TEMA

33

Programación en lenguaje ensamblador. Instrucciones básicas. Formatos. Direccionamientos

Mª Luisa Garzón Villar

CUERPO DE PROFESORES DE ENSEÑANZA SECUNDARIA

ÍNDICE SISTEMÁTICO

1. INTRODUCCIÓN

El lenguaje ensamblador fue el primer intento de sustituir el lenguaje máquina por otro más parecido al utilizado por el hombre. La transformación consistió en la sustitución de las cadenas de unos y ceros por palabras mnemotécnicas en cada una de las instrucciones en lenguaje máquina.

Por este motivo, se presentan prácticamente los mismos inconvenientes que en la utilización del lenguaje máquina: el problema de que cada máquina tiene el suyo propio y, por lo tanto, los programas no son transportables; el de la necesidad de conocer perfectamente el hardware del equipo, ya que se trabaja directamente con direcciones de memoria y registros del microprocesador; y la "incomodidad" de trabajar con instrucciones elementales que provocan la descripción detallada de todas las acciones que habrá de llevar a cabo el ordenador.

Pero por este mismo motivo, este lenguaje incorpora la ventaja del lenguaje máquina en cuanto a la mínima ocupación de memoria y mínimo tiempo de ejecución comparados con el resultado de la compilación del mismo programa escrito en otro lenguaje.

Para el desarrollo de este tema, y para así poder explicar con detalle algunos conceptos, nos centraremos en un modelo de microprocesador concreto, la serie 80x86 de Intel, del que recordaremos algunos conceptos previos necesarios.

1.1. Registros internos del microprocesador

Son 14 registros de 16 bits, divididos en las siguientes categorías:

- 4 registros de datos o de almacenamiento temporal:
  - **AX** (Acumulador): principal registro utilizado en las operaciones aritméticas.
  - **BX** (Base): se utiliza para indicar desplazamientos.
  - **CX** (Contador): se utiliza como contador en bucles y en operaciones de tipo repetitivo.
  - **DX** (Dato): se utiliza en operaciones aritméticas.

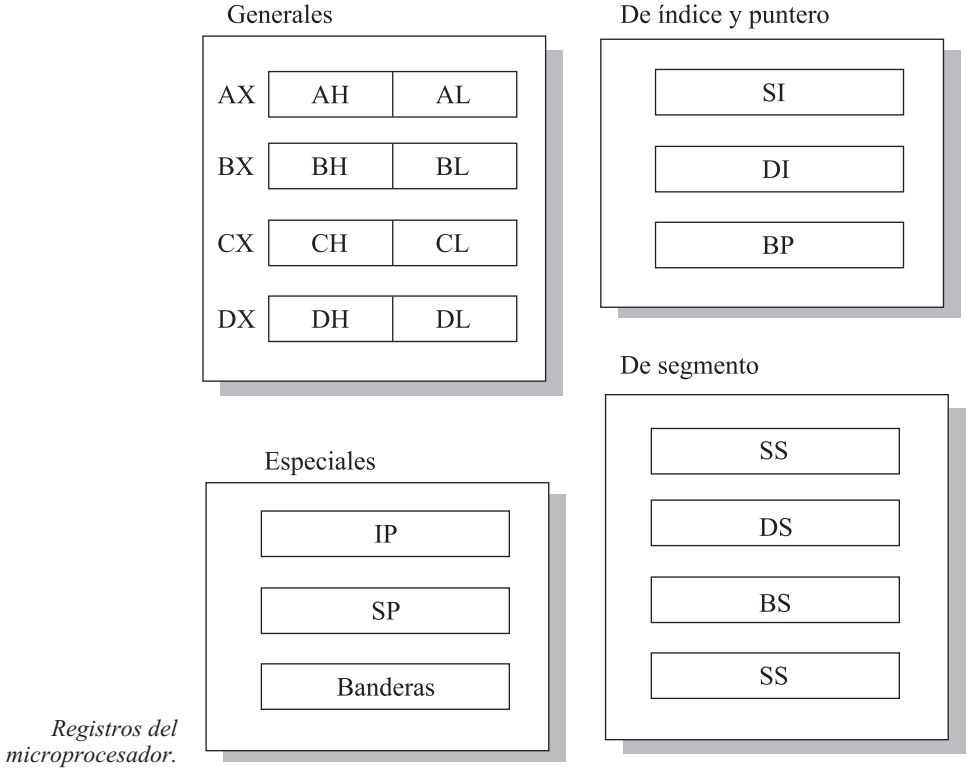
  Existe la posibilidad de referirse a la mitad inferior o superior de estos registros utilizando la denominación AH, para la parte superior (más significativa) del acumulador, y AL, para la parte inferior (menos significativa) del acumulador; esta denominación se puede aplicar también a los demás registros de datos.

