UNIDAD III
CONJUNTO DE INSTRUCCIONES (LENGUAJE ENSAMBLADOR)

- Introducción
  - Instrucciones con 0, 1,2 operandos
- Instrucciones para movimiento de datos
  - MOV
  - XCHG
  - LAHF, SAHF
  - PUSH Y POP
  - PUSHF Y POPF
  - IN, OUT
- Instrucciones aritméticas
  - ADD, ADC
  - SUB, SBB
  - NEG
  - INC, DEC
  - CMP
  - MUL, IMUL
  - DIV, IDIV
  - Otras instrucciones
- Instrucciones lógicas
  - AND, OR, XOR, NOT, TEST
  - CORRIMIENTOS Y ROTACIONES
  - STRINGS
- Instrucciones de transferencia de control
  - JMP, CALL, RET
  - LOOP, LOOPZ, etc. (Ciclos condicionados)
- Otras instrucciones

INTRODUCCION

Una instrucción en lenguaje ensamblador está dividida en campos. El primer campo se llama Codigo de Operación (OP CODE), el cual le indica a la computadora que hacer. Los otros campos se llaman operandos, los cuales indican la información necesaria por la instrucción para llevar a cabo su tarea. Las instrucciones pueden contener varios operandos, pero mas operandos y mas longitud de esos operandos implican el uso de mas memoria, y mas tiempo para transferir cada instrucción hacia el CPU.

Anatomía de una instrucción:

OPCODE	{operando1},	{operando2}

Generalmente una instrucción puede contener 0,1 o 2 operandos. Las instrucciones con 0 operandos trabajan en algunos casos con registros implícitos. En el caso de ser un solo operando, la instrucción es específica y realiza una tarea particular sobre el operando. En el caso de dos operandos, no se pueden incluir dos localidades de memoria al mismo tiempo. Es decir, una localidad de memoria solo puede combinarse con un registro o un valor en una misma instrucción.

Ejemplos de Instrucciones con 0 operandos:	NOP, LAHF,	PUSHF,	POPF
Ejemplos de Instrucciones con 1 operando:	PUSH AX,	INC BX,	POP BX
Ejemplos de Instrucciones con 2 operandos:	MOV AH,FF	ADD AX,BX	

INSTRUCCIONES PARA MOVIMIENTO DE DATOS

Las instrucciones de movimiento de datos son las mas frecuentemente utilizadas en el conjunto de instrucciones de cualquier computadora. Muchos de los problemas de procesamiento de datos consisten en movimiento de información de un lugar a otro. Un programa puede hacer algún procesamiento de la información cuando ésta es movida, pero el grueso del trabajo es justamente el movimiento.

MOV

Sintaxis: **MOV** Destino, Origen

Descripción: Es la instrucción de movimiento mas común, y requiere dos operandos. Los datos son movidos del "Origen" al "Destino" sin alterar el Origen.

Restricciones:

- El Destino No puede ser el registro CS, ya que CS siempre tendrá el código del programa que se está ejecutando. Alterarlo en condiciones no controladas sería un caos. La Unica manera de cambiarlo es con el uso de INT, JMP o CALL.
- No se puede mover un valor inmediato a un registro de segmento. Si se requiere hacer este movimiento, debe usarse un registro auxiliar. Por ejemplo MOV DS, 3 es inválido, por lo que se debe hacer uso de otro registro por ejemplo AX. Es decir, Mov AX, 3 y luego MOV DS,AX
- El "Destino" debe ser siempre una localidad o un registro.
- El "Destino" Y "Origen" no pueden ser dos localidades de memoria. Por ejemplo, MOV [2000], [3000] es inválido.

Uso de la instrucción MOV

- MOV registro, registro
- MOV registro, memoria
- MOV memoria, registro
- MOV [BYTE PTR o WORD PTR] memoria, valor inmediato
- MOV registro, valor inmediato (Excepto registros de segmentos)
- MOV registro de segmento, memoria
- MOV registro, registro de segmento
- MOV memoria, registro de segmento

Notas Importantes

- La instrucción MOV no afecta banderas.
- "Origen" y "Destino" deben ser ambos de 8 bits o ambos de 16 bits.

