

ESCUELA POLITECNICA NACIONAL

ESCUELA DE INGENIERÍA

**DESARROLLO DE UN ENSAMBLADOR DIDÁCTICO PARA LOS
MICROCONTROLADORES DE LA FAMILIA 8051/52**

**PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE
INGENIERO EN ELECTRÓNICA
MENCIÓN TELECOMUNICACIONES**

HUERTAS TIXI MARCELO AUGUSTO

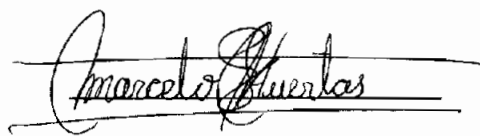
DIRECTOR: Ing. JAIME VELARDE

Quito, 27 de Abril del 2004

DECLARACIÓN

Yo Marcelo Huertas, declaro que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

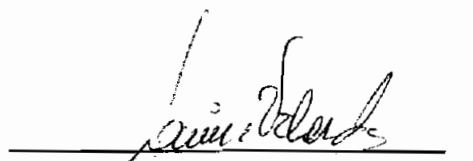
La Escuela Politécnica Nacional, puede hacer uso de los derechos correspondientes a este trabajo, según lo establecido por la Ley de Propiedad Intelectual, por su reglamento y por la normatividad institucional vigente.

A handwritten signature in black ink, reading "Marcelo Huertas", is written over a set of three horizontal lines. The signature is cursive and stylized.

Marcelo Huertas

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Huertas Tixi Marcelo Augusto bajo mi supervisión.



Ing. Jaime Velarde
DIRECTOR DEL PROYECTO

AGRADECIMIENTO

Agradezco a todas las ingenieras e ingenieros que me impartieron el conocimiento a lo largo de toda mi carrera estudiantil, a las personas que me ayudaron con la realización de este trabajo y en forma muy especial al Ingeniero Jaime Velarde por haber dirigido y brindado su apoyo en el presente proyecto.

ÍNDICE

RESUMEN Y PRESENTACIÓN

CAPÍTULO 1

1. INTRODUCCIÓN A LOS ENSAMBLADORES.	1
1.1 FUNCIONES BÁSICAS DEL ENSAMBLADOR.	1
1.2 MANEJO DE MACRO INSTRUCCIONES.	6
1.2.1 Definición de una macro instrucción.	7
1.2.2 Llamada a una macro instrucción.	8
1.2.3 Expansión de una macro instrucción.	8
1.3 TABLAS Y LÓGICA DE LOS ENSAMBLADORES.	8
1.4 TIPOS DE ENSAMBLADORES.	10
1.4.1 Ensambladores de un paso.	10
1.4.2 Ensambladores de dos pasos.	11
1.4.3 Ensambladores de varios pasos.	14

CAPÍTULO 2

2. DISEÑO DEL PROGRAMA ENSAMBLADOR.	19
2.1 LENGUAJE DE PROGRAMACIÓN.	19
2.2 CONDICIONES QUE DEBE CUMPLIR EL PROGRAMA.	19
2.2.1 Estructura del módulo fuente ".asm".	20
2.2.2 Área de comandos .	21
2.2.3 Área de etiquetas.	21

2.2.4	Área de instrucciones.	21
2.2.5	Área de definición de tablas.	22
2.2.6	Estructura del archivo de listado (.lst).	22
2.2.7	Estructura del módulo objeto ".obj".	23
2.2.8	Relocalización de programas objeto.	24
2.2.9	Estructura del archivo ejecutable ".HEX".	26
2.2.10	Condiciones y características de las pseudo instrucciones.	26
2.2.10.1	Pseudo instrucción "Link".	26
2.2.10.2	Pseudo instrucción "Include".	27
2.2.10.3	Pseudo instrucción "Public".	29
2.2.10.4	Pseudo instrucción "Extern".	29
2.2.10.5	Pseudo instrucción "Equ".	29
2.2.10.6	Pseudo instrucción "Ds".	30
2.2.10.7	Pseudo instrucción "Db".	31
2.2.10.8	Pseudo instrucción "Dw".	32
2.2.10.9	Pseudo instrucciones "Proc", "Endproc" y etiquetas locales.	32
2.2.10.10	Instrucción "Org".	34
2.2.10.11	Pseudo instrucción "End".	34
2.2.10.12	Comodín "\$".	35
 2.3 ALGORITMOS Y DIAGRAMAS DE FLUJO DEL PROGRAMA PRINCIPAL.		 35
2.3.1	Analizador de macro instrucciones.	36
2.3.2	Paso 1.	39
2.3.3	Paso 2.	41
2.3.4	Programa enlazador.	42
 2.4 DIAGRAMAS DE FLUJO DE SUBROUTINAS.		 45
2.4.1	Subrutina "buscar_definicion".	45
2.4.2	Subrutina "analizar_expresion".	48
2.4.3	Subrutina "leer_campo_etiqueta".	49

2.4.4	Subrutina "leer_campo_instruccion".	49
2.4.5	Subrutina "leer_campo_operando".	50
2.4.6	Subrutina "separador".	50
2.4.7	Subrutina para borrar datos de una tabla.	51
2.4.8	Subrutina "buscar_etiqueta".	52
2.4.9	Subrutina "buscar_sim_mat".	52
2.4.10	Subrutina "calcular_desplazamientos".	53
2.4.11	Subrutina "calcular_longitud_operando".	54
2.4.12	Subrutina "formato_operando".	54
2.4.13	Subrutina "compare".	55
2.4.14	Subrutina "gen_archivo_1st".	56
2.5 BASE DE DATOS Y TABLAS UTILIZADAS POR EL PROGRAMA ENSAMBLADOR.		57
2.5.1	Tabla de símbolos "TABSIM".	58
2.5.2	Tabla de opcodes "TABOP".	59
2.5.3	Tabla para uso del enlazador "TABLINK".	60
2.5.4	Tabla para almacenar argumentos de macro instrucciones "TABARG".	60
2.5.5	Tabla para variables externas "TAB_VAR_EXTERN".	61
2.5.6	Tabla de símbolos matemáticos SIM_MAT.	62
2.5.7	Tabla "MACRO".	62

CAPÍTULO 3

3. PRUEBAS Y RESULTADOS. 63

3.1 PRUEBAS CON EL ENSAMBLADOR TRANSPARENTE PARA EL USUARIO.		63
3.1.1	Definición de etiquetas y resolución de expresiones.	63
3.1.2	Definición y expansión de macro instrucciones.	68

3.1.3 Enlace de programas objeto.	69
3.2 PRUEBAS CON EL ENSAMBLADOR DE PASOS.	70
3.3 COMPARACIÓN DE LOS RESULTADOS CON RESPECTO A OTROS ENSAMBLADORES.	74
3.3.1 Resultados del proceso de ensamblado con el programa AVMAC51.	75
3.3.2 Resultados del proceso de ensamblado con el programa CYS-8051.	76
3.3.3 Resultados del proceso de ensamblado con el programa desarrollado en este proyecto.	77
3.4 SEMEJANZAS Y DIFERENCIAS CON OTROS PROGRAMAS ENSAMBLADORES.	78
3.4.1 Semejanzas.	78
3.4.2 Diferencias.	78
3.4.2.1 Diferencias con el programa AVMAC51.	78
3.4.2.2 Diferencias con el programa CYS-8051.	79

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES.	80
5. BIBLIOGRAFÍA.	82
6. ANEXOS.	
A. Mensajes de error.	83
B. Etiquetas predefinidas en tabla de datos TABSIM.	85

CAPÍTULO 1

1. INTRODUCCIÓN A LOS ENSAMBLADORES

Para que un microcontrolador cumpla su función se lo debe proveer de un programa en lenguaje de máquina.

Lo que se conoce como lenguaje o código de máquina es el conjunto de códigos numéricos (comúnmente expresados en hexadecimal) para cada una de las instrucciones que el microcontrolador es capaz de reconocer y ejecutar. Aun cuando finalmente es con esta información con la que el microprocesador trabaja, es muy difícil programar a este nivel. El lenguaje ensamblador fue desarrollado para realizar la programación sin tener que recordar todos los códigos y realizar las labores de cálculo de localidades de memoria. Los códigos mnemónicos del ensamblador son una substitución de los códigos numéricos y es aquí donde los programas ensambladores entran en funcionamiento empleando las técnicas de ensamblaje que en el presente proyecto se van a analizar.

1.1 FUNCIONES BASICAS DEL ENSAMBLADOR

El programa ensamblador se encarga de convertir las sentencias de un programa escrito en lenguaje ensamblador, llamado módulo fuente (archivo .ASM), en su correspondiente programa en código de máquina, llamado módulo objeto (archivo .OBJ).

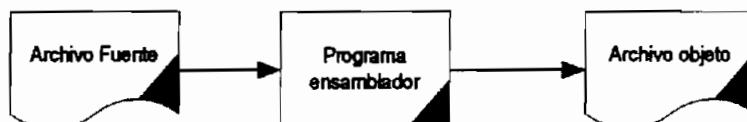


FIG.1.1 Secuencia de ensamblado.

Además se genera un archivo de listado con extensión .LST donde se encuentran tanto el módulo fuente como el código de máquina, las tablas de etiquetas y también contiene las indicaciones de error, si los hubiere.

El módulo objeto generado es casi un programa ejecutable, ya que aún falta por resolver las referencias externas con otros módulos que conforman la aplicación.

Los diferentes módulos objeto se unen a través de un programa llamado enlazador (linker) para generar un nuevo archivo cuya característica principal es la de ser un programa ejecutable o módulo objeto con direcciones definidas.

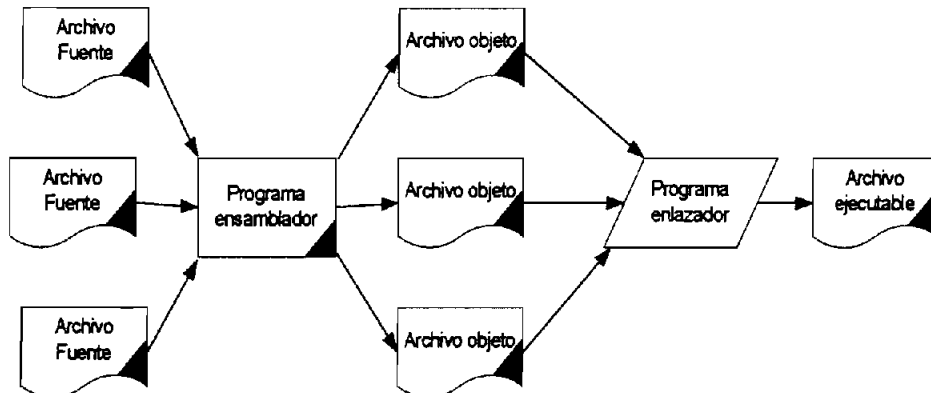


FIG. 1.2 Módulos necesarios para obtener el programa ejecutable

Para efectuar la traducción del módulo fuente a módulo objeto es necesario realizar las siguientes funciones que no necesariamente se realizan en el orden presentado.

1. Conversión de los códigos nemónicos a sus equivalentes en lenguaje de máquina.
2. Conversión de los operandos simbólicos a sus valores de direcciones.
3. Construcción de las instrucciones de máquina según el formato adecuado.
4. Conversión de las constantes de datos especificadas en el módulo fuente a sus respectivos valores hexadecimales.
5. Escritura del módulo objeto y del listado de ensamblado.

Todas estas funciones, a excepción de la número 2, se realizan procesando el módulo fuente línea a línea. Sin embargo la traducción de las direcciones siempre representa un proceso más elaborado.

Para comprender cuales son estas dificultades se analizará el caso de algunas sentencias:

1. ADD A,# 23

MOV 40H,A

Para este caso, se debe buscar en la tabla de códigos de operación el valor correspondiente al mnemónico, transformar el número 23 de decimal a binario, verificar que el valor del mismo no supere el rango permitido y ubicarlo en el campo de operandos.

2. ADD A,#DAT1

MOV RESULT,A

En este caso se han empleado nombres simbólicos o etiquetas. El problema que puede surgir es que en el momento de ensamblar estas sentencias, no se conozca el valor de estas etiquetas, o bien porque estén definidos más adelante en el archivo (referencias adelantadas), o bien porque hagan referencia a direcciones o datos definidos en otros módulos (referencias externas).

3. ADD A,#(dir5+dir3)*dir5-24

MOV result-1,A

En este caso, además de los problemas anteriores, hay que evaluar las expresiones para obtener los valores de los operandos o de las direcciones.

Se plantean por lo tanto 4 problemas para poder ensamblar un módulo fuente:

1. Asignar valores a los nombres simbólicos o etiquetas.

2. Resolver las referencias adelantadas.

3. Resolver las referencias externas.
4. Evaluar las expresiones.

Además de traducir el módulo fuente, el ensamblador debe procesar las proposiciones llamadas instrucciones para el ensamblador o también conocidas como pseudo instrucciones. Estas proposiciones no se traducen en instrucciones de máquina pero pueden influir en el módulo objeto. Algunos ejemplos de estas instrucciones son las proposiciones ORG, EQU, DB, DW entre otras, las cuales le indican al ensamblador que genere constantes como parte del módulo objeto.

Ejemplo:

INICIO EQU 20H (Asigna a la etiqueta INICIO el valor 20 H)

Si tomamos como ejemplo un ensamblador de dos pasos, las funciones que este realizará, en forma general, serán las siguientes:

Paso 1 (definir los símbolos).

1. Asignar direcciones a todas las proposiciones del programa.
2. Guardar todas las direcciones asignadas a las etiquetas para usarlas en el paso dos.
3. Realizar el procesamiento de las pseudo instrucciones.

Paso 2 (ensamblar las instrucciones y generar el módulo objeto).

1. Ensamblar las instrucciones, lo cual significa la traducción de los códigos de operación y su ubicación en las diferentes direcciones.
2. Generar los valores de datos definidos por las pseudo instrucciones.
3. Realizar los procesamientos faltantes de las pseudo instrucciones.

4. Escribir el módulo objeto y el listado del ensamblado.

Al final del proceso de ensamblado, el módulo objeto generado estará conformado por tres tipos de registros: de encabezamiento, de texto y de fin.

El registro de encabezamiento contiene el nombre del programa su dirección inicial y la longitud del programa; el registro de texto contiene las instrucciones traducidas al código de máquina y los datos del programa, junto con una indicación de la dirección que ocupará; el registro de fin marca el final del módulo objeto y especifica la dirección de la primera instrucción ejecutable del programa.

A continuación se muestra un ejemplo de módulo objeto y en la figura 1.3 la manera como este se encuentra estructurado.

HCOPY 000000001077

T00000001D7202D69292D4B1010360320262900003320074B10105D3
FSFEC032010

T00001D130F20160100030F200D4B10105D3E2003454F46

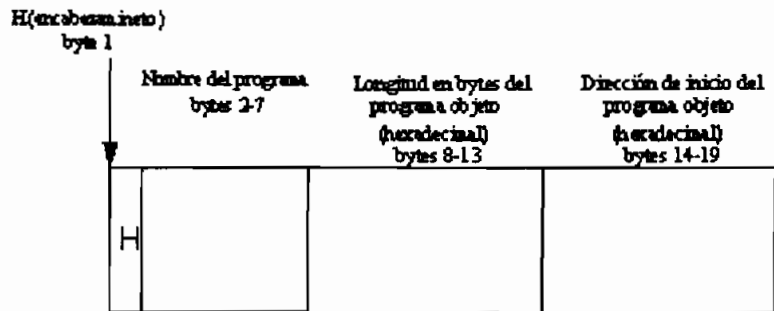
T0010361DB410B400844075101000E32019332FFADB2013A004332008
57C003B850

T001070073B2FEF4F000005

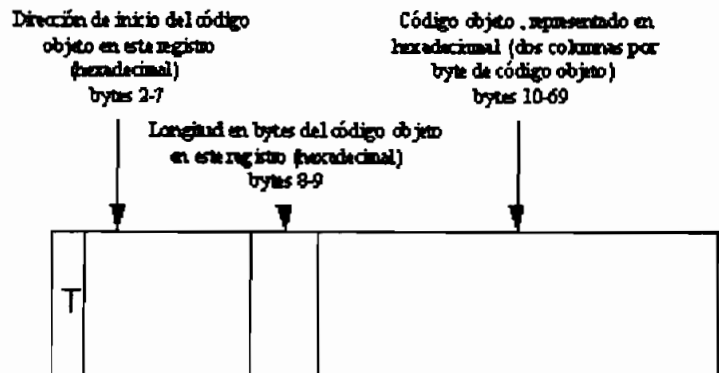
E000000

ESTRUCTURA DEL PROGRAMA OBJETO

REGISTRO DE ENCABEZAMIENTO



REGISTRO DE TEXTO



REGISTRO DE FIN

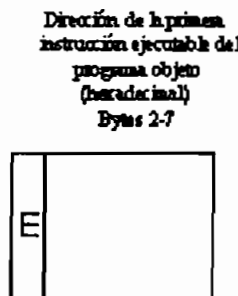


FIG. 1.3 Estructura del módulo objeto.

Los módulos objeto y los archivos ejecutables con formato INTEL para los microcontroladores de la familia 8051/52 no contienen el registro de encabezamiento.

1.2 MANEJO DE MACRO INSTRUCCIONES

Una macro instrucción no es más que una conveniencia notacional para el programador, que representa un grupo de instrucciones en lenguaje ensamblador agrupadas bajo un nombre.

Cuando el programador escribe el nombre de una macro instrucción, es como si hubiese escrito todas las instrucciones que la componen. Para poder llamar a las macro instrucciones por su nombre es preciso haberlas definido previamente.

Cuando un macroensamblador realiza la traducción de lenguaje ensamblador a lenguaje de máquina y encuentra el nombre de una macro instrucción el programa la sustituye por todas las instrucciones que la componen. A este proceso se le llama expansión de la macro.

De esta manera asociados a las macro instrucciones aparecen tres conceptos:

1. DEFINICIÓN de la macro instrucción.
2. LLAMADA a la macro instrucción.
3. EXPANSIÓN de la macro instrucción.

1.2.1 Definición de una macro instrucción

La definición de una macro instrucción consta de tres partes: encabezamiento, cuerpo y final.

Ejemplo:

```
TEXT MACRO      MENSAJE          ; Encabezamiento
    MOV         DPTR,#MENSAJE    ; Cuerpo
    LCALL      MOSTRAR_CADENA    ; Cuerpo
    ENDM                          ; Final de la macro
```

FIG. 1.4 Definición de una macro instrucción

Se puede observar que el nombre de la macro es TEXT y además lleva un parámetro denominado "MENSAJE".

1.2.2 Llamada a una macro instrucción

Para llamar a una macro instrucción, solo se necesita escribir su nombre en el campo instrucción y en el campo operando se debe especificar los nombres de los parámetros en el caso de que existan. Tal como se muestra en la figura 1.5.

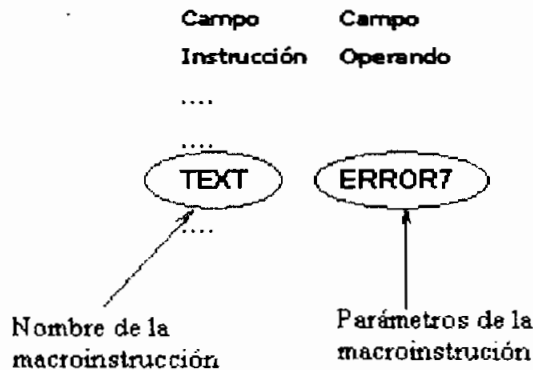


FIG.1.5 Llamada a una macro instrucción.

1.2.3 Expansión de una macro instrucción

El macroensamblador sustituirá cada llamada a la macro instrucción por su cuerpo. Para el ejemplo de la figura 1.5 el resultado será:

Campo Instrucción	Campo Operando
....	
....	
MOV	DPTR,#ERROR7
LCALL	MOSTRAR_CADENA
...	

FIG. 1.6 Expansión de una macro instrucción

1.3 TABLAS Y LÓGICA DE LOS ENSAMBLADORES

Para realizar la traducción de lenguaje ensamblador a código de máquina los ensambladores generalmente manejan dos tablas principales: la tabla de código de operación y la tabla de símbolos.

La tabla de códigos de operación se utiliza para examinar los códigos mnemónicos y traducirlos a sus equivalentes en lenguaje de máquina, y la tabla de símbolos, para almacenar los valores asignados a las etiquetas.

También es necesario un contador de localidades, CONTLOC, que es de gran ayuda en la asignación de direcciones. El valor inicial de este contador se lo toma del valor de la proposición ORG, en caso de que esta pseudo instrucción no exista CONTLOC toma el valor 00. Después de procesar cada proposición del módulo fuente se suma a CONTLOC la longitud de la instrucción ensamblada o la longitud del área generada si se trata de una pseudo instrucción. De esta forma cada vez que se encuentra una etiqueta en el módulo fuente, el valor de CONTLOC proporciona la dirección asociada a esa etiqueta.

Por ejemplo, sea la siguiente secuencia:

Dirección	Etiqueta	Instrucción	Operando
1031	SIMB	MOVE	@R1,#10H
1033	INICIO	MOVE	A,#20H

FIG.1.7 Secuencia de instrucciones.

Luego de analizar la primera línea los resultados serán los siguientes: la etiqueta SIMB tomará el valor que tenga en ese instante CONTLOC, que para este caso es 1031, debido a que la instrucción: MOVE @R1,#10H, tiene una longitud de 2 Bytes el siguiente valor que tomará CONTLOC será 1031+2, es decir que la siguiente instrucción se almacenará en la memoria de programa a partir de la dirección 1033.

La tabla de operación contiene el mnemónico del código de operación y su equivalente en lenguaje de máquina. Debido a que las instrucciones del microcontrolador 8051/52 no tienen una longitud constante esta tabla también contiene el tamaño de cada instrucción. Para calcular el valor de CONTLOC esta tabla es revisada por varias ocasiones en la primera pasada.

La tabla de símbolos ("TABSIM") que contiene el nombre y el valor de todas las etiquetas del módulo fuente, también contiene banderas para indicar la presencia de errores. Estas banderas se las utiliza por ejemplo para indicar si una etiqueta ha sido definida en dos ocasiones y generar un código de error.

En el paso 1 el ensamblador utiliza esta tabla para guardar el nombre de las etiquetas y el valor de la dirección que poseen. En el paso 2 el ensamblador accede a esta tabla para buscar el valor de los símbolos empleados como operandos e insertarlos en las instrucciones que están siendo ensambladas. Es posible que el módulo fuente sea leído tanto en el paso 1 como en el paso 2.

Existen ciertos datos como el valor del contador de localidades y las banderas de error que deben comunicarse entre los dos pasos a través de la tabla de símbolos. Por esta razón el paso 1 genera un archivo temporal que se usa como entrada al paso dos y contiene las proposiciones fuente junto con las direcciones asignadas y los indicadores de error.

1.4 TIPOS DE ENSAMBLADORES.

1.4.1 Ensambladores De Un Paso.

Este tipo de ensamblador se lo emplea en aplicaciones que requieren alta velocidad en la traducción y ejecución de programas. El principal problema de estos ensambladores es que no pueden resolver las referencias adelantadas. Tampoco pueden insertar los valores de los operandos simbólicos de las instrucciones y etiquetas que aún no han sido definidas en el módulo fuente.

Para resolver el problema de las referencias adelantadas y operandos simbólicos se impone una restricción al usuario y es que debe definirlos al inicio del programa o antes de que haga referencia a ellas.

Para el caso de operandos simbólicos estos se van guardando en la tabla de símbolos conforme se los va encontrando en el módulo fuente, en esta tabla se activa una bandera para indicar que no están definidos y además se guarda la

dirección de programa en donde se encuentran. Cuando se encuentra su valor se lo reemplaza en las direcciones indicadas en la tabla y se desactivan las banderas de operandos no definidos.

1.4.2 Ensambladores de dos pasos

Debido a que un programa en lenguaje ensamblador consta de una serie de sentencias de una línea, parece natural tener un ensamblador que lea una sentencia, la traduzca a lenguaje de máquina y escriba el código de máquina generado en un archivo. El proceso se repetiría hasta que todo el programa se haya traducido. Por desgracia, este método no funciona.

Considérese la situación en la que la primera sentencia sea un salto hacia la etiqueta L. El ensamblador no puede ensamblar esta instrucción hasta que conozca la dirección de la sentencia L. La sentencia L puede estar cerca del fin del programa, lo que impide que el ensamblador encuentre la dirección sin leer primero casi todo el programa. Esta dificultad se llama referencia adelantada, porque el símbolo L se ha usado antes de que se haya definido; es decir, se ha hecho referencia a un símbolo cuya definición aparecerá más tarde.

Este problema se resuelve leyendo el módulo fuente dos veces. Cada lectura del módulo fuente se llama una pasada, y todo traductor que lea el módulo de entrada dos veces se llama un traductor de dos pasadas. En la pasada 1, el ensamblador de dos pasadas colecciona todas las definiciones de símbolos, incluyendo las etiquetas de las sentencias, y las almacena en una tabla. En el momento en que empieza la segunda pasada, ya se conocen los valores de todos los símbolos; ya no existe el problema de las referencias adelantadas y puede leerse cada sentencia, ensamblarla y obtener el código salida. Este método es conceptualmente sencillo, aunque requiere una pasada adicional sobre la entrada.

Tomemos como ejemplo la siguiente secuencia:

```

LCD    EQU    0A000H
N      EQU    0EH
      SJMP   INI1
      ORG   20H
INI1   MOV    A, #01H
      MOV   DPTR, #LCD
NUM    MOVX   @DPTR, A
      MOV   A, #02H
      MOV   DPTR, #LCD+1
SUM    MOVX   @DPTR, A
      MOV   A, #(N*2+1)
      END
    
```

FIG. 1.8 Secuencia de instrucciones.

