LENGUAJE ENSAMBLADOR

TEMAS:

MODOS DE DIRECCIONAMIENTO

SALTOS

ESTRUCTURAS DE CONTROL

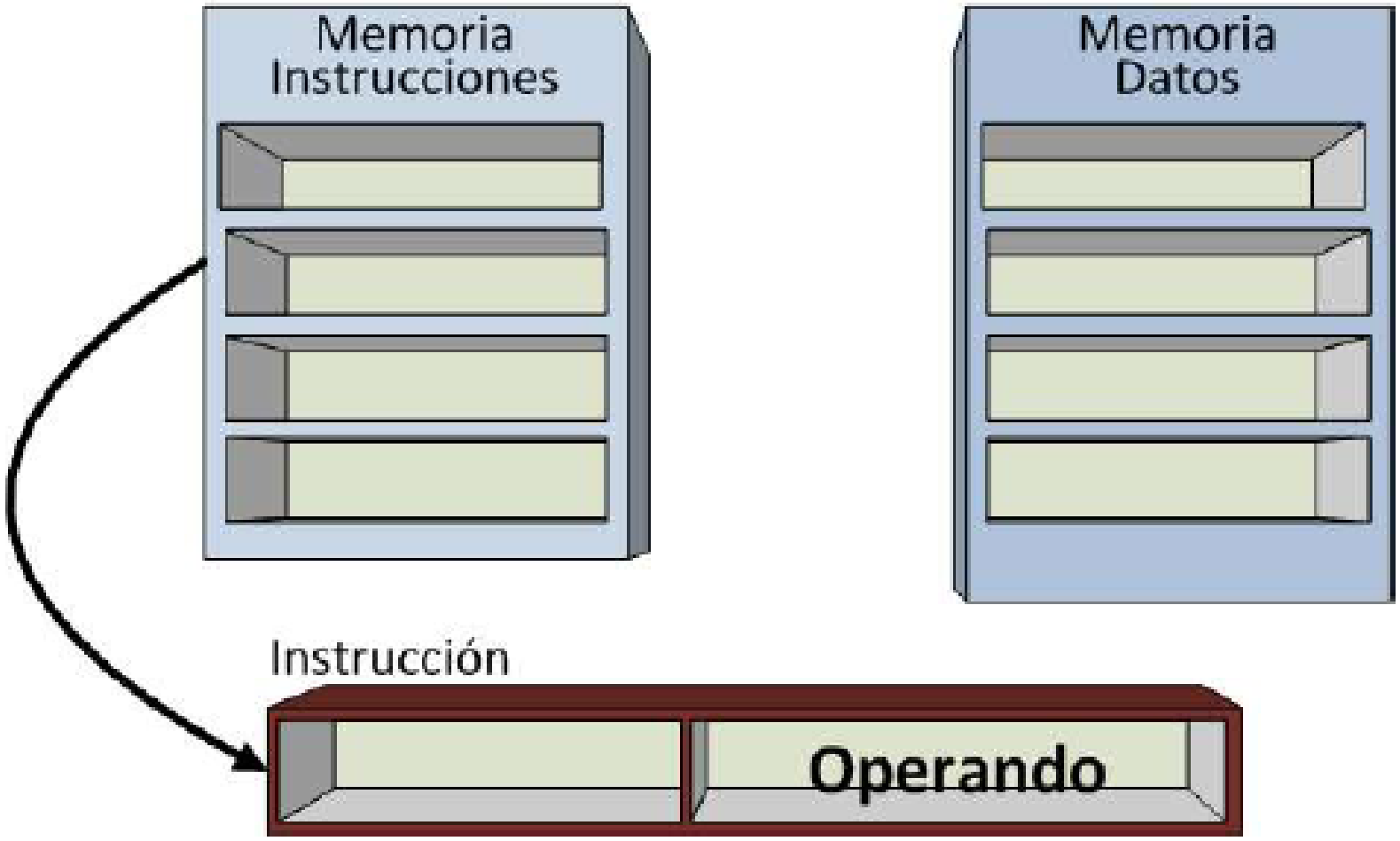
MODOS DE DIRECCIONAMIENTOS EN EL MSX88

- Los modos de direccionamiento son las diferentes maneras de **especificar un operando dentro de una instrucción** en lenguaje ensamblador.
- Un modo de direccionamiento especifica la forma de **calcular la dirección de memoria efectiva** de un operando mediante el uso de la información contenida en registros y/o constantes, contenida dentro de una instrucción de la máquina o en otra parte.
- Vamos a tomar la referencia del operando fuente para indicar el modo de direccionamiento.

