4. ENSAMBLADORES Y MACROENSAMBLADORES

En el capítulo previo se habló sobre las instrucciones del procesador y se abordó lo referente al uso del debug como una herramienta para escribir programas en ensamblador, revisar que no tuvieran errores de sintaxis y de semántica y así poder ejecutarlos.

Este capítulo nos permite ir un paso adelante, ahora se podrá escribir (editar) un programa en ensamblador y pasarlo por un proceso de revisión y preparación previa a su ejecución. Esto se hará a través del uso de software destinado para tal fin.

4.1 DEFINICIÓN Y USOS

Definición: Un ensamblador es un programa que traduce mnemónicos de un procesador a su correspondiente lenguaje de máquina.

Por la ***forma*** en que ***trabajan*** existen dos tipos de ensambladores:

- *Ensambladores de Línea:* Son aquellos que reciben una sola línea de un programa y la ensambla independientemente del resto del programa.
- *Ensambladores de Archivo:* Son aquellos que ensamblan todo un programa almacenado en un archivo.

Por el ***tipo*** de ***información*** que manejan los ensambladores se dividen también en:

- ***Ensambladores Propios (residentes):*** Ensamblan programas escritos en lenguaje del procesador con que trabaja la máquina.
- ***Ensambladores Cruzados (crossassembler):*** Ensamblan programas escritos en lenguaje de un procesador diferente al de la computadora de trabajo, pero no puede ejecutarse.
- ***Macroensambladores:*** Ensambladores propios o cruzados que permiten la definición y expansión de MACROS.

Facilidades de los ensambladores de Archivo:

1. Permite definir etiquetas (nombre que marca una dirección importante).
2. Permite reservar memoria con una etiqueta asignada.
3. Permite ensamblar programas almacenados en archivos.
4. Permite definir constantes.
5. Permite dar números en distintas bases.
6. Permite expresiones aritméticas.

El macroensamblador recibe archivos ASCII editados en cualquier editor de texto, con programas en lenguaje ensamblador, con una extensión **.ASM**, y en un formato especial.

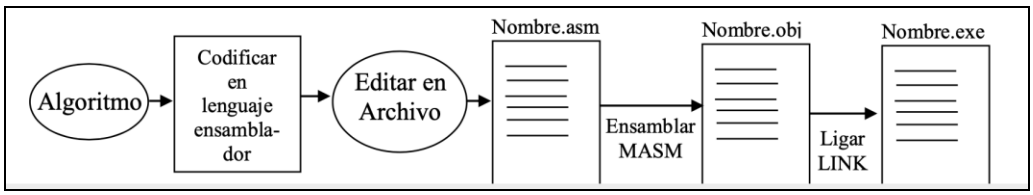
4.2. PASO DE PARÁMETROS

MASM (ensamblador del 8086/88)

Recibe archivos ASCII editados en cualquier editor de texto que contenga programas en lenguaje ensamblador.

Tales archivos deben tener extensión **.ASM** y con una forma específica.

La siguiente gráfica describe la secuencia de pasos desde la creación de un programa en ensamblador hasta la generación del programa ejecutable.



El archivo objeto no se puede ejecutar porque no tiene la dirección de memoria donde se será ligado y ejecutado.

Definición: Una ***pseudoinstrucción*** es una instrucción para el programa ensamblador, esto es, que solo se ejecuta en el momento de ensamblar, además de no generar código.

4.2.1 PSEUDOINSTRUCCIONES PARA DEFINIR SEGMENTOS.

SEGMENT Define el inicio de un nuevo segmento, el formato es:

Nombre* SEGMENT *alineación combinación clase

Los parámetros del SEGMENT dan información para el ligador.

Alineación Nos define la dirección a partir de donde puede colocarse el segmento.

- **PARA** (PARAGRAFH). La dirección del segmento es un múltiplo de 16.
- **PAGE**. La dirección inicial del segmento es donde empieza una página (múltiplo de 100h).

