

---

## **Conjunto de Instrucciones del 8088**



# XCHG

Intercambia el contenido de cualquier registro con el contenido de cualquier otro registro o localidad de memoria, no incluyendo los registros de segmento o intercambios de memoria a memoria.

## Ejemplo:

AX = 1F60      BX = 472D      CX = 310B      DX = 8472

### **XCHG AX,BX**

AX = 472D      BX = 1F60      CX = 310B      DX = 8472

### **XCHG CL,DH**

AX = 472D      BX = 1F60      CX = 3184      DX = 0B72

# IN y OUT

---

## **IN:**

Lee un dato de un dispositivo de E/S y lo almacena en el registro AL o AX.

## **OUT:**

Transfiere un dato del registro AL o AX a un puerto de E/S.



# IN y OUT

---

Existen dos formas para el direccionamiento de puertos con las instrucciones IN y OUT, las cuales son: **puerto fijo** y **puerto variable**.

- **Puerto fijo:** Permite la transferencia de datos entre AL o AX y un puerto de E/S con dirección de 8 bits.

*Ejemplos:* `IN AX, 85`      `IN AL, 3F`      `OUT 42, AL`      `OUT 2A, AX`

- **Puerto variable:** Permite la transferencia de datos entre AL o AX y un puerto de E/S con dirección de 16 bits, y el numero de puerto se almacena en DX.

*Ejemplos:* `IN AX, DX`      `IN AL, DX`      `OUT DX, AL`      `OUT DX, AX`

---



# IN y OUT

---

**Tabla 2.** Instrucciones IN y OUT.

Código de operación	Función
IN AL, pp	Un dato de 8 bits se transfiere del puerto pp a AL
IN AX,pp	Un dato de 16 bits se transfiere del puerto pp a AX
IN AL,DX	Un dato de 8 bits se transfiere del puerto DX a AL
IN AX,DX	Un dato de 16 bits se transfiere del puerto DX a AX
OUT pp,AL	Un dato de 8 bits se transfiere de AL al puerto DX
OUT pp,AX	Un dato de 16 bits se transfiere de AX al puerto pp
OUT DX,AL	Un dato de 16 bits se transfiere de AL al puerto DX
OUT DX,AX	Un dato de 16 bits se transfiere de AX al puerto DX

Nota: pp= un puerto de E/S de con dirección de 8 bits y DX = contiene la dirección de un puerto de E/S con dirección de 16 bits.



# PUSH y POP

Se utilizan para empujar o sacar datos del **Segmento de Pila**.

El registro **SP** (Apuntador de Pila) almacena la dirección del segmento de Pila de donde se van a ingresar o sacar datos.

En el 8088 se dice que la Pila crece a direcciones mas pequeñas, ya que cuando se **ingresan** datos se **decrementa** el apuntador de pila, y cuando se **extraen** datos se **incrementa** el apuntador. Es decir, conforme se van ingresando datos SP apunta a direcciones cada vez mas pequeñas.



# PUSH y POP

---

## PUSH:

Ingresa **2 bytes** de información a la pila.

Cuando un dato es metido a la pila, el byte **mas significativo** es almacenado en la localidad direccionada por **SP-1**, y el byte **menos significativo** es almacenado en la localidad **SP-2**.

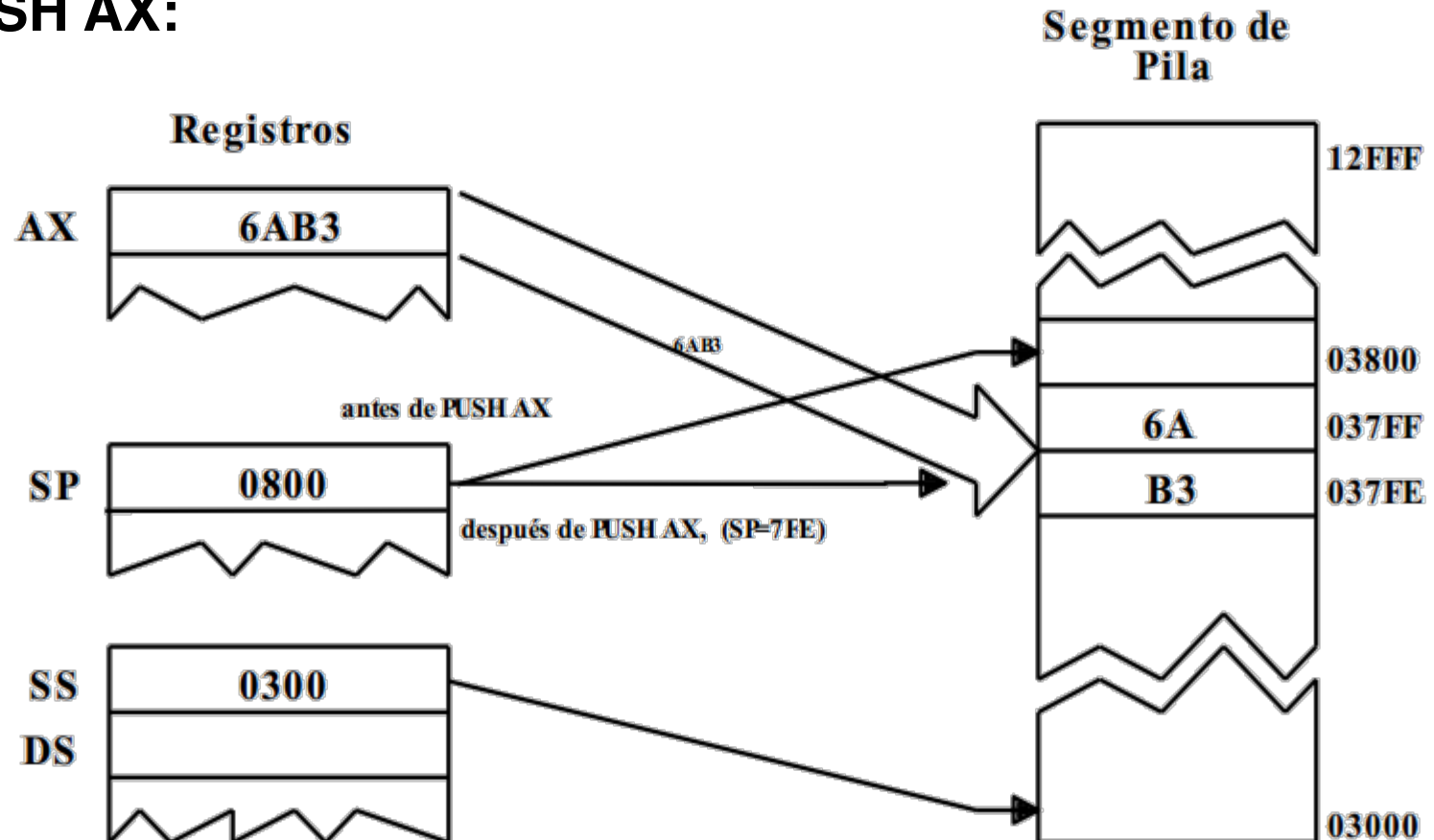
Después de que el dato se ha empujado a la pila, el registro **SP** es **decrementado en 2**.



# PUSH y POP

Ejemplo:

**PUSH AX:**





# PUSH y POP

---

## POP:

Realiza lo inverso a la instrucción PUSH. POP **remueve 2 bytes** de la pila.

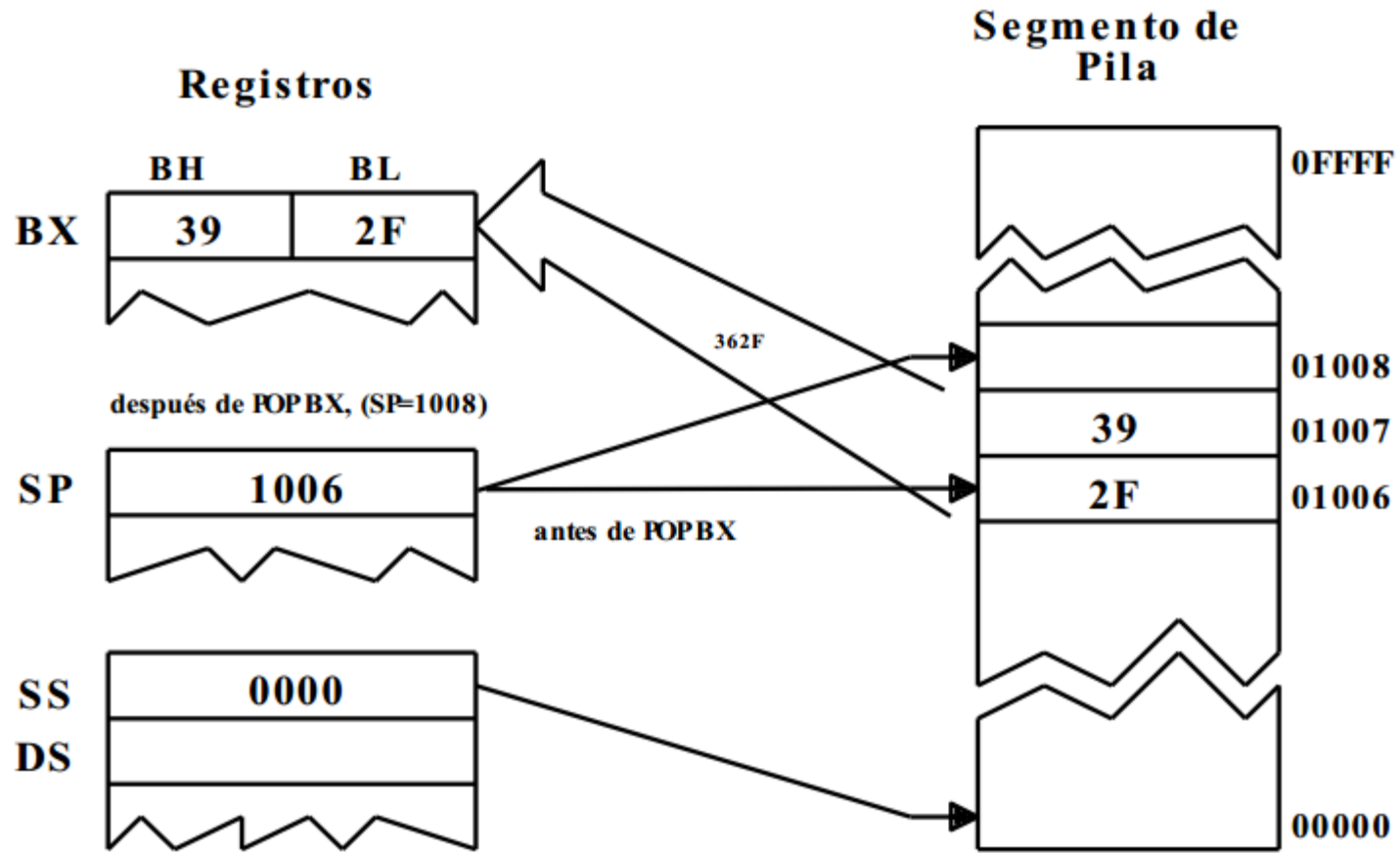
Suponga que la instrucción POP BX es ejecutada. El byte apuntado por **SP** es removido de la pila y almacenado en el registro BL. Después se remueve el byte apuntado por **SP+1** y se coloca en el registro BH.

Después de que ambos datos fueron extraídos de la pila, **SP** es **incrementado en 2**.



# PUSH y POP

## POP BX:



# PUSH y POP

---

Las instrucciones PUSH y POP pueden operar sobre registros, memoria y registros de segmento.

## Importante:

Sí se permite empujar a la pila el registro de segmento de Código (CS), pero no se permite extraer un valor de la pila y almacenarlo en CS, ya que no se permite la modificación de este registro.

**PUSH CS** ✓ (permitido)

**POP CS ó MOV CS,valor** , etc ✗



# PUSHF y POPF

---

## **PUSHF:**

Copia el contenido del registro de banderas a la pila.

## **POPF:**

Realiza la operación inversa de PUSHF, remueve de la pila un dato de 16 bits que es cargado como el contenido del registro de banderas.

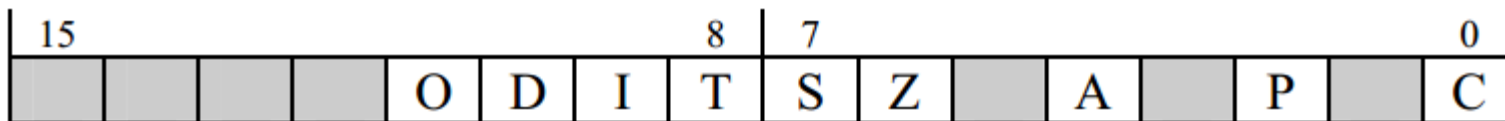


# LAHF

Copia el byte menos significativo del registro de banderas en el registro AH.

Después de la ejecución de esta instrucción, los bits 7,6,4,2 y 0 de AH son iguales a las banderas S, Z, A, P y C respectivamente.

