APÉNDICE D

Directivas, Operadores e Instrucciones del Lenguaje Ensamblador

DIRECTIVAS (PSEUDO-OPERACIONES)

- Las directivas son comandos que afectan al ensamblador, y no al microprocesador, por lo que no generan código objeto. Se utilizan para definir segmentos, símbolos, procedimientos o subrutinas, reservar memoria, etc.

Directiva EQU (EQUivalence)

Sintaxis: `nombre EQU expresión`

- La directiva EQU asigna un nombre simbólico al valor de una expresión. El compilador, cuando encuentre en el código dicho nombre simbólico, lo sustituirá por el valor de la expresión. La expresión indicada puede ser:

1. Una constante numérica.
2. Una referencia de dirección (cualquier modo de direccionamiento).
3. Cualquier combinación de símbolos y operaciones que generen un valor numérico.
4. Otro nombre simbólico.

Ejemplos:

```
COLUMNAS    EQU    80
FILAS       EQU    25
PANTALLA    EQU    FILAS*COLUMNAS
MOVER       EQU    MOV
EDICION     EQU    "PULSAR TECLA"
DIRECCION   EQU    [SI+BX]
```

Directiva DB (Define Byte)

Sintaxis: `[nom_variable] EQU expresión`

- La directiva DB reserva memoria para datos de tipo byte (8 bits), o para una variable de este tipo, inicializando o no dicho byte y los posteriores.
- El nombre de la variable es opcional, y se asigna dicho nombre para el primer byte (recordar que el lenguaje máquina no entiende de variables, si no de direcciones de memoria. Una variable es una etiqueta o referencia para el compilador, el cual sustituye cada referencia a esta variable por la dirección de memoria correspondiente).

- La expresión es el valor inicial de la variable, y puede ser:

  1. Una constante numérica positiva o negativa con rango de byte (-128 a +127, ó de 0 a 255).
  2. Una cadena de caracteres delimitada por comillas simples o dobles.
  3. Un signo interrogante (?), que indica valor indefinido. Sólo reserva espacio en la memoria.
  4. **n DUP(expresión)**, que repite 'n' veces la 'expresión'.

```
Ejemplos:       VALOR   DB      111
                TEXTO   DB      "HOLA, SOY UN TEXTO"
                CALCULO DB      15*2
                RESERVA DB      15 DUP(?)
                NODEFIN DB      ?
```

- De modo similar a DB, se pueden definir otros tipos de variables en lenguaje ensamblador. Lo único que varía es el tamaño de los datos y el nombre de la directiva:

DB (Define Byte)	**Reservar datos de tamaño byte (8 bits)**
DW (Define Word)	**Reservar datos de tipo palabra (16 bits)**
DD (Define Doubleword)	**Reservar datos de tipo doble palabra (32 bits)**
DQ (Define Quadword)	**Reservar datos de tipo quádruple palabra (64 bits)**
DT (Define Terabyte)	**Reservar datos de tipo terabyte (80 bits o 10 bytes)**

Directiva SEGMENT

Sintaxis:

```
nombre  SEGMENT [alineamiento] [combinación] ['clase']
   ...
   ...
   ...
nombre  ENDS
```

- Las directivas SEGMENT y ENDS marcan el principio y el final del segmento cuyo nombre se especifica.
- Un segmento es un bloque de sentencias que puede contener definiciones de variables y/o instrucciones.
- El parámetro 'alineamiento' es opcional, e indica cómo se alineará el segmento en la memoria, y sus posibles valores son los siguientes:
  - **BYTE** : Ninguno. El segmento puede empezar en cualquier posición.
  - **WORD** : El segmento empieza en una frontera de palabra. La dirección de comienzo es múltiplo de 2.
  - **PARA** : El segmento empieza en una frontera de párrafo. La dirección de comienzo es múltiplo de 16. Es el valor por defecto.
  - **PAGE** : El segmento empieza en una frontera de página. La dirección de comienzo es múltiplo de 256.