- 4 registros de segmentos que contienen la dirección de comienzo de cada segmento:
  - **CS** (Registro del segmento de código): indica la posición de comienzo del segmento de código, o sea las instrucciones del programa.
  - **DS** (Registro del segmento de datos): indica la posición donde empieza el segmento de datos, es decir, el área de memoria donde están almacenados los datos del programa.
  - **SS** (Registro del segmento de la pila): indica la posición de memoria donde empieza la pila.
  - **ES** (Registro del segmento extra): indica la posición de memoria donde comienza el segmento extra, un segmento de datos adicional que también se utiliza para transferencia de datos entre segmentos.

- 2 registros punteros de pila:
  - **BP** (Puntero base): indica la dirección de comienzo de la pila.
  - **SP** (Puntero de pila): indica la posición de la cabecera de la pila.
- 2 registros índices que se usan como desplazamiento relativo a un campo de datos:
  - **SI** (Índice fuente).
  - **DI** (Índice destino).
- 1 registro puntero de instrucciones, también llamado contador de instrucciones (PC):
  - **IP** (Puntero de instrucción): que contiene un desplazamiento sobre el segmento de código e indica, junto con el registro CS, la posición de la siguiente instrucción a ejecutar (CS:IP).
- 1 registro de banderas:
  - **FLAGS** (Banderas): que almacenan información de control y estado de las operaciones del microprocesador. Existen nueve banderas en este registro y cada una ocupa un bit, de forma que los siete bits restantes no se utilizan:
- 6 banderas de estado: normalmente asociadas a las instrucciones aritméticas o de comparación, registran el estado del microprocesador.
  - **Bit 0, CF** (Bandera de acarreo): para indicar acarreo en las operaciones aritméticas ("1" acarreo, "0" no hay acarreo).
  - **Bit 11, OF** (Bandera de overflow): para indicar si ha habido o no desbordamiento aritmético ("1" overflow, "0" no hay overflow).
  - **Bit 6, ZF** (Bandera de resultado cero o comparación igual): indica si el resultado de la última operación ha sido cero ("1" ha sido cero, "0" ha sido distinto de cero).
  - **Bit 7, SF** (Bandera de resultado o comparación negativa): guarda el bit más significativo del resultado (bit de signo).
  - **Bit 2, PF** (Bandera de paridad): indica paridad par o impar.
  - **Bit 4, AF** (Bandera de acarreo auxiliar): indica si hay necesidad de ajuste en operaciones aritméticas en BCD.
- 3 banderas de control: que registran el modo de funcionamiento del procesador:
  - **Bit 10, DF** (Bandera de dirección): indica si una operación con una cadena de caracteres se realizará hacia adelante o hacia atrás, incrementando o decrementando los registros índices SI y DI ("1" se empieza por el final, "0" se empieza por el principio).
  - **Bit 9, IF** (Bandera de interrupciones): indica si se permiten o no las interrupciones por parte de los dispositivos externos ("1" permitidas, "0" no permitidas).
  - **Bit 8, TF** (Bandera de atrape): utilizada por el programa Debug para el control de la operación paso a paso ("1" lee, ejecuta y se para por cada instrucción).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CF		PF		AF		ZF	SF	TF	IF	DF	OF				

Registro de banderas (FLAGS).