Después de la primera pasada la tabla de símbolos contendrá los siguientes valores:

AREA PARA ETIQUETA O SIMBOLOS	INDICA CUANTAS VARIABLES FALTAN POR DEFINIR PARA CONOCER EL VALOR DEL SIMBOLO LCD	ACQUI SE ALMACENA EL VALOR O LA EXPRESION QUE DEFINE A LCD	INDICA SI OTRAS VARIABLES DEPENDEN DEL SIMBOLO LCD
LCD	0	0A000H	0
N	0	0EH	0
INI1	0	20H	0
NUM	0	25H	0
LCD+1	0	0A001H	0
SUM	0	30H	0
N*2+1	0	10H	0

FIG. 1.9 Tabla de símbolos después de la primera pasada.

Finalmente el programa en fase de ensamblado tiene la siguiente forma luego del paso 1.

Programa en fase de ensamblado		(Paso 1)
Dirección	Código	Sentencia Fuente
		LCD EQU 0A000H
		N EQU 0EH
0000	8000	SJMP INI1
		ORG 20H
0020	7401	INI1 MOV A,#01H
0022	900000	MOV DPTR,#LCD
0025	F0	NUM MOVX @DPTR,A
0026	7402	MOV A,#02H
0028	900000	MOV DPTR,#LCD+1
002B	F0	SUM MOVX @DPTR,A
002C	7400	MOV A,#(N*2+1)
		END

FIG. 1.10 Programa luego del paso 1

Se debe observar que no se han reemplazado los valores de las expresiones o símbolos en el código de máquina ya que este procedimiento es responsabilidad del paso 2.

En la segunda fase se toma los valores de la tabla de símbolos y se insertan en el código objeto para completar su ensamble. De esta forma al finalizar la segunda fase el programa ejemplo de la figura 1.8 quedará como sigue:

Dirección	Código	Sentencia Fuente
		LCD EQU 0A000H
		N EQU 0EH
0000	8020	SJMP INI1
		ORG 20H
0020	7401	INI1 MOV A,#01H
0022	90A000	MOV DPTR,#LCD
0025	F0	NUM MOVX @DPTR,A
0026	7402	MOV A,#02H
0028	90A001	MOV DPTR,#LCD+1
002B	F0	SUM MOVX @DPTR,A
002C	741D	MOV A,#(N*2+1)
		END

Fig 1.11 Programa luego del paso 2.

1.4.3 Ensambladores de varios pasos.

En los ensambladores anteriores se requiere que todo símbolo empleado en el campo operando, es decir en el lado derecho, debe estar definido con anterioridad en el módulo fuente. La razón de ello es por el proceso de definición de símbolos en un ensamblador de dos pasos.

Por ejemplo:

```
ALFA EQU BETA
BETA EQU DELTA
DELTA DS 1
```

FIG. 1.12 Etiquetas con dependencia.

No puede asignarse un valor al símbolo BETA cuando se encuentra en el primer paso porque DELTA aún no ha sido definido y, en consecuencia, ALFA no se puede evaluar en el segundo paso. Esto quiere decir que todo ensamblador que solo haga dos pasos secuenciales por el módulo fuente no podrá resolver tal secuencia de definiciones.

La solución general es un ensamblador que pueda realizar tantos pasos como sean necesarios para procesar las definiciones de los símbolos. No es necesario que este tipo de ensamblador haga más de dos pasadas por todo el módulo, en lugar de eso, las porciones del módulo que incluyen referencias hacia delante en la definición de símbolos se guardan durante el paso 1. Los pasos adicionales se hacen sobre estas definiciones almacenadas conforme avanza el ensamblado.

Este proceso va seguido de un paso 2 normal. El procedimiento implica almacenar en la tabla de símbolos las definiciones de las etiquetas que comprenden las referencias hacia delante. Esta tabla también indica las etiquetas que dependen de los valores de otros, para facilitar su evaluación.

A continuación se muestra una secuencia de proposiciones de definición de símbolos que implican referencias hacia delante.

1. MLOG EQU MAXLEN/2
2. MAXLEN EQU BUFEND-BUFFER
3. PREV EQU BUFFER-1
4. BUFFER DS 20
5. BUFEND EQU 30

FIG. 1.13 Programa con referencias adelantadas

En la figura 1.14 se muestra la tabla de símbolos producida por el paso 1 para la línea 1.

El símbolo MAXLEN aún no ha sido definido por lo que no se podrá calcular ningún valor para MLOG. La expresión que define a MLOG se almacena en la tabla de símbolos en lugar de su valor. La entrada &1 indica que aún no ha sido definido el primer símbolo de la expresión (MAXLEN / 2). El símbolo MAXLEN también es introducido en la tabla con la bandera "*" que significa indefinido. Luego se especifican los símbolos cuyos valores dependen de MAXLEN (en este caso MLOG).

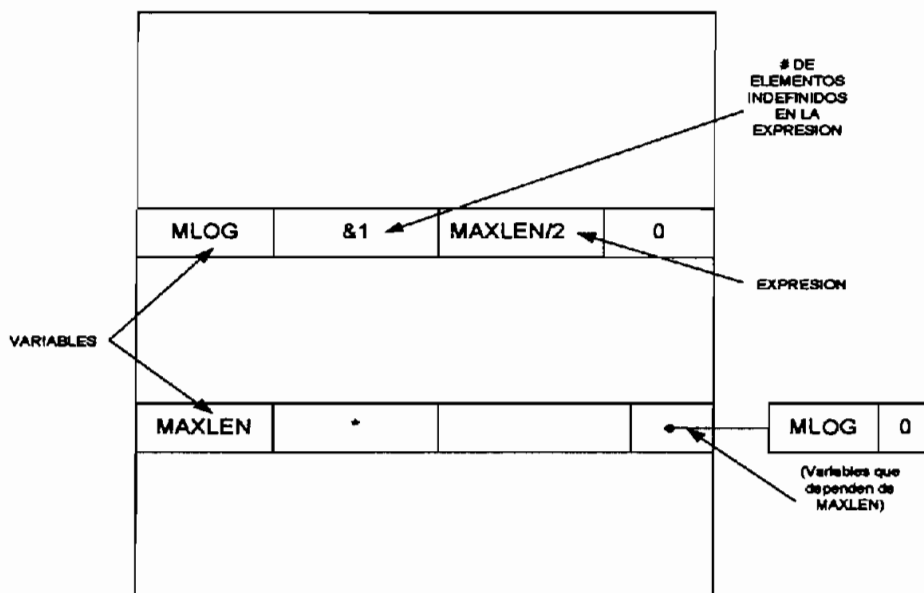


FIG. 1.14 Tabla de símbolos luego de analizar la primera línea

El mismo procedimiento se sigue para MAXLEN, en la figura 1.15 se muestra el estado de la tabla de símbolos después de que el ensamblador analiza la línea dos.

En este caso "&2" indica que existen dos variables indefinidas en la expresión que define el valor de MAXLEN. BUFEND y BUFFER ingresan a la tabla con la bandera " * " debido a que no están definidos. A continuación se pone las variables que dependen de BUFEND y BUFFER (en este caso MAXLEN).

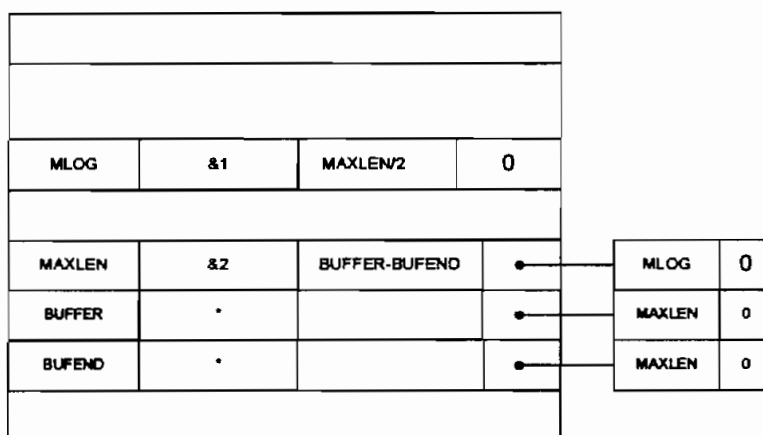


FIG. 1.15 Tabla de símbolos luego de analizar la línea 2

Cuando el ensamblador analice la línea 3 generará la siguiente tabla:

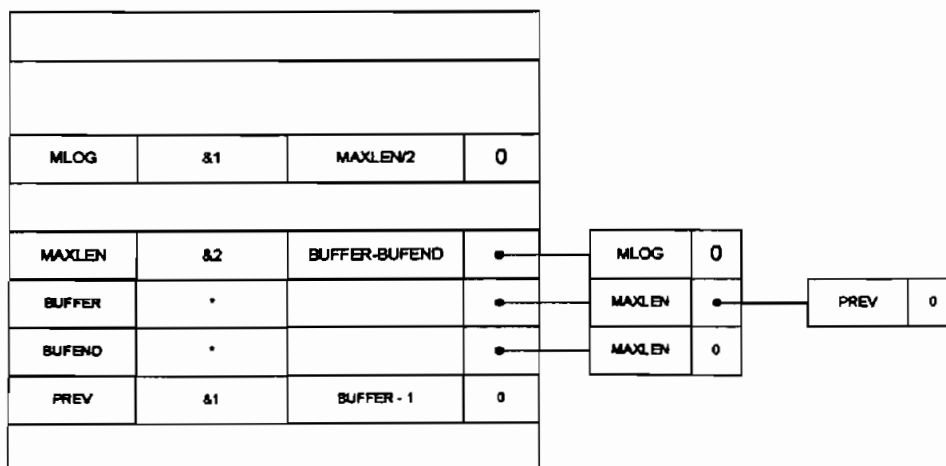


Figura 1.16 Tabla de símbolos luego de analizar la línea 3

Al llegar a la línea 4, se observa que BUFFER queda definido con el valor que en ese instante tenga el Contador de Localidades. Si se asume que CONLOC=1031, esta dirección será el valor que definirá a BUFFER. El ensamblador entonces examina la lista de símbolos que dependen de BUFFER. En este punto el símbolo PREV queda definido, pero MAXLEN todavía no ya que no se ha definido el símbolo BUFEND, por esta razón el ensamblador cambia la bandera "&2" por "&1" como se muestra en la figura 1.17.

MLOG	&1	MAXLEN2	0
MAXLEN	&1	BUFFER-BUFEND	●
BUFFER	1031		
BUFEND	*		●
PREV	1030		0

MLOG 0

MAXLEN 0

FIG. 1.17 Tabla de símbolos luego de analizar la línea 4.

Después de analizar la línea 5, el valor de BUFEND y MAXLEN quedan definidos. Esto ocasiona que MLOG ya se pueda calcular y la tabla de símbolos quedaría de la siguiente forma:

MLOG	0	809h	0
MAXLEN	0	1013h	0
BUFFER	0	1031	0
BUFEND	0	1Eh	0
PREV	0	1030	0

Figura 1.18 Tabla de símbolos luego de analizar la línea 5

Si todavía quedan símbolos sin definir luego de que el ensamblador ejecute los dos pasos sobre el módulo fuente, se considerará a estas variables como errores.

CAPÍTULO 2

2. DISEÑO DEL PROGRAMA ENSAMBLADOR.

En la actualidad la mayoría de las funciones de programación, tienden a desarrollarse en ambiente Windows, el cual es un entorno de desarrollo visual.

El programa ensamblador utilizado hasta el día de hoy en los laboratorios de sistemas microprocesados de la Escuela Politécnica Nacional, para programar los microcontroladores INTEL 8051\52, es el AVMAC51, el mismo que forma parte del paquete SIDES2000 y fue desarrollado en entorno DOS. Por tal motivo, a través del trabajo realizado en este proyecto de titulación, se pretende reemplazar este programa por otro que funcione bajo ambiente WINDOWS.

2.1 LENGUAJE DE PROGRAMACIÓN.

Para el desarrollo del programa ensamblador se ha elegido la herramienta de programación Visual Basic. El motivo por el cual se eligió usar este lenguaje de desarrollo es debido a la compatibilidad que debe tener con el programa SIDES2000 ya que el programa ensamblador será insertado dentro de este paquete.

SIDES2000 es un paquete de software desarrollado con Visual Basic en el área de Sistemas Digitales de la Escuela Politecnica Nacional. Este programa permite editar, ensamblar, enlazar y simular programas destinados a ejecutarse en los microcontroladores de la familia INTEL 8051\52.

2.2 CONDICIONES QUE DEBE CUMPLIR EL PROGRAMA:

La mayor parte de las condiciones con las que debe cumplir el programa ensamblador vienen dadas por el formato del archivo del módulo fuente con extensión ".ASM", el formato de los archivos de listado y el formato del módulo objeto que el programa deberá generar.

2.2.1 Estructura del módulo fuente “.asm”.

Para una mejor comprensión del tema, a este archivo se lo presenta dividido en áreas, cada una de las cuales presenta instrucciones, seudo instrucciones y otra información útil para el ensamblador.

Las áreas que conforman el módulo fuente son las siguientes:

1. Área de comandos.
2. Área de definición de etiquetas.
3. Área de instrucciones y
4. Área para definición de tablas

A continuación se verá con más detalle el formato y la información que cada una de estas áreas contiene.

En esta sección solo se mencionará las seudo instrucciones que se utilizan con mayor frecuencia en cada área. El análisis de lo que realiza cada seudo instrucción se verá con detalle más adelante.

Para ello se tomará el siguiente ejemplo:

```

:Area de comandos
LINK  c:\ejemplos,prueba1.obj
LINK  c:\ejemplos,PRUEBA2.OBJ
include  c:\ejemplos\macros.txt,prueba
:Area de definicion de variables
public   supr,avpag,repag
extern  bloqnum,klm
bin     equ    10100h
supr    equ    20h
avpag   equ    repag
repag   equ    3
:Area de instrucciones
prueba  20,30
cjne   a,bloqnum-1,klm+2
mov    bloqnum+bin,a
mov    a,bloqnum-klm
mov    a,[bloqnum-(klm+supr)*repag]
org    15h
mov    a,#0ah
sjmp  $
:Area para tablas
maced101  db    "abcd"
end

```

FIG.2.1 Módulo fuente ".ASM".

2.2.2 Área de comandos.

El área de comandos no contiene ninguna instrucción pero puede contener a las pseudo instrucciones: link e include.

La pseudo instrucción Link permite enlazar el módulo objeto que se va a generar con otros archivos de extensión ".OBJ" almacenados en memoria.

La pseudo instrucción Include permite agregar macro instrucciones al programa principal. Estas macro instrucciones pueden estar almacenadas dentro de cualquier archivo de texto.

2.2.3 Área de etiquetas.

En esta área se definen las variables que se utilizarán en el archivo fuente, así como también se definen las etiquetas que serán públicas y se enuncian las etiquetas con referencia externa. Las pseudo instrucciones que se emplean en esta área son: public, extern, equ.

2.2.4 Área de instrucciones.

Esta área puede contener etiquetas, instrucciones y pseudo instrucciones.

Aquí se pueden colocar todas las instrucciones del microcontrolador que se muestran en el ANEXO C de acuerdo a las necesidades del programa.

Las pseudo instrucciones que más se emplean en esta área son las siguientes: org, equ, proc, endproc y los nombres de las macro instrucciones que después de haberlas definido en el área de comandos se las puede emplear en el resto del programa como si se tratase de una instrucción.

2.2.5 Área de definición de tablas.

Esta área puede estar localizada al final del programa. Se la emplea para almacenar mensajes o datos en la memoria de programa del microcontrolador, mediante las pseudo instrucciones: DS, DB, DW. Esta área por lo general no contiene ninguna instrucción.

Es recomendable colocar estas tablas al final del programa o luego de un lazo infinito debido a que, por estar almacenadas en la memoria de programa, el microcontrolador puede confundir estos datos y tratarlos como instrucciones.

2.2.6 Estructura del archivo de listado (.LST).

En este archivo se presenta de forma paralela al módulo fuente y al módulo objeto, para lograr esto se debe presentar la traducción línea a línea del archivo ".ASM".

El siguiente es un ejemplo del archivo de listado ".LST" generado por el ensamblador para el módulo fuente de la figura 2.1.

```

Lenguaje de máquina      #línea  Lenguaje ensamblador
1                          ;Area de comandos
2                          ; link   c:\ejemplos,prueba1.obj
3                          ; LINK   c:\ejemplos,PRUEBA2.OBJ
4                          ; include c:\ejemplos\macros.txt,prueba
5                          ;Area de definicion de variables
6                          public  supr,avpag,repag
7                          extern  bloqnum,klm
0014  -                    8          brn    equ    10100b
0020  -                    9          supr   equ    20h
0003  -                    10         avpag  equ    repag
0003  -                    11         repag  equ    3
12                          ;Area de instrucciones
0000  751E40              13       MOV    30,#40h
0003  1514                14       DEC    20
0005  0540                15       INC    40h
0007  851440              16       MOV    20,40h
000A  9500F3              17       cjne  a,bloqnum-1,klm*2
000D  F500                18       mov   bloqnum*brn,a
000F  E500                19       mov   a,bloqnum-klm
0011  E500                20       mov   a,(bloqnum-(klm*supr)*repag)
0015  740A                21       org  15h
0017  8DFC                22       mov   a,#0ah
23       jmp  $
24       ;Area para tablas
0010  61 62 63 64        25       marceloi db  "abcd"
26                          end
    
```

FIG.2.2 Archivo de listado.

2.2.7 Estructura del módulo objeto “.OBJ”.

En este archivo se encuentra el código de máquina en formato INTEL hexadecimal y se encuentra organizado como una secuencia de registros.

Existen cinco tipos de registros: el registro de datos, registro de fin, registro de variables definidas, registro de variables externas y registro de modificación.

El registro de datos, el cual contiene los códigos de máquina, está configurado de la siguiente manera:

:	Contador 2 caracteres	Dirección de carga 4 caracteres	Tipo de registro 00	Bytes de datos 2 caracteres/byte	Checksum 2 caracteres	CR	LF
---	--------------------------	------------------------------------	------------------------	-------------------------------------	--------------------------	----	----

FIG. 2.3 Registro de datos.

El registro final contiene la dirección de la primera instrucción ejecutable, su formato es el siguiente:

:	Contador 2 caracteres	Dirección de inicio 4 caracteres	Tipo de registro 01	Checksum 2 caracteres	CR	LF
---	--------------------------	-------------------------------------	------------------------	--------------------------	----	----

FIG. 2.4 Registro de fin.

El registro de definición tiene como primer carácter la letra "D" y contiene el valor y nombre de una etiqueta pública.

El registro de referencia externa empieza con la letra "E" y contiene únicamente los nombres de las variables externas que se emplean en ese programa.

El registro de modificación empieza con la letra "M" seguido de la dirección de carga, el número de caracteres que se debe modificar, un símbolo que indica la operación que se debe realizar y la etiqueta o expresión con variables externas que falta por resolver.

Un ejemplo de módulo objeto se muestra a continuación:

```

:10002100753246140540E540D221C221B50000F51A
:050010000E500E50000
:10001500740A00FE606F6C612060756E646FE70D0A
:00002500C0A0000E0007000001925041E00
:0000001FF
D.SUPR.21
D.AVPAG.0
D.RFPAG.3
E.BLOQNUM
E.KLM
M.46,2,+BLOQNUM-1
M.47,2,+KLM+2
M.49,2,+BLOQNUM
M.51,2,+BLOQNUM-KLM
M.53,2,+ (BLOQNUM-(KLM+33)*3)

```

↑
Etiqueta o expresión que falta por resolver

FIG. 2.5 Módulo objeto.

2.2.8 Relocalización de programas objeto.

Esta propiedad es muy importante sobre todo si se desea enlazar varios módulos objeto para obtener un solo programa ejecutable.

La relocalización se realiza a través de dos procedimientos:

El primer procedimiento consiste en alterar la dirección de carga de los códigos de máquina en el módulo objeto.

Por ejemplo si a un grupo de instrucciones que tienen como dirección de carga el número 000, se desea desplazarlos hacia la dirección 50, lo que se debe hacer es sumar un desplazamiento de 50 a la dirección de carga inicial de cada uno de los registros de datos del módulo objeto.

El procedimiento siguiente consiste en cambiar el contenido de las instrucciones absolutas si el módulo objeto tuviese alguna, esto se realiza usando la información contenida en los registros de modificación del módulo objeto.

En la siguiente figura se muestra un programa almacenado a partir de la dirección 0000.

```

:100000090A001740190A000F07415F00490A0027B
:1000100F00490A003F0740F904000E0900000F016
:020020080E17D
:0000001FF

```

Dirección inicial de carga

FIG. 2.6 Módulo objeto almacenado a partir de la dirección "0000".

A continuación se muestra el mismo programa pero desplazado a la dirección 50H.

```

:100500090A001740190A000F07415F00490A0022B
:10006800F00490A003F0740F904000E0900000F0C6
:020070080E12D
:0000001FF

```

Programa relocalizado a la
dirección 50h

FIG. 2.7 Módulo objeto relocalizado

Debe tomarse en cuenta que la dirección inicial de carga de cada uno de los registros de datos del módulo objeto también se encuentran modificados por el mismo desplazamiento.

2.2.9 Estructura del archivo ejecutable “.HEX”.

Este archivo es el resultado final del proceso de ensamblado, en donde la información binaria se presenta en formato INTEL hexadecimal dentro de un archivo de texto.

Al igual que el módulo objeto, se encuentra organizado como una secuencia de varios registros de datos y un registro final.

Este archivo no contiene registros de definición, registros de referencias externas ni registros de modificación.

Un ejemplo de este tipo de archivo se muestra a continuación:

```
:10002100753240140540E540D221C22185FF33F588
:0500310000E500E50000
:10001500740A80FE686F6C61206D756E646FE70D04
:000025000C0A0B03E8007B000088BD648CB2
:10003300E509F6C28701877580FFC283D283C202B6
:10004300D202D208D209D20AD217D2187401850a41
:10005300FB80F9011C90000aF0B50aFD74029000C0
:100063000BF00490000CF00490000DF0740F90005E
:1000730040E0900000F0F00490000DF0740F900049
:0800830040E0900000F080FE57
:00002101DE
```

FIG. 2.8 Archivo ejecutable.

2.2.10 Condiciones y características de las pseudo instrucciones.

2.1.10.1 Pseudo instrucción “Link”.

Definición: permite indicar la localización y definición de los archivos objeto que se van a enlazar.

Formato: Esta pseudo instrucción ocupa los campos “Instrucción” y “Operando” del módulo fuente.

En el campo instrucción va la palabra “Link” con la cual se indica al ensamblador que se desea realizar un enlace.

Al campo operando se lo divide en varios suboperandos separados por una “,” de la siguiente manera: el primer suboperando contiene la dirección del o los archivos objeto que se van a enlazar.

En el suboperando 2 y en los que le siguen a continuación se encuentran los nombres de los archivos objeto que se desea enlazar.

En el siguiente ejemplo se demuestra con mayor claridad lo que se acaba de describir:

Etiqueta	Instrucción	Operando
	Link	c:\ejemplo\,arch1.obj,arch2.obj,...

Fig2.9 Seudo instrucción Link.

2.2.10.2 Seudo instrucción “Include”.

Definición: Permite indicar la localización del archivo donde se encuentran almacenadas las macro instrucciones.

Formato: En el campo instrucción del módulo fuente va la palabra “INCLUDE” la cual le indica al ensamblador que se va a definir la localización de una o varias macro instrucciones a través de lo que exista en el campo operando.

Al igual que en la seudo instrucción anterior, se divide al campo operando en suboperandos separados por una “,”.

En el primer suboperando se especifica la dirección del archivo que contiene las macro instrucciones. Es importante indicar que el nombre del archivo debe contener la extensión del mismo ya que las macro pueden estar almacenadas dentro de archivos “.TXT”, “.ASM” o cualquier otro archivo de texto.

El suboperando 2 y los que le siguen a continuación contienen los nombres de las macro que se van a usar.

Una vez que se ha definido la localización y el nombre de las macro ya se las puede usar en cualquier parte del módulo fuente como si se tratase de una instrucción.

En el siguiente ejemplo se definirá una macro llamada "prueba" la misma que se encuentra dentro de "c:\ejemplo\macros.txt".

El archivo "c:\ejemplo\macros.txt" contiene lo siguiente:

```

May      macro    &1,&2,&3
          mov     a,&1
          dec     a
          db     "hola tania",120
          endm
lucky    macro    &1,&2,&3
          dw     100h,20h
          endm
Prueba   macro    a,b
          mov     b,#40h
          dec     a
          inc     40h
          mov     a,40h
          endm

```

FIG.2.10 Archivo "macros.txt".

A continuación se presenta la forma como se define una macro y la manera como se la invoca:

```

;Área de comandos
include  c:\ejemplos\macros.txt,prueba ;<--- Definición de macro
;Área de instrucciones
prueba  a,50 ;<---- Invocación de macro

```

FIG.2.11 Definición e invocación de una macro instrucción.

Cuando el analizador de macro instrucciones revisa el módulo fuente, expande todas las invocaciones a macros que va encontrando, quedando como resultado final lo siguiente:

```

;Area de comandos
include c:\ejemplos\macros.bit,prueba
;Area de instrucciones
MOV     50,#40h
DEC     A
INC     40h
MOV     A,40h

```

FIG.2.12 Archivo con macro instrucciones expandidas.

El resultado final se almacena en un archivo que tiene la misma dirección y el mismo nombre del módulo fuente pero con extensión “.MCR”. Este es el archivo que el paso 1 tomará como referencia para comenzar con la traducción de lenguaje ensamblador a lenguaje de máquina.