- **WORD**. La dirección inicial del segmento es una dirección par.
- **BYTE**. El segmento inicia donde sea.

Si no se especifica la alineación toma por default una alineación tipo **PARA** y será un múltiplo de 16.

Combinación Define la forma en que el segmento puede combinarse con otros segmentos que tengan el mismo nombre y la misma clase.

- **OMITIRLA**. Segmento privado, es decir, no puede combinarse.
- **STACK**. Segmento para usarse con el SS:SP
- **PUBLIC**. Este segmento puede unirse con todos los segmentos del mismo nombre y la misma clase para formar uno solo.
- **COMMON**. Todos los segmentos del mismo nombre y clase se colocan a partir de la misma dirección.

Cuando se tienen 2 segmentos con el mismo nombre y clase y PUBLIC al ligar se unirán en un solo segmento no importando que estén en archivos distintos.

Cuando se usa el **COMMON** van a utilizar el mismo espacio en memoria y si son de diferente tamaño utilizará el tanto de memoria del mayor.

Clase Indica el tipo de datos que contiene el segmento.

- '**DATA**' datos
- '**CODE**' código
- '**STACK**' stack

Sin embargo, se pueden definir otras clases. Siempre van entre comillas.

El programa **LINK** puede ligar varios archivos objeto para crear un ejecutable.

ENDS Define el final de un segmento. Formato:

Nombre ENDS

Ejemplo:

```
DATOS SEGMENT PARA 'DATA'
        Aquí se definen variables y constantes del programa
DATOS ENDS

CODIGO SEGMENT PARA 'CODE'
        Aquí van el Programa principal y las subrutinas
CODIGO ENDS
```

4.2.2 PSEUDOINSTRUCCIONES (DIRECTIVAS) PARA RESERVAR MEMORIA Y DEFINIR CONSTANTES.

DB Sirve para reservar un byte en la memoria con un valor determinado. El formato es:

Etiqueta DB val1[, val2,....,valn]

DW Reserva un dato de 2 bytes (una palabra) con un valor inicial

[etiqueta] DW val1[, val2,....,valn]

DD Reserva un dato de 4 bytes (una doble palabra) con un valor inicial

[etiqueta] DD val1[, val2,....,valn]

DQ Reserva 8 bytes de memoria (cuadruple palabra) con un valor inicial

[etiqueta] DQ val1[, val2,....,valn]

DT Reserva un dato de 10 bytes con un valor inicial

[etiqueta] DT val1[,val2,....,valn]

Nota: val_i representa una expresión formada por números en cualquiera de las siguientes bases:

XXXB binario
XXXO octal
XXX
XXXD decimal
XXXH hexadecimal

O bien **expresiones aritméticas** de esos números con los siguientes operadores

+ suma, **-** resta, - negación aritmética (complemento a 2), * multiplicación y / división.

También pueden ser etiquetas o expresiones aritméticas que involucren etiquetas o bien cadenas de ASCII's limitadas por apóstrofes.

Ejemplo:

```
DATOS SEGMENT PARA 'DATA'
        UNO     DB    08H
        DOS     DW    0F48H, 20O, 10111B
        TRES    DD    (45FH+178H)*2
        CUATRO  DQ    78
        CINCO   DW    DOS, 'B'
        SEIS    DB    'ES UNA CADENA'
DATOS ENDS
```

EQU Permite definir constantes

Etiqueta **EQU** valor

Ejemplo:

CONSTANTE EQU 34

ORG Define el desplazamiento inicial para ensamblar las siguientes líneas en el texto

ORG valor

4.2.3 PSEUDOINSTRUCCIONES PARA DEFINIR PROCEDIMIENTOS

PROC Define el inicio de una subrutina

Nombre **PROC** tipo

Si se omite el tipo, por default el procedimiento es de tipo NEAR

ENDP Indica el final de una subrutina

Nombre **ENDP**

Uso del MASM

Para ensamblar un programa con MASM de forma condensada se utiliza:

c:\>masm archivo;

Si no tiene errores el programa entonces se liga para crear el ejecutable:

c:\>link archivo;

sino primero hay que corregir errores.