- El parámetro 'combinación' es también opcional, y se utiliza para combinar segmentos que tengan el mismo nombre ('nombre' se refiere al indicado en el parámetro 'clase', no el nombre del propio segmento). Sus posibles valores son los siguientes:
  - **PUBLIC** : El segmento se concatenará a otros del mismo 'nombre' en la fase de montaje (link).
  - **COMMON** : El segmento, y todos los del mismo 'nombre', empezarán en la misma dirección, solapándose entre sí.
    AT <expresión> : El segmento se ubicará en la dirección cuyo segmento es<expresión>. Esta dirección no se usa para forzar la carga del segmento en una dirección fija. Lo que sí se permite es definir variables dentro del mismo segmento. Se usa normalmente para referenciar zonas de memoria, como los vectores de interrupción o la memoria ROM.
  - **STACK** : Define el segmento como segmento de pila (stack).
  - **MEMORY** : El segmento se ubicará en una dirección superior a la de los otros segmentos. Sólo puede haber un segmento de este tipo.
- El parámetro 'clase' es el nombre que se utiliza para agrupar segmentos en la fase de montaje. Este nombre ha de especificarse entre comillas simples.

Directiva ASSUME

Sintaxis:

```
ASSUME  reg_segmento:nom_segmento,reg_segmento:nom_segmento...
ASSUME  NOTHING
```

- Indica al ensamblador qué segmento va direccionar cada uno de los registros de segmento. Esta directiva sigue, normalmente, a la directiva SEGMENT.
- El parámetro 'reg_segmento' puede ser: *CS, DS, ES ó SS*.
- El parámetro 'nom_segmento' puede ser:

  a) El nombre asignado por la directiva SEGMENT.
  b) El nombre de un grupo.
  c) SEG variable.
  d) SEG etiqueta.
  e) la palabra NOTHING.

- La palabra NOTHING anula una directiva ASSUME anterior.
- La directiva ASSUME no inicializa los registros de segmento, simplemente conduce al compilador dónde está cada uno y su uso (pila, segmento, datos o extra). Para inicializar, por ejemplo, el registro DS, hay que indicar:

```
MOV   AX,nom_segmento_datos
MOV   DS,AX
```

- Lo mismo se haría con ES, no así con CS y SS, los cuales inicializa el cargador de programas del sistema operativo.

Directiva PROC (PROCedure)

Sintaxis:

```
nombre  PROC    [atributo]
   ...
   ...
   ...
nombre  ENDP
```

- Las directivas PROC y ENDP definen el inicio y el final de un procedimiento, que es un conjunto de sentencias a las que se puede acceder directamente llamando al procedimiento. Un procedimiento posee un nombre, el cual es utilizado para ser llamado.
- El parámetro 'atributo' es opcional, y puede ser NEAR ó FAR (por defecto es NEAR). Si el procedimiento es el principal de un programa, ha de utilizarse el atributo FAR.
- Para que un procedimiento en un módulo sea accesible desde otros módulos se ha de especificar la directiva PUBLIC.
- Ejemplo:

```
PUBLIC  PROC1           ; Se define como público
PROC1   PROC    FAR     ; comienzo del procedimiento (lejano)
        (instrucciones) ; Instrucciones del procedimiento
        RET             ; Instrucción para retornar
PROC1   ENDP            ; Final del procedimiento
```

- Para llamar a un procedimiento se utiliza la instrucción CALL:

```
CALL nombre_procedimiento
```

Directiva END

Sintaxis: `END [expresión]`

- Indica el final del programa fuente. El operando 'expresión' indica la dirección de comienzo del programa fuente, que es, normalmente, una etiqueta.
- Ejemplo:

```
END
END     INICIO
```

OPERADORES

Operador OFFSET

Sintaxis:

```
OFFSET  <variable>
OFFSET  <etiqueta>
```

- Devuelve el OFFSET (desplazamiento) de la variable o etiqueta especificada. El desplazamiento es la posición desde el principio del segmento hasta la expresión indicada.