Registros del microprocesador.

1.2. La memoria del ordenador. Direccionamiento

La arquitectura de estos ordenadores obliga a que la memoria sea dividida en bloques o segmentos de 64 KB de longitud, capaces de ser direccionados con una palabra de 16 bits; de esta forma, para acceder a un dato de la memoria, se utilizarán dos registros, el registro que indica donde empieza el segmento y un registro que indica el desplazamiento con respecto a dicho comienzo, de forma que las direcciones de memoria tendrán el siguiente aspecto:

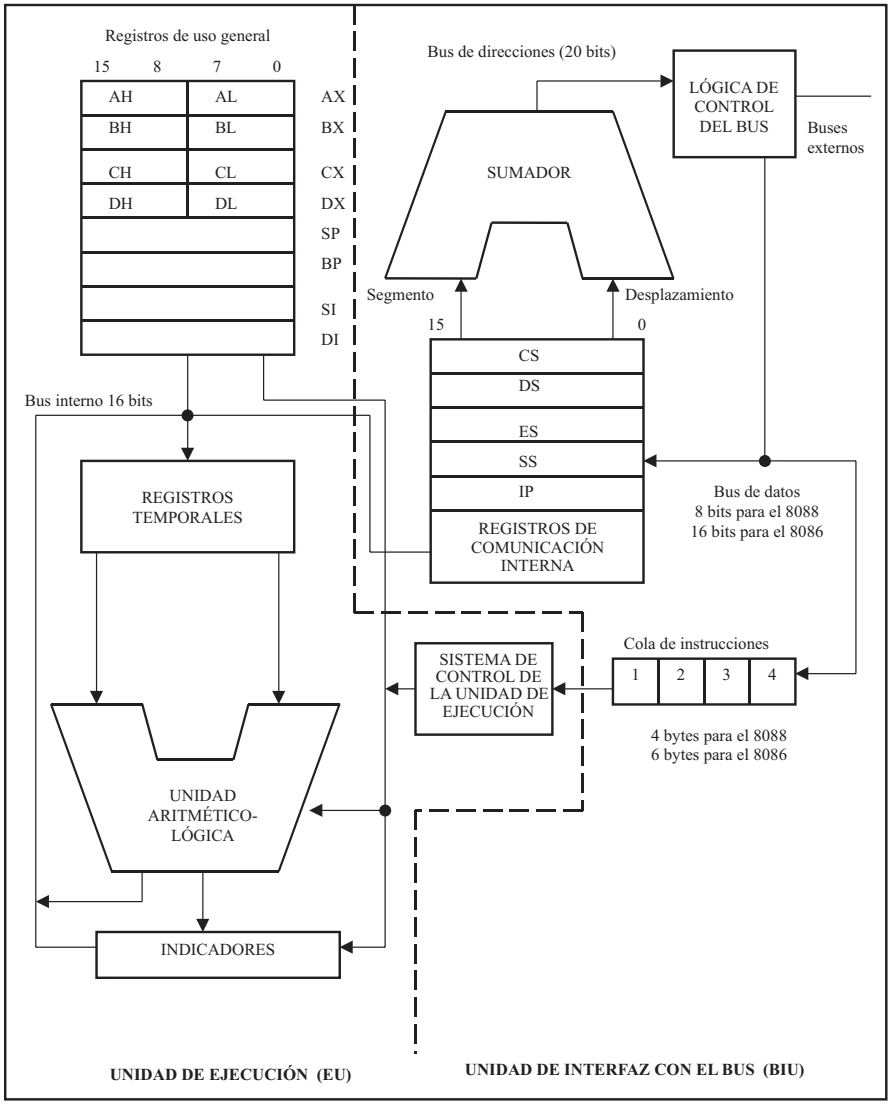
Segmento: desplazamiento.

Segmentos y registros asociados

Dentro de un programa, podemos encontrarnos con cuatro segmentos, cada uno de los cuales se direcciona mediante un registro de segmento y uno o varios registros de desplazamiento, de la siguiente forma:

- Segmento de código, donde cada instrucción se direcciona mediante:
  - Registro de segmento: CS.
  - Registro de desplazamiento: IP.