**Registro de banderas:**



# SAHF

---

Copia el contenido de AH en el byte menos significativo del registro de banderas.

Después de la ejecución de esta instrucción, las banderas S, Z, A, P y C son iguales a los bits 7, 6, 4, 2 y 0 de AH, respectivamente.



# Suma (ADD)

---

Siempre que una instrucción aritmética o lógica se ejecuta, cambia el contenido del registro de banderas. Las banderas denotan el resultado de la operación aritmética.

Cualquier instrucción ADD modifica el contenido de las banderas de signo, cero, acarreo, acarreo auxiliar, paridad y sobreflujo.

## **Importante:**

No se permite la suma con registros de segmento



# Suma (ADD)

---

**Tabla 6.** Instrucciones de Suma.

Instrucciones	Comentario
ADD AL,BL	$AL = AL + BL$
ADD CX,DI	$CX = CX + DI$
ADD CL,44H	$CL = CL + 44H$
ADD BX,35AFH	$BX = BX + 35AFH$
ADD [BX],AL	$DS:[BX] = DS:[BX] + AL$
ADD CL,[BP]	$CL = CL + SS:[BP]$
ADD BX,[SI+2]	$BX = BX + DS:[SI+2]$
ADD CL,TEMP	$CL = CL + [offset\ TEMP]$
ADD BX,TEMP[DI]	$BX = BX + [offset\ TEMP + DI]$
ADD [BX+DI],DL	$DS:[BX+DI] = DS:[BX+DI] + DL$





# Suma con acarreo (ADC)

Realiza la suma de dos operandos más el bit de acarreo.

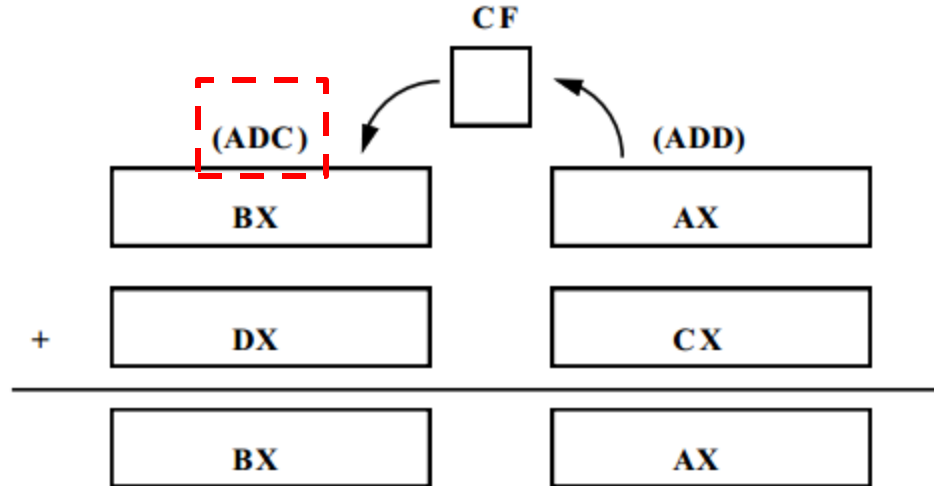
**Tabla 8.** Instrucciones Suma con acarreo.

Instrucciones	Comentarios
ADC AL,AH	$AL = AL + AH + CF$
ADC CX,BX	$CX = CX + BX + CF$
ADC [BX],DH	$DS:[BX] = DS:[BX] + DH + CF$
ADC BX,[BP+2]	$BX = BX + SS:[BP+2] + CF$

# Suma con acarreo (ADC)

## Ejemplo:

Suponga que un número de 32 bit contenido en los registros BX y AX se suma a un número de 32 bit contenido en los registros DX y CX. Para que el resultado de la suma sea correcto, se tiene que considerar el acarreo que pudiera existir al sumar AX y CX, por lo tanto se tiene que hacer uso de la instrucción ADC.



**Figura 4.** Suma con acarreo mostrando como la bandera de acarreo (CF) une dos sumas de 16 bits para formar suma de 32 bits.

# Incremento (INC)

---

Incrementa en 1 el contenido de un registro o localidad de memoria. Al igual que la suma, no se permite el incremento de los registros de segmento.

Esta instrucción modifica el contenido del registro de banderas, a excepción de que no modifica el valor del bit de acarreo.

**Ejemplo:**

**INC AX**



# Incremento (INC)

**Tabla 7.** Instrucciones de Incremento.

Instrucciones	Comentarios
INC BL	$BL = BL + 1$
INC SP	$SP = SP + 1$
INC BYTE PTR[BX]	$DS:[BX] = DS:[BX] + 1$
INC WORD PTR[SI]	$DS:[SI] = DS:[SI] + 1$
INC DATA1	$DS:DATA1 = DS:DATA1 + 1$

Cuando se incrementa un dato que está almacenado en memoria, se tiene que especificar si el dato a incrementar es de 8 o de 16 bits.

**BYTE PTR:** Indica que el incremento va a ser de un dato de 8 bits.

**WORD PTR:** Indica que el incremento va a ser de un dato de 16 bits.

# Resta (SUB)

## Ejemplos:

**Tabla 9.** Instrucciones de Resta.

Instrucciones	Comentarios
SUB CL,BL	CL = CL - BL
SUB AX,SP	AX = AX - SP
SUB DH,6FH	DH = DH - 6FH
SUB AX,0CCCCH	AX=AX-DS:0CCCCH
SUB [DI],CH	DS:[DI]=DS:[DI]-CH
SUB CH,[BP]	CH=CH-SS:[BP]
SUB AH,TEMP	AH=AH-DS:TEMP
SUB DI,TEMP[BX]	DI=DI-TEMP[BX]

# Resta con préstamo (SBB)

Una instrucción resta con préstamo (SBB) funciona como una resta regular, excepto que la bandera de acarreo (CF), la cual actúa como préstamo también se subtrae de la diferencia.

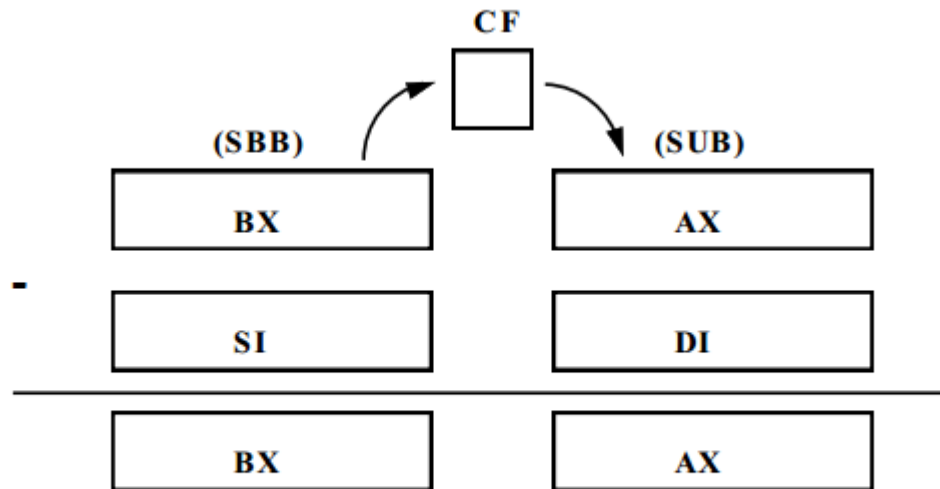
**Tabla 11.** Instrucciones de Resta con Acarreo.

Instrucciones	Comentarios
SBB AH,AL	$AH = AH - AL - CF$
SBB AX,BX	$AX = AX - BX - CF$
SBB CL,3	$CL = CL - 3 - CF$
SBB BYTE PTR[DI],3	$DS:[DI] = DS:[DI] - 3 - CF$
SBB [DI],AL	$DS:[DI] = DS:[DI] - AL - CF$
SBB DI,[BP+2]	$DI = DI - SS:[BP+2] - CF$

# Resta con préstamo (SBB)

## Ejemplo:

Un número de 32 bit almacenado en BX y AX se sustrae de un número de 32 bit almacenado en SI y DI, la bandera de acarreo propaga el préstamo entre las dos sustracciones de 16 bit requeridas para realizar la operación.



**Figura 5.** Resta con préstamo mostrando como la bandera de acarreo (CF) propaga el préstamo.

# Decremento (DEC)

---

Decrementa en 1 el contenido de un registro o localidad de memoria.

Esta instrucción modifica el contenido del registro de banderas, a excepción de que no modifica el valor del bit de acarreo.

**Ejemplo:**

**DEC AX**





# Decremento (DEC)

---

**Tabla 10.** Instrucciones de Decremento.

Instrucciones	Comentarios
DEC BH	BH = BH - 1
DEC SP	SP = SP - 1
DEC BYTE PTR[DI]	DS:[DI]=DS:[DI]-1
DEC WORD PTR[BP]	SS:[BP]=SS:[BP]-1
DEC NUMB	DS:NUMB=DS:NUMB-1

Al igual que con la instrucción INC, cuando se va a decrementar un valor contenido en memoria se tiene que especificar por medio de las directivas BYTE PTR o WORD PTR si se esta trabajando con un dato de 8 o de 16 bits.



# Instrucciones Lógicas y de Manipulación de Bits

Incluyen operaciones lógicas, corrimientos y rotaciones.

## **Instrucciones:**

- AND
- OR
- XOR
- TEST
- NOT
- NEG
- SHL/SAL
- SHR
- SAR
- ROL
- RCL
- ROR
- RCR



# Instrucciones Lógicas y de Manipulación de Bits

---

No se permite el uso de estas instrucciones en los registros de segmento.

Todas las instrucciones lógicas modifican el registro de banderas.

Las operaciones lógicas siempre ponen el bit de acarreo y el de sobreflujo en ceros, y el resto de los bits de banderas cambian para reflejar la condición del resultado.



# AND

A * B = X		
0	0	0
0	1	0
1	0	0
1	1	1

**Figura 6.** Tabla de verdad para la operación AND (multiplicación lógica).

**Tabla 18.** Instrucciones AND

Instrucciones	Comentarios
AND AL,BL	AL = AL AND BL
AND CX,DX	CX = CX AND DX
AND CL,33H	CL = CL AND 33H
AND DI,4FFFFH	DI = DI AND 4FFFFH
AND AX,[DI]	AX = AX AND DS:[DI]
AND ARRAY[SI],AL	ARRAY[SI]=ARRAY[SI] AND AL

# OR

A + B = X		
0	0	0
0	1	1
1	0	1
1	1	1

**Figura 8.** Tabla de verdad para la operación OR (suma lógica).

**Tabla 19.** Instrucciones OR

Instrucciones	Comentarios
OR AH,BL	AH = AH OR BL
OR SI,DX	SI = SI OR DX
OR DH,0A3H	DH = DH OR 0A3H
OR SP,990DH	SP = SP OR 990DH
OR DX,[BX]	DX = DX OR DS:[BX]
OR DATES[DI+2],AL	DATES[DI+2]=DATES[DI+2] OR AL

# XOR

$$A \oplus B = X$$

0	0	0
0	1	1
1	0	1
1	1	0

**Figura 10.** Tabla de verdad para la operación XOR.

**Tabla 20.** Instrucciones OR Exclusiva

Instrucciones	Comentarios
XOR CH,DH	CH = CH XOR DH
XOR SI,BX	SI = SI XOR BX
XOR CH,0EEH	CH = CH XOR 0EEH
XOR DI,0DDH	DI = DI XOR 0DDH
XOR DX,[SI]	DX = DX XOR DS:[DI]
XOR DATE[DI+2],AL	DATE[DI+2]=DATE[DI+2] XOR AL

# Enmascaramiento

---

Consiste en realizar una operación lógica entre un dato y un cierto patrón.

Esto con el fin de conocer el estado de ciertos bits, o modificarlos de acuerdo a un patrón requerido.



# Enmascaramiento (**AND**)

**Poner los bits 3 y 7 en 0:**

<i>Dato:</i>		xxxx	xxxx	(cualquier valor)
<i>Mascara:</i>	<b>AND</b>	0111	0111	(77h)
<i>Resultado:</i>		0xxx	0xxx	

Es importante recordar que:

0 *AND* cualquier-valor, es 0 lógico siempre

1 *AND* cualquier-valor, es cualquier-valor



0	0	0
0	1	0
1	0	0
1	1	1



# Enmascaramiento (**AND**)

---

**Conocer el estado de los bits 0 y 5:**

<i>Dato:</i>		xxxx	xxxx	(cualquier valor)
<i>Mascara:</i>	<b>AND</b>	0010	0001	(21h)
<i>Resultado:</i>		00x0	000x	

## Ejemplo en lenguaje C:

```
if( dato & 0x21 ) {  
    realizar-ciertas-acciones;  
}
```

Representación de 21h en lenguaje C

Operador **AND** lógico  
en lenguaje C



# Enmascaramiento (**OR**)


**Poner en 1 lógico el nibble mas significativo:**

<i>Dato:</i>		xxxx	xxxx	(cualquier valor)
<i>Mascara:</i>	<b>OR</b>	1111	0000	(F0h)
<i>Resultado:</i>		1111	xxxx	

Es importante recordar que:

1 *OR* cualquier-valor, es 1 lógico siempre

0 *OR* cualquier-valor, es cualquier-valor



0	0	0
0	1	1
1	0	1
1	1	1

# Enmascaramiento (**XOR**)

Esta operación lógica se suele utilizar para invertir bits.

