3. CONJUNTO DE INSTRUCCIONES DEL PROCESADOR 8086/8088

Todo procesador, por muy pequeño o grande que sea, tiene un lenguaje propio, el cual puede ser único y donde dicho procesador puede reconocer y ejecutar el cual consta de un conjunto de instrucciones. A este conjunto de instrucciones se le ha denominado Lenguaje de Máquina y el cual pertenece al microprocesador. Este lenguaje se representa como un conjunto de números, en binario, que pueden realizar las operaciones que se llevan a cabo en el microprocesador a través de su circuitería, en un ciclo de instrucción. Estas instrucciones, están grabadas en el hardware y no pueden ser cambiadas. Un ejemplo de este tipo de lenguaje es el lenguaje Ensamblador.

El lenguaje Ensamblador, es un lenguaje de bajo nivel y con el cual se puede controlar completamente las funciones del microprocesador, así como la de sus periféricos de manera directa y con un alto rendimiento. Es la directa más cercana al lenguaje de máquina para un microprocesador en específico.

A pesar de que la programación en un lenguaje de alto nivel puede ser más simple, programar en lenguaje Ensamblador tiene algunas ventajas:

- Brinda un mayor control sobre el hardware.
- Genera código más pequeño.
- Tiene una ejecución más rápida.

En particular en este capítulo se explicará el conjunto de instrucciones para los microprocesadores de la familia INTEL.

Para iniciar con el conjunto de instrucciones a revisar, se considera que las instrucciones del procesador tienen la siguiente sintaxis:

Mnemónico operando1 [,operando2]

Donde: **Mnemónico** es un conjunto de caracteres que identifica a la instrucción, generalmente es la abreviación de la acción que se quiere realizar.

El operando puede ser:

reg	registro de 8 o 16 bits (no incluidos los registros segmento y los índices)
regp1	registro de propósito general de 16 bits
regseg	registro segmento
regind	registro índice
mem	localidad de memoria [dir], byte [reg], word [reg]
dato	operando de 8 o 16 bits
dir	dirección de 16 bits
con	contador que puede ser 1, CL o CX
[ ]	contenido de

Además, se considera que las instrucciones se clasifican en: instrucciones de transferencia de datos, aritméticas y lógicas, rotación y corrimiento, transferencia de programa, ciclos y manejo de cadenas, como a continuación se describen.

3.1. INSTRUCCIONES DE TRANSFERENCIA DE DATOS

Las instrucciones de transferencia de datos, como su nombre lo indica, sirven para transferir datos de un origen a un destino.

MOV (move). Realiza el movimiento (copia) de la información. MOV copia la información de origen al destino. El valor del origen no resulta afectado. Las banderas no resultan afectadas, a menos que se realice el movimiento al registro banderas.

Sintaxis: **MOV** destino, origen

a) MOV reg1,
mem/reg2/regind/regseg/datoo
b) MOV mem, reg/regseg/regind
c) MOV regseg, mem/regind/regseg/reg1

Ejemplo:

```
MOV AX, BX
MOV CH, [210]
MOV AX, [200]
MOV [300], BX
MOV [SI], BX
```

LEA Carga un registro de 16 bits con una dirección específica. Transfiere el desplazamiento de operando fuente al operando destino. La fuente puede ser una referencia a memoria y el destino debe ser un registro de 16 bits. Las banderas no se ven afectadas.

Sintaxis: **LEA** reg, dir

Ejemplo:

```
LEA BX, [200]   ; BX←200
MOV AX, 250     ;es equivalente a LEA AX, [250]
```

XCHG Intercambia el contenido de sus operandos y modifica los operandos, a menos que éstos tengan el mismo valor. Las banderas no se ven afectadas.

Sintaxis: **XCHG** oper1, oper2

a) XCHG reg/regind, reg/mem/regind
b) XCHG mem, reg/regind

Ejemplo:

```
XCHG AX, CX
XCHG [200], SI
XCHG AH, AL
```

PUSH Coloca en el tope del stack el contenido de un registro de 16 bits. Almacena el valor de una dirección de RAM interna en el stack. En primer lugar, incrementa el valor de SP, y después guarda el valor de la dirección de RAM, en la posición apuntada por el SP ya incrementado. Las banderas no se ven afectadas.