- Segmento de datos, donde cada dato se direcciona mediante:
  - Registro de segmento: DS.
  - Registros de desplazamiento BX, SI, o DI.
- Segmento de pila, donde cada elemento de la pila se direcciona mediante:
  - Registro de segmento: SS.
  - Registros de desplazamiento SP o BP.
- Segmento extra, donde cada dato se direcciona mediante:
  - Registro de segmento: ES.
  - Registros de desplazamiento BX, SI, o DI.



Modelo de CPU para el 8088/8086.

2. ELEMENTOS DE UN PROGRAMA EN ENSAMBLADOR

En un programa ensamblador, nos podemos encontrar con dos tipos de sentencias: las instrucciones y las directivas (o pseudo-operaciones o pseudo-ops).

Las instrucciones se aplican en tiempo de ejecución, y las directivas, en tiempo de ensamblaje del programa. Estas últimas se utilizan para indicar al ensamblador qué hacer con las instrucciones y los datos.

2.1. Instrucciones

El formato de una instrucción es el siguiente:

{etiqueta} VERBO-INSTRUCCIÓN {operandos} {comentario}

Nota: las llaves indican que el campo correspondiente es opcional, por lo tanto, en la sintaxis de una instrucción, solamente es obligatorio el verbo de la instrucción, ya que existen instrucciones que no llevan ningún operando.

La instrucción se escribirá en una sola línea y los distintos campos que componen la instrucción deberán estar separados, al menos, por un espacio en blanco.

Ej.: TRASPASO: MOV AX, 300; aquí iría un comentario sobre la instrucción.

2.1.1. El campo etiqueta

Corresponde al nombre simbólico de la primera posición de las que ocupa la instrucción, y se utiliza para poder romper la secuencia del programa. Este nombre podrá tener hasta 31 caracteres de longitud, y son admitidos los siguientes: letras de la A a la Z (las minúsculas se convierten en mayúsculas), números del 0 al 9 y caracteres especiales como ?, _, $, @, . (punto), con las siguientes condiciones:

- El primer carácter no puede ser un dígito.
- Si se usa el carácter "." deberá estar el primero.
- No se pueden utilizar los nombres reservados a otros elementos, como, por ejemplo, AX, AH, AL, BX, MOV, ADD, etc.
- Los dos puntos ":" que separan la etiqueta del verbo de la instrucción se utilizan para declarar la etiqueta como NEAR (en el caso de llevar los ":") o como FAR. Esto indicará si la referencia a la etiqueta se realizará desde dentro del mismo segmento de código o desde otro segmento de código distinto, ya que, al ser desde dentro del mismo segmento (NEAR), el microprocesador solo deberá cargar el IP al bifurcar a esa instrucción, mientras que si es FAR, el microprocesador deberá cargar los registros IP y CS al bifurcar a ella.

2.1.2. El campo verbo de la instrucción

Suele tener una longitud de dos a seis caracteres y representa la acción que ejecutará la instrucción.

2.1.3. El campo comentario

Se pueden incluir comentarios, que simplificarán la comprensión del programa, al final de cada instrucción, o utilizar una línea completa para ellos; en cualquiera de los casos, el comentario deberá ir precedido por ";".

2.1.4. El campo operando

Es el que indica los datos con los cuales se va a operar. Existen instrucciones de dos, uno o ningún operando, pero en el caso de haber dos, al primero se le denomina destino y al segundo fuente; en este caso, deberán ir separados por una coma.

```
Ej.:    PUSHF       ; esta instrucción no lleva operandos.
Ej.:    PUSH AX     ; esta es una instrucción con un operando.
Ej.:    MOV AX, BX  ; este es un ejemplo de una instrucción con dos operandos.
```

Los operandos pueden ser registros, direcciones de memoria o valores inmediatos (constantes). La forma de referenciarlos se denomina modo de direccionamiento.