**Invertir (complementar) el nibble menos significativo:**

<i>Dato:</i>		xxxx	xxxx	(cualquier valor)
<i>Mascara:</i>	<b>XOR</b>	0000	1111	(0Fh)
<i>Resultado:</i>		xxxx	$\bar{x} \bar{x} \bar{x} \bar{x}$	

Es importante recordar que:

1 XOR cualquier-valor, es el valor invertido

0 XOR cualquier-valor, es cualquier-valor

0	0	0
0	1	1
1	0	1
1	1	0

# NOT y NEG

---

La instrucción **NOT** invierte todos los bit de un byte o palabra. (Realiza el **complemento a uno**).

La instrucción **NEG** realiza el **complemento a dos** a un número, lo cual significa que el signo aritmético de un número cambia de positivo a negativo o de negativo a positivo.

La función **NOT** se considera una operación lógica y la función **NEG** se considera una operación aritmética.

---



# NOT y NEG

---

**Tabla 22.** Instrucciones NOT y NEG.

Instrucciones	Comentarios
NOT CH	CH=CMPLT1(CH)
NEG CH	CH=CMPLT2(CH)
NEG AX	AX=CMPLT2(AX)
NOT TEMP	TEMP=CMPLT1(TEMP)
NOT BYTE PTR[BX]	[BX]=CMPLT1([BX])



---

# **Corrimientos y Rotaciones**



# Corrimientos

---

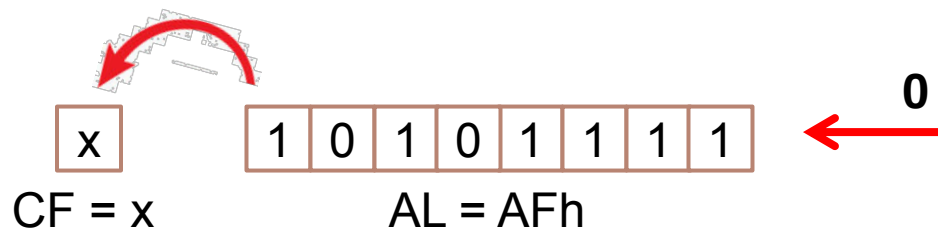
Las instrucciones de corrimiento posicionan o mueven números a la izquierda o a la derecha dentro de un registro o localidad de memoria, excepto los registros de segmento.



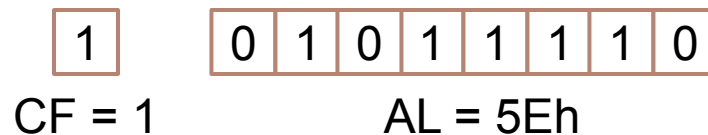
# Corrimiento a la izquierda

## SHL: Shift Logical Left

### Ejemplo:



### SHL AL,1



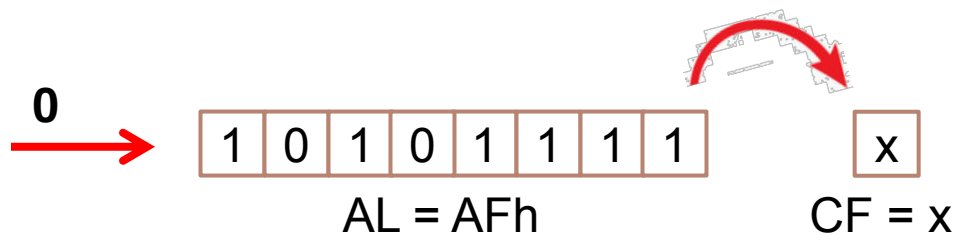
Carry Flag (CF)



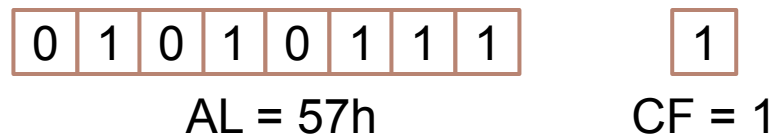
# Corrimiento a la derecha

## SHR: Shift Logical Right

### Ejemplo:



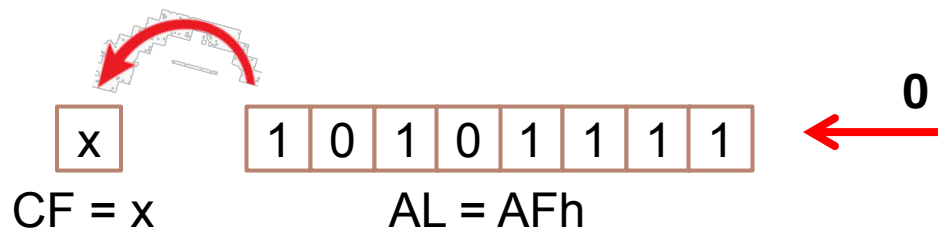
### SHR AL,1



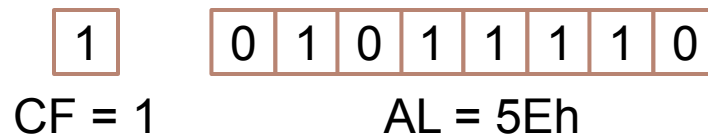
# Corrimiento a la izquierda **aritmético**

SAL: Shift Arithmetic Left

**Ejemplo:**



**SAL AL,1**

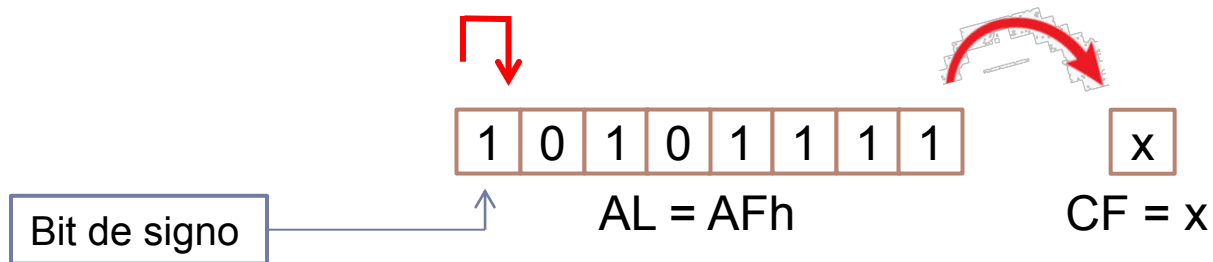


**Carry Flag (CF)**

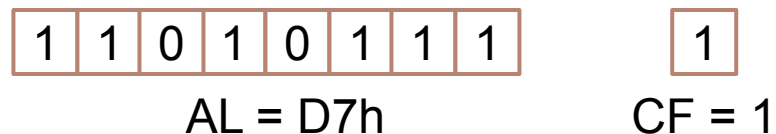
# Corrimiento a la derecha **aritmético**

SAR: Shift Arithmetic Right

**Ejemplo:**



**SAR AL,1**



# Corrimientos

Cuando se desea realizar un corrimiento de mas de un bit se usa el registro CL para indicar la cuenta de corrimientos. El registro CL no se modifica al ejecutarse la instrucción de corrimiento.

## Ejemplos:

**Hace un corrimiento de dos bits a la izquierda en el registro AX.**

```
MOV CL,2  
SHL AX,CL
```

**Hace un corrimiento aritmético de siete bits a la derecha en el registro BL.**

```
MOV CL,7  
SAR BL,CL
```



# Corrimientos

---

**Tabla 23.** Instrucciones de Corrimiento.

Instrucciones	Comentarios
SHL AX,1	Corrimiento de AX 1 lugar a la izquierda
SHR BX,1	Corrimiento de BX 1 lugares a la derecha
SAL DATA1,CL	Corrimiento aritmético de DATA CL lugares a la izquierda
SAR SI,CL	Corrimiento aritmético de SI CL lugares a la derecha



# Corrimientos

Las operaciones de corrimientos también se pueden utilizar como operaciones aritméticas simples tales como:

**Corrimiento a la izquierda:** multiplicación por potencias de  $2^n$

**Corrimiento a la derecha:** división por potencias de  $2^n$

**Ejemplos:**

**Multiplicación por potencias de  $2^n$  :**

$$2Dh * 2 = 0x5A$$

$$2Dh \ll 1 = 0x5A$$



Operador **Corrimiento  
lógico a la izquierda**  
en lenguaje C

$$36h * 4 = 0xD8$$

$$36h \ll 2 = 0xD8$$

# Corrimientos

---

## División por potencias de $2^n$ :

$2Dh / 2 = 0x16$

$2Dh \gg 1 = 0x16$



$36h / 8 = 0x06$

$36h \gg 3 = 0x06$

Operador **Corrimiento  
lógico a la derecha**  
en lenguaje C



# Corrimientos

---

## Ejemplo 26

;Multiplica AX por 10 (1010)

SHL AX,1	;2 veces AX
MOV BX,AX	
SHL AX,1	;4 veces AX
SHL AX,1	;8 veces AX
ADD AX,BX	;10 veces AX

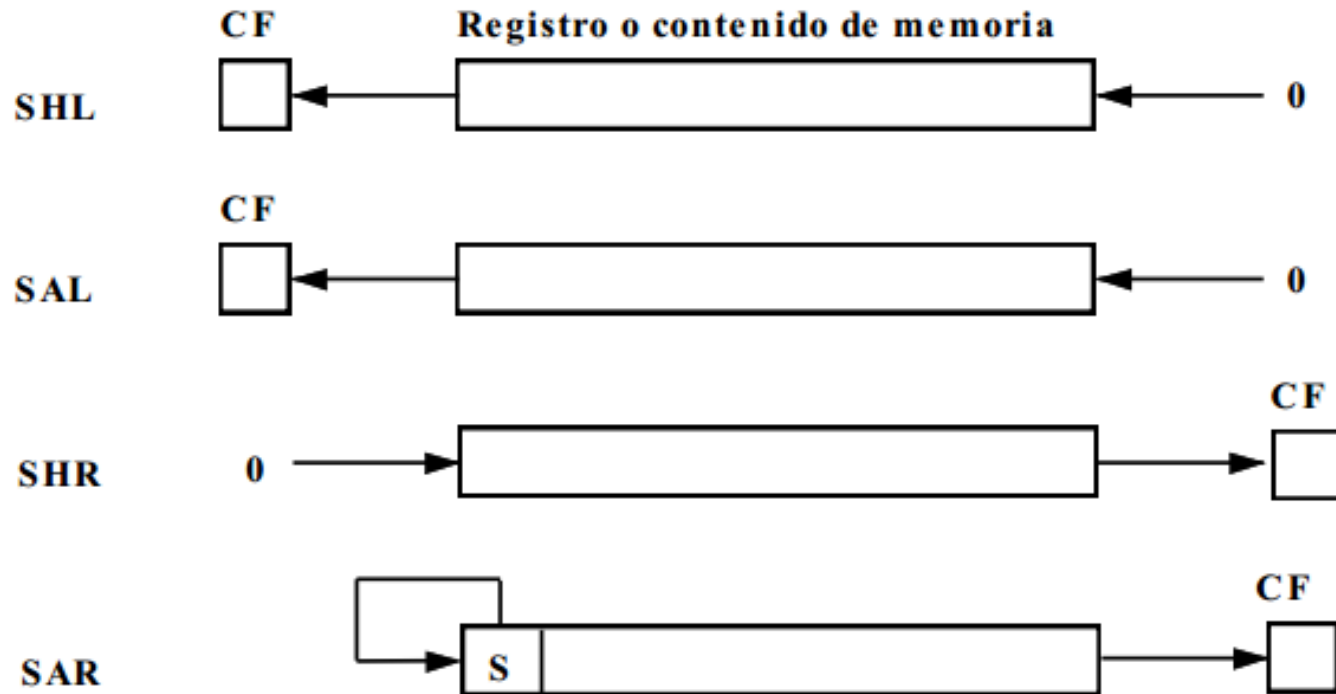
;Multiplica AX por 18 (10010)

SHL AX,1	;2 veces AX
MOV BX,AX	
SHL AX,1	;4 veces AX
SHL AX,1	;8 veces AX
SHL AX,1	;16 veces AX
ADD BX,AX	;18 veces AX





# Corrimientos



# Rotaciones

---

Las instrucciones de rotación posicionan datos binarios mediante la rotación de la información en un registro o localidad de memoria ya sea de un extremo u otro o a través de la bandera de acarreo.