MODOS DE DIRECCIONAMIENTO

- Inmediato
- Directo de memoria o Absoluto
- Directo de Registro
- Indirecto de memoria (en desuso)
- Indirecto con registro
- Indirecto con Desplazamiento
  - basado, indexado o relativo al PC
  - Pila (o relativo al SP)

MDD INMEDIATO



MDD INMEDIATO

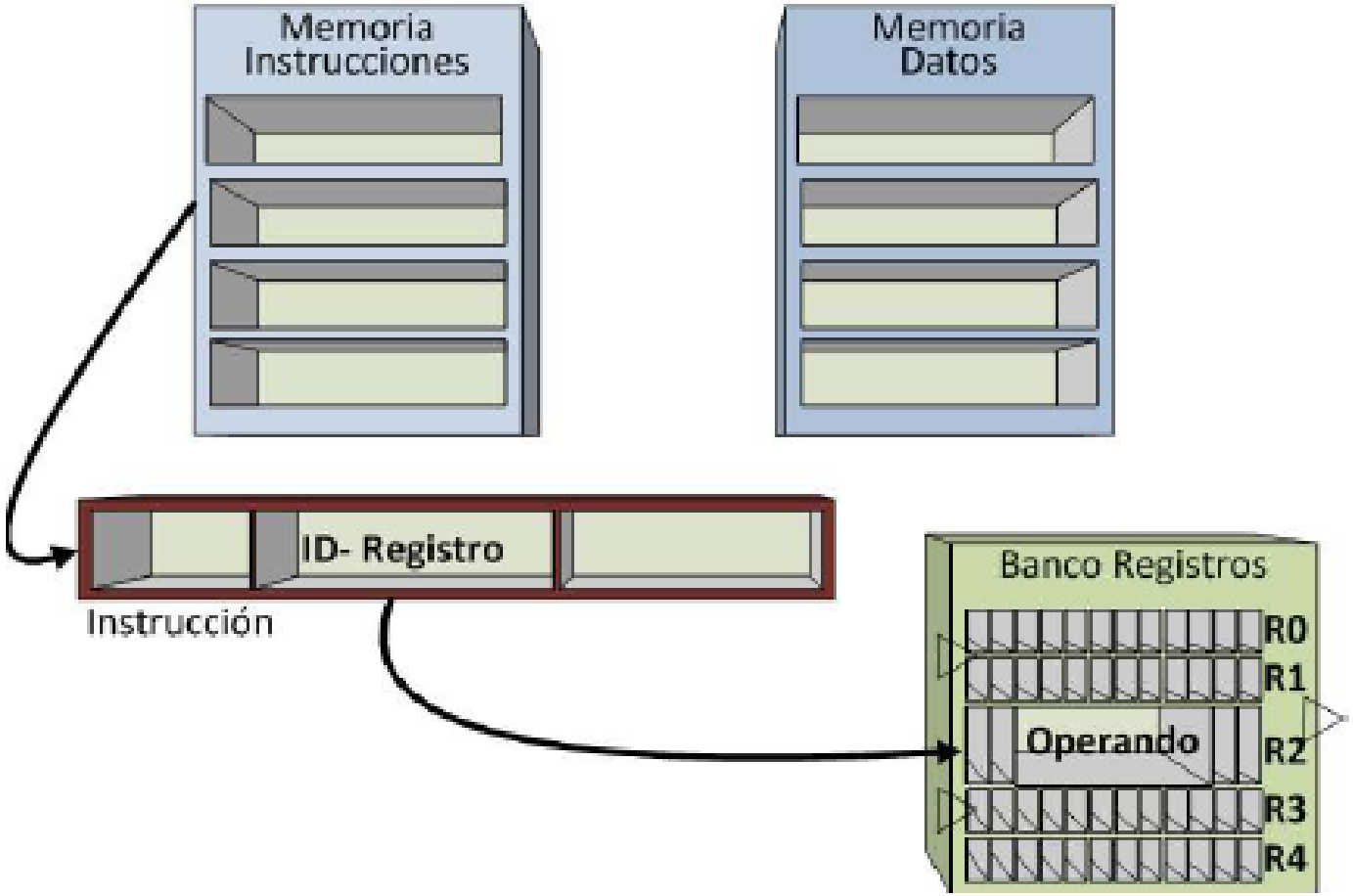
Inmediato: En este modo el operando es especificado en la instrucción misma. Un valor numérico será guardado en un registro o en una posición de memoria No hay referencias adicionales a la memoria después de la búsqueda de la instrucción

```
COLOR EQU 4
X DB  0
Y DW 0

MOVE AX, 5
MOV BX, COLOR
```