Estructura básica de un programa en macroensamblador.

```
DATOS SEGMENT PARA 'DATA'
        ..... DATOS
ENDS

PILA SEGMENT PARA STACK 'STACK'
        DW 100 DUP(0) PILA
ENDS
```

```
CODIGO SEGMENT PARA 'CODE'
        ASSUME DS:DATOS, CS:CODIGO, SS:PILA, ES:NOTHING

; PROGRAMA PRINCIPAL MAIN
PROC FAR
        PUSH DS             ; Sirve para que cuando el programa termine XOR
        AX,AX               ; pueda regresar al debug o al sistema operativo PUSH
        AX                  ;
        MOV AX,DATOS        ;
        MOV DS, AX          ; Actualiza los registros de segmentos de datos y extra (*)
        MOV ES, AX          ;
        ……..                ; código del programa principal
        RET MAIN
ENDP

SUB1 PROC
        …..                 ; código de la primera subrutina
SUB1 ENDP
…….
SUBN PROC
        ……
SUBN ENDP

CODIGO ENDS END
MAIN
```

Nota. El programa principal puede ir al inicio o al final del segmento de código.

DUP Es un operador de MASM que indica que tiene que repetir la pseudoinstrucción *n* veces con valores iniciales marcados entre paréntesis.

D**x** n **DUP** (val1,val2, ..., valm)

Ejemplo:

```
DB 5 DUP (1,2)
```

En la memoria encontraríamos lo siguiente:

```
XXX   01
XXX+1  02
XXX+2  01
XXX+3  02
XXX+4  01
```

ASSUME es la pseudoinstrucción que sirve para indicarle al ensamblador cuales segmentos son usados por los registros de segmentos y se pueden calcular correctamente las direcciones de los operandos.

Las instrucciones (*) sirven para indicarle al procesador qué segmentos usará para DATOS y EXTRA, es decir, los únicos registros que se cargan automáticamente son el segmento de código y el de stack.

END Indica al MASM que el ensamble termina

END dir ← dirección del programa principal en debug (una etiqueta en MASM)

4.3. EJEMPLO DE PROGRAMA

Programa que limpia pantalla y escribe una cadena en el renglón12, columna 30.

```
DATOS SEGMENT PARA 'DATA'
        CADENA      DB      'ESTO ES UNA PRUEBA$'
        REN         DB      12
        COL         DB      30
DATOS ENDS

PILA SEGMENT PARA STACK 'STACK'
        DW    100     DUP (0)
PILA ENDS

CODIGO SEGMENT PARA 'CODE'
        ASSUME      DS:CODIGO, DS:DATOS, SS:PILA, ES:NOTHING

    LIMPIA_PANTALLA PROC NEAR
            MOV AX, 0600H
            MOV BH, 71H
            MOV CX, 0000H
            MOV DX, 184FH
            INT 10H
            RET LIMPIA_PANTALLA
    ENDP
```

```
POSICIONA_CURSOR PROC NEAR MOV
        AH,02
        MOV BH,00
        MOV DH,REN
        MOV DL, COL
         INT 10H
        RET POSICIONA_CURSOR
ENDP

LETRERO     PROC NEAR MOV
        AH,09
        LEA DX, CADENA INT 21H
        RET LETRERO
ENDP

PRINCIPAL PROC FAR
        PUSH DS
        XOR AX, AX
         PUSH AX
        MOV AX, DATOS
        MOV DS, AX
        MOV ES, AX
        CALL LIMPIA_PANTALLA
        CALL POSICIONA_CURSOR
        CALL LETRERO
        MOV AH,00
        INT 21H
        RET
PRINCIPAL ENDP
CODIGO   ENDS
END PRINCIPAL
```