- Ejemplo:

```
MOV     DX,OFFSET TEXTO ; Mover a DX el desplazamiento ó
                        ; posición de la variable TEXTO
```

Operador DUP (DUPiclate)

Sintaxis: **`num DUP(valor)`**

- Define en memoria la repetición de 'num' veces de 'valor'.
- El parámetro 'valor' puede ser un valor entero, una cadena de caracteres entrecomillada u otro operador.
- Ejemplos:

```
DB      20 DUP(100)     ; Repetir 20 veces 100
DB      64 DUP('HOLA')  ; Repetir 64 veces 'HOLA'
DB      256 DUP(?)      ; Asignar 256 bytes indefinidos
```

INSTRUCCIONES

Instrucción MOV (MOVe)

Sintaxis: **`MOV op_destino,op_fuente`** **`; Sintaxis básica`**

FORMATOS	EJEMPLOS
MOV reg8,reg8	MOV AH,BL
MOV reg16,reg16	MOV AX,BX
MOV segreg,reg16	MOV ES,AX
MOV reg16,segreg	MOV CX,DS
MOV reg,mem	MOV CH,[AA55H]
MOV mem,segreg	MOV [DI],DS
MOV segreg,mem	MOV SS,[4584H]
MOV reg,inmediato	MOV CX,2450H
MOV mem,inmediato	MOV BYTE PTR [SI],20

- Mueve o transfiere un byte o una palabra desde el operando fuente al operando destino. El dato transferido se copia, por lo que no desaparece del operando fuente. Tanto el operando fuente como el operando destino pueden ser un registro o un elemento de memoria. El operando fuente puede ser un valor inmediato.

- Ejemplos:

```
MOV     AX,BX                   ; AX=BX
MOV     DL,CH                   ; DL=CH
MOV     BX,1500                 ; BX=1500
MOV     AX,[2458H]              ; AX=contenido en DS:2458H
MOV     [2458H],AL              ; contenido en DS:2458H=AL
MOV     [BX],AL                 ; contenido en DS:BX=AL
MOV     AL,[BX]                 ; AL=contenido en DS:BX
MOV     DS,AX                   ; DS=AX
```

```
MOV     CL,[SI+4]               ; CL=contenido en DS:SI+4
MOV     AX,[BP+7]               ; AX=contenido en SS:BP+7
MOV     [BX+DI-4],DX            ; contenido en DS:BX+DI-4=DX
MOV     DS,[BP+SI]              ; DS=contenido en SS:BP+SI
MOV     AX,[BP+SI+4]            ; AX=contenido en SS:BP+SI+4
MOV     BYTE PTR [1547H],25     ; contenido en DS:1547H=25
MOV     WORD PTR [1457H],AA55H  ; contenido en DS:1457H=AA55H
```

- El tamaño de los dos operandos han de ser el mismo (8 bits o 16 bits). En la siguiente expresión:

```
MOV     AX,1
```

- El operando fuente se traduciría a un operando de 16 bits (0001H), por lo que AH valdría 0, yAL valdría 1. En los siguientes ejemplos:

```
MOV     AX,[4520H]
MOV     AL,[4520H]
```

- El ensamblador reconoce enseguida el tamaño de los operandos. En el primer caso se realiza un acceso a memoria, leyendo una palabra. En el segundo caso se realiza un acceso a memoria, leyendo 8 bits.
- Cuando el operando se encierra entre corchetes, indica "el contenido de memoria en", es decir, que no se referencia a un valor sino a una posición de memoria. Normalmente, las referencias a memoria se basan en el segmento DS, y se direcciona a través de una posición predeterminada, o usando los registros BX, SI ó DI como punteros base que indican la posición dentro del segmento.
- Si se utiliza el registro BP, éste direcciona al segmento de pila (SS).
- Cuando se hace una escritura directa a memoria, es necesario indicar qué tamaño posee el dato. Para ello, se ha indicar un puntero a memoria, de tamaño BYTE o WORD:

```
MOV     BYTE PTR [4520H],25     ; Escritura en memoria de 8 bits
MOV     WORD PTR [4520H],25     ; Escritura en memoria de 16 bits
                                  (4520H=25 y 4521H=0)
```

- Para referenciar a una posición de memoria localizada en otro segmento, se escribe el nombre del registro de segmento antes de los corchetes:

```
MOV     ES:[45],AL              ; Escribir en ES:45 el valor de AL
```

- Los registros de segmento no se pueden inicializar con un valor inmediato, por lo que es necesario un registro intermedio de 16 bits o un valor WORD en una posición de memoria.