2.1.4.1. Modos de direccionamiento

La mayoría de las operaciones, salvo algunas con cadenas de caracteres, se realizan entre registros o registros y memoria. Existen varias formas de referenciar los datos, de indicar dónde se encuentran; son lo que se denomina modos de direccionamiento.

- **Direccionamiento inmediato**.

  En este modo de direccionamiento, el dato aparece de forma explícita en la instrucción.

  Ejemplo: MOVE AX, 0A46Bh ;el operando fuente aparece de forma explícita.

- **Direccionamiento mediante registro**.

  El operando de la instrucción es un registro.

  Ejemplo: MOVE AX, BX ;el operando fuente es ahora el registro BX.

- **Direccionamiento absoluto**.

  El operando de la instrucción contiene la dirección de memoria donde se encuentra el dato.

  Ejemplo: MOV AX, [073BH] ;el dato con el que se va a operar está en la dirección de memoria 073BH.

  Ejemplo: MOV AX, DATO ;el dato con el que se va a operar está en la dirección de memoria denominada DATO.

- **Direccionamiento indirecto mediante registro**.

  El operando de la instrucción contiene el registro donde se encuentra la dirección de memoria donde está el dato. Los registros que se pueden utilizar en este caso son: BX, BP, DI y SI.

  Ejemplo: MOV AX, [BX] ;al estar BX entre corchetes, se indica que el registro lo que contiene no es el dato, sino la dirección de memoria donde se encuentra el dato.

- **Direccionamiento relativo a un registro base**.

  El operando de la instrucción contendrá un registro base con una dirección que, al sumarle un desplazamiento, nos llevará al dato con el que queremos operar. Los registros que se pueden utilizar en este caso son BX y BP.

  Ejemplo: MOV AX, [BX + 4] ;en este caso, el dato se localizará cuatro lugares mas allá (desplazamiento) de la dirección que indica BX.

– **Direccionamiento indexado**.

  Igual que el método anterior, pero utilizando los registros DI y SI.

  Ejemplo: MOV AX, [DI + 4] ;el dato se localizará cuatro lugares mas allá (desplazamiento) de la dirección que indica DI.

– **Direccionamiento indexado relativo a base**.

  El operando que contiene la instrucción contendrá un registro base (BX o BP), un registro índice (DI o SI) y un desplazamiento.

  Ejemplo: MOV AX, [BX + SI + 4] ;el dato se localizará cuatro lugares mas allá (desplazamiento) de la dirección que indica la suma de BX + SI.

Los cinco últimos modos de direccionamiento, los que se refieren a posiciones de memoria, pueden ir precedidos por un registro de segmento, de la siguiente forma:

```
MOV AX, ES : [BX]
```

Esta instrucción movería a AX lo que hay dentro de la dirección que indique BX en el segmento extra y no en el segmento DS donde se buscaría por defecto si no llevara **prefijo de segmento**.

2.2. Directivas

El formato de una directiva es el siguiente:

{nombre} VERBO-DIRECTIVA {operandos} {comentario}

Nota: las llaves indican que el campo correspondiente es opcional, por lo tanto, en la sintaxis de una directiva, solamente es obligatorio el verbo de la directiva, ya que existen directivas que no llevan ningún operando.

Como ocurría con las instrucciones, se escriben en una sola línea, y los distintos campos que la componen deberán estar separados, al menos, por un espacio en blanco.

```
Ej.:  CR EQU 13   ;     retorno de carro.
```

Esta directiva en concreto, asignará el nombre CR al valor 13. El campo comentario, es análogo al de las instrucciones.

2.3. Constantes

Como hemos indicado antes, las instrucciones y las directivas pueden llevar como operandos valores constantes. En este lenguaje, las constantes podrán ser de cinco tipos distintos: binarias, decimales, hexadecimales, octales y de tipo carácter. Existe la posibilidad de utilizar números negativos, siempre y cuando se les coloque el signo menos en el caso de valores decimales o se conviertan a complemento a 2 en el caso de binario, octal o hexadecimal. Teniendo en cuenta que si el primer dígito de una constante hexadecimal es una letra (de la A a la F), debe anteponerse un cero, para que el ensamblador pueda distinguir que se trata de una constante numérica y no una cadena de caracteres.