2.2.10.3 Seudo instrucción “Public”.

Definición: Permite identificar las variables externas que se van a definir dentro del módulo fuente.

Formato: En el campo instrucción del módulo fuente va la palabra “Public” y en el campo operando las variables separadas por comas.

2.2.10.4 Seudo instrucción “Extern”.

Definición: Permite identificar las variables que se emplearán en el programa pero que serán definidas en el momento de enlazar el módulo objeto.

Formato: En el campo instrucción del módulo fuente va la palabra “Extern” y en el campo operando están las variables separadas por comas.

2.2.10.5 Seudo instrucción “Equ”.

Definición: A través de la seudo instrucción “EQU” se puede asignar o cambiar el valor de las etiquetas.

Formato: En el campo instrucción debe estar la palabra “EQU”. Mientras que en el campo operando debe registrarse el valor que se le desea asignar a la variable. A continuación se muestra un ejemplo de las tres seudo instrucciones anteriores:

	public	supr,Avpag,repag
	extern	bloqnum,klm
bin	equ	10100b
supr	equ	20h,bit
avpag	equ	repag
repag	equ	3
ghi	equ	22o

FIG.2.13 Seudo instrucciones public, extern, equ.

2.2.10.6 Seudo instrucción “Ds”.

Definición: Esta seudo instrucción altera el contador de localidades para reservar espacio en memoria.

Formato: En el campo instrucción debe estar la palabra DS y en el campo operando debe constar el valor con el que se desea alterar el contador de localidades.

Ejemplo:

Si el contador de localidades tiene inicialmente un valor de 00h, al escribir la siguiente secuencia:

```
Plastic    ds    20
           db    20,1001b,0ah
```

El archivo “.LST” resultante será el siguiente:

		1	plastic	ds	20
		2		db	20,1001b,0Ah
0014	14 09 0A				

FIG.2.14 Archivo de listado resultante para la seudo instrucción DS.

Como se puede ver, el resultado de la seudo instrucción:

```
“db 20,1001b,0Ah”
```

se almacenará en la localidad número 14h, dejando las primeras veinte localidades sin ninguna instrucción.

2.2.10.7 Seudo instrucción "Db".

Definición: Esta seudo instrucción reserva espacio en la memoria de programa al igual que "DS" con la diferencia de que permite guardar valores o caracteres iniciales. De esta forma se puede tener almacenado en memoria ROM tablas de datos o mensajes de texto.

Formato: En el campo instrucción debe constar la palabra DB y en el campo operando se debe escribir lo que se desea almacenar en la memoria de programa. Las cadenas de texto deben estar encerradas entre comillas.

Si dentro del mensaje se coloca el símbolo "\" y uno de los caracteres que se muestra en la tabla 2.1, el programa deberá reemplazar estos símbolos por su código equivalente.

CARACTERES CON "\"		
CARÁCTER	CARÁCTER INSERTADO	Equivalente
\n	Nueva línea	0A
\r	Retomo de carro	0D
\t	Tabulador	09
\0	Carácter nulo	0
\x	La letra x debe estar seguida por 2 dígitos hexadecimales.	\x0A devuelve 0A

TABLA 2.1 opciones para seudo instrucción DB.

En el siguiente cuadro se muestra el archivo ".LST" resultante si se ejecuta un ejemplo con cada uno de estos caracteres:

Lenguaje de máquina	#línea	Lenguaje ensamblador
0000 41 0A	1	db "A\n"
0002 41 0D	2	db "A\r"
0004 41 09	3	db "A\t"
0006 41 40	4	db "A\x40"
	5	END
	6	

FIG.2.15 Archivo de listado para la seudo instrucción DB.

2.2.10.8 Seudo instrucción "Dw".

Definición: Permite reservar espacios en palabras de dos bytes. Por esta razón no están permitidas cadenas de más de dos caracteres de longitud.

Formato: En el campo instrucción debe constar la palabra DW. En el campo operando se debe escribir lo que se desea almacenar en la memoria de programa. Las cadenas de texto deben estar encerradas entre comillas.

El siguiente ejemplo muestra el formato de esta instrucción y los resultados luego de ensamblarla.

Para las siguientes sentencias:

```

plastic   dw   "hola a todos"
          dw   100h,1000
          dw   0101010b
  
```

FIG. 2.16 Ejemplo para seudo instrucción "DW".

El Archivo ".LST" que se generará será el siguiente:

```

0000   686F           1   plastic   dw   "hola a todos"
          0100           2           dw   100h,1000
0004   03E8
0006   002A           3           dw   0101010b
  
```

FIG. 2.17 Archivo "LST".

2.2.10.9 Seudo instrucciones "Proc", "Endproc" y etiquetas locales.

Definición: A través de "Proc" y "Endproc" se pueden establecer procedimientos dentro de un programa principal. La característica más importante de estos procedimientos es la de definir variables locales.

Formato: Debe tener una etiqueta, en el campo instrucción debe estar la palabra "Proc" para iniciar un procedimiento y la palabra "Endproc" para finalizar el procedimiento.

Las variables que se definan dentro del procedimiento son locales, es decir, tienen significado solo dentro del procedimiento, fuera de él no tienen validez, esto implica que el nombre de una variable local se puede volver a utilizar fuera del procedimiento. La condición inversa también se cumple, es decir que el nombre de una etiqueta del programa principal puede volver a utilizarse dentro de un procedimiento sin ningún problema.

Una condición importante es que todos los procedimientos PROC-ENDPROC deben tener un nombre. El nombre asignado a un procedimiento es la etiqueta que aparece en la línea que contiene a la pseudo instrucción "PROC".

La principal ventaja de utilizar procedimientos es el permitir modularizar la programación.

En el siguiente ejemplo se muestra la manera en que se pueden utilizar estas pseudo instrucciones.

Para el siguiente programa:

inicio	mov	a, inicio	
ejemplo	proc		
inicio	mov	a, inicio	
etiq1	mov	a, etiq1	
	endproc		
etiq1	mov	a, etiq1	
	end		

Procedimiento dentro del programa principal y variables locales

FIG.2.18 Forma de empleo de las pseudo instrucciones PROC y END PROC.

El archivo ".LST" será el siguiente:

0000	E500	1	inicio	mov	a, inicio
		2	ejemplo	proc	
0002	E502	3	inicio	mov	a, inicio
0004	E504	4	etiq1	mov	a, etiq1
		5		endproc	
0006	E506	6	etiq1	mov	a, etiq1
		7		end	

FIG.2.19 Archivo de listado

2.2.10.10 Instrucción "Org".

Definición: Permite colocar instrucciones de programa en una dirección determinada por el usuario. Para lograr esto, el ensamblador altera el contador de localidades con el valor que se especifique en el campo operando.

Ejemplo:

Si se desea colocar un grupo de instrucciones en la dirección 9h y otro grupo en la dirección 18h se lo hará de la siguiente manera:

```

inicio   org      09h
          mov      a,inicio
ejemplo  proc
inicio   mov      a,inicio
etiq1    mov      a,etiq1
          endproc
          org      18h
etiq1    mov      a,40h
          mov      a,etiq1
          end
  
```

FIG.2.20 Archivo fuente con seudo instrucción ORG.

El archivo ".LST" resultante será el siguiente:

```

0009  E509      1      inicio   org      09h
          2      ejemplo  mov      a,inicio
          3      inicio   proc
000B  E50B      4      inicio   mov      a,inicio
000D  E50D      5      etiq1    mov      a,etiq1
          6      endproc
          7      org      18h
0018  E540      8      etiq1    mov      a,40h
001A  E51A      9      etiq1    mov      a,etiq1
          10     end
  
```

FIG.2.21 Archivo de listado.

2.2.10.11 Seudo instrucción "End".

Sirve para indicar al ensamblador que el programa ha terminado y que debe concluir con todas las operaciones pendientes, tal como, la impresión de caracteres del módulo objeto o archivos de listado.

2.2.10.12 Comodín “\$”.

Este comodín se emplea cada vez que se desee conocer el valor del contador de localidades. La característica principal es que se lo puede emplear para definir etiquetas o cualquier expresión matemática.

Hasta aquí se han estudiado las condiciones con las que deben cumplir los archivos generados por el programa, así como también las características y condiciones que deben cumplir las pseudo instrucciones propias del ensamblador.

En lo que resta del capítulo se tratará la estructura del programa ensamblador, los procesos que se debe dar a cada instrucción y las soluciones que se emplearon para cumplir con las características y condiciones de los archivos que debe generar y de las pseudo instrucciones que debe interpretar.

2.3 ALGORITMOS Y DIAGRAMAS DE FLUJO DEL PROGRAMA PRINCIPAL

El proceso de ensamblado sigue una secuencia de varios pasos hasta obtener el resultado final que es el archivo ejecutable. Es por este motivo que el programa ensamblador está constituido por varios subprogramas que realizan diversos procesos sobre el módulo fuente, quitando datos innecesarios (como los comentarios) y agregando otros (como es el caso de expansión de macro instrucciones).

La siguiente figura muestra con más detalle cada uno de estos subprogramas:

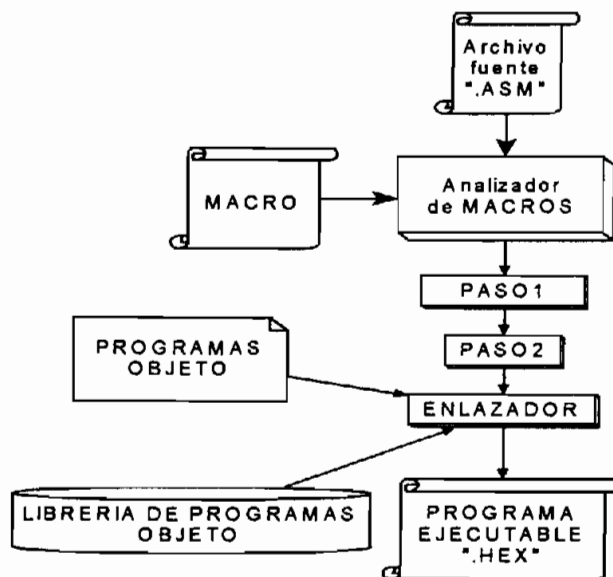


FIG. 2.22 Diagrama de flujo del proceso de ensamblado.

2.3.1 Analizador de macro instrucciones

Tiene como primer objetivo encontrar sentencias dentro del módulo fuente que definan la ubicación y nombre de una macro. Durante este proceso el módulo fuente no debe sufrir ninguna modificación.

Una vez definido el nombre y la ubicación de la macro, el segundo objetivo de este subprograma es encontrar invocaciones con los nombres de las macro instrucciones y luego expandirlas en el mismo sitio en donde se las invocó.

Para mayor seguridad y para no alterar el módulo fuente las expansiones se realizan sobre una copia del módulo fuente que tiene el mismo nombre pero cuya extensión es ".MCR".

El diagrama de flujo para el analizador de macro instrucciones está en las siguientes figuras:

La figura 2.23 muestra la forma en que el programa busca las sentencias que definen el nombre y la ubicación de las macro instrucciones dentro del módulo fuente.

La figura 2.24 indica la manera en que el programa expande las macro instrucciones en el lugar donde se las invocó.

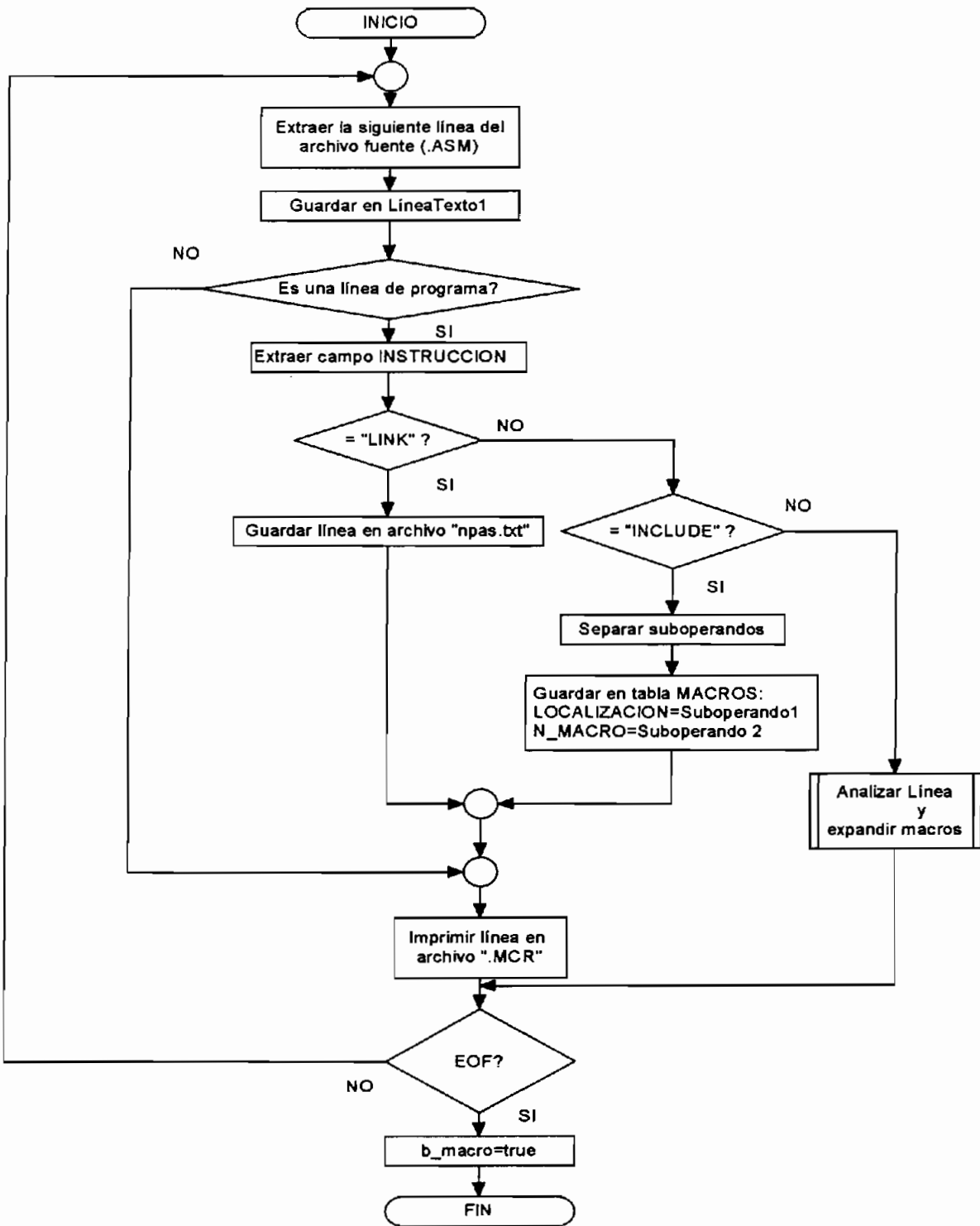


FIG 2.23 Analizador de macro instrucciones.

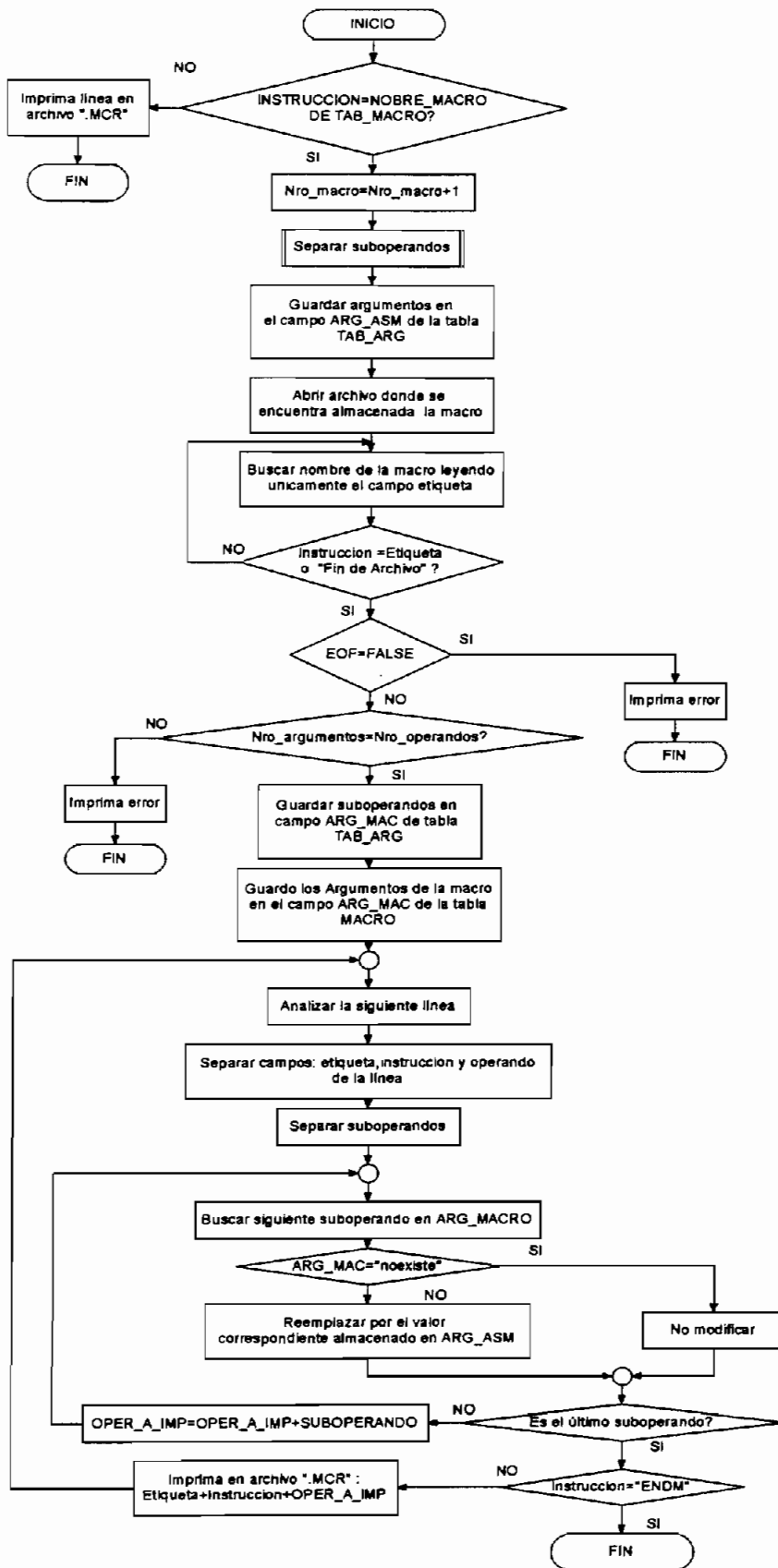


FIG.2.24 Subrutina Analizar línea y expandir macros.

2.3.2 Paso1

El paso 1 tiene por objetivo evaluar todas las etiquetas, conocer sus valores y guardarlos en la tabla de símbolos llamada "TABSIM". En este paso se debe asignar valores a todos los nombres simbólicos, resolver referencias adelantadas y referencias externas.

Debido a que se pretende mostrar el ensamble en forma didáctica, también deberá realizar un corto análisis de las instrucciones para mostrar los resultados en un archivo de texto con formato de archivo de listado. El archivo ".LST" resultante de este paso no es el definitivo puesto que las referencias adelantadas se resuelven en el segundo paso.

Para el siguiente programa:

```

;Area de comandos
;Area de definicion de variables
;Area de instrucciones
etiql equ etiql
etiql equ 10100b
;Area de instrucciones
mov a,etiql
mov r0,etiql
sjmp $
;Area para tablas

end

```

FIG.2.25 Módulo fuente.

El archivo de listado será el siguiente:

		1		;Area de comandos
		2		
		3		;Area de definicion de variables
		4		
		5		;Area de instrucciones
,ETIQ2	=	6	etiql	equ etiql
0014	=	7	etiql	equ 10100b
		8		;Area de instrucciones
0000	E5ETIQ1	9	mov	a,etiql
0002	A814	10	mov	r0,etiql
0004	80FE	11	sjmp	\$
		12		;Area para tablas
		13		
		14		end

FIG.2.26 Archivo de listado preliminar.

En la figura 2.27 se muestra el diagrama de flujo del paso1.

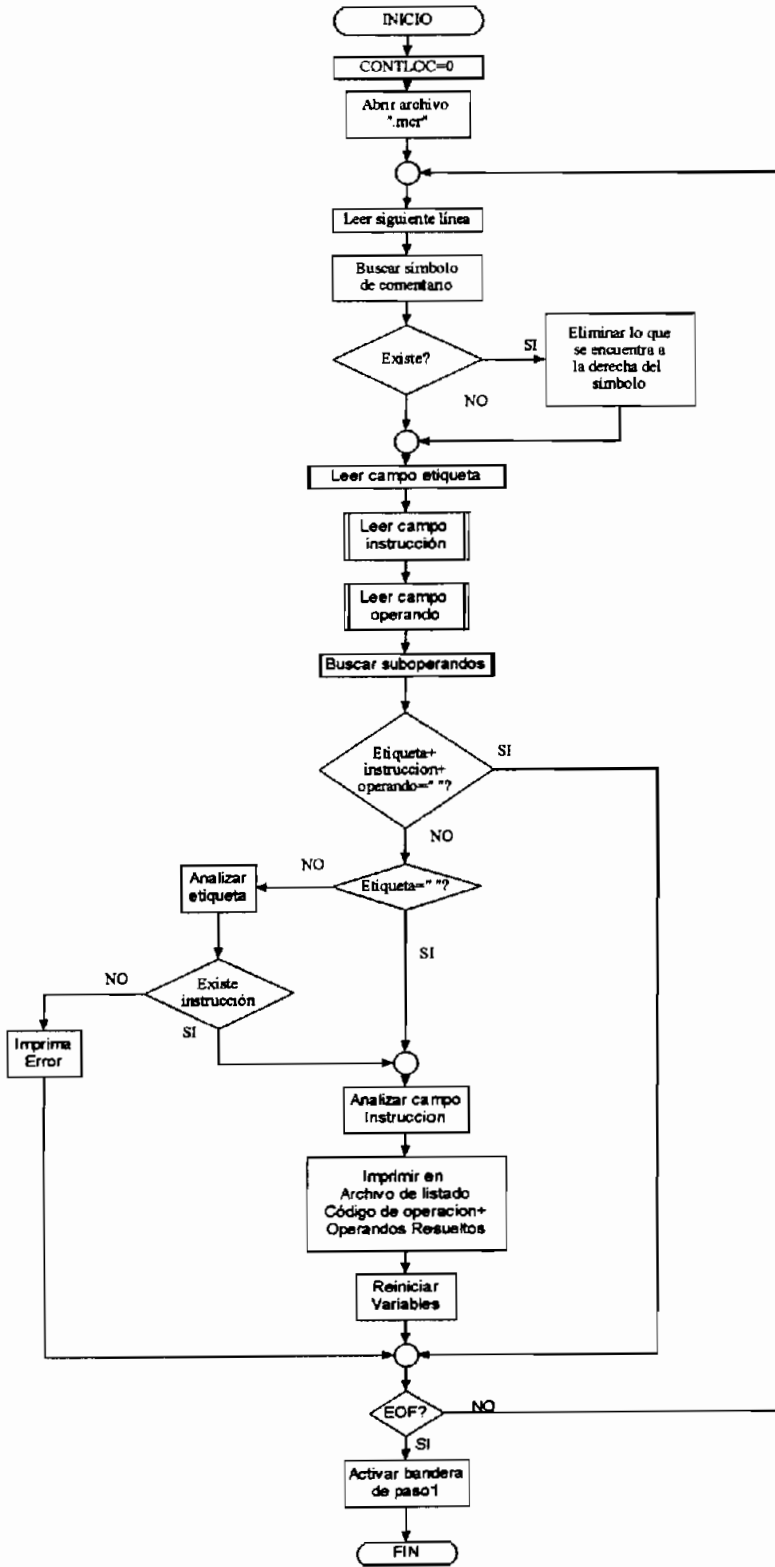


FIG. 2.27 Diagrama de flujo para el paso 1.

2.3.3 Paso 2.

El paso dos debe cumplir con los siguientes objetivos:

1. Ensamblar instrucciones (traducción de los códigos de operación y asignación de direcciones).
2. Generar los valores de datos definidos por las pseudo instrucciones (DB, DW, etc.).
3. Realizar los procesamientos faltantes de las pseudo instrucciones.
4. Escribir el módulo objeto y el archivo de listado del ensamblado definitivo.

El archivo ".LST" definitivo generado por el paso2 para el ejemplo de la figura 2.25 es el siguiente:

		1			;Area de comandos
		2			
		3			;Area de definicion de variables
		4			
		5			;Area de instrucciones
0000	-	6	etiq1	equ	etiq2
0014	-	7	etiq2	equ	10100b
		8			;Area de instrucciones
0000	E500	9		mov	a,etiq1
0002	A814	10		mov	r0,etiq2
0004	80FE	11		sjmp	\$
		12			;Area para tablas
		13			
		14			end

FIG. 2.28 Archivo de listado definitivo

La figura 2.29 muestra el esquema del paso 2 para el proceso de ensamblado.

Puesto que, en la primera fase, ya se resolvió la mayoría de las etiquetas y expresiones, en este paso se procede a crear el módulo objeto y el archivo de listado definitivo. Para lograrlo es necesario tomar los valores almacenados en TABSIM y traducir los códigos de lenguaje ensamblador a lenguaje de máquina.

