Lenguaje ensamblador

1. ASPECTOS IMPORTANTES

DIRECTIVAS: palabras reservadas para controlar la forma en que el ensamblador enlista los programas al momento de ensamblados y enlazarlos.

OPERADORES: palabras reservadas que se utilizan para cambiar o analizar operandos en un programa en ensamblador.

INSTRUCCIONES: palabras reservadas para la ejecución de instrucciones en los programas.

TIPOS DE DATOS:

a) DB: define un dato de 1 byte
b) DW: define un dato de 2 bytes
c) DD: define un dato de 4 bytes
d) DF: define un dato de 6 bytes
e) DQ: define un dato de 8 bytes
f) DT: define un dato de 10 bytes

SINTAXIS: nombre Dn expresión

Ejemplos:

```
uno db 1                ;define una variable de nombre uno con un valor inicializado en 1
texto db “h’, ‘o’, l’, ‘a’
```

Notas:

a) Uso de ‘ **?** ‘: se usa para declarar datos o variables sin valor de inicialización.
   Ejemplo:
   dos db ? ; define una variable de nombre ***dos*** sin inicialización.

b) Uso de ‘ **DUP** ’: se usa para ampliar el tamaño del dato declarado.
   Ejemplo:
   tres db 3 dup (4) ;inicializa el dato tres con tres bytes conteniendo el valor 4 en cada uno.

c) Cadenas de caracteres,
   Ejemplo:
   cadena db ‘texto variado’

d) Constantes: se definen con la palabra reservada **EQU** con la sintaxis: nombre EQU valor.
   Ejemplo:
   const equ 52

2. ESTRUCTURA DE UN PROGRAMA .EXE

DIRECTIVAS:

1. **TITLE:** se usa para definir el nombre del programa.

SINTAXIS: TITLE nombre
Ejemplo: TITLE cambiar_variables

2. **SEGMENT:** se usa para definir segmentos en el programa.

SINTAXIS:

nombre SEGMENT opción

(instrucciones)

nombre ENDS

Opción puede tomar los valores:

- Alineación: límite de inicio de segmento en la memoria. PARA almea el segmento a párrafo de memoria, 10H.
- Combinar para 'enlazar' programas, procedimientos o subrutinas con otros programas al realizar el ensamblado, con valores:

  i. STACK, para definir la pila.
  ii. COMMON: para ligar o combinar programas de forma separada.
  iii. PUBLIC: para ligar o combinar programas de forma separada.
  iv. AT: para ligar o combinar programas de forma separada. No muy utilizado.
  v. NONE (u omisión de valor): para no enlazar programas.

- Clase: se denota entre apóstrofes para agrupar segmentos al enlazar programas.

Ejemplo:
pila SEGMENT PARA STACK 'stack' ;por omisión de tamaño se inicializa con 1024 bytes
pila ENDS

Ejemplo de estructura de programa .EXE:

```
TITLE     estructura_programa_exe
pila      SEGMENT PARA STACK 'stack'
          ;por omisión de tamaño se inicializa con 1024 bytes
pila      ENDS
datos     SEGMENT PARA 'data'
          ;definición de los datos del programa
datos     ENDS
codigo    SEGMENT PARA 'code'
          ;definición de las instrucciones del programa
codigo    ENDS
END       estructura_programa_exe
```

3. **ESPECIFICACIÓN DE SEGMENTOS:** Para especificar el segmento del cual se accesa un dato, se utiliza el operador **:**

```
SINTAXIS:   registro_segmento:direccion
Ejemplo:    MOV ES:[DI + 20], AX
            MOV  DX, DS:[BX]
```

4. **PROC:** se usa para definir procedimientos en el programa. Se definen en el segmento de código.

```
SINTAXIS:   nombre PROC valor   ;valor referencia la posibilidad de
                                desplazamientos en uno o varios
                                segmentos
Ejemplo:
corto PROC NEAR   ;para desplazamientos cercanos en el mismo segmento
largo PROC FAR    ;para desplazamientos lejanos en el mismo o a otros
segmentos
```

5. **ASSUME:** se usa para definir la posición en memoria de los segmentos del programa en el código del mismo y cargar los registros de segmento con estas posiciones. Se establecen en el segmento de código.

```
SINTAXIS:   ASSUME ss:nombre_segmento_pila,
ds:nombre_segmento_datos, cs:nombre_segmento_codigo …
```

Al iniciar un programa, el sistema operativo carga automáticamente las posiciones de memoria para los registros de segmento CS y SS.