INSTRUCCIONES ARITMETICAS

Instruccione INC (INCrement)

Sintaxis:

```
INC     reg
INC     mem
```

- Incrementa (suma 1 a) el contenido de un registro o de una posición de memoria.
- Ejemplos:

```
INC     AX                  ; AX=AX+1
INC     DL                  ; DL=DL+1
INC     WORD PTR ES:[DI+4]  ; Increm. palabra contenida
                            ; en ES:DI+4
```

Instruccion DEC (DECrement)

Sintaxis:

```
DEC     reg
DEC     mem
```

- Decrementa (resta 1 a) el contenido de un registro o de una posición de memoria.
- Ejemplos:

```
DEC     AX                  ; AX=AX-1
DEC     DL                  ; DL=DL-1
DEC     BYTE PTR ES:[DI+4]  ; Decrem. byte contenido en
                            ; ES:DI+4
```

Instruccion ADD (ADDition)

Sintaxis: `ADD op_destino, op_fuente ; Sintaxis básica`

```
FORMATOS                    EJEMPLOS
ADD     reg,inmediato       ADD AX,3500
ADD     mem,inmediato       ADD BYTE PTR [SI],35
ADD     reg,reg             ADD BX,DX
ADD     mem,reg             ADD [BX],AX
ADD     reg,mem             ADD AH,[BX]
```

- Suma al operando de destino el valor o contenido del operando fuente, almacenándose el resultado en el operando de destino. Ambos operandos han de ser del mismo tamaño (byte o palabra).

Instruccion SUB (SUBstract)

Sintaxis: `SUB op_destino, op_fuente ; Sintaxis básica`

```
FORMATOS                    EJEMPLOS
SUB     reg,inmediato       SUB AX,3500
SUB     mem,inmediato       SUB BYTE PTR [SI],35
SUB     reg,reg             SUB BX,DX
SUB     mem,reg             SUB [BX],AX
SUB     reg,mem             SUB AH,[BX]
```

- Resta al operando de destino el valor o contenido del operando fuente, almacenándose el resultado en el operando de destino. Ambos operandos han de ser del mismo tamaño (byte o palabra).

Instrucciones MUL (MULTiply) e IMUL

Sintaxis:

```
MUL    reg
MUL    mem
IMUL   reg
IMUL   mem
```

- Realiza una multiplicación con el acumulador. Si se realiza con AL, el operando de la instrucción debe ser de 8 bits, y ambos generan un resultado de 16 bits que se almacena en el registro AX. Sin embargo, si se realiza con AX, el operando de la instrucción debe ser de 16 bits, con lo que el resultado será de 32 bits y se almacenará en el par de registros DX (palabra de mayor peso) yAX (palabra de menor peso).
- La diferencia entre MUL e IMUL, es que la instrucción MUL realiza una multiplicación sin signo (sin complemento a dos), mientras que la instrucción IMUL realiza una multiplicación con signo (con complemento a dos).
- Ejemplos:

```
MUL     BL              ; AX=AL*BL
MUL     CX              ; DXAX=AX*CX
MUL     BYTE PTR [BX]   ; AX=AL*byte contenido en DS:BX
MUL     WORD PTR [SI]   ; DXAX=AX*palabra contenida en DS:SI
IMUL    CL              ; AX=AL*CL
IMUL    DX              ; DXAX=AX*DX
IMUL    BYTE PTR [SI+2] ; AX=AL*byte contenido en DS:SI+2
IMUL    WORD PTR [BX+DI]; DXAX=AX*palabra contenida en DS:BX+DI
```