Ej.: 32 = 00100000b = 20h = 40q -32 = 11100000b = 0E0h = 340q

2.4. Operadores

Los operadores son modificadores que se utilizan en el campo de operandos y que pueden combinarse entre sí. Existen cinco clases de operadores en ensamblador:

2.4.1. Operadores aritméticos

Son los que operan sobre valores numéricos y pueden ser:

- Operador: +

  Formato: *operando1 + operando2.*

  Función: suma *operando1* y *operando2.*

  Ejemplo: TABLA_MAS_DOS DW TABLA + 2 ;se define un objeto, TABLA_MAS_DOS, tipo palabra inicializándola al valor de TABLA más dos.

- Operador: **-**

  Formato: *operando1 - operando2.*

  Función: resta *operando2* de *operando1.*

  Ejemplo: DIFER DW TABLA1 - TABLA2 ;se define un objeto, DIFER, inicializándolo a la diferencia de los valores de TABLA1 y TABLA2.

- Operador: *****

  Formato: *operando1 * operando2.*

  Función: multiplica *operando1* por *operando2.*

  Ejemplo: TOTAL EQU 60 * 24 ;se pone de nombre TOTAL, al valor resultado de la multiplicación, 1440.

- Operador: /

  Formato: *operando1 / operando2.*

  Función: divide *operando1* entre *operando2* y retorna el cociente.

  Ejemplo: COCIENTE EQU 31416/10000 ;se pone de nombre COCIENTE al resultado de la división, 3.

- Operador: **MOD**

  Formato: *operando1 MOD operando2.*

  Función: divide *operando1* entre *operando2* y retorna el resto.

  Ejemplo: RESTO EQU 31416/10000 ;se pone de nombre RESTO al resto de la división, 1416.

- Operador: **SHL** (SHift Left)

  Formato: *operando SHL expresión.*

  Función: desplaza a la izquierda en *operando* el número de bits que indica *expresión.*

  Ejemplo: DATO1 EQU 110010b

  DATO2 EQU DATO1 SHL 2

  ;DATO2 es un nombre que se pone al valor 11001000b.

– Operador: **SHR** (SHift Right)

Formato. *operando SHR expresión.*

Función: desplaza a la derecha en *operando* el número de bits que indica *expresión.*

Ejemplo: DATO1 EQU 110010b

DATO2 EQU DATO1 SHR 2

;DATO2 es un nombre que se pone al valor 1100b.

2.4.2. Operadores lógicos

Operan sobre valores binarios bit a bit:

– Operador: **AND**

Formato: *operando1 AND operando2.*

Función: calcula el valor lógico "Y" de *operando1* y *operando2.*

Ejemplo: DATO EQU 00110100b AND 11010111b

;DATO sería el nombre del valor 00010100b.

– Operador: **OR**

Formato: *operando1 OR operando2.*

Función: calcula el valor lógico "O" de *operando1* y *operando2.*

Ejemplo: DATO EQU 00110100b OR 11010111b

;DATO sería el nombre del valor 11110111b.

– Operador: **XOR**

Formato: *operando1 XOR operando2.*

Función: calcula el valor lógico "O exclusivo" de *operando1* y *operando2.*

Ejemplo: DATO EQU 00110100b XOR 11010111b

;DATO sería el nombre del valor 11100011b.

– Operador: **NOT**

Formato. *NOT operando.*

Función: obtiene el valor opuesto a cada bit.

Ejemplo: DATO EQU NOT 00110100b

;DATO sería el nombre del valor 11001011b.

2.4.3. Operadores relacionales

Comparan dos valores numéricos o dos direcciones de memoria del mismo segmento y producen "0", si la relación es falsa, o "0FFFFh", si la relación es verdadera:

– Operador: **EQ** (EQual)

Formato: *operando1 EQ operando2.*

Función: verdad si los dos operandos son iguales.