XCHG

Sintaxis: **XCHG** operando1, operando2

Descripción: Permite intercambiar el valor de dos locaciones. La instrucción puede intercambiar el valor de dos registros o un registro y una localidad de memoria

Restricciones: Un registro de segmento no puede ser una de las locaciones del intercambio. El valor de dos localidades de memoria no pueden ser intercambiadas con XCHG

Uso de la instrucción XCHG

- XCHG registro, registro Ejemplo: XCHG AX,BX
- XCHG registro, memoria Ejemplo: XCHG AH,[2000]
- XCHG memoria, registro Ejemplo: XCHG [2000],BL

Notas Importantes

- La instrucción XCHG toma el lugar de tres instrucciones de mover, y no requiere una localidad de almacenamiento temporal. Por ejemplo, Si no existiera la instrucción XCHG, el procedimiento para intercambiar los valores de AX y BX sería:
  1. Mover AX a una localidad temporal.
  2. Mover BX a AX
  3. Mover la localidad temporal a BX.

  Sin embargo, esto no es necesario, pues el XCHG lleva a cabo esta operación con una simple instrucción.
- "Operando 1" y "Operando 2" deben ser ambos de 8 bits o ambos de 16 bits.

LAHF Y SAHF

Sintaxis: LAHF
SAHF

Descripción:

- LAHF toma los 8 bits mas bajos del registro de banderas y los mueve hacia el registro AH.
- SAHF toma los 8 bits de AH y los mueve hacia los 8 bits mas bajos del registro de banderas.

Uso de las instrucciones LAHF y SAHF

Estas instrucciones no requieren operandos pues llevan implícito al registro AH.

PUSH Y POP

Sintaxis: PUSH operando POP operando

Descripción:

- La instrucción PUSH guarda los 16 bits de su operando (registro o localidad de memoria) en la pila.
- La instrucción POP saca los ultimos 16 bits introducidos a la pila y los asigna a su operando (registro o localidad de memoria).

Restricciones:

- Para todas las operaciones con la pila (PUSH o POP) la unidad básica de información es una palabra (16 bits o 2 bytes), es decir, todas las cosas que sean guardadas o sacadas de la pila son de una o mas palabras de longitud. No hay operaciones de un byte disponibles con PUSH o POP. Por ejemplo para guardar AL se requiere guardar AX.

Uso de PUSH y POP:

PUSH registro	Ejemplo:	PUSH AX
PUSH localidad de memoria	Ejemplo:	PUSH [2000]
POP registro	Ejemplo:	POP BX
POP localidad de memoria	Ejemplo:	POP [2000]

Notas importantes:

- La pila (Stack) sirve como un lugar para guardar información temporal con tecnología LIFO (Last Input, First Output). Su uso es muy importante en las subrutinas, pues así se pueden guardar los parámetros entre funciones, localidades de retorno, etc.
- Cualquier localidad de memoria que puede ser especificada por el programa con los modos de direccionamiento también puede ser guardada o sacada de la pila.

PUSHF Y POPF

Sintaxis: PUSHF POPF

Descripción:

- La instrucción PUSHF guarda los 16 bits del registro de banderas (PSW) en la pila.
- La instrucción POPF saca los ultimos 16 bits introducidos a la pila y los asigna al registro de banderas (PSW).

Uso de PUSHF y POPF:

- Estas instrucciones no requieren operandos pues llevan implícito al registro de banderas.

Notas importantes:

- La combinación de PUSH Y POPF permite cambiar el contenido del registro de banderas.
- La combinación de PUSHF y POP permite examinar el contenido del registro de banderas.

IN Y OUT

Sintaxis:

IN AL, DX	OUT DX, AL
IN AL, Puerto (8 bits hex)	OUT Puerto (8 bits hex) , AL
IN AX, DX	OUT DX, AX
IN AX, Puerto (8 bits hex)	OUT Puerto (8bits hex) , AX

Descripción:

- La instrucción IN transfiere un valor leído desde el puerto especificado y lo transfiere hacia AL (8 bits) o AX (16 bits).
- La instrucción OUT envía el valor del registro AL o AX hacia el puerto especificado.

Notas importantes:

- La dirección del puerto debe escribirse en hexadecimal
- La dirección del puerto puede escribirse directamente en las instrucciones IN o OUT sin necesidad de usar el registro DX siempre y cuando la dirección de ese puerto sea de 8 bits. Ejemplo:

  IN AL,20 (Lee los datos del puerto Num, 20 y coloca la información en AL)
  IN AL,378 (Error, la dirección del puerto excede los 8 bits)
  En este caso, se debe usar el registro DX para indicar el puerto.