Cod Operación	*Operando*

MMD DIRECTO DE REGISTRO



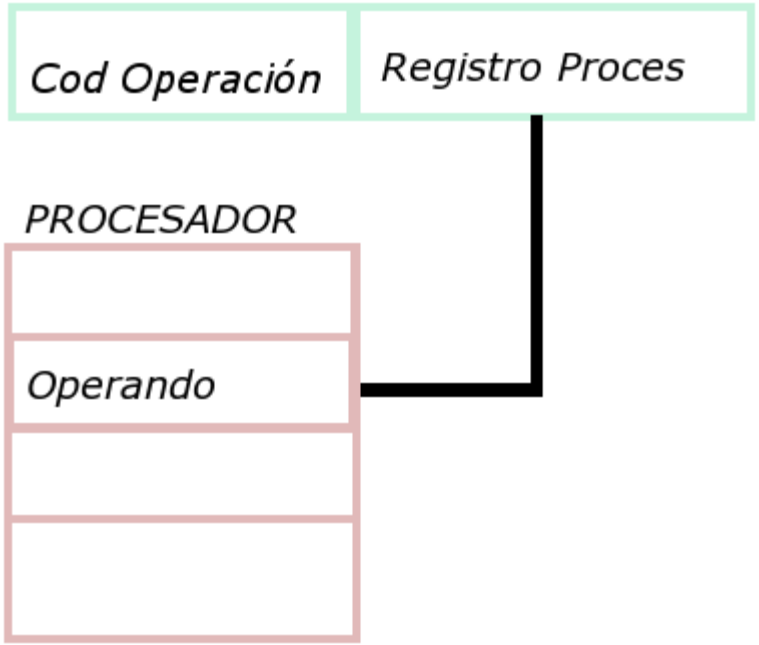
MDD REGISTRO