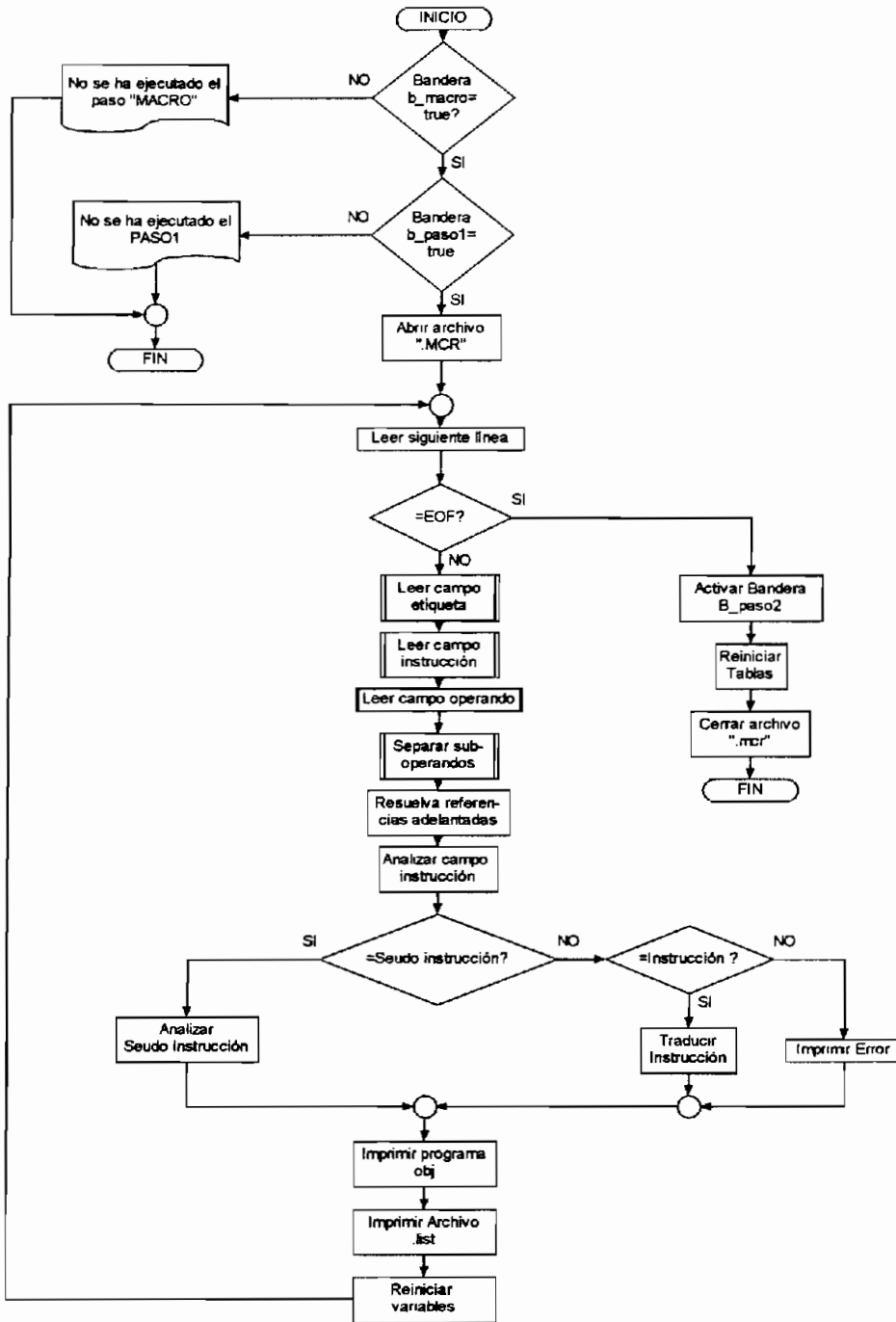


FIG. 2.29 Paso 2.

2.3.4 Programa enlazador.

Los objetivos de este programa son los siguientes:

1. Almacenar los valores que contengan las etiquetas públicas.

2. Resolver expresiones matemáticas que contienen variables externas.
3. Unir los archivos objeto sin que se presenten conflictos en el almacenamiento en memoria.

Para cumplir con los objetivos anteriormente mencionados el programa enlazador debe tomar los valores almacenados en los registros que vienen incorporados en el módulo objeto.

```

:090000008500FD F500E500E50086
:10001500E52080FE686F6C61206D756E646FE70D7D
:0D0025000C0A0B03E8007B00001925641C89
:00000001EF
D,SUPR,20
D,AVPAG,3
D,REPAG,3
E,BLOQNUM
E,KLM
M,1,2,+,BLOQNUM-1
M,2,2,+,KLM+2
M,4,2,+,BLOQNUM+20
M,6,2,+,BLOQNUM-KLM
M,8,2,+,BLOQNUM-(KLM+32)*3)

```

Estos registros son utilizados por el programa enlazador

FIG 2.30 Módulo objeto con referencias externas.

De esta forma para resolver el problema de las variables externas, primero se debe recolectar todos los datos de las variables públicas definidas en los distintos programas objeto que se van a enlazar.

Una vez que se ha recolectado toda la información de las etiquetas públicas el programa ya puede resolver las expresiones matemáticas pendientes y reemplazar su valor en las localidades de memoria que correspondan.

Se debe mencionar que las direcciones y nombres de los archivos objeto están almacenados en un archivo de texto ubicado en la siguiente dirección "c:\npas.txt".

En la figura 2.31 muestra el diagrama de flujo para este programa.

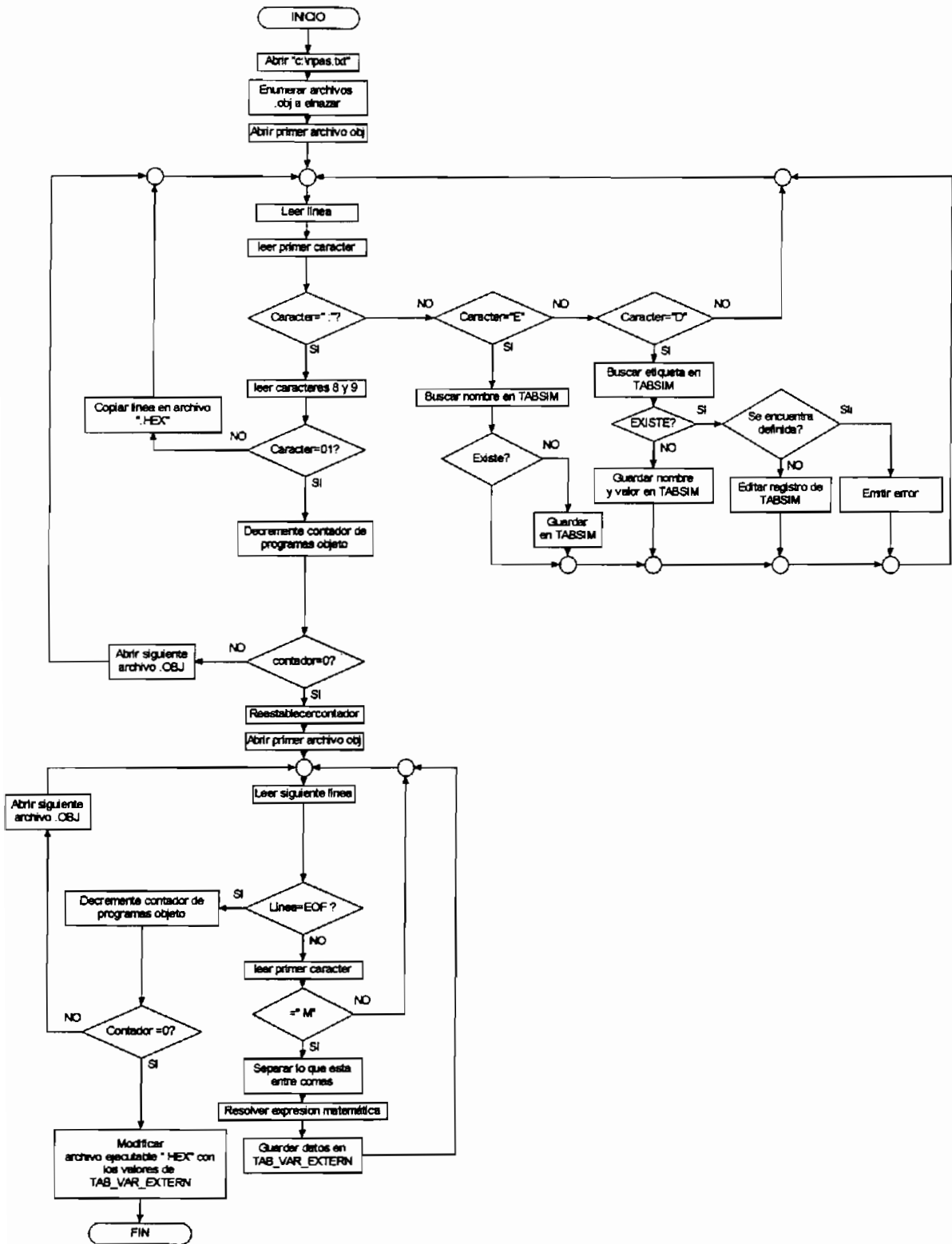


FIG 2.31 Enlazador.

2.4 DIAGRAMAS DE FLUJO DE SUBRUTINAS.

2.4.1 Subrutina “buscar_definicion”.

Esta subrutina permite determinar la naturaleza y el valor en formato hexadecimal de una variable. Los datos que se analizan en esta subrutina ingresan a través de la variable “resultA”, de esta forma si en resultA se encuentra almacenada una etiqueta, el resultado que devolverá será el valor en formato hexadecimal de dicha etiqueta, siempre y cuando se encuentre completamente definida dentro de la tabla de símbolos TABSIM, caso contrario, la subrutina devolverá el valor “00” y activará una bandera llamada “b_etiqueta” la cual le indica al programa principal que dicha etiqueta no se encuentra definida aún.

Procede de igual manera con las expresiones matemáticas, es decir, si todas las variables de una expresión están definidas, la subrutina devolverá un valor en formato hexadecimal, caso contrario, devolverá el valor “00” más la bandera “b_etiqueta” con un valor de “verdadero”.

Los siguientes son los objetivos que debe cumplir esta subrutina:

- Todo valor numérico (independientemente de su base) que ingrese a la subrutina será devuelto en formato hexadecimal.
- Deberá tener la capacidad de resolver expresiones matemáticas.
- Podrá devolver el valor de etiquetas ya definidas en el paso 1. Si no esta definida devolverá el valor 00.
- Deberá aceptar operaciones con bits direccionables. Si los bits no están dentro del rango permitido deberá emitir un error.

Para cumplir con estos objetivos la subrutina utiliza los datos que se encuentran almacenados en la tabla de símbolos “TABSIM” y para la evaluación de expresiones matemáticas utiliza la subrutina “analizar_expresión” la cual se verá con mayor detalle más adelante.

En el siguiente ejemplo se puede ver la forma como trabaja esta subrutina.

Para el siguiente grupo de instrucciones:

```

lcd equ 1
leds equ 3h
mov dptr,#lcd+1
swit equ 0Ah
Suma equ (lcd+leds)
Mult equ (lcd*swit)
div equ swit/leds

```

FIG 2.32 Archivo fuente.

Se obtendrá los siguientes resultados:

0001	=	1	lcd equ 1
0003	=	2	leds equ 3h
0000	=	3	mov dptr,#lcd+1
000A	=	4	swit equ 0Ah
0004	=	5	Suma equ (lcd+leds)
000A	=	6	Mult equ (lcd*swit)
0003	=	7	div equ swit/leds

FIG. 2.33 Archivo de listado.

Como se puede observar en la línea 5 de la figura 2.33, el valor que se asigna a la etiqueta "Suma" es el resultado de sumar el valor de "lcd" más el valor de la etiqueta "leds". Las mismas que se encuentran ya definidas en la tabla de símbolos TABSIM.

El diagrama de flujo para esta subrutina es el siguiente:

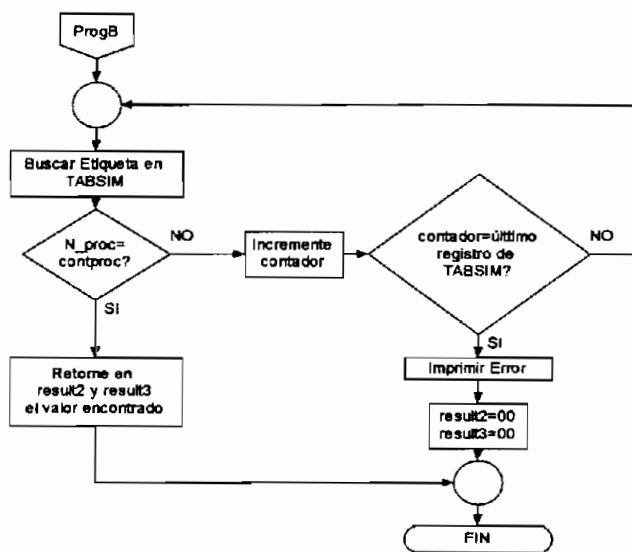
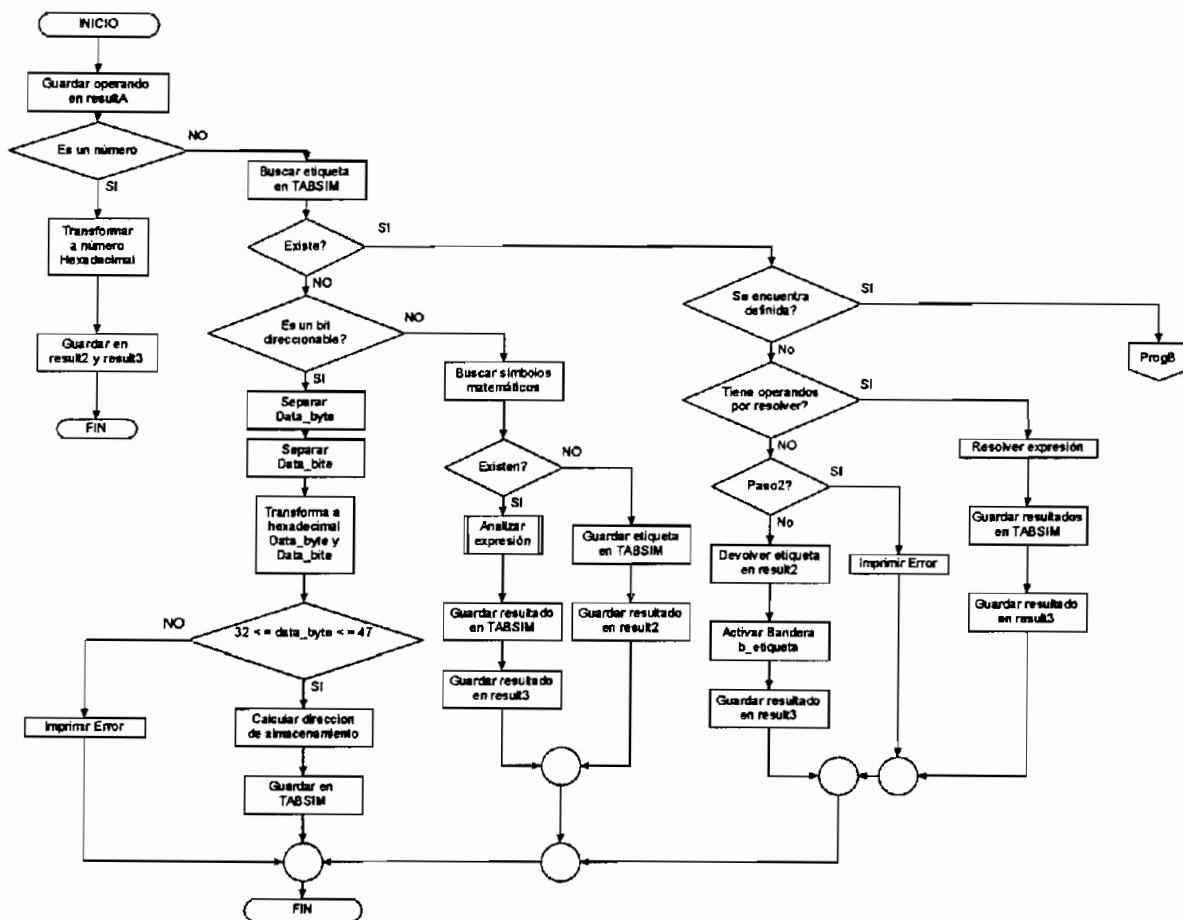


FIG 2.34 Subrutina "buscar_definicion".

2.4.2 Subrutina "analizar_expresion".

Esta subrutina evalúa las expresiones matemáticas que se encuentran almacenadas en la variable resultA, usando los datos que se encuentran guardados en la tabla de símbolos TABSIM. Para lograr este objetivo la subrutina busca símbolos matemáticos, tales como: +, -, *, /, (), { } dentro de la expresión almacenada en resultA. De igual manera identifica y asigna valores a las etiquetas que componen la ecuación matemática.

Si después de haber identificado todas las etiquetas que intervienen en la ecuación existe al menos una que no ha sido definida, la subrutina no evalúa la expresión y activa una bandera que indica que la expresión tiene aún etiquetas por resolver.

El diagrama de flujo para esta subrutina es el siguiente:

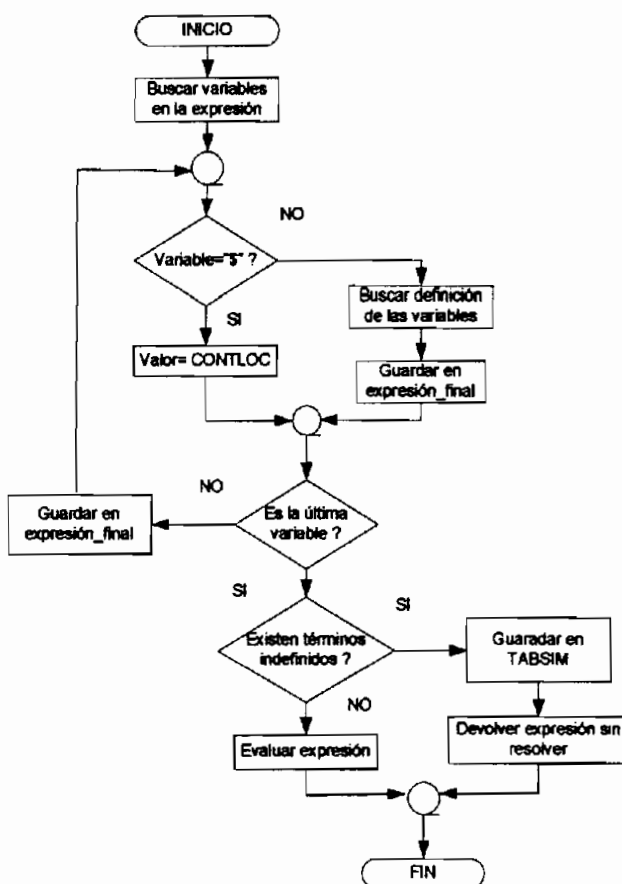


FIG. 2.35 subrutina "analizar_expresion"

2.4.3 Subrutina "leer_campo_etiqueta".

Permite reconocer lo que contiene en el campo etiqueta del módulo fuente ".ASM".

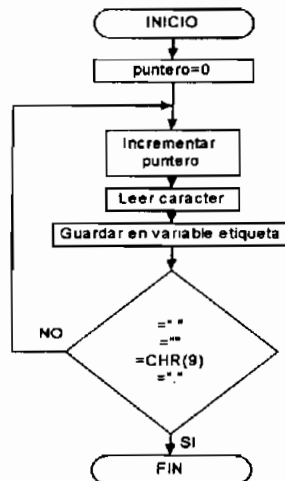


FIG. 2.36 Subrutina "leer_campo_etiqueta".

2.4.4 Subrutina "leer_campo_instruccion".

Con esta subrutina el programa puede leer las instrucciones, seudo instrucciones y llamadas a macro instrucciones que se encuentran escritas en el campo instrucción del módulo fuente.

Su diagrama de flujo es el siguiente:

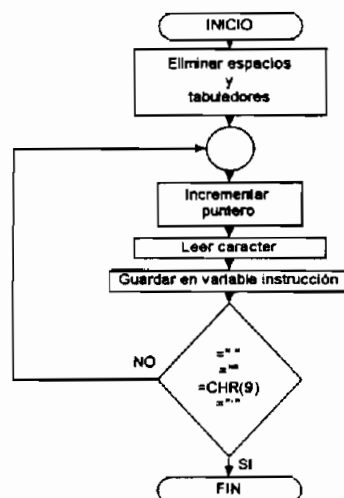


FIG. 2.37 Subrutina "leer_campo_instrucción"

2.4.5 Subrutina "leer_campo_operando".

Permite leer el contenido del campo operando. Para lograr esto el programa analiza caracter por caracter la línea de texto hasta encontrar cualquiera de los siguientes caracteres:

- “”
- “ ”, espacios en blanco.
- “;”, caracteres de comentarios.
- Tabuladores.

Su diagrama de flujo es el siguiente:

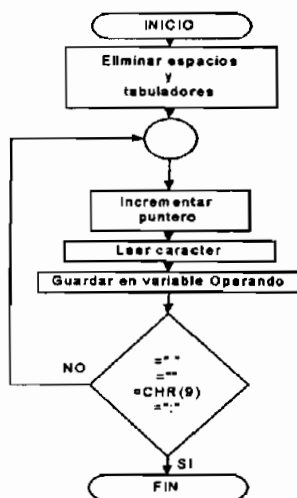


FIG. 2.38 Subrutina "leer_campo_operando".

2.4.6 Subrutina "separador".

Debido a que el campo operando suele contener más de un suboperando. Esta subrutina permite descomponer al campo operando y guardar sus partes en una matriz de datos llamada operx(i).

Por ejemplo, la siguiente sentencia tiene 3 suboperandos:

db "hola mundo",10h,20

Luego de pasar por la subrutina "separador" lo que quedará almacenado en la matriz será lo siguiente:

Operx(1)= "hola mundo"

Operx(2)=10h

Operx(3)=20

Su diagrama de flujo es el siguiente:

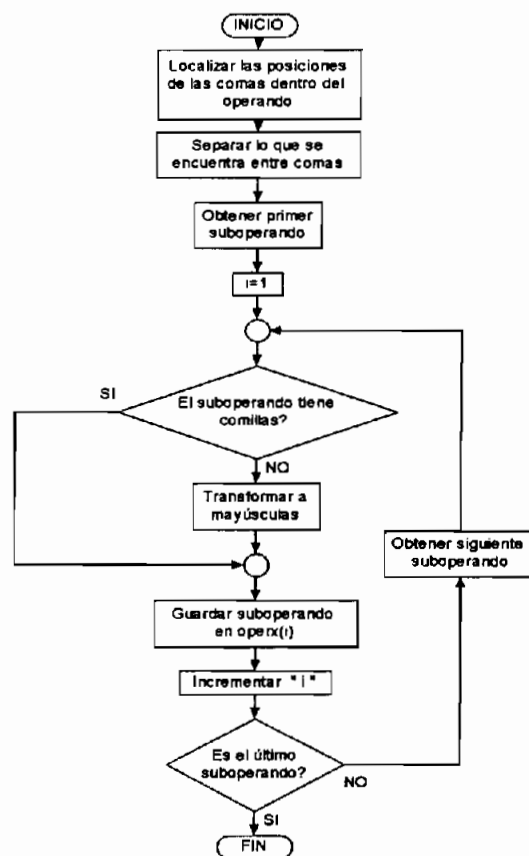


FIG. 2.39 Subrutina separador

2.4.7 Subrutina para borrar datos de una tabla.

Borra los datos almacenados en los registros de las tablas empleadas en el programa.

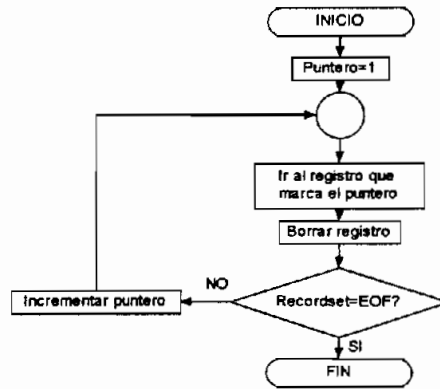


FIG. 2.40 Subrutina para borrar datos de una tabla.

2.4.8 Subrutina "buscar_etiqueta".

Esta subrutina busca dentro de la tabla de datos TABSIM una cadena de caracteres que coincida con lo que se encuentre almacenado en la variable "eti".

El diagrama de flujo es el siguiente:

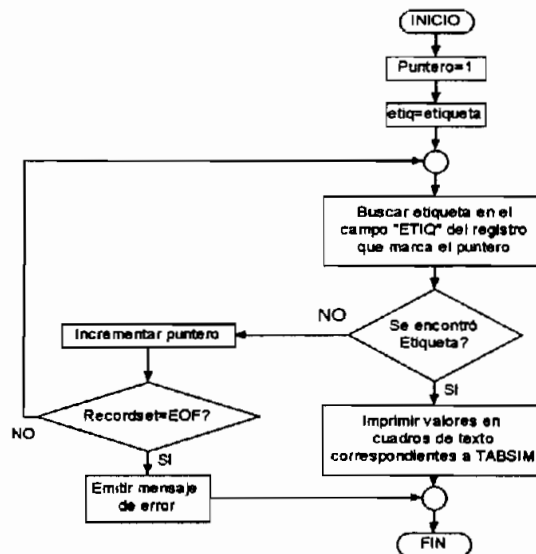


FIG. 2.41 Subrutina "buscar etiqueta"

2.4.9 Subrutina "buscar_sim_mat":

Busca símbolos matemáticos en una expresión. Estos símbolos matemáticos se encuentran almacenados en la tabla de datos SIM_MAT, cualquier símbolo que no se encuentre dentro de esta tabla será considerado como un carácter de texto.