Sintaxis: **PUSH** regp1/regseg/mem/regind

Ejemplo:

```
PUSH AX
PUSH CS
PUSH [SI]
```

POP Coloca en algún registro de 16 bits el contenido del stack. Copia el contenido de la posición de RAM direccionado por el SP, en la dirección de RAM que indica el segundo byte de la instrucción. El valor del SP se decrementa. Las banderas no se ven afectadas.

Sintaxis: **POP** regp1/regseg/regind/mem

Ejemplo:

```
POP AX POP SI
POP WORD[200]
```

NOTA: El orden de salida con POP, debe ser el orden inverso que en el de entrada con

PUSH. **PUSHA** Coloca en el stack, los registros de propósito general, los apuntadores y

los índice, en
el orden: AX, CX, DX, BX, SP, BP, SI y DI. Las banderas que se ven alteradas son la del overflow, dirección, trap, interrupción, signo, cero, carry auxiliar, paridad y carry.

Sintaxis: **PUSHA**

POPA Coloca el contenido del stack en los registros de propósito general, los apuntadores y los índices, en el orden: DI, SI, BP, SP, BX, DX, CX y AX. Las banderas que se ven alteradas son la del overflow, dirección, trap, interrupción, signo, cero, carry auxiliar, paridad y carry.

Sintaxis:
POPA

PUSHF Coloca el contenido del registro de banderas en el stack (primero la parte alta y después la parte baja). Las banderas no se ven afectadas.

Sintaxis: **PUSHF**

POPF Coloca el contenido del stack en el registro de banderas (el primer byte del stack en la parte baja y el Segundo byte en la parte alta). Las banderas cambian de acuerdo con los valores que se recuperan del stack.

Sintaxis: **POPF**

LAHF Coloca las banderas del orden bajo en el registro AH. El registro banderas no cambia de valor con esta instrucción. Las banderas no se ven afectadas.

Sintaxis:
LAHF

SAHF Coloca el contenido de AH en la parte baja del registro de banderas. Se remplazan los valores previos del registro banderas. Las banderas de orden bajo toman los valores que se coloque en AH.

Sintaxis:
SAHF

3.2. OPERACIONES ARITMÉTICAS Y LÓGICAS

Este conjunto de instrucciones es ejecutado por la Unidad aritmeticológica (ALU) y pueden modificar el registro de banderas.

ADD Realiza la suma entre dos operandos, el resultado se almacena en el destino. Esta instrucción puede modificar las banderas de signo, cero, paridad, carry, carry auxiliar y overflow.

Sintaxis: **ADD** destino, origen

a) **ADD reg/regind, reg/mem/regind/dato**
b) **ADD mem, reg/regind**

Ejemplo:

```
ADD AX, BX
ADD [200], AH
ADD SI, [BX]
```

ADC Realiza la suma entre dos operandos y el carry (ADD op1+op2+cy), si el carry está apagado, la operación es igual a la de ADD. Esta instrucción puede modificar las banderas de signo, cero, paridad, carry, carry auxiliar y overflow.

Sintaxis: **ADC** oper1, oper2

a) **ADC reg/regind, reg/mem/regind/dato**
b) **ADC mem, reg/regind**

Ejemplo:

```
ADC BH, F0
```

SUB Realiza la resta entre dos operandos, y el resultado se almacena en el destino, la sintaxis es equivalente a la del ADD. Esta instrucción puede modificar las banderas de signo, cero, paridad, carry, carry auxiliar y overflow.

Sintaxis: **SUB** destino, origen

Ejemplo:

```
SUB AX, BX
SUB [BX], CX
```

SBB. Realiza la resta entre dos operandos y el carry (op1-op2-cy), si el carry está apagado la operación es igual a la de SUB, la sintaxis es equivalente a la del ADD. Esta instrucción puede modificar las banderas de signo, cero, paridad, carry, carry auxiliar y overflow.

Sintaxis: **SBB** oper1, oper2

Ejemplo:

```
SBB BH, CL
SBB CX, [SI]
```

INC Incrementa en 1 el valor del operando. Si el valor a incrementar es 0FFH, entonces el resultado será 0, aunque la bandera del carry no resulta afectada por ello. Esta instrucción puede modificar las banderas de signo, cero, paridad, carry auxiliar y overflow.