Instrucciones DIV (DIVide) e IDIV

Sintaxis:

```
DIV     reg
DIV     mem
IDIV    reg
IDIV    mem
```

- Realiza una división entre un número de 16 bits y otro de 8 bits, o entre un número de 32 bits y otro de 16 bits.
- En el primer caso, el dividendo ha de estar en el registro AX, y el divisor será el operando, con un tamaño de 8 bits. Como resultado, el cociente de 8 bits se almacena en AL, y el resto de 8 bits se almacena en AH.
- En el segundo caso, el dividendo ha de estar en la pareja de registros DX (palabra de mayor peso) y AX (palabra de menor peso), y el divisor ser el operando, con un tamaño de 16 bits. Como resultado, el cociente de 16 bits se almacena en AX, y el resto de 16 bits se almacena enDX.
- La diferencia entre DIV e IDIV es que la instrucción DIV realiza una división sin signo (sin complemento a dos), mientras que la instrucción IDIV realiza una división con signo (con complemento a dos).
- Ejemplos:

```
DIV     CL              ; AX / CL
DIV     BX              ; DXAX / BX
DIV     BYTE PTR [SI+2] ; AX / contenido en DS:SI+2
```

```
DIV     WORD PTR [DI+BX]; DXAX / contenido en DS:DI+BX
IDIV    DH              ; AX / DH
IDIV    CX              ; DXAX / CX
IDIV    BYTE PTR [DI+BX]; AX / contenido en DS:DI+BX
IDIV    WORD PTR [SI+8] ; DXAX / contenido en DS:si+8
```

INTRUCCIONES LOGICAS

Instruccion NOT (NOT)

Sintaxis:

```
NOT     reg
NOT     mem
```

- Realiza un NOT lógico, bit a bit, con el operando, es decir, que invierte el valor de cada uno de los bits del operando.

- Ejemplos:

```
NOT     DI
NOT     BYTE PTR [BX]
```

Instruccion AND (AND)

Sintaxis:

```
AND     reg,reg
AND     reg,mem
AND     reg,inmediato
AND     mem,reg
AND     mem,inmediato
```

- Realiza un AND lógico, bit a bit, entre el operando destino y el operando fuente, almacenando el resultado en el operando destino. Como la filosofía de esta operación es "valor 1 si los dos bits son 1", se puede utilizar como máscara, filtrando sólo aquellos bits (1) del primer operando que coincidan con los bits (1) del segundo operando.

- Ejemplos:

```
AND     AX,BX                   ; AX=AX AND BX
AND     SI,ES:[DI]              ; SI=SI AND ES:[DI]
AND     BX,0A34H                ; BX=BX AND 0A34H
AND     ES:[BX],CX              ; ES:[BX]=ES:[BX] AND CX
AND     BYTE PTR [SI+4],5       ; [SI+4]=[SI+4] AND 5
```

Instruccion OR (OR)

Sintaxis:

```
OR      reg,reg
OR      reg,mem
OR      reg,inmediato
```

```
OR      mem,reg
OR      mem,inmediato
```

- Realiza un OR lógico, bit a bit, entre el operando destino y el operando fuente, almacenando el resultado en el operando destino. La filosofía de esta operación es "valor 0 si los dos bits son 0". Su utilidad es poner algunos bits a 1, inalterando el resto.

- Ejemplos:

```
OR      AL,BL                   ; AL=AL OR BL
OR      DI,[BX]                 ; DI=DI OR [BX]
OR      CL,34                   ; CL=CL OR 34
OR      [DI],BX                 ; [DI]=[DI] OR BX
OR      BYTE PTR [DI],8         ; [DI]=[DI] OR 8
```

Instrucción XOR (eXclusive OR)

Sintaxis:

```
XOR     reg,reg
XOR     reg,mem
XOR     reg,inmediato
XOR     mem,reg
XOR     mem,inmediato
```

- Realiza un XOR lógico, bit a bit, entre el operando destino y el operando fuente, almacenando el resultado en el operando destino. La filosofía de esta operación es "valor 0 si los dos bits son iguales". Su utilidad es invertir el valor de algunos bits, inalterando el resto; o bien hacer un XOR consigo mismo para poner el operando a 0.

- Ejemplos:

```
XOR     CX,BX                   ; CX=CX XOR BX
XOR     DI,ES:[SI]              ; DI=DI XOR ES:[SI]
XOR     AX,4500H                ; AX=AX XOR 4500H
XOR     [BX],AX                 ; [BX]=[BX] XOR AX
XOR     BYTE PTR [SI+BX],50     ; [SI+BX]=[SI+BX] XOR 50
```

Instrucción LOOP

Sintaxis:

```
        MOV     CX,valor_inicial
etiqueta ...
        ...
        LOOP    etiqueta
```

- Gestiona un bucle, tomando el registro CX como contador del mismo. En sí, la instrucción LOOPdecrementa el valor de CX y, si no es igual a cero, salta a la etiqueta especificada.

