Ensamblador para el 80x86

ARCHIVOS FUENTES, OBJETOS Y EJECUTABLES

Son necesarias 3 herramientas para escribir programas en lenguaje ensamblador: un editor de textos para hacer los archivos fuentes, el MASM que sirve para generar archivos objetos a partir de los archivos fuentes, y el LINK para combinar uno o mas archivos objetos hacia un archivo ejecutable que puede ser ejecutado por DOS.

Después de que se crea un programa fuente en MASM, este debe ser almacenado en un archivo. Este es referido como archivo fuente, que es un archivo de texto que contiene enunciados en lenguaje ensamblador, cada uno de estos termina con los caracteres CR y LF (Retorno de carro y salto de línea). Generalmente los nombres de los archivos fuentes tienen la extensión ASM.

El resultado de ensamblar un archivo fuente es un archivo binario con el código maquina y las instrucciones para el encadenador (LINK). este archivo es llamado archivo objeto y tiene la extensión por defecto OBJ.

Uno o mas archivos objeto son combinados por el encadenador para formar un programa ejecutable, el cual tiene la extensión por defecto EXE.

Hacer los archivos fuentes para lo cual se puede utilizar cualquier editor de textos es solo un aspecto para el desarrollo de programas en MASM. Se tiene que ensamblar el código, encadenar con las librerías para crear el archivo ejecutable, y finalmente, depurar el programa cuando este falla. MASM viene con un conjunto de herramientas que nos pueden ayudar en uno o mas de estos pasos. El MASM y el LINK son accedidos vía la línea de comandos por los comandos nombrados MASM.EXE y LINK.EXE. También hay una utilería llamada MAKE que nos permite automatizar los pasos para crear los archivos ejecutables. Finalmente el depurador de pantalla completa CODEVIEW o DEBUG de MS-DOS que nos ayudan a encontrar errores dentro de los programas.

Hasta que e l MASM traslada los archivos fuentes en lenguaje ensamblador a archivos objeto conteniendo el c_ digo m_ quina, MS- DOS no puede ejecutar los archivos objeto. Se debe de utilizar el encadenador para generar el archivo ejecutable. El encadenador combina uno o mas archivos objetos en un archivo ejecutable. también hace algunas tareas que hacen posible organizar un programa muy grande en peque_ os archivos fuentes. En tales casos, el código en archivo fuente seguido hace referencia a variables y funciones que están definidas en otro archivo. Cuando el MASM traslada el archivo fuente a un archivo objeto, especifica la información acerca de las funciones externas y variables referidas en este archivo. El encadenador es capaz de manejar varios archivos objeto resolviendo las referencias externas y generando un programa ejecutable.

LST Este archivo generado por el MASM muestra información sobre los segmentos y símbolos usados en el programa, el nombre por defecto es NUL.LST, no se genera este archivo.

CRF Este archivo contiene acerca de cada símbolo en el código del programa fuente mostrando el número de línea donde fue declarado y que otras líneas se refieren a este, el nombre por defecto de este es NUL.CRF, no se genera este archivo, si este se genera se requiere usar el programa CREF para que este sea transformado a una forma legible.

DECLARACION DE SEGMENTOS

Declaración Completa de Segmentos

```
nom_seg SEGMENT [alinamiento] [combinación] ['tipo_clase']
      ---------
      ---------
nom_seg ENDS
```

nomseg : Nombre definido por el usuario.

alineamiento : Este atributo especifica el limite físico donde el LINK debe establecer este segmento en memoria (MASM pasa esta información al LINK en el archivo objeto), Si no se especifica ningún alineamiento este se asume como **PARA**. Estos pueden ser:

BYTE : El segmento se establece en el siguiente byte disponible.

WORD : El segmento se establece en la siguiente palabra disponible.

DWORD : El segmento se establece en la siguiente palabra doble disponible.

PARA : El segmento se establece en el siguiente párrafo disponible.

PAGE : El segmento se establece en la siguiente pagina (256 bytes) disponible.

combinación :Este atributo describe como el LINK debe combinar los segmentos lógicos con nombres indeciso en un solo segmento físico. Los posibles atributos de combinación son los siguientes:

PUBLIC o **MEMORY** : Todos los segmentos con cualquiera de estos atributos de combinación, que contengan el mismo nombre de segmento serán encadenados en un solo segmento físico.

COMMON : Todos los segmentos con este atributo de combinacion y el mismo nombre de segmento, el LINK comenzara estos segmentos logicos con el mismo nombre en la misma dirección de memoria fisica, tal que los segmentos logicos se encuentran traslapados.

STACK : Este atributo de combinacion indica que todos los segmentos con este atributo de combinacion serán encadenados en uno solo , el registro **SS** contendra la dirección de este segmento y **SP** el tamaño de este segmento cuando DOS carga el programa antes de ejecutarlo.

AT xxxx : Este atributo de combinacion no define mas datos al programa, solamente nos permite hacer referencia a una dirección especifica de memoria por medio de nombres simbolicos, xxxx es la dirección fija de memoria que se le asignara a este segmento.

Ejemplo:

```
Bios_Data_Area SEGMENT AT 40h
    ORG 10h
    Equipment_flag DW ?
Bios_Data_Area ENDS
```

tipo de clase : La clase de segmento especificada entre comillas simples. Cuando el LINK acomoda los segmentos, establece los segmentos que pertenecen al mismo tipo de clase, uno despues de otro. La mayoria de los programas en MASM usan los tipos de clase:

'CODE' Para los segmentos de codigo.
'DATA' Para los segmentos de datos.
'STACK' Para los segmentos de stack.

Esto ayuda a tener los segmentos de codigo, datos y stack en forma contigua respectivamente.

Algunas veces cuando se hacen subrutinas para lenguajes de alto nivel el tipo de clase debe ser el especificado por el lenguaje de alto nivel.

Programas EXE

Ejemplo de la declaracion de los segmentos:

```
DATA SEGMENT PARA PUBLIC 'DATA'
    definicion de
    datos (variables)
    espacio en memoria donde
    se almacenará informacion.
DATA ENDS
```

```
CODE SEGMENT PARA PUBLIC 'CODE'
    ASSUME CS:CODE,DS:DATA,ES:DATA,SS:STACK
ENTRY:
    MOV AX,DATA
    MOV DS,AX
    aqui debe estar el
    codigo, instrucciones, etc.
CODE ENDS

STACK SEGMENT PARA STACK 'STACK'
    aqui se define el espacio
    nesesario para el stack.
STACK ENDS
END ENTRY
```

Programas COM

Ejemplo de la declaracion del segmento:

```
CODE SEGMENT
    ASSUME CS:CODE,DS:CODE,ES:CODE
    ORG 100H
ENTRY:
    JMP INICIO
    Definicion de datos (variables)
    espacio en memoria donde se
    almacenará informacion.
INICIO:
    aqui debe estar el codigo, instrucciones
```

```
    etc.
CODE ENDS
END ENTRY
```

Etiquetas: Nombres que se les asignan a las direcciónes de cualquier instruccion o localizacion de memoria. Una etiqueta toma el lugar de una dirección. Tan pronto como el ensamblador encuentra una etiqueta , esta es remplazada por la dirección correcta:

Ejemplo:

```
    JMP 0110
```

Hacer un salto a la dirección 0110.

```
    JMP INICIO


INICIO:
```

Hacer un salto a la dirección donde se encuentra la etiqueta inicio, una vez que el programa se ensamble suponiento que la etiqueta INICIO se encuentra en el offset 0110. Este JMP INICIO será remplazado por un JMP 0110.

Directivas: Estas son instrucciones para el ensamblador, estas no generan codigo como las instrucciones del procesador, pero indican al ensamblador y al editor de encadenamiento como debe generar codigo.

Ejemplos de directivas:

Declaracion de segmentos y atributos.

ASSUME : Esta directiva indica el nombre del segmento que el registro de segmento direccióna.

ORG : Para establecer la localizacion del contador del valor numerico que se daran a las etiquetas.

ORG 100h Indica que ensamble codigo a partir del offset 100h , esta directiva es nesesaria para los programas **COM**.