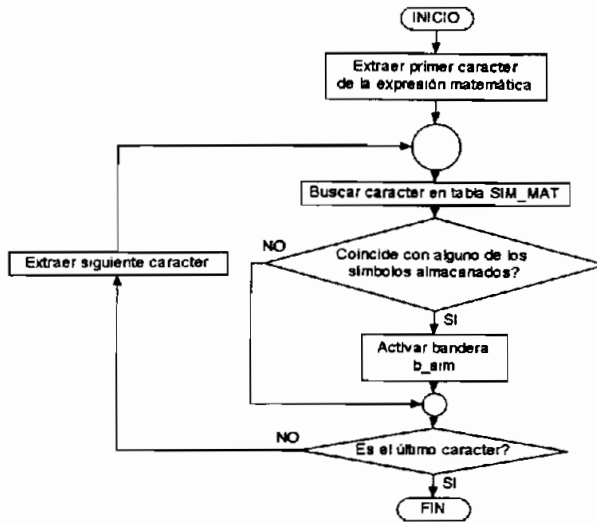


FIG. 2.42 Buscador de símbolos matemáticos.

2.4.10 Subrutina “calcular_desplazamientos”:

Esta subrutina se emplea en los casos en que se utilice instrucciones relativas. Estas instrucciones de salto no apuntan a una dirección específica, sino que, calculan un intervalo de salto. De esta manera cuando por algún motivo se desee relocalizar el archivo ejecutable en algún lugar de la memoria de programa no será necesario alterar el valor de estas instrucciones de salto.

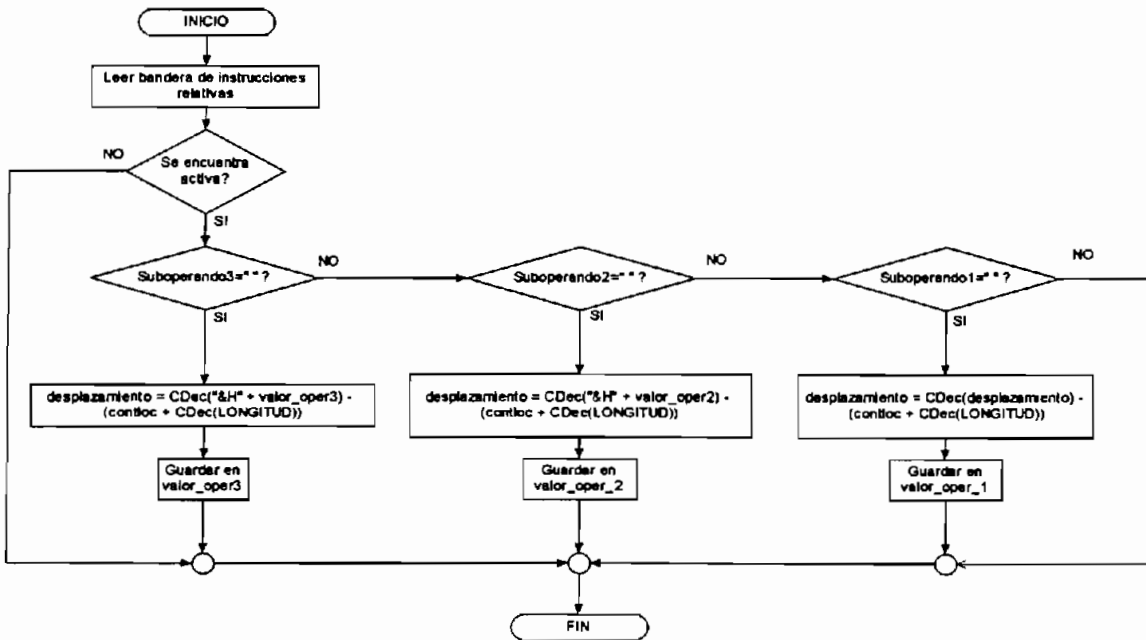


FIG. 2.43 Subrutina calcular desplazamientos.

2.4.11 Subrutina “calcular_longitud_operando”

Permite obtener el número de caracteres por los que debe estar conformado el o los operandos de una instrucción.

Por ejemplo, la instrucción: LCALL DATA, tiene una longitud de 3 bytes distribuidos de la siguiente manera:

- El primer byte es empleado por el código de la instrucción.
- El segundo y tercer byte son empleados para la dirección de salto, por lo tanto esta dirección estará conformada por 4 caracteres. Este es el valor que entrega esta subrutina.

Su diagrama de flujo es el siguiente:

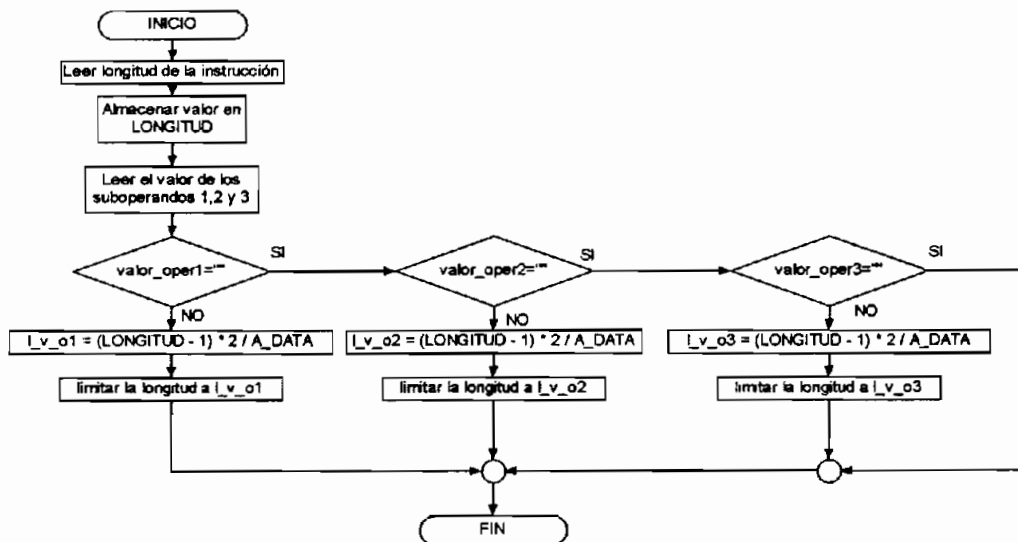


FIG. 2.44 Calcula la longitud del operando.

2.4.12 Subrutina “formato_operando”:

A través de esta subrutina se obtiene un campo operando estándar con el cual se puede buscar la instrucción en la tabla TABOP sin cometer ningún error.

Por ejemplo, para la sentencia MOV A,#40H, el operando estándar generado por esta subrutina es el siguiente:

MOV A,#DATA, es decir, todo suboperando que sea modificable por el usuario se lo reemplazará por la palabra DATA, de esta forma será más fácil realizar la búsqueda de la instrucción en la tabla de códigos TABOP. Además, si uno de los suboperandos es una etiqueta, esta subrutina le asignará un valor.

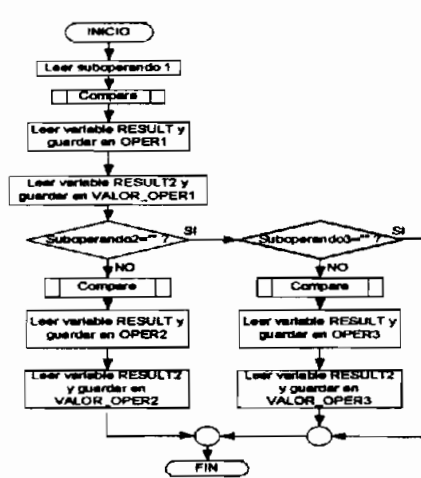


FIG. 2.45 Formato de operando.

2.4.13 Subrutina "compare"

Esta subrutina trabaja junto con la subrutina "formato_operando" y permite reconocer si lo que está en el operando es un dato o una etiqueta predefinida.

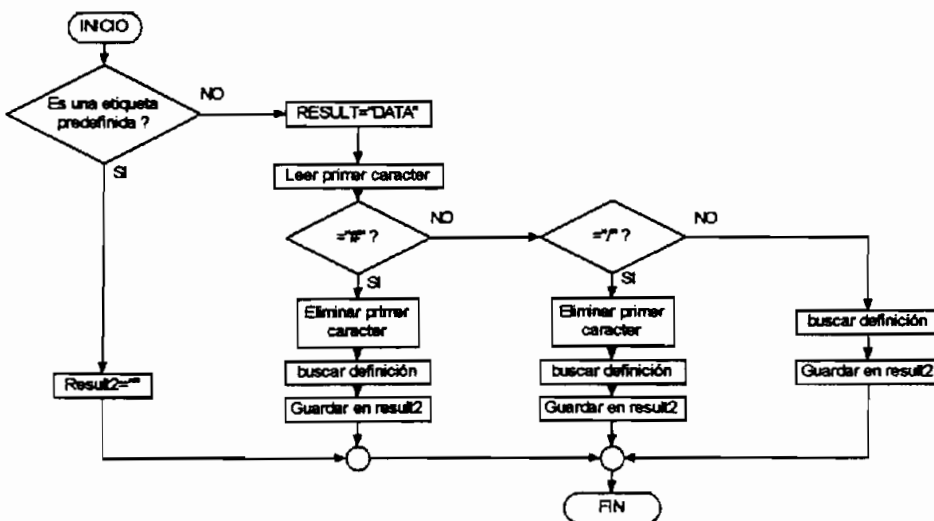


FIG. 2.46 Subrutina "compare".

2.4.14 ISubrutina "gen_archivo_lst".

Esta subrutina escribe el archivo de listado línea a línea usando lo que se encuentra almacenado en la variable print_lst.

Por lo genral el formato con el que se imprime este archivo es el siguiente:

Dirección + Código de máquina + Número de línea + Línea original del archivo fuente.

Las seudo instrucciones EQU, DB, DW tienen un formato de impresión distinto.

- Para el caso de la seudo instrucción EQU, el programa derá imprimir un código de 4 caracteres, que corresponden al valor asignado a una etiqueta, más la línea original del archivo fuente.
- Para el caso de la seudo instrucción DB, se debe imprimir en la sección del código de máquina, cuatro palabras de dos caracteres por línea, cada uno separado por un espacio.
- Con DW se debe imprimir, en la sección del código de máquina, dos palabras de cuatro caracteres separadas con un espacio.

En la siguiente figura se muestra un ejemplo con estas dos seudo instrucciones.

0000	68 6F 6C 61	1	plastic	db	"hola a todos"
0004	20 61 20 74				
0008	6F 64 6F 73				
000C	0100 03E8	2		dw	100h,1000
		3		dw	0101010b
0010	002A				

FIG.2.47 Archivo de listado.

Su diagrama de flujo es el siguiente:

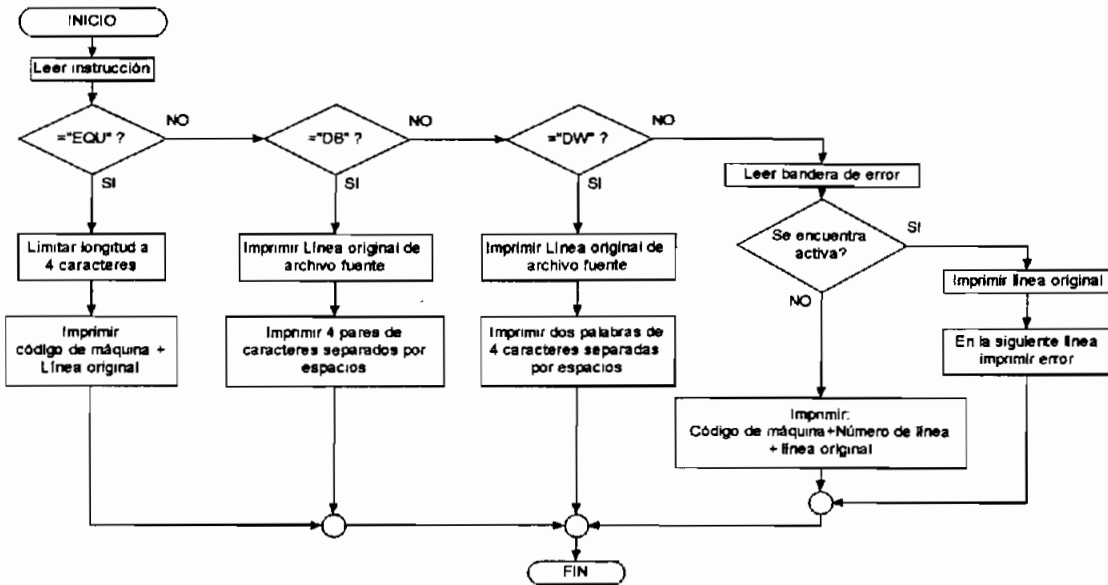


FIG.2.48 Subrutina para generar archivo .LST.

A continuación se describirá las características de las tablas de la base de datos que utiliza el programa ensamblador.

2.5 BASE DE DATOS Y TABLAS UTILIZADAS POR EL PROGRAMA ENSAMBLADOR.

El programa ensamblador utiliza la base de datos "ensamblador.mdb" localizada en la carpeta "c:\ensamblador2" la misma que contiene las siguientes tablas:

- Tabla de símbolos denominada TABSIM.
- Tabla de opcodes llamada TABOP.
- Tabla para uso del enlazador TABLINK.
- Tabla para almacenar argumentos de macro instrucciones TABARG.
- Tabla para variables externas TAB_VAR_EXTERN.
- Tabla de símbolos matemáticos TABSIM.

- Tabla para localización de las macro instrucciones.

2.5.1 Tabla de símbolos "TABSIM".

Esta tabla almacena las etiquetas creadas por el usuario; también contiene 174 direcciones de datos predefinidos a través de etiquetas, las mismas que se encuentran enlistadas en el anexo B.

La tabla está constituida por los campos: ID, ETIQ, DEFI, EXP_VAL, LOCALIZACION, NPROC, PUBLIC_EXTERN, MODIFICACION, LONGITUD, BANDERA_M.

- El campo "ID" enumera los registros de la tabla.
- El campo "ETIQ" almacena el nombre de las etiquetas.
- El campo "DEFI" almacena caracteres que le indican al programa principal si el contenido del campo ETIQ se encuentra definido completamente o si una expresión almacenada en ETIQ tiene variables por resolver.

Los caracteres que se utilizan para este trabajo son los siguientes:

- "*" indica que la variable o expresión no se encuentra definida.
- "&0" indica que no existen variables por resolver.
- "&#" indica el número de variables que faltan por resolver en una expresión matemática.

Además esta tabla contiene los siguientes campos:

- El campo EXP_VAL almacena el valor de cada etiqueta o expresión.
- El campo LOCALIZACION contiene la dirección de carga de las instrucciones que necesitan ser alteradas por el programa enlazador y

generalmente corresponden a variables utilizadas con instrucciones absolutas o a variables externas.

- El campo "NPROC" almacena el número del procedimiento en el cual se encuentra una etiqueta. Existe una excepción para las variables predefinidas, en cuyo caso el carácter almacenado en este campo es "*" que indica que se pueden usar bajo cualquier número de procedimiento.
- El campo "PUBLIC_EXTERN" almacena los caracteres "P" y "E" los mismos que indican si una variable es pública o externa.
- El campo "MODIFICACION" si tiene almacenado el número "1" indica que es una variable sujeta a ser modificada por el programa enlazador, el caso contrario sucede si tiene almacenado un "0".
- El campo "LONGITUD" es utilizado por el programa enlazador para guardar el número de caracteres que necesitan ser modificados antes de realizar el proceso de enlace.
- El campo "BANDERA_M" es utilizado por el programa enlazador para saber que etiquetas necesitan ser modificadas en su valor.

2.5.2 Tabla de opcodes "TABOP".

En esta tabla se encuentran almacenados todos los opcodes del microcontrolador 8051/52.

Id	CODI	LONG	A_data	REL_ABS_AD	MNEM	OPER
1					noexiste	
2 00	1		0		NOP	
3 01	2		1 A		AJMP	data
4 02	3		1 A		LJMP	data
5 03	1		0 A		RR	A
6 04	1		0 A		INC	A
7 05	2		1 A		INC	data
8 06	1		0 A		INC	@RD

FIG. 2.49 Tabla con opcodes.

Esta conformada por los campos: "id", "CODI", "LONG", "A_data", "REL_ABS_ADD", "MNEM", "OPER".

- El campo "id" enumera los registros de la tabla.
- El campo "CODI" contiene el código de máquina de la instrucción.
- El campo "LONG" almacena la longitud de la instrucción.
- El campo "A_data" contiene el área o número de bytes que ocupa el operando en el código de máquina.
- El campo "REL_ABS_ADD" identifica a la instrucción como absoluta o relativa.
- El campo "MNEM", contiene el mnemónico del código de máquina.
- El campo "OPER" almacena el operando de la instrucción en formato estándar.

2.5.3 Tabla para uso del enlazador "TABLINK".

Esta tabla esta conformada por los campos: "id", "Dirección_obj", "Nombre_obj".

	id	Direccion_obj	Nombre_obj
▶		noexiste	
*		0	

FIG. 2.50 Tabla de datos empleada para el enlazado.

En esta tabla se almacenan la dirección y el nombre de los archivos objeto que se desean enlazar.

2.5.4 Tabla para almacenar argumentos de macro instrucciones "TABARG".

Está constituida por los campos: "id", "ARG_ASM", "ARG_MACRO".

	Id	ARG_ASM	ARG_MACRO
▶		noexiste	noexiste
*	0		

FIG. 2.51 Tabla para almacenar argumentos de macro instrucciones.

- El campo "ARG_ASM" contiene los argumentos de las invocaciones a macro que se realizan en el módulo fuente.
- El campo "ARG_MACRO" almacena los argumentos que contiene la macro en su archivo de origen el cual puede ser un archivo ".ASM" o cualquier archivo de texto.

2.5.5 Tabla para variables externas "TAB_VAR_EXTERN".

Esta tabla se encuentra conformada por los campos: ID, "VARIABLE", "LOCALIZACION", "LONGITUD", "SIMBOLO", "PROGRAMA".

	ID	VARIABLE	LOCALIZACION	LONGITUD	SIMBOLO	PROGRAMA
▶		noexiste				
*	0					

FIG. 2.52 Tabla de datos para almacenar variables externas.

- El campo "VARIABLE" almacena el nombre de la variable tanto pública como externa.
- "LOCALIZACION" contiene la dirección de carga de la variable.
- "LONGITUD" almacena el número de caracteres que se van a modificar en el momento en que el programa enlazador empiece a modificar el módulo objeto.
- "SIMBOLO" indica la operación que se debe ejecutar al momento de realizar la modificación.
- "PROGRAMA" indica a que módulo objeto pertenece la variable que se pretende modificar.

2.5.6 Tabla de símbolos matemáticos SIM_MAT.

Esta compuesta por los campos: "Id1", "SIMBOLO1", "PRIORIDAD".

	Id1	SIMBOLO1	PRIORIDAD
	0	noexiste	
	1	*	1
	2	/	1
	3	+	2
	4	-	2
▶	5	(3
	6)	3
	7	{	4
	8	}	4
*	0		

FIG. 2.53 Símbolos matemáticos reconocidos por el programa ensamblador.

- El campo "SIMBOLO1" contiene todos los símbolos matemáticos que puede reconocer el programa.
- El campo "PRIORIDAD" indica al programa que símbolo primero debe ser considerado para resolver una expresión.

Por ejemplo en la expresión $(2+4)*2$ el programa primero deberá resolver lo que se encuentra dentro del paréntesis ya este tiene mayor prioridad que el símbolo $*$.

2.5.7 Tabla "MACRO".

La tabla esta conformada por los campos: "id", "NOMBRE_MACRO", "LOCALIZACIÓN".

	id	NOMBRE_MACRO	LOCALIZACION
▶	0	noexiste	
*	0		

FIG. 2.54 Tabla con nombre y localización de macro instrucciones.

- "NOMBRE_MACRO" almacena el nombre de la macro.
- "LOCALIZACIÓN" contiene la dirección del archivo desde el cual se va a extraer la macro.

CAPITULO 3

3. PRUEBAS Y RESULTADOS

3.1 PRUEBAS CON EL ENSAMBLADOR TRANSPARENTE PARA EL USUARIO.

3.1.1 Definición de etiquetas y resolución de expresiones.

A continuación se presentan los resultados que entrega el programa al resolver los cuatro problemas principales que existen al momento de definir una etiqueta.

1. Asignación de valores a las etiquetas.

En la figura 3.1 se muestra un grupo de etiquetas a las cuales se asigna un valor con bases hexadecimal, octal, decimal, y binario a través de la pseudo instrucción "EQU".

hexa	equ	17h
octal	equ	22o
decimal	equ	10d
decimal2	equ	5
binario	equ	110011b

FIG. 3.1 Pseudo instrucción "EQU".

El archivo ".LST" resultante es el siguiente:

0017	=	1	hexa	equ	17h
0012	=	2	octal	equ	22o
000A	=	3	decimal	equ	10d
0005	=	4	decimal2	equ	5
0033	=	5	binario	equ	110011b

FIG. 3.2 Archivo LST.

La otra forma de asignar valores es simplemente escribiendo una palabra en el campo etiqueta del módulo fuente. El programa ensamblador asignará a esta etiqueta el valor que en ese instante tenga el contador de localidades.

0017	=	1	hexa	equ	17h
0012	=	2	octal	equ	22o
000A	=	3	decimal	equ	10d
0005	=	4	decimal2	equ	5
0033	=	5	binario	equ	110011b
0000	A840	6			;Area de instrucciones
0002	18	7		mov	R0,40h
0003	E502	8	nuevo	dec	R0
		9		mov	a,nuevo

FIG. 3.3 Archivo .LST.

Como se puede ver en la figura anterior el valor que el programa ensamblador asignó a la etiqueta "nuevo" es el número "02" el cual es el valor que tenía el contador de localidades al momento de analizar la línea 8.

Se debe recalcar que todas las asignaciones se guardarán como valores hexadecimales en el archivo de listado.

2. Resolución de referencias adelantadas.

En el siguiente ejemplo la etiqueta "LED" se define por otra etiqueta llamada "BIT" cuya definición se encuentra más adelante en el programa (referencia adelantada).

La etiqueta "LED" únicamente tendrá un valor real cuando el programa ensamblador conozca todas las variables de la expresión que la definen, para este caso cuando conozca el valor de la etiqueta "BIT".

LED	EQU	BIT+1
BIT	EQU	005

FIG. 3.4 Módulo fuente.

El programa ensamblador resuelve el problema de las referencias adelantadas en el "paso dos" una vez que ya se conoce todas las definiciones de las etiquetas.

El archivo ".LST" resultante para el ejemplo anterior es el siguiente:

0006	=	1	LED	EQU	BIT+1
0005	=	2	BIT	EQU	005

FIG. 3.5 Archivo de listado.

3. Resolución de referencias externas.

Debido a que las referencias externas se encuentran definidas en otros módulos objeto se hace necesario poner estas etiquetas en un lugar donde el ensamblador pueda identificarlas sin que genere ningún problema.

Las referencias externas se resuelven al enlazar el módulo objeto con otro que tenga la misma etiqueta pero definida como pública.

En el siguiente ejemplo se tiene definida una variable externa

```

LED EQU BIT+1
BIT EQU 005
    EXTERN PRINT
;Area de instrucciones
    MOV A,#PRINT
    END
    
```

FIG. 3.6 Módulo fuente.

El archivo ".LST" resultante es el siguiente:

0006	=	1	LED	EQU	BIT+1
0005	=	2	BIT	EQU	005
		3		EXTERN	PRINT
		4		;Area de instrucciones	
0000	7400	5		MOV	A,#PRINT
		6			
		7		END	

FIG. 3.7 Archivo de listado.

Debe notarse que el ensamblador para no generar ningún error asigna un valor predefinido de "00" a las variables externas tal como se muestra en la línea 5 para la variable "PRINT".

El módulo objeto resultante es el siguiente:

```

:020000007400BA
:00000001FF
E,PRINT
M(1)2+,PRINT
    
```

Localización de la variable
PRINT

FIG. 3.8 Módulo objeto.

Como se puede ver el registro de modificación "M" indica la posición donde se encuentra la variable externa dentro del registro de datos, el número de caracteres que ocupa , la operación que se debe realizar en el instante de la modificación y el nombre de la variable.

Al enlazar este módulo objeto con otro que tenga a la misma variable "PRINT" pero definida como pública quedará resuelto el problema de las referencias externas.

El siguiente módulo objeto cumple con la condición antes mencionada.

```

:06000900E509E508E50D21
:05001800658040E51B9E
:00000001FF
D,PRINT,20
    
```

FIG. 3.9 Módulo objeto con etiqueta pública.

Como se puede observar la variable "PRINT" se encuentra definida en el registro "D" del módulo objeto.

El programa ejecutable ".HEX" resultante de enlazar los programas objeto de la figura 3.8 y 3.9 es el siguiente:

```

:0200000074206A
:06000C00E509E508E50D1E
:05001800658040E51B9B
:00000001FF
    
```

Referencia externa
resuelta

FIG. 3.10 Archivo ".HEX".

4. Cálculo de expresiones en la definición de etiquetas y operandos.

En la siguiente figura se presentan algunas expresiones matemáticas que el programa ensamblador puede resolver. No importa la ubicación, ya que las expresiones pueden estar en el área de definición de etiquetas o en los operandos de las instrucciones.

```

bloqnum equ 10h
klm equ 30h
supr equ klm+1
repag equ (supr+klm)*2
;Area de instrucciones

cjne a,bloqnum-1,klm+2
mov bloqnum,a
mov a,bloqnum-klm
mov a,(bloqnum-(klm+supr)*repag)
    
```

FIG. 3.11 Módulo fuente.