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

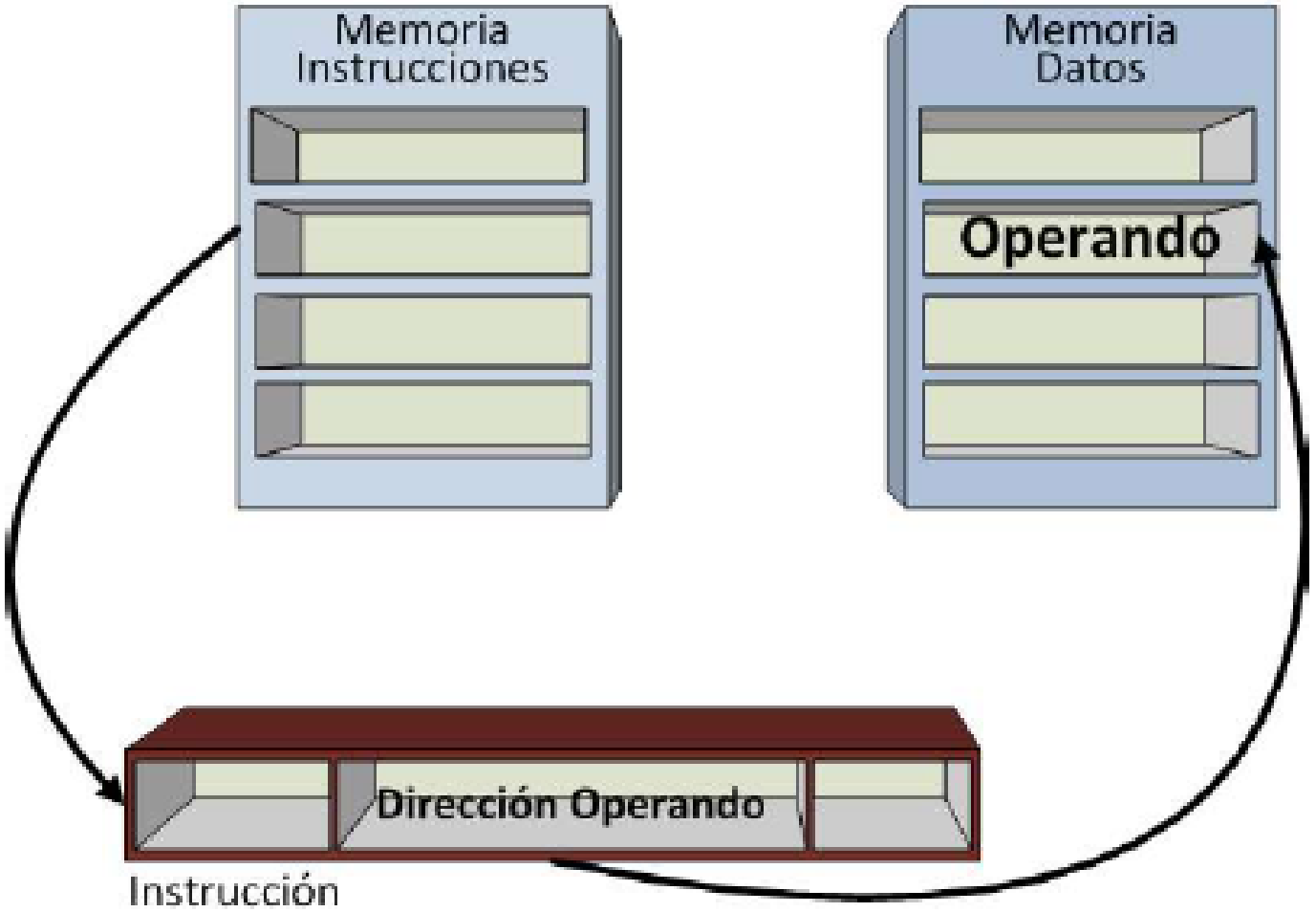
AX, BX, CX, etc.