END : Para marcar el final de un archivo fuente e indicar el punto de entrada del programa, esta dirección será cargada a los registros CS:IP cuando el programa entre en ejecucion.

Declaracion Simplificada de Segmentos

Esta comienza con la directiva .MODEL, la cual indica el modelo de memoria, el cual indica como se direcciónaran las instrucciones y datos, los modelos de memoria pueden ser los siguientes:

Modelo	Segs. de Codigo	Segs. de Datos
Small	Uno	Uno
Medium	Múltiples	Uno
Compact	Uno	Múltiples
Large	Múltiples	Múltiples

El modelo de memoria mas utilizado es el **Small** el cual tiene un segmento de codigo y un segmento de datos.

Ejemplo:

```
.MODEL SMALL
.DATA
    definicion de datos (variables)
    donde se almacenará informacion

.CODE
INICIO:
    aqui debe de estar
    el codigo del programa.

.STACK
    aqui se define el espacio nesesario
    para el stack.
END INICIO
```

Programa COM que imprime un mensaje en pantalla

```
; Este es un programa COM
CODE SEGMENT  ; declaración del segmento

    ASSUME CS:CODE,DS:CODE,ES:CODE,SS:CODE

    ORG 100h  ; origen del valor de las etiquetas 100h
ENTRY:        ;etiqueta
    JMP INICIO ; Salta a la dirección de la etiqueta inicio
    SALUDO DB 'Este es un programa COM $'
               ; definición de datos
INICIO:
     MOV AH,9  ; servicio 9, impr. un string terminado en '$'
     MOV DX,OFFSET SALUDO ; OFFSET SALUDO será reemplazado por
el
                          ; ensamblador por la dirección donde
                          ; se encuentra la etiqueta SALUDO
     INT 21h   ; Transferir a MS-DOS
     INT 20h   ; Servicio de MS-DOS para terminar proceso
CODE ENDS      ; fin del segmento
END ENTRY      ; Se indica el punto de entrada 100h es el valor
               ; de ENTRY
```

Para ensamblar este programa:

```
A>MASM PROG1COM;
```

Encadenar :

```
A>LINK PROG1COM;
```

Convertir a COM :

```
A>EXE2BIN PROG1COM.EXE PROG1COM.COM
```

Programa EXE que imprime un mensaje en pantalla

```
; Este es un programa EXE
DATA SEGMENT PARA PUBLIC 'DATA' ; Def. del segmento DATA
    SALUDO DB 'Este es un programa EXE $' ; Def. de datos
DATA ENDS   ; Fin del segmento DATA

CODE SEGMENT PARA PUBLIC 'CODE' ; Def. del segmento CODE
    ASSUME CS:CODE,DS:DATA,ES:DATA,SS:STACK
```

```
ENTRY:                ; Punto de entrada del programa
    MOV AX,DATA       ; AX tiene la dirección del segmento DATA
MOV DS,AX         ; DS tiene la dirección del segmento DATA
    MOV AH,9          ; Servicio 9 Impr. un string terminado en
'$'
    MOV DX,OFFSET SALUDO ; OFFSET SALUDO será reemplazado por
el
                      ; ensamblador por la dirección del offset
                      ; donde este la etiqueta SALUDO.
    INT 21h           ; Transferir a MS-DOS
    MOV AH,4Ch        ; Función para terminar un proceso
    INT 21h           ; Transferir a MS-DOS
CODE ENDS  ; Fin del segmento CODE

STACK SEGMENT PARA STACK 'STACK' ; Def. del segmento STACK
    DW 200h DUP(0)  ; Define un espacio de 1 Kb.
                      ; 512 Words.
STACK ENDS ; Fin del segmento STACK
END ENTRY  ; ENTRY será el punto de entrada del programa EXE
```

Para ensamblar este programa:

```
A>MASM PROG1COM;
```

Encadenar :

```
A>LINK PROG1COM;
```

Programa EXE que imprime un mensaje en la pantalla (Declaración de segmentos en la forma simplificada)

```
; Este es un programa EXE pero la declaración de segmentos
; en la forma simplificada

 .MODEL SMALL         ; Establecer modelo de memoria SMALL
 .DATA                ; Def. del segmento DATA
    SALUDO DB 'Este es un programa EXE $' ; Def. de datos
                      ; Fin del segmento DATA

 .CODE                ; Def. del segmento CODE
ENTRY:                ; Punto de entrada del programa
    MOV AX,@DATA      ; AX tiene la dirección del segmento DATA
    MOV DS,AX         ; DS tiene la dirección del segmento DATA
```

```
    MOV AH,9          ; Servicio 9 Impr. un string terminado en
'$'
    MOV DX,OFFSET SALUDO ; OFFSET SALUDO será reemplazado por
el
                      ; ensamblador por la dirección del offset
                      ; donde este la etiqueta SALUDO.
    INT 21h           ; Transferir a MS-DOS
    MOV AH,4Ch        ; Función para terminar un proceso
    INT 21h           ; Transferir a MS-DOS
                      ; Fin del segmento CODE

 .STACK                ; Def. del segmento STACK
    DW 200h DUP(0)    ; Define un espacio de 1 Kb.
                      ; 512 Words.
                      ; Fin del segmento STACK
END ENTRY  ; ENTRY será el punto de entrada del programa EXE
```

Para ensamblar este programa:

```
A>MASM PROG1COM;
```

Encadenar :

```
A>LINK PROG1COM;
```

REGISTROS DEL 80x86

Registros de Segmentos:

CS - Segmento de Código
DS - Segmento de Datos
ES - Segmento Extra
SS - Segmento de Stack

Registros de Propósito General

AX - Acumulador (16 bits)
AH - Parte alta de AX (8 bits)
AL - Parte baja de AX (8 bits)

BX - Base (16 bits)
BH - Parte alta de BX (8 bits)
BL - Parte baja de BX (8 bits)

CX - Contador (16 bits)
CH - Parte alta de CX (8 bits)
CL - Parte baja de CX (8 bits)

DX - Datos (16 bits)
DH - Parte alta de DX (8 bits)
DL - Parte baja de DX (8 bits)

Registros Indices

SI - Indice fuente
DI - Indice destino

Registros de Stack

SP - Apuntador al stack
BP - Apuntador a la base

IP - Contador del programa

FLAGS - Registro de banderas.

Bit 0 - Carry : Esta bandera esta en 1 después de una operación, el resultado excede el tamaño del operando.

Ejemplo:

```
 11111111 +
 00000001
 ________
100000000
```

El resultado no cabe en 8 bits por lo tanto CF=1

Bit 2 - Paridad : Esta bandera esta en 1 cuando el número de bits en 1 en el byte de menor orden es par, después de una operación.

Ejemplo:

```
 00000010 +
 00000001
 ________
 00000011
```

Número par de unos por lo tanto PF=1

Bit 4 - Auxiliar : Utilizada para operaciones con números BCD.

Bit 6 - Cero : Esta bandera esta en 1 si el resultado de una operación es cero.

Ejemplo:

```
 11111111 +
 00000001
 ________
100000000
```

El resultado final es cero por lo tanto ZF=1.

Bit 7 - Signo : Esta bandera es 1 si después de una operación el bit mas significativo esta en 1.

Ejemplo:

```
01111111 +
00000001
__________
10000000
```

El bit mas significativo es 1 por lo tanto SF=1

Bit 11 - Overflow : Si el resultado de una oración de números con signo esta fuera del rango.

Ejemplo:

```
01111111 +      = 127 +
00000001        =   1
__________
10000000        = 128
```

Rango de números de 8 bits con signo es -128..127, como 128 esta fuera del rango OF=1.

DATOS EN EL 80x86

Constantes

Para definir valores constantes dentro de un programa es necesario que estos sean caracteres encerrados en comillas, o que comiencen con un dígito (constantes hexadecimales).

Caracter : Cualquier valor numérico, representado por su correspondiente caracter encerrado en comillas

'A', '2', '0'

Decimal : Cualquier valor numérico, opcionalmente se le puede agregar una **d** al final para indicar que es decimal.

