	ALM		13	0
24	CNU	$22	0	0
25	ALM		14	0
26	CNU	$22	0	0
27	ALM		15	0
28	CNU	$22	0	0
29	ALM		16	0
30	CNU	$22	0	0
31	ALM		17	0
32	CNU	$22	0	0
33	ALM		18	0
34	CNU	$22	0	0
35	ALM		19	0
36	CNU	$22	0	0
37	ALM		20	0
38	CNU	$22	0	0
39	ALM		21	0
40	CNU	$22	0	0
41	ALM		22	0
42	CVA		3	0
43	CNU	$44	0	0
44	CNU	$45	0	0
45	CNU	$20	0	0
46	CNU	$47	0	0
47	CNU	$52	0	0
48	CNU	$41	0	0
49	CNU	$44	0	0

50	CNU	$4F	0	0
51	CNU	$22	0	0
52	CNU	$22	0	0
53	CNU	$22	0	0
54	CNU	$22	0	0
55	CNU	$22	0	0
56	CNU	$22	0	0
57	CNU	$22	0	0
58	CNU	$22	0	0
59	CNU	$22	0	0
60	CNU	$22	0	0
61	CNU	$22	0	0
62	CNU	$22	0	0
63	OPR		14	0
64	ALM		3	0
65	END		0	0

---> PROGRAMA OBJETO EN LENGUAJE ENSAMBLADOR DEL MICROPROCESADOR M6800

CAS	CMEMOR	OPERADOR	OPERANDO
1		NAM	EJM7
2		ORG	1000
3	1000	LDAA	#$54
4	1002	STAA	24
5	1005	LDAA	24
6	1008	STAA	3
7	1011	LDAA	#$45
8	1013	STAA	24
9	1016	LDAA	24
10	1019	STAA	4
11	1022	LDAA	#$53
12	1024	STAA	24
13	1027	LDAA	24
14	1030	STAA	5
15	1033	LDAA	#$49
16	1035	STAA	24
17	1038	LDAA	24
18	1041	STAA	6
19	1044	LDAA	#$53
20	1046	STAA	24
21	1049	LDAA	24
22	1052	STAA	7
23	1055	LDAA	#$20
24	1057	STAA	24
25	1060	LDAA	24
26	1063	STAA	8
27	1066	LDAA	#$22

28	1068	STAA	24
29	1071	LDAA	24
30	1074	STAA	5
31	1077	LDAA	#$22
32	1079	STAA	24
33	1082	LDAA	24
34	1085	STAA	10
35	1088	LDAA	#$22
36	1090	STAA	24
37	1093	LDAA	24
38	1096	STAA	11
39	1099	LDAA	#$22
40	1101	STAA	24
41	1104	LDAA	24
42	1107	STAA	12
43	1110	LDAA	#$22
44	1112	STAA	24
45	1115	LDAA	24
46	1118	STAA	13
47	1121	LDAA	#$22
48	1123	STAA	24
49	1126	LDAA	24
50	1129	STAA	14
51	1132	LDAA	#$22
52	1134	STAA	24
53	1137	LDAA	24
54	1140	STAA	15
55	1143	LDAA	#$22
56	1145	STAA	24
57	1148	LDAA	24
58	1151	STAA	16
59	1154	LDAA	#$22
60	1156	STAA	24
61	1159	LDAA	24
62	1162	STAA	17
63	1165	LDAA	#$22
64	1167	STAA	24
65	1170	LDAA	24
66	1173	STAA	18
67	1176	LDAA	#$22
68	1178	STAA	24
69	1181	LDAA	24
70	1184	STAA	19
71	1187	LDAA	#$22
72	1189	STAA	24
73	1192	LDAA	24
74	1195	STAA	20
75	1198	LDAA	#$22
76	1200	STAA	24
77	1203	LDAA	24
78	1206	STAA	21
79	1209	LDAA	#$22
80	1211	STAA	24
81	1214	LDAA	24
82	1217	STAA	22

```
 83   1220   LDAA   #$54
 84   1222   STAA    24
 85   1225   LDAA   #$45
 86   1227   STAA    25
 87   1230   LDAA   #$53
 88   1232   STAA    26
 89   1235   LDAA   #$49
 90   1237   STAA    27
 91   1240   LDAA   #$53
 92   1242   STAA    28
 93   1245   LDAA   #$20
 94   1247   STAA    29
 95   1250   LDAA   #$22
 96   1252   STAA    30
 97   1255   LDAA   #$22
 98   1257   STAA    31
 99   1260   LDAA   #$22
100   1262   STAA    32
101   1265   LDAA   #$22
102   1267   STAA    33
103   1270   LDAA   #$22
104   1272   STAA    34
105   1275   LDAA   #$22
106   1277   STAA    35
107   1280   LDAA   #$22
108   1282   STAA    36
109   1285   LDAA   #$22
110   1287   STAA    37
111   1290   LDAA   #$22
112   1292   STAA    38
113   1295   LDAA   #$22
114   1297   STAA    39
115   1300   LDAA   #$22
116   1302   STAA    40
117   1305   LDAA   #$22
118   1307   STAA    41
119   1310   LDAA   #$22
120   1312   STAA    42
121   1315   LDAA   #$22
122   1317   STAA    43
123   1320   LDAA   #$44
124   1322   STAA    44
125   1325   LDAA   #$45
126   1327   STAA    45
127   1330   LDAA   #$20
128   1332   STAA    46
129   1335   LDAA   #$47
130   1337   STAA    47
131   1340   LDAA   #$52
132   1342   STAA    48
133   1345   LDAA   #$41
134   1347   STAA    49
135   1350   LDAA   #$44
136   1352   STAA    50
137   1355   LDAA   #$4F
```

138	1357	STAA	51
139	1360	LDAA	#$22
140	1362	STAA	52
141	1365	LDAA	#$22
142	1367	STAA	53
143	1370	LDAA	#$22
144	1372	STAA	54
145	1375	LDAA	#$22
146	1377	STAA	55
147	1380	LDAA	#$22
148	1382	STAA	56
149	1385	LDAA	#$22
150	1387	STAA	57
151	1390	LDAA	#$22
152	1392	STAA	58
153	1395	LDAA	#$22
154	1397	STAA	59
155	1400	LDAA	#$22
156	1402	STAA	60
157	1405	LDAA	#$22
158	1407	STAA	61
159	1410	LDAA	#$22
160	1412	STAA	62
161	1415	LDAA	#$22
162	1417	STAA	63
163	1420	LDAA	#$54
164	1422	STAA	24
165	1425	LDAA	#$45
166	1427	STAA	25
167	1430	LDAA	#$53
168	1432	STAA	26
169	1435	LDAA	#$49
170	1437	STAA	27
171	1440	LDAA	#$53
172	1442	STAA	28
173	1445	LDAA	#$20
174	1447	STAA	29
175	1450	LDAA	#$44
176	1452	STAA	30
177	1455	LDAA	#$45
178	1457	STAA	31
179	1460	LDAA	#$20
180	1462	STAA	32
181	1465	LDAA	#$47
182	1467	STAA	33
183	1470	LDAA	#$52
184	1472	STAA	34
185	1475	LDAA	#$41
186	1477	STAA	35
187	1480	LDAA	#$44
188	1482	STAA	36
189	1485	LDAA	#$4F
190	1487	STAA	37
191	1490	LDAA	#$22
192	1492	STAA	38

```
193   1495   LDAA   #$22
194   1497   STAA    39
195   1500   LDAA   #$22
196   1502   STAA    40
197   1505   LDAA   #$22
198   1507   STAA    41
199   1510   LDAA   #$22
200   1512   STAA    42
201   1515   LDAA   #$22
202   1517   STAA    43
203   1520   LDAA   #$54
204   1522   STAA     3
205   1525   LDAA   #$45
206   1527   STAA     4
207   1530   LDAA   #$53
208   1532   STAA     5
209   1535   LDAA   #$49
210   1537   STAA     6
211   1540   LDAA   #$53
212   1542   STAA     7
213   1545   LDAA   #$20
214   1547   STAA     8
215   1550   LDAA   #$44
216   1552   STAA     9
217   1555   LDAA   #$45
218   1557   STAA    10
219   1560   LDAA   #$20
220   1562   STAA    11
221   1565   LDAA   #$47
222   1567   STAA    12
223   1570   LDAA   #$52
224   1572   STAA    13
225   1575   LDAA   #$41
226   1577   STAA    14
227   1580   LDAA   #$44
228   1582   STAA    15
229   1585   LDAA   #$4F
230   1587   STAA    16
231   1590   LDAA   #$22
232   1592   STAA    17
233   1595   LDAA   #$22
234   1597   STAA    18
235   1600   LDAA   #$22
236   1602   STAA    19
237   1605   LDAA   #$22
238   1607   STAA    20
239   1610   LDAA   #$22
240   1612   STAA    21
241   1615   LDAA   #$22
242   1617   STAA    22
243   1620   SWI
244   1621   END
```

ESCUELA POLITECNICA NACIONAL

FACULTAD DE INGENIERIA ELECTRICA

ESPECIALIZACION: ELECTRONICA Y TELECOMUNICACIONES

TESIS DE GRADO: COMPILADOR DE BASIC A ASSEMBLER
DEL MICROPROCESADOR M6800

ALUMNO: LORGIO VICENTE TORO R.

DIRECTOR: ING. CESAR ESQUETINI

QUITO 1987

---> PROGRAMA FUENTE ORIGINAL

```
10  PROGRAM EJM8
20  REM          APLICACION DE FUNCIONES
30  VAR A,P
40  REM  DEFINICION DE LA FUNCION
50  DEF FNB(P)
60    LET FNB = P+11
70  ENDFN
80  REM  PROGRAMA PRINCIPAL
90  LET  A= FNB(9)
100 END
```


---> EL PROGRAMA A COMPILARSE SERA EL ORIGINALMENTE INGRESADO.

---> IDENTIFICADORES EN EL PROGRAMA FUENTE

NOMBRE	TIPO	DIRECCION
A	V	3
P	V	4
FNB	F	5

---> PROGRAMA FUENTE TRADUCIDO A CODIGO INTERMEDIO

CCD	OPERADOR	OPERANDO HEXADECIMAL	OPERANDO NUMERICO	REGISTRO INDICE
1	INS		6	0
2	CVA		4	0
3	CNU		11	0
4	OPR		8	0
5	ALM		5	0
6	RTS		0	0
7	CNU		9	0
8	ALM		4	0
9	LLS		2	0
10	CVA		5	0
11	ALM		3	0
12	END		0	0

---> PROGRAMA OBJETO EN LENGUAJE ENSAMBLADOR DEL MICROPROCESADOR M6800

CAS	CMEMOR	OPERADOR	OPERANDO
1		NAM	EJM8
2		ORG	1000
3	1000	LDAA	4
4	1003	STAA	7
5	1006	LDAA	#11
6	1008	STAA	8
7	1011	LDAA	8
8	1014	ADDA	7
9	1017	STAA	7
10	1020	LDAA	7
11	1023	STAA	5
12	1026	RTS	0
13	1027	LDAA	#9
14	1029	STAA	7
15	1032	LDAA	7
16	1035	STAA	4
17	1038	JSR	1000
18	1041	LDAA	5
19	1044	STAA	7
20	1047	LDAA	7

```
21    1050      STAA    3
22    1053      SWI
23    1054      END

---> LA COMPILACION FUE EXITOSA
```

```
==========================================================
==========================================================
===                                                    ===
===        ESCUELA POLITECNICA NACIONAL                ===
===        FACULTAD DE INGENIERIA ELECTRICA            ===
===        ESPECIALIZACION: ELECTRONICA Y TELECOMUNICACIONES ===
===                                                    ===
===        TESIS DE GRADO: COMPILADOR DE BASIC A ASSEMBLER ===
===                        DEL MICROPROCESADOR M6800   ===
===                                                    ===
===        ALUMNO: JORGIO VICENTE TORO R.              ===
===        DIRECTOR: ING. CESAR ESQUETINI              ===
===        QUITO 1987                                  ===
===                                                    ===
==========================================================
==========================================================
```

---> PROGRAMA FUENTE ORIGINAL

```
10   PROGRAM EJM9
20   REM          APLICACION DE SUBRUTINAS Y MULTIPLICACION
30   VAR A,A1
40   REM  DEFINICION DE LA SUBRUTINA
50   SUB
60     LET A= A*5
70   RETURN
80   REM  PROGRAMA PRINCIPAL
90   LET  A= -2
100  GOSUB 50
110  END
```

---> EL PROGRAMA A COMPILARSE SERA EL ORIGINALMENTE INGRESADO.

---> IDENTIFICADORES EN EL PROGRAMA FUENTE

NOMBRE	TIPO	DIRECCION
A	V	3
A1	V	4

---> PROGRAMA FUENTE TRADUCIDO A CODIGO INTERMEDIO

CCD	OPERADOR	OPERANDO HEXADECIMAL	OPERANDO NUMERICO	REGISTRO INDICE
1	INS		5	0
2	CVA		3	0
3	CNU		5	0
4	OPR		11	0
5	ALM		3	0
6	RTS		0	0
7	CNU		2	0
8	OPR		7	0
9	ALM		3	0
10	LLS		2	0
11	END		0	0

---> PROGRAMA OBJETO EN LENGUAJE ENSAMBLADOR DEL MICROPROCESADOR M6800

CAS	CMEMOR	OPERADOR	OPERANDO
1		NAM	EJM9
2		ORG	1000
3	1000	LDAA	3
4	1003	STAA	6
5	1006	LDAA	#5
6	1008	STAA	7
7	1011	LDAA	#0
8	1013	STAA	0
9	1016	LDA8	#0
10	1018	TST	6
11	1021	BEQ	53
12	1023	TST	7
13	1026	BNE	2
14	1028	BRA	46
15	1030	TST	6
16	1033	BPL	20
17	1035	NEG	6
18	1038	TST	7
19	1041	BPL	5
20	1043	NEG	7
21	1046	BRA	20

```
22   1048   LDAA   #1
23   1050   STAA   0
24   1053   BRA    13
25   1055   TST    7
26   1058   BPL    8
27   1060   LDAA   #1
28   1062   STAA   0
29   1065   NEG    7
30   1068   ADDB   6
31   1071   DEC    7
32   1074   BNE    $F8
33   1076   STAB   6
34   1079   TST    0
35   1082   BEQ    3
36   1084   NEG    6
37   1087   LDAA   6
38   1090   STAA   3
39   1093   RTS    0
40   1094   LDAA   #2
41   1096   STAA   6
42   1099   NEG    6
43   1102   LDAA   6
44   1105   STAA   3
45   1108   JSR    1000
46   1111   SWI
47   1112   END
```

---> LA COMPILACION FUE EXITOSA

```
==========================================================
==========================================================
===                                                    ===
===        ESCUELA POLITECNICA NACIONAL                ===
===        FACULTAD DE INGENIERIA ELECTRICA            ===
===        ESPECIALIZACION: ELECTRONICA Y TELECOMUNICACIONES ===
===                                                    ===
===        TESIS DE GRADO: COMPILADOR DE BASIC A ASSEMBLER ===
===                        DEL MICROPROCESADOR M6800   ===
===                                                    ===
===        ALUMNO: LORGIO VICENTE TORO R.              ===
===        DIRECTOR: ING. CESAR ESQUETINI              ===
===        QUITO 1987                                  ===
===                                                    ===
==========================================================
==========================================================
```

---> PROGRAMA FUENTE ORIGINAL

```
10   PROGRAM EJM9
20   DATA 2,1,2,2,2,1
30   VAR  I,J,K
40   DIM A(2,2),B(2),C(2)
41   REM LECTURA DE DATOS
50   FOR I=1 TO 2
60     FOR J=1 TO 2
70       READ A(I,J)
80       NEXT
90   NEXT
110  FOR I=1 TO 2
120      READ B(I)
130  NEXT
131  REM MULTIPLICACION DE MATRICES
140  FOR I=1 TO 2
160    LET C(I) = 0
170    FOR K=1 TO 2
180      LET C(I) = C(I) + A(I,K)*B(K)
190    NEXT
200  NEXTT
11   REM APLICACION DE ARREGLOS.
12   REM DETECCION DE ERRORES.
210  END
```

---> EL PROGRAMA FUENTE ORIGINAL TUVO LINEAS SIN NUMERO
DE PROPOSICION O BIEN CON NUMEROS REPETIDOS O NO

HUBO ORDEN ASCENDENTE. POR LO TANTO EL PROGRAMA FUENTE A COMPILARSE SERA EL SIGUIENTE:

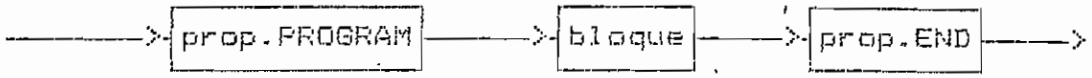
```
 10   PROGRAM EJM9
 11   REM APLICACION DE ARREGLOS.
 12   REM DETECCION DE ERRORES.
 20   DATA 2,1,2,2,2,1
 30   VAR  I,J,K
 40   DIM A(2,2),B(2),C(2)
 41   REM LECTURA DE DATOS
 50   FOR I=1 TO 2
 60     FOR J=1 TO 2
 70       READ A(I,J)
 80       NEXT
 90   NEXT
 110  FOR I=1 TO 2
 120    READ B(I)
 130  NEXT
 131  REM MULTIPLICACION DE MATRICES
 140  FOR I=1 TO 2
 160    LET C(I) = 0
 170    FOR K=1 TO 2
 180      LET C(I) = C(I) + A(I,K)*B(K)
 190    NEXT
 200  NEXTT
 210  END


200  NEXTT
-> SIMBOLO INDEFINIDO O FUERA DE CONTEXTO
210  END
       $

-> SE ESPERABA PROPOSICION "NEXT"
-> ULTIMA PROPOSICION DEBE SER "END"
---> IDENTIFICADORES EN EL PROGRAMA FUENTE
```

NOMBRE	TIPO	DIRECCION
I	V	3
J	V	4
K	V	5
A	A	6
B	A	10
C	A	12

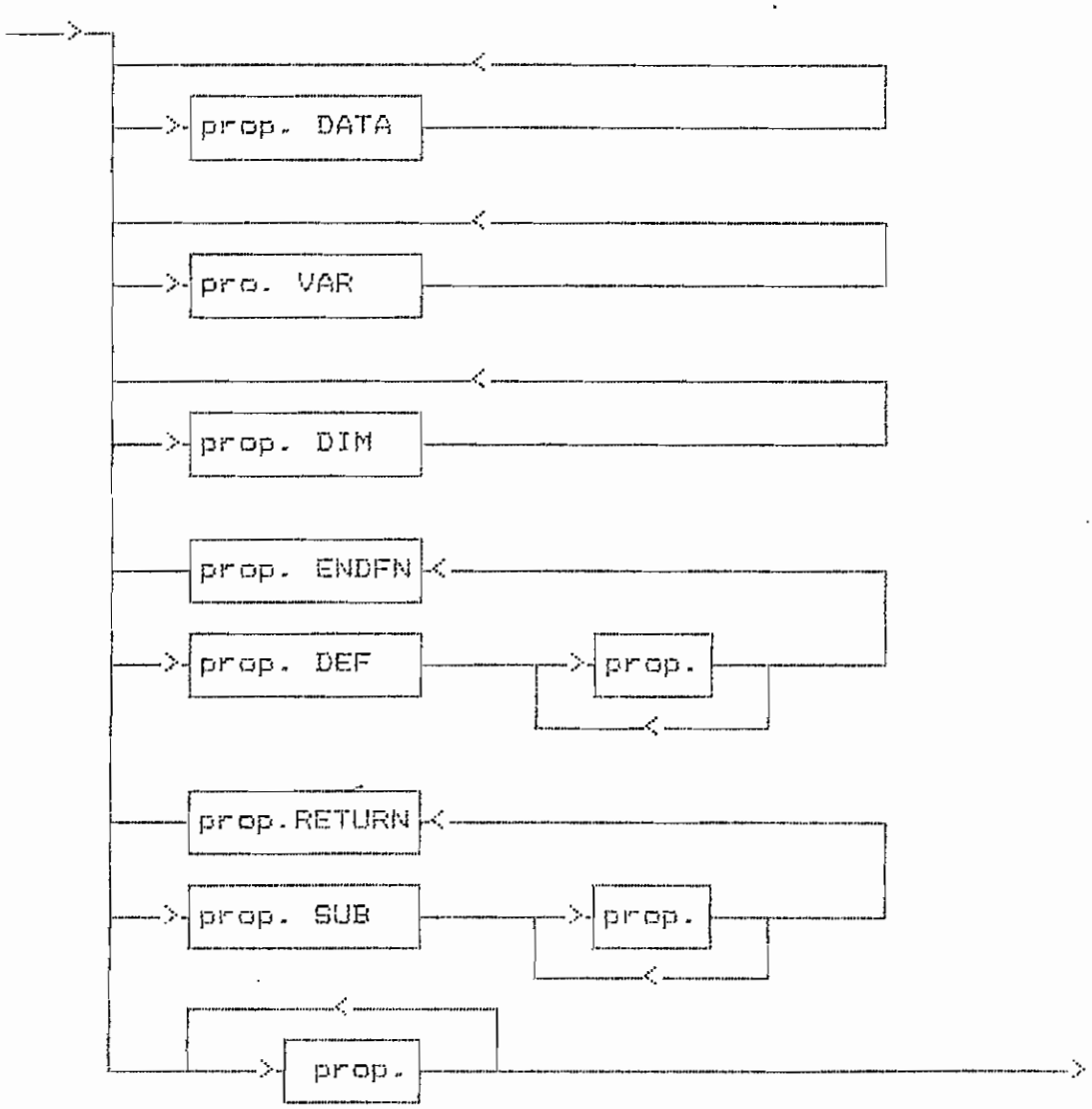
```
---> LA COMPILACION SE DETIENE POR ERRORES
     EN EL PROGRAMA FUENTE
```

ANEXO 2.- SINTAXIS DEL LENGUAJE FUENTE (VERSION BASIC).