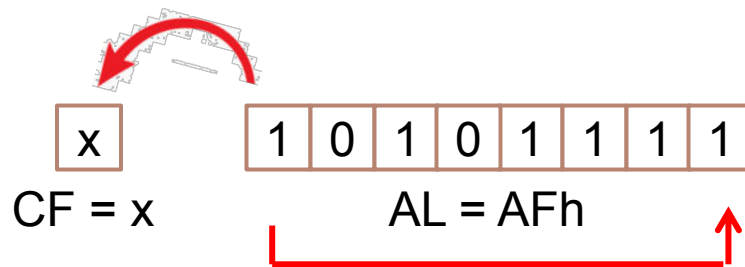
Al igual que con los Corrimientos, si se va a realizar una rotación de mas de un bit se tiene que usar al registro CL para indicar la cuenta de rotaciones.



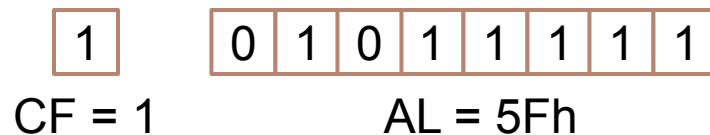
# Rotación a la izquierda

## ROL: Rotate Left

### Ejemplo:



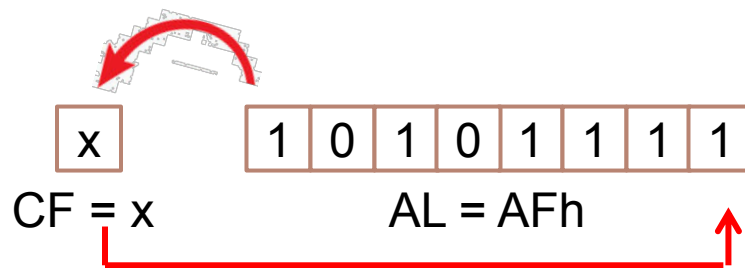
### ROL AL,1



# Rotación a la izquierda **con acarreo**

RCL: Rotate Left through Carry

**Ejemplo:**



**RCL AL,1**

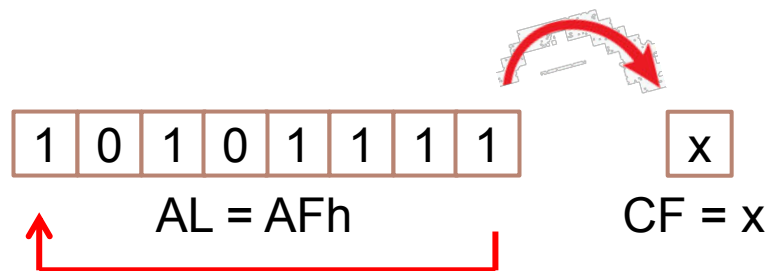
1      0 1 0 1 1 1 1 x

CF = 1    Si la bandera de acarreo previamente estaba en 1: **AL = 5Fh**  
Si estaba en 0: **AL = 5Eh**

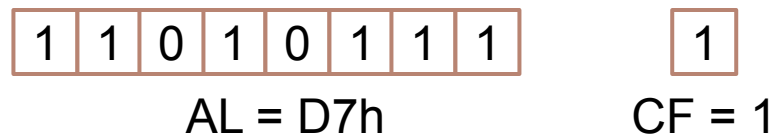
# Rotación a la derecha

ROR: Rotate to Right

**Ejemplo:**



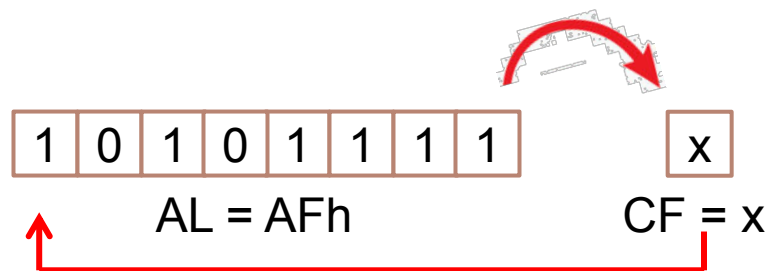
**ROR AL,1**



# Rotación a la derecha **con acarreo**

RCR: Rotate Right through Carry

**Ejemplo:**



**RCR AL,1**



Si la bandera de acarreo previamente estaba en 1: **AL = D7h**

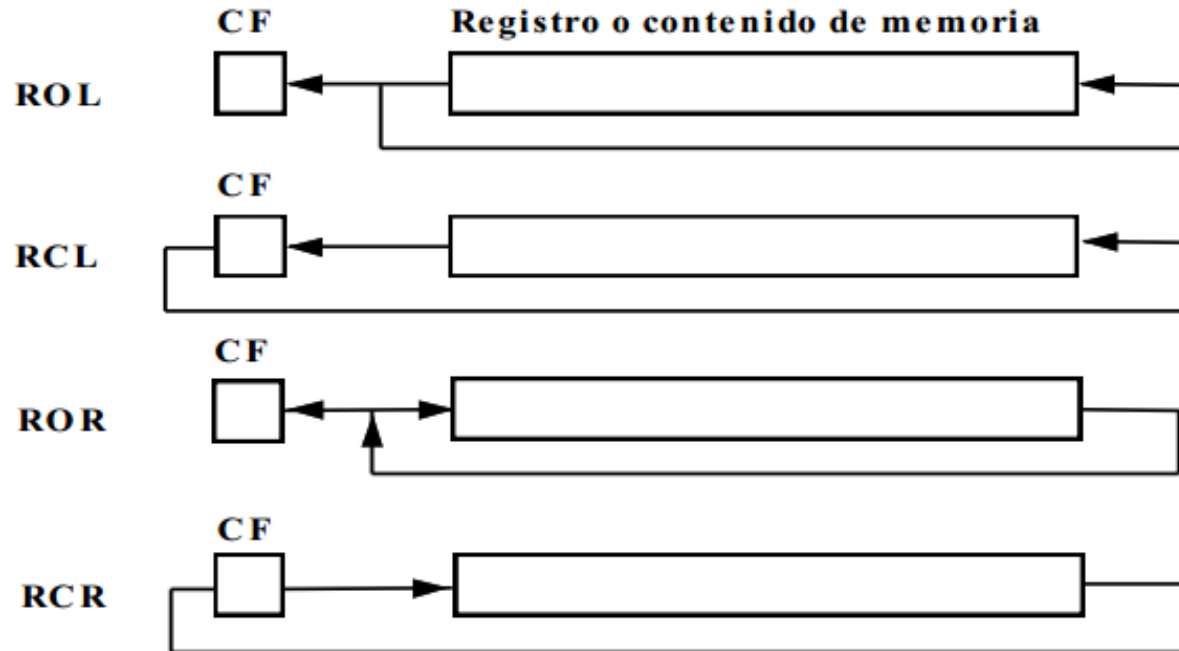
Si estaba en 0: **AL = 57h**

# Rotaciones

**Tabla 24.** Instrucciones de Rotación.

Instrucciones	Comentarios
ROL SI,1	Rota a SI 1 lugares a la izquierda
ROR AX,CL	Rota a AX CL lugares a la derecha
RCL BL,1	Rota a BL 1 lugares a la izquierda, a través de CF
RCR AH,CL	Rota a AH CL lugares a la derecha, a través de CF

# Rotaciones



**Figura 13.** Conjunto de operaciones de rotación disponibles en el 8088.



# TEST

La instrucción **TEST** realiza una operación **AND**.

La diferencia es que la instrucción AND cambia el operando destino, mientras que la instrucción **TEST** no lo hace.

Una operación **TEST** sólo afecta la condición del registro de banderas, el cual indica el resultado de la prueba.

Por ejemplo, si se usa la instrucción TEST para probar un único bit, la bandera de cero Z será:

$Z = 1$  si el bit era 0

$Z = 0$  si el bit era 1

# TEST

---

**Tabla 21.** Instrucciones TEST.

Instrucciones	Comentarios
TEST DL,DH	Realiza: DL AND DH
TEST CX,BX	Realiza: CX AND BX
TEST AX,04H	Realiza: AH AND 04



# Multiplicación (MUL e IMUL)

---

La multiplicación se realiza en byte o palabra y puede ser:

- con signo (**IMUL**)
- sin signo (**MUL**).

El resultado después de la multiplicación es siempre un número de doble ancho. Si se multiplican **dos números de 8 bit se genera un producto de 16 bit**, y si multiplicamos **dos números de 16 bit se genera un producto de 32 bits**.

Algunas banderas (OF o CF) cambian cuando la instrucción multiplicación se ejecuta y produce resultados predecibles. Las otras banderas también cambian pero sus resultados son impredecibles y por tanto no son usados.

---



# Multiplicación (MUL e IMUL)

## Multiplicación de 8 bits:

Con la multiplicación de 8 bit, ya sea con signo o sin signo, el multiplicando está siempre en el registro **AL**. El multiplicador puede estar en cualquier registro de 8 bit o en cualquier localidad de memoria.

El resultado de la multiplicación se almacena en el registro **AX**.

**Tabla 13.** Instrucciones de Multiplicación de 8 bits.

Instrucciones	Comentarios
MUL CL	$AX = AL * CL$
IMUL DH	$AX = AL * DH$
IMUL BYTE PTR[BX]	$AX = AL * DS:[BX]$
MUL TEMP	$AX = AL * DS:TEMP$

# Multiplicación (MUL e IMUL)

## Multiplicación de 16 bits:

La multiplicación de palabra es similar a la multiplicación de byte. Las diferencias son que **AX** contiene el multiplicando en lugar de AL, y el resultado se almacena en **DX-AX** en lugar de AX. El registro DX siempre contiene los 16 bit más significativos del producto y AX los 16 bit menos significativos.

**Tabla 14.** Instrucciones de Multiplicación de 16 bits.

Instrucciones	Comentarios
MUL CX	$DX-AX = AX * CX$
IMUL DI	$DX-AX = AX * DI$
MUL WORD PTR[SI]	$DX-AX = AX * DS:[SI]$

# División (DIV e IDIV)

---

La división se lleva a cabo con números de 8 y 16 bit que son números enteros con **signo (IDIV)** o **sin signo (DIV)**.

El **dividendo es siempre de doble ancho**, el cual es dividido por el operando.

Esto quiere decir que una división de 8 bit, divide un número de 16 bit entre uno de 8 bit, la división de 16 bit divide un numero de 32 bit entre uno de 16 bit.

Ninguna de las banderas cambian en forma predecible con una división.



# División (DIV e IDIV)

---

Una división puede resultar en dos tipos diferentes de errores. Uno de estos es intentar dividir entre cero y el otro es un sobreflujo por división.

Un ejemplo donde ocurriría un sobreflujo es el siguiente:

supongamos que  $AX = 3000$  (decimal) y que lo dividimos entre dos.

Puesto que el cociente de una división de 8 bit se almacena en AL, y 1500 no cabe, entonces el resultado causa un sobreflujo por división.



# División (DIV e IDIV)

---

## División de 8 bit:

Una división de 8 bit emplea el registro **AX** para almacenar el dividendo que será dividido entre el contenido de un registro de 8 bit o una localidad de memoria.

Después de la división, el **cociente** se almacena en **AL** y el **residuo** en **AH**.

Para una división con signo el cociente es positivo o negativo, pero el residuo siempre es un número entero positivo.

Por ejemplo, si  $AX=0010H$  (+16) y  $BL=FDH$  (-3) y se ejecuta la instrucción `IDIV BL`, entonces  $AX$  queda  $AX=01FBH$ . Esto representa un cociente de -5 con un residuo de 1.

---





# División (DIV e IDIV)

---

## División de 8 bit:

**Tabla 15.** Instrucciones de División de 8 Bits.

Instrucciones	Comentarios	
DIV CL	AL= AX / CL,	AH=Residuo
IDIV BL	AL= AX / BL,	AH=Residuo
DIV BYTE PTR[BP]	AX=AX / SS:[BP],	AH=Residuo



# División (DIV e IDIV)

## División de 16 bit:

La división de 16 bit es igual que la división de 8 bit excepto que en lugar de dividir AX se divide **DX-AX**, es decir, se tiene un dividendo de 32 bit.

Después de la división, el **cociente** se guarda en **AX** y el **residuo** en **DX**.

**Tabla 16.** Instrucciones de División de 16 bits.

Instrucciones	Comentarios
DIV CX	$AX = (DX-AX)/CX$ , DX=Residuo
IDIV SI	$AX = (DX-AX)/SI$ , DX=Residuo
DIV NUMB	$AX = (DX-AX)/DS:NUMB$ , DX=Residuo

# XLAT

---

Se utiliza para realizar una técnica directa de conversión de un código a otro, por medio de una *lookup table*.

Una instrucción XLAT primero **suma** el contenido de **AL** con el contenido del registro **BX** para formar una dirección del segmento de datos, luego el dato almacenado en esta dirección es cargado en el registro **AL**.

El registro BX almacenaría la dirección de inicio de la tabla y AL almacenaría la posición del dato que se quiere cargar en este registro.