10, 20, 12345d, 76d, 12345, 20d

Hexadecimal : Cualquier valor numérico representado en hexadecimal, pero al final debe indicarse **h**.

10h, 20h, 0B800h, 1Ah, 1234h

Octal : Cualquier valor numérico representado en octal, pero al final debe indicarse **o**.

10o, 777o, 12345o, 40o

Binario : Cualquier valor numérico representado en binario, pero al final debe indicarse **b**.

1110001b, 101b, 11001100b

Definición de datos

DB - Definir bytes
DW - Definir words
DD - Definir doublewords

DQ - Definir quadwords
DT - Definir grupos continuos de 16 bytes

Ejemplos:

```
DB 'A' ; Definir un byte.
DB 65
DB 41h
DB 101o
DB 1000001b
DB 'ABCDE' ; Definir una cadena de bytes.
DB 'A','B','C','D','E'
DB 65,66,67,68,69
DB 'ABC',68,'E'
DW 1234h ; Definir una palabra
DB 34h,12h ; Lo mismo que el caso anterior, ya que los
; valores enteros almacenan primero el byte ;
menos significativo y después el byte mas ;
significativo

DB ? ; Definir un byte, sin indicar un valor en especifico
; puede ser cualquiera.
```

Directiva **DUP**( ) : esta directiva precedida de un número indica al MASM que reserve el número de bytes necesarios para almacenar el número de veces indicado antes de DUP( ), el valor indicado dentro del paréntesis.

```
DB 64 DUP(0) ; Esto indica que se reserve un espacio de 64
; bytes inicializados con 0.

DB 2 DUP('HOLA') ; Este reservara el espacio suficiente para
; almacenar 2 cadenas incializadas con los
; bytes 'HOLA'.

DB 512 DUP(?) ; Reservar 512 bytes sin importar que valor
; tengan.

DW 100h DUP(0) ; Reservar 256 words, o 512 bytes
; inicializados con 0.
```

Los números siempre almacenan primero los bytes menos significativos, hasta los bytes mas significativos, por lo tanto, las siguientes definiciones de bytes serian equivalentes:

```
DB 01h,23h,45h,67h,89h,1Ah,0BCh,0DEh
DW 2301h , 6745h , 1A89h , 0DEBCh
DD 67452301h  , 0DEBC1A89h
DQ 0DEBC1A8967452301h
DT 0123456789ABCDEF0123h
DB 23h,01h,0EFh,0CDh,0ABh,89h,67h,45h,23h,01h
```

Generación de código

Las definiciones de datos dentro de un programa generan código, así como las instrucciones de lenguaje maquina. Las directivas no generan código, pero estas indican al MASM como generarlo y almacenan cierta información para el LINK dentro del archivo OBJ.

Ejemplo de generación de código de una instrucción :

MOV BX,5

El código de operación de MOV es el siguiente :

1011 w reg	dato	datos si w=1

W=1 si es un registro de 16 bits, y los 3 bits del registro son los siguietes:

```
000 AX    001 CX    010 DX    011 BX
000 SP    101 BP    110 SI    111 DI
```

W=0 si es un registro de 8 bits, y los 3 bits del registro son los siguientes:

```
000 AL    001 CL    010 DL    011 BL
100 AH    101 CH    110 DH    111 BH
```

Registros de segmento:

```
     00 ES     01 CS     10 SS     11 DS
```

Por lo tanto la instrucción MOV BX,5 generara lo siguiente:

1011 1 011 00000101 00000000
MOV BX , 05 (byte mas significativo es 0)

Los bytes son los siguientes

0BBh 05h 00h

Por lo tanto :

MOV BX,5

equivale a :

DB 0BBh,5,0

Ejemplo de la generación de código de la instrucción:

PUSH BP

El código de operación de PUSH es el siguiente:

BP= 101

01010 101

Esta instrucción es de un solo byte, que es el siguiente :

55h

Por lo tanto si definimos:

DB 55h

equivale a:

PUSH BP

Ejemplo de la generación de código de la instrucción **INT** para ejecutar un interrupt de software.

El código de operación de INT es el siguiente:

Por lo tanto la instrucción **INT 5** generará lo siguiente :

11001101 00000101

Los bytes serán los siguientes :

0CDh,5

La instrucción **INT 20h** generara los siguientes bytes :

0CDh,20h

Por lo tanto un programa con las siguientes instrucciones :

```
INT 5
INT 20h
```

Generara los siguientes bytes :

0CDh,5,0CDh,20h

Si nosotros escribimos directamente en un archivo estos bytes, estaremos escribiendo un programa **COM**, el cual realiza un **INT 5** que realiza una impresión de lo que esta en pantalla, y un **INT 20h** función de DOS para terminar un programa **COM**.

Ejemplo:

Conversión a decimal de los bytes :

0CDh = 205
5 = 5
0CDh = 205
20h = 32

Si creamos el archivo :

```
C>COPY CON PRNTSCR.COM
 _ ^Z

C>PRNTSCR
```

Definición de etiquetas para referencias a datos

Para direcciónar datos de la memoria también es posible hacerlo por medio de etiquetas, esto no evitara especificar la dirección donde se encuentra el dato de la memoria, ejemplo:

```
LISTA DB 0,1,2,3,4,5

SALUDO DB  'Esta es una prueba de un programa $'

VAR1 DW 0

BUF1 DW 5 DUP(0)
```

OFFSET este operador me permite obtener la dirección de una etiqueta, el ensamblador remplazara:

OFFSET ETIQUETA

Por el valor de la dirección de ETIQUETA.

Ejemplos:

```
SAL DB 'Hola a todos $'

MOV AL,SAL ; Guarda en AL el caracter contenido en el
; offset de la etiqueta SAL del segmento DS

MOV DX,OFFSET SAL ; DX almacenará la dirección del
; desplazamiento de SAL.
```

Modos de direcciónamiento

Registro: Tal como indicar el valor contenido en un registro como:

AX, BX, CX, etc.

Ejemplo: Transferencias de registro a registro, la ejecución de estas instrucciones se realiza dentro del CPU.

```
MOV AX,CX
MOV DX,SI
MOV AH,CL
```

Inmediato: Una constante numérica que será guardada en un registro o una localidad en la memoria.

```
COLOR EQU 4 ; Con la directiva EQU definimos un nombre a una
                ; constante

X DB 0
Y DW 0

MOV AX,5 ; Guardar 5 al registro AX
MOV BX,COLOR ; Guardar 4 al registro BX
MOV BYTE PTR X,COLOR ; Guardar 4 al BYTE que se encuentra en
; la dirección de la etiqueta X.
MOV WORD PTR Y,10 ; Guardar 10 a la palabra que se encuentra
                        ; en la dirección de la etiqueta Y.
```

Directo : En el direcciónamiento directo movemos datos de memoria a registro o de un registro a memoria.

```
X DB 0
Y DW 0

MOV AL,X  ; Del byte que esta en el offset de la etiqueta
; X del segmento DS al registro AL.

MOV AX,Y  ; De la palabra que esta en el offset de la
; etiqueta Y del segmento DS al registro AX.

MOV X,AH ; Del registro AH, al byte que esta en el offset
; de la etiqueta X del segmento DS.
```

```
MOV Y,AX ; De AX a la palabra que se encuentra en el
; offset de la etiqueta Y del segmento DS.
```

Indirecto : En el direcciónamiento indirecto direcciónamos memoria utilizando los registros:

Nota : Todas las operaciones de acceso a memoria se asume que son en el segmento DS, pero cuando se utiliza el registro **BP** estas serán en el segmento SS.