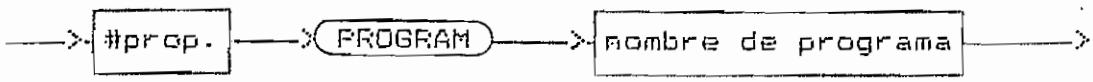
programa



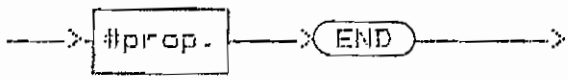
bloque



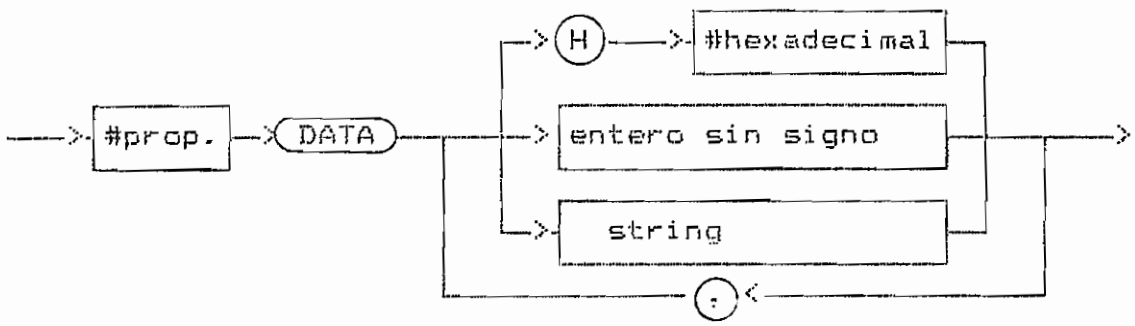
prop. PROGRAM



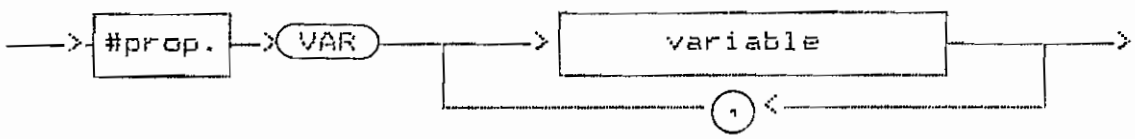
prop. END



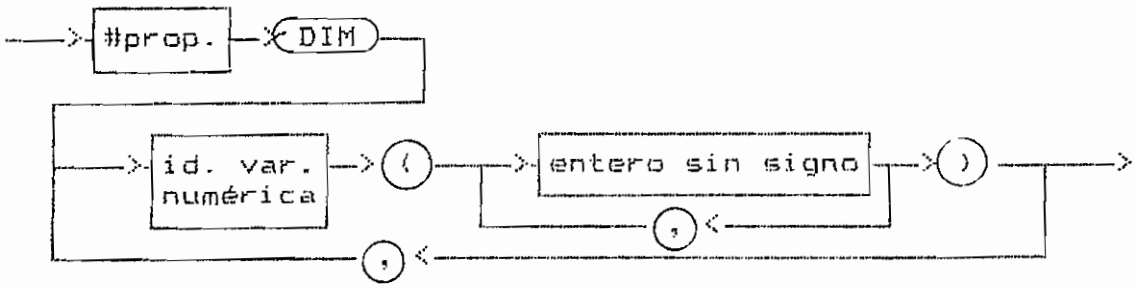
prop. DATA



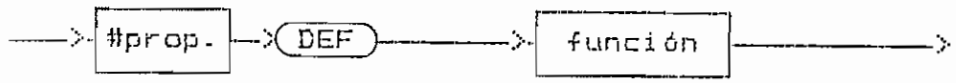
prop. VAR



prop. DIM



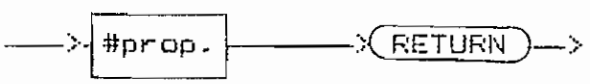
prop. DEF



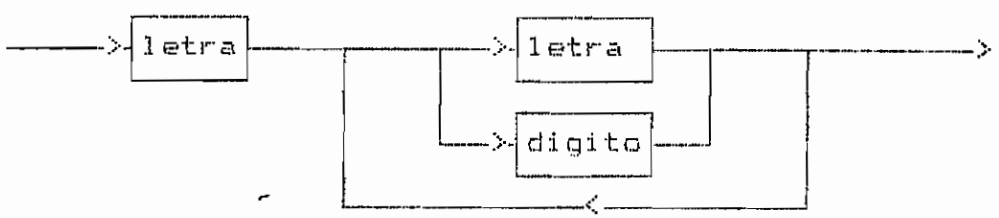
prop. ENDFN

prop. SUB

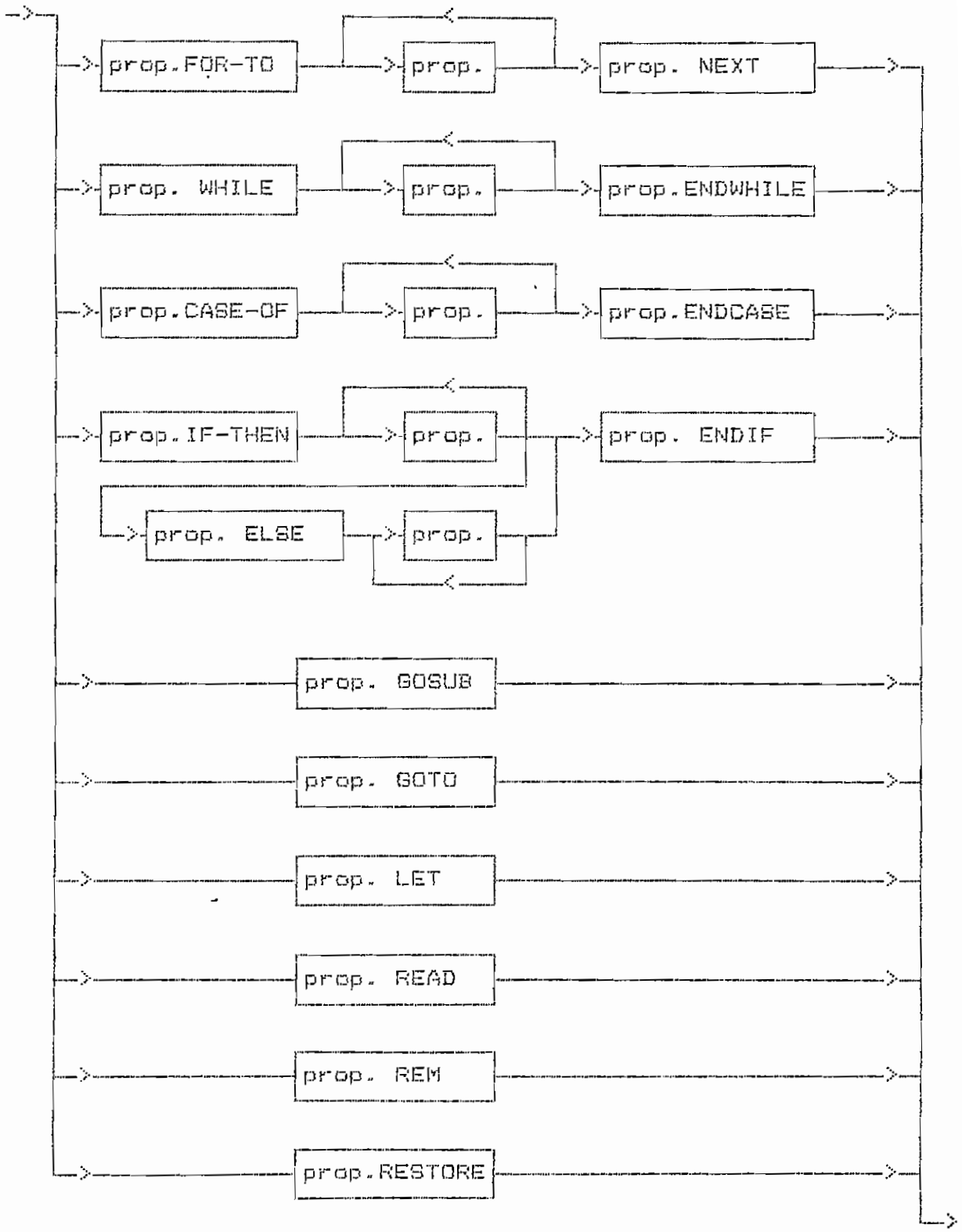
prop. RETURN



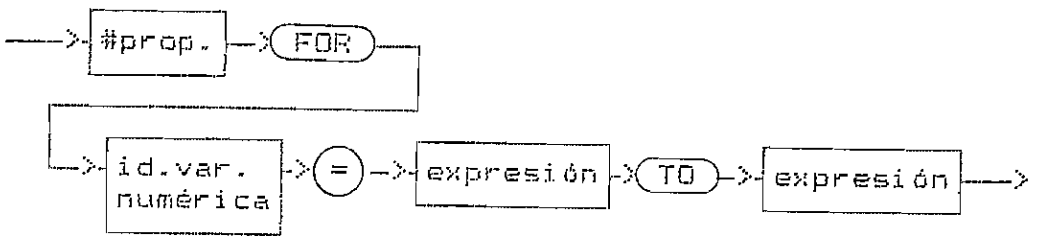
nombre de programa



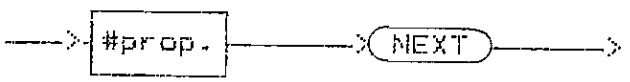
proposición



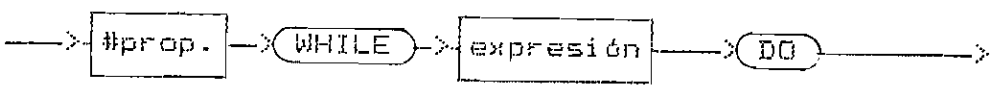
prop. FOR-TO



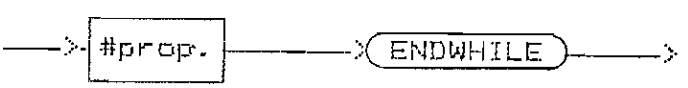
prop. NEXT



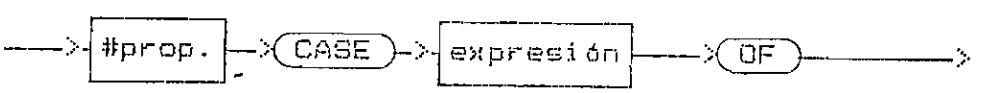
prop. WHILE-DO



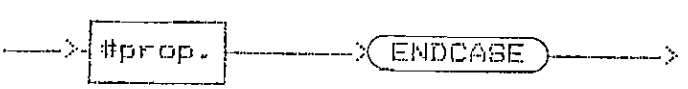
prop. ENDWHILE



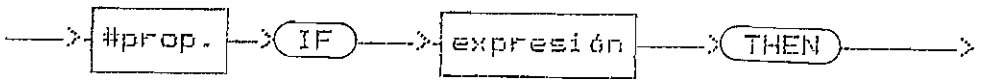
prop. CASE-OF



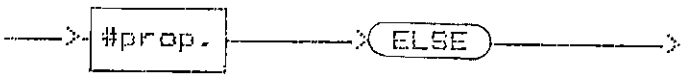
prop. ENDCASE



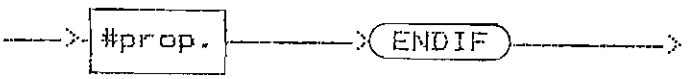
prop. IF-THEN



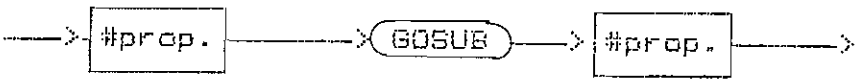
prop. ELSE



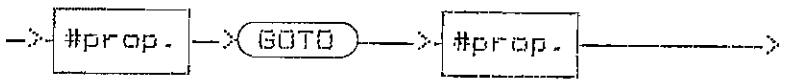
prop. ENDIF



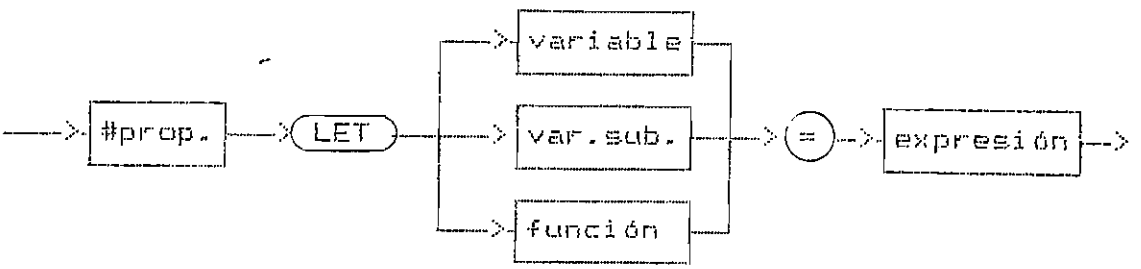
prop. GOSUB



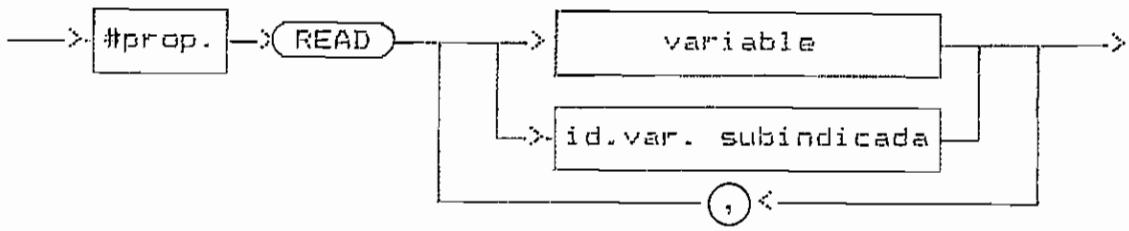
prop. GOTO



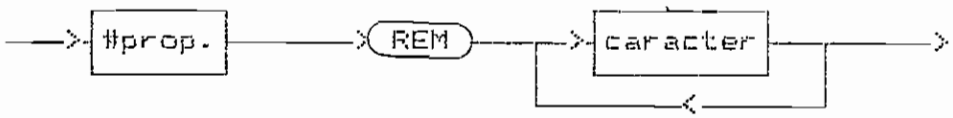
prop. LET



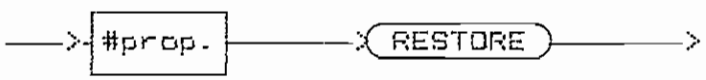
prop. READ



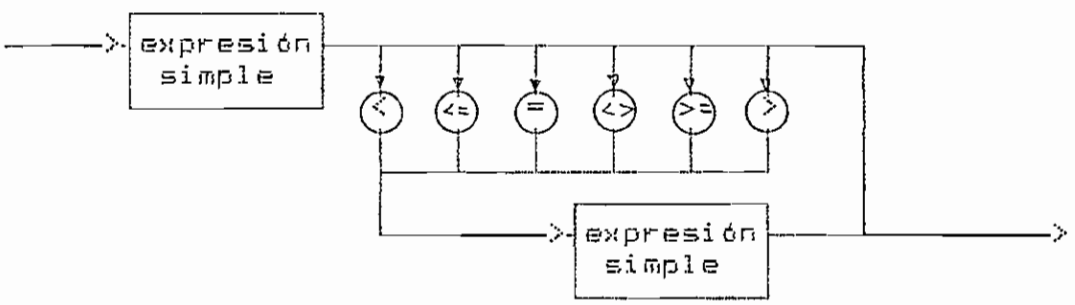
prop. REM



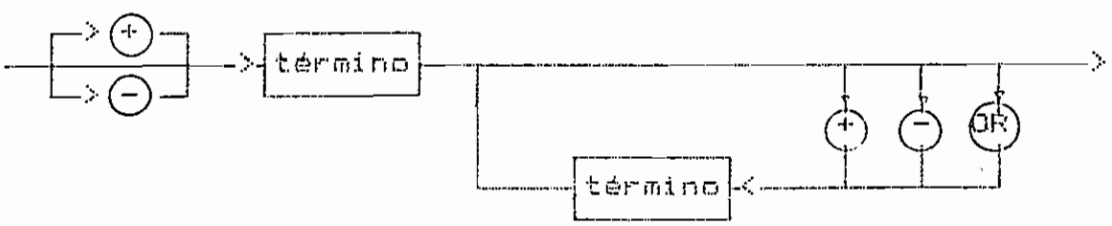
prop. RESTORE



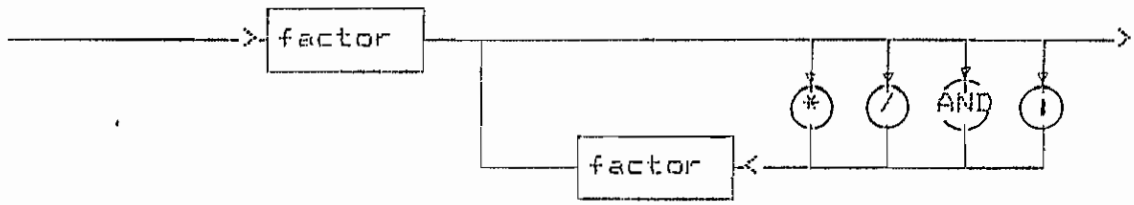
expresión



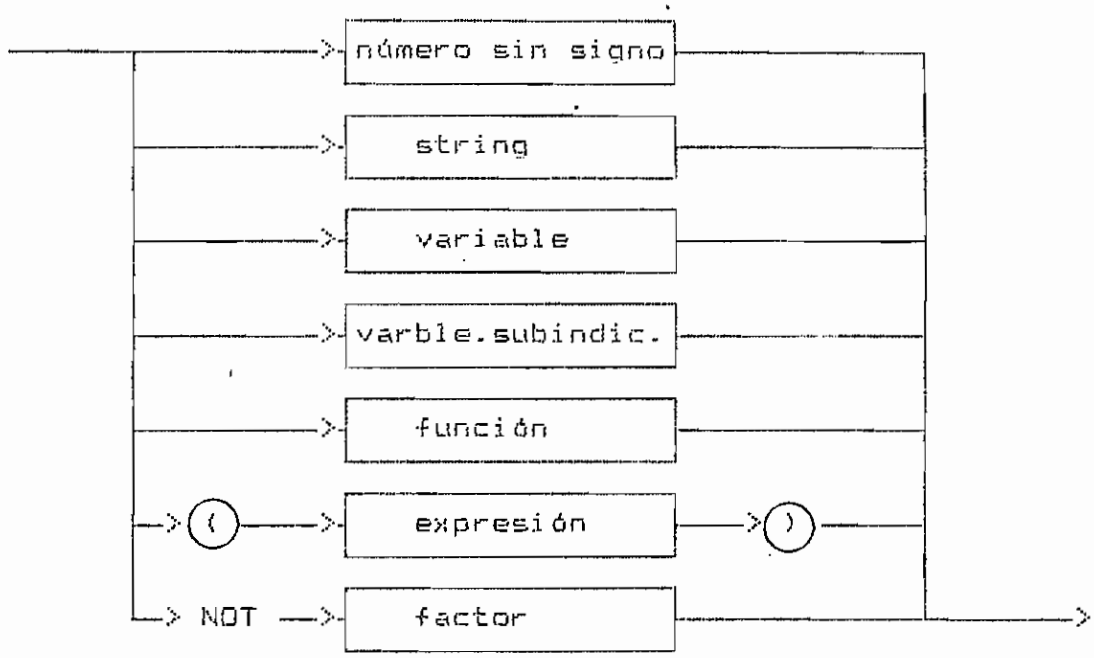
expresión simple



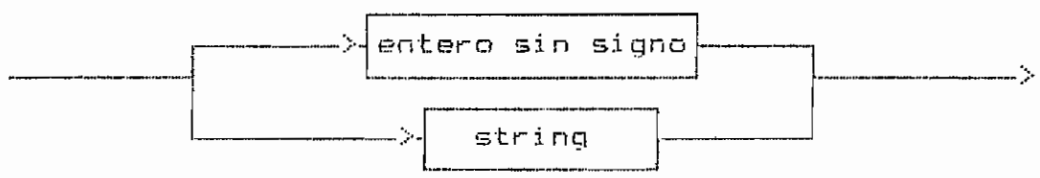
término



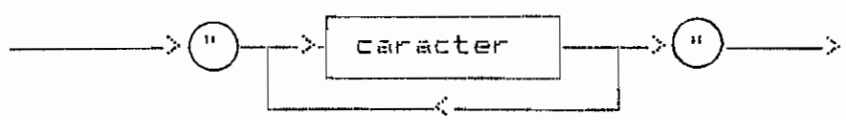
factor



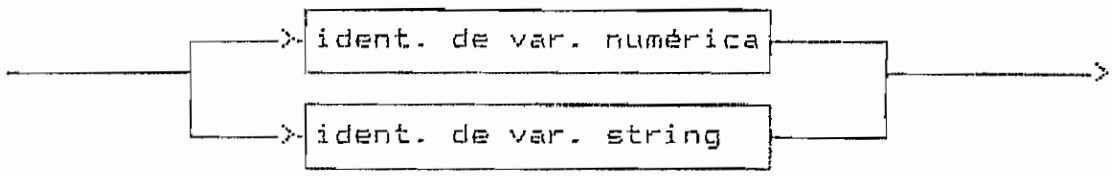
constante



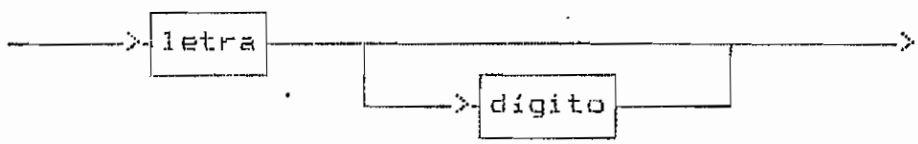
string



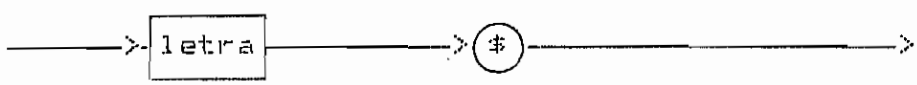
variable



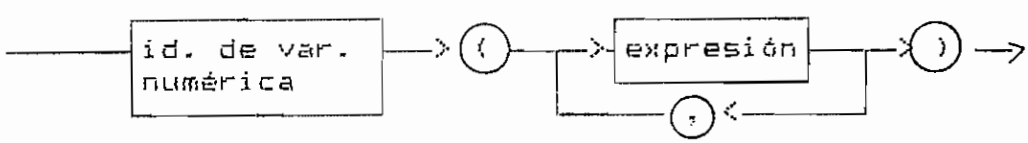
id. de var. numérica



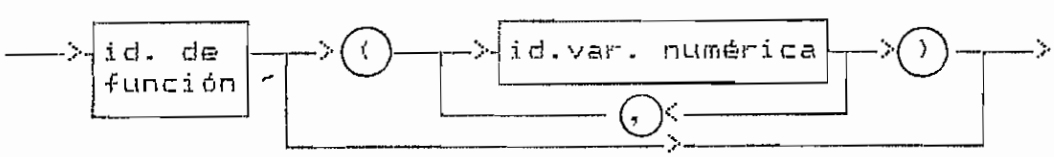
id. de var. string



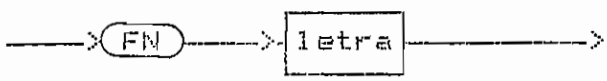
variable subindicada



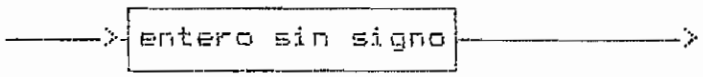
función



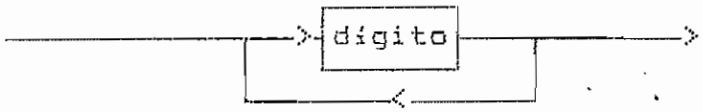
id. de función



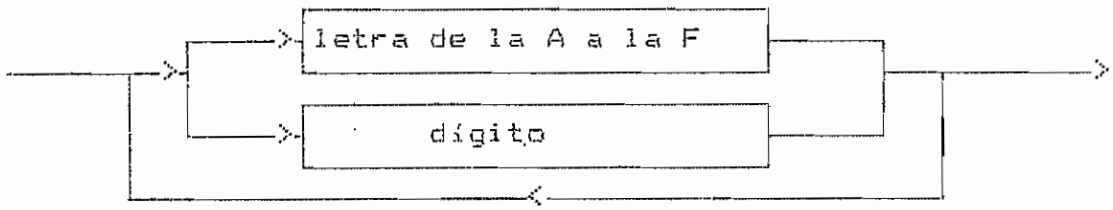
#prop. (número de proposición)



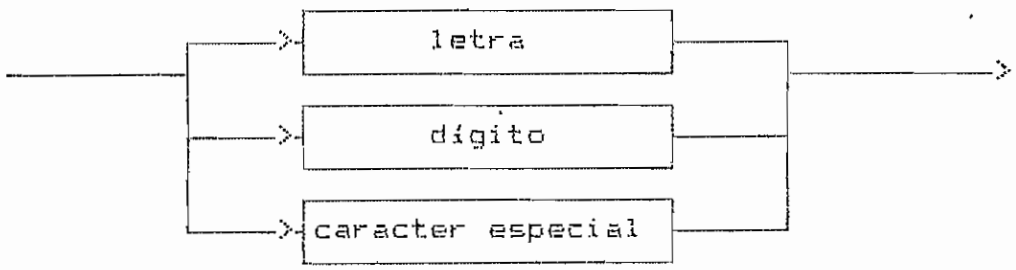
entero sin signo



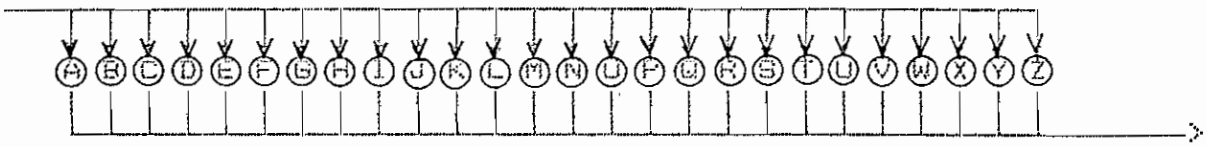
número hexadecimal



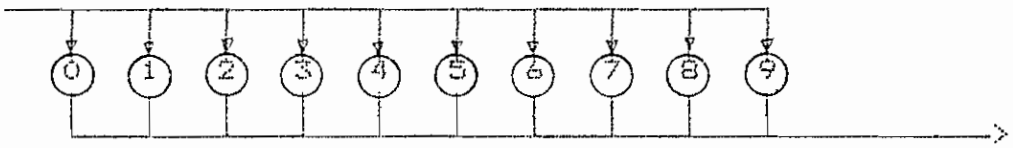
caracter



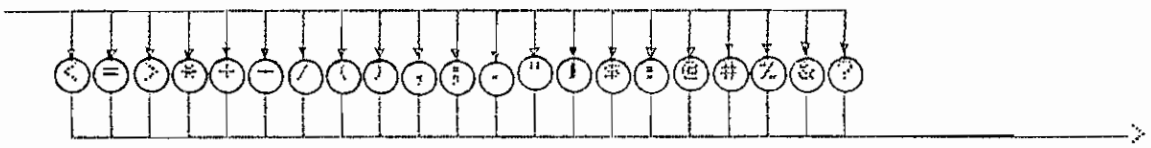
letra



dígito

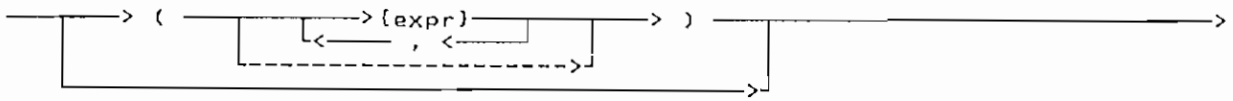


caracter especial

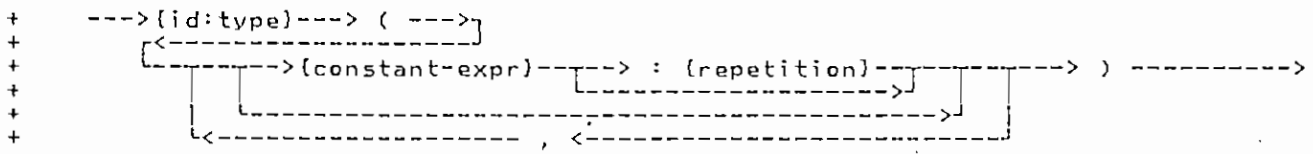


(también se incluye el espacio en blanco)

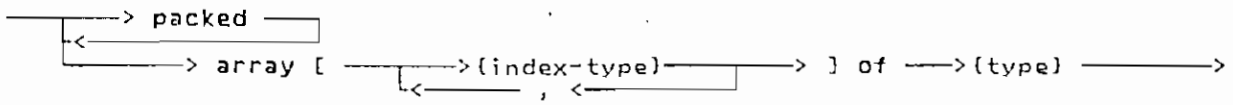
actual-parameters:



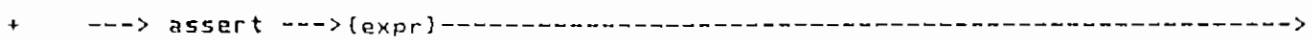
array-structure:



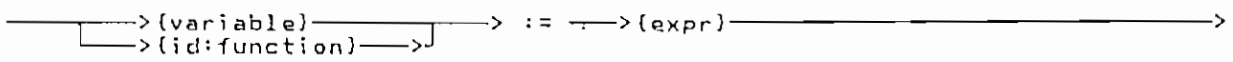
array-type:



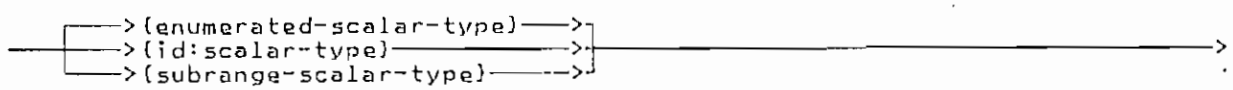
assert-statement:



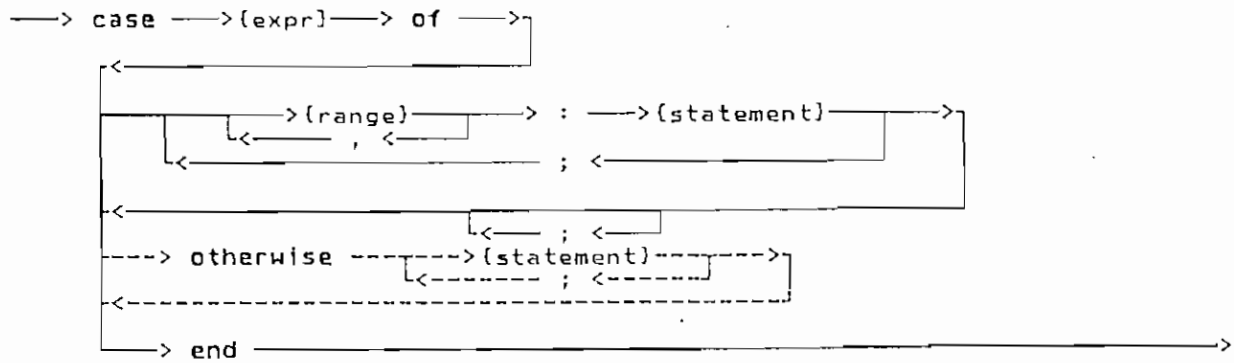
assignment-statement:



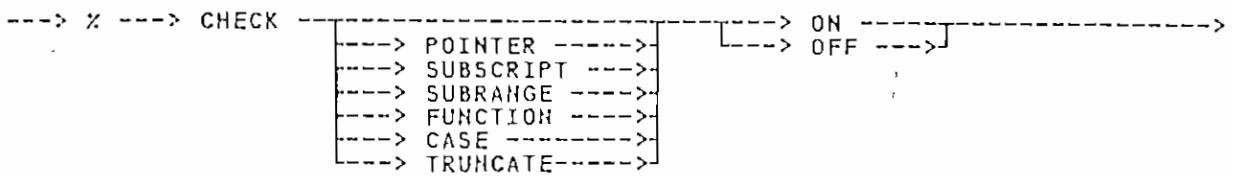
base-scalar-type:



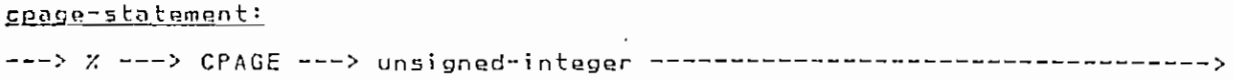
case-statement:



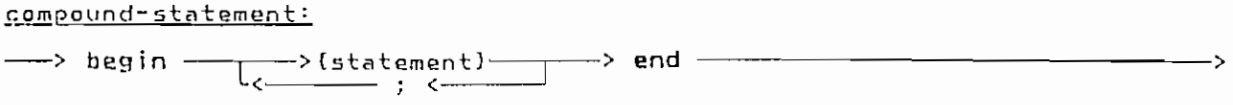
check-statement:



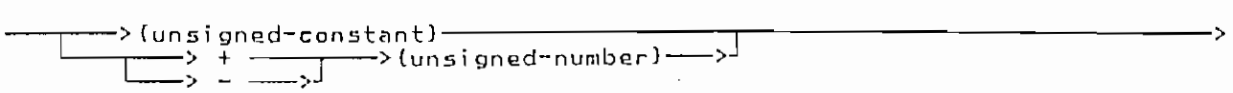
cpage-statement:



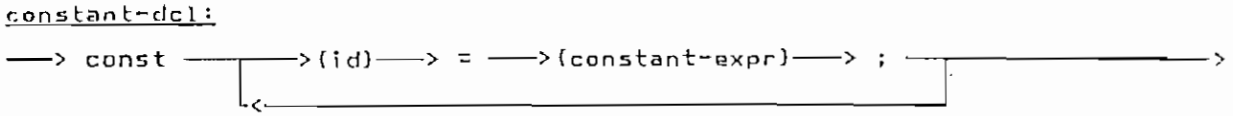
compound-statement:



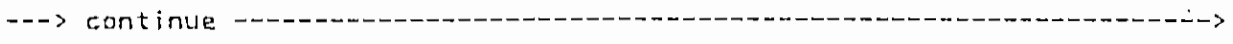
constant:



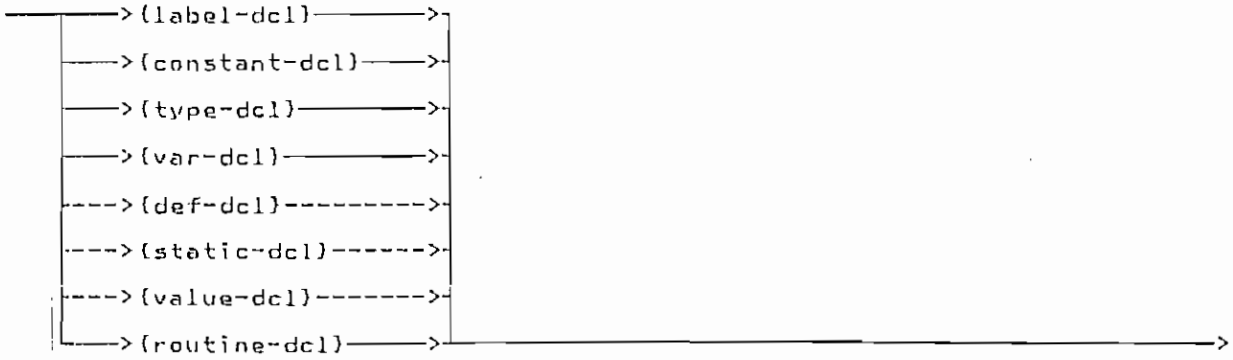
constant-dcl:



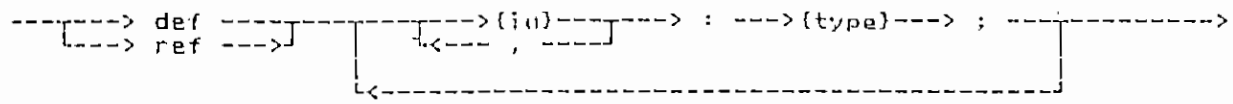
continue-statement:



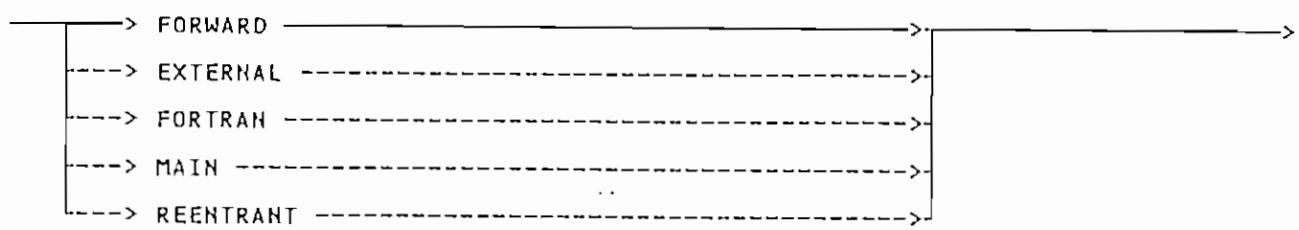
declaration:



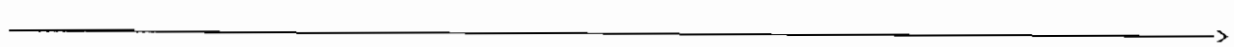
def-dcl:



directive:

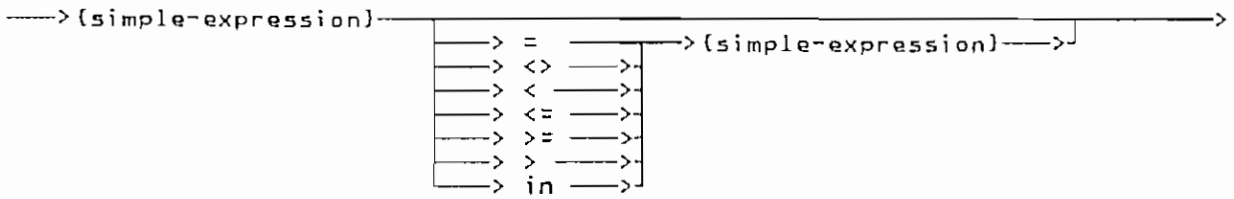


empty-statement:

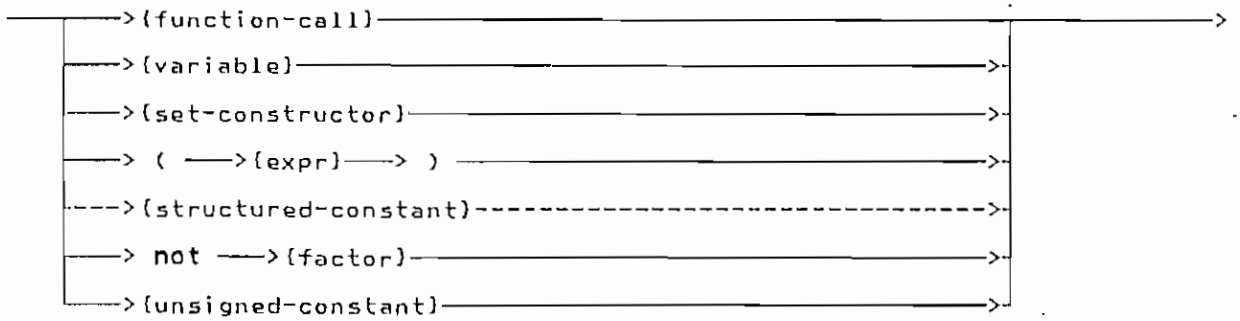


enumerated-scalar-type:

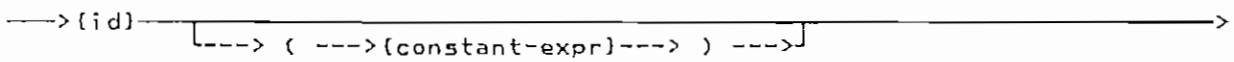
expr:
constant-expr:



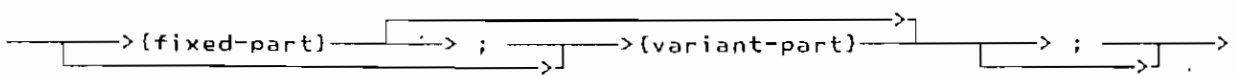
factor:



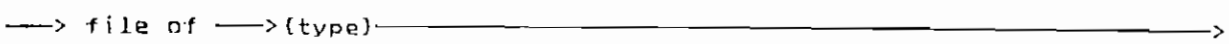
field:



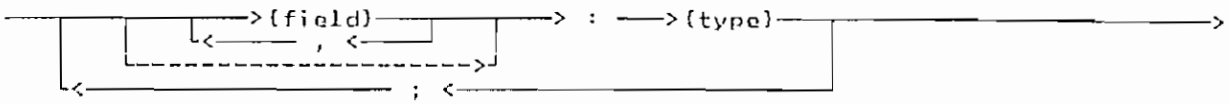
field-list:



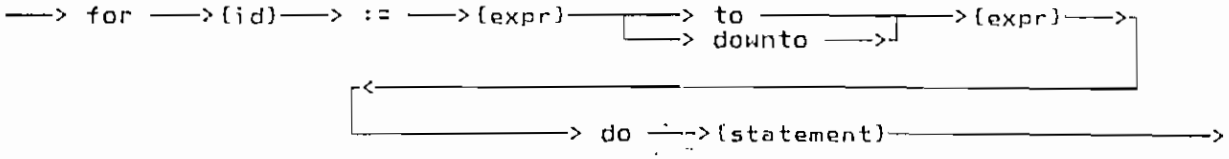
file-type:



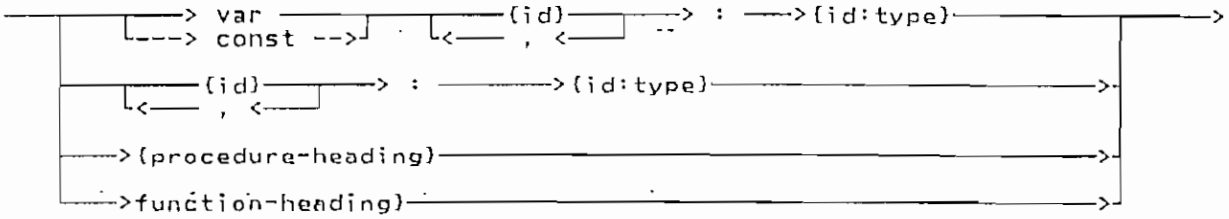
fixed-part:



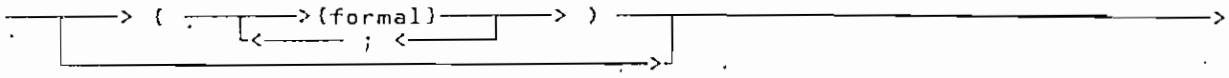
for-statement:



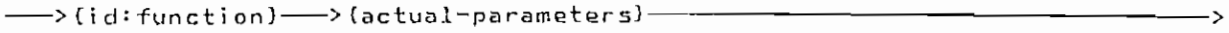
formal:



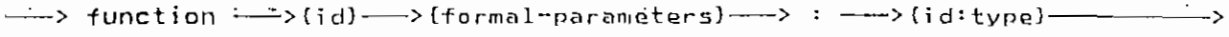
formal-parameters:



function-call:

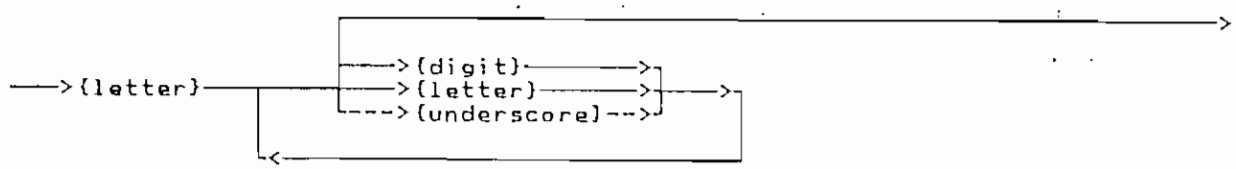


function-heading:

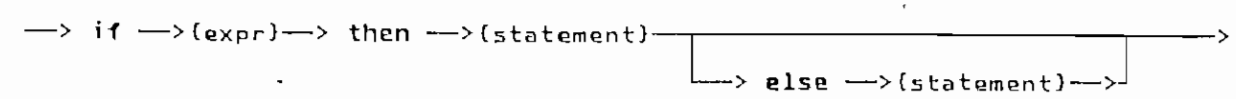


goto-statement:

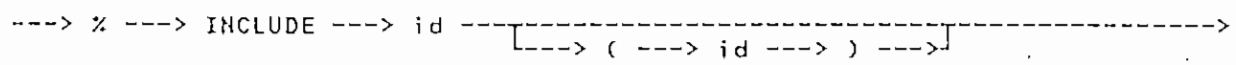
id:



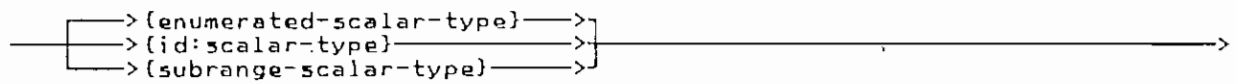
if-statement:



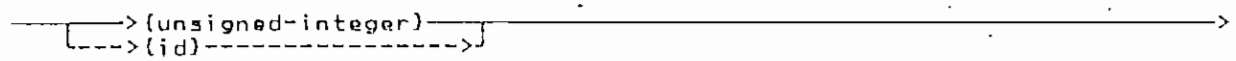
include-statement:



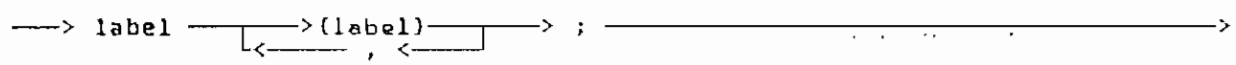
index-type:



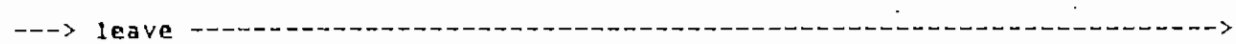
label:



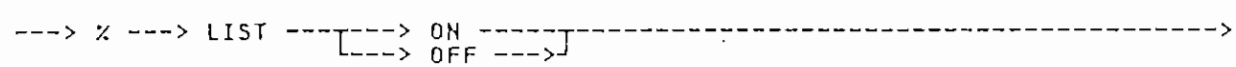
label-dcl:



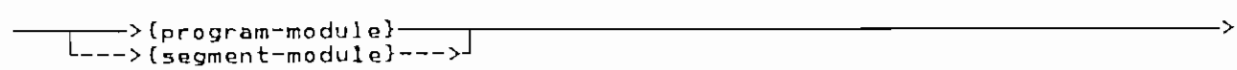
leave-statement:



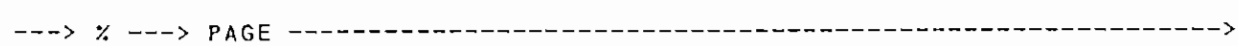
list-statement:



module:

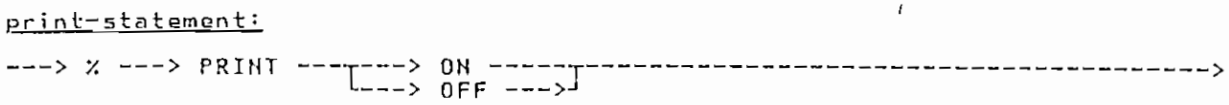


page-statement:

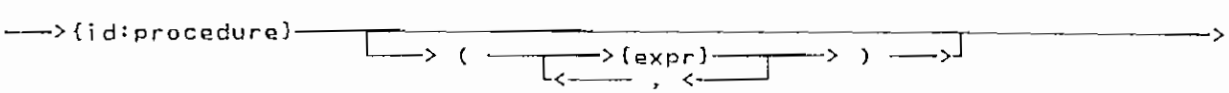


pointer-type:

print-statement:

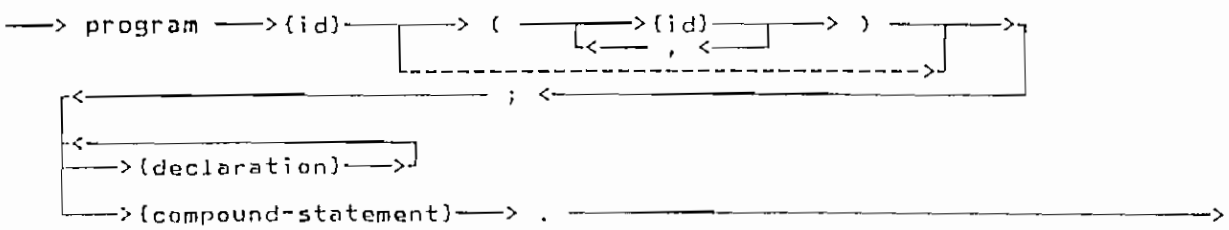


procedure-call:

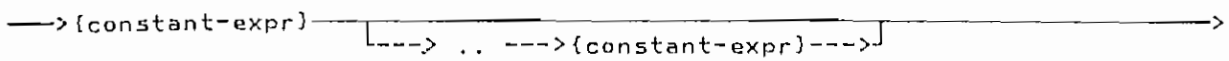


procedure-heading:

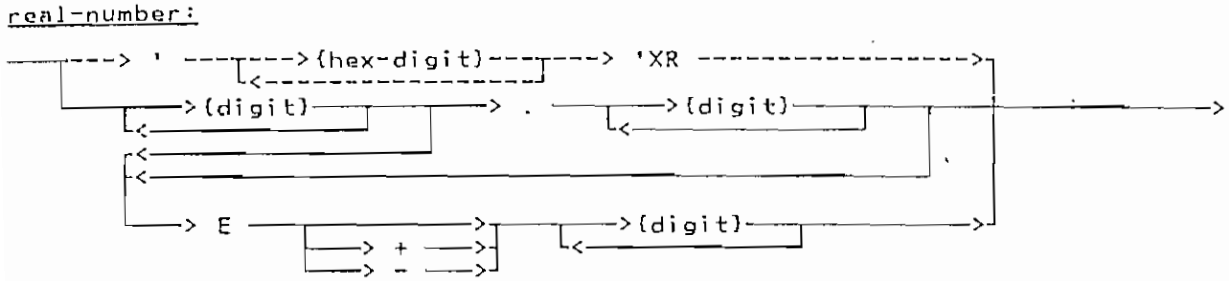
program-module:



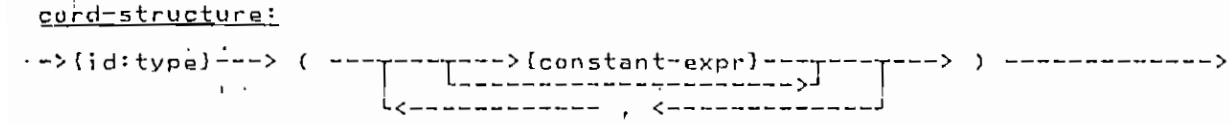
range:



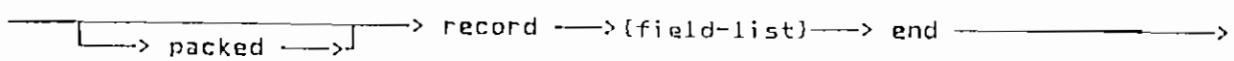
real-number:



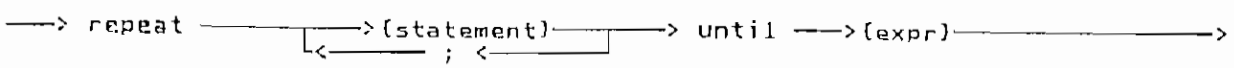
cond-structure:



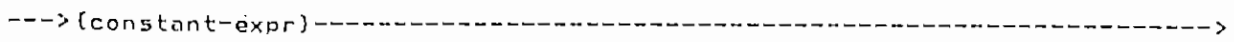
record-type:



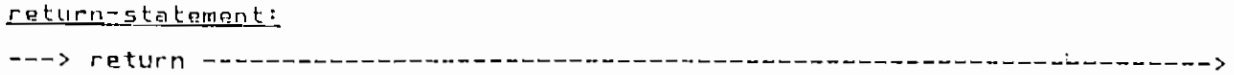
repeat-statement:



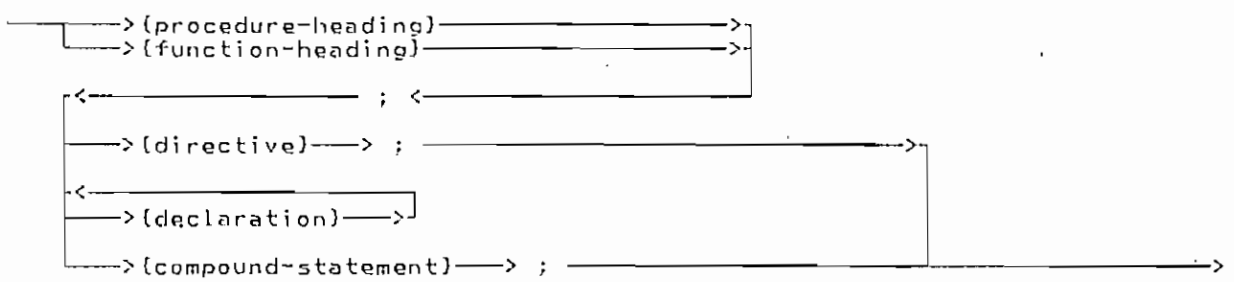
repetition:



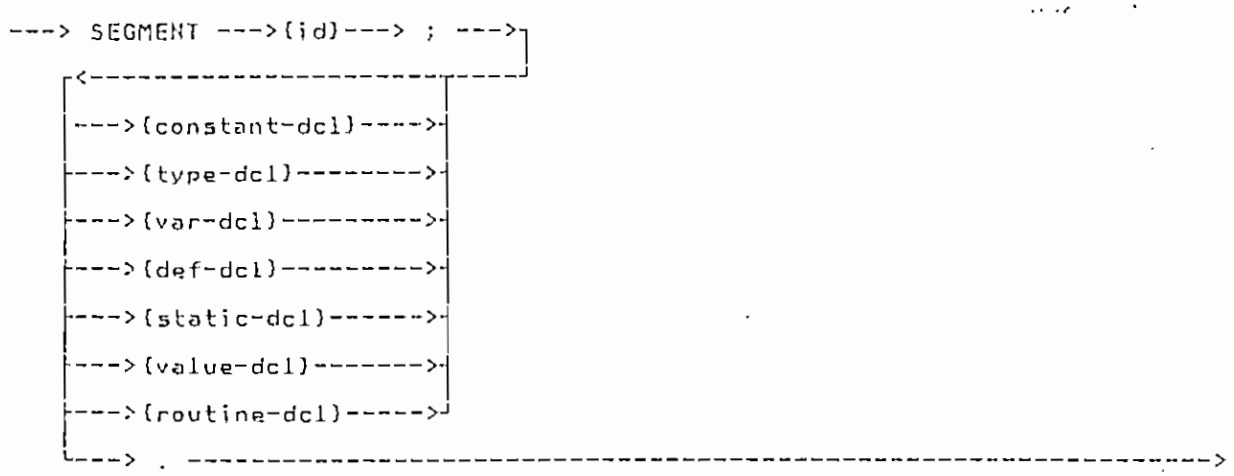
return-statement:



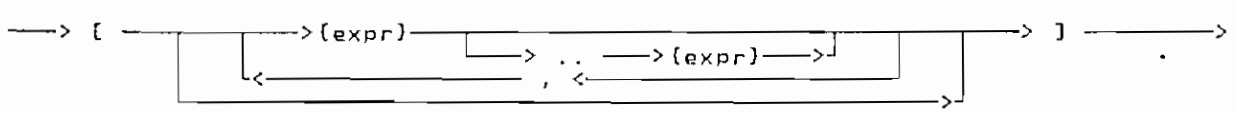
routine-dcl:



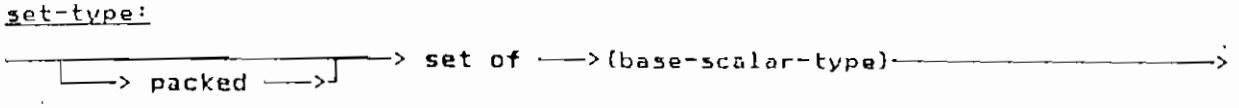
segment-module:



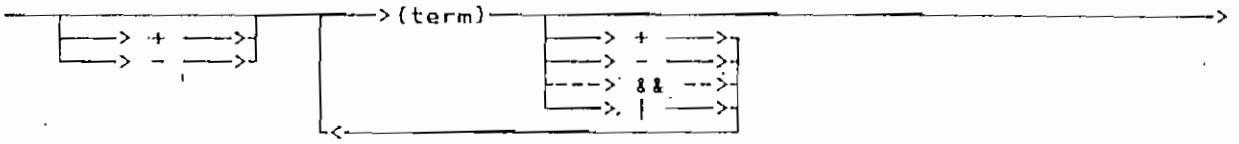
set-constructor:



set-type:

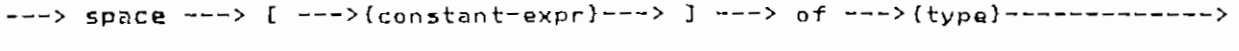


simple-expression:

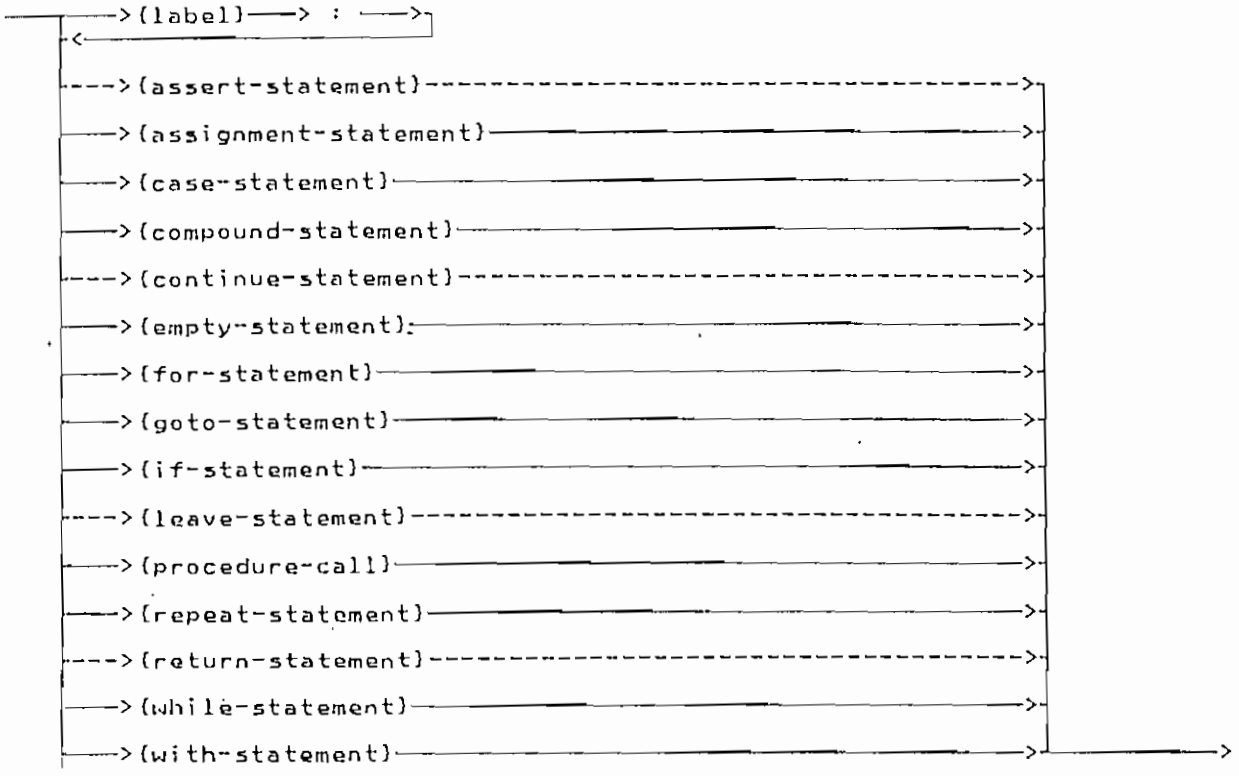


skip-statement:

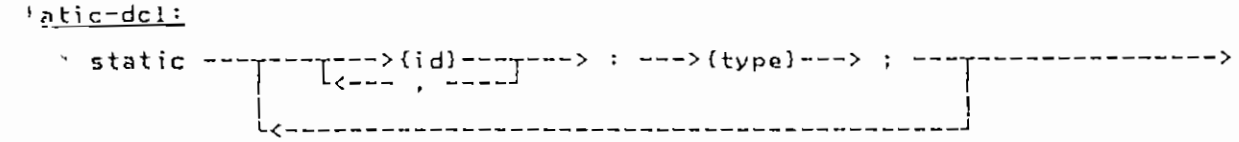
space-type:

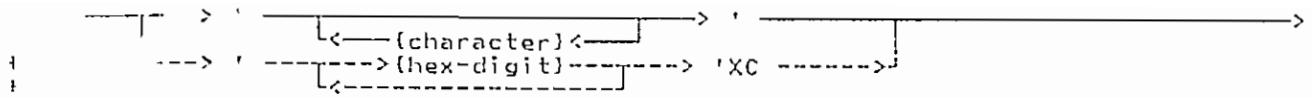


statement:

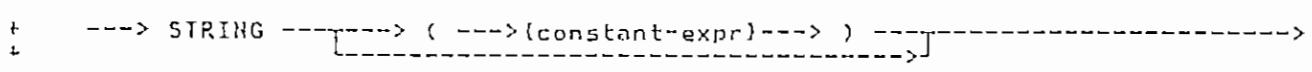


atic-dcl:

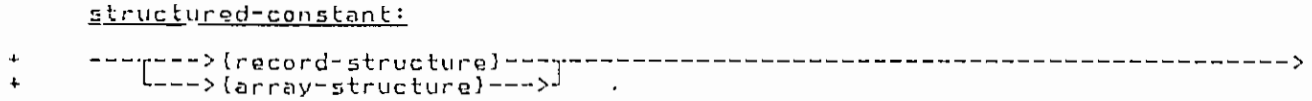




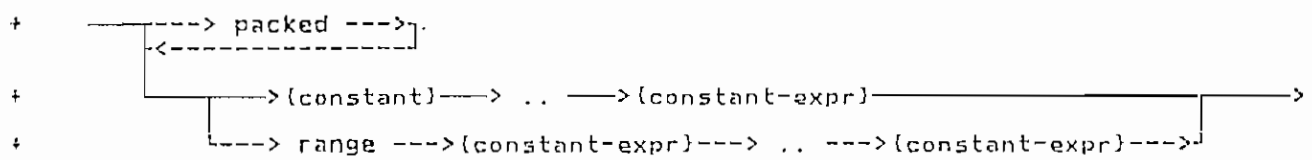
string-type:



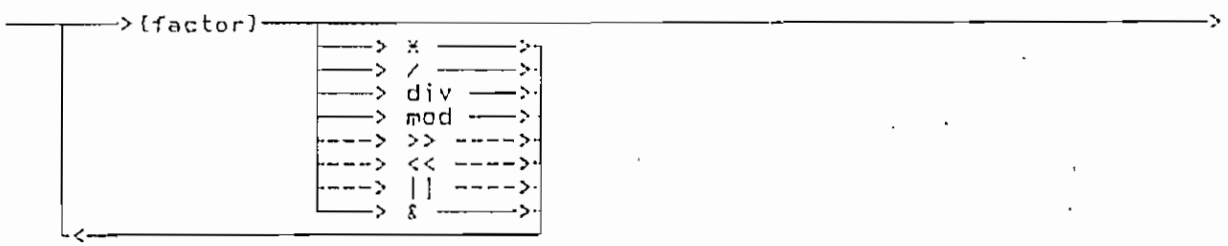
structured-constant:



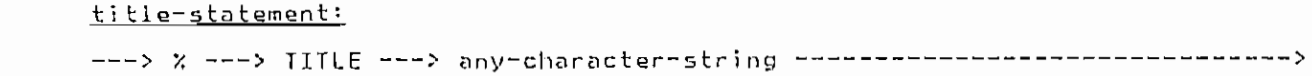
subrange-scalar-type:



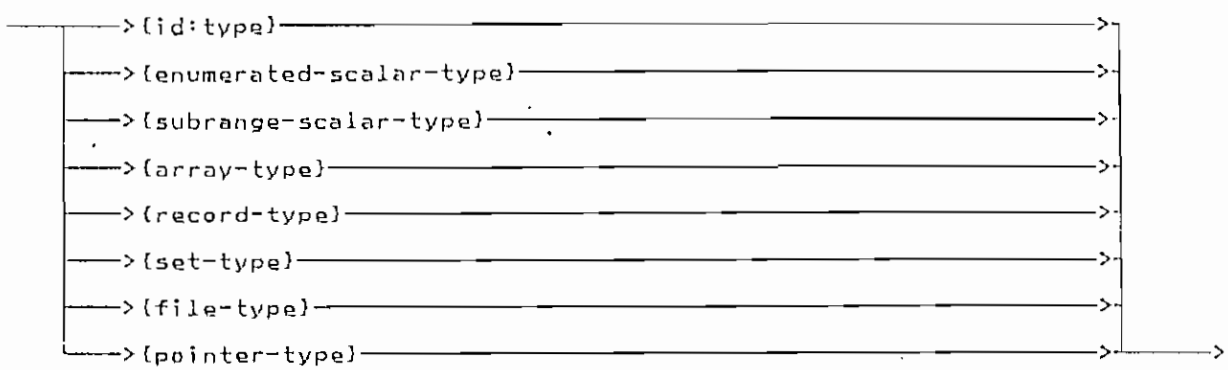
term:



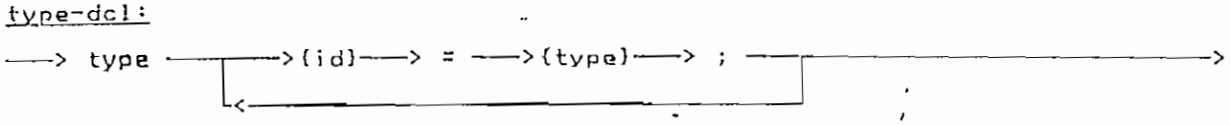
title-statement:



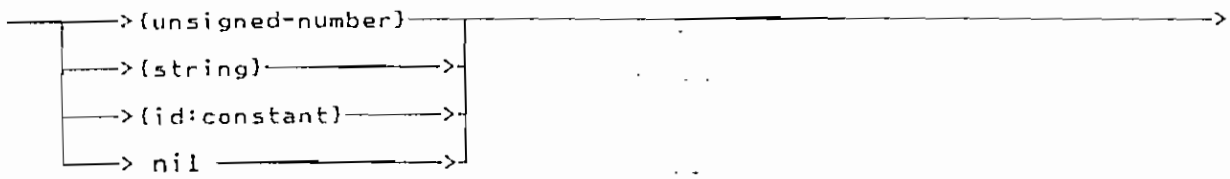
type:



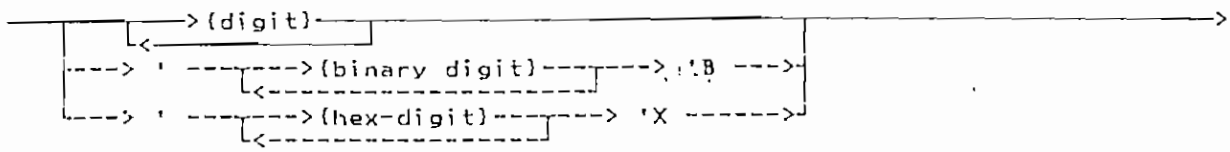
type-dcl:



unsigned-constant:

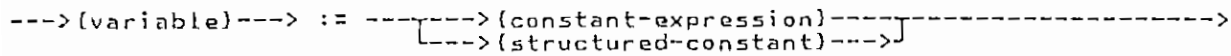


unsigned-integer:

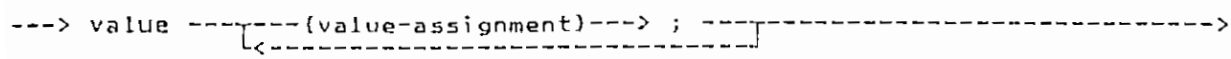


unsigned-number:

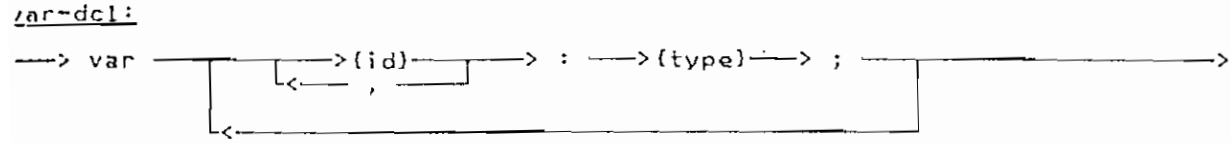
value-assignment:



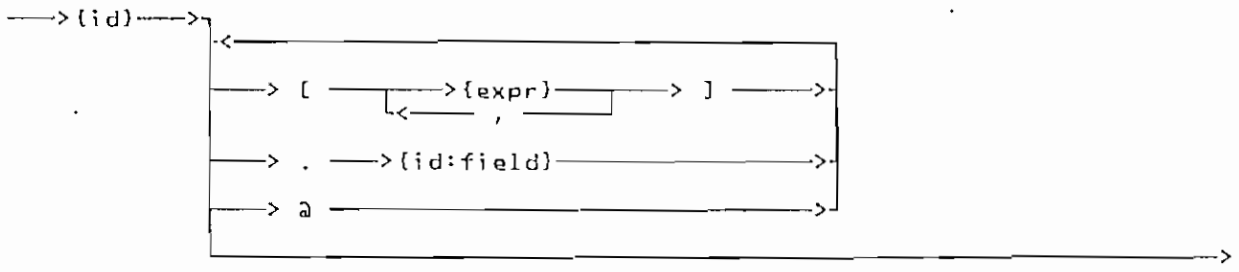
value-dcl:



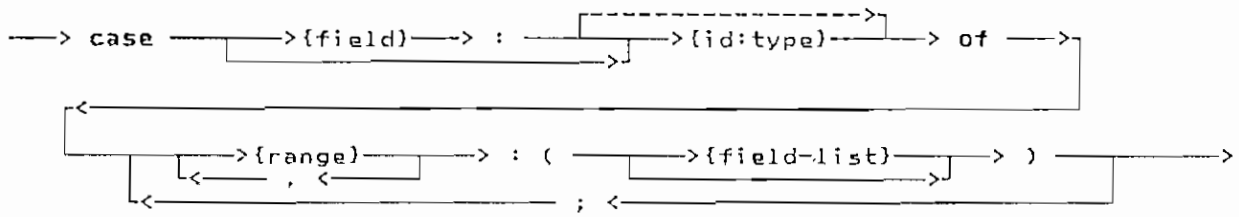
var-dcl:



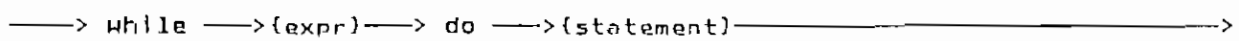
variable:



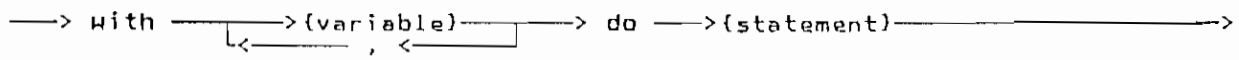
variant-part:



while-statement:



with-statement:



ANEXO 4.- INSTRUCIONES EN CODIGO ENSAMBLADOR CORRESPONDIENTES CODIGO INTERMEDIO.