Esta instrucción no posee operando, ya que siempre opera sobre AL.

---



# LEA

Se utiliza para cargar un registro con la dirección de un dato especificado por un operando (variable).

## Ejemplo:

### **LEA AX,DATA**

AX se carga con la dirección de DATA (16 bits), cabe notar que se almacena la dirección y no el contenido de la variable.

En una comparación de la instrucción LEA con MOV se puede observar lo siguiente:

**LEA BX,[DI]** carga la dirección especificada por [DI] en el registro BX. Es decir, estaría copiando el contenido de DI en BX.

**MOV BX,[DI]** carga el contenido de la localidad direccionada por DI en el registro BX.

# LDS y LES

---

Estas instrucciones cambian el rango del segmento de Datos o del segmento Extra, y permiten cargar una nueva dirección efectiva.

Las instrucciones **LDS** y **LES** cargan **un registro de 16 bits** con un dato que representaría una dirección (un desplazamiento), y cargan el **registro de segmento DS** o **ES** con una nueva dirección de segmento.

Estas instrucciones utilizan cualquier modo de direccionamiento a memoria válido para seleccionar la localidad del nuevo desplazamiento y nuevo valor de segmento.

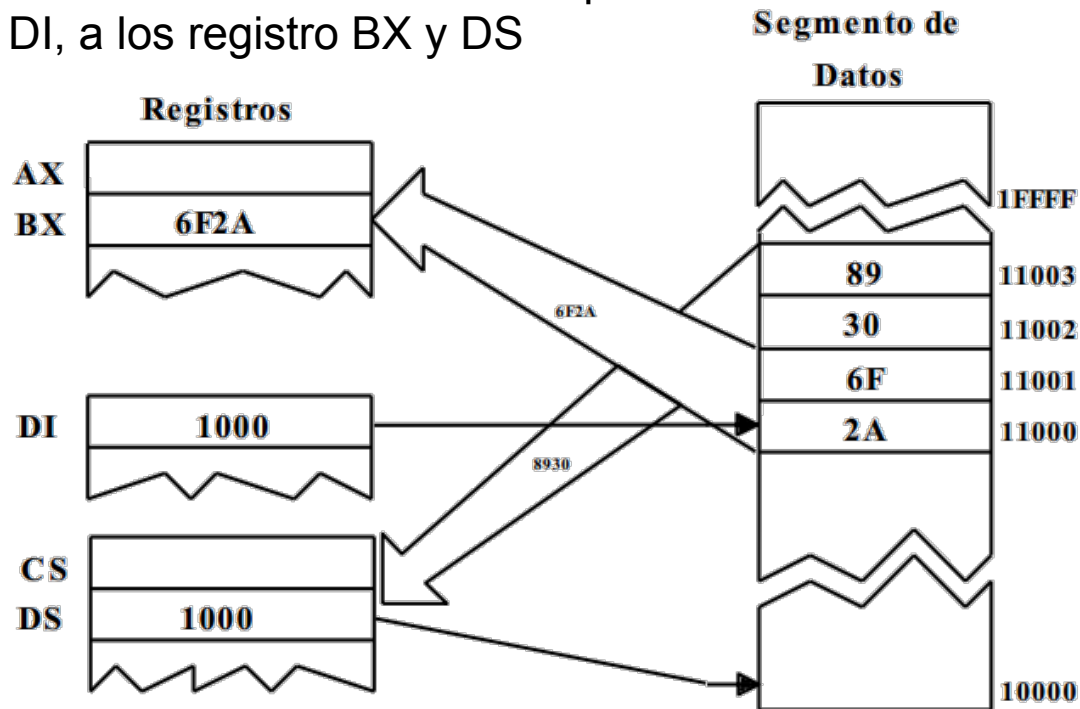
---



# LDS y LES

En el diagrama se muestra la operación de la instrucción **LDS BX,[DI]**, la cual transfiere el contenido (2 bytes) de la localidad de memoria apuntada por DI, al registro BX; y transfiere los siguientes 2 bytes al registro de segmento DS.

Es decir, transfiere la dirección de 32 bits que esta almacenada en memoria y es apuntada por DI, a los registro BX y DS



**Figura 3.** LDS BX,[DI] carga el registro BX con el contenido de la localidad 11000H y 11001H y el registro DS con el contenido de la localidad 11002H y 11003H. Esta instrucción cambia el rango de segmento 1000H-1FFFFH a 89300H-992FFH.

# Ejemplos de LEA, LDS y LES

---

**Tabla 4.** Instrucción XCHG

Código de operación	Función
LEA AX,DATA	AX se carga con la dirección de DATA (16 bits)
LDS DI,LIST	DI y DS se cargan con la dirección de LIST (32 bits- DS:DI)
LES BX,CAT	BX y ES se cargan con la dirección de CAT (32 bits -ES:BX)



# Comparación CMP

La instrucción de comparación **CMP** es una resta que no cambia el valor de los operandos con los que se trabaja, solamente cambia las banderas.

Se emplea una comparación para verificar el contenido de un registro o de una localidad de memoria con otro valor.

Normalmente después de un CMP hay una instrucción de salto condicional, la cual prueba la condición en las banderas.

No se permite usar esta instrucción en los registros de segmento.





# Comparación CMP

---

**Tabla 12.** Instrucciones de Comparación.

Instrucciones	Comentarios
CMP CL,BL	Realiza:CL-BL. CL y BL no cambian
CMP AX,SP	Compara: AX-SP
CMP AX,0CCCCCH	Compara:AX-DS:0CCCC
CMP [DI],CH	Compara: DS:[DI]-CH
CMP CL,[BP]	Compara: CL-SS:[BP]
CMP AH,TEMP	Compara: AH-DS:TEMP
CMP DI,TEMP[BX]	Compara: DI-DS:TEMP[BX]



# Control del Programa

---

Las instrucciones de control de programa permiten que el flujo de ejecución del programa cambie.

Esos cambios en el flujo ocurren después de decisiones hechas con las instrucciones **CMP** o **TEST** seguidas por una instrucción de **salto condicional**, o por medio de una instrucción de **salto incondicional**.



# Salto incondicional **JMP**

El salto (JMP), es el tipo principal de instrucciones de control de programa, permite al programador saltar a secciones de un programa y bifurcar a cualquier parte de la memoria para la siguiente instrucción.

```
MOV AX, 1A7Fh
MOV CL, 2
@@next: ROL AX, CL
INC AX
JMP @@next
ADD BX, 8
```

Nunca se llega  
a ejecutar esta  
instrucción



# Salto incondicional **JMP**

---

Si se visualizan instrucciones JMP en lenguaje máquina, nos podemos encontrar con tres tipos diferentes de saltos:

- Salto **Corto**
- Salto **Cercano**
- Salto **Lejano**

El ensamblador escoge la mejor forma de la instrucción de salto.

Nunca se usa una dirección hexadecimal con cualquier instrucción de salto.



# Salto incondicional **JMP**

---

## **Salto Corto:**

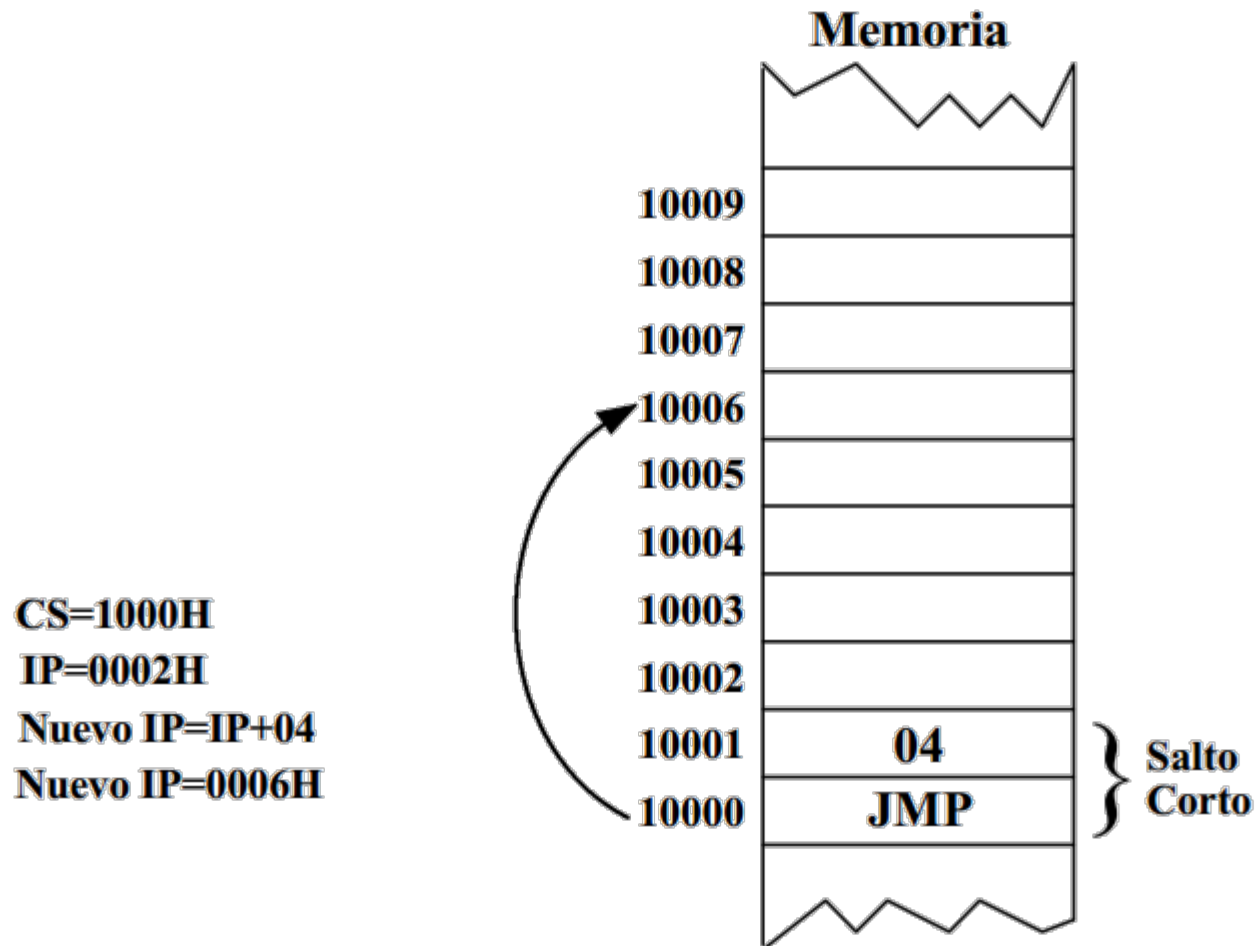
Después del código de operación de la instrucción de salto (de tamaño 1 byte), le sigue 1 byte que representa una **distancia** que toma valores entre **+127 a -128**.

Cuando el 8088 ejecuta una instrucción de salto corto, **suma el valor de distancia a IP** (el apuntador de instrucción), lo que ocasiona que el procesador bifurque hacia la instrucción almacenada en la nueva dirección apuntada por IP.