Sin embargo, no sucede lo mismo con el segmento de datos. Por esta razón, se debe definir y cargar la posición en memoria de este segmento del programa en el registro respectivo. El inconveniente de hacer esto es que no se puede cargar datos directamente desde la memora hacia algún registro de segmento. Por ello, para lograr cargar esta posición en el registro de segmento de datos se debe utilizar algún registro intermedio (que usualmente es el registro AX) para realizar esta carga de información.

De esta manera, las primeras instrucciones que se definen en el segmento de código serán:

```
ASSUME ss: pila, ds:datos, cs: codigo
MOV AX, datos
MOV DS, AX
```

3. TERMINACION DE LA EJECUCION DE UN PROGRAMA.

Para cerrar (terminar) un programa y devolver el control de la ejecución de instrucciones al sistema operativo, se utiliza la interrupción de DOS 21H, la cual, según el valor almacenado en el registro acumulador (AX), permite realizar distintas operaciones, tales como: leer datos del teclado o del ratón, enviar información al monitor o la impresora.
Al cargar el registro AX con el valor 4C00H (o solamente cargando el registro AH con el valor 4CH) y ejecutar la interrupción 21H del DOS, se finaliza la ejecución de un programa y se regresa el control al DOS.

Ejemplo:

```
Inicio      PROC FAR
            ASSUME ss:pila, ds:datos, cs: codigo
            MOV AX, datos
            MOV DS, AX          ;carga de la posición del segmento de datos en
            memoria en el registro os
            MOVAH, 4CH
            INT 21H             ;terminación del programa
      Inicio ENDP
```

4. DIRECTIVAS SIMPLIFICADAS DE DECLARACION DE SEGMENTOS.

Otra manera de definir los segmentos en los programas es inicializando el modelo de memoria a utilizar en el programa. Para ello, se emplea la directiva:

.MODEL modelo_de_memoria ;incluyendo el **punto,** inmediatamente luego del titulo del programa

Modelos de memoria:

MODELO	CANTIDAD SEGMENTOS DE CODIGO	CANTIDAD SEGMENTOS DE DATOS
TINY	*	*
SMALL	1	1
MEDIUM	≥1	1
COMPACT	1	≥1
LARGE	≥1	≥1;con arreglos ≤64kb
HUGE	≥1	≥1;con arreglos >64kb

* el modelo TINY se usa exclusivamente para programas .COM

FORMATO PARA DEFINIR SEGMENTOS: para definir el nombre de los segmentos del programa se utilizan las directivas:

SINTAXIS:

```
.STACK tamaño     ;para declarar el segmento de pila. tamaño es opcional y por
                  omisión se declara una pila de 1024 bytes
.DATA             ;para declarar el segmento de datos
.CODE nombre      ;para declarar el segmento de código. nombre es opcional y
                  usualmente no se escribe.
```

Igualmente que con la nomenclatura revisada anteriormente, se debe cargar la dirección de memoria del segmento de datos en el registro DS, pero la forma de

hacerlo se simplifica a las siguientes instrucciones en el segmento de código:

```
MOV AX, @data
MOV DS, AX
```

Ejemplo de estructura de programa .EXE con la nomenclatura de modelos de memoria:

```
TITLE   estructura_programa_exe
.MODEL SMALL
.STACK 64               ;usualmente se define con este valor ¡nidal, pero se puede
inicializar con otro valor según la necesidad
.DATA                   ;definición de los datos del programa
.CODE
inicio  PROC FAR    ;procedimiento principal del programa
        ;definición de las instrucciones del programa
inicio  ENDP
        END estructura_programa_exe
```

5. ESTRUCTURA DE UN PROGRAMA .COM

Los programas .COM son programas como los .EXE, pero se diferencian de estos por tres características principales:

1. *Segmentación:* los programas .COM ocupan solamente un segmento de memoria (64 kb) en el cual se alojan los datos, las instrucciones del operaciones y los datos de pila; a diferencia de los programas .EXE, los cuales definen estos elementos en segmentos separados.
2. *Inicialización:* por la característica anterior, los programas .COM requieren una inicialización del segmento de código en el cual se definen los datos del programa, para lo que se requiere de algunos ajustes que se revisarán más adelante.
3. *Tamaño:* los programas .COM son generalmente más pequeños en tamaño que los .EXE (aunque realicen las mismas instrucciones) debido a la cantidad de segmentos necesarios para su ejecución. Los primeros solamente definen características para un único segmento, los segundos definen como mínimo dos (código y datos).