- Ejemplo:

```
; ABC.ASM -> Imprime el abecedario
CODIGO  SEGMENT
        ASSUME CS:CODIGO
INICIO: MOV     DL,65           ; Carácter inicial (A)
        MOV     CX,26           ; Número de letras
BUCLE:  MOV     AH,2            ; Servicio para imprimir carácter
        INT     21H
        INC     DL              ; Incrementar carácter actual
        LOOP    BUCLE           ; Repetir mientras CX no sea cero
        MOV     AX,4C00H        ; Salir al DOS
        INT     21H
CODIGO  ENDS
        END     INICIO
```

INSTRUCCIONES DE SALTO INCONDICIONAL

Instrucción JMP (JuMP)

Sintaxis:

```
JMP     direc
JMP     etiqueta
```

- Realiza un salto de ejecución incondicional hacia la dirección o etiqueta especificada.

- Ejemplos:

```
JMP     100H            ; Salta a CX:100h
JMP     55AAH:100H      ; Salto lejano a otro segmento
JMP     WORD PTR [BX]   ; Salto a la dirección contenida en
                        ; la dirección de memoria especificada
                        ; por BX (salto indirecto)
JMP     REPITE          ; Salto a la etiqueta REPITE
```

Instrucciónes CALL Y RET (RETurn)

Sintaxis:

```
        CALL    direc   ; Saltar a direc
        ...             ; Dirección de retorno
        ...
direc   ...             ; Dirección de salto
        ...
        RET
```

- Realiza un salto incondicional hacia la dirección, etiqueta o procedimiento especificado. A diferencia de la instrucción JMP, la instrucción CALL realiza un salto a una subrutina con retorno. El salto puede ser cercano o lejano.
- En el primer caso, la dirección a la que salta corresponde al offset dentro del segmento de código actual, por lo que, antes de realizar el salto, CALL guarda en la pila el contenido de IP, el cual apunta a la instrucción inmediatamente después de la instrucción CALL.
- En el segundo caso, la dirección a la que salta corresponde a un offset dentro de otro segmento de código, por lo que, antes de realizar el salto, CALL guarda

en la pila el contenido de CS e IP(CS:IP), el cual apunta a la instrucción inmediatamente después de la instrucción CALL.

- Una vez realizado el salto, se ejecutarán las instrucciones que allí hubieran hasta encontrar la sentencia RET, la cual extrae de la pila la dirección de retorno almacenada con CALL.

SALTOS CONDICIONALES

- Hasta ahora habíamos realizado un salto incondicional a través de las instrucciones CALL y JMP. Asimismo, en las prácticas, habíamos comprobado algún que otro salto condicional (JE, JNE óJZ), pero no nos habíamos adentrado en su funcionamiento con el fin de no aturdir o confundir. Y es que, a diferencia de otros lenguajes de programación de más alto nivel, el lenguaje ensamblador no posee construcciones o estructuras de salto condicional, tales como IF condición THEN GOTO xxx. En realidad, el lenguaje ensamblador posee un conjunto de instrucciones de salto condicionales, las cuales dirigen la ejecución a un lugar determinado del programa, dependiendo del valor que posean los flags. Existen en total 17 instrucciones de salto condicional, dependiendo del valor de algún flag determinado, producto de la última operación realizada. La sintaxis general y común de estas instrucciones es la siguiente:

```
Jx      etiqueta_corta
```

- Todas las instrucciones de salto condicional comienzan por la letra J (Jump o salto). El lugar donde salta se refiere a una etiqueta corta, es decir, que est, a -128 y +127 bytes desde la instrucción de salto.
- A continuación se desglosan las distintas instrucciones de salto condicional:

INSTRUCCION	SIGNIFICADO	SALTAR SI
JA	Si está por encima	CF=0 y ZF=0
JAE	Si está por encima o igual	CF=0
JB	Si está por debajo	CF=1
JBE	Si está por debajo o igual	CF=1 ó ZF=1
JC	Si hay acarreo	CF=1
JCXZ	Si CX=0	CX=0
JE	Si es igual	ZF=1
JG (*)	Si es mayor	ZF=0 y SF=OF
JGE (*)	Si es mayor o igual	SF=OF
JL (*)	Si es menor	SF no = OF
JLE (*)	Si es menor o igual	ZF=1 ó SF<>OF
JNA	Si no está por encima	CF=1 ó ZF=1
JNAE	Si no está por encima ni igual	CF=1
JNB	Si no está por debajo	CF=0
JNBE	Si no está por debajo ni igual	CF=0 y ZF=0
JNC	Si no hay acarreo	CF=0
JNE	Si no es igual	ZF=0
JNG (*)	Si no es mayor	ZF=1 ó SF<>OF
JNGE (*)	Si no es mayor ni igual	SF no = OF
JNL (*)	Si no es menor	SF=OF

```
JNLE (*)        Si no es menor ni igual         ZF=0 y SF=OF
JNO  (*)        Si no hay desbordamiento        OF=0
JNP             Si no hay paridad (impar)       PF=0
JNS  (*)        Si no hay signo                 SF=0
JNZ             Si no es cero                   ZF=0
JO   (*)        Si hay desbordamiento           OF=1
JP              Si hay paridad (par)            PF=1
JPE             Si hay paridad par              PF=1
JPO             Si hay paridad impar            PF=0
JS   (*)        Si hay signo                    SF=1
JZ              Si es cero                      ZF=1
-----------------------------------------------------------------
(*) Se usan para aritmética con valores con signo
```

EJEMPLOS:

```
        ADD     BL,CL           ; BL=BL+CL
        JC      GRANDE          ; Saltar a GRANDE si hay acarreo

        SUB     BL,CL           ; BL=BL-CL
        JNZ     DISTINTO        ; Saltar a DISTINTO si BL no es cero
```

COMPARACIONES

- Las operaciones aritméticas, usadas para realizar saltos condicionales, pecan de modificar los valores, salvo si se guardan en la pila. Pero esta técnica, además de engorrosa, podría despistar al programador y causarle no pocos problemas. Otra forma más sencilla de realizar saltos condicionales es a través de comparaciones, las cuales cotejan el valor de un registro con otro valor. Para realizar estas comparaciones se utiliza la instrucción CMP (CoMPare), cuya sintaxis es la siguiente:

```
CMP     operando_destino,operando_fuente
```

- Básicamente, comparar un valor con otro es como restar uno de otro. El resultado no afecta a los operandos ni se guarda en ninguna parte: únicamente cambia el valor de los flags. El resultado será el siguiente:

```
CONDICION               OF   SF   ZF   CF
OPERANDOS SIN SIGNO
  fuente < destino      NI   NI   0    0     NI  : No Importa
  fuente = destino      NI   NI   1    0     0/1 : 1 ó 0 dependiendo
  fuente > destino      NI   NI   0    1           del valor de los
OPERANDOS CON SIGNO                              operandos
  fuente < destino      0/1  0    0    NI
  fuente = destino      0    0    1    NI
  fuente > destino      0/1  1    0    NI
```

- Cuando se utiliza la instrucción CMP para evaluar un salto condicional (se usa justamente antes de la instrucción de salto condicional), hay que tener en cuenta la siguiente tabla:

```
SALTAR SI         SIN SIGNO CON SIGNO
 destino>fuente       JA        JG
 destino=fuente       JE        JE
 destino<>fuente      JNE       JNE
```