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
```



MMD DIRECTO O ABSOLUTO (DE MEMORIA)



MDD DIRECTO MEMORIA

Directo: en el direccionamiento directo copiamos datos a registro a memoria o a de un registro a memoria.

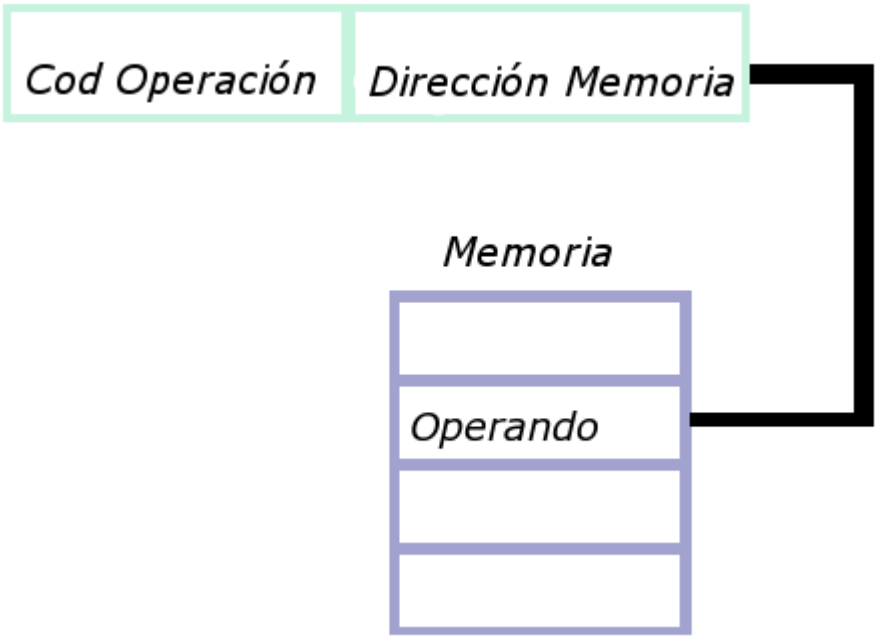
```
X DB  0
Y DW 0
```

MOV AL, X *; se copia el byte contenido de la memoria de la etiqueta X al registro AL*

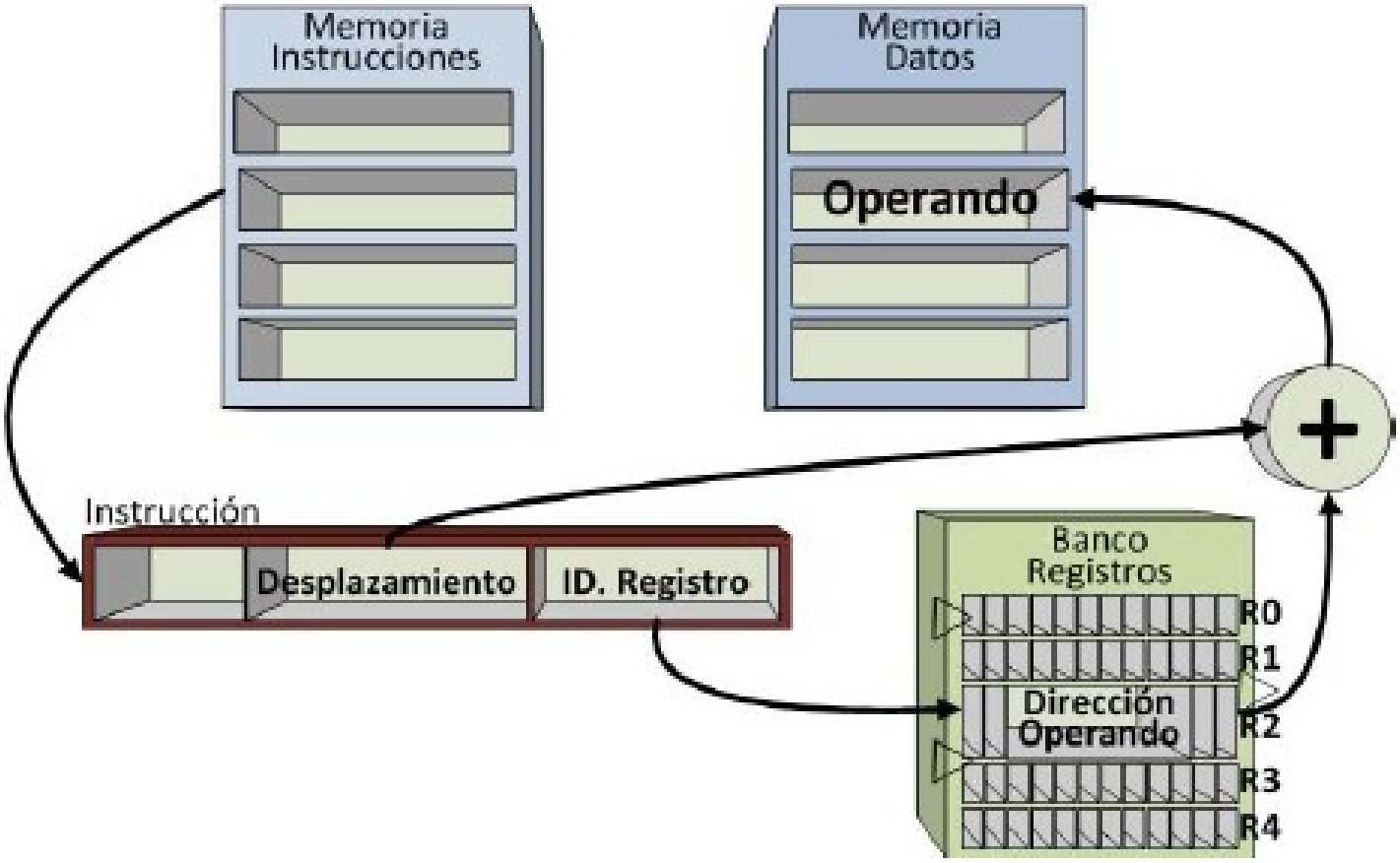
MOV AX, Y*; se copia la palabra contenida en la memoria de la etiqueta Y al registro AX*

¿Este direccionamiento es directo?
MOV X, AH*; se copia el byte del registro AH en la posición de memoria de la etiqueta X*

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



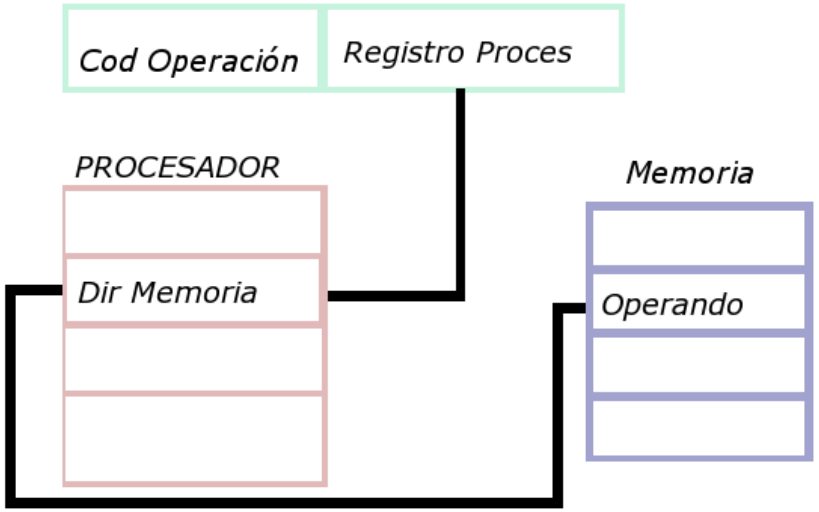
MDD INDIRECTO VIA REGISTRO



MDD VIA REGISTRO

Direccionamiento indirecto vía registro:

- El campo de operando de la instrucción contiene un identificador de registro en el que se encuentra la dirección efectiva del operando.
- En este modo el campo de la dirección de la instrucción da la dirección en donde la dirección efectiva se almacena en la memoria. La dirección efectiva en este modo se obtiene del siguiente cálculo:
- Dir. efectiva = Dir. de la parte de la instrucción + Contenido del registro del procesador.



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

MOV BX, OFFSET LISTA ; ***direccionamiento inmediato***

MOV AL, [BX]; ***direccionamiento indirecto vía registro****: estamos guardando en AL el byte de la LISTA que se encuentra indicado por el contenido del registro BX*

¿Esta instrucción tiene direccionamiento indirecto vía registro?

MOV [BX], AX; guarda la palabra que se encuentra en AX, en la posición de memoria indicada por el contenido de BX ¿Cómo está quedando la TABLA?

FLAGS DE LA CPU

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.

Decimal
Codificado
en **B**inario

FLAGS DE LA CPU

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.

INSTRUCCIONES DE SALTO