CODIGO INTERM.	CODIGO ENSAMBLADOR	TIPO DE DIRECCIONAM.
	DPL = DPL+1	
CNU	LDAA # COPNUM	INMED - 2
	STAA DPL	EXTEN - 3
CVA	- si es identificador de var. numérica simple o string:	
	DPL = DPL+1	
	LDAA COPNUM	EXTEN - 3
	STAA DPL	EXTEN - 3
	- si es un arreglo:	
	DPL = DPL+1	
	LDX # CREGIX	INMED - 3
	LDAA COPNUM, X	INDEX - 2
	STAA DPL	EXTEN - 3
	- si es un string ==> para cada caracter:	
	DPL = DPL+1	
	LDAA # NUMHEX	INMED - 2
	STAA DPL	EXTEN - 3
ALM	- para variable numérica simple	
	LDAA DPL	EXTEN - 3
	STAA DPL	EXTEN - 3
	DPL = DPL-1	
	- si es un arreglo:	
	LDAA DPL	EXTEN - 3
	LDX # CREGIX	INMED - 2
	STAA COPNUM, X	INDEX - 2
	DPL = DPL-1	
	- si es un string ==> para cada caracter:	
	LDAA # NUMHEX	INMED - 2
	STAA dirección	EXTEN - 3

```
            LDAA DPL                        EXTEND - 3
OPR 13      DPL = DPL-i
 (AND)      ANDA DPL                        EXTEND - 3
            STAA DPL                        EXTEND - 3

OPR 14      DPL = DPL -40
 (**)       concatenar

            TST  DPL                        EXTEND - 3
OPR 15      BEQ  05                         RELATI - 2
 (NOT)      CLR  DPL                        EXTEND - 3
            BRA  02                         RELATI - 2
            INC  DPL                        EXTEND - 3
```

Tabla ANX4.1.- Instrucciones en ensamblador correspondientes al codigo internedio.

si OPR es:	Branch será
1	BGT
2	BGE
3	BEQ
4	BNE
5	BLE
6	BLT

Tabla ANX4.2.- Instrucciones de saltos corrspondientes a la operaciones 1 hasta la 6.

ANEXO 5 CODIGO ENSAMBLADOR M6800

CHAPITRE 11

Programmation du microprocesseur 6800

11.1 INTRODUCTION

Pour le progammeur, le système basé sur un 6800 est constitué de l'ensemble d'instructions, des accumulateurs, du registre d'index, des indicateurs, de la mémoire, des portes d'E/S, des interruptions, du pointeur de pile et de la pile.

Par commodité, nous avons reproduit, ici encore, le résumé de l'ensemble d'instructions du 6800 (voir fig. 11-1) qui comporte également des détails relatifs aux opérations particulières (sauts, branchements, interruptions et retours). L'ensemble d'instructions a été étudié de manière approfondie au chapitre 10.

Le modèle de programmation du 6800 est illustré par la figure 11-2. Il donne la liste des registres de l'UCM accessibles au programmeur ainsi que des informations sur le *CCR* (registre des codes condition),

La figure 11-3 représente l'image mémoire du système 6800 que nous utiliserons tout au long de ce chapitre. Les 512_{10} adresses les plus basses, (0000H à 01FFH), sont réservées aux programmes de l'utilisateur et aux données. Dans le cas d'une application réelle, certaines de ces cases RAM peuvent être réservées à des objectifs particuliers. Les 1024_{10} adresses les plus hautes, (FC00H à FFFFH), sont celles de la ROM. La ROM contient un programme moniteur d'initialisation du système ainsi que les vecteurs d'interruption (mémoires FFF8H à FFFFH). Elle doit contenir également de nombreux autres sous-programmes utiles à l'utilisateur. Le système 6800 est à E/S intégrées mémoire. Pour le programmeur, l'accès à un périphérique est de l'ordre d'un "rangement de l'accumulateur en mémoire 5002H" pour une sortie, et l'accès à une entrée, de l'ordre d'un "changement de l'accumulateur à partir de la mémoire 5000H". Les portes d'E/S sont traitées comme si elles étaient des cases mémoires.

Nous avons résumé au chapitre 6 les étapes de fabrication d'un programme : (1) définir et analyser le problème, (2) tracer l'organigramme de la solution, (3) écrire le programme en assembleur, (4) écrire ou générer la version en code machine du programme en assembleur, (5) mettre le programme au point et (6) documenter le programme. Toutes les techniques appliquées au mt.t. et aux 8080/8085 peuvent être appliquées au 6800. Les modifications interviennent au niveau du format de l'assembleur, des mnémoniques et des codes op. Il n'y a pas de normalisation des mnémoniques et des codes op.

Le programme source (ou partie en assembleur) du format d'instruction du 6800 est représenté figure 11-4, à droite. Le *champ d'étiquette* peut, ou non, comporter un "titre" de ligne. Le *champ de mnémonique* est parfois appelé champ de l'opérateur. Il contient les mnémoniques du 6800 qui spécifient une opération. Le *champ d'opérande* contient les informations sur le mode d'adressage et l'opérande. L'opérande peut être une valeur, une adresse ou une étiquette de référence. Le *champ de commentaires* est utilisé à l'explication de la fonction de chaque ligne du programme.

La figure 11-5 illustre plusieurs conventions d'assembleur qui sont spécifiques au 6800. Dans cet exemple, chaque ligne en assembleur représente une instruction de "chargement de l'accumulateur *A*" et, par conséquent, débute avec le même mnémonique : LDA *A*. Cependant, le mode d'adressage utilisé en chaque ligne est différent.

OPERATIONS	MNEMONIQUE	MODES D'ADRESSAGE IMMED. OP	~	#	DIRECT OP	~	#	INDEX OP	~	#	ETENDU OP	~	#	IMPLICITE OP	~	#	OPER. LOGIQUE/ARITHM. (Toutes les étiquettes de registre se réfèrent au contenu)	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				M → M	•	↕	↕	↕	R	S
	COMA													43	2	1	A → A	•	↕	↕	↕	R	S
	COMB													53	2	1	B → B	•	↕	↕	↕	R	S
Complement, 2's (Negate)	NEG							60	7	2	70	6	3				00 − M → M	•	↕	↕	↕	①	②
	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	Convertit. add. binaire de caract. DCB en format DCB	•	↕	↕	↕	↕	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	LOAA	86	2	2	96	3	2	A6	5	2	B6	4	3				M → A	•	↕	↕	↕	R	•
	LOAB	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													36	4	1	A → M_{SP}, SP − 1 → SP	•	•	•	•	•	•
	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	•	↕	↕	↕	⑥	↕
	ROLA													49	2	1	A	•	↕	↕	↕	⑥	↕
	ROLB													59	2	1	B	•	↕	↕	↕	⑥	↕
Rotate Right	ROR							66	7	2	76	6	3				M	•	↕	↕	↕	⑥	↕
	RORA													46	2	1	A	•	↕	↕	↕	⑥	↕
	RORB													56	2	1	B	•	↕	↕	↕	⑥	↕
Shift Left, Arithmetic	ASL							68	7	2	78	6	3				M	•	↕	↕	↕	⑥	↕
	ASLA													48	2	1	A	•	↕	↕	↕	⑥	↕
	ASLB													58	2	1	B	•	↕	↕	↕	⑥	↕
Shift Right, Arithmetic	ASR							67	7	2	77	6	3				M	•	↕	↕	↕	⑥	↕
	ASRA													47	2	1	A	•	↕	↕	↕	⑥	↕
	ASRB													57	2	1	B	•	↕	↕	↕	⑥	↕
Shift Right, Logic	LSR							64	7	2	74	6	3				M	•	↕	R	↕	⑥	↕
	LSRA													44	2	1	A	•	↕	R	↕	⑥	↕
	LSRB													54	2	1	B	•	↕	R	↕	⑥	↕
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	•	↕	↕	↕	↕	↕
Subtract 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
																		H	I	N	Z	V	C

LEGENDE:

OP Code op (hexa);
~ Nbre de cycles d'UC;
# Nbre d'octets de progr.;
+ Plus arithm.;
− Moins arithm.;
• ET logique;
M_{SP} Contenu de la mém. pointée par le point. de pile;
+ OU logique;
⊕ XOU logique;
M Complément de M;
→ Transféré à;
0 Bit = Zéro;
00 Octet = Zéro;
H Demi-retenue du bit 3;
I Masque d'interruption;
N Bit de signe;
Z Zéro (octet);
V Dépassem., compl. à 2;
C Retenue du bit 7;
R Réinit. toujours;
S Init. toujours;
↕ Test et init. si vrai; sinon RAZ;
• Non affecté;
CCR Registre des codes condition;
LS Le moins significatif;
MS Le plus significatif.

(*a*) Instructions accumulateur et mémoire

Fig. 11-1 L'ensemble d'instructions du 6800 (*Courtesy of Motorola, Inc.*)

OPERATIONS DE POINTEUR	MNEMONIQUE	IMMED. OP	IMMED. ~	IMMED. #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	ETENDU OP	ETENDU ~	ETENDU #	IMPLICITE OP	IMPLICITE ~	IMPLICITE #	OPER. LOGIQUE/ ARITHM.	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				$X_H - M, X_L - (M+1)$	•	•	①	!	②	•
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				$M \rightarrow X_H, (M+1) \rightarrow X_L$	•	•	③	!	R	•
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	6	2	BE	5	3				$M \rightarrow SP_H, (M+1) \rightarrow SP_L$	•	•	③	!	R	•
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3				$X_H \rightarrow M, X_L \rightarrow (M+1)$	•	•	③	!	R	•
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M, SP_L \rightarrow (M+1)$	•	•	③	!	R	•
Indx Reg → Stack Pntr	TXS													35	4	1	$X - 1 \rightarrow SP$	•	•	•	•	•	•
Stack Pntr → Indx Reg	TSX													30	4	1	$SP + 1 \rightarrow X$	•	•	•	•	•	•

(*b*) Instructions de manipulation de pile et de registre d'index

OPERATIONS	MNEMONIQUE	RELATIVE OP	RELATIVE ~	RELATIVE #	INDEX OP	INDEX ~	INDEX #	ETENDU OP	ETENDU ~	ETENDU #	IMPLICITE OP	IMPLICITE ~	IMPLICITE #	TEST BRANCH	5 H	4 I	3 N	2 Z	1 V	0 C
Branch Always	BRA	20	4	2										Aucun	•	•	•	•	•	•
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				Voir opér. part.	•	•	•	•	•	•
Jump To Subroutine	JSR				AD	8	2	BD	9	3					•	•	•	•	•	•
No Operation	NOP										01	2	1	Avanc. comm. prog. seulement	•	•	•	•	•	•
Return From Interrupt	RTI										3B	10	1		——— ① ———						
Return From Subroutine	RTS										39	5	1		•	•	•	•	•	•
Software Interrupt	SWI										3F	12	1	Voir opér. part.	•	•	•	•	•	•
Wait for Interrupt*	WAI										3E	9	1		•	②	•	•	•	•

(*c*) Instructions de saut et de branchement

OPERATIONS	MNEMONIQUE	IMPLICITE OP	IMPLICITE ~	IMPLICITE #	OPER. LOGIQUES	5 H	4 I	3 N	2 Z	1 V	0 C
Clear Carry	CLC	0C	2	1	$0 \rightarrow C$	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	2	1	$0 \rightarrow I$	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	$0 \rightarrow V$	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	$1 \rightarrow C$	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	$1 \rightarrow I$	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	$1 \rightarrow V$	•	•	•	•	S	•
Acmltr A → CCR	TAP	06	2	1	$A \rightarrow CCR$	——— ① ———					
CCR → Acmltr A	TPA	07	2	1	$CCR \rightarrow A$	•	•	•	•	•	•

NOTES CONDITION CODE REGISTRE:

(Bit Init. si test vrai et sinon RAZ)

① (Bit V) Test: Résultats = 10000000 ?

② (Bit C) Test: Résultats ≠ 00000000 ?

③ (Bit C) Test: Valeur décimale du carac. DCB MS > 9 ? (Pas RAZ si Init. précédemment)

④ (Bit V) Test: Opérande = 10000000 avant exécution?

⑤ (Bit V) Test: Opérande = 01111111 avant exécution?

⑥ (Bit V) Test: Init. égal au résultat de N ⊕ C après décal.

⑦ (Bit N) Test: Bit de signe de l'OLRS du résultat = 1 ?

⑧ (Bit V) Test: Dépas. compl. à 1 après soust. des OLPS ?

⑨ (Bit N) Test: Résultat < 0 ? (Bit 15 = 1)

⑩ (All) Charger CCR à partir de la pile (voir opér. part.)

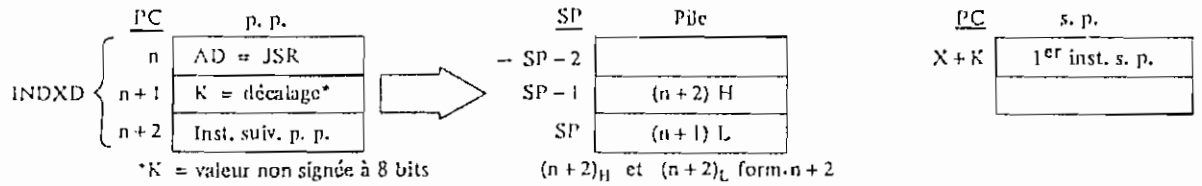
⑪ (Bit I) Init. à Interrupt. SI Init. préalablement, une interr. non masquable est nécessaire pour sortir de l'état WAIT

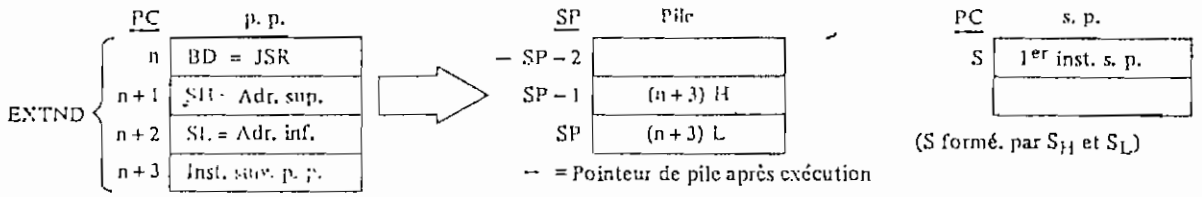
⑫ (Tous) Init. selon contenu de l'accumulateur A.

(*c*) Instructions de manipulation du *CCR*.

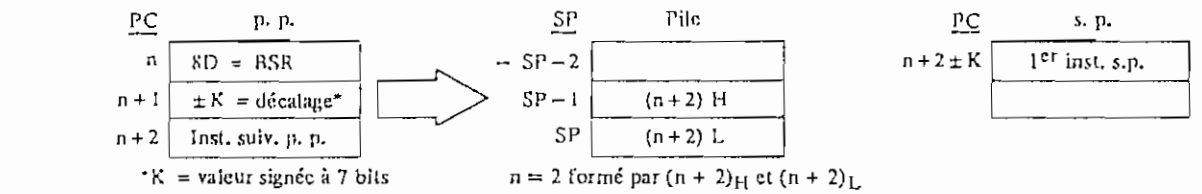
Fig. 11-1 (*suite*)

JSR, JUMP TO SUBROUTINE: (SAUT A S.P.)



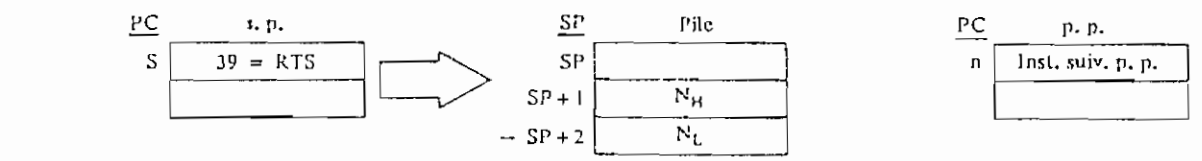


BSR, BRANCH TO SUBROUTINE: (BRANCHEMENT A S.P.)



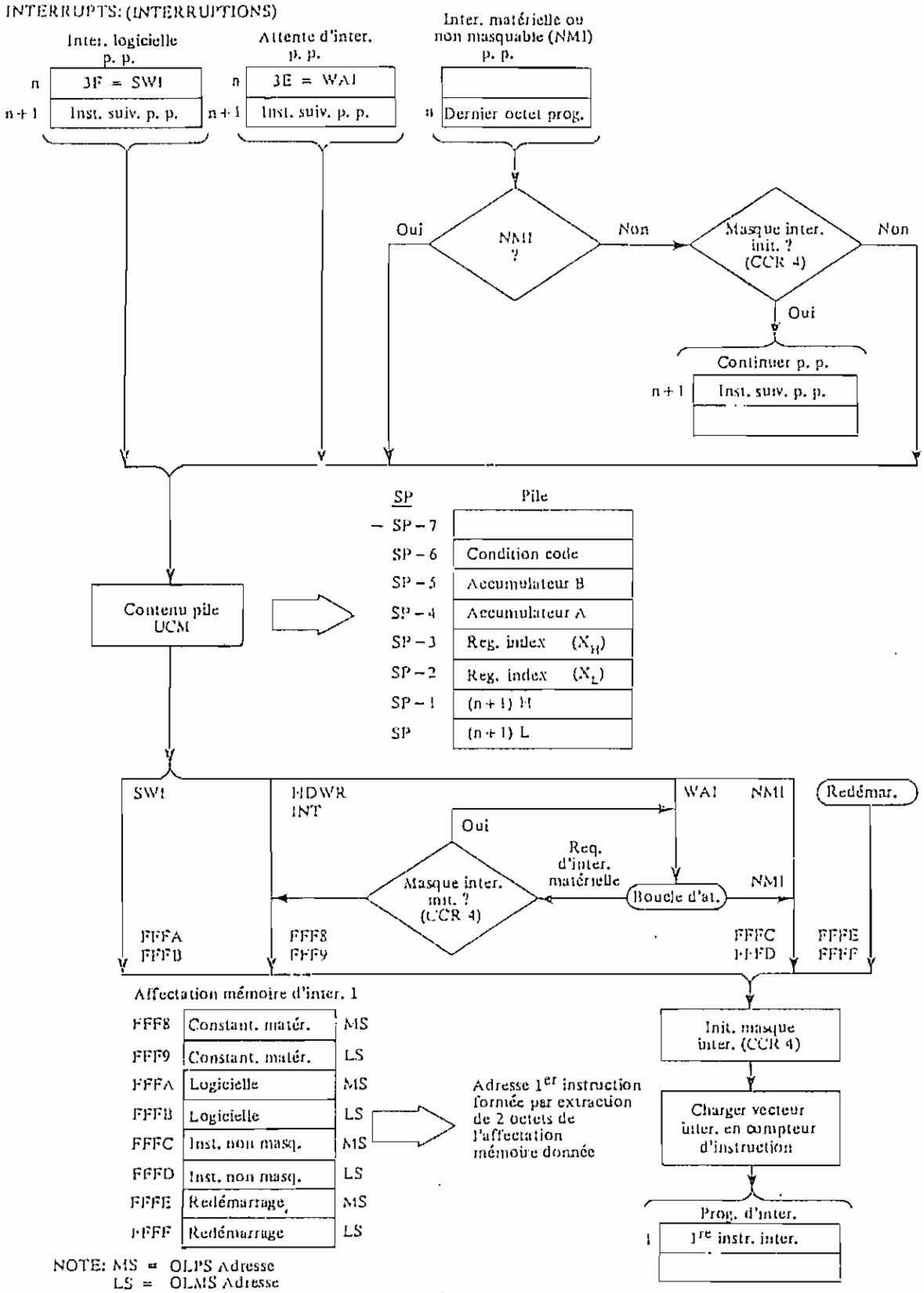
JMP, JUMP: (SAUT)

RTS, RETURN FROM SUBROUTINE: (RETOUR DE SOUS-PROGRAMME)



(*e*) Opérations particulières

Fig. 11-1 (*suite*)



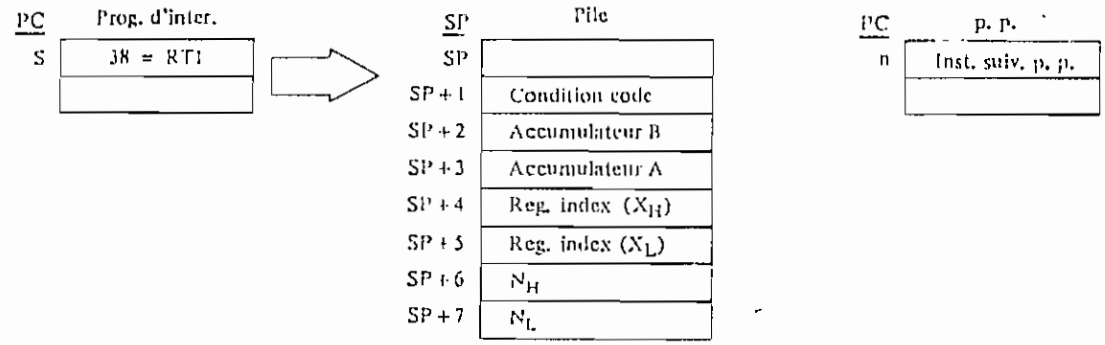
NOTE: MS = OLPS Adresse
LS = OLMS Adresse

1. Les mémoires correspondant aux adresses indiquées doivent être réservées aux vecteurs d'inter.

(*e*) Opérations particulières (*suite*)

Fig. 11-1 (*suite*)

RTI, RETURN FROM INTERRUPT:(RETOUR D'INTERRUPTION)



(*c*) (*suite*)

Fig. 11-1 (*suite*)

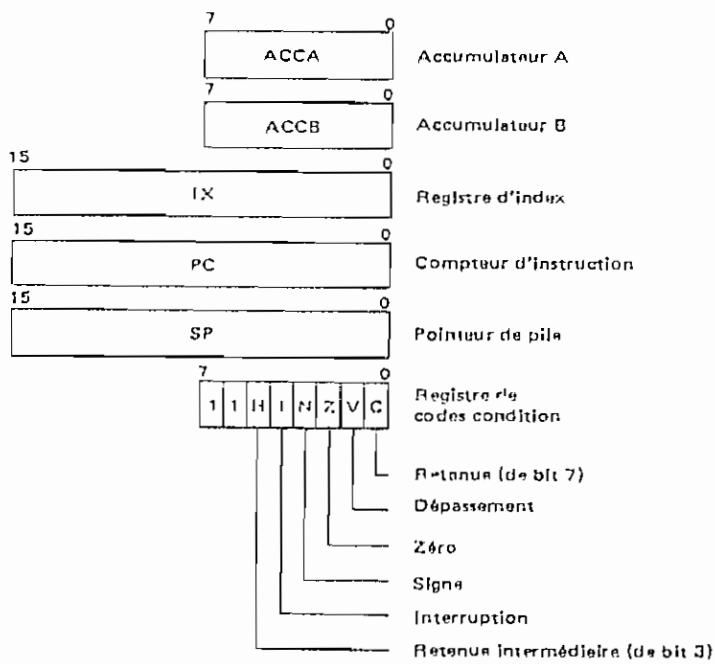


Fig. 11-2 Le modèle de programmation du 6800 (*Courtesy of Motorola, Inc.*)

La première ligne en assembleur représentée en figure 11-5 illustre une opération de "chargement immédiat de l'accumulateur *A*". Le signe # dans le champ d'opérande signifie qu'il s'agit d'un adressage immédiat. Le signe $ signifie que le nombre qui suit est un hexa.

Les seconde et troisième lignes de la figure 11-5 représentent l'opération de "chargement de l'accumulateur *A*" en adressage direct et étendu. Aucun symbole particulier n'est utilisé dans ces deux cas. Si l'opérande (adresse) est FFH ou moins, l'assembleur affecte automatiquement le code op du "chargement direct de l'accumulateur *A*". Si, par contre, l'opérande (adresse) est supérieur à FFH, l'assembleur affecte le code op du "chargement étendu de l'accumulateur *A*".

La dernière ligne de la figure 11-5 représente l'opération de "chargement indexé de l'accumulateur *A*". Le nombre 3 dans le champ d'opérande représente le décalage, et le *X* signifie que l'adressage est indexé.