El archivo ".LST" con los resultados de las expresiones es el siguiente:

```

0010 = 1 bloqnum equ 10h
0030 = 2 klm equ 30h
0031 = 3 supr equ klm+1
00C2 = 4 repag equ (supr+klm)*2
      5 ;Area de instrucciones
      6
0000 B50F2F 7 cjne a,bloqnum-1,klm+2
0003 F510 8 mov bloqnum,a
0005 E5E0 9 mov a,bloqnum-klm
0007 E58E 10 mov a,(bloqnum-(klm+supr)*repag)
    
```

FIG. 3.12 Archivo de listado.

En el caso de que exista una etiqueta repetida dentro de un mismo procedimiento el programa emite un mensaje de error al momento de escribir el archivo de listado.

En el ejemplo siguiente se encuentra repetida la etiqueta BIT por dos ocasiones.

```

LED EQU BIT+1
BIT EQU 005
EXTERN PRINT
;Area de instrucciones
BIT MOV A,#PRINT
END
    
```

FIG. 3.13 Módulo fuente.

El programa emite un mensaje de error a partir de la segunda etiqueta repetida.

```

BIT+1 =          1      LED    EQU    BIT+1
0005 =          2      BIT     EQU    005
          3          EXTERN  PRINT
          4          ;Area de instrucciones
0000 7400        5      BIT     MOV    A,#PRINT
Error:Etiqueta 'BIT' repetida en línea # 5
          6
          7          END
    
```

FIG. 3.14 Archivo de listado.

3.1.2 Definición y expansión de macro instrucciones

Para expandir una macro primero se debe indicar al programa enlazador de donde se debe obtener los datos de la macro. Una vez definida la localización, se puede invocar a las macro instrucciones desde cualquier parte del programa como si se tratase de una instrucción.

Para mostrar los resultados de definición e invocación de las macro instrucciones se tomará como ejemplo el siguiente archivo de texto el cual almacena a las macro instrucciones May, lucky y Prueba.

```

May      macro    &1,&2,&3
          mov     a,&1
          dec    a
          db     "hola tania",120
          endm

lucky    macro    &1,&2,&3
          dw     100h,20h
          endm

Prueba   macro    a,b
          mov    b,#40h
          dec   a
          inc   40h
          mov   a,40h
          endm
    
```

FIG.3.16 Archivo de texto.

Este archivo de texto se encuentra ubicado en la dirección "C:\ejemplos \macros".

A continuación se verá la forma en que se define una macro, su invocación y el resultado de la expansión de la macro.

Módulo fuente

```

;Area de instrucciones
include c:\ejemplos\macros.txt,prueba
;Area de definición de etiquetas
repag equ 10111b
avpag equ 10h
;Area de instrucciones
mov a,#repage
prueba repag,avpag
end
    
```

Definición de la macro

Invocación de la macro

FIG. 3.17 Módulo fuente.

El archivo con las macro instrucciones expandidas (se almacena en la misma dirección del módulo fuente y con el mismo nombre pero con extensión ".MCR") es el siguiente:

```

;Area de instrucciones
: include c:\ejemplos\macros.txt,prueba
;Area de definición de etiquetas
repag equ 10111b
avpag equ 10h
;Area de instrucciones
mov a,#repage
MOV avpag,#40h
DEC REPAG
INC 40h
MOV REPAG,40h
end
    
```

Macro expandida

FIG. 3.18 Archivo ".MCR".

Los pasos 1 y 2 tomarán este archivo para completar el ensamblado.

3.1.3 Enlace de programas objeto.

En el siguiente módulo fuente se especifica al programa ensamblador que se desea realizar un enlace entre el módulo objeto que se va a generar y un módulo denominado "externa.obj". Para lograr esto se emplea la pseudo instrucción "LINK" que permite identificar la localización del módulo objeto "externa.obj".

```

:Area de comandos
link      c:\ensamblador2\obj_lib.externa
:Area de definición de etiquetas
public   print
print    equ 20h
:Area de instrucciones
mov      R0,40h
mov      a,print
dec      a
end

```

Ubicación del programa objeto

FIG. 3.19 Módulo fuente.

El módulo objeto generado para este programa es el siguiente:

```

:05000000A840E52014FA
:00000001FF
D,PRINT,20

```

FIG. 3.20 Módulo objeto.

El módulo objeto denominado "externa", con el cual se va a enlazar, es el siguiente:

```

:0200000074008A
:00000001FF
E,PRINT
M,1,2,+,PRINT

```

FIG. 3.21 Módulo objeto "externa".

Una vez ejecutado el programa enlazador el archivo ejecutable resultante es el siguiente:

```

:05000000A840E52014FA
:0200060074008A
:00000001FF

```

FIG. 3.22 Archivo ".HEX".

3.2 PRUEBAS CON EL ENSAMBLADOR DE PASOS.

El programa ensamblador está compuesto por varios subprogramas tal como el analizador de macro instrucciones, el paso1, el paso2 y el enlazador. Cada uno de estos subprogramas entrega un resultado diferente.

El ensamblador de pasos lo que hace es presentar estos resultados en distintas pantallas al usuario.

Para ver su funcionamiento se tomará como ejemplo el siguiente programa:

```

;Area de comandos
    link    c:\ensamblador2\obj_lib,externa.obj
    include c:\ejemplos\macros.txt,prueba
;Area de definición de etiquetas
bloqnum    equ    supr
klm        equ    15
supr       equ    10h
repag      equ    avpag+1
avpag      equ    10101b
;Area de instrucciones
prueba    bloqnum,klm
mov       a,bloqnum
org       2h
mov       @r0,a
ajmp     $+2+128
clr       p0.3
setb     p0.3
clr       20h.2
setb     20h.2
setb     21h.2
setb     23h.0
vector    db     "micro",art,11fe,art+10h,100h,art+11fe
Mayr     dw     "marc",art,11fe,art+10h,100h,art+11fe
endproc
end
    
```

FIG. 3.23 Módulo fuente.

Al finalizar la ejecución del programa analizador de macro instrucciones se obtendrá en la pantalla principal del programa el siguiente resultado:

```

;Area de comandos
;
;    link    c:\ensamblador2\obj_lib,externa.obj
;    include c:\ejemplos\macros.txt,prueba
;Area de definición de etiquetas
bloqnum    equ    supr
klm        equ    15
supr       equ    10h
repag      equ    avpag+1
avpag      equ    10101b
;Area de instrucciones
MOV       bloqnum,#40h
DEC       KLM
INC       40h
MOV       KLM,40h
mov       a,bloqnum
org       2h
mov       @r0,a
ajmp     $+2+128
clr       p0.3
setb     p0.3
clr       20h.2
setb     20h.2
setb     21h.2
setb     23h.0
vector    db     "micro",art,11fe,art+10h,100h,art+11fe
Mayr     dw     "marc",art,11fe,art+10h,100h,art+11fe
endproc
end
    
```

← Macro expandida

FIG. 3.24 Archivo ".MCR".

Como se puede observar, en la figura 3.24 este paso únicamente expande las invocaciones a macro en un archivo de texto con extensión “.MCR”.

El siguiente proceso corresponde al “PASO1” el cual tiene la misión de recolectar todos los datos que definan a etiquetas o expresiones utilizadas en el programa. Es por este motivo que el paso 1 simplemente deja algunas etiquetas enunciadas (en especial las que tienen referencias adelantadas) para que el paso 2 sea el que las defina.

A continuación se muestra el resultado de este paso:

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
;Area de comandos
link c:\ensamblador2\obj_11b,externa.obj
include c:\ejemplos\macros.txt,prueba
;Area de definición de etiquetas
bloqnum equ supr
k1m equ 15
supr equ 10h
repag equ avpag+1
avpag equ 10101b
;Area de instrucciones
MOV bloqnum,#40h
DEC KLM
INC 40h
MOV KLM,40h
mov a,bloqnum
org 2h
mov @r0,a
ajmp $+2+128
clr p0.3
setb p0.3
clr 20h.2
setb 20h.2
setb 21h.2
setb 23h.0
vector db "micro",art,11fe,art+10h,100h,art+11fe
Mayr dw "marc",art,11fe,art+10h,100h,art+11fe
endproc
end
SUPR =
000F =
0010 =
AVPAG+1 =
0015 =
0000 75UM40
0003 150F
0005 0540
0007 850F40
000A ES@LOQNUM
0002 F6
0003 0185
0005 C283
0007 0283
0009 C202
000B 0202
000D 020A
000F 0218
0011 6D 69 63 72
0015 6F 00 00 00
0019 00 00
001B 6D61 0000
001F 0000 0000
0023 0100 0000

```

FIG. 3.25 Archivo de listado preliminar.

En la figura anterior las etiquetas que tienen referencias adelantadas se encuentran marcadas dentro de un óvalo. Estas etiquetas serán definidas por el “PASO2”.

El siguiente proceso corresponde al “PASO2” y es el encargado de presentar el archivo de listado definitivo con todas las expresiones matemáticas resueltas y las etiquetas, con referencias adelantadas, definidas.

A continuación se muestra el archivo ".LST" resultante luego de ejecutarse este proceso.

```

1          ;Area de comandos
2          ;          link    c:\ensamblador2\obj_lib,externa.obj
3          ;          include c:\ejemplos\macros.txt,prueba
4          ;Area de definición de etiquetas
0010      =          5          bloqnum equ    supr
000F      =          6          klm   equ    15
0010      =          7          supr  equ    10h
0016      =          8          repag equ    avpag+1
0015      =          9          avpag equ    10101b
0000      751040     10         ;Area de instrucciones
0003      150F      11         MOV    bloqnum,#40h
0005      0540      12         DEC    KLM
0007      850F40    13         INC    40h
000A      E910      14         MOV    KLM,40h
0002      F6        15         mov   a,bloqnum
0003      0185      16         org   2h
0005      C283      17         mov   @r0,a
0007      D283      18         ajmp  $+2+128
0009      C202      19         clr   p0.3
000B      D202      20         setb p0.3
000D      D20A      21         clr   20h.2
000F      D218      22         setb 20h.2
0011      6D 69 63 72 23         setb 21h.2
0015      6F 00 00 00 24         setb 23h.0
0019      00 00     25         vector db    "micro",art,life,art+10h,100h,art+life
001B      6D61 0000 26         Mayr dw    "marc",art,life,art+10h,100h,art+life
001F      0000 0000
0023      0100 0000
27         endproc
28         end
    
```

FIG. 3.26. Archivo de listado definitivo.

Este paso ya genera un módulo objeto y se lo presenta en la siguiente figura:

```

:0C000000750F4015100540851040E510FC
:10000200F60185C283D283C202D20AD2186D0D
:100012006963726F00000000006D6100000000063
:05002200000100000008
:00000001FF
    
```

FIG. 3.27 Módulo objeto.

El último proceso que se realiza para obtener el programa ejecutable ".HEX" es el "ENLAZADO".

Debido a que en el módulo fuente existe la sentencia:

```
link    c:\ensamblador2\obj_lib,externa.obj
```

el módulo objeto de la figura 3.27 se enlazará con el programa "externa.obj" que se indica a continuación:

```
:0200000074008A
:00000001FF
E.PRINT
M.1.2+.PRINT
```

FIG. 3.28 Módulo objeto "externa.obj".

Debido a que el programa de la figura 3.27 no contiene ninguna variable pública denominada "PRINT". El programa enlazador lo reemplazará por el valor "00".

El resultado de este enlace es el siguiente archivo ejecutable con extensión ".HEX":

```
:0C000000750F4015100540851040E510FC
:10000200F60185C283D283C202D202D20AD2186D0D
:100012006963726F00000000006D6100000000063
:05002200000100000008
:02002800740062
:00000001FF
```

FIG. 3.29 Archivo ".HEX".

3.3 COMPARACIÓN DE LOS RESULTADOS CON RESPECTO A OTROS ENSAMBLADORES.

Para realizar esta comparación se va a tomar un programa de ejemplo y se lo procesará con los ensambladores "AVMAC51", "CYS-8051" y con el ensamblador que se ha diseñado en este proyecto.

El módulo fuente que se va a tomar de ejemplo es el siguiente:

```
;Area de comandos
;Area de definición de etiquetas
bloqnum equ Ofah
k1m equ 15
supr equ 10h
avpag equ 10101b
repag equ avpag+1
lcd equ 0a000h
;Area de instrucciones
MOV k1m,#40h
DEC BLOQNUM
INC 40h
MOV BLOQNUM,40h
mov a,bloqnum
mov @r0,a
ajmp $+2+128
clr p0.3
setb p0.3
clr 20h.2
setb 20h.2
setb 21h.2
setb 23h.0
mov a,#($+k1m)*avpag
movx @dptr,a
inc a
mov dptr,#lcd+2
movx @dptr,a
inc a
mov dptr,#lcd+3
sjmp $
```

FIG. 3.30 Módulo fuente.

3.3.1 Resultados del proceso de ensamblado con el programa AVMAC51.

Al procesar el programa de la figura 3.30 con el ensamblador "AVMAC51" se obtienen los siguientes resultados.

Archivo de listado ".LST":

```

1          ;Area de comandos
2          ;Area de definicisn de etiquetas
3  =00FA   bloqnum equ      Ofah
4  =000F   k1m             equ      15
5  =0010   supr           equ      10h
6  =0015   avpag         equ      10101b
7  =0016   repag         equ      avpag+1
8  =A000   lcd           equ      0a000h
9          ;Area de instrucciones
10 0000' 75 OF 40        MOV      k1m,#40h
11 0003' 15 FA          DEC      BLOQNUM
12 0005' 05 40         INC      40h
13 0007' 85 40 FA      MOV      BLOQNUM,40h
14 000A' E5 FA          mov     a,bloqnum
15 000C' F6            mov     @r0,a
16 000D' ...X          ajmp    $+2+128
17 000F' C2 83         clr     p0.3
18 0011' D2 83         setb   p0.3
19 0013' C2 02         clr     20h.2
20 0015' D2 02         setb   20h.2
21 0017' D2 0A         setb   21h.2
22 0019' D2 18         setb   23h.0
23 001B' 74 ..X       mov     a,#($+k1m)*avpag
24 001D' F0            movx   @dptr,a
25 001E' 04            inc    a
26 001F' 90 A002       mov    dptr,#lcd+2
27 0022' F0            movx   @dptr,a
28 0023' 04            inc    a
29 0024' 90 A003       mov    dptr,#lcd+3
30 0027' 80 FE        sjmp   $
31          end
    
```

FIG. 3.31 Archivo de listado.

Módulo objeto ".OBJ":

```

* r^llinke2 1.12 4/20/04 04:42:30 -w * Avocet 8051 Assembler v1.12, #01027
( ( M CODE ) C DATA D BIT B XDATA X- & W Operand out of range. r Bit
address not in legal area. Inpage Jump goes out of page. % - r * 3CDA 8 8C 8 8:JAA * | 7 à 89C-Æ W1
P u @ @ | @...@úúúú r
|
Rr W1 +
| • y89- || ÅfòfÅ, ò, ó
ò t
| 5- œø | , ø | L ep, ò |
    
```

FIG. 3.32 Módulo objeto generado por el ensamblador AVMAC51.

Como se puede ver el módulo objeto producido por el programa AVMAC51 no es un archivo diseñado para que el usuario pueda entenderlo.

Este módulo objeto necesita ser procesado por otro programa enlazador denominado "AVLINK" para obtener el archivo ejecutable final.

Archivo Ejecutable ".HEX":

```
:20000000750F4015FA05408540FAE5FAF6018FC283D283C20202020AD2187472F0049042
:09002000A002F00490A00380FE90
:00000001FF
```

FIG. 3.33 Archivo ".HEX".

Este archivo ejecutable puede admitir en cada registro de datos hasta un total de 75 caracteres.

3.3.2 Resultados del proceso de ensamblado con el programa CYS-8051.

El archivo de listado producido por este programa al ensamblar la secuencia de instrucciones de la figura 3.30 es el siguiente:

Archivo de listado ".LST":

```
The Cybernetic Micro Systems 8051 Family Assembler, Version 3.02 Page 1
04-20-04
; Area de comandos
; Area de definicion de etiquetas
00FA =      bloqnum equ      Ofah
000F =      klm      equ      15
0010 =      supr      equ      10h
0015 =      avpag      equ      10101b
0016 =      repag      equ      avpag+1
A000 =      lcd      equ      0a000h
; Area de instrucciones
0000 750F40      MOV      klm,#40h
0003 15FA      DEC      BLOQNUM
0006 0540      INC      4ch
0007 8540FA      MOV      BLOQNUM,4ch
000A E5FA      mov      a,bloqnum
000C F6      mov      r0,a
000D 018F      ajmp     $+2+126
000F C263      clr      p0.3
0011 0263      setb    p0.3
0013 C220      clr      2ch.2
0015 0220      setb    2ch.2
0017 0221      setb    21h.2
0019 0223      setb    25h.0
0018 7472      mov      a,#($+klm)*avpag
001D F0      movx    @dptr,a
001E 04      inc     a
001F 90A002      mov     dptr,#lcd+2
0022 F0      movx    @dptr,a
0023 04      inc     a
0024 90A003      mov     dptr,#lcd+3
0027 80FE      sjmp   $
0000      end
The Cybernetic Micro Systems 8051 Family Assembler, Version 3.02 Page 2
04-20-04
;XT Symbol Name Type Value
AVPAG . . . . . I 0015
BLOQNUM . . . . . I 00FA
KLM . . . . . I 000F
LCD . . . . . I A000
REPAG . . . . . I 0016
SUPR . . . . . I 0010
;XZ
00 Errors (0000)
-----0-----
```

FIG. 3.34 Archivo .LST generado por el ensamblador CYS-8051.

Módulo objeto ".OBJ":

Este programa no genera ningún módulo objeto.

Archivo ejecutable ".HEX":

```
:10000000750F4015FA05408540FAE5FAF6018FC2F2
:1000100083D283C220D220D221D2237472F00490E2
:09002000A002F00490A00380FE90
```

FIG. 3.35 Archivo ".HEX".

Como se puede observar de la figura 3.35, este archivo contiene hasta un máximo de 43 caracteres por registro.

El archivo ejecutable producido por el programa CYS-8051 no presenta el registro de fin.

3.3.3 Resultados del proceso de ensamblado con el programa desarrollado en este proyecto.

El siguiente es el resultado del proceso de ensamblado del módulo fuente de la figura 3.30 usando el programa diseñado en este trabajo.

Archivo de listado ".LST":

Lenguaje de máquina	#Línea	Lenguaje ensamblador
	1	;Area de comandos
	2	;Area de definición de etiquetas
00FA =	3	bloqnum equ Ofah
000F =	4	klm equ 15
0010 =	5	supr equ 10h
0015 =	6	avpag equ 10101b
0016 =	7	repag equ avpag+1
A000 =	8	lcd equ 0a000h
	9	;Area de instrucciones
0000 750F40	10	MOV klm,#40h
0003 15FA	11	DEC BLOQNUM
0005 0540	12	INC 40h
0007 85FA40	13	MOV BLOQNUM,40h
000A E5FA	14	mov a,bloqnum
000C F6	15	mov @r0,a
000D 018F	16	ajmp \$+2+128
000F C283	17	clr p0.3
0011 D283	18	setb p0.3
0013 C202	19	clr 20h.2
0015 D202	20	setb 20h.2
0017 D20A	21	setb 21h.2
0019 D218	22	setb 23h.0
001B 7472	23	mov a,#(\$+klm)*avpag
001D F0	24	movx @dptr,a
001E 04	25	inc a
001F 90A002	26	mov dptr,#lcd+2
0022 F0	27	movx @dptr,a
0023 04	28	inc a
0024 90A003	29	mov dptr,#lcd+3
0027 80FE	30	sjmp \$
	31	end

FIG. 3.36 Archivo de listado.

Módulo objeto ".OBJ":

```
|:10000000750F4015FA054085FA40E5FAF6018FC2F2
|:1000100083D283C202D202D20AD2187472F0049040
|:09002000A002F00490A00380FE90
|:00000001FF
```

FIG. 3.37 Módulo objeto.

Archivo ejecutable ".HEX":