- Si la dirección del puerto excede los 8 bits, se debe utilizar el registro DX para poner ahí la dirección hexadecimal. Ejemplo:

  Para leer 8 bits de datos del puerto 378 y colocarlos en el registro AL se hace lo sig.
  MOV DX,378
  IN AL, DX

Acerca de los puertos:

- La PC se encuentra interconectada con dispositivos externos a través de ciertos circuitos llamados puertos, los cuales permiten que la computadora tenga una ventana al mundo exterior.
- Los puertos mas comunes de la PC son el serial (usado comúnmente para el mouse) y el paralelo (usado comúnmente para la impresora). Ambos puertos pueden utilizarse para controlar dispositivos mas complejos (como sensores, motores, etc).

INSTRUCCIONES ARITMÉTICAS

ADD

Sintaxis: **ADD** destino, fuente

Descripción: Realiza la operación destino = destino + fuente es decir, suma los dos operandos hexadecimales (destino y fuente) y guarda el resultado en el operando destino, borrando su contenido anterior.

Ejemplos:
ADD AH, AL (Realiza la Suma AH + AL y el resultado lo pone en AH)
ADD CX, BX (Realiza la suma CX + BX y el resultado lo pone en CX)
ADD AH,11 (Suma 11 al contenido de AH y el resultado se pone en AH)
ADD BYTE PTR [2000],11 (Suma 11 al contenido de la localidad [2000])

Notas Importantes: La Instrucción ADD cambia el valor de las banderas SF,ZF,PF,CF,AF,OF de acuerdo al resultado de la suma.

ADC (ADD with Carry) (Adición con acarreo)

Sintaxis: ADC destino, fuente

Descripción: ADC Lleva a cabo la suma de dos operandos (igual que la instrucción ADD) y además suma el bit de la bandera de Carry (CF). El resultado se guarda en el operando destino. Esta instrucción puede utilizarse para realizar operaciones de precisión múltiple (por ejemplo, la suma de dos cantidades de 32 bits).

Ejemplo: ADC AX, BX (AX = AX + BX + CF)

Notas Importantes: La Instrucción ADC cambia el valor de las banderas SF,ZF,PF,CF,AF,OF de acuerdo al resultado de la suma.

SUB

Sintaxis: SUB destino, fuente

Descripción: Realiza la operación destino = destino - fuente y guarda el resultado en el operando destino, borrando su contenido anterior.

Ejemplos:
SUB AH, AL (AH = AH – AL)
SUB CL,02 (CL = CL – 2)
SUB BYTE PTR [2000],02 ([2000] = [2000] - 02)

Notas Importantes: La Instrucción SUB cambia el valor de las banderas SF,ZF,PF,CF,AF,OF de acuerdo al resultado de la resta. La bandera de Carry (CF) representa un borrow (Prestamo).

SBB

Sintaxis: SBB destino, fuente

Descripción: Realiza la operación destino = destino - fuente - CF y guarda el resultado en el operando destino, borrando su contenido anterior.
SBB Lleva a cabo la resta de los dos operandos (igual que la instrucción SUB) y además resta el bit de la bandera de Carry (CF). Esta instrucción puede utilizarse para realizar operaciones de precisión múltiple (por ejemplo, la resta de dos cantidades de 32 bits).

Ejemplo: SBB AX, CX (AX = AX – CX - CF)

Notas Importantes: La Instrucción SBB cambia el valor de las banderas SF,ZF,PF,CF,AF,OF de acuerdo al resultado de la resta. La bandera de Carry (CF) representa un borrow (Prestamo).

NEG

Sintaxis: NEG destino

Descripción: Genera el complemento a 2 del operando especificado en "destino" y lo almacena en este mismo operando.

Ejemplo: NEG AX
Si AX guarda el valor de 1234, entonces la instrucción NEG AX dejaría almacenado en el registro AX el valor EDCC.

Importante: El complemento a 2 de un número equivale a su representación negativa (signo inverso) , y equivale a convertir los ceros por unos y unos por ceros de ese numero y sumar uno al resultado. Se afectan las Banderas: SF,ZF,PF,CF,AF,OF

INC

Sintaxis: INC operando

Descripción: Suma 1 al operando. Es decir operando = operando + 1

Ejemplos: INC AX (Suma 1 al contenido de AX) (Equivale a ADD AX,1)
INC BYTE PTR [2000] (Suma 1 al contenido de la localidad [2000]

Importante: Banderas afectadas: SF,ZF,PF,AF,OF

DEC

Sintaxis: DEC operando

Descripción: Resta 1 al operando. Es decir operando = operando - 1

Ejemplos: DEC AX (Resta 1 al contenido de AX) (Equivale a SUB AX,1)
DEC BYTE PTR [2000] (Resta 1 al contenido de la localidad [2000]

Importante: Banderas afectadas: SF,ZF,PF,AF,OF

CMP

Sintaxis: CMP operando1, operando2

Descripción: El comando CMP compara dos valores por medio de restarlos. Es decir, CMP realiza la resta de los dos operandos, pone las banderas de status pero ignora el resultado pues no lo almacena en ninguna parte.