```
JMP dirección      ; Salta siempre

JZ  dirección      ; Salta si el flag Z=1
JNZ dirección      ; Salta si el flag Z=0
JS  dirección      ; Salta si el flag S=1
JNS dirección      ; Salta si el flag S=0
JC  dirección      ; Salta si el flag C=1
JNC dirección      ; Salta si el flag C=0
JO  dirección      ; Salta si el flag O=1
JNO dirección      ; Salta si el flag O=0
```

EJEMPLOS DE SALTOS

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

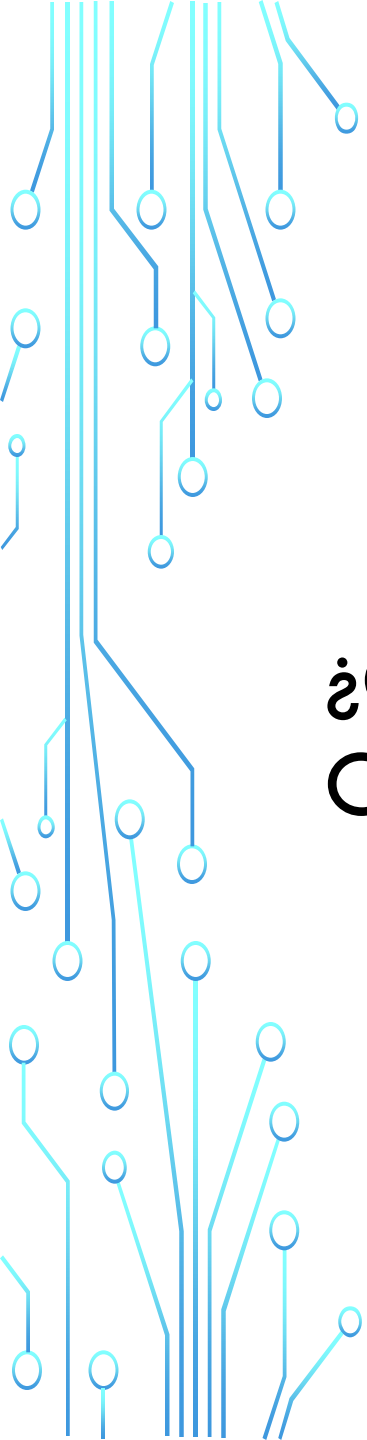
Ejemplo:

```
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:
```



¿CÓMO ARMAR ESTRUCTURAS DE CONTROL EN ASSEMBLER?

¿CÓMO HACER UN IF?

En pascal sería:

```
IF AL = 4 THEN
  BEGIN
     BL = 1;
     CL = CL + 1;
END;
```

En assembler:

```
          CMP AL, 4 ; (a)
          JZ Then ; (b)
          JMP Fin_IF ; (c)
Then:     MOV BL, 1 ; (d)
          INC CL ; (e)
Fin_IF: HLT
```

CMP alterará los flags y en particular, nos interesa ver al flag Z, ya que si dicho flag está en 1, implica que al resta AL con 4, el resultado dio 0, por lo que AL tiene que valer 4.Entonces, si esa condición es verdadera, deberíamos ejecutar las instrucciones que están dentro del THEN. Si no, deberíamos evitar ejecutarlas.

Si la comparación en (a) establece el flag Z en 1, el salto en (b) se produce, haciendo que la ejecución continúe en la etiqueta Then. Ahí, se ejecutan las instrucciones (d) y (e) que hacen lo que se encuentra en el THEN del IF y continúa la ejecución en la instrucción apuntada por la etiqueta Fin_IF .

Si el flag Z quedó en 0 en (a), el salto en (b) no se produce, por lo que la ejecución continúa en la

próxima instrucción, el JMP en (c), que saltea las instrucciones y continúa la ejecución en la instrucción apuntada por la etiqueta Fin_IF , que señala el final del IF

IF …. THEN … ELSE

```
IF AL = 4 THEN
   BEGIN
      BL = 1;
      CL = CL + 1;
    END
ELSE
    BEGIN
       BL = 2;
       CL = CL – 1;
    END;
```

```
        ORG 3000h
        CMP AL, 4 ;
        JZ Then ;
        JMP Else ;
Then:   MOV BL, 1 ;
        INC CL ;
        JMP Fin_IF ;
Else:   MOV BL, 2 ;
        DEC CL ;
Fin_IF: HLT ;
        END
```

Este salto incondicional hace que no se ejecute el else si es que pasé por el then

LAZOS

```
ORG 2000h
        MOV AX, 10
Lazo: ... ;
          ... ; <Instrucciones a
  repetir>
          ... ;
        DEC AX
        JNZ Lazo
        JMP Fin
```

El ejemplo plantea un esquema básico de iteración usado comúnmente como algo equivalente a un **“ For i := n downto 1”** de Pascal. En AX tengo la cantidad de repeticiones

El programa inicializa AX en 10, hace lo que tenga que hacer dentro del bucle, decrementa AX en 1 y salta a la instrucción apuntada por la etiqueta *Lazo* (el DEC) siempre y cuando el flag Z quede en 0, o sea, que el resultado de la operación anterior no haya dado como resultado 0.

Como AX se decrementa en cada iteración, llegará un momento en que AX será 0, por lo que el salto condicional no se producirá y continuará la ejecución del programa en la siguiente instrucción luego de dicho salto.

LAZOS. ¿QUÉ ESTRUCTURA DE CONTROL IMPLEMENTA ESTE CÓDIGO?