# Salto incondicional **JMP**

## Salto Corto:



# Salto incondicional **JMP**

---

## **Salto Cercano:**

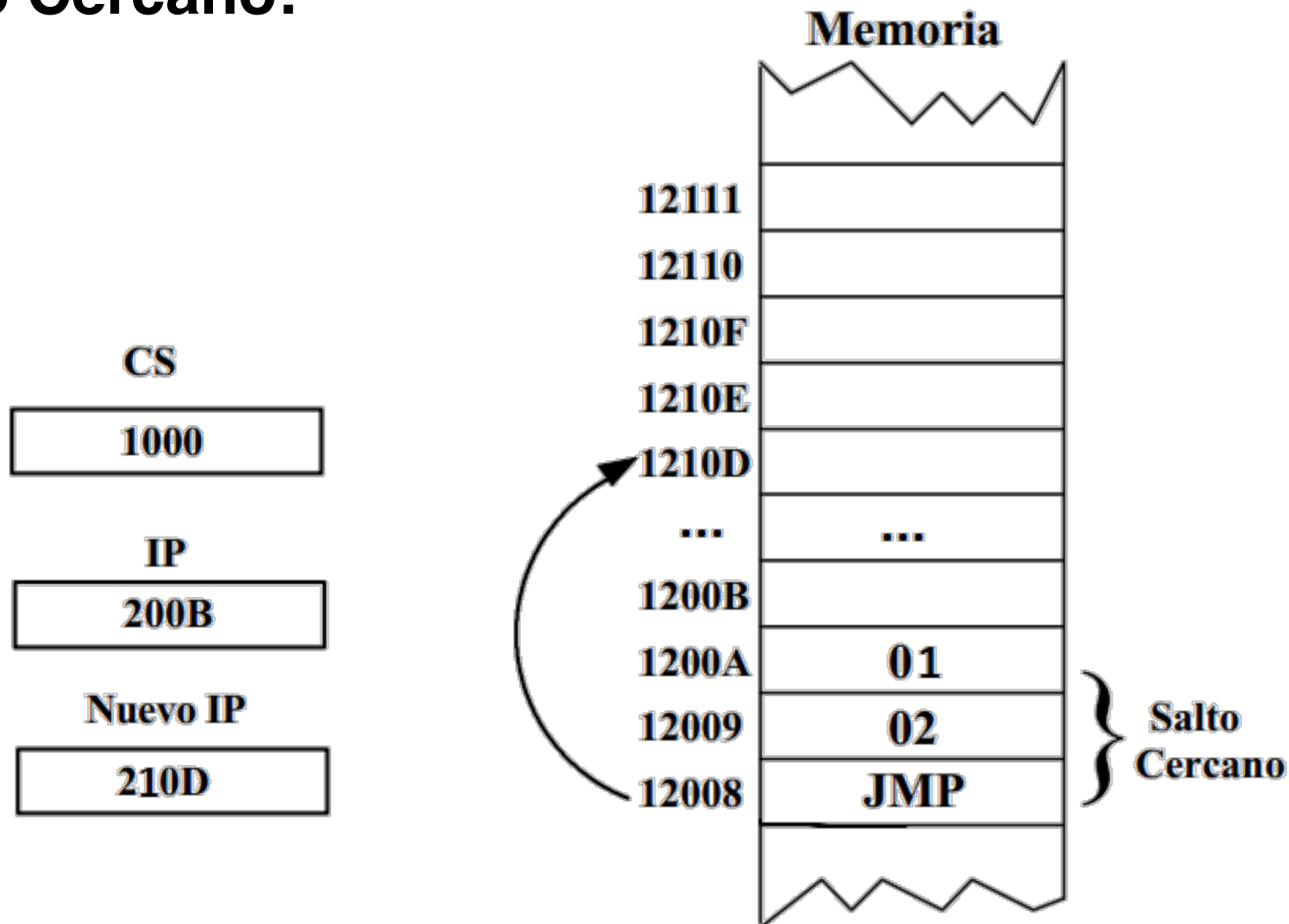
El salto cercano es similar al salto corto excepto que la distancia de salto puede ser mayor.

En este tipo de salto después del código de operación le siguen 2 bytes que indican la distancia de desplazamiento, la cual puede tomar valores de  **$\pm 32\text{Kb}$** .



# Salto incondicional **JMP**

## Salto Cercano:



**FIGURA 3** Un JMP cercano, el cual suma el desplazamiento a el contenido del registro IP.



# Salto incondicional **JMP**

---

## **Salto Lejano:**

Los saltos lejanos obtienen un nuevo segmento y dirección de desplazamiento para efectuar el salto.

Después del código de operación de la instrucción le siguen 4 bytes. De estos cuatro bytes, los 2 primeros establecen el **nuevo valor de IP**, y los siguientes 2 el **nuevo valor de CS** (registro de segmento de código).

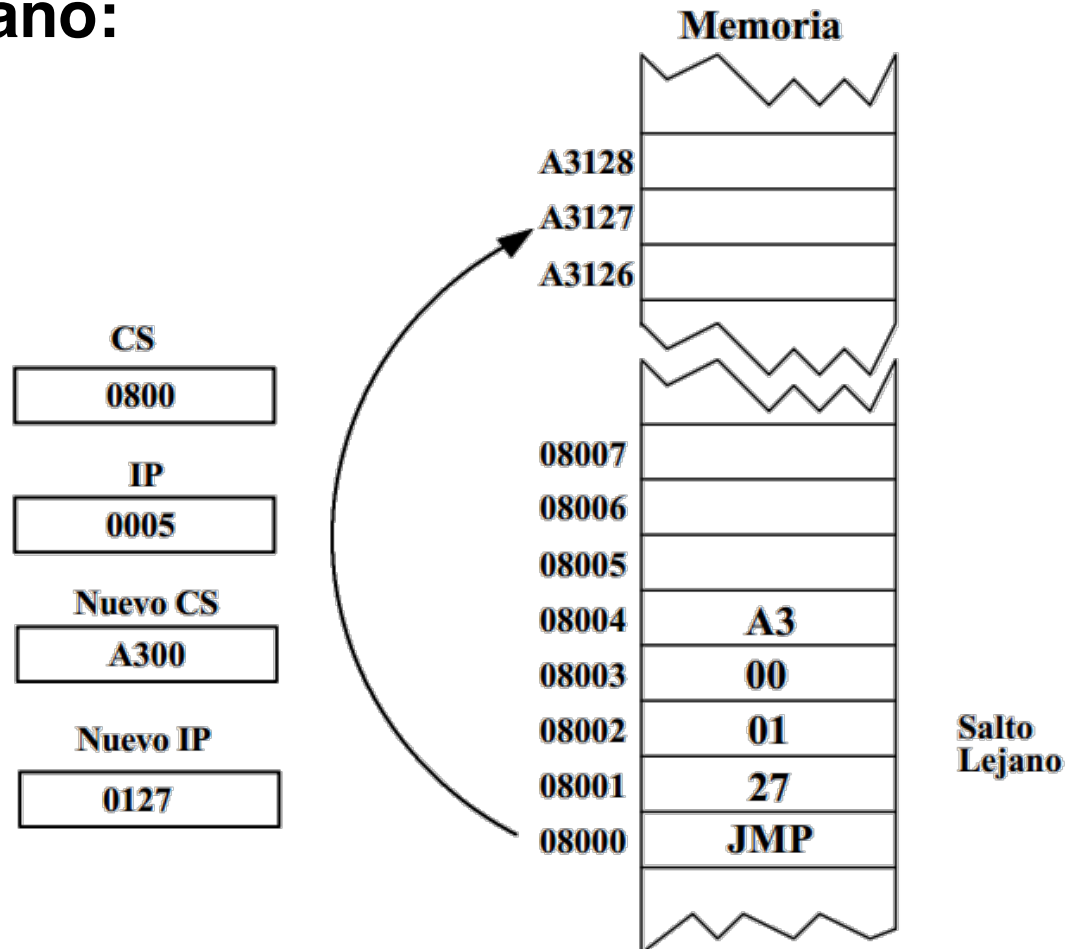
A diferencia de los tipos de salto anteriores, los nuevos valores no se suman a IP y CS, sino que se establecen directamente en los registros.

---



# Salto incondicional **JMP**

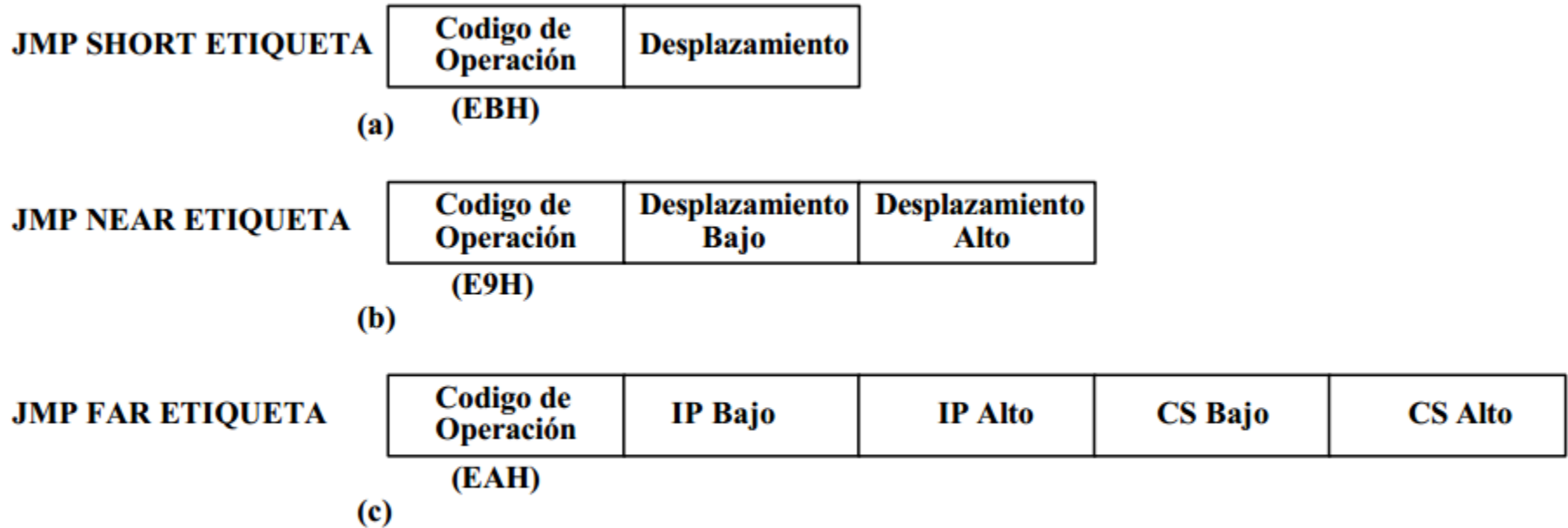
## Salto Lejano:



**FIGURA 4** Un JMP lejano que reemplaza el contenido de los registros CS e IP con los 4 bytes que siguen al código de instrucción.

# Salto incondicional **JMP**

---



**FIGURA 1.** Los tipos de instrucciones de salto. (a) JMP corto (2 bytes), (b) JMP cercano (3 bytes) y (c) JMP lejano (5 bytes).



# Salto incondicional **JMP**

---

## **Salto con Registros como Operando:**

La dirección del salto está en el **registro** especificado por la instrucción de salto.

Distinto al salto cercano, el contenido del registro se transfiere directamente en el apuntador de instrucción (no se suma al apuntador de instrucción como en los saltos cortos y cercanos).

**Ejemplo:**

**JMP AX**

---



# Salto incondicional **JMP**

---

## **Salto Indirectos Usando un Índice:**

La dirección de salto también se puede obtener por medio de direccionamiento indirecto a memoria.

**Ejemplo:**

**JMP [SI]**

La instrucción de salto puede direccionar a un nuevo valor de desplazamiento de 16 bits (IP), o un nuevo valor de segmento y un nuevo valor de desplazamiento (IP y CS, 32 bits en total).

Se asume que se está direccionando a un nuevo valor desplazamiento de 16 bits a menos que se utilice la directiva FAR PTR. **Ejemplo: JMP FAR PTR [SI]**

---



---

## **Salto Condicionado**



# Salto s condicionados

---

Los saltos condicionados son siempre saltos cortos.

Los saltos condicionados examinan los siguientes bits del registro de banderas:

signo (**S**), cero (**Z**), acarreo (**C**), paridad (**P**), y desbordamiento (**O**)

Si la condición bajo prueba es verdadera, ocurre una bifurcación a la etiqueta asociada con la instrucción de salto.

Si la condición es falsa, no se realiza el salto y por lo tanto se ejecuta la instrucción que le sigue.

---



# Saltos condicionados

En JA es "C = 0 y Z = 0"

**TABLA 1.** Instrucciones de salto condicionales

Instrucción	Condicion probada	Comentario
JA	C=0 y Z≠0	Salta si esta arriba
JAE	C=0	Salta si esa arriba o igual
JB	C=1	Salta si esta abajo
JBE	C=1 o Z=1	Salta si esta abajo o igual
JC	C=1	Salta si hay acarreo
JE o JZ	Z=1	Salta si es igual o Sata si es 0
JG	Z=0 y S=0	Salta si es mayor
JGE	S=0	Salta si es mayor o igual a
JL	S≠0	Salta si es menor
JLE	Z=1 o S=1	Salta si es menor o igual
JNC	C=0	Salta si no hay acarreo
JNE o JNZ	Z=0	Salta si no es igual o Salta si no es 0
JNO	O=0	Salta si no hay sobre flujo
JNS	S=0	Salta si no hay signo
JNP	P=0	Salta si no hay paridad
JO	O=1	Salta si hay sobre flujo
JP	P=1	Salta si hay paridad
JS	S=1	Salta si hay signo
JCXZ	CX=0	Salta si CX=0



# Saltos condicionados

---

Cuando se comparan **números sin signo**, se usan las instrucciones de salto:

JA, JB, JAE, JBE, JE, y JNE.