Ejemplos: CMP AX, BX (Resta AX – BX, el resultado se ignora pero PSW es actualizado)
CMP AH, FF (Resta AH – FF, el resultado se ignora pero PSW es actualizado)
CMP BYTE PTR [2000], FF (Compara [2000] con FF , ignora el resultado)

Importante: Banderas Afectadas: SF,ZF,PF,CF,AF,OF
Uno de los operandos debe ser un registro a menos que "operando2" sea inmediato. "Operando1" puede tener cualquier modo de direccionamiento excepto el inmediato.
Si después de comparar dos cantidades:
ZF = 1 indica que operando1 = operando2
CF = 1 indica que operando2 > operando1
ZF = 0 indica que operando1 <> operando2
CF = 0 y ZF = 0 indica que operando1 > operando2

MUL

Sintaxis: MUL operando

Descripción: MUL multiplica dos enteros sin signo para producir un resultado sin signo.
Para **8 bits** realiza la operación **AX = AL * operando de 8 bits**
(Operando de 8 bits: registro o localidad de memoria BYTE PTR)
Para **16 bits** realiza la operación **DX:AX = AX * operando de 16 bits**
(Operando de 16 bits: AX, BX, CX, DX, SI, DI, BP, SP, WORD PTR)

Ejemplo: El siguiente fragmento de programa permite multiplicar 05 * 10 (Hexadecimal)

```
MOV AL,05
MOV BL,10
MUL BL
```

Importante: Banderas Afectadas: CF, OF
"Operando" no puede ser un dato inmediato (Constante). La longitud de "Operando" determina si la operación se realizará en 8 o 16 bits.
8 BITS: MUL pone las banderas CF = 1 y OF =1 si AH <> 0 (Resultado > 255)
16 BITS: MUL pone las banderas CF = 1 y OF = 1 si DX <> 0 (Resultado > 65535)

IMUL

Sintaxis: IMUL operando
Descripción: IMUL multiplica dos enteros con signo.
Para **8 bits** realiza la operación **AX = AL * operando de 8 bits**
(Operando de 8 bits: registro o localidad de memoria BYTE PTR)
Para **16 bits** realiza la operación **DX:AX = AX * operando de 16 bits**
(Operando de 16 bits: AX, BX, CX, DX, SI, DI, BP, SP, WORD PTR)
Ejemplo: IMUL BL (Multiplicación con Signo de 8 bits AX = AL * BL)
Importante: Banderas Afectadas: CF, OF
"Operando" no puede ser un dato inmediato (Constante).
La longitud de "Operando" determina si la operación se realizará en 8 o 16 bits.
Si el resultado es positivo, las banderas se ponen igual que MUL
Si el resultado es negativo, CF = 1 y OF =1 si AH o DX <> todos 1s

DIV

Sintaxis: DIV operando
Descripción: DIV lleva a cabo una división sin signo del dividendo y produce un cociente y un remanente.
Para **8 bits** realiza la operación
AX / operando de 8 bits = AH (remanente) y AL (Cociente)
(Operando de 8 bits: registro o localidad de memoria BYTE PTR)
Para **16 bits** realiza la operación
DX:AX / operando de 16 bits = DX (remanente) y AX (Cociente)
(Operando de 16 bits: AX, BX, CX, DX, SI, DI, BP, SP, WORD PTR)
Ejemplo: DIV BH (AX / BH El cociente se pone en AL y el remanente en AH)
Importante: Ninguna de las banderas está definida después de una instrucción de División
"Operando" no puede ser un dato inmediato (Constante).
La longitud de "Operando" determina si la operación se realizará en 8 o 16 bits.