Registro Indirecto :

```
LISTA DB 0,1,2,3,4,5

MOV BX,OFFSET LISTA  ; Inmediato
MOV BP,SP ; Registro

MOV AL,[BX] ; Registro indirecto, estamos obteniendo el byte
;que se encuentra en el offset indicado en
; BX del segmento DS, en este caso el primer
; byte de LISTA.

MOV [BX],AX ; Guardar en la palabra que se encuentra en el
;offset indicado por BX del segmento DS, el
; valor de 0.

MOV AL,[BP] ; Estamos obteniendo el byte que se encuentra
; en el offset indicado en BP del segmento SS,
; en este caso el primer  byte de LISTA.

MOV [BP],AX ; Guardar en la palabra que se encuentra en el
; offset indicado por BP del segmento SS, el
;  valor de 0.
```

Base relativo :

```
LISTA DB 0,1,2,3,4,5

MOV BX,2 ; Inmediato
MOV BP,SP ; Registro

MOV AH,LISTA[BX] ; AH tomara el valor que se encuentra en el
```

```
; offset de LISTA mas el número de bytes
; indicados en BX, del segmento DS
```

Otra forma de hacer la misma asignación es la siguiente:

```
MOV AH,[BX+LISTA]


MOV AH,LISTA[BP] ; AH tomara el valor que se encuentra en el
; offset de LISTA mas el número de bytes
; indicados en BP, del segmento SS
```

Otra forma de hacer la misma asignación es la siguiente:

```
MOV AH,[BP+LISTA]
```

Directo indexado :

```
LISTA DB 1,2,3,4,5,6

MOV SI,2 ; Inmediato
MOV AL,LISTA[SI] ; AL guarda el byte que se encuentra en el
; offset de la etiqueta LISTA mas el
; número de bytes indicados por SI, del
; segmento DS
```

Otra forma de hacer la misma asignación es la siguiente:

```
MOV AL,[SI+LISTA]
```

Base indexado :

```
LISTA DB 0,1,2,3,4,5,6,7

MOV BX,OFFSET LISTA ; Inmediato
MOV SI,3            ; Inmediato
MOV BP,SP           ; Registro

MOV AL,[BX+SI] ; AL almacenará el byte que se encuentra en el
; offset BX+SI del segmento DS

     MOV AL,[BP+SI]   ; AL almacenará el byte que se encuentra
en                          el offset BP+SI del segmento SS
```

Acceso a datos en otros segmentos

Cuando se desean accesar memoria en otro segmento que no sea **SS**, ni **DS**, puede indicarse el segmento deseado, ejemplo:

```
MOV ES:[BX],AL  ; Guardar el valor de AL al segmento ES,
; offset indicado en BX

MOV CS:Y,AL ; Guardar en el segmento CS offset de la
;  etiqueta Y el valor de AL
```

INSTRUCCIONES DEL 80X86

Instrucciones de Transferencias de Datos

XCHG - Intercambia el contenido de los operandos fuente y destino.

Ejemplos:

```
XCHG AX,BX     ; Intercambiar los valores en AX y BX
XCHG CH,CL     ; Intercambiar los valores en CH y CL
XCHG AX,VAR    ; Intercambiar el valor de AX con la palabra
; que se encuentra en memoria VAR.
```

PUSH - Colocar en el stack el valor del operando, este operando debe ser una palabra, las operaciones realizadas son :

SP <- SP-2
SS:[SP] <- operando

Ejemplos:

```
PUSH AX   ; Guardar en el stack el valor en AX
PUSH DS   ; Guardar en el stack el valor en DS
PUSH WORD PTR VAR   ; Guardar en el stack el valor que esta
; en una dirección de memoria.
```

PUSHF - Colocar en el stack una copia del registro de banderas.

Ejemplo:

```
PUSHF     ; Guarda las banderas en el stack.
```

POP - Quitar una palabra del stack y guardarla en el operando destino indicado, las operaciones realizadas son:

operando **<-** SS:[SP]
SP **<-** SP+2

Ejemplos:

```
POP AX
POP BX
POP WORD PTR VAR
```

POPF - Quitar una palabra del stack y situarla en el registro de banderas.

Ejemplo:

```
POPF
```

LEA - Transfiere la dirección del offset del operando fuente al operando destino.

Ejemplos:

```
ETQ DB 0

LEA DX,ETQ ; DX contiene la dir. del offset de ETQ.

MOV BX,200h
LEA SI,[BX] ; SI contiene el valor 200h
```

LDS - Carga el registro DS, y el registro indicado, con la dirección que se encuentra en la dirección especificada.

Ejemplo:

```
VIDEOADDR DD 0B8000000h

LDS SI,VIDEOADDR ; DS contiene la dir. del seg. 0B800h, y SI
; la dir. del offset 0000h.
```

LES - Carga el registro ES, y el registro indicado con la dirección que se encuentra en la dirección especificada.

Ejemplo:

```
LES DI,VIDEOADDR ; ES <- 0B800h, DI <- 0000h
```

LAHF - Copia el byte de menor orden del registro de banderas al registro AH.

Ejemplo:

```
      LAHF
```

SAHF - Copia el contenido del registro AH, en el byte de menor orden del registro de banderas.

Ejemplo:

```
SAHF
```

Instrucciones para operaciones aritmeticas

ADD - Añade el contenido deloperando fuente al operando destino.

Ejemplos:

```
ADD AX,BX       ; AX=AX+BX
ADD AH,AL       ; AH=AH+AL
ADD AX,VAR      ; AX=AX+VAR
ADD AX,5        ; AX=AX+5
ADD BYTE PTR VAR,0Ah; VAR=VAR+10
```

SUB - Resta el contenido del operando fuente del operando destino.

Ejemplos:

```
SUB AX,BX       ; AX=AX-BX
SUB AX,VAR      ; AX=AX-VAR
SUB WORD PTR VAR,10 ; VAR=VAR-10
SUB CX,5        ; CX=CX-5
```

INC - Incrementa en 1 el valor del operando.

Ejemplos:

```
INC AX    ; AX=AX+1
INC CL    ; CL=CL+1
INC WORD PTR VAR    ; VAR=VAR+1
```

DEC - Decrementa en 1 el valor del operando.

Ejemplos:

```
DEC AX    ; AX=AX-1
DEC BYTE PTR VAR    ; VAR=VAR-1
```

```
DEC CL    ; CL=CL-1
```

NEG - Calcula el complemento a dos del operando y almacena el resultado en el mismo operando.

Ejemplos:

```
NEG AX    ; AX=0-AX
NEG BL    ; BL=0-BL
NEG BYTE PTR VAR    ; VAR=0-VAR
```

MUL - Multiplica enteros sin signo.

Caso 1- Si el operando es un byte, multiplica el contenido de AL por el contenido del operando y almacena el resultado en AX.

AX=AL*operando

Ejemplo:

```
MOV AL,3
MOV BL,100
MUL BL    ; AX=3 * 100  AX=300
```

Caso 2-Si el operando es una palabra multiplica el contenido de AX por el operando, y el resultado será almacenado en los registros DX:AX.

DX:AX=AX*operando

Ejemplo:

```
MOV AX,1000h
MOV BX,100h
MUL BX    ; DX:AX=1000h * 100h
          ; DX = 10h     AX=0
```

IMUL - Multiplica enteros con signo.

Caso 1- Si el operando es un byte, multiplica el contenido de AL por el contenido del operando y almacena el resultado en AX.

AX=AL*operando

Ejemplo:

```
MOV AL,1
NEG AL    ; AL=-1
MOV BL,2
NEG BL    ; BL=-2
IMUL BL    ; AX=-1 * -2  AX=2
```

Caso 2-Si el operando es una palabra multiplica el contenido de AX por el operando, y el resultado será almacenado en los registros DX:AX.

DX:AX=AX*operando

DIV - Divide números enteros sin signo.

Caso 1- Si el operando es un byte, divide el contenido de AX entre el contenido del operando, almacena el resultado en AL y el residuo en AH.

AL=AX/operando
AH=AX MOD operando

Ejemplo:

```
MOV AX,10
MOV BL,3
DIV BL    ; AL=3    AH=1
```

Caso 2-Si el operando es una palabra divide el contenido de DX:AX entre el operando, el resultado será almacenado en AX y el residuo será almacenado en DX.

AX=DX:AX/operando

Ejemplo:

```
MOV DX,0
MOV AX,12345
MOV BX,1000
DIV BX    ; AX=12345/1000     DX=12345 MOD 1000
; AX=12              DX=345
```

IDIV - Divide números enteros con signo.