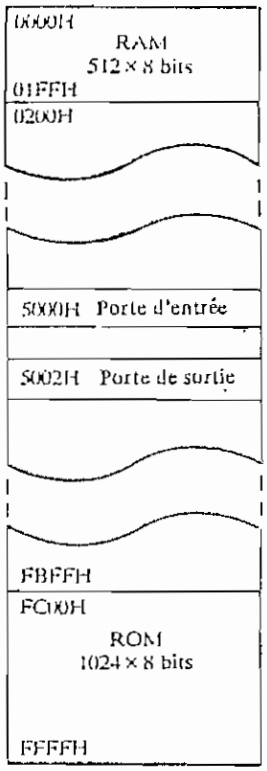


Fig. 11-3 Image mémoire du système 6800 illustré dans ce chapitre

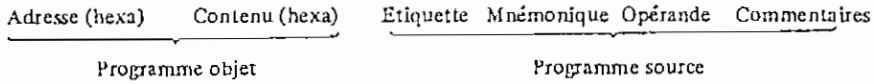


Fig. 11-4 Format des programmes source et objet du 6800

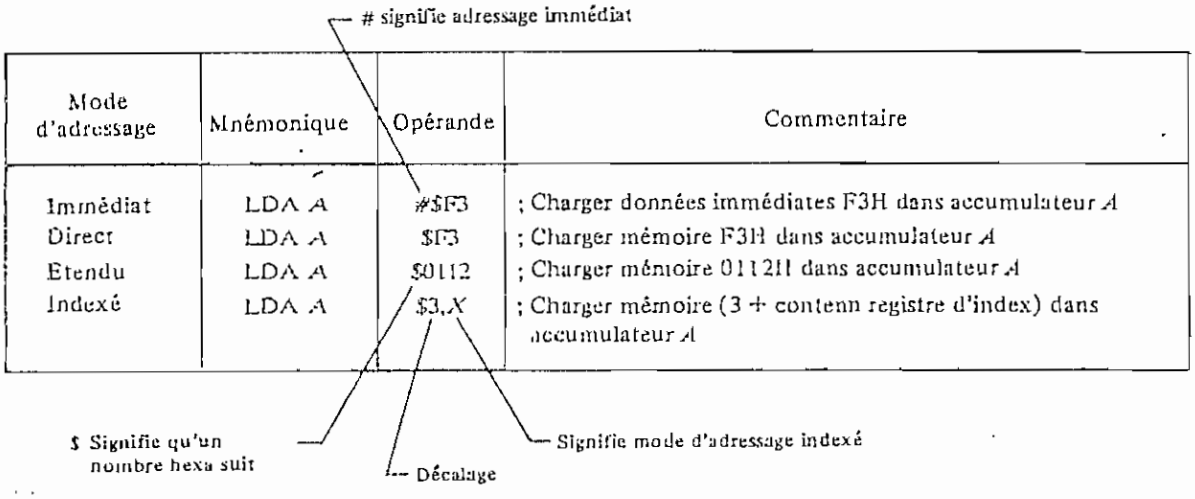


Fig. 11-5 Quelques pratiques conventionnelles en assembleur 6800

```
================================================================================
================================================================================
====                                                                        ====
====      ESCUELA POLITECNICA NACIONAL                                      ====
====      FACULTAD DE INGENIERIA ELECTRICA                                  ====
====      TESIS DE GRADO                                                    ====
====                                                                        ====
====      ALUMNO :    TORO LORGIO                                           ====
====                                                                        ====
====                                                                        ====
================================================================================
================================================================================
```



```
PROGRAM COMPILADOR ;

  CONST   NMXID = 21 ;        NMXDT = 21  ;     NMXCD = 501  ;
          NMXPR = 5  ;        NMXNP = 101 ;     NMXGT = 21   ;
          NMXPL = 501;        NMXAS = 1001;     NMXSC = 21   ;
          NMXSL = 51 ;        LMXST = 20  ;     ORIGEN= 1000 ;
          LMXHX = 5  ;
  TYPE    T1 = PACKED ARRAY (.1..10.)     OF CHAR ;
          T2 = PACKED ARRAY (.1..73.)     OF CHAR ;
          T3 = PACKED ARRAY (.1..5.)      OF CHAR ;
          T31= PACKED ARRAY (.1..NMXPR.) OF INTEGER ;
          T32= PACKED ARRAY (.1..LMXST.) OF CHAR ;
          T33= PACKED ARRAY (.1..LMXHX.) OF CHAR ;
          T30= PACKED ARRAY (.0..NMXID.) OF
               RECORD
                 NOMBRE : T1          ;
                 DIR    : INTEGER     ;
                 CASE TIPO:CHAR OF
                      'S': (SERIEHEX: PACKED ARRAY (.1..LMXST.) OF T33);
                      'A': (NSUB,SUB1,SUB2: INTEGER) ;
                      'F': (NPAR:0..NMXPR; CDFU:INTEGER; DIRP:T31)
               END ;
          T40= PACKED ARRAY (.0..NMXDT.) OF
               RECORD
                 TIPOD  : CHAR        ;
                 DATOH  : T33         ;
                 DATON  : INTEGER     ;
                 DATST  : T32
               END ;
          T51= RECORD
                 COPRDR : STRING(3)   ;
                 TIPVAR : CHAR        ;
                 CTOPND : CHAR        ;
                 COPHEX : T33         ;
                 COPNUM : INTEGER     ;
                 CREGIX : INTEGER
```

```
            END ;
        T50= PACKED ARRAY (.0..NMXCD.) OF T51 ;
        T60= PACKED ARRAY (.0..NMXNP.) OF
             RECORD   NPNUP,NPCCD: INTEGER   END;
        T61= PACKED ARRAY (.0..NMXCD.) OF
             RECORD CINUM, CICME:INTEGER END    ;
        T70= PACKED ARRAY (.0..NMXGT.) OF
             RECORD GTNUP, GTCCD:INTEGER END    ;
        T71= PACKED ARRAY (.0..NMXSL.) OF
             RECORD SLNUM, SLCAS:INTEGER END    ;
        T81 = RECORD
                CMEMOR : INTEGER   ;
                OPRDOR : STRING(4) ;
                TIPDIR : CHAR      ;
                TIPOPN : CHAR      ;
                OPNHEX : T33       ;
                OPNNUM : INTEGER   ;
                INDICE : STRING(2)
              END ;
        T80 = PACKED ARRAY (.0..NMXAS.) OF T81    ;
        T90= PACKED ARRAY (.0..NMXSC.) OF INTEGER ;
VAR     I                : INTEGER   ;      CTRAUX: CHAR    ;
        PANTALLA,ENTRADA : TEXT      ;
DEF     CC,LL,NUMERO     : INTEGER   ;      CTR    : CHAR    ;
        SALIDA,FUENTE    : TEXT      ;      PLI    : BOOLEAN ;
        CLASE            : STRING(5) ;      HUBOERR: BOOLEAN ;
        LINEA,LINEAUX    : T2        ;      CMI,CM : INTEGER ;
        NAMEPROG,SIMB    : T1        ;      NUMHEX : T33     ;
        SERIE            : T32       ;      TPCONS : CHAR    ;
                                            TPEXPRE: CHAR    ;
        DPL              : 0..NMXPL  ;      RCDINT : T51     ;
        CID              : 0..NMXID  ;      TIDENT : T31     ;
        CDT              : 0..NMXDT  ;      TDATOS : T41     ;
        CCD              : 0..NMXCD  ;      TCODIG : T51     ;
        CNP              : 0..NMXNP  ;      TNUPRO : T61     ;
        CIN              : 0..NMXCD  ;      TNUCIN : T61     ;
        CGT              : 0..NMXGT  ;      TGOTOS : T71     ;
        CSL              : 0..NMXSL  ;      TSALTO : T71     ;
        CAS              : 0..NMXAS  ;      TASSEM : T81     ;
        CSC              : 0..NMXSC  ;      TSACND : T91     ;

PROCEDURE OBSIMB         ; EXTERNAL ;
PROCEDURE ANALISIS       ; EXTERNAL ;
PROCEDURE CODIGASS       ; EXTERNAL ;
PROCEDURE ORDENARDATOS   ; EXTERNAL ;

BEGIN /* COMPILADOR */
CC:=0          ;  LL:=0      ;     NUMERO:=0     ;     CTR:=' '   ;
PLI:=FALSE     ;  CLASE:=' ';      SIMB:=' '     ;     LINEA:=' ';
LINEAUX:=' '   ;  DPL:=0     ;     HUBOERR:=FALSE;     NUMHEX:=' ';
NAMEPROG:=' ';  CMI:=3       ;     CM:=ORIGEN    ;     TPCONS:=' ';
SERIE  ':=' '; TPEXPRE:=' ';
RCDINT.COPRDR        := ' ' ;
TIDENT (.0.).NOMBRE  := ' ' ;      CID:= 0       ;
TDATOS (.0.).TIPOD   := ' ' ;      CDT:= 0       ;
```

```
TCODIG (.0.).COPRDR := ' ' ;      CCD:= 1      ;
TNUPRO (.0.).NPNUP  := 0   ;      CNP:= 0      ;
TGOTOS (.0.).GTNUP  := 0   ;      CGT:= 0      ;
TASSEM (.0.).CMEMOR := 0   ;      CAS:= 0      ;
TNUCIN (.0.).CINUM  := 0   ;      CIN:= 0      ;
TSALTO (.0.).SLNUM  := 0   ;      CSL:= 0      ;
TSACND (.0.)       := 0   ;      CSC:= 0      ;

WRITELN (SALIDA,'==================================',
                '==================================');
WRITELN (SALIDA,'==================================',
                '==================================');
WRITELN (SALIDA,'===                                ',
                '                                ===');
WRITELN (SALIDA,'===        ESCUELA POLITECNICA NACI',
                'ONAL                            ===');
WRITELN (SALIDA,'===        FACULTAD DE INGENIERIA E',
                'LECTRICA                        ===');
WRITELN (SALIDA,'===        ESPECIALIZACION: ELECTRO',
                'NICA Y TELECOMUNICACIONES       ===');
WRITELN (SALIDA,'===                               ',
                '                                ===');
WRITELN (SALIDA,'===        TESIS DE GRADO: COMPILAD',
                'OR DE BASIC A ASSEMBLER         ===');
WRITELN (SALIDA,'===                          DEL MICR',
                'OPROCESADOR M6800               ===');
WRITELN (SALIDA,'===                               ',
                '                                ===');
WRITELN (SALIDA,'===        ALUMNO: JORGIO VICENTE T',
                'ORO R.                          ===');
WRITELN (SALIDA,'===        DIRECTOR: ING. CESAR ESC',
                'UETINI                          ===');
WRITELN (SALIDA,'===        QUITO 1987              ',
                '                                ===');
WRITELN (SALIDA,'===                                ',
                '                                ===');
WRITELN (SALIDA,'==================================',
                '==================================');
WRITELN (SALIDA,'==================================',
                '==================================');

ORDENARDATOS    ;
CC:=0;  LL:=0   ;
RESET (INPUT)   ;
OBSIMB          ;
ANALISIS        ;

WRITELN (SALIDA,'---> IDENTIFICADORES EN EL PROGRAMA FUENTE');
WRITELN (SALIDA);   WRITELN (SALIDA);
WRITELN (SALIDA,'    ------------------------------  ');
WRITELN (SALIDA,'      NOMBRE    TIPO  DIRECCION    ');
WRITELN (SALIDA,'    ------------------------------  ');

I:=1 ;
WHILE I<=CID DO
```

```
    BEGIN /* B1 */
    WITH TIDENT[.I.] DO   WRITELN(SALIDA,  NOMBRE:17, TIPO:1, DIR:8) ;
    I:=I+1
    END /* B1 */ ;
  WRITELN (SALIDA,'      ------------------------          ');

  TERMIN  (ENTRADA ) ;
  TERMOUT (PANTALLA) ;
  WRITELN (PANTALLA);   WRITELN (PANTALLA);   WRITELN (PANTALLA);
  WRITELN (PANTALLA, '---> DESEA LISTAR EL CODIGO INTERMEDIO? (Y/N)');
  READLN  (ENTRADA,  CTRAUX);
  IF (CTRAUX='Y') THEN
  BEGIN /* ESCRIBA CODIGO INTERMEDIO */
  WRITELN (SALIDA);   WRITELN (SALIDA);   WRITELN (SALIDA);

  WRITELN (SALIDA,'---> PROGRAMA FUENTE TRADUCIDO A CODIGO INTERMEDIO');
  WRITELN (SALIDA);   WRITELN (SALIDA);
  WRITELN (SALIDA,'      ------------------------',
                  '-------------------------      ');
  WRITELN (SALIDA,'        CCD  OPERADOR  OPERANDO ',
                  '    OPERANDO  REGISTRO         ');
  WRITELN (SALIDA,'                              HEXADECIMAL',
                  'AL  NUMERICO  INDICE          ');
  WRITELN (SALIDA,'      ------------------------',
                  '-------------------------      ');
  WRITELN (SALIDA);

  I:=1 ;
  WHILE I<CCD  DO
    BEGIN
    WITH TCODIGI[.I.] DO
    WRITELN (SALIDA, I:8, COPRDR:8, COPHEX:12, CCPNUM:12, CREGIX:10);
    I:=I+1
    END ;
  WRITELN (SALIDA,'      ------------------------',
                  '-------------------------      ');
  WRITELN (SALIDA);   WRITELN (SALIDA);
  END /* ESCRIBA CODIGO INTERMEDIO */
  ELSE

  WRITELN (SALIDA);   WRITELN (SALIDA);

  IF (HUBOERR) THEN
    BEGIN
    WRITELN (SALIDA,'---> LA COMPILACION SE DETIENE POR ERRORES');
    WRITELN (SALIDA,'     EN EL PROGRAMA FUENTE                ');
    END
  ELSE
    BEGIN /* ELSE */
    CODIGASS ;
    WRITELN (SALIDA,'---> PROGRAMA OBJETO EN LENGUAJE ENSAM-    ');
    WRITELN (SALIDA,'     BLADOR DEL MICROPROCESADOR M6800      ');
    WRITELN (SALIDA);   WRITELN (SALIDA);
    WRITELN (SALIDA,'      ------------------------------------');
    WRITELN (SALIDA,'          CAS  CMEMOR  OPERADOR  OPERANDO  ');
```

```
    WRITELN (SALIDA,'      ----------------------------------------');
    WRITELN (SALIDA);
    WRITELN (SALIDA, '1':12, 'NAM':15, ' ':5, NAMEPROG:-5          );
    WRITELN (SALIDA, '2':12, 'ORG':15, ' ':5, ORIGEN:-5            );

    I:=1;
    WHILE I<=CAS DO
      BEGIN
      WITH TASSEM(.I.) DO
      IF TIPOPN='N' THEN
      WRITELN (SALIDA, (I+2):12, CMEMOR:7, OPRDOR:9,TIPJIR:5, OPNNUM:-5,
                       INDICE:2)
      ELSE
      WRITELN (SALIDA, (I+2):12, CMEMOR:7, OPRDOR:9,TIPJIR:5, OPNHEX:-5,
                       INDICE:2) ;
      I:=I+1
      END ;
    WRITELN (SALIDA,(I+2):12, CM:7     ,'SWI ':9);
    WRITELN (SALIDA,(I+3):12, (CM+1):7,'END ':9);
    WRITELN (SALIDA,'      ----------------------------------------');

    WRITELN (SALIDA);   WRITELN (SALIDA);   WRITELN (SALIDA);
    WRITELN (SALIDA,'---> LA COMPILACION FUE EXITOSA               ');
    END /* ELSE */ ;

  WRITELN(SALIDA);   WRITELN(SALIDA);   WRITELN(SALIDA);
  WRITELN(SALIDA);   WRITELN(SALIDA);   WRITELN(SALIDA);
  WRITELN (PANTALLA, '---> SUS RESULTADOS ESTAN EL EN ARCHIVO "FILE ',
                     'SALIDA".');
  WRITELN(SALIDA);   WRITELN(SALIDA);   WRITELN(SALIDA)
  END /* COMPILADOR */ .
```

```
==========================================================================
==========================================================================
====                                                                  ===
====      ESCUELA POLITECNICA NACIONAL                                ===
====      FACULTAD DE INGENIERIA ELECTRICA                            ===
====      TESIS DE GRADO                                              ===
====                                                                  ===
====      ALUMNO :   TORO LORGIO                                      ===
====                                                                  ===
====                                                                  ===
==========================================================================
==========================================================================
```

```
SEGMENT ORDENARDATOS ;

PROCEDURE ORDENARDATOS          ; EXTERNAL ;
PROCEDURE OBSIMB                ; EXTERNAL ;
PROCEDURE FINLINE(FL:INTEGER)   ; EXTERNAL ;

PROCEDURE ORDENARDATOS ;

  CONST   NMXNP= 101 ;          VMXNP = 99999 ;
  TYPE    T1 = PACKED ARRAY (.1..10.) OF CHAR ;
          T2 = PACKED ARRAY (.1..73.) OF CHAR ;
  VAR     I,IAUX,NAUX : INTEGER ;
          TODOBIEN    : BOOLEAN ;
          NUML        : PACKED ARRAY (.0..NMXNP.) OF INTEGER ;
          LAR         : PACKED ARRAY (.0..NMXNP.) OF T2      ;
          LAUX        : T2      ;
  REF     NUMERO      : INTEGER ;      SALIDA,FUENTE : TEXT ;
          PLI         : BOOLEAN ;      CLASE    : STRING(5) ;
          LINEA       : T2      ;      LINEAUX  : T2        ;

  BEGIN /* ORDENARDATOS */

  WRITELN(SALIDA);   WRITELN(SALIDA);   WRITELN(SALIDA);
  WRITELN (SALIDA,'---> PROGRAMA FUENTE ORIGINAL    ');
  WRITELN(SALIDA);  WRITELN(SALIDA);
  RESET (FUENTE);   REWRITE (INPUT);
  WHILE NOT EOF(FUENTE) DO
    BEGIN   READLN (FUENTE,LINEA);  WRITELN (INPUT, LINEA);
            WRITELN(SALIDA, LINEA)
    END;
  WRITELN (INPUT, 'E FIN DE DATOS') ;
  RESET (INPUT);     WRITELN(SALIDA);   WRITELN(SALIDA);
  WRITELN(SALIDA);   WRITELN(SALIDA);   WRITELN(SALIDA);
  TODOBIEN:= TRUE;   I:= 0        ;     IAUX:= 0       ;
  NAUX:= 0        ;  NUML(.I.):= 0 ;    LAUX:= ' '     ;
  LAR(.1.):= ' ' ;
```

```
OBSIMB ;
WHILE NOT EOF(INPUT) DO
  BEGIN /* B1 */
  I:= I+1;    LINEAUX:= LINEA;    LAR(.I.):= LINEA;   PLI:= FALSE;
  IF (CLASE<>'ENTER') THEN
    BEGIN
    WRITELN(SALIDA)  ;        I:= I-1   ;
    TODOBIEN:= FALSE ;        FINLINE(1)
    END         ELSE
  IF (NUMERO>VMXNP) THEN BEGIN TODOBIEN:=FALSE; FINLINE(93) END ELSE
    BEGIN /* B2 */
    NUML(.I.):= NUMERO;       IAUX:= I  ;
    IF (NUML(.IAUX.) < NUML(.IAUX-1.)) THEN TODOBIEN:= FALSE;
    WHILE (NUML(.IAUX.) < NUML(.IAUX-1.))   DO
      BEGIN
      LAUX              := LAR(.IAUX.)       ;
      LAR(.IAUX.)       := LAR(.(IAUX-1).)   ;
      LAR(.(IAUX-1).)   := LAUX              ;
      NAUX              := NUML(.IAUX.)      ;
      NUML(.IAUX.)      := NUML(.(IAUX-1).)  ;
      NUML(.(IAUX-1).):= NAUX                ;
      IAUX              := IAUX-1
      END ;
    IF (NUML(.IAUX.) = NUML(.(IAUX-1).)) THEN
      BEGIN
      FINLINE(2);         TODOBIEN:= FALSE ;
      WHILE (IAUX <= 1) DO
        BEGIN
        LAR (.(IAUX-1).) := LAR (.IAUX.) ;
        NUML(.(IAUX-1).) := NUML(.IAUX.)  ;
        IAUX             := IAUX+1
        END   ;
      I:= I-1 ;
      END
    END /* B2 */ ;
  WHILE NOT PLI DO OBSIMB ;
  END /* B1 */ ;

IF TODOBIEN THEN
  BEGIN
  WRITELN(SALIDA);    WRITELN(SALIDA);          WRITELN(SALIDA);
  WRITELN(SALIDA,'---> EL PROGRAMA A COMPILARSE SERA EL ');
  WRITELN(SALIDA,'     ORIGINALMENTE INGRESADO.');
  END
ELSE
  BEGIN /* B1 */
  WRITELN(SALIDA);    WRITELN(SALIDA);          WRITELN(SALIDA);
  WRITELN(SALIDA,'---> EL PROGRAMA FUENTE ORIGINAL TUVO LINEAS ',
                 'SIN NUMERO ')                                  ;
  WRITELN(SALIDA,'     DE PROPOSICION O BIEN CON NUMEROS REPETI',
                 'DOS  O NO') ;
  WRITELN(SALIDA,'     HUBO ORDEN ASCENDENTE.   POR LO TANTO EL',
                 'PROGRAMA ') ;
  WRITELN(SALIDA,'     FUENTE A COMPILARSE SERA EL SIGUIENTE: ');
  WRITELN(SALIDA);    WRITELN(SALIDA);          WRITELN(SALIDA);
```