IDIV

Sintaxis: IDIV operando
Descripción: IDIV lleva a cabo una división con signo del dividendo y produce un cociente y un remanente. Igual que DIV, solo que IDIV toma en cuenta el signo de ambos operandos.
Para **8 bits** realiza la operación
AX / operando de 8 bits = AH (remanente) y AL (Cociente)
(Operando de 8 bits: registro o localidad de memoria BYTE PTR)
Para **16 bits** realiza la operación
DX:AX / operando de 16 bits = DX (remanente) y AX (Cociente)
(Operando de 16 bits: AX, BX, CX, DX, SI, DI, BP, SP, WORD PTR)
Ejemplo: IDIV BH (AX / BH El cociente se pone en AL y el remanente en AH)
Importante: Ninguna de las banderas está definida después de una instrucción de División
"Operando" no puede ser un dato inmediato (Constante).
La longitud de "Operando" determina si la operación se realizará en 8 o 16 bits.
Rango para el Cociente para 8 bits: +127 y –128
Rango para el Cociente para 16 bits: +32767 y –32768

INSTRUCCIONES LÓGICAS

AND, OR, XOR, NOT, TEST

Sintaxis y Descripción:

Instrucción	Operación realizada	El resultado es puesto en
AND op1, op2	Op1 AND Op2	op1
OR op1, op2	Op1 OR Op2	op1
XOR op1, op2	Op1 XOR Op2	op1
TEST op1, op2	Op1 AND Op2	(Resultado: Se ignora. PSW se actualiza)
NOT op1	Cambia 1s por 0s y 0s por 1s	op1

Estas instrucciones efectúan operaciones bit por bit sobre sus operandos de acuerdo a las siguientes tablas de verdad:

Op1	Op2	Op1 **AND** Op2	Op1 **OR** Op2	Op1 **XOR** Op2
0	**0**	0	0	0
0	**1**	0	1	1
1	**0**	0	1	1
1	**1**	1	1	0

Tabla de Verdad de "NOT"

Op1	**NOT** Op1
0	1
1	0

Importante:

- Estas instrucciones operan **BIT por BIT**.
- En estas instrucciones, **CF y OF** no tienen significado ya que no son operaciones aritméticas.
- **SF, ZF y PF** sí son afectadas y reflejan el resultado de la operación.
- La instrucción **NOT** no afecta ninguna de las banderas.
- **NOT** no puede tener direccionamiento inmediato (constante).
- La instrucción **TEST** hace lo mismo que la instrucción **AND**, excepto que no pone el resultado en ningún lado. Al igual que la instrucción **CMP** es utilizada solamente para poner banderas.
- En las instrucciones **AND, OR, XOR y TEST**, el **"Op1"** No puede ser una constante. No se pueden hacer operaciones sobre dos localidades de memoria. Las combinaciones permitidas para **Op1, Op2** serían
  - Registro, Registro
  - Registro, Memoria
  - Registro, Constante
  - Memoria, Registro
  - Memoria, Constante.
- La instrucción **TEST** funciona de la misma manera que CMP, la diferencia es que TEST generalmente se usa para comparar (verificar) un solo bit, mientras que CMP se usa para comparar un byte o palabra completos. Después de la ejecución de TEST, ZF = 1 si el bit a verificar es cero. ZF = 0 si el bit a verificar es Uno.
- En la instrucción **TEST**, generalmente Op2 es un valor inmediato, que es 1 para verificar el bit de mas a la derecha, 2 para el siguiente bit, 4 para el siguiente, etc.

Ejemplos: Suponer que AL = 1001 0110 en binario (AL = 96 Hexadecimal)
AH = 0101 1010 en binario (AH = 5A Hexadecimal)

Entonces, aplicando la tabla de verdad correspondiente a cada caso, y procediendo bit por bit:

La Instrucción **OR AL, AH** producirá AL = 1101 1110 (AL = DE Hexadecimal)
La Instrucción **AND AL, AH** producirá AL = 0001 0010 (AL = 12 Hexadecimal)
La Instrucción **XOR AL, AH** producirá AL = 1100 1100 (AL = CC Hexadecimal)

Ejemplo con TEST: Suponer que AL = 1001 0110 binario (AL = 96 Hexadecimal)
Para verificar si el primer bit de AL (bit mas significativo, de la izquierda, en la posición 2^7) es "1", utilizaríamos la instrucción **TEST AL,80** (Porque 80 Hexadecimal = 1000 0000 binario).
Lo cual haría un AND entre AL y 80 de la siguiente manera (ignorando el resultado, pero actualizando las banderas):

1001 0110 (Registro **AL**)
1000 0000 (Valor **80** Hexadecimal)
1000 0000 **Resultado** (TEST AL,80)
Como el resultado no fue cero, se pone **ZF = 0** lo que indica que el **primer bit de AL es "1"**