En el caso de los programas .COM, el DOS (o el ensamblador) define automática - mente un espacio al final del segmento para la pila del programa: en caso de que este espacio sea insuficiente, se define un espacio adicional fuera del segmento.

Para especificar al procesador de que se trata de un programa .COM al momento de realizar el ensamblado del código fuente, se debe incluir la instrucción ORG 100H inmediatamente después de la línea de declaración del segmento de código, ya sea:

codigo SEGMENT PARA ‘code’

ASSUME cs: codigo, ds: codigo, ss: codigo, es: codigo

ORG 100H

O igualmente con la utilización de la nomenclatura de modelo de memoria se declara:

.CODE

ORG 100H

Ejemplos de estructura de un programa .COM

TITLE estructura_programa_com codigo SEGMENT PARA ‘code’ ASSUME cs:codigo, ds: codigo, ss: codigo, es: codigo ORG 100H Inicio: JMP principal ;definición de los datos, se puede colocar al final principal PROC NEAR ;definición de las instrucciones del programa MOVAH, 4CH INT 21H ;terminación del programa principal ENDP codigo ENDS END estructura_programa_com	TITLE estructura_programa_com .MODEL SMALL .CODE ORG 100H Inicio: JMP principal ;definición de los datos, se puede colocar al final principal PROC NEAR ;definición de las instrucciones del programa MOV AH, 4CH INT 21H ;terminación del programa principal ENDP END estructura_programa_com

6. INSTRUCCIONES GENERALES

a) **OPERANDO OFFSET:** devuelve la dirección de memoria (en el segmento de datos) de una variable o una etiqueta.

SINTAXIS:
OFFSET operando ; el operando puede ser una variable o una etiqueta

Ejemplo:

tabla DB 1, Z 3, 4, 5

MOV BX, OFFSET tabla

MOV tabla +3, [BX

MOV [BX), 8

b) **LEA:** al igual que el operando OFFSET, la Instrucción LEA se usa para obtener la dirección de memoria (en el segmento de datos) de una variable o una etiqueta. Usualmente se usa con SI, Dl y BX.
SINTAXIS: LEA destino, nombre

Ejemplo:

```
tabla DB 1, 2, 3, 4, 5
LEA BX, tabla
MOV tabla +3, [BX]
MOV [BX], 8
```

c) **MOV:** se usa para transferir el dato de origen al destino.

SINTAXIS: MOV destino, origen ;tanto el origen como el destino pueden ser registros o datos en memoria, pero ambos no pueden ser elementos de memoria simultáneos

Ejemplo:

```
TITLE cambiar_variables
valor DB 1
MOV valor, BX
MOV BX, valor
MOV BX, AX
```

d) **XCHG:** se usa para intercambiar los valores de los operandos de origen y destino.

SINTAXIS: XCHG destino, origen ;tanto el origen como el destino pueden ser registros o datos en memoria, pero ambos no pueden ser elementos de memoria simultáneos

Ejemplo:

```
XCHG BX, AX
XCHG DX, dato
```

e) **INC:** se usa para aumentar el valor del operando en 1. Afecta el contenido de los bits del registro de banderas OF, SF y ZF.

SINTAXIS: INC operando
Ejemplo:

```
INC CX
INC cuenta
```

f) **DEC:** se usa para disminuir el valor del operando en 1. Afecta el contenido de los bits del registro de banderas OF, SF y ZF.

SINTAXIS: DEC operando
Ejemplo:

```
DEC CX
DEC cuenta
```

7. INSTRUCCIONES ARITMÉTICAS BÁSICAS.

a) **ADD:** se usa para sumar el operando de origen al de destino. El resultado se almacena en el operando destino.
SINTAXIS: ADD destino, origen
Ejemplo:
ADD AX, 20
ADO DX, BX

b) **SUB:** se usa para restar el operando de origen al de destino. El resultado se almacena en el operando destino.
SINTAXIS: SUB destino, origen
Ejemplo:
SUS AX, 20
SUB DX, BX

c) **MUL (sin signo) / IMUL (con signo):** se usa para multiplicar el operando especificado (multiplicador) al contenido del registro AX.