```
        MOV AL, 0
        MOV CL, 10
Volver: ADD AL, AL
        DEC CL
        CMP CL, 1
        JNZ Volver
Fin:    HLT
```

Podemos verlo como un **Repeat….until** de Pascal, donde sabemos la cantidad de veces, pero podría ser que no supiéramos cuantas veces ejecuta.

¿Podemos verlo como For este caso? SI también se lo puede interpretar como FOR

¿Cómo hacemos un **While <condición> do** , de Pascal?

MDD VIA REGISTRO - EJEMPLO

```
        ORG 1000h
        tabla DB 1, 2, 3, 4, 5
        fin_tabla DB ?
        resultado DB 0
        ORG 2000h
        MOV BX, OFFSET tabla ; (a)
        MOV CL, OFFSET fin_tabla – OFFSET tabla ; (b)
Loop:   MOV AL, [BX] ; (c)
        INC BX ; (d)
        XOR resultado, AL ; (e)
        DEC CL
        JNZ Loop
        HLT
        END
```

(a) Inicializa BX con "OFFSET tabla". Esto indica que se debe cargar en BX la dirección de tabla, no el contenido de dicha variable.

(b) Se asigna a CL la diferencia entre la dirección de tabla y la dirección de fin_tabla. Lo que se logra es calcular cuantos bytes hay entre el comienzo y el final de la tabla. De esta manera, obtenemos la cantidad de datos que contiene la tabla.

(c) Vemos que en el MOV aparece BX entre corchetes. Esto significa que se debe asignar en AL el contenido de la celda de memoria cuya dirección es el valor contenido en BX. Así, como BX se había inicializado con la dirección del comienzo de la tabla, esto causa que se cargue en AL el contenido de la primer entrada de la tabla.

(d) Se incrementa BX, con lo que ahora apunta al siguiente byte de la tabla

(e) Se calcula una operación XOR con el contenido de la variable resultado y el byte que se acaba de obtener en AL, dejando el resultado de esa operación en la misma variable.

EJERCICIO

1. REALIZAR PASO A PASO LOS MOMENTOS DE LOS REGISTROS DE LA CPU
2. ARMAR LA MEMORIA DE DATOS
3. ¿QUÉ REALIZA EL SIGUIENTE PROGRAMA?

Tener en cuenta que la tabla es de Word (2 bytes)

```
        ORG 1000h
        tabla dw 5, 2, 10, 4, 5, 0, 4, 8, 1, 9
        min dw 0FFFFh
        ORG  2000h
        MOV BX, OFFSET tabla
        MOV CL, 0
        MOV AX, min
Loop:   CMP [BX], AX
        JNC Menor
        MOV AX, [BX]
Menor:  ADD BX, 2
        INC CL
        CMP CL, 10
        JNZ Loop
        MOV min, AX
        HLT
        END
```

Recordar que el CMP hace **Destino – Fuente** y no almacena nada en destino. Solo altera los flags de la CPU

¿Que valor queda en min?

BIBLIOGRAFÍA E INFORMACIÓN

- Organización y Arquitectura de Computadoras. W. Stallings, 5ta Ed.

  Repertorios de instrucciones
  - Capítulo 9: características y funciones
  - Capítulo 10: modos de direccionamiento y formatos
  - Apéndice 9A: Pilas

  Ciclo de instrucción:
  - Capítulo 3 apartado 3.2.
  - Capítulo 11 apartados 11.1. y 11.3.

  Organización de los registros
  - Capítulo 11 apartado 11.2.

  Formatos de instrucciones
  - Capítulo 10 apartado 10.3. y 10.4.
- Simulador MSX88

Link de interés: www.williamstallings.com