Caso 1- Si el operando es un byte, divide el contenido de AX entre el contenido del operando, almacena el resultado en AL y el residuo en AH.

AL=AX/operando
AH=AX MOD operando

Ejemplo:

```
MOV AX,-2
MOV BL,-1
IDIV BL    ; AL=2    AH=0
```

Caso 2-Si el operando es una palabra divide el contenido de DX:AX entre el operando, el resultado será almacenado en AX y el residuo será almacenado en DX.

AX=DX:AX/operando

Ejemplo:

```
MOV DX,0
MOV AX,12345
MOV BX,1000
IDIV BX   ; AX=12345/1000     DX=12345 MOD 1000
; AX=12              DX=345
```

Instrucciones para manipulacion de bits

XOR - Realiza un XOR logico de los operandos y almacena el resultado en el operando destino.

Ejemplos:

```
XOR AX,BX
XOR AX,AX ; Hacer AX=0
XOR AX,0Fh ; Invertir los 4 bits menos significativos de AX
XOR AX,[BX+VAR]
XOR BYTE PTR [BX],0Fh
```

NOT - Invierte los bits del operando destino.

Ejemplos:

```
NOT CL
NOT AX
NOT WORD PTR VAR
```

AND - Realiza un AND logico de los operandos y almacena el resultado en el operando destino.

Ejemplos:

```
AND BX,AX
AND AX,VAR      ; Se toma una palabra de la VAR
AND VAR,BX
AND CL,0Fh      ; Apagar los 4 bits mas significativos de CL
```

OR - Realiza un OR logico de los operandos y almacena el resultado en el operando destino.

Ejemplos:

```
OR AX,CX
OR CL,0F0h      ; Prender los 4 bits mas significativos de CL
OR DX,TEMP
OR AL,AH
```

SHL - Desplaza los bits del operando destino a la izquierda.

Ejemplos:

```
SHL AX,1

MOV CL,4
SHL BL,CL ; Si es mas de un bit debe utilizarse CL para
;  indicar el número de bits a desplazar.
```

SHR - Desplaza los bits del operando destino a la derecha.

Ejemplos:

```
SHR AX,1

MOV CL,4
SHR BL,CL ; Si es mas de un bit debe utilizarse CL para
;  indicar el número de bits a desplazar.

     SHR BYTE PTR VAR,4
```

ROL - Rota a la izquierda todos los bits del operando destino, tantos bits como lo indique el operando fuente.

Ejemplos:

```
MOV AX,8000h ; El bit mas significativo en 1
ROL AX,1     ; AX toma el valor de 1

MOV CL,2     ; Si es mas de 1 bit, se debe de indicar en CL.
ROL AX,CL

ROL BYTE PTR VAR,CL
```

ROR - Rota a la derecha todos los bits del operando destino, tantos bits como lo indique el operando fuente.

Ejemplos:

```
MOV AX,1      ; El bit menos significativo en 1
ROR AX,1      ; AX toma el valor de 8000h

MOV CL,2      ; Si es mas de 1 bit, se debe de indicar en CL.
ROR AX,CL

ROR BYTE PTR VAR,CL
```

Instrucciones de control

Saltos

JMP -Hace un salto incondicional a una dirección especifica.

Formatos del JMP

JMP etiqueta Asume que la etiqueta se encuentra en el mismo segmento de codigo, la etiqueta debe de estar entre -32768 y 32767 del actual IP.

JMP FAR PTR etiqueta La etiqueta se encuentra en otro segmento, CS e IP tomaran el segmento y offset de la etiqueta.

JMP registro El contenido del registro será copiado en IP.

JMP WORD PTR dir Salta a la dirección indicada en la dirección DS:dir.

JMP DWORD PTR dir Salta a la dirección indicada por la palabra doble que especifica segmento y offset, que se encuentra en la dirección DS:dir

Ejemplos:

```
JMP INICIO2

    ETQ1 DW OFFSET DOS  ; En esta dirección esta el offset ;
    ;donde esta la etiqueta2

    ETQ2 DD 12345678h ; En esta dirección esta una palabra
    ; doble que indica otra dirección.

INICIO2:
    MOV AX,OFFSET UNO
    JMP AX

UNO:
    JMP WORD PTR ETQ1

DOS:
    JMP DWORD PTR ETQ2
```

Ejemplo de un salto hacia otro segmento

```
DATA SEGMENT PARA PUBLIC 'DATA'
    ;
DATA ENDS

STACK SEGMENT PARA PUBLIC 'STACK'
    ;
STACK ENDS

CODE1 SEGMENT PARA PUBLIC 'CODE'
    ASSUME CS:CODE1,DS:DATA,ES:DATA,SS:STACK
ENTRY:
    ;
    JMP FAR PTR SALIDA
    ;
CODE1 ENDS

CODE2 SEGMENT PARA PUBLIC 'CODE'
    ASSUME CS:CODE2
    ;
SALIDA:
    MOV AH,4Ch
    INT 21h

CODE2 ENDS
END ENTRY
```

Saltos condicionales

CMP - Esta instruccion es considerada como una instruccion aritmetica, ya que sustrae el operando fuente del operando destino, solo que el resultado no se almacena en el operando destino, el resultado se utiliza solo para activar las banderas.

Ejemplos :

```
CMP AX,BX
CMP AX,VAR
CMP AX,0
CMP BYTE PTR VAR,1
```

Comparaciones de números sin signo

JA o JNBE - Salta si esta por arriba de, o lo mismo si no esta por debajo o igual de. (CF=0 and ZF=0).

Ejemplo:

```
CMP AL,'A'      ; Comparar AL con 65
JA ETIQUETA     ; Salta si esta por arriba de
```

JAE o JNB - Salta si esta por arriba o es igual, o lo mismo si no esta por debajo de. (CF=0).

Ejemplo:

```
CMP AL,65       ; Comparar AL con 65
JAE ES_MAY      ; Salta si esta por arriba o es igual.
```

JB o JNAE - Salta si esta por debajo de, o salta si no esta por arriba ni es igual. (CF=1).

Ejemplo:

```
CMP AL,10       ; Compara AL con 10
JB UN_DIGITO    ; Salta si esta por debajo de.
```

JBE o JNA - Salta si esta por debajo de o es igual, o salta si no esta por arriba de. (CF=1 or ZF=1).

Ejemplo:

```
CMP AL,'Z'      ; Compara AL con 'Z'
JBE ES_MAY      ; Salta si esta por debajo o es igual.
```

JE o JZ - Salta si es igual. (ZF=1), esta instruccion tambien se aplica para comparaciones de enteros con signo.

Ejemplo:

```
CMP AL,'S'      ; Compara AL con 'S'
JE ESTA_BIEN    ; Salta si es igual.
```

JNE o JNZ - Salta si no es igual. (ZF=0). Esta instruccion tambien se aplica para comparacion de enteros con signo.

Ejemplo:

```
     CMP AL,0
     JNZ ES_UNO
     ;
ES_UNO:
     CMP AL,1
     JNZ ES_DOS
     ;
ES_DOS:
```

Ejemplo :

```
 ; Programa ejemplo que convierte una cadena a mayusculas
DATA SEGMENT PARA PUBLIC 'DATA'
    CAD DB 30,0,30 DUP(0) ; Buffer para almacenar la cadena
DATA ENDS

CODE SEGMENT PARA PUBLIC 'CODE'
    ASSUME CS:CODE,DS:DATA,ES:DATA,SS:STACK
INICIO:
    MOV AX,DATA
    MOV DS,AX  ; DS <- Segmento DATA
```

```
    MOV AH,0Ah ; Funcion leer un string del teclado
    MOV DX,OFFSET CAD    ; DX <-Offset del string
    INT 21h    ; Llamar a DOS
    ;
    MOV BX,OFFSET CAD ; BX <-Offset del string
OTRO_CAR:
    CMP BYTE PTR [BX],'a' ; Comparar el car. en DS:BX con 'a'
    JB NO_MIN             ; Si esta por debajo de, salta a
NO_MIN
    CMP BYTE PTR [BX],'z' ; Comparar el car. en DS:BX con 'z'
    JA NO_MIN             ; Si esta por arriba, de salta a
NO_MIN
    ; si es mayuscula
    SUB BYTE PTR [BX],32  ; Convertir a minuscula
NO_MIN:
    INC BX     ; BX apunta al siguiente caracter
    CMPBYTEPTR [BX],13 ;Si este es un CR, aqui terminoel Str.
    JNZ OTRO_CAR ; Si no fue igual a 13, salta a OTRO_CAR
    ;
    MOV BYTE PTR [BX],'$' ;Indicar el terminador donde estael
CR
    ;
    MOV AH,9   ; Funcion para imprimir un string
    MOV DX,OFFSET CAD+2  ; DX <- Offset de CAD mas 2 bytes
    INT 21h    ; Llamar a DOS
    ;
    MOV AH,4Ch ; Funcion para terminar un programa
    INT 21h    ; Llamar a DOS
CODE ENDS

STACK SEGMENT PARA STACK 'STACK'
    DB 100h DUP(0)  ; Espacio para el Stack
STACK ENDS
END INICIO
```

Comparaciones para enteros con signo

JG o JNLE - Salta si es mayor, lo mismo, salta si no es menor ni igual. (ZF=0 or OF=SF).