```
|:10000000750F4015FA054085FA40E5FAF6018FC2F2
|:1000100083D283C202D202D20AD2187472F0049040
|:09002000A002F00490A00380FE90
|:00000001FF
```

FIG. 3.38 Archivo ".HEX".

El módulo objeto y el archivo ejecutable generado por el programa ensamblador son iguales debido a que no se han definido variables públicas ni externas en el módulo fuente.

3.4 SEMEJANZAS Y DIFERENCIAS CON OTROS PROGRAMAS ENSAMBLADORES.

3.4.1 Semejanzas.

- El programa ensamblador diseñado en este trabajo acepta la mayoría de las pseudo instrucciones que utilizan tanto el programa AVMAC51 como el programa CYS-8051. A excepción de las pseudo instrucciones DEFSEG y SEG del programa AVMAC51. En el software diseñado, estas funciones fueron reemplazadas por un mejor procesamiento de la pseudo instrucción "ORG" por parte del programa ensamblador.
- Acepta todo el repertorio de instrucciones del microcontrolador 8051/52

3.4.2 Diferencias

3.4.2.1 Diferencias con el programa AVMAC51

- El programa ensamblador diseñado presenta un interfaz programa-usuario mucho más amigable.

- Los archivos que se desean enlazar pueden ser especificados dentro del módulo fuente usando la seudo instrucción "LINK" o usando las ventanas que se encuentran en la interfaz programa-usuario y que fueron especialmente diseñadas para el efecto. Al contrario del programa ensamblador AVMAC51, que para introducir los archivos objeto a enlazar se debe colocar instrucciones en la línea de comandos en ambiente DOS del programa AVLINK.
- El programa AVMAC51 no reconoce referencias adelantadas.
- Al ensamblar el programa de la figura 3.30 el ensamblador AVMAC51 obtuvo el módulo .OBJ en 850 ms, en tanto que el programa diseñado en este proyecto lo obtuvo en 980ms.

3.4.2.2 Diferencias con el programa CYS-8051

- El programa ensamblador diseñado tiene un mayor repertorio de seudo instrucciones.
- El programa CYS-8051 no genera archivos objeto.
- El programa CYS-8051 no permite realizar enlaces con otros archivos OBJ.
- El programa ensamblador diseñado en este trabajo presenta un interfaz programa-usuario más amigable.
- El programa CYS-8051 no reconoce las seudo instrucciones que sirven para introducir procedimientos dentro del programa principal, razón por la cual no se pueden definir variables locales.
- Al ensamblar el programa de la figura 3.30 el ensamblador CYS-8051 obtuvo el módulo .EXE en 798 ms.

CAPITULO 4

4. CONCLUSIONES Y RECOMENDACIONES

En el presente proyecto se diseñó un programa ensamblador, mediante el uso del lenguaje de programación Visual Basic, para el Laboratorio de Sistemas Microprocesados. Durante el desarrollo de este proyecto se llegó a las siguientes conclusiones:

- El tratar de adaptar un programa desarrollado en entorno DOS dentro de otro programa creado para ambiente Windows, puede ser una buena alternativa, tal como se lo hizo al incorporar el programa ensamblador AVMAC-51 dentro del SIDES2000, pero la desventaja radica en que las interfaces programa-usuario no son muy amigables y para utilizar estos programas creados en DOS, se necesita insertar algunos comandos dentro del código fuente del programa que lo va a alojar, razón por la cual esta adaptación no es muy fácil de realizar.
- El programa diseñado en este proyecto facilita el proceso de ensamblado para el usuario, ya que se lo realiza de una manera gráfica y didáctica, sin la necesidad de escribir ningún tipo de comando, tal como se lo hace en los programas ensambladores creados para DOS.
- Al realizar una comparación de los ensambladores creados para DOS y el programa desarrollado en este proyecto se llega a la conclusión de que este último tiene muchas ventajas sobre los otros ensambladores, es decir, se ganó mucho en cuanto a versatilidad. Pero al analizar los tiempos que se demoran en obtener los archivos objeto, se observó que los ensambladores para DOS son ligeramente más rápidos que el programa desarrollado en este proyecto.
- Con el diseño de este software, se dispone de una mayor flexibilidad para realizar cambios o mejoras en la interfaz del ensamblador o para ampliar el

número de marcas de microcontroladores que el programa puede admitir, ya que para realizar estas adecuaciones se necesita hacer muy pocos cambios en el código fuente.

- El programa ensamblador presentado en este proyecto está inicialmente desarrollado para trabajar con los microcontroladores INTEL de la familia 8051/52 pero puede ser adaptado para que funcione con otras marcas de microcontroladores. Para lograr esta compatibilidad se deberá agregar en la base de datos "ensamblador .mdb" una tabla de códigos de instrucción y una tabla que contenga las etiquetas predefinidas propias del microcontrolador. Para el caso de la tabla de instrucciones, TABOP, se debe indicar su dirección al control de datos 1 (data1) a través de la propiedad "DatabaseName". De igual manera para la tabla de símbolos TABSIM, se deberá indicar su dirección al control de datos 2 (data2).
- Es importante que al indicar la dirección de un archivo de texto que contenga macro instrucciones, se especifique su extensión ya que existen archivos de texto con diferentes extensiones tales como: .TXT, .DOC, .ASM, .RTF, etc.
- Se recomienda usar etiquetas que no empiecen con números, debido a que pueden originar conflictos con el analizador de macro instrucciones una vez que este realiza la expansión.
- El módulo fuente puede ser escrito con letras mayúsculas o minúsculas, esto no tiene importancia ya que el ensamblador transforma a todas las letras a mayúsculas antes de empezar la traducción, con excepción de los mensajes de texto que sean introducidos entre comillas en las pseudo instrucciones DB y DW.

BIBLIOGRAFÍA

- Andrew S. Tanenbaum.
Organización de computadoras: un enfoque estructurado.
Cuarta edición, PRENTICE HALL, México, 2000.
- Leland I. Beck.
Software de Sistemas.
Primera edición, ADDISON WESLEY Iberoamericana, México, 1988.
- Pedro Miguel Anasagasti.
Fundamentos de los computadores.
Octava edición, Paraninfo, Madrid, 2001.
- MCS 51 MICROCONTROLLER FAMILY USER'S MANUAL.
INTEL, USA, febrero 1994.
- AVOCET SYSTEMS, INC.
AVMAC 8051 User's Manual
Primera edición, AVOCET SYSTEMS, INC, USA, mayo 1986

ANEXO A.

MENSAJES DE ERROR.

Instrucción:

1. "Error: " + instrucción + " es una Instrucción o Pseudo instrucción desconocida en línea#" + Str(n_línea).
2. "Error: No existe operando del tipo "" + oper_std + "" para instrucción " + instrucción + " en línea#" + Str(n_línea).
3. "Error: No existe operando del tipo ' ' + oper_std + ' ' para instrucción " + instrucción + " en línea#" + Str(n_línea).
4. "Error: " + instrucción + " es una Instrucción o Pseudo instrucción es desconocida en línea#" + Str(n_línea).

Para etiquetas:

5. "Error:Etiqueta repetida ".
6. "Error:etiqueta " & resultA & " no se encuentra definida en línea# " & n_línea.
7. "Error en línea#" + Str(n_línea) + ":" + resultA + " no se encuentra definido".
8. "Error:No existe etiqueta ni instrucción".
9. "Error: Etiqueta " + result3 + " no se encuentra definida".

Procedimientos:

10. "Error: PROC sin etiqueta".
11. "No se encuentra definida la etiqueta: " + resultA + " en el PROC# " + Str(cont_proc).
12. "Error:Etiqueta repetida en Proc#" + Str(cont_proc).
13. "Error:Etiqueta "" + etiqueta + "" repetida en línea #" + Str(n_línea).
14. "Error:No existe etiqueta ni instruccion".

Manejo de bits:

15. "Error en línea#" + Str(n_línea) + "bit fuera de rango".

16. "Error en línea#" + Str(n_línea) + ":bit direccionable fuera de rango".

Macro instrucciones:

17. "no coincide el número de argumentos con la macro en línea #" + Str(n_linea).

18. "Error en línea#: " + Str(n_linea) + " ,No se encontró ninguna macro con el nombre: " + instruccion + " en la direccion" + LOCALIZACION.

ANEXO B.

ETIQUETAS PREDEFINIDAS EN TABLA DE DATOS TABSIM.

TABSIM

Id	ETIQ	DE FI	EXP _VA L	LOCALIZA CION	N_PROC	PUBLIC_EXT ERN	MODI FICA CION	LON GITU D	BANDE RA_M
0	noexiste	&0	14			Coexiste			
1	ACC	&0	E0		*				
2	B	&0	F0		*				
3	DPH	&0	83		*				
4	DPL	&0	82		*				
5	IE	&0	A8		*				
6	IP	&0	B8		*				
7	P0	&0	80		*				
8	P1	&0	90		*				
9	P2	&0	A0		*				
10	P3	&0	B0		*				
11	PSW	&0	D0		*				
12	SBUF	&0	99		*				
13	SCON	&0	98		*				
14	SP	&0	81		*				
15	PCON	&0	87		*				
16	TCON	&0	88		*				
17	TMOD	&0	89		*				
18	TH0	&0	8C		*				
19	TH1	&0	8D		*				
20	TL0	&0	8A		*				
21	TL1	&0	8B		*				
22	IEN0	&0	A8		*				
23	IP0	&0	A9		*				
24	IEN1	&0	B8		*				
25	IP1	&0	B9		*				
26	IRCON	&0	C0		*				
27	CCEN	&0	C1		*				
28	CCL1	&0	C2		*				
29	CCH1	&0	C3		*				

TABSIM

Id	ETIQ	DE FI	EXP _VA L	LOCALIZA CION	N_PROC	PUBLIC_EXT ERN	MODI FICA CION	LON GITU D	BANDE RA_M
30	CCL2	&0	C4		*				
31	CCH2	&0	C5		*				
32	CCL3	&0	C6		*				
33	CCH3	&0	C7		*				
34	T2CON	&0	C8		*				
35	CRCL	&0	CA		*				
36	CRCH	&0	CB		*				
37	RCAP2L	&0	CA		*				
38	RCAP2H	&0	CB		*				
39	TL2	&0	CC		*				
40	TH2	&0	CD		*				
41	ADCON	&0	D8		*				
42	ADDAT	&0	D9		*				
43	DAPR	&0	DA		*				
44	P5	&0	F8		*				
45	PAGE	&0	0800		*				
46	RESET	&0	00		*				
47	EXTI0	&0	03		*				
48	TIMER0	&0	0B		*				
49	EXTI1	&0	13		*				
50	TIMER1	&0	1B		*				
51	SINT	&0	23		*				
52	CY	&0	D7		*				
53	AC	&0	D6		*				
54	F0	&0	D5		*				
55	RS1	&0	D4		*				
56	RS0	&0	D3		*				
57	OV	&0	D2		*				
58	P	&0	D0		*				
59	PSW.7	&0	D7		*				
60	PSW.6	&0	D6		*				
61	PSW.5	&0	D5		*				
62	PSW.4	&0	D4		*				

TABSIM

Id	ETIQ	DE FI	EXP _VA L	LOCALIZA CION	N_PROC	PUBLIC_EXT ERN	MODI FICA CION	LON GITU D	BANDE RA_M
63	PSW.3	&0	D3		*				
64	PSW.2	&0	D2		*				
65	PSW.0	&0	D0		*				
66	TF1	&0	8F		*				
67	TR1	&0	8E		*				
68	TF0	&0	8D		*				
69	TR0	&0	8C		*				
70	IE1	&0	8B		*				
71	IT1	&0	8A		*				
72	IE0	&0	89		*				
73	IT0	&0	88		*				
74	TCON.7	&0	8F		*				
75	TCON.6	&0	8E		*				
76	TCON.5	&0	8D		*				
77	TCON.4	&0	8C		*				
78	TCON.3	&0	8B		*				
79	TCON.2	&0	8A		*				
80	TCON.1	&0	89		*				
81	TCON.0	&0	88		*				
82	SM0	&0	9F		*				
83	SM1	&0	9E		*				
84	SM2	&0	9D		*				
85	REN	&0	9C		*				
86	TB8	&0	9B		*				
87	RB8	&0	9A		*				
88	TI	&0	99		*				
89	RI	&0	98		*				
90	SCON.7	&0	9F		*				
91	SCON.6	&0	9E		*				
92	SCON.5	&0	9D		*				
93	SCON.4	&0	9C		*				
94	SCON.3	&0	9B		*				
95	SCON.2	&0	9A		*				

TABSIM

Id	ETIQ	DE FI	EXP _VA L	LOCALIZA CION	N_PROC	PUBLIC_EXT ERN	MODI FICA CION	LON GITU D	BANDE RA_M
96	SCON.1	&0	99		*				
97	SCON.0	&0	98		*				
98	EA	&0	AF		*				
99	ES	&0	AC		*				
100	ET1	&0	AB		*				
101	EX1	&0	AA		*				
102	ET0	&0	A9		*				
103	EX0	&0	A8		*				
104	ET2	&0	AD		*				
105	IE.7	&0	AF		*				
106	IE.5	&0	AD		*				
107	IE.4	&0	AC		*				
108	IE.3	&0	AB		*				
109	IE.2	&0	AA		*				
110	IE.1	&0	A9		*				
111	IE.0	&0	A8		*				
112	RD	&0	B7		*				
113	WR	&0	B6		*				
114	T1	&0	B5		*				
115	T0	&0	B4		*				
116	INT1	&0	B3		*				
117	INT0	&0	B2		*				
118	TXD	&0	B1		*				
119	RXD	&0	B0		*				
120	P3.7	&0	B7		*				
121	P3.6	&0	B6		*				
122	P3.5	&0	B5		*				
123	P3.4	&0	B4		*				
124	P3.3	&0	B3		*				
125	P3.2	&0	B2		*				
126	P3.1	&0	B1		*				
127	P3.0	&0	B0		*				
128	PS	&0	BC		*				

TABSIM

Id	ETIQ	DE FI	EXP _VA L	LOCALIZA CION	N_PROC	PUBLIC_EXT ERN	MODI FICA CION	LON GITU D	BANDE RA_M
129	PT1	&0	BB		*				
130	PX1	&0	BA		*				
131	PT0	&0	B9		*				
132	PX0	&0	B8		*				
133	PT2	&0	BD		*				
134	IP.5	&0	BD		*				
135	IP.4	&0	BC		*				
136	IP.3	&0	BB		*				
137	IP.2	&0	BA		*				
138	IP.1	&0	B9		*				
139	IP.0	&0	B8		*				
140	P0.7	&0	87		*				
141	P0.6	&0	86		*				
142	P0.5	&0	85		*				
143	P0.4	&0	84		*				
144	P0.3	&0	83		*				
145	P0.2	&0	82		*				
146	P0.1	&0	81		*				
147	P0.0	&0	80		*				
148	P1.7	&0	97		*				
149	P1.6	&0	96		*				
150	P1.5	&0	95		*				
151	P1.4	&0	94		*				
152	P1.3	&0	93		*				
153	P1.2	&0	92		*				
154	P1.1	&0	91		*				
155	P1.0	&0	90		*				
156	P2.7	&0	A7		*				
157	P2.6	&0	A6		*				
158	P2.5	&0	A5		*				
159	P2.4	&0	A4		*				
160	P2.3	&0	A3		*				
161	P2.2	&0	A2		*				

TABSIM

Id	ETIQ	DE FI	EXP _VA L	LOCALIZA CION	N_PROC	PUBLIC_EXT ERN	MODI FICA CION	LON GITU D	BANDE RA_M
162	P2.1	&0	A1		*				
163	P2.0	&0	A0		*				
164	T2EX	&0	91		*				
165	T2	&0	90		*				
166	TF2	&0	CF		*				
167	EXF2	&0	CE		*				
168	RCLK	&0	CD		*				
169	TCLK	&0	CC		*				
170	EXEN	&0	CB		*				
171	TR2	&0	CA		*				
172	C/T2	&0	C9		*				
173	CP/RL2	&0	C8		*				
174	TIMER2	&0	2BH		*				
175	Etiqueta	#	Valo	Direccion	Nro_procedim	Publica	o		
		de r			iento	Externa			
		ele							
		me							
		nto							
		s							
		ind							
		efin							
		ido							
		s							

ANEXO C.

INSTRUCCIONES DEL MICROCONTROLADOR 8051\52.

TABOP

Id	CODI	LONG	A_data	REL_ABS_ADDRESS	MNEM	OPER
1						.noexiste
2	00	1	0		NOP	
3	01	2	1	A	AJMP	Data
4	02	3	1	A	LJMP	Data
5	03	1	0	A	RR	A
6	04	1	0	A	INC	A
7	05	2	1	A	INC	Data
8	06	1	0	A	INC	@R0
9	07	1	0	A	INC	@R1
10	08	1	0	A	INC	R0
11	09	1	0	A	INC	R1
12	0A	1	0	A	INC	R2
13	0B	1	0	A	INC	R3
14	0C	1	0	A	INC	R4
15	0D	1	0	A	INC	R5
16	0E	1	0	A	INC	R6
17	0F	1	0	A	INC	R7
18	10	3	2	R	JBC	data,data
19	11	2	1	A	ACALL	Data
20	12	3	1	A	LCALL	Data
21	13	1	0	A	RRC	A
22	14	1	0	A	DEC	A
23	15	2	1	A	DEC	Data
24	16	1	0	A	DEC	@R0
25	17	1	0	A	DEC	@R1
26	18	1	0	A	DEC	R0
27	19	1	0	A	DEC	R1
28	1A	1	0	A	DEC	R2
29	1B	1	0	A	DEC	R3
30	1C	1	0	A	DEC	R4
31	1D	1	0	A	DEC	R5
32	1E	1	0	A	DEC	R6

TABOP

Id	CODI	LONG	A_data	REL_ABS_ADDRESS	MNEM	OPER
33	1F	1	0	A	DEC	R7
34	20	3	2	R	JB	data,data
35	21	2	1	A	AJMP	Data
36	22	1	0	A	RET	
37	23	1	0	A	RL	A
38	24	2	1	A	ADD	A,#data
39	25	2	1	A	ADD	A,data
40	26	1	0	A	ADD	A,@R0
41	27	1	0	A	ADD	A,@R1
42	28	1	0	A	ADD	A,R0
43	29	1	0	A	ADD	A,R1
44	2A	1	0	A	ADD	A,R2
45	2B	1	0	A	ADD	A,R3
46	2C	1	0	A	ADD	A,R4
47	2D	1	0	A	ADD	A,R5
48	2E	1	0	A	ADD	A,R6
49	2F	1	0	A	ADD	A,R7
50	30	3	2	R	JNB	data,data
51	31	2	1	A	ACALL	Data
52	32	1	0	A	RETI	
53	33	1	0	A	RLC	A
54	34	2	1	A	ADDC	A,#data
55	35	2	1	A	ADDC	A,data
56	36	1	0	A	ADDC	A,@R0
57	37	1	0	A	ADDC	A,@R1
58	38	1	0	A	ADDC	A,R0
59	39	1	0	A	ADDC	A,R1
60	3A	1	0	A	ADDC	A,R2
61	3B	1	0	A	ADDC	A,R3
62	3C	1	0	A	ADDC	A,R4
63	3D	1	0	A	ADDC	A,R5
64	3E	1	0	A	ADDC	A,R6
65	3F	1	0	A	ADDC	A,R7
66	40	2	1	R	JC	Data
67	41	2	1	A	AJMP	Data

TABOP

Id	CODI	LONG	A_data	REL_ABS_ADDRESS	MNEM	OPER
68	42	2	1	A	ORL	data,A
69	43	3	2	A	ORL	data,#data
70	44	2	1	A	ORL	A,#data
71	45	2	1	A	ORL	A,data
72	46	1	0	A	ORL	A,@R0
73	47	1	0	A	ORL	A,@R1
74	48	1	0	A	ORL	A,R0
75	49	1	0	A	ORL	A,R1
76	4A	1	0	A	ORL	A,R2
77	4B	1	0	A	ORL	A,R3
78	4C	1	0	A	ORL	A,R4
79	4D	1	0	A	ORL	A,R5
80	4E	1	0	A	ORL	A,R6
81	4F	1	0	A	ORL	A,R7
82	50	2	1	R	JNC	Data
83	51	2	1	A	ACALL	Data
84	52	2	1	A	ANL	data,A
85	53	3	2	A	ANL	data,#data
86	54	2	1	A	ANL	A,#data
87	55	2	1	A	ANL	A,data
88	56	1	0	A	ANL	A,@R0
89	57	1	0	A	ANL	A,@R1
90	58	1	0	A	ANL	A,R0
91	59	1	0	A	ANL	A,R1
92	5A	1	0	A	ANL	A,R2
93	5B	1	0	A	ANL	A,R3
94	5C	1	0	A	ANL	A,R4
95	5D	1	0	A	ANL	A,R5
96	5E	1	0	A	ANL	A,R6
97	5F	1	0	A	ANL	A,R7
98	60	2	1	R	JZ	Data
99	61	2	1	A	AJMP	Data
100	62	2	1	A	XRL	data,A
101	63	3	2	A	XRL	data,#data
102	64	2	1	A	XRL	A,#data

TABOP

Id	CODI	LONG	A_data	REL_ABS_ADDRESS	MNEM	OPER
103	65	2	1	A	XRL	A,data
104	66	1	0	A	XRL	A,@R0
105	67	1	0	A	XRL	A,@R1
106	68	1	0	A	XRL	A,R0
107	69	1	0	A	XRL	A,R1
108	6A	1	0	A	XRL	A,R2
109	6B	1	0	A	XRL	A,R3
110	6C	1	0	A	XRL	A,R4
111	6D	1	0	A	XRL	A,R5
112	6E	1	0	A	XRL	A,R6
113	6F	1	0	A	XRL	A,R7
114	70	2	1	R	JNZ	Data
115	71	2	1	A	ACALL	Data
116	72	2	1	A	ORL	C,data
117	73	1	0	A	JMP	@A+DPTR
118	74	2	1	A	MOV	A,#data
119	75	3	2	A	MOV	data,#data
120	76	2	1	A	MOV	@R0,#data
121	77	2	1	A	MOV	@R1,#data
122	78	2	1	A	MOV	R0,#data
123	79	2	1	A	MOV	R1,#data
124	7A	2	1	A	MOV	R2,#data
125	7B	2	1	A	MOV	R3,#data
126	7C	2	1	A	MOV	R4,#data
127	7D	2	1	A	MOV	R5,#data
128	7E	2	1	A	MOV	R6,#data
129	7F	2	1	A	MOV	R7,#data
130	80	2	1	R	SJMP	data
131	81	2	1	A	AJMP	data
132	82	2	1	A	ANL	C,data
133	83	1	0	A	MOVC	A,@A+PC
134	84	1	0	A	DIV	AB
135	85	3	2	A	MOV	data,data
136	86	2	1	A	MOV	data,@R0
137	87	2	1	A	MOV	data,@R1

TABOP

Id	CODI	LONG	A_data	REL_ABS_ADDRESS	MNEM	OPER
138	88	2	1	A	MOV	data,R0
139	89	2	1	A	MOV	data,R1
140	8A	2	1	A	MOV	data,R2
141	8B	2	1	A	MOV	data,R3
142	8C	2	1	A	MOV	data,R4
143	8D	2	1	A	MOV	data,R5
144	8E	2	1	A	MOV	data,R6
145	8F	2	1	A	MOV	data,R7
146	90	3	1	A	MOV	DPTR,#data
147	91	2	1	A	ACALL	data
148	92	2	1	A	MOV	data,C
149	93	1	0	A	MOVC	A,@A+DPTR
150	94	2	1	A	SUBB	A,#data
151	95	2	1	A	SUBB	A,data
152	96	1	0	A	SUBB	A,@R0
153	97	1	0	A	SUBB	A,@R1
154	98	1	0	A	SUBB	A,R0
155	99	1	0	A	SUBB	A,R1
156	9A	1	0	A	SUBB	A,R2
157	9B	1	0	A	SUBB	A,R3
158	9C	1	0	A	SUBB	A,R4
159	9D	1	0	A	SUBB	A,R5
160	9E	1	0	A	SUBB	A,R6
161	9F	1	0	A	SUBB	A,R7
162	A0	2	1	A	ORL	C,/data
163	A1	2	1	A	AJMP	data
164	A2	2	1	A	MOV	C,data
165	A3	1	0	A	INC	DPTR
166	A4	1	0	A	MUL	AB
167	A5				reserved	
168	A6	2	1	A	MOV	@R0,data
169	A7	2	1	A	MOV	@R1,data
170	A8	2	1	A	MOV	R0,data
171	A9	2	1	A	MOV	R1,data
172	AA	2	1	A	MOV	R2,data

TABOP

Id	CODI	LONG	A_data	REL_ABS_ADDRESS	MNEM	OPER
173	AB	2	1	A	MOV	R3,data
174	AC	2	1	A	MOV	R4,data
175	AD	2	1	A	MOV	R5,data
176	AE	2	1	A	MOV	R6,data
177	AF	2	1	A	MOV	R7,data
178	B0	2	1	A	ANL	C,/data
179	B1	2	1	A	ACALL	data
180	B2	2	1	A	CPL	data
181	B3	1	0	A	CPL	C
182	B4	3	2	R	CJNE	A,#data,data
183	B5	3	2	R	CJNE	A,data,data
184	B6	3	2	R	CJNE	@R0,#data,data
185	B7	3	2	R	CJNE	@R1,#data,data
186	B8	3	2	R	CJNE	R0,#data,data
187	B9	3	2	R	CJNE	R1,#data,data
188	BA	3	2	R	CJNE	R2,#data,data
189	BB	3	2	R	CJNE	R3,#data,data
190	BC	3	2	R	CJNE	R4,#data,data
191	BD	3	2	R	CJNE	R5,#data,data
192	BE	3	2	R	CJNE	R8,#data,data
193	00	3	2	R	CJNE	R7,#data,data
194	C0	2	1	A	PUSH	data
195	C1	2	1	A	AJMP	data
196	C2	2	1	A	CLR	data
197	C3	1	0	A	CLR	C
198	C4	1	0	A	SWAP	A
199	C5	2	1	A	XCH	A,data
200	C6	1	0	A	XCH	A,@R0
201	C7	1	0	A	XCH	A,@R1
202	C8	1	0	A	XCH	A,R0
203	C9	1	0	A	XCH	A,R1
204	CA	1	0	A	XCH	A,R2
205	CB	1	0	A	XCH	A,R3
206	CC	1	0	A	XCH	A,R4
207	CD	1	0	A	XCH	A,R5

TABOP

Id	CODI	LONG	A_data	REL_ABS_ADDRESS	MNEM	OPER
208	CE	1	0	A	XCH	A,R6
209	CF	1	0	A	XCH	A,R7
210	D0	2	1	A	POP	data
211	D1	2	1	A	ACALL	data
212	D2	2	1	A	SETB	data
213	D3	1	0	A	SETB	C
214	D4	1	0	A	DA	A
215	D5	3	2	R	DJNZ	data,data
216	D6	1	0	A	XCHD	A,@R0
217	D7	1	0	A	XCHD	A,@R1
218	D8	2	1	R	DJNZ	R0,data
219	D9	2	1	R	DJNZ	R1,data
220	DA	2	1	R	DJNZ	R2,data
221	DB	2	1	R	DJNZ	R3,data
222	DC	2	1	R	DJNZ	R4,data
223	DD	2	1	R	DJNZ	R5,data
224	DE	2	1	R	DJNZ	R6,data
225	DF	2	1	R	DJNZ	R7,data
226	E0	1	0	A	MOVX	A,@DPTR
227	E1	2	1	A	AJMP	data
228	E2	1	0	A	MOVX	A,@R0
229	E3	1	0	A	MOVX	A,@R1
230	E4	1	0	A	CLR	A
231	E5	2	1	A	MOV	A,data
232	E6	1	0	A	MOV	A,@R0
233	E7	1	0	A	MOV	A,@R1
234	E8	1	0	A	MOV	A,R0
235	E9	1	0	A	MOV	A,R1
236	EA	1	0	A	MOV	A,R2
237	EB	1	0	A	MOV	A,R3
238	EC	1	0	A	MOV	A,R4
239	ED	1	0	A	MOV	A,R5
240	EE	1	0	A	MOV	A,R6
241	EF	1	0	A	MOV	A,R7
242	F0	1	0	A	MOVX	@DPTR,A

TABOP

Id	CODI	LONG	A_data	REL_ABS_ADDRESS	MNEM	OPER
243	F1	2	1	A	ACALL	data
244	F2	1	0	A	MOVX	@R0,A
245	F3	1	0	A	MOVX	@R1,A
246	F4	1	0	A	CPL	A
247	F5	2	1	A	MOV	data,A
248	F6	1	0	A	MOV	@R0,A
249	F7	1	0	A	MOV	@R1,A
250	F8	1	0	A	MOV	R0,A
251	F9	1	0	A	MOV	R1,A
252	FA	1	0	A	MOV	R2,A
253	FB	1	0	A	MOV	R3,A
254	FC	1	0	A	MOV	R4,A
255	FD	1	0	A	MOV	R5,A
256	FE	1	0	A	MOV	R6,A
282	00	1	0	A	MOV	R7,A

ANEXO D.

MANUAL DEL USUARIO.

1.1 Introducción:

Este programa permite ensamblar archivos fuente escritos en lenguaje ensamblador usados en la programación del microcontrolador 8051/52.

El programa tiene la característica de presentar todos los resultados del proceso de ensamblado de una forma didáctica, emplea además una interfase programa-usuario más amigable que la desplegada por programas equivalentes en entorno DOS.

El programa permite dos tipos de ensamblado: directo y por pasos.

El ensamblado directo es transparente para el usuario y genera en un solo proceso los archivos de listado ".LST" y el módulo objeto definitivo ".OBJ".

El ensamblado por pasos presenta al usuario los siguientes archivos: el módulo fuente ".ASM", el archivo con macros expandidas ".MCR", un archivo de listado ".LST" preliminar producido por el paso 1 el cual contiene algunas etiquetas no definidas, un archivo de listado ".LST" definitivo que es producido por el paso 2 y por último el módulo objeto ".OBJ".

El módulo objeto producido por cualquiera de los dos tipos de ensamblado debe ser procesado por el enlazador para obtener el archivo ejecutable ".HEX".

De igual manera se facilita el enlazado de diferentes módulos objeto, ya que para realizarlo el usuario simplemente debe escoger entre los archivos ".OBJ" que tiene a su disposición en una librería cuyos elementos son mostrados a través de un cuadro de texto, el mismo que se encuentra localizado en la pantalla principal del programa.

El programa se encuentra conformado por las siguientes ventanas:

Ventana "Visor": Permite elegir entre ver el cuadro de texto principal o las tablas de datos.

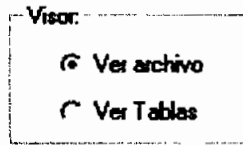


FIG. 1 Ventana "Visor".

Ventana "Tipo de ensamblado": Permite escoger entre un ensamblado directo y un ensamblado por pasos.

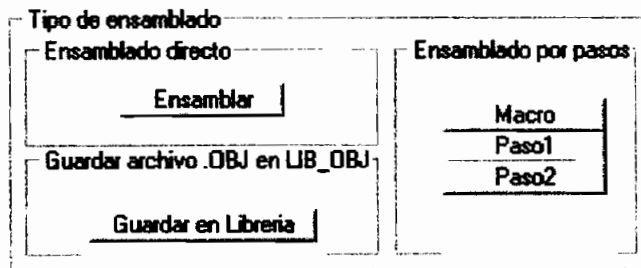


FIG. 2 Ventana "Tipo de ensamblado".

La ventana del enlazador: Permite seleccionar los archivos objeto que se desea enlazar.

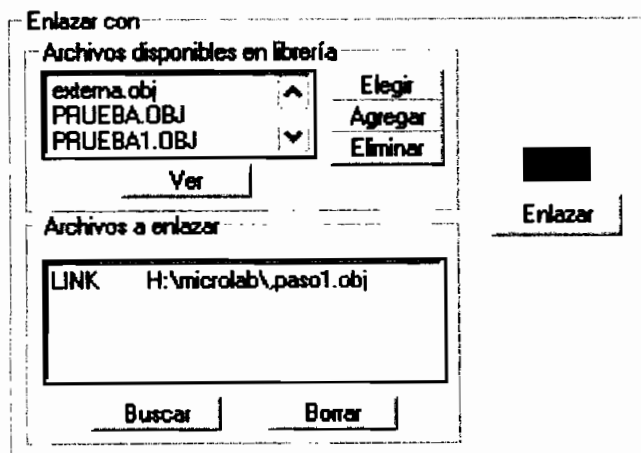


FIG. 3 Ventana del enlazador.

El cuadro de texto principal

Sirve para visualizar el módulo fuente y todos los resultados generados por los distintos pasos del programa enlazador.