FILE: ORDENARD PASCAL A1 VM/SP RELEASE 3.1 EXPRESS PUT8401+ SLU301

```
   IAUX:= 0        ;     REWRITE(INPUT) ;                          289
   WHILE (IAUX < I) DO
     BEGIN
     IAUX:= IAUX+1;
     WRITELN (INPUT , LAR(.IAUX.)) ;
     WRITELN (SALIDA, LAR(.IAUX.))
     END ;
   WRITELN (INPUT, '& FIN DE DATOS')
   END /* B1 */    ;
 WRITELN(SALIDA)  ;     WRITELN(SALIDA);              WRITELN(SALIDA);
END /* ORDENARDATOS */ ;
```

```
==========================================================================
==========================================================================
====                                                                  ===
====      ESCUELA POLITECNICA NACIONAL                                ===
====      FACULTAD DE INGENIERIA ELECTRICA                            ===
====      TESIS DE GRADO                                              ===
====                                                                  ===
====      ALUMNO :    TORO LORGIO                                     ===
====                                                                  ===
====                                                                  ===
==========================================================================
==========================================================================
```



```
SEGMENT ERROR ;

PROCEDURE ERROR (NE:INTEGER);  EXTERNAL;


PROCEDURE ERROR ;
  REF      SALIDA: TEXT;   HUBOERR: BOOLEAN ;

  BEGIN /* ERROR */
  HUBOERR:= TRUE ;
  CASE NE OF
    0 : WRITELN (SALIDA, '  ')                                          ;
    1 : WRITELN (SALIDA, '-> ERROR: FALTA NUMERO DE LINEA')             ;
    2 : WRITELN (SALIDA, '-> NUMERO DE PROPOSICION REPETIDO')           ;
    3 : WRITELN (SALIDA, '-> CARACTER "|","¬","@ " ", SON INVALIDOS'    ,
                         ' EN UN STRING')                               ;
    4 : WRITELN (SALIDA, '-> PARAMETRO DE FUNCION DEBE SER NUMERICO')   ;
    5 : WRITELN (SALIDA, '-> SE ESPERABA UN SUBINDICE')                 ;
    6 : WRITELN (SALIDA, '-> SUBINDICE INCORRECTO O FALTA ")"' )        ;
    7 : WRITELN (SALIDA, '-> SE ESPERABA COMA LUEGO DE SUBINDICE')      ;
    8 : WRITELN (SALIDA, '-> SE ESPERABA ")"  LUEGO DE SUBINDICE')      ;
    9 : WRITELN (SALIDA, '-> SE ESPERABA PARAMETRO DE FUNCION')         ;
   10: WRITELN (SALIDA, '-> PARAMETRO INVALIDO O FALTA ")"')            ;
   11: WRITELN (SALIDA, '-> SE ESPERABA "," O ")" LUEGO DE PARAMETRO');
   12: WRITELN (SALIDA, '-> SE ESPERABA EXPRESION LUEGO DE"(" ')        ;
   13: WRITELN (SALIDA, '-> EXPRESION INCORRECTA ENTRE PARENTESIS '     ,
                        'O FALTA ")" ')                                 ;
   14: WRITELN (SALIDA, '-> SE ESPERABA ")" LUEGO DE EXPRESION')        ;
   15: WRITELN (SALIDA, '-> UNA EXPRESION NO PUEDE EMPEZAR CON ESTE'    ,
                        ' SIMBOLO')                                     ;
   16: WRITELN (SALIDA, '-> TIPO INCORRECTO EN EXPRESION  ')            ;
   17: WRITELN (SALIDA, '-> SE ESPERABA TERMINO LUEGO DE OPERADOR ')    ;
   18: WRITELN (SALIDA, '-> EXPRESION INCORRECTA O BIEN SE ESPERABA'    ,
                        ' OPERADOR DE RELACION')                        ;
   19: WRITELN (SALIDA, '-> SE ESPERABA EXPRESION LUEGO DE OPERADOR '   ,
```

```
                                'DE RELACION')                          ;
   20: WRITELN (SALIDA, '-> CONDICION INCORRECTA, LUEGO DE EXPRESION',  ,
                        ' DEBE IR UN OPERADOR DE RELACION')             ;
   21: WRITELN (SALIDA, '-> UNA CONSTANTE NO PUEDE EMPEZAR CON ESTE',   ,
                        ' SIMBOLO')                                     ;
   22: WRITELN (SALIDA, '-> TIRA DE CARACTERES INCORRECTA')             ;
   23: WRITELN (SALIDA, '-> TIPOS DE DATO Y VARIABLE INCONGRUENTES')    ;
   24: WRITELN (SALIDA, '-> SE ESPERABA UNA CONSTANTE LUEGO DE "+" ',   ,
                        'O LUEGO DE "-" ')                              ;
   25: WRITELN (SALIDA, '-> SE ESPERABA EXPRESION LUEGO DE OPERADOR ',  ,
                        'LOGICO')                                       ;
   26: WRITELN (SALIDA, '-> EL NUMERO DE SUBINDICES DEL ARREGLO NO ',   ,
                        'CORRESPONDE A SU DECLARACION')                 ;
   27: WRITELN (SALIDA, '-> IDENTIFICADOR NO FUE DECLARADO COMO ',      ,
                        'ARREGLO')                                      ;

   30: WRITELN (SALIDA, '-> IDENTIFICADIR DEBE SER VARIABLE SIMPLE')    ;
   31: WRITELN (SALIDA, '-> PRIMER PROPOSICION DEBE SER "PROGRAM"')     ;
   32: WRITELN (SALIDA, '-> ULTIMA PROPOSICION DEBE SER "END"')         ;
   33: WRITELN (SALIDA, '-> SE ESPERABA PALABRA RESERVADA')             ;
   34: WRITELN (SALIDA, '-> SIMBOLO INDEFINIDO O FUERA DE CONTEXTO')    ;
   35: WRITELN (SALIDA, '-> SE ESPERABA PROPOSICIONES LUEGO DE ',       ,
                        'FUNCION O SUBRUTINA')                          ;
   36: WRITELN (SALIDA, '-> SE ESPERABA CONSTANTE LUEGO DE COMA O ',    ,
                        'LUEGO DE PALABRA RESERVADA')                   ;
   37: WRITELN (SALIDA, '-> A UNA CONSTANTE DEBE SEGUIR UNA COMA')      ;
   38: WRITELN (SALIDA, '-> SE ESPERABA IDENT. DE VARIABLE LUEGO DE ',  ,
                        'COMA O LUEGO DE PALABRA RESERVADA')            ;
   39: WRITELN (SALIDA, '-> SIMBOLO LUEGO DE PAL. RESERVADA O LUEGO',   ,
                        ' DE COMA DEBE SER IDENT. DE VARIABLE')         ;
   40: WRITELN (SALIDA, '-> A UN IDENT.DE VARIABLE DEBE SEGUIR COMA');  ;
   41: WRITELN (SALIDA, '-> FALTA DIMENSIONAR VARIABLE')                ;
   42: WRITELN (SALIDA, '-> SE ESPERABA "(" LUEGO DE ID.DE VARIABLE');  ;
   43: WRITELN (SALIDA, '-> SUBINDICE DEBE SER ENTERO')                 ;
   44: WRITELN (SALIDA, '-> SE ESPERABA ","O ")"')                      ;
   45: WRITELN (SALIDA, '-> SIMBOLO LUEGO DE ")" DEBE SER COMA')        ;
   46: WRITELN (SALIDA, '-> SIMBOLO LUEGO DE SUBINDICE DEBE SER ',      ,
                        'COMA O ")"')                                   ;
   47: WRITELN (SALIDA, '-> SE ESPERABA PROPOSICION "ENDFN"')           ;
   48: WRITELN (SALIDA, '-> SE ESPERABA PROPOSICION "RETURN"')          ;
   49: WRITELN (SALIDA, '-> SE ESPERABA IDENTIFICADOR DE FUNCION')      ;
   50: WRITELN (SALIDA, '-> SIMBOLO LUEGO DE "DEF" DEBE SER IDENT.',    ,
                        'DE FUNCION')                                   ;
   51: WRITELN (SALIDA, '-> SIMBOLO LUEGO DE ID.DE FUNCION DEBE SER',   ,
                        ' "("')                                         ;
   52: WRITELN (SALIDA, '-> SE ESPERABA PARAMETRO DE FUNCION')          ;
   53: WRITELN (SALIDA, '-> PARAMETRO DEBE SER ID.DE VARIABLE SIMPLE');  ;
   54: WRITELN (SALIDA, '-> SIMBOLO LUEGO DE PARAMETRO DEBE SER ',      ,
                        'COMA O ")"')                                   ;
   55: WRITELN (SALIDA, '-> SIMBOLOS INVALIDOS LUEGO DE ")"')           ;
   56: WRITELN (SALIDA, '-> SE ESPERABA PROPOSICION "NEXT"')            ;
   57: WRITELN (SALIDA, '-> SE ESPERABA PROPOSICION "ENDIF"')           ;
   58: WRITELN (SALIDA, '-> VARIABLE DEBE SER NUMERICA')                ;
   59: WRITELN (SALIDA, '-> SIMBOLO INVALIDO LUEGO DE PAL.RESERVADA');  ;
   60: WRITELN (SALIDA, '-> PROPOSICION INCOMPLETA')                    ;
```

```
61: WRITELN (SALIDA, '-> SIMBOLO,LUEGO DE IDENT. DE VARIABLE O DE',
                     ' FUNCION, DEBE SER "="')                    ;
62: WRITELN (SALIDA, '-> EXPRESION INCORRECTA O PROP. INCOMPLETA');
63: WRITELN (SALIDA, '-> LUEGO DE EXPRESION DEBE IR PALABRA "TO"');
64: WRITELN (SALIDA, '-> SIMBOLOS INVALIDOS LUEGO DE EXPRESION')  ;
65: WRITELN (SALIDA, '-> SIMBOLO INCORRECTO LUEGO DE IDENTIFICADOR',
                     ' DE VARIABLE')                              ;
66: WRITELN (SALIDA, '-> SIMBOLO, LUEGO DE PAL. RESERVADA DEBE ' ,
                     'SER ENTERO')                                ;
67: WRITELN (SALIDA, '-> SIMBOLO INCORRECTO LUEGO DE ENTERO')    ;
68: WRITELN (SALIDA, '-> CONDICION INCORRECTA O PROP. INCOMPLETA');
69: WRITELN (SALIDA, '-> SIMBOLO, LUEGO DE CONDICION DEBE SER ' ,
                     'PALABRA "THEN"')                            ;
70: WRITELN (SALIDA, '-> LUEGO DE EXPRESION DEBE IR PALAB."OF"')  ;
71: WRITELN (SALIDA, '-> SE ESPERABA ENTERO LUEGO DE PALABRA ' ,
                     'RESERVADA O LUEGO DE COMA')                 ;
72: WRITELN (SALIDA, '-> SIMBOLO LUEGO DE PALB.RESERVADA O DE ' ,
                     'COMA DEBE SER ENTERO')                      ;
73: WRITELN (SALIDA, '-> SIMBOLO LUEGO DE ENTERO DEBE SER COMA');
74: WRITELN (SALIDA, '-> NO PUEDE ASIGNARSE UN VALOR A ESTE ' ,
                     'SIMBOLO')                                   ;
75: WRITELN (SALIDA, '-> ENTRE ITEMS DEBE IR UNA COMA')          ;
76: WRITELN (SALIDA, '-> LUEGO DE COMA SE ESPERABA EXPRESION')   ;
77: WRITELN (SALIDA, '-> SIMBOLO INVALIDO LUEGO DE NOMBRE DE ' ,
                     'PROGRAMA')                                  ;
78: WRITELN (SALIDA, '-> SE ESPERABA PROPOSICION "ENDWHILE"')    ;
79: WRITELN (SALIDA, '-> SIMBOLO LUEGO DE CONDICION DEBE SER ' ,
                     'PALABRA "DO"')                              ;
80: WRITELN (SALIDA, '-> SE ESPERABA PROPOSICION "ENDCASE"')     ;
81: WRITELN (SALIDA, '-> ESTA FUNCION DEBE TENER PARAMETROS')    ;
82: WRITELN (SALIDA, '-> IDENTIFICADOR YA FUE DECLARADO')        ;
83: WRITELN (SALIDA, '-> IDENTIFICADOR NO HA SIDO DECLARADO')    ;
84: WRITELN (SALIDA, '-> EL NUMERO DE PARAMETROS DE LA FUNCION ' ,
                     'NO CORRESPONDE')                            ;
85: WRITELN (SALIDA, '-> EL NOMBRE DE PROGRAMA DEBE EMPEZAR CON ' ,
                     'UNA LETRA')                                 ;
86: WRITELN (SALIDA, '-> UN "GOTO" TRANSFIERE EL CONTROL A UNA ' ,
                     'LINEA QUE NO EXISTE')                       ;
87: WRITELN (SALIDA, '-> EL NUMERO DE PARAMETROS EN ESTA FUNCION ',
                     'ES MAYOR AL NUMERO MAXIMO PERMITIDO')       ;
88: WRITELN (SALIDA, '-> EL NUMERO DE IDENTIFICADORES USADOS EN ' ,
                     'EL PROGRAMA ES MAYOR AL NUMERO MAXIMO PERMITIDO');
89: WRITELN (SALIDA, '-> EL NUMERO DE DATOS, INGRESADOS POR LA ' ,
                     'PROP. "DATA" ES MAYOR AL NUMERO MAXIMO PERMITIDO');
90: WRITELN (SALIDA, '-> EL NUMERO INSTRUCCIONES DE CODIGO GENERA',
                     'DO ES MAYOR AL NUMERO MAXIMO PERMITIDO')    ;
91: WRITELN (SALIDA, '-> EL NUMERO DE PROPOSICIONES DEL PROGRAMA ',
                     'FUENTE ES MAYOR AL NUMERO MAXIMO PERMITIDO');
92: WRITELN (SALIDA, '-> EL NUMERO DE TRANFERENCIAS INCONDICIONA' ,
                     'LES DE CONTROL, EN EL CODIGO GENERADO, ES MAYOR ' ,
                     'AL NUMERO MAXIMO PERMITIDO')                ;
93: WRITELN (SALIDA, '-> EL VALOR DEL NUMERO DE PROPOSICION ES ' ,
                     'MAYOR AL VALOR MAXIMO PERMITIDO')           ;
94: WRITELN (SALIDA, '-> NO EXISTEN DATOS PARA ASIGNAR A ESTA ' ,
                     'VARIABLE')                                  ;
```

```
   95: WRITELN (SALIDA, '-> NUMERO HEXADECIMAL INCORRECTO')                ;
   96: WRITELN (SALIDA, '-> DATO INCORRECTO')                              ;
   97: WRITELN (SALIDA, '-> SE PARALIZA COMPILACION POR HABER LLEGADO',
                        ' AL TOPE DE LA MEMORIA. DISMINUYA EL NUMERO',
                        ' DE OPERACIONES EN SU PROGRAMA')
 END /* CASE */
 END /* ERROR */ ; .
```

```
==========================================================================
==========================================================================
====                                                                  ===
====      ESCUELA POLITECNICA NACIONAL                                ===
====      FACULTAD DE INGENIERIA ELECTRICA                            ===
====      TESIS DE GRADO                                              ===
====                                                                  ===
====      ALUMNO :   TORO LORGIO                                      ===
====                                                                  ===
====                                                                  ===
==========================================================================
==========================================================================
```



```
SEGMENT FINLINE;

PROCEDURE FINLINE (FL:INTEGER) ; EXTERNAL ;
PROCEDURE ERROR   (NE:INTEGER) ; EXTERNAL ;
PROCEDURE OBSIMB               ; EXTERNAL ;

PROCEDURE FINLINE ;
  TYPE  T1 = PACKED ARRAY (.1..73.) OF CHAR ;
  REF   SALIDA : TEXT    ;        LINEAUX : T1      ;
        CC     : INTEGER ;        PLI     : BOOLEAN ;

  BEGIN /* FINLINE */
  WRITELN (SALIDA, LINEAUX);    WRITELN (SALIDA,'$':CC-1);
  WHILE NOT PLI DO OBSIMB  ;    ERROR (FL)
  END /* FINLINE */ ;
```

```
================================================================================
================================================================================
====                                                                        ====
====      ESCUELA POLITECNICA NACIONAL                                      ====
====      FACULTAD DE INGENIERIA ELECTRICA                                  ====
====      TESIS DE GRADO                                                    ====
====                                                                        ====
====      ALUMNO :    TORO LORCIO                                           ====
====                                                                        ====
====                                                                        ====
================================================================================
================================================================================
```



```
SEGMENT POSICION ;

FUNCTION POSICION:INTEGER ; EXTERNAL ;

FUNCTION POSICION ;

  CONST    NMXID = 21 ;        NMXPR = 5 ;
           LMXST = 20 ;        LMXHX = 5 ;
  TYPE     T1 = PACKED ARRAY (.1..10.)  OF CHAR ;
           T31= PACKED ARRAY (.1..NMXPR.) OF INTEGER ;
           T33= PACKED ARRAY (.1..LMXHX.) OF CHAR    ;
           T30= PACKED ARRAY (.0..NMXID.) OF
                RECORD
                  NOMBRE : T1          ;
                  DIR    : INTEGER     ;
                  CASE TIPO:CHAR OF
                       'S': (SERIEHEX: PACKED ARRAY (.1..LMXST.) OF T33);
                       'A': (NSUB,SUB1,SUB2: INTEGER) ;
                       'F': (NPAR:0..NMXPR; CDFU: INTEGER; DIRP:T31)
                END ;
  REF      CID : 0..NMXID   ;        SIMB   : T1  ;
                                     TIDENT : T30 ;
  VAR      I   : INTEGER    ;

  BEGIN /* POSICION */
  TIDENT(.0.).NOMBRE := SIMB ;
  I := CID ;
  WHILE TIDENT(.I.).NOMBRE <> SIMB DO I:=I-1 ;
  POSICION := I
  END /* POSICION */ ; .
```

```
==========================================================================
==========================================================================
====                                                                  ===
====      ESCUELA POLITECNICA NACIONAL                                ===
====      FACULTAD DE INGENIERIA ELECTRICA                            ===
====      TESIS DE GRADO                                              ===
====                                                                  ===
====      ALUMNO :   TORO LORGIO                                      ===
====                                                                  ===
====                                                                  ===
==========================================================================
==========================================================================
```



```
SEGMENT PASEAHEX ;

  PROCEDURE PASEAHEX (DC:CHAR) ;  EXTERNAL ;

  PROCEDURE PASEAHEX ;

    CONST   LMXHX = 5 ;
    REF     NUMHEX : PACKED ARRAY [1..LMXHX] OF CHAR ;
    BEGIN /* PASEAHEX */
    CASE DC OF
      ' ': NUMHEX:= '$20';  '!': NUMHEX:= '$21';
      '"': NUMHEX:= '$22';  '#': NUMHEX:= '$23';
      '$': NUMHEX:= '$24';  '%': NUMHEX:= '$25';  '&': NUMHEX:= '$26';
      '(': NUMHEX:= '$28';  ')': NUMHEX:= '$29';  '*': NUMHEX:= '$2A';
      '+': NUMHEX:= '$2B';  ',': NUMHEX:= '$2C';  '-': NUMHEX:= '$2D';
      '.': NUMHEX:= '$2E';  '/': NUMHEX:= '$2F';  '0': NUMHEX:= '$30';
      '1': NUMHEX:= '$31';  '2': NUMHEX:= '$32';  '3': NUMHEX:= '$33';
      '4': NUMHEX:= '$34';  '5': NUMHEX:= '$35';  '6': NUMHEX:= '$36';
      '7': NUMHEX:= '$37';  '8': NUMHEX:= '$38';  '9': NUMHEX:= '$39';
      ':': NUMHEX:= '$3A';  ';': NUMHEX:= '$3B';  '<': NUMHEX:= '$3C';
      '=': NUMHEX:= '$3D';  '>': NUMHEX:= '$3E';  '?': NUMHEX:= '$3F';
      '@': NUMHEX:= '$40';  'A': NUMHEX:= '$41';  'B': NUMHEX:= '$42';
      'C': NUMHEX:= '$43';  'D': NUMHEX:= '$44';  'E': NUMHEX:= '$45';
      'F': NUMHEX:= '$46';  'G': NUMHEX:= '$47';  'H': NUMHEX:= '$48';
      'I': NUMHEX:= '$49';  'J': NUMHEX:= '$4A';  'K': NUMHEX:= '$4B';
      'L': NUMHEX:= '$4C';  'M': NUMHEX:= '$4D';  'N': NUMHEX:= '$4E';
      'O': NUMHEX:= '$4F';  'P': NUMHEX:= '$50';  'Q': NUMHEX:= '$51';
      'R': NUMHEX:= '$52';  'S': NUMHEX:= '$53';  'T': NUMHEX:= '$54';
      'U': NUMHEX:= '$55';  'V': NUMHEX:= '$56';  'W': NUMHEX:= '$57';
      'X': NUMHEX:= '$58';  'Y': NUMHEX:= '$59';  'Z': NUMHEX:= '$5A'
    END /* CASE */
  END /* PASEAHEX */ ;.
```

```
==========================================================================
==========================================================================
====                                                                  ====
====      ESCUELA POLITECNICA NACIONAL                                ====
====      FACULTAD DE INGENIERIA ELECTRICA                            ====
====      TESIS DE GRADO                                              ====
====                                                                  ====
====      ALUMNO :    TORO LORGIO                                     ====
====                                                                  ====
====                                                                  ====
==========================================================================
==========================================================================
```



```
SEGMENT GENCODIG;

PROCEDURE GENCODIG (GC1:STRING(3); GC2,GC3:CHAR; GC4,GC5:INTEGER)
                                                    ; EXTERNAL  ;
PROCEDURE FINLINE  (FL:INTEGER)                     ; EXTERNAL  ;

PROCEDURE GENCODIG ;
CONST     NMXCD= 501 ;    LMXHX = 5  ;
TYPE      T3 = PACKED ARRAY (.1..5.)      OF CHAR ;
          T33= PACKED ARRAY (.1..LMXHX.) OF CHAR ;
          T51= RECORD
                 COPRDR : STRING(3)  ;
                 TIPVAR : CHAR       ;
                 CTOPND : CHAR       ;
                 COPHEX : T33        ;
                 COPNUM : INTEGER    ;
                 CREGIX : INTEGER
               END ;
          T50= PACKED ARRAY (.0..NMXCD.) OF T51 ;
REF       CCD     : 0..NMXCD ;        SALIDA : TEXT ;
          TCODIG  : T50      ;        NUMHEX : T33  ;
VAR       I       : INTEGER  ;

BEGIN /* GENCODIG */
IF CCD<NMXCD THEN
  BEGIN
  WITH TCODIG(.CCD.) DO
    BEGIN
    COPRDR:= GC1;    TIPVAR:=GC2;    CTOPND:= GC3;
    IF GC3='H' THEN COPHEX:=NUMHEX  ELSE COPHEX:=' ';
    COPNUM:= GC4;    CREGIX:= GC5
    END ;
  CCD := CCD+1
  END
ELSE
```

```
  BEGIN                                                          298
  FINLINE(90);
  WRITELN(SALIDA,'-> EL NUMERO MAXIMO DE INSTRUCCIONES DE CODIGO ',
                 'PERMITIDO ES :', (NMXCO-1):5)
  END
END /* GENCODIG */ ; .
```

```
==========================================================================
==========================================================================
====                                                                  ===
====      ESCUELA POLITECNICA NACIONAL                                ===
====      FACULTAD DE INGENIERIA ELECTRICA                            ===
====      TESIS DE GRADO                                              ===
====                                                                  ===
====      ALUMNO :   TORO LORGIO                                      ===
====                                                                  ===
====                                                                  ===
==========================================================================
==========================================================================
```



```
SEGMENT CONSTANTE ;

PROCEDURE CONSTANTE ;  EXTERNAL ;
PROCEDURE OBSIMB    ;  EXTERNAL ;
PROCEDURE OBCTR     ;  EXTERNAL ;
PROCEDURE FINLINE (FL:INTEGER)  ;  EXTERNAL ;

PROCEDURE CONSTANTE ;

  CONST  LMXST = 20 ;     LMXHX = 5 ;
  TYPE   T1  = PACKED ARRAY (.1..10.) OF CHAR ;
         T32 = PACKED ARRAY (.1..LMXST.) OF CHAR ;
         T33 = PACKED ARRAY (.1..LMXHX.) OF CHAR ;

  REF    TPCONS,CTR : CHAR ;      PLI : BOOLEAN ;
         CLASE : STRING(5) ;      SIMB:  T1     ;
         NUMHEX     : T33  ;      SERIE : T32   ;
  VAR    N1     : INTEGER   ;

  PROCEDURE C_STRING ;
    PROCEDURE LLENESERIE ;
      BEGIN /* LLENESERIE */
      WHILE N1<=LMXST DO  BEGIN SERIE(.N1.):= ' '; N1:=N1+1 END
      END /* LLENESERIE */ ;

    BEGIN /* C_STRING */
    N1:=1;  IF SIMB<>'"' THEN OBCTR;
    IF CTR='"' THEN FINLINE(22) ELSE
    IF PLI     THEN FINLINE(22) ELSE
      BEGIN /* B1 */
      WHILE N1<=LMXST DO
        BEGIN /* B2 */
        SERIE (.N1.):= CTR;  N1:=N1+1;
        IF (CTR='|')|(CTR='¬')|(CTR=' ') THEN FINLINE (3)  ELSE
          BEGIN
          OBCTR ;
```

```
        IF PLI THEN BEGIN FINLINE(22); LLENESERIE END ELSE
        IF CTR='''' THEN BEGIN OBCTR ; LLENESERIE END ELSE
        IF N1>LMXST THEN FINLINE (22)
       END
     END /* B2 */
   END /* B1 */
 END /* C_STRING */ ;

PROCEDURE C_HEXADC ;

  BEGIN /* C_HEXADC */
  NUMHEX(.1.):='$'; OBCTR; N1:=1;
  WHILE N1<LMXHX DO
    BEGIN /* B1 */
    IF (CTR IN (.'0'..'9'.)) | (CTR IN (.'A'..'F'.)) THEN
      BEGIN /* B2 */
      N1:=N1+1; NUMHEX(.N1.):=CTR; OBCTR
      END /* B2 */
    ELSE BEGIN FINLINE(95); N1:=5 END
    END /* B1 */
  END /* C_HEXADC */ ;

BEGIN /* CONSTANTE */
TPCONS:=' '; N1:=0; NUMHEX:=' ' ;
IF (CTR='''')|(SIMB='''') THEN BEGIN TPCONS:='S'; C_STRING END ELSE
IF (CTR='H') THEN BEGIN TPCONS:='H'; C_HEXADC END ELSE
  BEGIN
  OBSIMB;
  IF PLI THEN FINLINE(36) ELSE
  IF CLASE<>'ENTER' THEN FINLINE(21) ELSE TPCONS:='N'
  END
END /* CONSTANTE */ ; .
```

```
==========================================================================
==========================================================================
====                                                                  ====
====      ESCUELA POLITECNICA NACIONAL                                ====
====      FACULTAD DE INGENIERIA ELECTRICA                            ====
====      TESIS DE GRADO                                              ====
====                                                                  ====
====      ALUMNO :   TORO LORGIO                                      ====
====                                                                  ====
====                                                                  ====
==========================================================================
==========================================================================
```

```
SEGMENT OBCTR ;

PROCEDURE OBCTR    ; EXTERNAL ;

PROCEDURE OBCTR ;
  REF  CC,LL : INTEGER ;       CTR   : CHAR ;
       PLI   : BOOLEAN ;       LINEA : PACKED ARRAY (.1..73.) OF CHAR ;

  BEGIN /* OBCTR */
  IF CC=LL THEN
    BEGIN
    PLI:=TRUE;  CC:=0;  LL:=0;
    WHILE NOT EOLN(INPUT)  DO
      BEGIN  LL:=LL+1; READ(CTR); LINEA(.LL.):=CTR   END;
    READLN
    END ;
  CC:=CC+1;  CTR:=LINEA(.CC.)
  END  /* OBTCTR */ ; .
```

```
==========================================================================
==========================================================================
====                                                                  ===
====      ESCUELA POLITECNICA NACIONAL                                ===
====      FACULTAD DE INGENIERIA ELECTRICA                            ===
====      TESIS DE GRADO                                              ===
====                                                                  ===
====      ALUMNO :    TORO LORGIO                                     ===
====                                                                  ===
====                                                                  ===
==========================================================================
==========================================================================
```



```
SEGMENT OBSIMB    ;

PROCEDURE OBSIMB                        ; EXTERNAL ;
PROCEDURE OBCTR                         ; EXTERNAL ;
PROCEDURE FINLINE(FL:INTEGER)           ; EXTERNAL ;

  PROCEDURE OBSIMB ;

    CONST    NCTR = 10;      NPR = 32 ;
    TYPE     T1    = PACKED ARRAY (.1..10.) OF CHAR ;
             T2    = PACKED ARRAY (.1..73.) OF CHAR ;
             T3    = PACKED ARRAY (.1..NPR.) OF T1  ;
    VAR      I,J,K,FF,N1   : INTEGER    ;    FILA     : T1   ;
             VV            : BOOLEAN    ;    PALABRA  : T3   ;
    REF      CC,LL,NUMERO  : INTEGER    ;
             CTR           : CHAR       ;    SIMB     : T1   ;
             PLI           : BOOLEAN    ;    LINEA    : T2   ;
             CLASE         : STRING(5)  ;    SALIDA   : TEXT ;

    PROCEDURE SUM ;  BEGIN   FF:=FF+1;  FILA(.FF.):=CTR   END;

    PROCEDURE SIMB1;  /* PALABRAS RESERVADAS Y VARIABLES */

      BEGIN /* SIMB1 */
      SUM;      OBCTR;      N1:=1;
      IF NOT (CTR IN (.'A'..'Z'.)) THEN
        BEGIN
        CLASE:='VRBLE';
        IF CTR IN (.'0'..'9'.) THEN   BEGIN SUM; OBCTR END  ELSE
        IF CTR = '$'  THEN  BEGIN CLASE:='STRIN'; SUM; OBCTR END
        END
      ELSE
        BEGIN /* B1 */
        WHILE  CTR IN (.'A'..'Z'.) DO
          BEGIN   N1:=N1+1; IF N1<NCTR THEN SUM; OBCTR    END ;
```

```
      IF (N1=3)&(FILA(.1.)='F')&(FILA(.2.)='N') THEN CLASE:='FUNCN'
      ELSE
        BEGIN /* B2 */
        I:=1;    J:=NPR;
        IF N1<NCTR THEN
          BEGIN
          REPEAT
            K:=(I+J) DIV 2;
            IF FILA <= PALABRA(.K.) THEN J:=K-1;
            IF FILA >= PALABRA(.K.) THEN I:=K+1;
          UNTIL I>J;
          IF (I-1)>J THEN  CLASE:='PRESV'  ELSE  CLASE:='INDEF'
          END
        ELSE CLASE := 'INDEF'
        END  /* B2 */
      END  /* B1 */
    END   /* SIMB1 */  ;


  PROCEDURE SIMB2 ;   /* NUMEROS ENTEROS */
    BEGIN /* SIMB2 */
    NUMERO:=0;   N1:=1;   CLASE:='ENTER';
    REPEAT  NUMERO:= 10*NUMERO + (ORD(CTR)-ORD('0'));
            N1:=N1+1;   IF N1<NCTR THEN SUM;   OBCTR
    UNTIL NOT (CTR IN ['0'..'9']);
    IF N1>=NCTR THEN   BEGIN FINLINE(51); CLASE:='INDEF' END
    END   /* SIMB2 */;


  PROCEDURE SIMB3;   /* DELIMITADORES E INDEFINIDOS */
    BEGIN /* SIMB3 */
    IF CTR='<' THEN
      BEGIN
      N1:=2;   SUM;   OBCTR;
      IF (CTR='=') | (CTR='>')  THEN N1:=1
      END
    ELSE  IF CTR='>' THEN
      BEGIN  N1:=2; SUM; OBCTR; IF CTR='=' THEN N1:=1  END
    ELSE  IF CTR='*' THEN
      BEGIN  N1:=2; SUM; OBCTR; IF CTR='*' THEN N1:=1  END
    ELSE  IF (CTR='+') | (CTR='-') | (CTR='/') | (CTR='=') |
             (CTR='(') | (CTR=')') | (CTR=',') | (CTR=';')
             (CTR='.') | (CTR='"') | (CTR='|')
          THEN N1:=1 ELSE N1:=3 ;
    CASE N1 OF
      1: BEGIN   SUM; OBCTR; CLASE:='DELIM'   END;
      2: CLASE:='DELIM';
      3: BEGIN   SUM; OBCTR; CLASE:='INDEF'   END
    END  /* CASE */
    END  /* SIMB3 */;


BEGIN /* OBSIMB */
PALABRA(.1 .):= 'AND       ';     PALABRA(.2 .):= 'CASE      ';
PALABRA(.3 .):= 'DATA      ';     PALABRA(.4 .):= 'DEF       ';
```

```
PALABRA(.5 .):= 'DIM        ';    PALABRA(.6 .):= 'DO         ';
PALABRA(.7 .):= 'ELSE       ';    PALABRA(.8 .):= 'END        ';
PALABRA(.9 .):= 'ENDCASE    ';    PALABRA(.10.):= 'ENDFN      ';
PALABRA(.11.):= 'ENDIF      ';    PALABRA(.12.):= 'ENDWHILE   ';
PALABRA(.13.):= 'FOR        ';    PALABRA(.14.):= 'GOSUB      ';
PALABRA(.15.):= 'GOTO       ';    PALABRA(.16.):= 'IF         ';
PALABRA(.17.):= 'LET        ';    PALABRA(.18.):= 'NEXT       ';
PALABRA(.19.):= 'NOT        ';    PALABRA(.20.):= 'OF         ';
PALABRA(.21.):= 'OR         ';    PALABRA(.22.):= 'PRINT      ';
PALABRA(.23.):= 'PROGRAM    ';    PALABRA(.24.):= 'READ       ';
PALABRA(.25.):= 'REM        ';    PALABRA(.26.):= 'RESTORE    ';
PALABRA(.27.):= 'RETURN     ';    PALABRA(.28.):= 'SUB        ';
PALABRA(.29.):= 'THEN       ';    PALABRA(.30.):= 'TO         ';
PALABRA(.31.):= 'VAR        ';    PALABRA(.32.):= 'WHILE      ';
WHILE CTR=' ' DO OBCTR;
FF:=0;   FILA:=' ';
IF CTR IN (.'A'..'Z'.) THEN SIMB1 ELSE
IF CTR IN (.'0'..'9'.) THEN SIMB2 ELSE SIMB3;
SIMB:=FILA
END   /* OBSIMB */;
```

```
==========================================================================
==========================================================================
====                                                                  ====
====      ESCUELA POLITECNICA NACIONAL                                ====
====      FACULTAD DE INGENIERIA ELECTRICA                            ====
====      TESIS DE GRADO                                              ====
====                                                                  ====
====      ALUMNO :    TORO LORGIO                                     ====
====                                                                  ====
====                                                                  ====
==========================================================================
==========================================================================
```



```
SEGMENT   CODIGASS ;
PROCEDURE CODIGASS ; EXTERNAL ;
PROCEDURE ERROR (NE:INTEGER)  ; EXTERNAL ;
PROCEDURE CODIGASS ;

  CONST   NMXPL = 501;     NMXAS = 1001;     NMXCD = 501;
          NMXSL = 51 ;     NMXSC =   21;     LMXST =  20;
          NMXPR =  5 ;     NMXID =   21;     LMXHX = 5 ;
  TYPE    T1  = PACKED ARRAY (.1..10.) OF CHAR;
          T3  = PACKED ARRAY (.1..5.) OF CHAR ;
          T31 = PACKED ARRAY (.0..NMXPR.) OF INTEGER  ;
          T33 = PACKED ARRAY (.1..LMXHX.) OF CHAR ;
          T30 = PACKED ARRAY (.0..NMXID.) OF
                RECORD
                  NOMBRE : T1          ;
                  DIR    : INTEGER     ;
                  CASE TIPO:CHAR OF
                     'S': (SERIEHEX: PACKED ARRAY (.1..LMXST.) OF T33);
                     'A': (NSUB,SUB1,SUB2: INTEGER) ;
                     'F': (NPAR:0..NMXPR; CDFU:INTEGER; DIRP:T31)
                END ;
          T51 = RECORD
                  COPRDR  : STRING(3) ;
                  TIPVAR  : CHAR      ;
                  CTOPND  : CHAR      ;
                  COPHEX  : T33       ;
                  COPNUM  : INTEGER   ;
                  CREGIX  : INTEGER
                END ;
          T50 = PACKED ARRAY (.0..NMXCD.) OF T51 ;
          T61 = PACKED ARRAY (.0..NMXCD.) OF
                RECORD CINUM, CICME:INTEGER  END ;
          T71 = PACKED ARRAY (.0..NMXSL.) OF
                RECORD SLNUM, SLCAS:INTEGER  END ;
          T81 = RECORD
                  CMEMOR : INTEGER   ;
```

```
                OPRDOR : STRING(4) ;
                TIPDIR : CHAR      ;
                TIPOPN : CHAR      ;
                OPNHEX : T33       ;
                OPNNUM : INTEGER   ;
                INDICE : STRING(2)
               END ;
        T80 = PACKED ARRAY (.0..NMXAS.) OF T81      ;
        T90 = PACKED ARRAY (.0..NMXSC.) OF INTEGER ;
REF     SALIDA : TEXT      ;        RCDINT : T51 ;
        DPL    : 0..NMXPL ;        CM : INTEGER ;
        NUMHEX : T33       ;
        CID    : 0..NMXID ;        TIDENT : T30 ;
        CCD    : 0..NMXCD ;        TCODIG : T50 ;
        CIN    : 0..NMXCD ;        TNUCIN : T61 ;
        CSL    : 0..NMXSL ;        TSALTO : T71 ;
        CAS    : 0..NMXAS ;        TASSEM : T80 ;
        CSC    : 0..NMXSC ;        TSACND : T90 ;
VAR     SAUX   : STRING(3);        OPRAUX : STRING(4) ;
        TOPE   : BOOLEAN   ;     N1,N2,N3 : INTEGER;

PROCEDURE GENASS  (GA1:INTEGER; GA2:STRING(4); GA3:CHAR      ;
                   GA4:CHAR   ; GA5:INTEGER  ; GA6:STRING(2)) ;
BEGIN /* GENASS */
CAS:=CAS+1 ;
IF (CAS>=(NMXAS-1)) THEN BEGIN ERROR(90); TOPE:=TRUE  END  ELSE
BEGIN /* B1 */
WITH TASSEM(.CAS.) DO
  BEGIN
  CMEMOR:=GA1;   OPRDOR:=GA2;     TIPDIR:=GA3;     TIPOPN:=GA4;
  OPNHEX:=NUMHEX ;                OPNNUM:=GA5;     INDICE:=GA6
  END
END /* B1 */
END /* GENASS */ ;


PROCEDURE VEAPILA (VP:INTEGER) ;
  BEGIN
  IF (VP>=(NMXPL-1))  THEN BEGIN ERROR(97); TOPE:=TRUE  END
  END ;


PROCEDURE LLENARSL ;
  VAR I1,I2 : INTEGER ;
  BEGIN /* LLENARSL */
  I1:=1 ;
  WHILE I1<=CSL DO
    BEGIN /* B1 */
    TNUCIN(.0.).CINUM:= TSALTO(.I1.).SLNUM;          I2:=CIN;
    WHILE TNUCIN(.I2.).CINUM<>TSALTO(.I1.).SLNUM DO I2:=I2-1 ;
    WITH TSALTO(.I1.) DO
      TASSEM(.SLCAS.).OPNNUM:= (TNUCIN(.I2.).CICME) ;
    I1:=I1+1
    END /* B1 */
  END /* LLENARSL */ ;
```

```
PROCEDURE LLENARSC ;
  VAR I:INTEGER ;
  BEGIN /* LLENARSC */
  I:=1 ;
  WHILE I<=CSC DO
    BEGIN
    WITH TASSEM(.TSACND(.I.).) DO
      OPNNUM:= OPNNUM - TASSEM(.(TSACND(.I.)+1).).CMEMBR ;
    I:= I+1
    END
  END /* LLENARSC */ ;

PROCEDURE PONERSL (SL:INTEGER) ;
  BEGIN
  CSL:=CSL+1 ;
  WITH TSALTO(.CSL.) DO BEGIN SLNUM:=SL; SLCAS:=CAS END
  END ;


PROCEDURE CONCATENAR ;
  VAR    N1,N2,N3,N4 : INTEGER ;

  BEGIN /* CONCATENAR */
  N1:= 0        ;          N2:= CAS-39 ;
  N3:= CAS-79   ;
  WHILE (N1<(LMXST+LMXST)) DO
    BEGIN /* B1 */
    DPL:= DPL+1;
    IF (TASSEM(.(N3+N1).).OPNHEX = '$22') THEN
      BEGIN
      NUMHEX:= TASSEM(.N2.).OPNHEX ;
      N2:= N2+2
      END
    ELSE  NUMHEX:= TASSEM(.(N3+N1).).OPNHEX;
    GENASS (CM,'LDAA','#','H',      0 ,' ') ;   CM:=CM+2 ;
    GENASS (CM,'STAA',' ','N',   DPL ,' ') ;   CM:=CM+3 ;
    N1:= N1+2
    END /* B1 */
  END /* CONCATENAR */ ;


PROCEDURE SIGNO;
  BEGIN /* SIGNO */
  GENASS ( CM,'TST ',' ','N',    DPL ,' ') ;   CM:=CM+3 ;
  GENASS ( CM,'BPL ',' ','N',     20 ,' ') ;   CM:=CM+2 ;
  GENASS ( CM,'NEG ',' ','N',    DPL ,' ') ;   CM:=CM+3 ;
  GENASS ( CM,'TST ',' ','N',(DPL+1),' ') ;   CM:=CM+3 ;
  GENASS ( CM,'BPL ',' ','N',      5 ,' ') ;   CM:=CM+2 ;
  GENASS ( CM,'NEG ',' ','N',(DPL+1),' ') ;   CM:=CM+3 ;
  GENASS ( CM,'BRA ',' ','N',     20 ,' ') ;   CM:=CM+2 ;
  GENASS ( CM,'LDAA','#','N',      1 ,' ') ;   CM:=CM+2 ;
  GENASS ( CM,'STAA',' ','N',      0 ,' ') ;   CM:=CM+3 ;
  GENASS ( CM,'BRA ',' ','N',     13 ,' ') ;   CM:=CM+2 ;
  GENASS ( CM,'TST ',' ','N',(DPL+1),' ') ;   CM:=CM+3 ;
```

```
    GENASS ( CM,'BPL ',' ','N',      8 ,' ') ;    CM:=CM+2 ;
    GENASS ( CM,'LDAA','#','N',      1 ,' ') ;    CM:=CM+2 ;
    GENASS ( CM,'STAA',' ','N',      0 ,' ') ;    CM:=CM+3 ;
    GENASS ( CM,'NEG ',' ','N',(DPL+1),' ') ;    CM:=CM+3
    END /* SIGNO */  ;

  PROCEDURE RESULT;
    BEGIN /* RESULT */
    GENASS ( CM,'STAB',' ','N',    DPL ,' ') ;    CM:=CM+3 ;
    GENASS ( CM,'TST ',' ','N',      0 ,' ') ;    CM:=CM+3 ;
    GENASS ( CM,'BEQ ',' ','N',      3 ,' ') ;    CM:=CM+2 ;
    GENASS ( CM,'NEG ',' ','N',    DPL ,' ') ;    CM:=CM+3
    END /* RESULT */ ;

  BEGIN /* CODIGASS */
  TOPE:=FALSE ;
  REPEAT
    CIN:=CIN+1;       RCDINT:=TCODIG(.CIN.) ;
    WITH TNUCIN (.CIN.) DO
      BEGIN CINUM:=CIN;  CICME:=CM END ;
    IF (TOPE=TRUE) THEN   BEGIN SAUX:='END'; ERROR(97) END    ELSE
    BEGIN /* TOPEFALSO */
    WITH RCDINT DO
    BEGIN /* WITH */
    SAUX:=COPRDR ;
    IF CTOPND='N' THEN NUMHEX:=' ' ELSE NUMHEX:=COPHEX ;
    IF (COPRDR='OPR') THEN
      BEGIN /* OPR */
      IF (COPNUM>0) & (COPNUM<7) THEN
        BEGIN /* IF */
        CASE COPNUM OF
          1 : OPRAUX:= 'BGT ' ;
          2 : OPRAUX:= 'BGE ' ;
          3 : OPRAUX:= 'BEQ ' ;
          4 : OPRAUX:= 'BNE ' ;
          5 : OPRAUX:= 'BLE ' ;
          6 : OPRAUX:= 'BLT '
        END /* CASE */ ;
        GENASS ( CM,'LDAA',' ',CTOPND,DPL,' ') ;    CM:=CM+3 ;
        DPL:=DPL-1 ;
        GENASS ( CM,'CMPA',' ',CTOPND,DPL,' ') ;    CM:=CM+3 ;
        GENASS ( CM,OPRAUX,' ',CTOPND,  5,' ') ;    CM:=CM+2 ;
        GENASS ( CM,'CLR ',' ',CTOPND,DPL,' ') ;    CM:=CM+3 ;
        GENASS ( CM,'BRA ',' ',CTOPND,  5,' ') ;    CM:=CM+2 ;
        GENASS ( CM,'LDAA','#',CTOPND,  1,' ') ;    CM:=CM+2 ;
        GENASS ( CM,'STAA',' ',CTOPND,DPL,' ') ;    CM:=CM+3
        END /* IF */
      ELSE
        BEGIN /* ELSE */
        CASE COPNUM OF
        7   : BEGIN GENASS ( CM,'NEG ',' ',CTOPND,DPL,' ') ;CM:=CM+3 END;
        8   : BEGIN
              GENASS ( CM,'LDAA',' ',CTOPND,DPL,' ') ;    CM:=CM+3 ;
              DPL:=DPL-1 ;
```

```
        GENASS ( CM,'ADDA',' ',CTOPND,DPL,' ') ;   CM:=CM+3 ;
        GENASS ( CM,'STAA',' ',CTOPND,DPL,' ') ;   CM:=CM+3
        END ;
    9 : BEGIN
        GENASS ( CM,'LDAA',' ',CTOPND,DPL,' ') ;   CM:=CM+3 ;
        DPL:=DPL-1 ;
        GENASS ( CM,'SUBA',' ',CTOPND,DPL,' ') ;   CM:=CM+3 ;
        GENASS ( CM,'NEGA',' ',CTOPND,  0,' ') ;   CM:=CM+1 ;
        GENASS ( CM,'STAA',' ',CTOPND,DPL,' ') ;   CM:=CM+3
        END ;
   10 : BEGIN
        GENASS ( CM,'LDAA',' ',CTOPND,DPL,' ') ;   CM:=CM+3 ;
        DPL:=DPL-1 ;
        GENASS ( CM,'ORAA',' ',CTOPND,DPL,' ') ;   CM:=CM+3 ;
        GENASS ( CM,'STAA',' ',CTOPND,DPL,' ') ;   CM:=CM+3
        END ;
   11 : BEGIN
        DPL:=DPL-1 ;
        GENASS ( CM,'LDAA','#',CTOPND,    0 ,' ') ;   CM:=CM+2 ;
        GENASS ( CM,'STAA',' ',CTOPND,    C ,' ') ;   CM:=CM+3 ;
        GENASS ( CM,'LDAB','#',CTOPND,    0 ,' ') ;   CM:=CM+2 ;
        GENASS ( CM,'TST ',' ',CTOPND,  DPL ,' ') ;   CM:=CM+3 ;
        GENASS ( CM,'BEQ ',' ',CTOPND,   53 ,' ') ;   CM:=CM+2 ;
        GENASS ( CM,'TST ',' ',CTOPND,(DPL+1),' ') ;  CM:=CM+3 ;
        GENASS ( CM,'BNE ',' ',CTOPND,    2 ,' ') ;   CM:=CM+2 ;
        GENASS ( CM,'BRA ',' ',CTOPND,   46 ,' ') ;   CM:=CM+2 ;
        SIGNO;
        GENASS ( CM,'ADDB',' ',CTOPND,  DPL ,' ') ;   CM:=CM+3 ;
        GENASS ( CM,'DEC ',' ',CTOPND,(DPL+1),' ') ;  CM:=CM+3 ;
        NUMHEX:= '$F8' ;
        GENASS ( CM,'BNE ',' ','H' ,    0 ,' ') ;   CM:=CM+2 ;
        RESULT
        END ;
   12 : BEGIN
        DPL:=DPL-1 ;
        GENASS ( CM,'LDAA','#',CTOPND,    0 ,' ') ;   CM:=CM+2 ;
        GENASS ( CM,'STAA',' ',CTOPND,    0 ,' ') ;   CM:=CM+3 ;
        GENASS ( CM,'TST ',' ',CTOPND,(DPL+1),' ') ;  CM:=CM+3 ;
        GENASS ( CM,'BNE ',' ',CTOPND,    4 ,' ') ;   CM:=CM+2 ;
        GENASS ( CM,'LDAB','#',CTOPND, 70000 ,' ') ;  CM:=CM+2 ;
        GENASS ( CM,'BRA ',' ',CTOPND,   68 ,' ') ;   CM:=CM+2 ;
        GENASS ( CM,'LDAB','#',CTOPND,    0 ,' ') ;   CM:=CM+2 ;
        GENASS ( CM,'TST ',' ',CTOPND,  DPL ,' ') ;   CM:=CM+3 ;
        GENASS ( CM,'BEQ ',' ',CTOPND,   61 ,' ') ;   CM:=CM+2 ;
        GENASS ( CM,'LDAA',' ',CTOPND,  DPL ,' ') ;   CM:=CM+3 ;
        GENASS ( CM,'CMPA',' ',CTOPND,(DPL+1),' ') ;  CM:=CM+3 ;
        GENASS ( CM,'BMI ',' ',CTOPND,   53 ,' ') ;   CM:=CM+2 ;
        SIGNO;
        GENASS ( CM,'INCB',' ',CTOPND,    0 ,' ') ;   CM:=CM+1 ;
        GENASS ( CM,'LDAA',' ',CTOPND,  DPL ,' ') ;   CM:=CM+3 ;
        GENASS ( CM,'SUBA',' ',CTOPND,(DPL+1),' ') ;  CM:=CM+3 ;
        GENASS ( CM,'STAA',' ',CTOPND,  DPL ,' ') ;   CM:=CM+3 ;
        GENASS ( CM,'CMPA',' ',CTOPND,(DPL+1),' ') ;  CM:=CM+3 ;
        NUMHEX:= '$F1' ;
        GENASS ( CM,'BPL ',' ','H' ,    0 ,' ') ;   CM:=CM+2 ;
```

```
          RESULT
          END ;
      13 : BEGIN
          GENASS ( CM,'LDAA',' ',CTOPND,DPL,' ') ;   CM:=CM+3 ;
          DPL:=DPL-1 ;
          GENASS ( CM,'ANDA',' ',CTOPND,DPL,' ') ;   CM:=CM+3 ;
          GENASS ( CM,'STAA',' ',CTOPND,DPL,' ') ;   CM:=CM+3
          END ;
      14 : BEGIN DPL:= DPL-40;  CONCATENAR     END ;
      15 : BEGIN
          GENASS ( CM,'TST ',' ',CTOPND,DPL,' ') ;   CM:=CM+3 ;
          GENASS ( CM,'BEQ ',' ',CTOPND,  5,' ') ;   CM:=CM+2 ;
          GENASS ( CM,'CLR ',' ',CTOPND,DPL,' ') ;   CM:=CM+3 ;
          GENASS ( CM,'BRA ',' ',CTOPND,  3,' ') ;   CM:=CM+2 ;
          GENASS ( CM,'INC ',' ',CTOPND,DPL,' ') ;   CM:=CM+3
          END ;
    END /* CASE */
    END /* ELSE */
    END /* OPR */   ELSE
  IF (COPRDR='CNU') THEN
    BEGIN
    DPL:=DPL+1 ;   VEAPILA (DPL) ;
    GENASS ( CM,'LDAA','#',CTOPND,COPNUM,' ') ;  CM:=CM+2 ;
    GENASS ( CM,'STAA',' ','N'   ,DPL   ,' ') ;   CM:=CM+3
    END    ELSE
  IF (COPRDR='CVA') THEN
    BEGIN /* CARGUEVRBLE */
    VEAPILA (DPL) ;
    IF (TIPVAR='V') THEN
      BEGIN
      DPL:= DPL+1 ;
      GENASS ( CM,'LDAA',' ',CTOPND,COPNUM ,' ') ;   CM:=CM+3 ;
      GENASS ( CM,'STAA',' ',CTOPND,   DPL ,' ') ;   CM:=CM+3
      END
    ELSE IF (TIPVAR='A') THEN
      BEGIN
      DPL:= DPL+1 ;
      GENASS ( CM,'LDX ','#',CTOPND,CREGIX ,'  ') ;   CM:=CM+3 ;
      GENASS ( CM,'LDAA',' ',CTOPND,COPNUM ,',X') ;   CM:=CM+2 ;
      GENASS ( CM,'STAA',' ',CTOPND,   DPL ,'  ') ;   CM:=CM+3
      END
    ELSE
      BEGIN /* B2 */
      TIDENT(.0.).DIR:= COPNUM ;
      N1:= CID                 ;
      WHILE (TIDENT(.N1.).DIR<>COPNUM) DO N1:=N1-1 ;
      N2:=1                    ;
      WHILE (N2<=LMXST) DO