Sintaxis: **INC** reg/regind/mem

Ejemplo:

```
INC CX
INC SI
INC BYTE[250]
INC WORD[200]
```

DEC Decrementa en uno el operando. Si el valor a decrementar es 0, entonces el resultado será

0xFF, aunque la bandera del carry no resulta afectada por ello, la sintaxis es equivalente a la de la instrucción INC. Esta instrucción puede modificar las banderas de signo, cero, paridad, carry auxiliar y overflow.

Sintaxis: **DEC** reg/regind/mem

Ejemplo:

```
DEC
WORD[20
0] DEC BX
```

NEG Obtiene el complemento a 2 del operando. Si el operando es cero, la bandera de carry, se limpia, en cualquier otro caso la bandera de carry se pone en uno. Esta instrucción puede modificar las banderas de signo, cero, paridad, carry, carry auxiliar y overflow.

Sintaxis: **NEG** reg/regind/mem

Ejemplo:

```
NEG AX
NEG BYTE[200]
```

MUL Multiplica el contenido del registro acumulador, con el contenido del registro dado en el operando. Esta multiplicación se hace sin considerar el signo. Las banderas que pueden llegar a modificarse son overflow y carry.

Sintaxis: **MUL** reg/regind/mem

La multiplicación se lleva a cabo de la siguiente forma:

1) AX ← AL * reg (si el registro es de 8 bits)
2) DXAX ← AX * reg (si el registro es de 16 bits)

Ejemplo:

```
MUL BH
MUL WORD[SI]
MUL CX
```

IMUL Multiplica el contenido del registro acumulador, con el contenido del registro dado considerando el signo, la sintaxis y semántica es equivalente a la del MUL. El valor de las banderas es indefinido.

Sintaxis: **IMUL** reg/regind/mem

Ejemplo:

```
IMUL DL
IMUL BYTE[200]
```

DIV Divide un operando que está en AX o en DX:AX entre su único argumento, esta división se considera sin signo. El valor de las banderas es indefinido.

Sintaxis: **DIV** oper

Ejemplo:

a) DIV reg_8bits **AL← AX/reg_8bits y en AH← residuo**
b) DIV reg_16bits **AX←DX:AX/reg_16bits y en DX ←residuo**

```
DIV BH
DIV BX
```

CMP Realiza la comparación entre dos operandos, esta instrucción puede modificar el registro de banderas. La comparación se realiza mediante la diferencia del registro fuente desde destino, sin modificar ninguno de los registros durante la comparación. Esta instrucción puede modificar las banderas de signo, cero, paridad, carry, carry auxiliar y overflow.

Sintaxis: **CMP** oper1, oper2

a) **CMP reg/mem/regind, reg/dato**
b) **CMP mem, reg/dato/regind**

Ejemplo:

```
CMP BYTE[200], 6
CMP AH, 7              ; (ah – 7)
CMP [CX], BX
CMP AX, BX
```

NOT Niega el valor del operando. Obtiene el complemento a 1 del operando. Las banderas no son modificadas.

Sintaxis: **NOT** reg/regind/mem

Ejemplo: NOT AX

```
NOT WORD[SI]
```

AND Realiza la operación AND entre dos operandos bit a bit, almacena el resultado en el registro destino. AND devuelve uno si y sólo si ambos bits son uno, en caso contrario devuelve cero. Las banderas de overflow (OF)y carry (CF) se limpian.

Sintaxis: **AND** oper1, oper2

a) **AND reg/regind, reg/mem/regind/dato**
b) **AND mem, reg/regind**

Ejemplo:

```
AND AX, [SI]
AND DL, BH
AND CX, 0001
AND BL, [450]
```

OR Realiza la operación OR inclusivo entre dos operandos bit a bit, devuelve cero si y sólo si ambos bits son cero, en caso contrario devuelve uno, su sintaxis es equivalente a la de la instrucción AND. Las banderas de overflow y carry se limpian.

Sintaxis: **OR** oper1, oper2

c) **OR reg/regind, reg/mem/regind/dato**
d) **OR mem, reg/regind**

Ejemplo:

```
OR AX, BX
OR SI, [DI]
```

XOR Obtiene el OR exclusivo de dos operandos realizando la operación bit a bit, donde devuelve cero si ambos valores son iguales, y uno si los valores son diferentes, su sintaxis es equivalente a la del AND. Las banderas de overflow y carry se limpian.