Aplicaciones de las instrucciones AND, OR, XOR:

Una aplicación común de AND es "**limpiar o apagar bits**" en un número binario a través de una "máscara" . La máscara es una secuencia de bits donde se pone "0" en la posición de los bits que se desean "limpiar o apagar" (es decir, ponerlos en cero) y "1" en la posición donde se desea que la información siga igual. Ejemplo: Para Limpiar los primeros 4 bits de un número binario:

XXXX XXXX Cualquier Número binario **(Op1)**
0000 1111 Máscara (ceros donde se va a limpiar, unos donde se queda igual) **(Op2)**
0000 XXXX Resultado: **Op1 AND Op2**

Una aplicación común de OR es "**prender bits**" en un número binario a través de una "máscara" . La máscara es una secuencia de bits donde se pone "1" en la posición de los bits que se desean "prender" (es decir, convertirlos en uno), y "0" en la posición donde se desea que la información permanezca igual. Ejemplo: Para prender los últimos 4 bits de un número binario:

XXXX XXXX Cualquier Número binario **(Op1)**
0000 1111 Máscara (unos donde se va prender, ceros donde se queda igual) **(Op2)**
XXXX 1111 Resultado: **Op1 OR Op2**

Una aplicación común de XOR es **"invertir selectivamente"** bits en un número binario a través de una "mascara". La máscara es una secuencia de bits donde se pone "1" en la posición de los bits que se desean "invertir" (es decir, si el bit es 0 cambiarlo a 1 y viceversa), y "0" en la posición donde se desea que la información permanezca igual. Ejemplo: Para invertir los últimos 4 bits de un número binario:

XXXX XXXX Cualquier Número binario **(Op1)**
0000 1111 Máscara (unos donde se van a invertir, ceros donde se queda igual) **(Op2)**
XXXX NOT(XXXX) Resultado: **Op1 XOR Op2**

CORRIMIENTOS Y ROTACIONES

Sintaxis:

[**Tipo de Corrimiento**] **Operando**, **Cuantos**
[**Tipo de Rotación**] **Operando**, **Cuantos**

- ✓ [**Tipo de Corrimiento**] Puede ser:
  - **SHL** (Shift Logical Left)
  - **SAL** (Shift Arithmetic Left)
  - **SHR** (Shift Logical Right)
  - **SAR** (Shift Arithmetic Right)
- ✓ [**Tipo de Rotación**] Puede ser:
  - **ROL** (Rotate Left)
  - **ROR** (Rotate Right)
  - **RCL** (Rotate Left Thru Carry)
  - **RCR** (Rotate Right Thru Carry)

- ✓ **"Operando"** puede tener cualquier modo de direccionamiento excepto el inmediato.
- ✓ **"Cuantos"** debe ser **1**, o el registro **"CL"** si se harán varios movimientos.

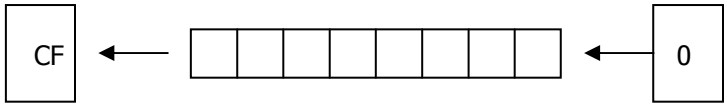
Importante:

- Una operación de corrimiento mueve todos los bits en un área de datos ya sea a la derecha o a la izquierda.
- En una operación de Rotación, el bit que es desplazado fuera llena el espacio vacante en el "Operando" y también se copia en la bandera de acarreo.
- Corrimientos y Rotaciones operan sobre los bits contenidos en los bytes o palabras.
- El operando puede ser un registro o una localidad de memoria.
- Un corrimiento a la izquierda equivale a multiplicar por 2.
- Un corrimiento a la derecha equivale a dividir entre 2.
- La OF es indefinida para corrimientos mayores de uno.
- La OF = 1 si el bit de signo cambia en un corrimiento/rotación de uno en uno.
- PF, SF y ZF son afectadas por corrimientos. PF, SF y ZF No son afectadas por rotaciones.

SAL Y SHL

Los corrimientos hacia la izquierda mueven los bits a la izquierda en el operando. SHL y SAL son idénticos en su operación. El bit desplazado fuera del registro brinca la bandera de acarreo.

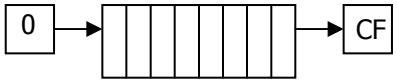
SHL: desplazamiento lógico a la izq.
SAL: desplazamiento aritmético a la izq



SAR Y SHR

Los corrimientos hacia la derecha mueven los bits hacia la derecha en el operando. El bit recorrido fuera del registro mete la bandera de acarreo. SAR difiere de SHR en que SAR utiliza el bit de signo para llenar el bit vacante de mas a la izquierda. De esta manera, los valores positivos y negativos retienen sus signos.