Cuando se comparan **números con signo**, se usan las instrucciones de salto:

JG, JL, JGE, JLE, JE, y JNE



# Saltos condicionados

---

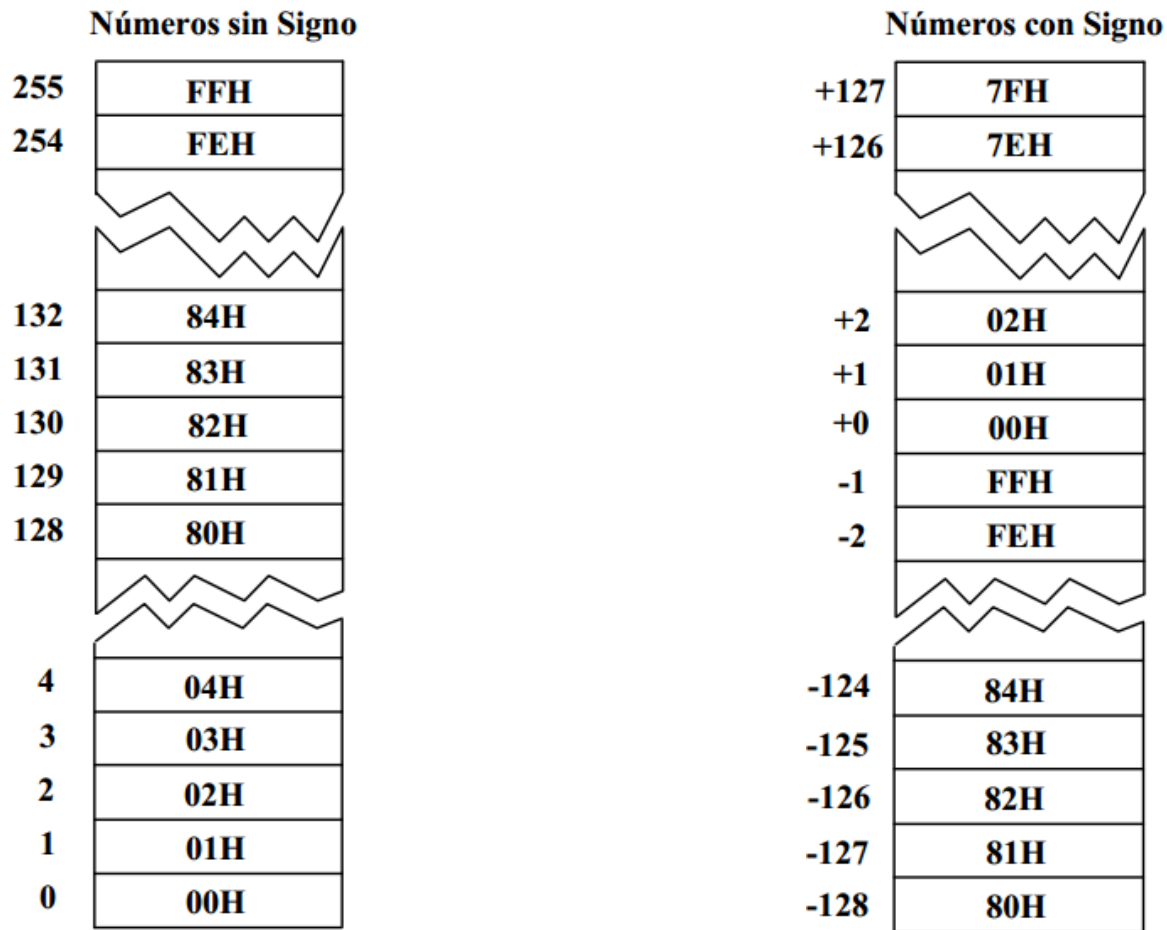
Cuando se trabaja con números sin signo, FFH esta arriba (es mayor) del 00H.

Sin embargo, al manejar los números con signo, un FFH (-1) es menor que 00H.

El siguiente diagrama muestra el orden de los números con signo y sin signo de 8 bits.



# Saltos condicionados



**FIGURA 5.** Numero con signo y sin signo.

# Saltos condicionados

---

## JCXZ:

Esta es la única instrucción de salto condicional que no examina los bits de bandera.

JCXZ examina el contenido del registro CX sin afectar los bits de bandera. Si **CX=0**, **ocurre el salto**, y si CX es diferente de cero, el salto no ocurre.



# Saltos

---

## Ejemplo 1 de saltos:

Programa que lee un dato de 8 bits del puerto 0x2456, si el dato es mayor o igual a 0x1F, envía un 0xAA en el puerto 0x2899, caso contrario envía un 0xBB.

```
MOV DX, 2456h
IN AL, DX
CMP AL, 1Fh
JAE @@mayor
MOV AL, 0BBh
JMP @@out
@@mayor: MOV AL, 0AAh
@@out:   MOV DX, 2899h
         OUT DX, AL
```



# Saltos

---

## Ejemplo 2:

Programa que lee un caracter (un dato de 8 bits) del puerto 0x80, si el dato es una letra *a*, envía un carácter *T* (indicando TRUE) en el puerto 0x92, caso contrario envía una *F* (indicando FALSE).

```
IN AL, 80h
CMP AL, 'a'
JE @@es_a
MOV AL, 'F'
JMP @@out
@@es_a: MOV AL, 'T'
@@out:  OUT 92h, AL
```



# Saltos

---

## Ejemplo 3:

Programa que lee un dato de 8 bits del puerto 0x2345, si el bit 3 del dato esta activo (si es un 1), envía un carácter *T* (indicando TRUE) en el puerto 0x92, caso contrario envía una *F* (indicando FALSE).

```
MOV DX, 2345h
IN AL, DX
TEST AL, 08h
JNZ @@es_1
MOV AL, 'F'
JMP @@out
@@es_1: MOV AL, 'T'
@@out:  OUT 92h, AL
```



---

# Ciclos





# LOOP

---

La instrucción LOOP es una combinación de un decremento en CX y un salto condicionado.

Esta instrucción decrementa a CX y si CX es diferente de cero, se salta a la dirección indicada por la etiqueta.

Si CX llega a cero, no ocurre el salto y por lo tanto se ejecuta la instrucción que le sigue.



# LOOP

---

## Ejemplo:

Escriba un programa que envíe los caracteres del abecedario por el puerto 0x92, uno a uno.

```
MOV AL, 'a'
MOV CX, 26 ;26 decimal, por lo que no se incluye la h
@@out: OUT 92h, AL
INC AL
LOOP @@out
```



# LOOPS Condicionados

---

La instrucción LOOP también tiene las formas condicionales: **LOOPE** y **LOOPNE**.

La instrucción **LOOPE** (**cicla mientras sea igual**) salta si CX es diferente de cero mientras una condición de igual existe. Se **saldrá del ciclo si la condición no es igual o si el registro CX se decrementa a cero.**

La instrucción **LOOPNE** (**cicla mientras no sea igual**) salta si CX es diferente de cero mientras una condición de no igual exista. **Se saldrá del ciclo si la condición es igual o si el registro CX se decrementa a cero.**



---

## Procedimientos



# Procedimientos

---

Un procedimiento es una porción reusable de programa que se almacena en memoria una vez, pero se usa tan seguido como es necesario. Esto hace ahorrar espacio de memoria y hace más fácil desarrollar programas.

La única desventaja de un procedimiento es que hacen que la computadora tome una pequeña cantidad de tiempo para encadenarse al procedimiento y regresar de él.

La instrucción **CALL** se encadena a las subrutinas, y la instrucción **RET** hace que se regrese de ella.

---



# Procedimientos

---

## Ejemplo:

Programa que imprime en pantalla en notación binaria el byte almacenado en AL.

```
printBin PROC
    push ax
    push cx
    mov ah,al
    mov cx,8
@@print: xor al, al ; se pone en 0 al registro AL
    shl ah,1
    adc al,'0'
    call putchar
    loop @@print

    pop cx
    pop ax
    ret
ENDP
```

El procedimiento ***putchar*** espera en el **registro AL** el carácter ASCII a imprimir en pantalla.

Archivo **printBin.asm**

## Ejemplo (cont.):

```
MODEL small
.STACK 100h

;----- Insert INCLUDE "filename" directives here
;----- Insert EQU and = equates here

INCLUDE procs.inc

LOCALS

.DATA
msg_bin db 'AL desplegado en Binario:',0
new_line db 13,10,0

.CODE

;----- Insert program, subroutine call, etc., here

Main PROC
    mov ax,@data      ; Inicializar DS al la direccion
    mov ds,ax         ; del segmento de Datos (.DATA)

    call clrscr
    mov dx, offset msg_bin
    call puts
    mov dx, offset new_line
    call puts
    mov al,0a7h       ; dato a imprimir en binario
    call printBin

    ; Fin del Programa ;
    mov ah,4Ch
    mov al,0
    int 21h
ENDP
```

# Archivo completo

```
MODEL small
.STACK 100h

;----- Insert INCLUDE "filename" directives here
;----- Insert EQU and = equates here

INCLUDE procs.inc

-----

LOCALS

.DATA
msg_bin db 'AL desplegado en Binario:',0
new_line db 13,10,0

.CODE

;----- Insert program, subroutine call, etc., here

Main PROC
    mov ax,@data    ; Inicializar DS al la direccion
    mov ds,ax       ; del segmento de Datos (.DATA)

    call clrscr
    mov dx, offset msg_bin
    call puts
    mov dx, offset new_line
    call puts
    mov al,0a7h      ; dato a imprimir en binario
    call printBin

    ; Fin del Programa ;
    mov ah,4Ch
    mov al,0
    int 21h
    ENDP

printBin PROC
    push ax
    push cx
    mov ah,al
    mov cx,8
@@print: xor al, al ; se pone en 0 al registro AL
    shl ah,1
    adc al,'0'
    call putchar
    loop @@print

    pop cx
    pop ax
    ret
    ENDP

END
```

Archivo printBin.asm



# Procedimientos

---

La pila almacena la dirección de regreso siempre que una subrutina es llamada durante la ejecución del programa.

La instrucción **CALL** empuja en la pila la dirección de la siguiente instrucción.

La instrucción **RET** remueve una dirección de la pila así que el programa regresa a la instrucción siguiente del CALL.



# Procedimientos

---

## **Tipos de Llamadas**

CALL Cercano

CALL Lejano

CALL con Operando de Registro

CALL con Direccionamiento Indirecto de Memoria



# Procedimientos

---

## **CALL Cercano**

Es de tres bytes de largo con el primer byte conteniendo el código de operación y el segundo y tercero contienen el desplazamiento o distancia de  $\pm 32K$ .

Esta es similar a la forma la instrucción de salto cercano.

Cuando la **CALL cercana** se ejecuta, primero coloca en la **Pila** la dirección de la siguiente instrucción a ejecutar (coloca el valor de **IP**). Después de guardar esta dirección de regreso, entonces **suma el desplazamiento** del byte 2 y 3 a **IP** para transferir el control al procedimiento.

No hay instrucciones CALL cortas.

---



# Procedimientos

---

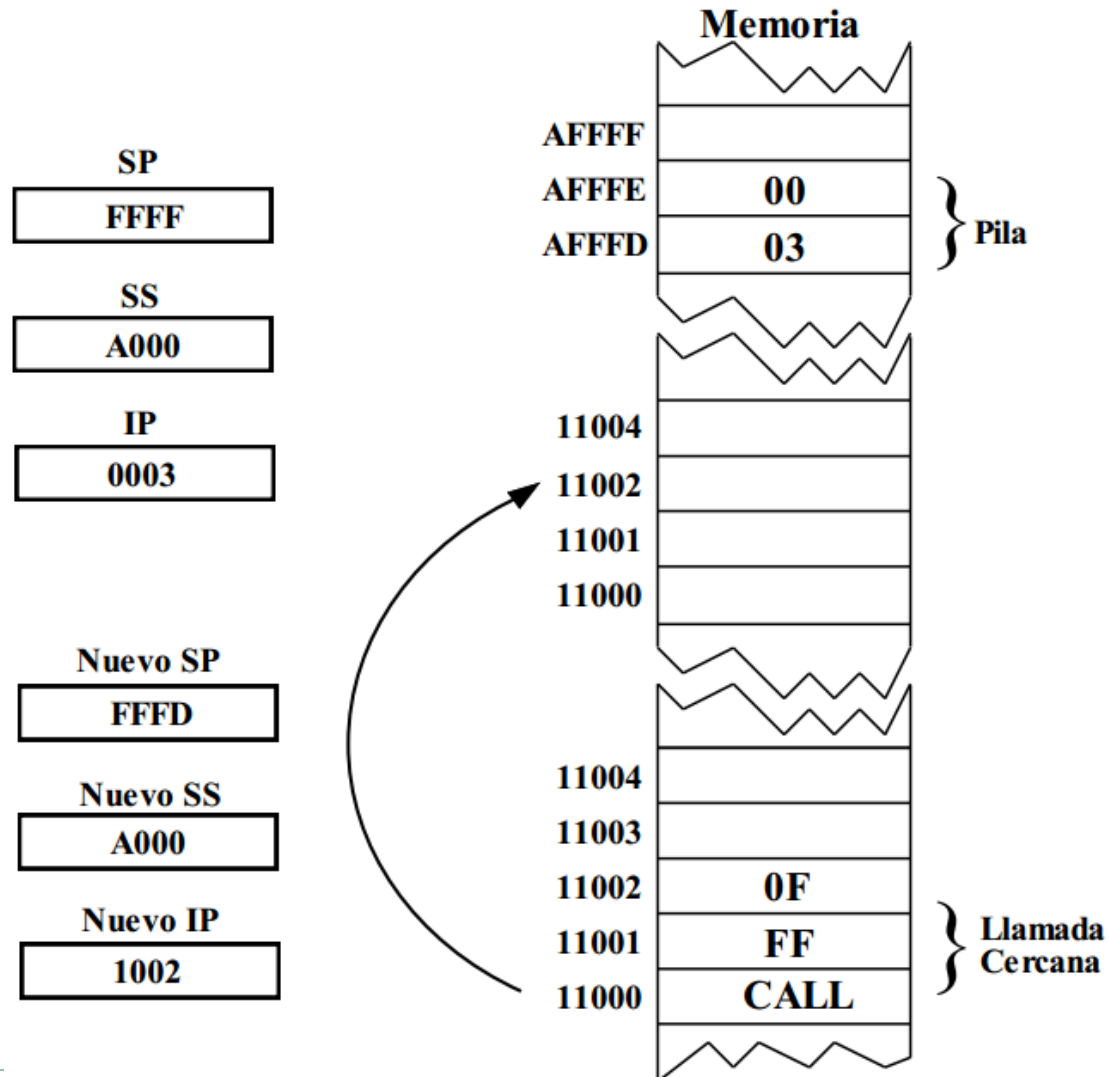
## **¿Porque guardar IP en la pila?**

El apuntador de instrucción siempre apunta a la siguiente instrucción en el programa.