Ejemplo: MOV AX, DATO EQ 15

;si DATO vale 15, AX tomará el valor 0FFFFh, si vale algo distinto de 15, AX tomará el valor 0.

– Operador. **NE** (Not Equal)

Formato: *operando1 NE operando2.*

Función: verdad si los dos operandos son distintos.

Ejemplo: MOV AX, DATO NE 15

;si DATO vale 15, AX tomará el valor 0, si vale algo distinto de 15, AX tomará el valor 0FFFFh.

– Operador: **LT** (Less Than)

Formato: *operando1 LT operando2.*

Función: verdad si el *operando1* es menor que el *operando2.*

Ejemplo: MOV AX, DATO LT 15

;si DATO es menor que 15, AX tomará el valor 0FFFFh, si no, AX tomará el valor 0.

– Operador: **GT** (Greater Than)

Formato: *operando1 GT operando2.*

Función: verdad si el *operando1* es mayor que el *operando2.*

Ejemplo: MOV AX, DATO GT 15

;si DATO es mayor que 15, AX tomará el valor 0FFFFh, si no, AX tomará el valor 0.

– Operador: **LE** (Less or Equal)

Formato: *operando1 LE operando2.*

Función: verdad si el *operando1* es menor o igual que el *operando2.*

Ejemplo: MOV AX, DATO LE 15

;si DATO es menor o igual que 15, AX tomará el valor 0FFFFh, si no, AX tomará el valor 0.

– Operador: **GE** (Greater or Equal)

Formato: *operando1 GE operando2.*

Función: verdad si el *operando1* es mayor o igual que el *operando2.*

Ejemplo: MOV AX, DATO GE 15

;si DATO es mayor o igual que 15, AX tomará el valor 0FFFFh, si no, AX tomará el valor 0.

2.4.4. Operadores de retorno de valores

Son operadores pasivos, que se encargan de suministrar información sobre las variables y las etiquetas del programa:

– Operador: **SEG**

Formato: *SEG operando*.

Función: devuelve el valor del segmento de la variable o de la etiqueta que va en el campo *operando.*

Ejemplo: MOV AX, SEG DATO

;AX tomará el valor del segmento donde se encuentre DATO.

– Operador: **OFFSET**

Formato: *OFFSET operando*.

Función: devuelve el valor del desplazamiento de la variable o de la etiqueta que va en el campo *operando.*

Ejemplo: MOV AX, OFFSET DATO

;AX tomará el valor del desplazamiento de DATO.

– Operador: **TYPE**

Formato: *TYPE operando.*

Función: el *operando* podrá ser una variable o una etiqueta; si es una variable, devuelve:

1: si la variable está definida como byte.

2: si la variable está definida como palabra.

4: si la variable está definida como doble palabra.

6: si la variable está definida como palabra larga.

8: si la variable está definida como cuádruple palabra.

10: si la variable está definida con DT (directiva para reservar diez bytes).

Si el *operando* es una etiqueta, devuelve,

-1: 0FFFFh: si la etiqueta es NEAR.

-2: 0FFFEh: si la etiqueta es FAR.

Ejemplo: DATO DD 35 ;define una doble palabra inicializada al valor 35.

MOV AX, TYPE DATO

;AX pasaría a tener valor 4.

– Operador: **SIZE**

Formato: *SIZE variable.*

Función: devuelve el número de bytes reservados para *variable*; sólo se aplica a variables definidas con DUP.

Ejemplo: DATO DW 20 DUP(0) ;se define dato como 20 palabras inicializadas a cero.

MOV AX, SIZE DATO

;AX pasaría a tener valor 40.

– Operador: **LENGTH**

Formato: *LENGTH variable.*

Función: devuelve el número de unidades (bytes o palabras) reservados para *variable*; sólo se aplica a variables definidas con DUP.

Ejemplo: DATO DW 20 DUP(0) ;se define dato como 20 palabras inicializadas a cero.

MOV AX, LENGTH DATO

;AX pasaría a tener valor 20.