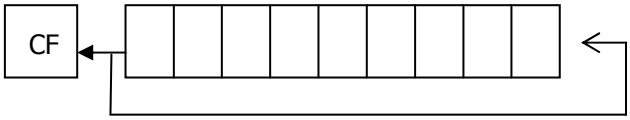
SHR: Shift Logical Right = Desplazamiento lógico a la derecha



SAR: Shift Arithmetic Right = Desplazamiento aritmético a la derecha

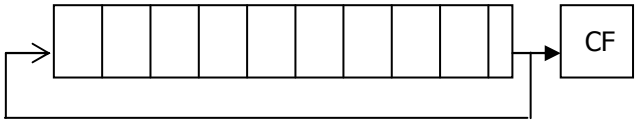
ROL

ROL: Rotate Left = Rotación a la Izquierda



ROR

ROR: Rotate Right = Rotación a la derecha

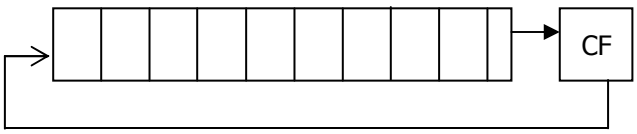


RCL

RCL: Rotate Left Thru Carry
= Rotación a la izquierda con Acarreo

RCR

RCR : Rotate Right Thru Carry
= Rotación a la derecha con Acarreo



INSTRUCCIONES DE TRANSFERENCIA DE CONTROL

TRANSFERENCIA NO CONDICIONADA

Las instrucciones de transferencia de control no condicionadas son usadas para mandar la secuencia de ejecución del programa a una diferente sección del programa sin evaluar ninguna condición. (Instrucciones JMP, CALL, RET)

JMP

Sintaxis: **JMP** operando

Descripción: Transfiere el control a la localidad especificada en "operando" sin guardar la dirección de retorno. Una instrucción JMP modifica el valor del registro IP (apuntador de instrucciones).

Ejemplos: **JMP 200** y **JMP [200]** son distintas.
En el caso de JMP 200, la secuencia del programa se pasa a la localidad 200.
En el caso de JMP [200], la secuencia del programa se pasa a la localidad contenida en la localidad 200.

JMP AX
En el caso de JMP AX, la secuencia del programa se pasa a la localidad contenida en el registro AX.

Importante:

- Si "**operando**" es inmediato (sin corchetes), se trata de un JMP directo, es decir, el control se pasa directamente a esa dirección especificada.
- Si "**operando**" es una dirección de memoria (Con corchetes), se trata de un JMP indirecto, es decir, la secuencia se pasa a la localidad indicada (contenida) en la localidad con corchetes.
- Si "**operando**" es un registro (AX, BX, CX, DX, BP, SP, SI, DI), la secuencia del programa se pasa a la localidad contenida en dicho registro, por lo que también se considera un JMP indirecto.

CALL, RET

Sintaxis: **CALL** operando
RET

Descripción: **CALL** es el equivalente a una llamada de subrutina. CALL Guarda la dirección de retorno, la cual retoma cuando una instrucción RET se ejecuta.
RET es la instrucción de Retorno de Subrutina, y pasa el control del programa a la siguiente línea de programa después del CALL que invocó esa subrutina.

Ejemplos:

```
100:  CALL 300
      INT 20
...
300:  ADD AX,BX
      MOV [2000],AX
      RET
```

Importante:

- Si "**operando**" es inmediato (sin corchetes), se trata de un CALL directo, es decir, el control se pasa directamente a esa dirección especificada.
- Si "**operando**" es una dirección de memoria (Con corchetes), se trata de un CALL indirecto, es decir, la secuencia se pasa a la localidad indicada (contenida) en la localidad con corchetes.
- Si "**operando**" es un registro (AX, BX, CX, DX, BP, SP, SI, DI), la secuencia del programa se pasa a la localidad contenida en dicho registro, por lo que también se considera un CALL indirecto.

TRANSFERENCIA CONDICIONAL (Jumps Condicionales)

Una instrucción de transferencia condicionada prueba el estado actual de la computadora para determinar si transfiere o no el control del programa. Para este propósito se examina (prueba) el resultado de una operación aritmética o lógica previa. Estas instrucciones son conocidas también como JUMPS Condicionales.