SINTAXIS: MUL multiplicador

IMUL multiplicador

Ejemplos:

OPERACIÓN	MULTIPLICADOR	MULTIPLICANDO	PRODUCTO
MUL CL	CL	AL	AX
MUL valor_1 Byte	valor_1 Byte	AL	AX
MUL BX	BX	AX	DX:AX
MUL valor_2Bytes	valor_2Bytes	AX	DX:AX
MUL EDX	EDX	EAX	EDX:EAX
MUL valor_4Bytes	Valor_4Bytes	EAX	EDX:EAX

d) **DIV (sin signo) / IDIV (con signo):** se usa para dividir el contenido del registro AX entre el operando especificado (divisor).

SINTAXIS: DIV divisor

Ejemplos:

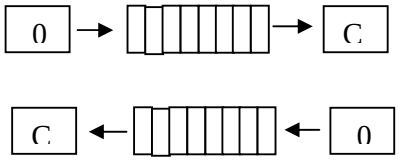
OPERACION	DIVISOR	DIVIDENDO	COCIENTE	RESIDUO
DIV CL	CL	AX	AL	AH
DIV valor_1Byte	Valor_1Byte	AX	AL	AH
DIV BX	BX	DX:AX	AX	DX
DIV valor_2Bytes	Valor_2Bytes	DX:AX	AX	DX
DIV EDX	EDX	EDX:EAX	EAX	EDX
DIV valor_4Bytes	Valor_4Bytes	EDX:EAX	EAX	EDX

e) **SHR /SHL:** usa para realizar el corrimiento lógico a la derecha /izquierda y rellena con ceros el corrimiento referenciado con el registro contador (CL, CX, ECX) o con un valor inmediato. **SHL** se utiliza para realizar multiplicaciones (por corrimiento) del operando por potencias de 2 (operando * 2, *4, *8, etc.). **SHR** se utiliza para realizar divisiones (por corrimiento) del operando por

potencias de 2 (operando *¡2,14,18.* etc.). Afecta el contenido de los bits del registro de banderas CF, OF, PF, SF y ZF. El último bit que se descarta del operando se almacena en almacena en el bit de carry (acarreo).

SINTAXIS:

SHR operando, contador / inmediato
SHL operando, contador / inmediato



Ejemplo:
SHL AX, 1
MOV CL, 5
SHR AX, CL

f) **SAR / SAL:** se usa para realizar el corrimiento aritmético (con signo) a la derecha / izquierda. En el caso de SAL, es igual a SHL, tomando en consideración que se trata de valores con signo. En el caso, de SAR, incluye por la izquierda del operando el valor del bit de signo del registro de banderas (SF). En ambos casos, el último valor (bit) desechado del operando se almacena en el bit de acarreo. Se realiza el corrimiento referenciado con el registro contador (CL, CX, ECX) o con un valor inmediato. Afecta el contenido de los bits del registro de banderas CF. OF, PF, SF, ZF.

SINTAXIS:
SAR operando, contador / inmediato
SAL operando, contador / inmediato

Ejemplo:
SHL AX,1
MOV CL,5
SHR AX, CL

g) **ROR / RCR:** se use para realizar rotaciones de bits en registros y operandos en memoria a la derecha según el valor especificado con el registro contador (CL, CX, ECX) o con un valor inmediato. Afecta el contenido de los bits del registro de banderas CF y OF.

ROR: rotación lógica a la derecha
RCR: rotación a la derecha con acarreo

SINTAXIS:
ROR operando *(*R / M), contador / inmediato
RCR operando (R / M), contador / inmediato

Ejemplo:
ROR AX, 1
MOV CL,5
ROR AX, CL

h) **ROL / RCL:** se usa para realizar rotaciones de bits en registros y operandos en memoria a la izquierda según el valor especificado con el registro contador (CL, CX, ECX) o con un valor Inmediato. Afecta el contenido de los bits del registro de banderas CF y OF.

ROL: rotación lógica a la izquierda
RCL: rotación a la izquierda con acarreo _____

SINTAXIS:
ROL operando *(*R / M), contador / inmediato
RCL operando (R / M), contador / inmediato

Ejemplo:
ROL AX, *1*
MOV CL, 5
ROL AX, CL