Para la instrucción **CALL**, el contenido de **IP** es empujado dentro de la pila así que el control del programa pasa a la instrucción seguida de **CALL** después de que el procedimiento termina.



# Procedimientos



# Procedimientos

---

## **CALL Lejano**

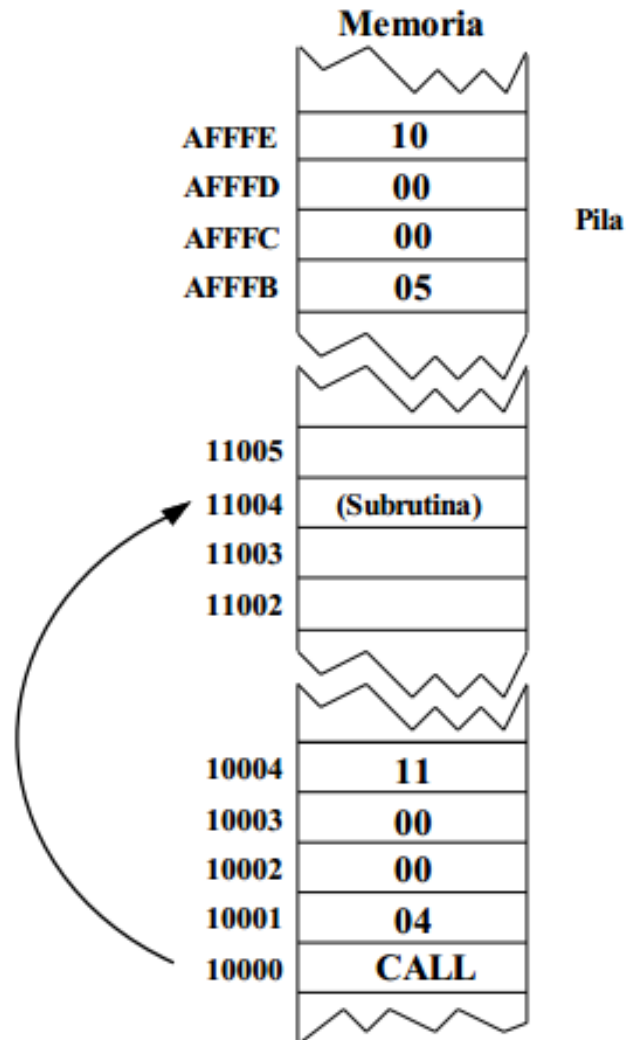
La instrucción **CALL** lejana es como el salto lejano porque esta puede llamar un procedimiento almacenado en cualquier parte de la memoria en el sistema.

La **CALL** lejana es una instrucción de 5 bytes que contiene un código de operación seguido por el próximo valor de **IP** y los bytes 4 y 5 contienen el nuevo valor para **CS**.

La instrucción **CALL** lejana coloca el contenido de **IP** y **CS** en la **Pila** antes de saltar a la dirección indicada por los bytes 2 al 5 de la instrucción.



# Procedimientos



**FIGURA 2.** Instruccion CALL lejano

# Procedimientos

---

## **CALLs con Operando de Registro**

Como los saltos, las CALLs pueden contener también un registro como operando.

Un ejemplo es la instrucción **CALL BX**. Esta instrucción empuja el contenido de IP en la Pila. Luego se salta a la dirección de desplazamiento localizado en BX, en el presente segmento de código.

Este tipo de CALL siempre usa una dirección de 16 bits almacenado en cualquier registro de 16 bits excepto los registros de segmento.





# Procedimientos

---

## **CALLs con Direcccionamiento Indirecto de Memoria**

Como los saltos, las CALLs pueden usar direccionamiento indirecto a memoria.

**Ejemplo:**

**CALL [SI]**



# Procedimientos

---

## **CALL vs Saltos**

La instrucción CALL difiere de las instrucciones de salto porque una CALL guarda en la pila una dirección de retorno.

La dirección de retorno permite que se pueda ejecutar la instrucción que seguía al CALL.



# Procedimientos

---

## **RET**

La instrucción de retorno (RET) obtiene un número de 16 bits de la pila y lo coloca en IP (retorno cercano), o un número de 32 bits y lo coloca en IP y CS (retorno lejano).

Las instrucciones de retorno cercano y lejano se definen con la directiva PROC en el procedimiento. Esta selecciona automáticamente la instrucción de retorno apropiada.



# Procedimientos

---

## **Ejemplo de Procedimientos**



# Procedimientos

## Ejemplo:

```
call step  
mov ax,cx  
mov cx,dx  
call step  
mov dx,bx
```

1

```
step PROC  
    add bx,dx  
    mov dx,cx  
    ret  
ENDP
```

- I. Se empuja en la Pila la dirección de retorno, es decir, el valor de **IP** (o si es una **CALL lejana**, se empuja **CS** y después **IP**)
  - I.1 Se pone el nuevo valor a **IP** de forma que apunte a la primera instrucción de la rutina (en este ejemplo, que apunte a 'add bx,dx'). El nuevo valor de **IP** se obtiene ya sea **sumándole el desplazamiento (CALL cercana)** o poniendo el **valor directamente** al igual que al registro **CS (CALL lejana)**

# Procedimientos

## Ejemplo:

```
call step  
mov ax, cx  
mov cx, dx  
call step  
mov dx, bx
```

```
step PROC  
    add bx, dx  
    mov dx, cx  
    ret  
ENDP
```

2



2. Se ejecutan secuencialmente las instrucciones de la rutina 'step', hasta llegar a RET.

2.1 La instrucción RET remueve un valor de la pila y lo coloca en IP (si es una CALL lejana remueve otro y lo coloca en CS)

# Procedimientos

## Ejemplo:

```
call step
mov  ax, cx
mov  cx, dx
call step
mov  dx, bx
```

3



```
step PROC
    add bx, dx
    mov dx, cx
    ret
ENDP
```

3. Causando que ahora IP apunte a 'mov ax,cx'. Por lo que continua la ejecución secuencial de las instrucciones.

---

## Instrucciones misceláneas





# Control del Bit de Acarreo

Hay tres instrucciones que pueden controlar el contenido de la bandera de acarreo:

**STC:** poner en uno el acarreo

**CLC:** borrar el acarreo

**CMC:** complementar el acarreo

El bit de acarreo indica el estado de la última operación aritmética, sin embargo también se le pueden dar otros usos. La tarea más común es para indicar un error en el retorno de una subrutina.

Por ejemplo, en una subrutina que lee datos de un archivo, esta operación puede ser desarrollada satisfactoriamente o puede ocurrir un error tal como: archivo no encontrado. Después de retornar de esa subrutina, se puede poner  $CF = 1$  indicando que ha ocurrido un error, y  $CF = 0$ , si no ha ocurrido error.



# NOP

---

Una **NOP** no realiza ninguna operación.

Cuando el microprocesador 8088/86 encuentra una instrucción de no operación (NOP), toma tres períodos de reloj para ejecutarla.

Se puede utilizar para programar un delay por software.



---

# Interrupciones



# Interrupciones

---

Una interrupción es una llamada generada por el hardware (derivada en el exterior por una señal de hardware) o una llamada generada por el software (derivada en el interior por una instrucción).

Cualquiera de ellas interrumpirá el programa porque llamara a una rutina de servicio de interrupción RSI.

Una interrupción es un evento que interrumpe la operación normal del procesador



# Interrupciones

---

Un vector de interrupción es un número de 4 bytes que contiene la dirección (valores de IP y de CS) de una **Rutina de Servicio de Interrupción** RSI (ISR *Interrupt Service Routine*).

Hay 256 vectores de interrupción diferentes, cada uno de los cuales contiene la dirección de la ISR que se va a invocar cuando ocurra la interrupción correspondiente.

Los vectores de interrupción se almacenan en los primeros 1024 bytes de memoria.



# Interrupciones

---

Una **Rutina de Servicio de Interrupción** es un procedimiento que será invocado en el momento en que ocurra la interrupción.

Es una secuencia de instrucciones encargadas de manejar el evento que causó la interrupción.

La llamada a la ISR es similar a una instrucción CALL lejana porque se coloca en la pila la dirección de retorno (IP y CS).

La diferencia es que cuando se invoca una ISR también se coloca en la pila una copia del registro de banderas.



# Interrupciones

---

Siempre que se ejecuta una interrupción por programación:

1. Se empujan las banderas en la pila
2. Se ponen en cero los bits de bandera IF y TF
3. Se empuja CS en la pila,
4. Se busca en el vector el nuevo valor para CS,
5. Se empuja IP a la pila
6. Se busca en el vector el nuevo valor para IP
7. Se salta a la nueva dirección localizada en CS e IP.



# Interrupciones

---

Una diferencia entre una ISR y un procedimiento *normal*, es que la ISR termina con una instrucción **IRET** en lugar de la instrucción RET que se utiliza en los procedimientos.

## **IRET:**

Diferente a la instrucción de retorno simple (RET), la instrucción IRET:

1. saca un dato de la pila para IP
2. saca un dato de la pila para CS
3. saca un dato de la pila para el registro de banderas.





# Interrupciones

---

**TABLA 1.** Vector de Interrupcion

Número	Dirección	Función
0	0H-3H	Division por cero
1	4H-7H	Paso sencillo
2	8H-BH	NMI (interrupcion por circuiteria)
3	CH-FH	Punto de ruptura o de paro (Breakpoint)
4	10H-13H	Interrupcion por sobre flujo
5-31	14H-7FH	Reservadas para futura utilizacion*
32-255	80H-3FFH	Interrupciones del usuario



# Interrupciones

---

Intel reservó los primeros 32 vectores de interrupción para el 80286 y futuros productos.

Los vectores de interrupciones restantes (32-255) están disponibles para el usuario.



# Interrupciones

---

El 8088/86 tiene disponibles tres instrucciones de interrupción diferentes para el programador: INTO, INT 3 e INT.

## **INTO, Interrupción de sobreflujo:**

Es una interrupción condicionada por programación que examina la bandera de sobreflujo (OF).

Si  $OF=0$ , la instrucción INTO no realiza operación, pero si  $OF=1$  y se ejecuta la instrucción INTO, ocurre una interrupción, ejecutándose la rutina almacenada en la dirección que indica el Vector de Interrupción 4.



# Interrupciones

---

## **INT 3, Breakpoint:**

Es una interrupción diseñada para ser usada como un punto de ruptura (breakpoint).

La diferencia entre este y otras interrupciones por programación es que INT 3 es una instrucción de un byte, mientras que las otras son instrucciones de dos bytes.

Un punto de ruptura ocurre para cualquier interruptor por programación, pero como INT 3 es de un byte de largo es más fácil de usar para esta función. Los puntos de ruptura ayuda para depurar programas defectuosos.



# Interrupciones

---

## **INT (interrupciones en general):**

Hay 256 instrucciones de interrupción por programación diferentes disponibles para el programador.

Cada instrucción INT tiene un operando numérico cuyo rango es entre 0 y 255 (00H-FFH).

Por ejemplo; el INT 100 (decimal) usa el vector de interrupción 100, el cual aparece en la dirección de memoria 190H -193H.

Se calcula la dirección del vector de interrupción multiplicando el número del tipo de interrupción por 4.

---



# Interrupciones

---

Como se mencionó, cuando se ejecuta la instrucción INT, se pone en cero la bandera de interrupción (IF), la cual controla las interrupciones de circuitería externa (interrupciones de hardware) en la terminal de entrada INTR (requisición de interrupción) del chip del procesador.

Cuando IF es igual a cero, el microprocesador deshabilita la terminal INTR, y cuando IF es igual a uno, el microprocesador habilita la terminal INTR.



# Interrupciones

---

## **Ejemplo:**

Ejemplo de una ISR



# Interrupciones

---

## **Ejemplo (cont.):**

Invocación a la ISR