Ejemplo:

```
MOV AL,1
CMP AL,0FFh    ; Comparar AL con -1
```

```
JG ES_MAYOR    ; Salta si es mayor.
```

JGE o JNL - Salta si es mayor o igual, lo mismo, salta si no es menor. (SF=OF).

Ejemplo:

```
CMP AX,0  ; Comparar AX con 0.
JGE ES_POSITIVO     ; Salta si es mayor o igual que 0.
```

JL o JNGE -Salta si es menor, lo mismo, salta si no es mayor ni igual. (SF<>OF).

Ejemplo:

```
CMP AX,0  ; Comparar AX con 0.
JL ES_NEGATIVO      ; Salta si es menor que 0.
```

JLE o JNG -Salta si es menor o igual, lo mismo, salta si no es mayor. (ZF=1 or SF<>OF).

Ejemplo:

```
CMP AX,0FFFFh  ; Comparar AX con -1
JLE ES_NEGATIVO     ; Salta si es menor o igual.
```

Verificando el estado de las banderas

JC	Salta si la bandera del carry es 1 (CF=1).
JNC	Salta si la bandera del carry es 0 (CF=0).
JO	Salta si la bandera del overflow es 1 (OF=1).
JNO	Salta si la bandera del overflow es 0 (OF=0).
JNP o JPO	Salta si no hay paridad (PF=0). Esto significa paridad par.
JP o JPE	Salta si hay paridad (PF=1). Esto significa paridad impar.
JS	Salta si la bandera del signo es 1 (SF=1).
JCXZ	Salta si el contenido de CX es 0.

Ciclos

LOOP - Decrementa CX, si CX es diferente de 0 entonces salta a la dirección indicada despues de LOOP.

Ejemplo:

```
     MOV CX,26       ; Número de veces a ejecutar el ciclo.
     MOV AL,'A'
CICLO:
     MOV AH,0Eh      ; Funcion de BIOS imprimir un caracter
                     ; AL el caracter a imprimir
     INT 10h         ; Llamar a BIOS.
     LOOP CICLO
```

LOOPE o LOOPZ - Decrementa CX, si CX es diferente de 0 y la bandera del cero esta activada (ZF=1), entonces efectua la transferencia de control.

Ejemplo:

```
     MOV CX,255      ; Maximo de veces a repetir
     MOV AL,0        ; AL =0
OTRO_CAR:
     MOV AH,0Eh      ; Funcion impr. el caracter en AL
     INT 10h         ; Llamar a BIOS
     INC AL          ; Incrementa AL
     ;
     MOV AH,00 ; Funcion esperar un caracter del teclado
     INT 16h   ; llamar a BIOS
     CMP AL,'S' ; Compara el caracter presionado con 'S'
     LOOPZ OTRO_CAR ; Si es igual, salta a OTRO_CAR
```

LOOPNE o LOOPNZ - Decrementa CX, si CX es diferente de 0 y la bandera del cero esta inactivada (ZF=0), entonces efectua la transferencia de control.

Ejemplo:

```
   MSG DB 'Esta es una prueba ',0

   MOV BX,OFFSET MSG
   MOV CX,80 ; 80 caracteres, maximo a imprimir.
OTRO_CAR:
   MOV AH,0Eh      ; Funcion del BIOS para imprimir caracter en
                   ;  AL
   MOV AL,[BX]     ; AL es el caracter contenido en la dir
DS:BX.
   INT 10h         ; llamar a BIOS
   INC BX          ; Pasar al sig. caracter.
   CMP BYTE PTR [BX],0 ; Comparar si este no es un 0.
```

```
   LOOPNZ OTRO_CAR ; Salta si no es cero a OTRO_CAR
```

Ejemplo : Programa que convierte a mayusculas ahora utilizando LOOP

```
DATA SEGMENT PARA PUBLIC 'DATA'
    CAD DB 30,0,30 DUP(0)  ; Definir espacio para el string
DATA ENDS

CODE SEGMENT PARA PUBLIC 'CODE'
    ASSUME CS:CODE,DS:DATA,ES:DATA,SS:STACK
INICIO:
    MOV AX,DATA
    MOV DS,AX  ; DS <- Segmento DATA
    MOV AH,0Ah ; Funcion para leer un string
    MOV DX,OFFSET CAD    ; DX <-Offset del string
    INT 21h    ; Llamar a MS-DOS
    ;
    MOV BX,OFFSET CAD+2  ; BX <-Offset del 3er caracter del
                         ; string
    XOR CH,CH  ; Parte alta de CX con 0
    MOV CL,[CAD+1]  ; Parte baja de CX con el número de cars.
que                      ; fueron leidos del teclado
OTRO_CAR:
    CMP BYTE PTR [BX],'a' ;Compara el car. en DS:BX con 'a'
    JB NO_MIN             ; salta si esta por debajo de
    CMP BYTE PTR [BX],'z' ; Compara el car en DS:BX con 'z'
    JA NO_MIN             ; salta si esta por arriba de
    ; si es mayuscula
    SUB BYTE PTR [BX],32  ; Convertir a mayuscula
NO_MIN:
    INC BX          ; Incrementar BX al sig. Caracter
    LOOP OTRO_CAR   ; Hacer el ciclo a OTRO_CAR
    ;
    MOV BYTE PTR [BX],'$' ; Poner el terminador al final del
                          ; string
    MOV AH,9        ; Funcion imprimir un string
    MOV DX,OFFSET CAD+2  ; DX Offset del 3er car. del string
    INT 21h         ; Llamar a DOS
    ;
    MOV AH,4Ch      ; Funcion para terminar proceso
    INT 21h         ; Llamar a DOS
CODE ENDS

STACK SEGMENT PARA STACK 'STACK'
    DB 100h DUP(0)
STACK ENDS
```

```
END INICIO
```

CALL -Guarda en el stack la dirección de la siguiente instruccion, y despues salta a la dirección especificada por el operando.

Formatos del CALL

CALL etiqueta Asume que la etiqueta se encuentra en el mismo segmento de codigo, la etiqueta debe de estar entre -32768 y 32767 del actual IP.

CALL FAR PTR etiqueta La etiqueta se encuentra en otro segmento, CS e IP tomaran el segmento y offset de la etiqueta.

CALL registro El contenido del registro será copiado en IP.

CALL WORD PTR dir Salta a la dirección indicada en la dirección DS:dir.

CALL DWORD PTR dir Salta a la dirección indicada por la palabra doble que especifica segmento y offset, que se encuentra en la dirección DS:dir

Nota: Para llamadas dentro del mismo segmento (CALL cercanos) , el offset de la siguiente instruccion despues del CALL constituyen la dirección de retorno, esto es que solo se guarda en el stack el IP. Para llamadas a fuera del segmento (CALL lejanos) , el actual CS e IP son guardados en el stack como la dirección de retorno.

Directivas para definir procedimientos

PROC - Define el inicio del procedimiento numbrado. La palabra reservada NEAR o FAR (opcional) indica como será accesado este procedimiento, solo con el offset o segmento y offset.