Todos los JUMPS condicionales tienen un desplazamiento de un byte, esto quiere decir que solo pueden "brincar" como máximo hasta 128 bytes. Si un JUMP condicional debe hacerse a una localidad a mas de 128 bytes, debe usarse una estrategia diferente. Por ejemplo, supongamos que el programa en la localidad 108 necesita brincar a la localidad 8000 si la bandera de cero ZF = 1. En este caso no es posible usar la instrucción JZ 8000 pues el brinco es mayor de 128 bytes; El siguiente código soluciona este problema.

CODIGO CORRECTO	**CODIGO INCORRECTO**
0106: CMP AX,BX	0106: CMP AX,BX
0108: **JNZ 010D**	0108: **JZ 8000**
010A: **JMP 8000**	010A: INT 20
010D: INT 20	

En la tabla anexa se encuentran descritas todas las instrucciones de salto condicional, su formato, y las pruebas que realizan para transferir el control.

Importante: Ninguna Bandera se afecta con un salto condicional.

CICLOS (Loop)

Sintaxis: **LOOP** localidad

- "Localidad" debe ser un dato inmediato. No se aceptan direccionamientos indirectos.
- "Localidad" debe indicar una dirección menor (mas arriba en código) que la dirección de la línea que contiene el Loop.

Descripción: La Instrucción LOOP permite ejecutar un bloque de instrucciones un número finito de veces, el cual debe ser especificado previamente en CX. Loop trabaja siempre implícitamente con el registro CX. Ninguna instrucción de ciclos afecta banderas.

Ejemplo: En el siguiente programa, se realiza un ciclo que se repite 10 veces, y permite calcular la suma A+9+8+7+6+5+4+3+2+1 = 37. Nótese que cada vez que se ejecuta la instrucción LOOP, CX se decrementa en 1 y se realiza la condición implícita de que CX <> 0, es decir, LOOP regresa el control a la linea 106 mientras CX <> 0. Cuando la condición de que CX = 0 se cumple, se transfiere el control a la línea inmediata debajo de LOOP.

```
0100: MOV    AX,0000
0103: MOV    CX,000A
0106: ADD    AX,CX
0108: LOOP   0106
010A: MOV    [2000],AX
010D: INT    20
```

Resumen de LOOP y Otras instrucciones de Lazo:

Nombre	**Instrucción**	**Alternativa**	**Condición**
Loop	**LOOP** localidad		CX <> 0
Loop while Zero or Equal	**LOOPZ** localidad	**LOOPE** localidad	ZF = 1 y CX <> 0
Loop while NonZero or Not Equal	**LOOPNZ** localidad	**LOOPNE**	ZF = 0 y CX <> 0
Jump if CX = 0	**JCXZ** localidad		CX = 0

Ejemplo con **LOOPNZ**:

```
0100:  MOV     CX,0005 ;Numero de veces a ejecutar el ciclo
0103:  MOV     SI,2000 ;Puntero para recorrer las localidades para buscar "FF"
0106:  MOV     AL,FF   ; Valor a Buscar "FF"
0108:  INC     SI      ;Incrementa la localidad a buscar
0109:  CMP     AL,[SI] ;Compara el FF con la localidad actual
010B:  LOOPNZ  0108    ;Repite mientras no se encuentre "FF" y CX<>0
010D:  JNZ     0150    ;Salto para decir que no lo encontró
010F:  INT     20      ;Si lo encontró, Termina el proceso
```

En este programa, se realiza un ciclo para ejecutarse 5 veces, y buscar en las localidades de memoria 2001 a la 2006 un valor "FF" . El ciclo se repetirá mientras NO se encuentre un "FF" y además CX <> 0. Si una de estas dos condiciones no se cumple, la secuencia pasa a la dirección 10D (Inmediata después de LOOPNZ).

Ejemplo con **TEST, JNZ, LOOP**:

```
0100:   MOV     CX,000A
0103:   TEST    CX,0001
0107:   JNZ     010B
0109:   ADD     AX,CX
010B:   LOOP    0103
010D:   MOV     [2000],AX
0110:   INT     20
```

En este programa se realiza la suma de los números pares hexadecimales entre 1 y A.
El resultado se pasa a la localidad 2000 antes de terminar el programa.
Nótese que la suma se realiza de la siguiente manera: A + 8 + 6 + 4 +2 = 1E
En la línea 103 se checa si es número Impar, de ser así, no se suma, solo se regresa al ciclo; si es par, se pasa a la línea 109 donde se incrementa el número actual al registro AX.