```

Archivo .ASM
;Area de comandos
;Area de definicion de variables
;Area de instrucciones
etiql equ etiql
etiql equ 10100b
;Area de instrucciones
mov a,etiql
mov r0,etiql
simp $
;Area para tablas

end

```

FIG. 4 Cuadro de texto principal.

Ventana de traducción rápida y botones de navegación.

La ventana de traducción rápida está constituida por los cuadros de texto Mnemónico y Opcode.

El cuadro de texto Mnemónico presenta la instrucción de la línea en donde se encuentra el cursor.

El cuadro de texto Opcode presenta una traducción rápida de la línea del módulo fuente en la que se hizo clic con el mouse.

Mnemónico	MOV R0,ETIQ2
Opcode	A8,ETIQ2
Ver archivos:	
<input type="checkbox"/> .ASM	<input type="checkbox"/> .LST
<input type="checkbox"/> .OBJ	<input type="checkbox"/> .HEX

FIG. 5 Ventana de traducción rápida y botones de navegación.

Los botones .ASM, .LST, .OBJ, .HEX permiten navegar a través de todos los archivos que va generando el programa ensamblador con forme va traduciendo el módulo fuente a lenguaje de máquina.

2. Funciones del programa.

2.1 Crear un módulo fuente.

- 1.- Para crear un nuevo archivo ASM vaya a barra de menús.
- 2.- Elija el menú Archivo.
- 3.- Elija "Nuevo".

2.2 Abrir un módulo fuente.

- 1.- Para abrir un archivo ASM vaya a barra de menús.
- 2.- Elija el menú Archivo.
- 3.- Elija "Abrir".

2.3 Guardar un módulo fuente.

Importante: Es recomendable que el nombre con el que se guarde el módulo fuente no contenga espacios ya puede ocasionar conflictos al momento de enlazar el programa.

Guardar desde el menú archivo.

- 1.- Para guardar un archivo ASM vaya a barra de menús.
- 2.- Elija el menú Archivo.
- 3.- Elija "Guardar".

Guardar desde la pantalla principal.

Si el archivo es un documento nuevo se desplegará el cuadro de diálogo guardar_como, para este caso:

- 1.- Elija una carpeta en donde se guardará el archivo.
- 2.- Escriba un nombre para el archivo.
- 3.- Haga clic en el botón "Guardar".

Si no es un archivo nuevo al aplastar el botón "Guardar" el programa sobrescribirá el documento con las últimas modificaciones realizadas por el usuario.

2.4 Ensamblar un archivo.

Ensamblado directo.

Se ejecuta un ensamblado transparente para el usuario.

Desde la barra de menús.

- 1 Elija el menú "Ensamblar" de la barra de menús.
- 2 Elija "Ensamblado _directo".
- 3 Elija "Ensamblar".

Desde la pantalla principal.

Hacer clic en el botón  de la pantalla principal.

Ensamblado por pasos.

Con esta opción es el usuario quien ejecuta los distintos procedimientos para obtener el módulo objeto definitivo que luego será procesado por el enlazador para crear el archivo ejecutable.

El siguiente gráfico muestra la secuencia de pasos que el usuario debe ejecutar:

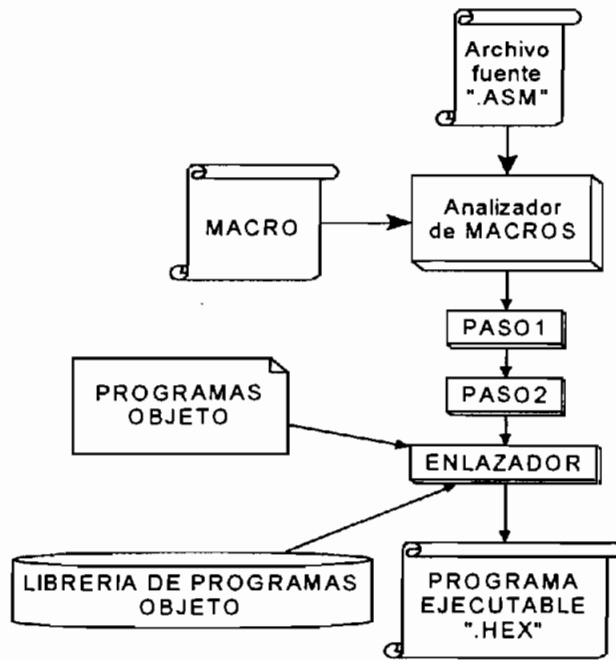


FIG. 6 Proceso de ensamblado.

Si el usuario no sigue la secuencia indicada se emitirán mensajes de texto indicando que se ha omitido uno o varios pasos.

Ensamblado por pasos desde la barra de menús.

- 1 Elija el menú "Ensamblar" de la barra de menús.
- 2 Elija "ensamblado por pasos".
- 3 Elegir submenú "Macro" para encontrar y analizar macro instrucciones.
- 4 Elegir "Paso1", en el cuadro de texto principal se presentará el archivo ".LST" con algunas etiquetas sin resolver.
- 5 Elegir "Paso2", se presenta en el cuadro de texto principal el archivo ".LST" definitivo.

Desde la pantalla principal.

Estos botones se localizan en la siguiente ventana:

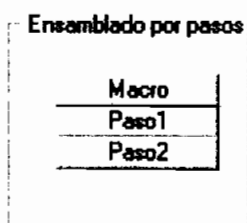


FIG. 7 Ensamblador de pasos.

- 1 Hacer clic en el botón **Macro** de la pantalla principal para encontrar y analizar macro instrucciones dentro del módulo fuente.
- 2 Hacer clic en el botón **Paso1** de la pantalla principal para ejecutar el paso1.
- 3 Hacer clic en el botón **Paso2** de la pantalla principal para ejecutar el paso2 y obtener el archivo de listado.

2.5 Enlazar.

Para realizar el enlazado de un módulo objeto con otro se tiene dos posibilidades, uno a través de la ventana "Enlazar con" de la pantalla principal y el segundo a través de la barra de comandos.

2.5.1 Enlazado a través de la pantalla principal.

La ventana "Enlazar con" de la pantalla principal es la siguiente:

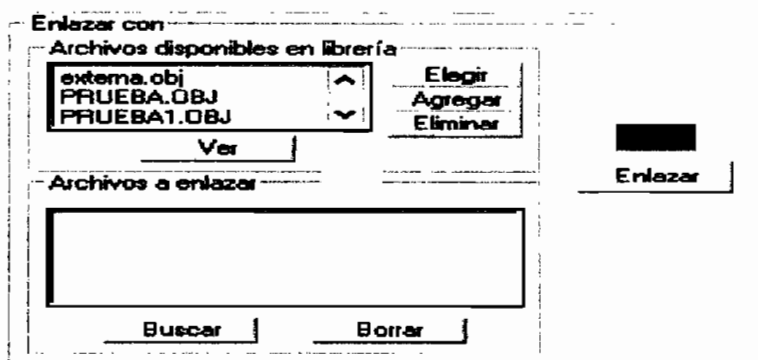


FIG. 8 Enlazador.

Esta ventana está conformada por la ventana de "Archivos disponibles en librería", la ventana "Archivos a enlazar" y el botón "Enlazar".

La ventana de "Archivos disponibles en librería" presenta de forma directa los archivos objeto almacenados en la librería con los que se puede enlazar el programa.



FIG. 9 Archivos de librería.

Botón Elegir: Selecciona el archivo elegido y lo coloca en la ventana "Archivos a enlazar".

Botón Agregar: Permite añadir un módulo objeto a la librería.

Botón Eliminar: Elimina un módulo objeto de la librería.

Botón Ver: Despliega en la ventana de texto principal el contenido del módulo objeto seleccionado.

Ventana "Archivos a enlazar": Despliega información de todos los archivos objeto que se van a enlazar.

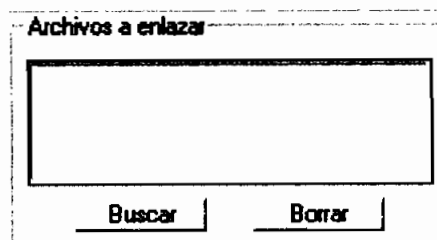


FIG. 10 Ventana con módulos .obj a enlazar.

El botón Buscar: Permite agregar en la pantalla "Archivos a enlazar" un módulo objeto que no se encuentra en la librería.

El botón Borrar: Elimina un módulo objeto de la pantalla "Archivos a enlazar".

Botón Enlazar: Ejecuta el enlace de todos los archivos objeto que se encuentran dentro del cuadro de texto de la ventana "archivos a enlazar".

2.5.2 Enlazado a través de la barra de menús:

- Elegir el menú enlazar de la barra de menús.
- Hacer clic en el submenú enlazar.

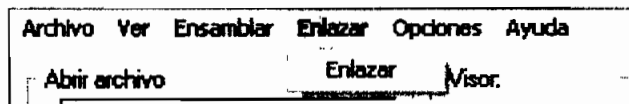


FIG. 11 Enlazado desde la barra de menús.

Visor

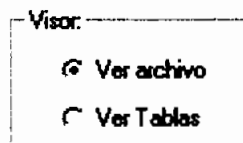


FIG. 12 visor.

Ver archivo: despliega en el cuadro de texto principal los archivos que el programa ensamblador genera. Este despliegue de información se realiza con la ayuda de los botones de la ventana "Ver archivo".

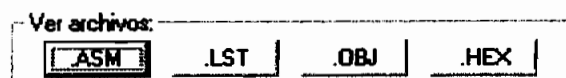


FIG. 13 Ventana "Ver archivos".

Ver tablas de datos: permite ver la información que contienen las tablas de datos usadas por el programa.

TABSIM	TABOP	SIM_MAT
175	FF	8
Etiqueta	1	}
# de elemo	0	
Valor	MOV	4
Direccion	R7A	
Nro_procec	A	
Publica o E		

FIG. 14 Vista de las tablas de datos.

Ver archivos

La ventana "Ver archivos" contiene botones que aparecen conforme se van generando los archivos de listado, archivos objeto y programas ejecutables.

El propósito de estos botones es el de permitir al usuario navegar por los distintos archivos generados por el ensamblador a través del cuadro de texto principal.

Archivo .LST

Lenguaje de máquina	Línea	Lenguaje ensamblador
	1	Area de comandos
	4	include c:\ejen
	5	Area de definicion de varie
	6	public supr,Avpag,repag
	7	extern bloqnum,klm
0014 =	8	bin equ 10100b
0020 =	9	supr equ 20h
0003 =	10	avpag equ repag
0003 =	11	repag equ 3
	12	Area de instrucciones
0000 B500FD	14	cjne a,bloqnum-1,klm
0003 F500	15	mov bloqnum+bin,a
0005 E500	16	mov a,bloqnum-klm
0007 E500	17	mov a,(bloqnum+(klm-
	18	org 15h
0015 740A	19	mov a,#0ah
0017 80FE	20	jmp \$
	21	Area para tablas
0019 61 62 63 64	22	marcelo1 db "abcd"
	23	end

Número total de errores: 0

Minerónico: _____

Opcode: _____

Ver archivos:

FIG. 15 Vista de la pantalla principal.

Botón ASM: Despliega en el cuadro de texto el módulo fuente.

Botón LST: Despliega en el cuadro de texto el archivo de listado.

Botón OBJ: Despliega en el cuadro de texto el módulo objeto.

Botón HEX: Despliega en el cuadro de texto el archivo ejecutable.

Cuadro de texto Mnemónico: Presenta la instrucción de la línea de texto donde se encuentra el cursor.

Cuadro de texto Opcode: Presenta la traducción a lenguaje de máquina del contenido del cuadro de texto "Mnemónico".

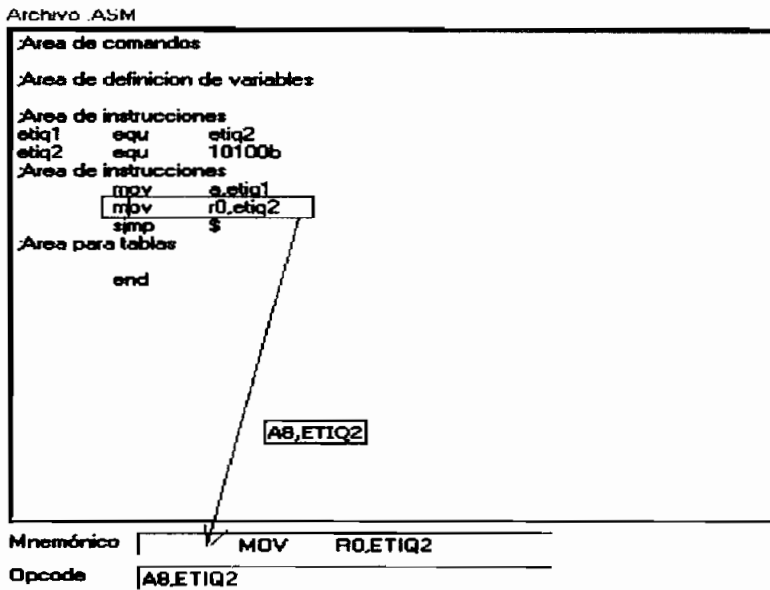


FIG. 16 Funcionamiento del traductor rápido.

2.6 Seudo instrucciones para el programa ensamblador.

2.6.1 Seudo instrucción Link .

Definición: permite indicar la localización y definición de los archivos objeto que se van a enlazar.

Formato: Esta seudo instrucción ocupa los campos "Instrucción" y "Operando" del módulo fuente.

En el campo instrucción va la palabra "Link" con la cual se indica al ensamblador que se desea realizar un enlace.

Al campo operando se lo divide en varios suboperandos separados por una "," de la siguiente manera:

El primer suboperando contiene la dirección de el o los archivos objeto que se van a enlazar.

En el suboperando 2 y en los que le siguen a continuación se encuentran los nombres de los archivos objeto que se desea enlazar.

En el siguiente ejemplo se demuestra con mayor claridad lo que se acaba de describir:

Etiqueta	Instrucción	Operando
	Link	c:\ejemplo\,arch1.obj,arch2.obj,...

FIG. 17 Seudo instrucción Link.

2.6.2 Seudo instrucción Include

Definición: Permite indicar la localización del archivo donde se encuentran almacenadas las macro instrucciones.

Formato: En el campo instrucción del módulo fuente va la palabra "INCLUDE" la cual le indica al ensamblador que se va a definir la localización de una o varias macro instrucciones a través de lo que exista en el campo operando.

Al igual que en la seudo instrucción anterior, se divide al campo operando en suboperandos separados por una ",".

En el primer suboperando se especifica la dirección del archivo que contiene las macro instrucciones. Es importante indicar que el nombre del archivo debe contener la extensión del mismo ya que las macro pueden estar almacenadas dentro de archivos ".TXT", ".ASM" o cualquier otro archivo de texto.

El suboperando 2 y los que le siguen a continuación contienen los nombres de las macro que se van a usar.

Una vez que se ha definido la localización y el nombre de las macro ya se las puede usar en cualquier parte del módulo fuente como si se tratase de una instrucción.

En el siguiente ejemplo se definirá una macro llamada "prueba" la misma que se encuentra dentro de "c:\ejemplo\macros.txt".

El archivo "c:\ejemplo\macros.txt" contiene lo siguiente:

```

May      macro    &1,&2,&3
          mov     a,&1
          dec    a
          db     "hola tania",120
          endm
lucky    macro    &1,&2,&3
          dw     100h,20h
          endm
Prueba   macro    a,b
          mov    b,#40h
          dec   a
          inc   40h
          mov   a,40h
          endm

```

FIG. 18 Archivo "macros.txt".

A continuación se presenta la forma como se define una macro y la manera como se la invoca:

```

;Área de comandos
include    c:\ejemplos\macros.txt,prueba ;<--- Definición de macro
;Área de instrucciones
prueba    a,50 ; <---- Invocación de macro

```

FIG. 19 Definición e invocación de una macro instrucción.

Cuando el analizador de macro instrucciones revisa el módulo fuente, expande todas las invocaciones a macros que va encontrando, quedando como resultado final lo siguiente:

```
Area de comandos
include c:\ejemplos\macros.txt,prueba
Area de instrucciones
MOV 50,#40h
DEC A
INC 40h
MOV A,40h
```

FIG. 20 Archivo con macro instrucciones expandidas.

El resultado final se almacena en un archivo que tiene la misma dirección y el mismo nombre del módulo fuente pero con extensión “.MCR”. Este es el archivo que el paso 1 tomará como referencia para comenzar con la traducción de lenguaje ensamblador a lenguaje de máquina.

2.6.3 Seudo instrucción PUBLIC:

Definición: Permite identificar las variables externas que se van a definir dentro del módulo fuente.

Formato: En el campo instrucción del módulo fuente va la palabra “Public” y en el campo operando las variables separadas por comas.

2.6.4 Seudo instrucción EXTERN:

Definición: Permite identificar las variables que se emplearán en el programa pero que serán definidas en el momento de enlazar el módulo objeto.

Formato: En el campo instrucción del módulo fuente va la palabra “Extern” y en el campo operando están las variables separadas por comas.

2.6.5 Seudo instrucción EQU:

Definición: A través de la seudo instrucción “EQU” se puede asignar o cambiar el valor de las etiquetas.

Formato: En el campo instrucción debe estar la palabra “EQU”. Mientras que en el campo operando debe registrarse el valor que se le desea asignar a la variable.

A continuación se muestra un ejemplo de las tres seudo instrucciones anteriores:

	public	supr,Avpag,repag
	extern	bloqnum,klm
bin	equ	10100b
supr	equ	20h.bit
avpag	equ	repag
repag	equ	3
ghi	equ	22o

FIG. 21 Seudo instrucciones public, extern, equ.

2.6.6 Seudo Instrucción "DS":

Definición: Esta seudo instrucción altera el contador de localidades para reservar espacio en memoria.

Formato: En el campo instrucción debe estar la palabra DS y en el campo operando debe constar el valor con el que se desea alterar el contador de localidades.

Ejemplo:

Si el contador de localidades tiene inicialmente un valor de 00h, al escribir la siguiente secuencia:

```
Plastic    ds    20
           db    20,1001b,0ah
```

El archivo ".LST" resultante será el siguiente:

		1	plastic	ds	20
0014	14 09 0A	2		db	20,1001b,0Ah

FIG. 22 Archivo de listado resultante para la seudo instrucción DS.

Como se puede ver el resultado de la seudo instrucción:

" db 20,1001b,0Ah "

se almacenará en la localidad número 14h dejando las primeras veinte localidades sin ninguna instrucción.

2.6.7 Seudo instrucción "DB":

Definición: Esta seudo instrucción reserva espacio en la memoria de programa al igual que "DS" con la diferencia de que permite guardar valores o caracteres iniciales. De esta forma se puede tener almacenado en memoria ROM tablas de datos o mensajes de texto.

Formato: En el campo instrucción debe constar la palabra DB y en el campo operando se debe escribir lo que se desea almacenar en la memoria de programa. Las cadenas de texto deben estar encerradas entre comillas.

Si dentro del mensaje se coloca el símbolo "\" seguido de uno de los caracteres que se muestra en la tabla 1, el programa deberá reemplazar estos símbolos por su código equivalente.

CARACTERES CON "\"		
CARÁCTER	CARÁCTER INSERTADO	Equivalente
\n	Nueva línea	0A
\r	Retorno de carro	0D
\t	Tabulador	09
\0	Carácter nulo	0
\x	La letra x debe estar seguida por 2 dígitos hexadecimales.	\x0A devuelve 0A

TABLA 1 opciones para seudo instrucción DB.

En el siguiente cuadro se muestra el archivo ".LST" resultante si se ejecuta un ejemplo con cada uno de estos caracteres:

Lenguaje de máquina		#línea	Lenguaje ensamblador	
		1	db	"A\n"
0000	41 0A	2	db	"A\r"
0002	41 0D	3	db	"A\t"
0004	41 09	4	db	"A\40"
0006	41 40	5	END	
		6		

FIG. 23 Archivo de listado para la pseudo instrucción DB.

2.6.8 Seudo instrucción "DW":

Definición: Permite reservar espacios en palabras de dos bytes. Por esta razón no están permitidas cadenas de más de dos caracteres de longitud.

Formato: En el campo instrucción debe constar la palabra DW. En el campo operando se debe escribir lo que se desea almacenar en la memoria de programa. Las cadenas de texto deben estar encerradas entre comillas.

El siguiente ejemplo muestra el formato de esta instrucción y los resultados luego de ensamblarla.

Para las siguientes sentencias:

```

plastic dw "hola a todos"
         dw 100h,1000
         dw 0101010b

```

FIG. 24 Ejemplo para pseudo instrucción "DW".

El Archivo ".LST" que se generará será el siguiente:

```

0000 686F          1 plastic dw "hola a todos"
0002 0100          2 dw 100h,1000
0004 03E8          3 dw 0101010b
0006 002A

```

FIG. 25 Archivo ".LST".

2.6.9 Seudo instrucciones "PROC", "ENDPROC" y etiquetas locales:

Definición: A través de "PROC" y "ENDPROC" se pueden establecer procedimientos dentro de un programa principal. La característica más importante de estos procedimientos es la de definir variables locales.

Formato: Debe tener una etiqueta, en el campo instrucción debe estar la palabra PROC para iniciar un procedimiento y la palabra ENDPROC para finalizar el procedimiento.

Las variables que se definan dentro del procedimiento son locales, es decir, tienen significado solo dentro del procedimiento, fuera de él no tienen validez, esto implica que el nombre de una variable local puede volver a utilizarse fuera del procedimiento. La condición inversa también se cumple, es decir que el nombre de una etiqueta del programa principal puede volver a utilizarse dentro de un procedimiento sin ningún problema.

Una condición importante es que todos los procedimientos PROC-ENDPROC deben tener un nombre. El nombre asignado a un procedimiento es la etiqueta que aparece en la línea que contiene a la pseudo instrucción "PROC".

La principal ventaja de utilizar procedimientos es el permitir modularizar la programación.

En el siguiente ejemplo se muestra la manera en que se pueden utilizar estas pseudo instrucciones.

Para el siguiente programa:

inicio	mov	a, inicio	
ejemplo	proc		
inicio	mov	a, inicio	Procedimiento dentro del programa principal y variables locales
etiq1	mov	a, etiq1	
	endproc		
etiq1	mov	a, etiq1	
	end		

FIG. 25 Forma de empleo de las pseudo instrucciones PROC y END PROC.

El archivo ".LST" será el siguiente:

0000	E500	1	inicio	mov	a,inicio
		2	ejemplo	proc	
0002	E502	3	inicio	mov	a,inicio
0004	E504	4	etiq1	mov	a,etiq1
		5		endproc	
0006	E506	6	etiq1	mov	a,etiq1
		7		end	

FIG. 26 Archivo de listado

2.6.10 Instrucción "ORG":

Definición: Permite colocar instrucciones de programa en una dirección determinada por el usuario. Para lograr esto, el ensamblador altera el contador de localidades con el valor que se especifique en el campo operando.

Ejemplo:

Si se desea colocar un grupo de instrucciones en la dirección 9h y otro grupo en la dirección 18h se lo hará de la siguiente manera:

	org	09h
inicio	mov	a,inicio
ejemplo	proc	
inicio	mov	a,inicio
etiq1	mov	a,etiq1
	endproc	
	org	18h
	mov	a,40h
etiq1	mov	a,etiq1
	end	

FIG. 27 Archivo fuente con seudo instrucción ORG.

El archivo ".LST" resultante será el siguiente:

		1		org	09h
0009	E509	2	inicio	mov	a,inicio
		3	ejemplo	proc	
000B	E50B	4	inicio	mov	a,inicio
000D	E50D	5	etiq1	mov	a,etiq1
		6		endproc	
		7		org	18h
0018	E540	8		mov	a,40h
001A	E51A	9	etiq1	mov	a,etiq1
		10		end	

FIG. 28 Archivo de listado.

2.6.11 Seudo instrucción "END"

Sirve para indicar al ensamblador que el programa ha terminado y que debe concluir con todas las operaciones pendientes, tal como, la impresión de caracteres del módulo objeto o archivos de listado.

2.6.12 Comodín "\$"

Este comodín se emplea cada vez que se desee conocer el valor del contador de localidades. La característica principal es que se lo puede emplear para definir etiquetas o cualquier expresión matemática.