Sintaxis:

```
nomproc PROC [ NEAR | FAR ]
    ........
    RET    ; Regresa a la dir. despues de donde se llamo
nomproc ENDP
```

Si no se especifica la palabra NEAR o FAR, las llamadas a los procedimientos serán de acuerdo al modelo de memoria especificado, y si la declaracion de segmentos fue en la forma extendida entonces se asumira que son NEAR.

RET RETN RETF - Esta instruccion se utiliza para transferir el control de la dirección que se encuentra almacenada en el stack. Esta dirección es usualmente guardada por una instruccion CALL.

RETN - Regreso cercano, solo obtiene una palabra del stack, transfiere el control, al offset almacenado en la palabra.

RETF - Regreso lejano, obtiene dos palabras del stack, transfiere el control, al segmento y offset indicados en las palabras.

RET - En este, el MASM determina la codificacion del RET como cercano o lejano, dependiendo del calificador asociado al procedimiento NEAR o FAR.

Ejemplos:

```
     CALL FAR PTR UNO  ; Este es un CALL lejano.


UNO PROC FAR
    .......
    .......
    .......
    RET    ; Este finalmente será un RETF
UNO ENDP



     CALL DOS  ; Este es un  CALL cercano.


DOS PROC NEAR
    .......
    .......
    .......
    RET   ; Este será finalmente un RETN
DOS ENDP


    MOV AX,OFFSET UNO
    CALL AX   ; CALL a la dirección indicada en AX

UNO PROC NEAR
    .......
    .......
    .......
    RET
UNO ENDP
```

Programa que imprime los números del 0000 al 01FFF en ; hexadecimal

```
CODE SEGMENT
    ASSUME CS:CODE
    ORG 100h

INICIO PROC NEAR
    MOV CX,200h     ; Número de veces a repetir el ciclo
    XOR DX,DX       ; DX = 0
```

```
CICLO:
    CALL WRITEHEX   ; Imprimir el valor en DX
    CALL WRITECRLF  ; Hacer un salto de linea
    INC DX          ; Incrementar DX
    LOOP CICLO      ; Regresar al ciclo
    ;
    INT 20h         ; Fin del programa
INICIO ENDP


WRITEHEX PROC NEAR
    ; Imprime en Hexadecimal.
    ; Entrada:
    ;   DX palabra con el valor a mostrar
    ; Llama:
    ;   WRITEHEX_DIG
    PUSH AX         ; Guardar en el stack los registros que se
    PUSH CX         ; utilizaran en el procedimiento, para no
    PUSH DX         ; perder sus valores
    ;
    MOV CX,4        ; CX = 4 número de digitos a imprimir, el
OTRO_DIG:           ; ciclo se ejecuta una vez por digito
    PUSH CX         ; Guarda el contador del ciclo
    MOV CL,4
    ROL DX,CL       ; Rotar el número 4 bits a la derecha
                    ; DX contiene en su nibble mas bajo, el
dig.
    CALL WRITEHEX_DIG   ; Imprimir el digito
    POP CX          ; Recuperar el contador del ciclo
    LOOP OTRO_DIG   ; Hacer el ciclo
    ;
    POP DX          ; Recuperar los registros guardados en
    POP CX          ; el stack
    POP AX
    RET             ; Regresar del procedimiento
WRITEHEX ENDP

WRITEHEX_DIG PROC NEAR
    ; Escribir un digito hexadecimal
    ; Entrada:
    ;   DL el nibble menos significatiivo de DL, el valor del
    ; digito
    PUSH AX         ; Guardar los registros que queremos
    PUSH DX         ; preservar
    AND DX,0Fh      ; Apagar el nibble mas alto
    CMP DL,10       ; Compara con 10
```

```
    JAE ES_MAYOR_A_10    ; Si es mayor o igual salta
    ADD DL,48        ; Suma 48 para imprimir su ASCII si es 0..9
    JMP IMPRIME_DIG
ES_MAYOR_A_10:
    ADD DL,55        ; Suma 55 para imprimir su ASCII si es A..F
IMPRIME_DIG:
    MOV AH,2         ; Servicio para imprimir un caracter
    INT 21h          ; Llamar a DOS
    POP DX           ; Recuperar los registros
    POP AX
    RET              ; Regresar del procedimiento
WRITEHEX_DIG ENDP


WRITECRLF PROC NEAR
    ; Hace un CR y LF
    PUSH AX
    PUSH DX
    ;
    MOV AH,2         ; Servicio para imprimir un caracter
    MOV DL,13        ; Caracter 13 = CR
    INT 21h          ; Llamar a DOS
    MOV DL,10        ; Caracter 10 = LF
    INT 21h          ; Llamar a DOS
    ;
    POP DX
    POP AX
    RET              ; Regresar del procedimiento
WRITECRLF ENDP

CODE ENDS
END INICIO
```

Sugerencias para el diseño modular

Estas sugerencias facilitarán el trabajo de programacion, estas sugerencias pueden seguirse en la definición de los subprogramas.

1 - Guarda en el stack y restaura todos los registros que se modificaran dentro del procedimientoo, excepto aquellos en los que se devolveran valores por el procedimiento.

2 - Se consistente acerca de que registros utilizaras para pasar informacion a un procedimiento, por ejemplo:

- DL,DX - Manadar valores, bytes y palabras.
- AL,AX - Regresar valores, bytes y palabras.
- BX:AX - Regresar valores, palabras dobles.
- CX - Contadores, o repetidores.
- CF - Encendida cuando ocurra un error, el error debe ser devuelto en en alguno de los registros, tal como AX o AL.

3-Definir todas las interacciones con comentarios, en el encabezado del procedimiento.

- Parametros de entrada.
- Informacion que retorna (registros alterados)
- Procedimiemtos llamados.
- Variables usadas (lectura o escritura).

Paso de parámetros en el stack.

Este es nesesario para hacer interfases con los lenguajes de alto nivel ya que estos cuando hacen llamados a procedimientos usan el stack para pasar parametros. Cuando se pasan parametros en el stack, hay que recordar que la hacer la llamada al procedimiento este guarda la dirección de retorno en el stack, (una palabra IP si es un procedimiento cercano o dos palabras CS e IP si es un procedimiento lejano), esto quiere decir que el ultimo parametro guardado esta despues de la dirección de retorno (una o dos palabras correspondintes al caso de la llamada cercana o lejana), como posteriormente se guarda en el stack el valor de BP, el registro SP se decrementa en 2 y el valor de este se le asigna a BP, en este momento tenemos ya otro dato en el stack, por lo que es nesesario sumar otra palabra a BP para accesar el ultimo parametro en el stack.

Ejemplo:

```
 .MODEL SMALL

 .DATA
    msg db 'Esta es una prueba ',0

 .CODE
START:
    mov ax,@DATA
    mov ds,ax
    ;
    mov ax,OFFSET msg
```

```
    push ds
    push ax
    call puts
    add sp,4
    ;
    mov ah,4Ch
    int 21h

PUTS PROC NEAR
    ; Entrada:
    ;   En el stack si BP = SP
    ;   SS:[BP+4] = Offset del string.
    ;   SS:[BP+6] = Segmento del string.
    ;
    push bp
    mov bp,sp
    push ax
    push bx
    push dx
    push ds
    ;
    mov ax,[bp+6]
    mov ds,ax
    mov bx,[bp+4]
CICLO:
    mov ah,2
    mov dl,[bx]
    int 21h
    inc bx
    cmp byte ptr [bx],0
    jne CICLO
    ;
    pop ds
    pop dx
    pop bx
    pop ax
    pop bx
    ret
PUTS ENDP

 .STACK 100h

END START
```

Prefijos :

REP - Este prefijo ocasiona que se repita la instruccion el número de veces indicado en CX. Este prefijo solo tiene sentido con las instrucciones MOVS y STOS. (INS y OUTS en los procesadores 80286 y posteriores).

REPE o **REPZ** - Este prefijo repite la instruccion el número de veces indicado en CX mientras la bandera del cero esta encendida. Este prefijo se utiliza con las instrucciones CMPS, SCAS.