        BEGIN
        DPL:= DPL+1 ;  VEAPILA (DPL) ;
        IF (TOPE=TRUE) THEN N2:=20 ;
        NUMHEX:= TIDENT(.N1.).SERIEHEX(.N2.) ;
        GENASS ( CM,'LDAA','#','H',     0 ,'  ') ;   CM:=CM+2 ;
        GENASS ( CM,'STAA',' ','N',   DPL ,'  ') ;   CM:=CM+3 ;
        N2:=N2+1
```

```
      END
    END /* B2 */
  END /* CARGUEVRBLE */     ELSE
IF (COPRDR='ALM') THEN
  BEGIN /* ALMACENAR */
  IF (TIPVAR='S') THEN
    BEGIN /* B1 */
    TIDENT(.0.).DIR:= COPNUM ;
    N1:= CID                 ;
    WHILE (TIDENT(.N1.).DIR<>COPNUM) DO N1:=N1-1 ;
    N2:= 1;      N3:=CAS-39;
    WHILE (N2<=LMXST) DO
      BEGIN
      NUMHEX:= TASSEM(.N3.).OPNHEX ;
      TIDENT(.N1.).SERIEHEX(.N2.):= NUMHEX ;
      GENASS ( CM,'LDAA','#','H'    , 0     ,'   ') ;   CM:=CM+2 ;
      GENASS ( CM,'STAA',' ','N',(COPNUM+N2-1),' '); CM:=CM+3 ;
      N3:= N3+2  ;
      N2:= N2+1
      END         ;
    DPL:= DPL-20
    END /* B1 */
  ELSE
    BEGIN /* B1 */
    GENASS ( CM,'LDAA',' ',CTOPND,DPL     ,'   ') ;   CM:=CM+3 ;
    IF (TIPVAR='V') THEN
      BEGIN
      GENASS ( CM,'STAA',' ',CTOPND,COPNUM,'   ') ;   CM:=CM+3 ;
      END
    ELSE
      BEGIN
      GENASS ( CM,'LDX ','#',CTOPND,CREGIX,'   ') ;   CM:=CM+3 ;
      GENASS ( CM,'STAA',' ',CTOPND,COPNUM,',X') ;   CM:=CM+2 ;
      END ;
    DPL:=DPL-1
    END /* B1 */
  END /* ALMACENAR */     ELSE
IF (COPRDR='LLS') THEN
  BEGIN
  GENASS ( CM,'JSR ',' ',CTOPND,COPNUM,'   ') ;   CM:=CM+3 ;
  PONERSL(COPNUM)
  END     ELSE
IF (COPRDR='RTS') THEN
  BEGIN GENASS ( CM,'RTS ',' ',CTOPND, 0,'   '); CM:=CM+1  END ELSE
IF (COPRDR='INS') THEN  DPL:=DPL+COPNUM                    ELSE
IF (COPRDR='SAI') THEN
  BEGIN
  GENASS ( CM,'JMP ',' ',CTOPND,COPNUM,'   ') ;   CM:=CM+3 ;
  PONERSL(COPNUM)
  END     ELSE
IF (COPRDR='SAC') THEN
  BEGIN
  GENASS ( CM,'TST ',' ',CTOPND,DPL     ,'   ') ;   CM:=CM+3 ;
  DPL:= DPL-1 ;
  GENASS ( CM,'BEQ ',' ',CTOPND,      0,'   ') ;   CM:=CM+2 ;
```

```
        PONERSL(COPNUM) ;
        CSC:=CSC+1;
        TSACND(.CSC.):= CAS
        END
      END /* WITH */
      END /* TOPEFALSO */-
    UNTIL (SAUX='END') ;
    IF (TOPE=FALSE) THEN
      BEGIN
      LLENARSL ;
      LLENARSC
      END
    END /* CODIGASS */ ; .
```

```
==========================================================================
==========================================================================
====                                                                  ===
====      ESCUELA POLITECNICA NACIONAL                                ===
====      FACULTAD DE INGENIERIA ELECTRICA                            ===
====      TESIS DE GRADO                                              ===
====                                                                  ===
====      ALUMNO :   TORO LORGIO                                      ===
====                                                                  ===
====                                                                  ===
==========================================================================
==========================================================================
```



```
SEGMENT EXPRESION ;

  PROCEDURE EXPRESION ; EXTERNAL ;
  PROCEDURE CONSTANTE ; EXTERNAL ;
  PROCEDURE OBSIMB    ; EXTERNAL ;
  FUNCTION  POSICION  : INTEGER  ;  EXTERNAL ;
  PROCEDURE FINLINE (FL:INTEGER) ;  EXTERNAL ;
  PROCEDURE PASEAHEX(DC:CHAR)    ;  EXTERNAL ;
  PROCEDURE GENCODIG (GC1:STRING(3);GC2,GC3:CHAR; GC4,GC5:INTEGER)
                                 ;  EXTERNAL ;
  PROCEDURE EXPRESION ;

  CONST   NMXID = 21 ;       NMXDT = 21 ;    NMXCD = 501  ;
          NMXPR = 5  ;       LMXST = 20 ;    LMXHX = 5    ;
  TYPE    T1 = PACKED ARRAY (.1..10.) OF CHAR ;
          T2 = PACKED ARRAY (.1..73.) OF CHAR ;
          T3 = PACKED ARRAY (.1..5.)  OF CHAR ;
          T31= PACKED ARRAY (.1..NMXPR.) OF INTEGER ;
          T32= PACKED ARRAY (.1..LMXST.) OF CHAR ;
          T33= PACKED ARRAY (.1..LMXHX.) OF CHAR ;
          T30= PACKED ARRAY (.0..NMXID.) OF
               RECORD
                 NOMBRE : T1          ;
                 DIR    : INTEGER     ;
                 CASE TIPO:CHAR OF
                      'S': (SERIEHEX: PACKED ARRAY (.1..LMXST.) OF T33);
                      'A': (NSUB,SUB1,SUB2: INTEGER) ;
                      'F': (NPAR:0..NMXPR; CDFU:INTEGER; DIRP:T31)
               END ;

  VAR   I : INTEGER   ; OPREL    : T1      ;   TPAUX : CHAR ;
                TPFACTOR,TPTERMINO,TPSIMPLE,CTRAUX : CHAR ;


  REF PLI    : BOOLEAN   ;  TIDENT   : T30    ;  TPEXPRE : CHAR ;
      CLASE  : STRING(5) ;  NUMERO   :INTEGER ;  SERIE   : T32  ;
```

```
    SIMB  : T1          ;   SALIDA   : TEXT   ;

PROCEDURE ESIMPLE   ;
   VAR   OPSUM : T1 ;
PROCEDURE ETERMINO  ;
   VAR   OPMUL : T1 ;
PROCEDURE EFACTOR   ;

PROCEDURE  CALLENTE  ;
  BEGIN /* CALLENTE  */
  GENCODIG ('CNU',' ','N',NUMERO,0) ;
  TPFACTOR:='N' ;       OBSIMB
  END /* CALLENTER */;


PROCEDURE  CALLCONST ;
  VAR   N1:INTEGER ;
  BEGIN /* CALLCONST */
  CONSTANTE ;          IF NOT PLI THEN OBSIMB ;
  TPFACTOR:='S' ;
  N1:=1 ;
  WHILE N1<=LMXST DO
    BEGIN /* B1 */
    CTRAUX:= SERIE1.N1.) ;
    PASEAHEX (CTRAUX)    ;
    GENCODIG ('CNU',' ','H',0,0) ;
    N1:=N1+1
    END /* B1 */
  END /* CALLCONST */ ;


PROCEDURE CALLVRBLE ;
  VAR   I:INTEGER;

  BEGIN /* CALLVRBLE */
  I:=POSICION;
  IF I=0 THEN FINLINE(83)  ELSE
    BEGIN /* B1 */
    OBSIMB ;
    IF (PLI=TRUE) ] (SIMB<>'(') THEN
      BEGIN
      WITH  TIDENT(.I.) DO
      IF (TIPO='V') THEN  GENCODIG ('CVA','V','N',DIR,0)
      ELSE  FINLINE (30)
      END      ELSE
    IF TIDENT(.I.).TIPO<>'A' THEN FINLINE(27)  ELSE
      BEGIN /* B2 */
      OBSIMB;
      IF PLI THEN FINLINE(5)      ELSE
        BEGIN /* B3 */
        EXPRESION ;
        IF TPEXPRE='S' THEN FINLINE(16)   ELSE
        IF PLI THEN FINLINE(6)   ELSE
        IF SIMB=')' THEN
          BEGIN
```

```
          IF TIDENT(.I.).NSUB<>1 THEN FINLINE(26)  ELSE
             BEGIN
             GENCODIG('ALM','V','N',2,0) ;
             WITH TIDENT(.I.) DO GENCODIG('CVA','A','N',DIR,2) ;
             OBSIMB
             END
          END      ELSE
        IF SIMB<>',' THEN FINLINE(7)  ELSE
          BEGIN  /* B4 */
          OBSIMB ;
          IF PLI THEN FINLINE(5)       ELSE
            BEGIN /* B5 */
            EXPRESION;
            IF TPEXPRE='S' THEN FINLINE(16)   ELSE
            IF PLI THEN FINLINE(6)     ELSE
            IF SIMB<>')' THEN  FINLINE(8)   ELSE
            IF TIDENT(.I.).NSUB<>2 THEN FINLINE(26)   ELSE
                 BEGIN
                 GENCODIG('CNU',' ','N',1,0) ;
                 GENCODIG('OPR',' ','N',9,0) ;
                 WITH TIDENT(.I.) DO  GENCODIG('CNU',' ','N',SUB1,0);
                 GENCODIG('OPR',' ','N',11,0);
                 GENCODIG('OPR',' ','N',8,0) ;
                 GENCODIG('ALM','V','N',2,0 );
                 WITH TIDENT(.I.) DO GENCODIG('CVA','A','N',DIR ,2);
                 OBSIMB
                 END
            END /* B5 */
          END /* B4 */
        END /* B3 */
      END /* B2 */
    END /* B1 */ ;
  TPFACTOR:='N'
  END /* CALLVRBLE */ ;

PROCEDURE CALLSTRIN ;

  BEGIN   /* CALLSTRIN */
  I:=POSICION;    OBSIMB;
  IF I=0 THEN FINLINE(83)  ELSE
    BEGIN
    WITH  TIDENT(.I.) DO
      IF (TIPO='S') THEN  GENCODIG ('CVA','S','N',DIR,0)
    END ;
  TPFACTOR:='S'
  END   /* CALLSTRIN */ ;


PROCEDURE CALLFUNCN ;
  VAR  I1:INTEGER ;
  BEGIN /* CALLFUNCN */
  I:=POSICION;  I1:=0 ;
  IF I=0 THEN FINLINE(83)    ELSE
    BEGIN /* B1 */
    OBSIMB ;
```

```
    IF (PLI=TRUE)|(SIMB<>'(') THEN
      BEGIN IF TIDENT(.I.).NPAR<>0 THEN FINLINE(81)
            ELSE WITH TIDENT(.I.) DO GENCODIG('CVA','V','N',DIR,0)
      END
    ELSE
      BEGIN /* B2 */
      REPEAT
        OBSIMB;
        IF PLI THEN FINLINE(9)   ELSE
          BEGIN
          EXPRESION;
          IF TPEXPRE='S' THEN FINLINE(16)   ELSE
          IF PLI THEN FINLINE(10)   ELSE
          IF (SIMB<>',') & (SIMB<>')') THEN  FINLINE(11)   ELSE
            BEGIN
            I1:=I1+1;
            IF I1>NMXPR THEN
              BEGIN
              FINLINE(87);
              WRITELN(SALIDA,'->EL # MAXIMO DE PARAMETROS '
                             'PERMITIDO ES : ',NMXPR);
              END     ELSE
              WITH TIDENT(.I.) DO GENCODIG('ALM','V','N',DIRP(.I1.),0)
            END
          END
      UNTIL (SIMB<>',') ;
      IF TIDENT(.I.).NPAR<>I1 THEN FINLINE(84)   ELSE
        BEGIN
        GENCODIG('LLS',' ','N',TIDENT(.I.).CDFU,0);
        GENCODIG('CVA','V','N',TIDENT(.I.).DIR,0 );
        IF NOT PLI THEN OBSIMB
        END
      END /* B2 */
    END /* B1 */   ;  TPFACTOR:='N'
  END /* CALLFUNCN */ ;

PROCEDURE CALLSUBEX ;
  BEGIN /* CALLSUBEX */
  OBSIMB ;
  IF PLI THEN FINLINE(10)  ELSE
    BEGIN
    EXPRESION;  TPFACTOR:=TPEXPRE ;
    IF PLI THEN FINLINE(13)  ELSE
    IF SIMB=')' THEN OBSIMB ELSE FINLINE(14)
    END
  END /* CALLSUBEX */ ;

PROCEDURE CALLNOT ;
  BEGIN /* CALLNOT */
  OBSIMB ;
  IF PLI THEN FINLINE(25)
  ELSE BEGIN
       EFACTOR ;
       IF TPFACTOR<>'L' THEN FINLINE(16)
       ELSE GENCODIG('OPR',' ','N',15,0)
```

```
      END ;
 TPFACTOR:='L'
 END /* CALLNOT */ ;

 BEGIN /* EFACTOR */
 TPFACTOR:=' ';
 IF (CLASE='ENTER') THEN  CALLENTER   ELSE
 IF (SIMB = '"'   ) THEN  CALLCONST   ELSE
 IF (CLASE='VRBLE') THEN  CALLVRBLE   ELSE
 IF (CLASE='STRIN') THEN  CALLSTRIN   ELSE
 IF (CLASE='FUNCN') THEN  CALLFUNCN   ELSE
 IF (SIMB = '('   ) THEN  CALLSUBEX   ELSE
 IF (SIMB = 'NOT' ) THEN  CALLNOT     ELSE   FINLINE (15)
 END /* EFACTOR */ ;


 BEGIN /* ETERMINO */
 EFACTOR ;     TPAUX:=TPFACTOR ;   TPTERMINO:=TPFACTOR;
 IF NOT PLI THEN
 WHILE (SIMB='*')|(SIMB='/')|(SIMB='AND')|(SIMB='|') DO
   BEGIN /* B1 */
   OPMUL:=SIMB;   OBSIMB;
   IF PLI THEN FINLINE(17)   ELSE
     BEGIN /* B2 */
     EFACTOR ;
     IF (TPAUX<>TPFACTOR) THEN FINLINE (16) ELSE
     IF OPMUL='*' THEN
       BEGIN GENCODIG('OPR',' ','N',11,0); TPTERMINO:='N' END ELSE
     IF OPMUL='/' THEN
       BEGIN GENCODIG('OPR',' ','N',12,0); TPTERMINO:='N' END ELSE
     IF OPMUL='|' THEN
       BEGIN GENCODIG('OPR',' ','N',14,0); TPTERMINO:='S' END
     ELSE    BEGIN GENCODIG('OPR',' ','N',13,0); TPTERMINO:='L' END
     END /* B2 */
   END /* B1 */
 END /* ETERMINO */ ;

 BEGIN /* ESIMPLE */
 IF (SIMB='+')|(SIMB='-') THEN
   BEGIN /* B1 */
   OPSUM:=SIMB;   OBSIMB;
   IF PLI THEN FINLINE(17) ELSE
     BEGIN  /* B2 */
     ETERMINO ;
     TPAUX:= TPTERMINO ;       TPSIMPLE:= TPTERMINO ;
     IF (TPTERMINO<>'N') THEN FINLINE (16) ELSE
     IF (OPSUM='-')  THEN GENCODIG ('OPR',' ','N',7,0)
     END /* B2 */
   END /* B1 */
 ELSE
   BEGIN /* B1 */
   ETERMINO ;
   TPAUX:= TPTERMINO ;       TPSIMPLE:= TPTERMINO ;
   END /* B2 */ ;
 WHILE (SIMB='+')|(SIMB='-')|(SIMB='OR') DO
```

```
    BEGIN /* B1 */
    IF (TPAUX='S') THEN FINLINE (16) ELSE
      BEGIN /* B2 */
      OPSUM:=SIMB;     OBSIMB;
      IF PLI THEN FINLINE(17)     ELSE
        BEGIN /* B3 */
        ETERMINO ;
        IF (TPAUX<>TPTERMINO) THEN FINLINE (16) ELSE
        IF OPSUM='+' THEN
          BEGIN GENCODIG('OPR',' ','N',8,0); TPSIMPLE:='N' END ELSE
        IF OPSUM='-' THEN
          BEGIN GENCODIG('OPR',' ','N',9,0); TPSIMPLE:='N' END
        ELSE  BEGIN GENCODIG('OPR',' ','N',10,0); TPSIMPLE:='L' END
        END /* B3 */
      END /* B2 */
    END /* B1 */
  END /* ESIMPLE */ ;

  BEGIN /* EXPRESION */
  ESIMPLE ;
  TPEXPRE:= TPSIMPLE ;      TPAUX:= TPSIMPLE ;
  IF NOT PLI THEN
  IF (SIMB='< ')|(SIMB='<=')|(SIMB='=')|(SIMB='<>')|(SIMB='>=')|
     (SIMB='> ') THEN
    BEGIN /* B1 */
    IF (TPAUX<>'N') THEN FINLINE (16) ELSE
      BEGIN /* B2 */
      OPREL:=SIMB;      OBSIMB ;
      IF PLI THEN FINLINE(19)  ELSE
        BEGIN /* B3 */
        ESIMPLE ;
        IF (TPAUX<>TPSIMPLE) THEN FINLINE(16) ELSE
          BEGIN /* B4 */
          TPEXPRE:= 'L' ;
          IF OPREL='< ' THEN GENCODIG('OPR',' ','N',1,0) ELSE
          IF OPREL='<=' THEN GENCODIG('OPR',' ','N',2,0) ELSE
          IF OPREL='= ' THEN GENCODIG('OPR',' ','N',3,0) ELSE
          IF OPREL='<>' THEN GENCODIG('OPR',' ','N',4,0) ELSE
          IF OPREL='>=' THEN GENCODIG('OPR',' ','N',5,0)
                        ELSE GENCODIG('OPR',' ','N',6,0)
          END /* B4 */
        END /* B3 */
      END /* B2 */
    END /* B1 */
  END /* EXPRESION */ ;
```

```
==========================================================================
==========================================================================
====                                                                  ===
====      ESCUELA POLITECNICA NACIONAL                                ===
====      FACULTAD DE INGENIERIA ELECTRICA                            ===
====      TESIS DE GRADO                                              ===
====                                                                  ===
====      ALUMNO :   TORO LORGIO                                      ===
====                                                                  ===
====                                                                  ===
==========================================================================
==========================================================================
```



```
/*--- ANALISIS ---------------------------------------------------------

SEGMENT ANALISIS;

 PROCEDURE ANALISIS  ; EXTERNAL ;
 PROCEDURE OBSIMB    ; EXTERNAL ;
 PROCEDURE OBCTR     ; EXTERNAL ;
 PROCEDURE CONSTANTE ; EXTERNAL ;
 PROCEDURE EXPRESION ; EXTERNAL ;
 FUNCTION  POSICION:INTEGER     ; EXTERNAL ;
 PROCEDURE PASEAHEX(DC:CHAR    ); EXTERNAL ;
 PROCEDURE ERROR   (NE:INTEGER); EXTERNAL ;
 PROCEDURE FINLINE (FL:INTEGER); EXTERNAL ;
 PROCEDURE GENCODIG (GC1:STRING(3); GC2,GC3:CHAR; GC4,GC5:INTEGER)
                                 ; EXTERNAL;

PROCEDURE ANALISIS ;

/*--- DECLARACIONES ----------------------------------------------------*/

 CONST   NMXID = 21 ;       NMXDT = 21 ;     NMXCD = 501  ;
         NMXPR = 5  ;       NMXNP = 101;     NMXGT = 21
         LMXST = 20 ;       NMXAS =1001;     LMXHX = 5
 TYPE    T1 = PACKED ARRAY (.1..10.)     OF CHAR ;
         T2 = PACKED ARRAY (.1..73.)     OF CHAR ;
         T3 = PACKED ARRAY (.1..5.)      OF CHAR ;
         T31= PACKED ARRAY (.1..NMXPR.) OF INTEGER ;
         T32= PACKED ARRAY (.1..LMXST.) OF CHAR    ;
         T33= PACKED ARRAY (.1..LMXHX.)     OF CHAR    ;
         T30= PACKED ARRAY (.0..NMXID.) OF
              RECORD
                NOMBRE : T1          ;
                DIR    : INTEGER     ;
                CASE TIPO:CHAR OF
                     'S': (SERIEHEX: PACKED ARRAY (.1..LMXST.) OF T33);
```

```
                    'A': (NSUB,SUB1,SUB2: INTEGER) ;
                    'F': (NPAR:0..NMXPR; CDFU:INTEGER; DIRP:T31)
            END ;
        T40= PACKED ARRAY (.0..NMXDT.) OF
            RECORD
              TIPOD  : CHAR      ;
              DATOH  : T33       ;
              DATON  : INTEGER   ;
              DATST  : T32
             END ;
        T51= RECORD
              COPRDR : STRING(3) ;
              TIPVAR : CHAR      ;
              CTOPND : CHAR      ;
              COPHEX : T33       ;
              COPNUM : INTEGER   ;
              CREGIX : INTEGER
            END ;
        T50= PACKED ARRAY (.0..NMXCD.) OF T51 ;
        T60= PACKED ARRAY (.0..NMXNP.) OF
            RECORD   NPNUP,NPCCD: INTEGER   END;
        T70= PACKED ARRAY (.0..NMXGT.) OF
            RECORD   GTNUP,GTCCD: INTEGER   END;
        T81 = RECORD
                CMEMOR : INTEGER   ;
                OPRDOR : STRING(4) ;
                TIPDIR : CHAR      ;
                TIPOPN : CHAR      ;
                OPNHEX : T33       ;
                OPNNUM : INTEGER   ;
                INDICE : STRING(2)
              END ;
        T80 = PACKED ARRAY (.0..NMXAS.) OF T81      ;
REF     CC,LL,NUMERD   : INTEGER   ;     CTR    : CHAR    ;
        SALIDA,FUENTE  : TEXT      ;     PCI    : BOOLEAN ;
        CLASE          : STRING(5) ;     SIMB   : T1      ;
        LINEA,LINEAUX  : T2        ;     CMI    : INTEGER ;
        NAMEPROG       : T1        ;     NUMHEX : T33     ;
        SERIE          : T32       ;     TPCONS : CHAR    ;
        DPL            : INTEGER   ;     TPEXPRE: CHAR    ;
        CID            : 0..NMXID  ;     TIDENT : T30     ;
        CDT            : 0..NMXDT  ;     TDATOS : T40     ;
        CCD            : 0..NMXCD  ;     TCODIG : T50     ;
        CNP            : 0..NMXNP  ;     TNUPRO : T60     ;
        CGT            : 0..NMXGT  ;     TGOTOS : T70     ;
        CAS            : 0..NMXAS  ;     TASSEM : T80     ;
VAR     CDTAUX         : 0..NMXDT  ;

/*--- LLENARGT ------------------------------------------------------*/

PROCEDURE LLENARGT ;
 VAR    I1,I2: INTEGER ;
 BEGIN /* LLENARGT */
 I1:=1 ;
 WHILE I1<=CGT  DO
```

```
      BEGIN /* B1 */
      TNUPRO(.0.).NPNUP:= TGOTOS(.I1.).GTNUP ;      I2:=CNP ;
      WHILE TNUPRO(.I2.).NPNUP<>TGOTOS(.I1.).GTNUP DO  I2:=I2-1 ;
      IF I2=0 THEN
        BEGIN
        ERROR(86) ;
        WRITELN(SALIDA,'-> LA PROPOSICION QUE NO EXISTE es LA',
                       ' NUMERO',TGOTOS(.I1.).GTNUP:5)
        END
      ELSE WITH TGOTOS(.I1.) DO
           TCODIG(.GTCCD.).COPNUM := TNUPRO(.I2.).NPCCD ;
      I1:=I1+1
      END /* B1 */
    END /* LLENARGT */ ;

/*--- PONERNP PONERGT PONERID PONERDT ------------------------------*/

PROCEDURE PONERNP (NP:INTEGER) ;
  BEGIN /* PONERNP */
  CNP:= CNP+1 ;
  IF CNP<NMXNP THEN
    WITH TNUPRO(.CNP.) DO  BEGIN NPNUP:=NP; NPCCD:=CCD END
  ELSE
    BEGIN
    FINLINE(91);
    WRITELN(SALIDA,'-> EL NUMERO MAXIMO DE PROPOSICIONES ',
                   'PERMITIDO ES :',(NMXNP-1):5)
    END
  END /* PONERNP */ ;

PROCEDURE PONERGT (GT:INTEGER) ;
  BEGIN /* PONERNP */
  CGT:= CGT+1 ;
  IF CGT<NMXGT THEN
       WITH TGOTOS(.CGT.) DO  BEGIN GTNUP:=GT; GTCCD:=CCD END
  ELSE
    BEGIN
    FINLINE(92);
    WRITELN(SALIDA,'-> EL NUMERO MAXIMO DE TRANSFERENCIAS ',
                   'INCONDICIONALES PERMITIDO ES :',(NMXGT-1):5)
    END
  END /* PONERNP */ ;

PROCEDURE PONERID (TP:CHAR) ;
  VAR     I   : INTEGER;

  BEGIN /* PONERID */
  CID:= CID+1 ;
  IF CID<NMXID THEN
    WITH TIDENT (.CID.) DO
      BEGIN /* B1 */
      NOMBRE := SIMB ;    DIR := CMI ;        TIPO := TP ;
      CASE TIPO OF
        'V': CMI:=CMI+1                    ;
```

```
        'S': BEGIN
             CMI:= CMI+LMXST ;
             I:= 1           ;
             WHILE (I<=LMXST) DO BEGIN SERIEHEX(.I.):=' '; I:=I+1 END
             END ;
        'A': BEGIN NSUB:=1; SUB1:=1; SUB2:=1      END   ;
        'F': BEGIN
             NPAR:=0;      CDFU:=CCD;       CMI:=CMI+1;      I:=1;
             WHILE I<=NMXPR DO BEGIN DIRP(.I.):=C; I:=I+1 END
             END
      END /* CASE */
      END /* B1 */
  ELSE
    BEGIN
    FINLINE(88);
    WRITELN(SALIDA,'-> EL NUMERO MAXIMO DE IDENTIFICADORES ',
                   'PERMITIDO ES :',(NMXID-1):5)
    END
  END /* PONERIC */ ;

PROCEDURE PONERDT (TD1:CHAR);

  BEGIN /* PONERDT */
  IF TD1='N' THEN BEGIN NUMHEX:=' '; SERIE:=' ' END  ELSE
    BEGIN
    NUMERO:=0 ;
    IF TD1='H' THEN SERIE:=' ' ELSE NUMHEX:=' ';
    END ;
  CDT:=CDT+1;
  IF CDT<NMXDT THEN
    BEGIN /* B1 */
    WITH TDATOS(.CDT.) DO
      BEGIN
      TIPOD:=TD1; DATOH:=NUMHEX; DATON:=NUMERO; DATST:=SERIE
      END
    END /* B1 */
  ELSE
    BEGIN
    FINLINE(89);
    WRITELN(SALIDA,'-> EL NUMERO MAXIMO DE DATOS ',
                   'PERMITIDO ES :',(NMXDT-1):5)
    END
  END  /* PONERDT */  ;

/*---- EMPIECE LINEAMAL ----------------------------------------------------*/

  PROCEDURE EMPIECE ;
    BEGIN /* EMPIECE */
    IF EOF(INPUT)=FALSE THEN
      BEGIN /* B1 */
      REPEAT     PLI:=FALSE ;       LINEAUX:=LINEA ;    OBSIMB ;
                 IF PLI THEN FINLINE(33)
      UNTIL  (PLI=FALSE) ;
      IF CLASE='PRESV' THEN
        BEGIN
```

```
        PONERNP(NUMERO) ;
        IF SIMB='REM' THEN  BEGIN WHILE NOT PLI DO OBSIMB; EMPIECE END
        END
      ELSE
        BEGIN  WRITELN(SALIDA,LINEA) ;     ERROR(34);
               WHILE NOT PLI DO OBSIMB ;   EMPIECE
        END
      END /* B1 */
    END   /* EMPIECE */ ;

  PROCEDURE LINEAMAL ; BEGIN FINLINE(34); EMPIECE END ;

/*--- P_PROGRAM ----------------------------------------------------------*/

  PROCEDURE P_PROGRAM ;

  VAR   I   :INTEGER ;

  BEGIN /* P_PROGRAM */
  WHILE (CTR=' ')&(NOT PLI) DO OBCTR ;
  IF PLI THEN BEGIN OBSIMB; NAMEPROG:='BASIC' END  ELSE
  IF NOT (CTR IN ['A'..'Z']) THEN FINLINE(85)   ELSE
    BEGIN
    I:=1;
    WHILE I<=5 DO BEGIN NAMEPROG[.I.]:=CTR; OBCTR; I:=I+1 END ;
    WHILE CTR<>' ' DO OBCTR;   OBSIMB;
    IF NOT PLI THEN FINLINE(77)
    END
  END /* P_PROGRAM */ ;

/*--- BLOQUE -------------------------------------------------------------*/

  PROCEDURE BLOQUE ;


    PROCEDURE PROPOSICION ;


/*--- P_FOR --------------------------------------------------------------*/

  PROCEDURE P_FOR ;
    VAR     I:INTEGER;     CCD1,CCD2:0..NMXCD;

  PROCEDURE P_FORTO ;

    BEGIN /* P_FORTO */
    OBSIMB ;
    IF PLI THEN                 FINLINE(60)     ELSE
    IF CLASE<>'VRBLE' THEN      FINLINE(58)     ELSE
      BEGIN /* B1 */
      I := POSICION ;
      IF (I=0) THEN             FINLINE(83)     ELSE
      IF TIDENT[.I.].TIPO<>'V' THEN FINLINE(30)   ELSE
        BEGIN /* B2 */
        OBSIMB ;
```

```
      IF PLI THEN              FINLINE(60)      ELSE
      IF SIMB<>'=' THEN        FINLINE(61)      ELSE
        BEGIN /* B3 */
        OBSIMB ;
        IF PLI THEN            FINLINE(60)      ELSE
          BEGIN /* B4 */
          EXPRESION ;      CCD1:=CCD;
          IF (TPEXPRE<>'N') THEN FINLINE (16) ELSE
          IF PLI THEN          FINLINE(62)      ELSE
          IF SIMB<>'TO' THEN   FINLINE(63)      ELSE
            BEGIN/* B5 */
            WITH TIDENT(.I.) DO GENCODIG('ALM','V','N',DIR,0) ;
            OBSIMB ;
            IF PLI THEN        FINLINE(60)      ELSE
              BEGIN
              WITH TIDENT(.I.) DO GENCODIG('CVA','V','N',DIR,0) ;
              EXPRESION;
              IF (TPEXPRE<>'N') THEN FINLINE (16) ELSE
              IF NOT PLI THEN FINLINE(64)      ELSE
                BEGIN
                GENCODIG('OPR',' ','N',2,0);      CCD2:=CCD;
                GENCODIG('SAC',' ','N',0,0)
                END
              END
            END /* B5 */
          END /* B4 */
        END /* B3 */
      END /* B2 */
    END /* B1 */
  END /* P_FORTO */ ;


  PROCEDURE P_NEXT  ;

  BEGIN /* P_NEXT */
  OBSIMB ;
  IF NOT PLI THEN   FINLINE(59)     ELSE
  IF TIDENT(.I.). TIPO = 'V' THEN
    BEGIN
    WITH TIDENT(.I.) DO GENCODIG('CVA','V','N',DIR,0) ;
    GENCODIG('CNU',' ','N',1,0);       GENCODIG('OPR',' ','N',8,0) ;
    GENCODIG('SAI',' ','N',CCD1,0);    TCODIG(.CCD2.).CAMPNUM:=CCD;
    END
  END /* P_NEXT */ ;

  BEGIN /* P_FOR */
  P_FORTO; EMPIECE;
  REPEAT   PROPOSICION
  UNTIL  ((SIMB='NEXT')|(SIMB='END')|(EOF(INPUT))) ;
  IF SIMB='NEXT' THEN BEGIN P_NEXT; EMPIECE END
  ELSE BEGIN IF SIMB='END' THEN FINLINE(0); ERROR(56) END
  END /* P_FOR */ ;

/*--- P_WHILE ------------------------------------------------------------*/
```

```
  PROCEDURE P_WHILE ;
    VAR     CCD1,CCD2:0..NMXCD;

  PROCEDURE P_WHILEDO ;

  BEGIN /* P_WHILEDO */
  CCD1:=CCD;       OBSIMB;
  IF PLI THEN    FINLINE(60)     ELSE
    BEGIN /* B1 */
    EXPRESION ;
    IF (TPEXPRE<>'L') THEN FINLINE (16) ELSE
    IF PLI THEN    FINLINE(68)      ELSE
      BEGIN
      CCD2:=CCD;          GENCODIG('SAC',' ','N',0,0) ;
      IF SIMB<>'DO' THEN FINLINE(79)     ELSE
        BEGIN OBSIMB; IF NOT PLI THEN FINLINE(59)  END
      END
    END /* B1 */
  END /* P_WHILEDO */ ;


  PROCEDURE P_ENDWHILE ;

  BEGIN /* P_ENDWHILE */
  OBSIMB ;
  IF NOT PLI THEN FINLINE(59)        ELSE
    BEGIN
    GENCODIG('SAI',' ','N',CCD1,0) ;
    TCODIG(.CCD2.).COPNUM:=CCD
    END
  END /* P_ENDWHILE */ ;

  BEGIN /* P_WHILE */
  P_WHILEDO;  EMPIECE;
  REPEAT   PROPOSICION
  UNTIL  ((SIMB='ENDWHILE')|(SIMB='END')|(EOF(INPUT))) ;
  IF SIMB='ENDWHILE' THEN BEGIN P_ENDWHILE; EMPIECE END
  ELSE BEGIN IF SIMB='END' THEN FINLINE(0); ERROR(78) END
       END /* P_WHILE */ ;

/*--- P_IF ----------------------------------------------------------

  PROCEDURE P_IF ;
    VAR      CCD1,CCD2:0..NMXCD;


  PROCEDURE P_IFTHEN ;

  BEGIN /* P_IFTHEN */
  OBSIMB ;
  IF PLI THEN          FINLINE(60)  ELSE
    BEGIN /* B1 */
    EXPRESION ;
    IF (TPEXPRE<>'L') THEN FINLINE (16) ELSE
    IF PLI THEN       FINLINE(68)  ELSE
```

```
      BEGIN
      CCD1:=CCD;      CCD2:=CCD;
      GENCODIG('SAC',' ','N',0,0) ;
      IF SIMB<>'THEN' THEN  FINLINE(69)      ELSE
        BEGIN OBSIMB; IF NOT PLI THEN FINLINE(59)  END
      END
    END /* B1 */
  END /* P_IFTHEN */ ;

  PROCEDURE P_ELSE ;
    BEGIN /* P_ELSE */
    OBSIMB ;
    IF NOT PLI THEN   FINLINE(59)      ELSE
      BEGIN
      CCD2:=CCD;    GENCODIG('SAI',' ','N',0,0);
      TCODIG(.CCD1.).COPNUM:=CCD ;
      END
    END /* P_ELSE */ ;

  PROCEDURE P_ENDIF ;
    BEGIN /* P_ENDIF */
    OBSIMB ;
    IF NOT PLI THEN   FINLINE(59)
    ELSE TCODIG(.CCD2.).COPNUM:=CCD
    END /* P_ENDIF */ ;


    BEGIN /* P_IF  */
    P_IFTHEN;  EMPIECE;
    REPEAT   PROPOSICION
    UNTIL  ((SIMB='ENDIF')|(SIMB='ELSE')|(SIMB='END')|(EOF(INPUT)));
    IF (SIMB<>'ENDIF')&(SIMB<>'ELSE')  THEN
       BEGIN IF SIMB='END' THEN FINLINE(0); ERROR(57) END
    ELSE
    IF SIMB='ENDIF' THEN BEGIN P_ENDIF; EMPIECE END  ELSE
      BEGIN /* B1 */
      P_ELSE;  EMPIECE;
      REPEAT PROPOSICION
      UNTIL ((SIMB='ENDIF')|(SIMB='END')|(EOF(INPUT))) ;
      IF SIMB='ENDIF' THEN BEGIN P_ENDIF; EMPIECE END
      ELSE BEGIN IF SIMB='END' THEN FINLINE(0); ERROR(57) END
      END   /* B1 */
    END /* P_IF */ ;

/*--- P_CASE ----------------------------------------------------------

  PROCEDURE P_CASE ;
    VAR     I:INTEGER ;  CCD1:0..NMXCD;

    PROCEDURE P_CASEOF ;

    BEGIN /* P_CASEOF */
    OBSIMB;
    IF PLI THEN   FINLINE(60)        ELSE
      BEGIN /* B1 */
```

```
      EXPRESION ;    GENCODIG('ALM','V','N',1,0);
      IF (TPEXPRE<>'N') THEN FINLINE(16) ELSE
      IF PLI THEN FINLINE(62)          ELSE
      IF SIMB<>'OF' THEN FINLINE(70) ELSE
        BEGIN /* B2 */
        I:=1;
        REPEAT
         OBSIMB ;
         IF PLI THEN    FINLINE(71)  ELSE
         IF CLASE<>'ENTER' THEN       FINLINE(72)   ELSE
         BEGIN /* B3 */
         GENCODIG('CVA','V','N',1,0);  GENCODIG('CNU',' ','N',I,0);
         GENCODIG('OPR',' ','N',3,0); GENCODIG('SAC',' ','N',(CCD+2),0);
         PONERGT (NUMERO)    ;          GENCODIG('SAI',' ','N',0,0);
         OBSIMB;                      I:=I+1;
         IF NOT PLI THEN
         IF SIMB<>',' THEN            FINLINE(73)
         END /* B3 */
        UNTIL (PLI=TRUE)
        END /* B2 */
      END /* B1 */ ;
    CCD1:=CCD ;                       GENCODIG('SAI',' ','N',0,0);
    END /* P_CASEOF */ ;

  PROCEDURE P_ENDCASE ;
    BEGIN /* P_ENDCASE */
    OBSIMB ;
    IF NOT PLI THEN                   FINLINE(59)
    ELSE  TCODIG(.CCD1.).COPNUM:=CCD
    END /* P_ENDCASE */ ;


    BEGIN /* P_CASE */
    P_CASEOF;  EMPIECE;
    REPEAT   PROPOSICION
    UNTIL  ((SIMB='ENDCASE')|(SIMB='END')|(EOF(INPUT))) ;
    IF SIMB='ENDCASE' THEN BEGIN P_ENDCASE;   EMPIECE   END
    ELSE BEGIN IF SIMB='END' THEN FINLINE(0); ERROR(60) END
    END /* P_CASE */ ;

/*--- P_GOTOSUB ----------------------------------------------------------*/

  PROCEDURE P_GOTOSUB ;
    VAR      PR:TI;

  BEGIN /* P_GOTOSUB */
  PR:=SIMB;                OBSIMB ;
  IF PLI THEN              FINLINE(60)     ELSE
  IF CLASE<>'ENTER' THEN   FINLINE(66)     ELSE
    BEGIN
    PONERGT(NUMERO);       OBSIMB;
    IF NOT PLI THEN        FINLINE(67)     ELSE
    IF PR='GOSUB' THEN GENCODIG('LLS',' ','N',0,0)
    ELSE  GENCODIG('SAI',' ','N',0,0)
    END
```

```
  END /* P_GOTOSUB */ ;

/*--- P_RESTORE ------------------------------------------------------------*/

  PROCEDURE P_RESTORE ;
    BEGIN /* P_RESTORE */
    OBSIMB;   IF NOT PLI THEN FINLINE(59)  ELSE CDTAUX:=0
    END /* P_RESTORE */ ;

/*--- P_LET ----------------------------------------------------------------*/

  PROCEDURE P_LET ;
    VAR  N1,N2,I: INTEGER ;          I1:BOOLEAN;
         TPLET: CHAR    ;

  PROCEDURE VARLET ;

    BEGIN /* VARLET */
      OBSIMB ;
      IF PLI THEN      FINLINE(60)      ELSE
      IF (SIMB<>'(') THEN
        BEGIN IF TIDENT(.I.).TIPO<>'V' THEN   FINLINE(30)  END   ELSE
      IF TIDENT(.I.).TIPO<>'A' THEN      FINLINE(27)      ELSE
        BEGIN /* B2 */
        OBSIMB;    I1:=FALSE ;
        IF PLI THEN    FINLINE(5)       ELSE
          BEGIN /* B3 */
          EXPRESION ;
          IF (TPEXPRE<>'N'  ) THEN FINLINE(16)  ELSE
          IF PLI THEN  FINLINE(6)       ELSE
          IF SIMB=')' THEN
            BEGIN
            IF TIDENT(.I.).NSUB<>I THEN FINLINE(26)      ELSE
               BEGIN
               GENCODIG('ALM','V','N',I,0); OBSIMB;
               IF PLI THEN FINLINE(60)
               END
            END     ELSE
          IF SIMB<>',' THEN          FINLINE(7)      ELSE
            BEGIN  /* B4 */
            OBSIMB ;
            IF PLI THEN              FINLINE(5)      ELSE
              BEGIN /* B5 */
              EXPRESION;
              IF (TPEXPRE<>'N'  ) THEN FINLINE(16)  ELSE
              IF PLI THEN              FINLINE(6)      ELSE
              IF SIMB<>')' THEN        FINLINE(8)      ELSE
              IF TIDENT(.I.).NSUB<>2 THEN          FINLINE(26) ELSE
                BEGIN
                GENCODIG('CNU',' ','N',1,0) ;
                GENCODIG('OPR',' ','N',9,0) ;
                WITH TIDENT(.I.) DO  GENCODIG('CNU',' ','N',SUB1,0) ;
                GENCODIG('OPR',' ','N',11,0);
                GENCODIG('OPR',' ','N',8,0) ;
                GENCODIG('ALM','V','N',1,0 );  OBSIMB                ;
```

```
BEGIN /* 0 */
I:=POSICION;
IF I=0 THEN        FINLINE(83)      ELSE
  BEGIN /* B1 */
  OBSIMB ;
  IF (PLI=TRUE) | (SIMB<>'(') THEN
    BEGIN
    IF TIDENT(.I.).TIPO<>'V' THEN  FINLINE(30)     ELSE
      BEGIN
      WITH TDATOS(.CDTAUX.) DO
        BEGIN
        NUMHEX:= DATOH;
        GENCODIG('CNU',' ',TIPOD, DATON,0)
        END;
      WITH TIDENT(.I.) DO GENCODIG('ALM','V','N',DIR,0)
      END
    END      ELSE
  IF TIDENT(.I.).TIPO<>'A' THEN     FINLINE(27)      ELSE
    BEGIN /* B2 */
    OBSIMB;
    IF PLI THEN    FINLINE(5)       ELSE
      BEGIN /* B3 */
      EXPRESION ;
      IF PLI THEN FINLINE(6)        ELSE
      IF SIMB=')' THEN
        BEGIN
        IF TIDENT(.I.).NSUB<>1 THEN FINLINE(26)      ELSE
           BEGIN
           GENCODIG('ALM','V','N',1,0) ;
           WITH TDATOS(.CDTAUX.) DO
             BEGIN
             NUMHEX:= DATOH;
             GENCODIG('CNU',' ',TIPOD, DATON,0)
             END;
           WITH TIDENT(.I.) DO GENCODIG('ALM','V','N',DIR,1) ;
           OBSIMB
           END
        END      ELSE
      IF SIMB<>',' THEN FINLINE(7) ELSE
        BEGIN  /* B4 */
        OBSIMB ;
        IF PLI THEN FINLINE(5)      ELSE
          BEGIN /* B5 */
          EXPRESION;
          IF PLI THEN       FINLINE(6)     ELSE
          IF SIMB<>')' THEN FINLINE(8)     ELSE
          IF TIDENT(.I.).NSUB<>2 THEN      FINLINE(26)  ELSE
            BEGIN
            GENCODIG('CNU',' ','N',1,0) ;
            GENCODIG('OPR',' ','N',9,0)  ;
            WITH TIDENT(.I.) DO    GENCODIG('CNU',' ','N',SJB1,0) ;
            GENCODIG('OPR',' ','N',11,0);
            GENCODIG('OPR',' ','N',8,0)  ;
            GENCODIG('ALM','V','N',1,0 );
            WITH TDATOS(.CDTAUX.) DO
```

```
                BEGIN
                NUMHEX:= DATOH;
                GENCODIG('CNU',' ',TIPOD, DATON,0)
                END;
              WITH TIDENT(.I.) DO  GENCODIG('ALM','A','N',DIR ,1) ;
              OBSIMB
              END
            END /* B5 */
          END /* B4 */
        END /* B3 */
      END /* B2 */
    END /* B1 */
  END /* 0 */
  END /* VARREAD */ ;

PROCEDURE READSTRING ;
  VAR  N1,I,N2 : INTEGER ;
       CTRAUX  : CHAR    ;
  BEGIN /* READSTRING */
  I:=POSICION;
  IF I=0 THEN      FINLINE(83)      ELSE
    BEGIN /* B1 */
    N2:= TIDENT(.I.).DIR ;
    N1:=0 ;
    IF (TDATOS(.CDTAUX.).TIPOD<>'S') THEN FINLINE(23) ELSE
    WHILE N1<LMXST  DO
      BEGIN
      CTRAUX:= TDATOS(.CDTAUX.).DATST(.(N1+1).) ;
      PASEAHEX (CTRAUX)                         ;
      TIDENT(.I.).SERIEHEX(.(N1+1).):= NUMHEX   ;
      GENCODIG ('CNU',' ','H',0,0)              ;
      GENCODIG ('ALM','V','N',(N2+N1),0)
      N1:=N1+1
      END
    END /* B1 */  ;
  OBSIMB
  END /* READSTRING */ ;

BEGIN /* P_READ   */
REPEAT
  OBSIMB ;
  IF PLI THEN                  FINLINE(38)     ELSE
  IF (CLASE<>'VRBLE')&(CLASE<>'STRIN')  THEN FINLINE(39) ELSE
    BEGIN /* B1 */
    CDTAUX:= CDTAUX+1 ;
    IF (CDTAUX > CDT)  THEN FINLINE(94)      ELSE
      BEGIN /* B2 */
      IF (CLASE='VRBLE') THEN VARREAD  ELSE READSTRING ;
      IF (NOT PLI)&(SIMB<>',') THEN    FINLINE(40)
      END /* B2 */
    END /* B1 */
UNTIL (PLI=TRUE)
END /* P_READ */ ;
```

```
/*--- PROPOSICION ----------------------------------------------------------*/

      BEGIN /* PROPOSICION */
      IF SIMB='FOR      ' THEN          P_FOR                          ELSE
      IF SIMB='WHILE    ' THEN          P_WHILE                        ELSE
      IF SIMB='IF       ' THEN          P_IF                           ELSE
      IF SIMB='CASE     ' THEN          P_CASE                         ELSE
      IF SIMB='GOTO     ' THEN BEGIN P_GOTOSUB; EMPIECE END  ELSE
      IF SIMB='GOSUB    ' THEN BEGIN P_GOTOSUB; EMPIECE END  ELSE
      IF SIMB='RESTORE  ' THEN BEGIN P_RESTORE; EMPIECE END  ELSE
      IF SIMB='LET      ' THEN BEGIN P_LET    ; EMPIECE END  ELSE
      IF SIMB='READ     ' THEN BEGIN P_READ   ; EMPIECE END
      ELSE  LINEAMAL
      END /* PROPOSICION */ ;

/*--- P-DEFFUN -------------------------------------------------------------*/

  PROCEDURE P_DEFFUN ;

  PROCEDURE P_DEF ;
  VAR   I,I1  :INTEGER  ;

  BEGIN /* P_DEF */
  OBSIMB ;
  IF PLI THEN     FINLINE(49)     ELSE
  IF (CLASE<>'FUNCN')  THEN       FINLINE(50) ELSE
    BEGIN /* B1 */
    I := POSICION ;
    IF I<>0  THEN FINLINE(82)     ELSE
      BEGIN /* B2 */
      PONERID('F');   OBSIMB ;
      IF NOT PLI THEN
        BEGIN /* B3 */
        IF SIMB<>'(' THEN           FINLINE(51)  ELSE
        , BEGIN /* B4 */
          I1:=0 ;
          REPEAT
            OBSIMB ;
            IF PLI THEN             FINLINE(52)  ELSE
            IF (CLASE<>'VRBLE')  THEN  FINLINE(53) ELSE
              BEGIN /* B5 */
              I:=POSICION ;
              IF I=0 THEN        FINLINE(83)  ELSE
              IF TIDENT(.I.).TIPO<>'V' THEN   FINLINE(30)  ELSE
                BEGIN
                I1:=I1+1;
                IF I1>NMXPR THEN
                  BEGIN
                  FINLINE(87);
                  WRITELN(SALIDA,'-> EL NUMERO MAXIMO DE PARAMETROS',
                                 ' PERMITIDO ES :', NMXPR:5)
                  END
                ELSE BEGIN
                    TIDENT(.CID.).DIRP(.I1.):= TIDENT(.I.).DIR ;
                    OBSIMB; IF PLI THEN      FINLINE(44)
```

```
                    END
                  END
                END /* B5 */
            UNTIL (SIMB<>',') ;
            IF NOT PLI THEN
            IF SIMB<>')' THEN          FINLINE(54)      ELSE
              BEGIN
              OBSIMB;IF NOT PLI THEN FINLINE(55)      ;
              TIDENT(.CID.).NPAR:= I1
              END
            END /* B4 */
          END /* B3 */
        END /* B2 */
      END /* B1 */
    END /* P_DEF */ ;

    BEGIN /* P_DEFFUN */
    P_DEF;  EMPIECE ;
    REPEAT PROPOSICION
    UNTIL((SIMB='ENDFN')|(SIMB='END')|(EOF(INPUT))) ;
    IF SIMB='ENDFN' THEN
      BEGIN
      OBSIMB;
      IF NOT PLI THEN FINLINE(59) ELSE GENCODIG('RTS',' ','N',0,0);
      EMPIECE
      END
    ELSE BEGIN IF SIMB='END' THEN FINLINE(0); ERROR(47) END
    END /* P_DEFFUN */ ;

/*--- P_SUBRT ------------------------------------------------------------*/

    PROCEDURE P_SUBRT ;
      BEGIN /* P_SUBRT */
      OBSIMB;   IF NOT PLI THEN FINLINE(59); EMPIECE;
      REPEAT   PROPOSICION
      UNTIL  ((SIMB='RETURN')|(SIMB='END')|(EOF(INPUT))) ;
      IF SIMB='RETURN' THEN
        BEGIN
        OBSIMB;
        IF NOT PLI THEN FINLINE(59) ELSE GENCODIG('RTS',' ','N',0,0);
        EMPIECE
        END
      ELSE BEGIN IF SIMB='END' THEN FINLINE(0); ERROR(48) END;
      END /* P_SUBRT */ ;

/*--- P_DATA -------------------------------------------------------------*/

    PROCEDURE P_DATA ;

      BEGIN /* P_DATA */
      REPEAT
        WHILE (CTR=' ') DO
          BEGIN
          OBCTR;  IF PLI THEN FINLINE(36)
          END;
```

```
      IF NOT PLI THEN CONSTANTE ;
      IF TPCONS='S' THEN PONERDT('S') ELSE
      IF TPCONS='H' THEN PONERDT('H') ELSE
      IF TPCONS='N' THEN PONERDT('N') ;
      OBSIMB ;
      IF (NOT PLI)&(SIMB<>',') THEN          FINLINE(37)
    UNTIL (PLI=TRUE)
    END /* P_DATA */ ;

/*--- P_VAR ------------------------------------------------------------*/

PROCEDURE P_VAR ;

  VAR         I:INTEGER  ;
  BEGIN /* P_VAR */
  REPEAT
    OBSIMB ;
    IF PLI THEN                FINLINE(38)    ELSE
    IF (CLASE<>'VRBLE')&(CLASE<>'STRIN') THEN FINLINE(39)
      BEGIN /* B1 */
      I := POSICION ;
      IF I<>0  THEN            FINLINE(82)    ELSE
        BEGIN
        IF (CLASE='STRIN') THEN PONERID('S')  ELSE PONERID('V') ;
        OBSIMB ;
        IF (NOT PLI) & (SIMB<>',') THEN    FINLINE(40)
        END
      END /* B1 */
  UNTIL (PLI=TRUE)
  END /* P_VAR */ ;

/*--- P_DIM ------------------------------------------------------------*/

  PROCEDURE P_DIM ;
  VAR   I,S1,S2 : INTEGER ;

  BEGIN /* P_DIM */
  REPEAT
    S1:=1;  S2:=1;             OBSIMB ;
    IF PLI THEN                FINLINE(38)  ELSE
    IF (CLASE<>'VRBLE') THEN FINLINE(39)  ELSE
      BEGIN /* B1 */
      I := POSICION ;
      IF I<>0  THEN            FINLINE(82)  ELSE
      BEGIN /* B1-1 */
      PONERID('A');            OBSIMB ;
      IF PLI       THEN        FINLINE(41)  ELSE
      IF SIMB<>'(' THEN        FINLINE(42)  ELSE
        BEGIN /* B2 */
        OBSIMB ;
        IF PLI           THEN  FINLINE(41)  ELSE
        IF CLASE<>'ENTER' THEN FINLINE(43)  ELSE
          BEGIN /* B3 */
```

```
        S1:=NUMERO;     TIDENT(.CID.).SUB1:=NUMERO;       OBSIMB ;
        IF PLI THEN       FINLINE(44)  ELSE
        IF SIMB=')' THEN                                        335
          BEGIN
          OBSIMB ;      IF NOT PLI THEN
          IF SIMB<>',' THEN          FINLINE(45)
          END    ELSE
        IF SIMB<>',' THEN            FINLINE(46)  ELSE
          BEGIN /* B4 */
          OBSIMB ;
          IF PLI              THEN   FINLINE(41)  ELSE
          IF CLASE<>'ENTER'   THEN   FINLINE(43)  ELSE
            BEGIN /* B5 */
            WITH TIDENT(.CID.) DO BEGIN NSUB:=2; SUB2:=NUMERO END ;
            S2:=NUMERO;       OBSIMB ;
            IF PLI            THEN   FINLINE(44)  ELSE
            IF SIMB<>')'      THEN   FINLINE(46)  ELSE
              BEGIN
              OBSIMB ;        IF NOT PLI THEN
              IF SIMB<>',' THEN FINLINE(45)
              END
            END /* B5 */
          END /* B4 */
        END /* B3 */
      END /* B2 */
    END /* B1-1 */
    END /* B1 */ ;
  CMI:= CMI+(S1*S2)
UNTIL (PLI=TRUE)
END /* P_DIM */ ;

/*--- BLOQUE ----------------------------------------------------------*/

  BEGIN /* BLOQUE */
  WHILE SIMB='DATA      ' DO BEGIN P_DATA; EMPIECE END ;  CDTAJX:=0 ;
  WHILE SIMB='PROGRAM   ' DO LINEAMAL                     ;
  WHILE SIMB='VAR       ' DO BEGIN P_VAR ; EMPIECE END ;
  WHILE ((SIMB='PROGRAM')|(SIMB='DATA'))  DO LINEAMAL ;
  WHILE SIMB='DIM       ' DO BEGIN P_DIM ; EMPIECE END ;
  WHILE ((SIMB='PROGRAM')|(SIMB='DATA')|(SIMB='VAR'))  DO LINEAMAL ;
  WHILE SIMB='DEF       ' DO P_DEFFUN;
  WHILE ((SIMB='PROGRAM')|(SIMB='DATA')|(SIMB='VAR')|
         (SIMB='DIM     ')|(SIMB='FNEND'))  DO  LINEAMAL ;
  WHILE SIMB='SUB       '  DO P_SUBRT;
  WHILE ((SIMB='PROGRAM')|(SIMB='DATA')|(SIMB='VAR')|
         (SIMB='DIM     ')|(SIMB='FNEND')|(SIMB='DEF')|
         (SIMB='RETURN  ')) DO   LINEAMAL ;
  IF SIMB='END' THEN ERROR(35) ;
  IF ((SIMB<>'END')&(EOF(INPUT)<>TRUE)) THEN
    REPEAT PROPOSICION
    UNTIL ((SIMB='END')|(EOF(INPUT)))
  END /* BLOQUE */ ;

/*--- ANALISIS --------------------------------------------------------*/
```

```
BEGIN /* ANALISIS */
GENCODIG ('INS',' ','N',0,0);
EMPIECE ;
IF SIMB='PROGRAM' THEN BEGIN P_PROGRAM; EMPIECE END  ELSE FINLINE(3
BLOQUE ;
IF EOF(INPUT) THEN  ERROR(32)  ELSE
IF SIMB<>'END' THEN BEGIN WRITELN(SALIDA,LINEA); ERROR(32) END   EL
   BEGIN
   OBSIMB;
   IF NOT PLI THEN FINLINE(59) ELSE GENCODIG('END',' ','N',0,0)
   END ;
LLENARGT ;
TCODIG(.1.).COPNUM:= CMI
END /* ANALISIS */ ; .
```

```
==========================================================================
==========================================================================
====
====      ESCUELA POLITECNICA NACIONAL
====      FACULTAD DE INGENIERIA ELECTRICA
====      TESIS DE GRADO
====
====      ALUMNO :   TORO LORGIO
====
====
==========================================================================
==========================================================================
```

```
/*  */
GLOBAL TXTLIB PASCALVS
LOAD COMPILADOR
START
```