destino<fuente	JB	JL
destino<=fuente	JBE	JLE
destino>=fuente	JAE	JGE

TIPOS DE SALTO

- Como se ha podido comprobar, las instrucciones de salto condicional realizan únicamente saltos a etiquetas cortas, es decir, a etiquetas que se encuentran entre -128 bytes y +127 bytes desde la instrucción de salto. Esto representa un gran inconveniente en nuestros programas, sobre todo si el código a ejecutar por otra sentencia de salto satura este límite. Para solventar el problema se conjunta el uso de sentencias de salto condicional con las sentencias de salto incondicional. Veamos un ejemplo:

```
          CMP     AL,100          ; Comparar AL con 100
          JA      MAYOR           ; Si AL>100 saltar a MAYOR
          JE      IGUAL           ; Si AL=100 saltar a IGUAL
          JB      MENOR           ; Si AL<100 saltar a MENOR
MAYOR:    JMP     MAYOR100        ; Ir a etiqueta cercana MAYOR100
IGUAL:    JMP     IGUAL100        ; Ir a etiqueta cercana IGUAL100
MENOR:    JMP     MENOR100        ; Ir a etiqueta cercana MENOR100
```

- Es conveniente conocer el tipo de saltos que se pueden realizar. De momento conocemos el salto a etiquetas cortas y a etiquetas cercanas. Básicamente, el funcionamiento del microprocesador es el siguiente:
    - Recoger siguiente instrucción (*CS:IP*).
    - Incrementar *IP* tantos bytes como ocupe la instrucción (apuntar a la la siguiente instrucción).
    - Decodificar la instrucción.
    - Ejecutar la instrucción.
- Los saltos de ejecución consisten en modificar el valor de IP (y de CS si es necesario). Básicamente hay dos tipos de salto: directos e indirectos.
- Los saltos directos especifican la dirección donde debe de saltar, mientras que los saltos indirectos especifican una dirección de memoria donde se encuentra la dirección a donde hay que saltar.
- En los saltos directos hay varios formatos de salto:
    - **Salto corto (short jump)**: Indica un byte en complemento a dos (-128 a +127) que es el número de bytes a restar o sumar a *IP*. Este valor lo calcula el ensamblador a través de las etiquetas.
    - **Salto cercano (near jump)**: Indica dos bytes (palabra) en complemento a dos (-32768 a +32767), que es el número de bytes a restar o sumar a *IP*. Este valor es calculado por el ensamblador a través de las etiquetas. Tanto el salto corto, como el salto cercano se refiere al mismo segmento de código.

- **Salto lejano (far jump)**: Indica dos palabras, que contienen el offset y el segmento donde hay que saltar. Este formato copia en *IP* y *CS* respectivamente los valores de la instrucción.

- En los saltos indirectos hay que especificar un direccionamiento a memoria, que es donde se almacena la dirección de salto. Dentro de esta dirección de memoria se pueden almacenar dos tipos de salto:
  - **Salto cercano (near jump)**: Lee de la dirección de memoria una palabra y la copia a *IP*.
  - **Salto lejano (far jump)**: Lee de la dirección de memoria una palabra y la copia a *IP*. Después lee la siguiente palabra y la copia a *CS*.
- En un salto indirecto ser necesario indicar un 'typecast', que indica a la instrucción el tamaño del dato a leer. Conocemos los 'typecast' BYTE PTR y WORD PTR. En el caso de utilizar un salto indirecto lejano, hay que especificar el 'typecast' DWORD PTR (Double WORD PoinTeR, o puntero a doble palabra).

- EJEMPLOS:

```
JMP CERCANA       ; Salto directo a etiqueta cercana (o corta)
JMP FAR PTR LEJANA; Salto directo a etiqueta en otro segmento
JMP 55AAHH:0100H  ; Salto directo a segmento 55AAH y Offset 0100H
JMP [BX]          ; Salto indirecto al mismo segmento (lee IP)
JMP WORD PTR [BX] ; Salto indirecto al mismo segmento (lee IP)
JMP FAR PTR [BX]  ; Salto indirecto a otro segmento (lee IP y CS)
JMP DWORD PTR [BX]; Salto indirecto a otro segmento (lee IP y CS)
SALTO   DW    CERCANO ; Desplazamiento de CERCANO (offset)
        ...
JMP     SALTO ; Salto indirecto al mismo segmento a CERCANO

SALTO   DD    LEJANO  ; Desplazamiento de LEJANO (segmento y offset)
        ...
JMP     SALTO   ; Salto indirecto a otro segmento a LEJANO

JMP SHORT CORTA     ; Salto corto a etiqueta corta
```