REPNE o **REPNZ** - Este prefijo repite la instruccion el número de veces indicado en CX mientras la bandera del cero se encuentre apagada. Este prefijo se utiliza con las instrucciones CMPS, SCAS.

La bandera de dirección:

DF = 0 - Las instrucciones para manejo de strings incrementan los indices SI y DI.

DF = 1 - Las instrucciones para manejo de strings decrementan los indices SI y DI.

Instrucciones:

CMPSB - (Compara string byte por byte) Compara el byte que se encuentra en la dirección indicada por DS:SI con el byte que se encuentra en la dirección indicada por ES:DI, esto solo afecta las banderas, despues SI y DI incrementan o decrementan en uno dependiendo del estado de la bandera de dirección.

CMPSW - (Compara string word por word) Compara la palabra que se encuentra en la dirección indicada por DS:SI con la palabra que se encuentra en la dirección indicada por ES:DI, esto solo afecta las banderas, despues SI y DI incrementan o decrementan en dos dependiendo del estado de la bandera de dirección.

Ejemplo de comparacion de strings

```
.MODEL SMALL
.DATA
   STR1 DB DUP(0)
   STR2 DB DUP(0)
   MAYOR DB 'Es mayor $'
   MENOR DB 'Es menor $'
```

```
    IGUAL DB 'Son iguales $'

 .CODE
INICIO:
    MOV AX,@DATA
    MOV DS,AX
    MOV ES,AX
    ; DS y ES apuntan al segmento donde estan los strings
    MOV CX,30             ; Número de bytes a comparar
    MOV SI,OFFSET STR1    ; SI apunta a str1
    MOV DI,OFFSET STR2    ; DI apunta a str2
    CLD                   ; Asegurar que SI y DI incrementaran
    REPE  CMPSB                ; Compara string byte por byte
mientras                          ; son iguales.
    JA ES_MAYOR           ; Si STR1 es mayor que STR2
    JB ES_MENOR           ; Si STR1 es menor que STR2
    MOV AH,9              ; Si son iguales
    MOV DX,OFFSET IGUAL
    INT 21h
    JMP FIN
     -------------
ES_MAYOR:
    MOV AH,9
    MOV DX,OFFSET MAYOR
    INT 21h
    JMP FIN
     -------------
ES_MENOR:
    MOV AH,9
    MOV DX,OFFSET MENOR
    INT 21h
     -------------
FIN:
    ...
```

MOVSB - (Mover un string byte por byte) Mueve un byte de la cadena fuente cuya dirección es indicada por los registros DS:SI, al byte de la cadena destino cuya dirección es indicada por ES:DI, despues SI y DI incrementan o decrementan en 1 dependiendo de la bandera de dirección.

MOVSW - (Mover un string word por word) Mueve una palabra de la cadena fuente cuya dirección es indicada por los registros DS:SI, a la palabra de la cadena destino cuya dirección es indicada por ES:DI, despues SI y DI incrementan o decrementan en 2 dependiendo de la bandera de dirección.

Ejemplo de copia de strings

```
 .MODEL SMALL
 .DATA
    STR1 DB 'Esta es una prueba $'
    STR2 DB 30 DUP(0)

 .CODE
INICIO:
    MOV AX,@DATA
    MOV DS,AX
    MOV ES,AX
    ; DS y ES apuntan al segmento donde estan los strings
    MOV CX,30             ; Número de bytes a transferir
    MOV SI,OFFSET STR1    ; SI apunta a str1
    MOV DI,OFFSET STR2    ; DI apunta a str2
    CLD                   ; Asegurar que SI y DI incrementaran
    REP MOVSB             ; Copiar string byte por byte
    ...
```

SCASB - Realiza la resta del byte que se encuentra en la dirección indicada por ES:DI con el valor que se encuentra en el registro AL, y establece las banderas de acuerdo al resultado, si la bandera de dirección se encuentra en cero, SI y DI incrementaranen uno, de lo contrario decrementaran en uno.

SCASW - Realiza la resta de la palabra que se encuentra en la dirección indicada por ES:DI con el valor que se encuentra en el registro AX, y establece las banderas de acuerdo al resultado, si la bandera de dirección se encuentra en cero, SI y DI incrementaranen dos, de lo contrario decrementaran en dos.

```
.MODEL SMALL
.DATA
    STR1 DB 30,0,30 DUP(0)

.CODE
INICIO:
    MOV AX,@DATA
    MOV DS,AX
    MOV ES,AX
    ; DS y ES apuntan al segmento donde estan los strings
    MOV AH,0Ah            ; Leer un string
    MOV DX,OFFSET STR1
    INT 21h
    ;
```

```
    MOV AL,0Dh            ; Byte a buscar en el string CR
MOV CX,30             ; Número de bytes a buscar

    MOV DI,OFFSET STR1+2 ; DI apunta a str1
    CLD                  ; Asegurar que SI y DI incrementaran
    REPNE SCASB          ; Compara string byte por byte
mientras
                         ; cada byte sea diferente de AL

    MOV BYTE PTR ES:[DI-1],0 ; Hacer el string asciiz
     ...
```

STOSB - Carga el byte contenido en AL, en el byte que se encuentra en la dirección indicada por ES:DI, DI incrementa o decrementa en uno dependiendo de la bandera de dirección.

STOSW - Carga la palabra contenida en AX, en la palabra que se encuentra en la dirección indicada por ES:DI, DI incrementa o decrementa en dos dependiendo de la bandera de dirección.

Ejemplo:
Llenar un area de memoria con un patron definido.

```
.MODEL SMALL
.DATA
    PATRON DW 201 DUP(0)

.CODE
    MOV AX,@DATA
    MOV DS,AX
    MOV ES,AX
    MOV DI,OFFSET PATRON     ; ES:DI apunta al buffer
    MOV CX,200               ; Número de palabras a llenar
    MOV AX,'\/'              ; Palabra que se utilizara
    CLD                      ; DI incrementara
    REP STOSW                ; Llenar el buffer
    MOV BYTE PTR ES:[DI],'$' ; Establece terminador al final
    MOV AH,9
    MOV DX,OFFSET PATRON
    INT 21h
    ...
```

LODSB - Carga el byte que esta en la dirección indicada por DS:SI en el registro AL, SI incrementa o decrementa en uno dependiendo del estado de la bandera de dirección.

LODSW - Carga la palabra que esta en la dirección indicada por DS:SI en el registro AX, SI incrementa o decrementa en dos dependiendo del estado de la bandera de dirección.

En este ejemplo se muestra una rutina que imprime un string, esta rutina imprime un string el cual tiene un byte 0 como terminador.

```
 .MODEL SMALL
 .DATA
    STR1 DB 'Esta es una prueba ',0

 .CODE
INICIO PROC NEAR
    MOV AX,@DATA
    MOV DS,AX
    MOV ES,AX
    ; DS y ES apuntan al segmento donde estan los strings
    MOV DX,OFFSET STR1
```

```
    CALL PUTS
    MOV AH,4CH
    INT 21H
INICIO ENDP
PUTS PROC NEAR
    ; Entrada:
    ;   DS:DX = Segmento:Offset del string terminado con 0
    ; Llama:
    ;   PUTCHAR
    PUSH AX
    PUSH DX
    PUSH SI
    PUSHF
    CLD         ; Asegurar que SI y DI incrementaran
    MOV SI,DX   ; SI Apunta al inicio del string
OTRO_CAR:
    LODSB       ; Carga en AL el byte que esta en DS:SI
    MOV DL,AL
    CALL PUTCHAR
    OR AL,AL
    JNE OTRO_CAR
    ;
    POPF
    POP SI
    POP DX
    POP AX
    RET
PUTS ENDP
PUTCHAR PROC NEAR
    ; Entrada:
    ;   DL = Caracter a imprimir
    PUSH AX
    MOV AH,2    ; Servicio para imprimir un caracter
    INT 21h     ; Llamar a MS-DOS
    POP AX
    RET
PUTCHAR ENDP

 .STACK 100H
END INICIO
```