BIBLIOGRAFIA

1) AHO,Alfred and ULLMAN,Jeffrey; Principles of Compiler Design, Addison-Wesley, Reading Mas. 1977.

2) GOTTFRIED, Byron; Programación Basic, Mc Graw-Hill, México.

3) GRIES,David; Construcción de Compiladores, Paraninfo, Madrid 1975.

4) GROGONO, Peter; Programación en Pascal, Fondo Educativo Interamericano, México 1974.

5) IBM Corporation, Pascal/VS Language Reference Manual, San José CA, segunda edición, 1981.

6) IBM Corporation, Pascal/VS Programer's Guide, San José CA, segunda edición, 1981.

7) NICHOLS, John; The Structure and Design of Programming Languages, Addison-Wesley, Reading Mass., 1977.

8) PRATT, Terrence; Programming Languages, Design and Implementation, Prentice-Hall, Englewood Cliffs N.J., 1975.

9) SANCHES D., Gonzalo y VALVERDE A., Juan; Compiladores e Intérpretes, Díaz de Santos, Madrid, primera edición, 1984.

10) SANCHIZ, Llorca y GALAN, Pascual; Compiladores: Teoría y Construcción, Paraninfo, Madrid 1986.

11) SCHICK, William and MERZ, Charles; Fortran para Ingeniería, Mc Graw-Hill, México 1974.