Sintaxis: **XOR** oper1, oper2

e) **XOR reg/regind, reg/mem/regind/dato**
f) **XOR mem, reg/regind**

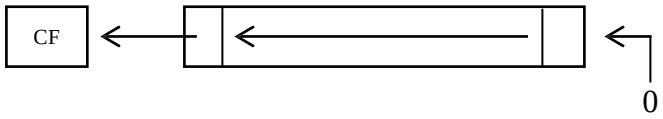
Ejemplo:

```
XOR AH, CL
XOR [100], SI
```

3.3. ROTACIONES Y CORRIMIENTOS

Este conjunto de instrucciones son parte de las operaciones que se pueden realizar de manera interna en la computadora, realizando rotaciones de sus bits, o bien corrimientos de estos, los cuales pueden ser sin signo o bien con signo.

SHL Realiza el corrimiento de los valores (en bits) del registro destino hacia la izquierda por el número de posiciones de bits que se especifican en el segundo operando. Se introducen ceros desde su extremo derecho del destino, y se desplazan a la izquierda. El indicador de acarreo se actualiza para que coincida con el último bit desplazado fuera del extremo izquierdo. Las banderas que se modifican son la de overflow, signo, cero, paridad y carry.



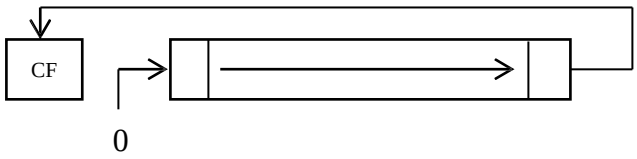
Sintaxis: **SHL** oper1, opr2

a) SHL reg/mem, CL/01

Ejemplo:

```
SHL AX, CL
SHL BX, 01
```

SHR. Desplaza los bits a la derecha por el número de posiciones especificadas en el número del segundo operando. Los 0‟s se desplazan a la izquierda. Si el bit de signo conserva su valor original, la bandera de overflow se limpia. Las banderas que se ven modificadas son la de overflow, signo, cero, paridad y carry.



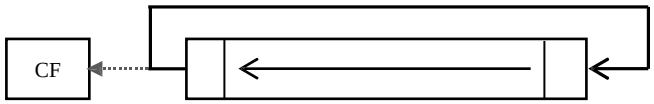
Sintaxis: **SHR** oper1, opr2

a) SHR reg/mem, CL/01

Ejemplo:

```
SHR BX, CL
SHR DX, 01
```

ROL Cambia la palabra o byte en el destino hacia la izquierda por el número de posiciones de bits especificado en el segundo operando. Como los bits se transfieren de su extremo izquierdo (orden superior) del destino, vuelven a entrar a la derecha (orden inferior). El indicador de acarreo se actualiza para que coincida con el último bit desplazado fuera del extremo izquierdo. Las banderas que se ven modificadas son: overflow y carry.



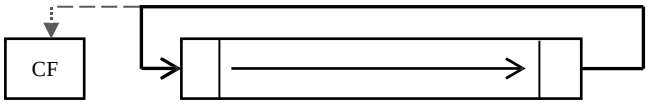
Sintaxis: **ROL** oper1, opr2

a) ROL reg/mem, CL/01

Ejemplo:

```
ROL BX, CL
ROL AX, 01
```

ROR Cambia la palabra o byte en el destino hacia la derecha por el número de posiciones de bit que se especifican en el segundo operando. Como los bits son transferidos a la derecha (orden bajo) final del destino, vuelven a entrar a la izquierda (orden superior) final. El indicador de acarreo se actualiza para que coincida con el último bit desplazado fuera del extremo derecho. Las banderas que se ven modificadas son: overflow y carry.



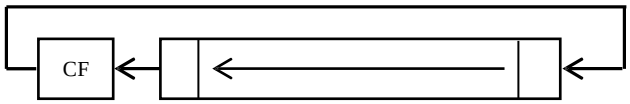
Sintaxis: **ROR** oper1, opr2

a) ROR reg/mem, CL/01

Ejemplo:

```
ROR DX, CL
ROR BX, 01
```

RCL Desplaza la palabra o byte en el destino a la izquierda por el número de posiciones de bit especificados en el segundo operando. El bit de más a la izquierda (orden superior) final del destino entra en la bandera de acarreo, y la bandera de acarreo desplaza gira alrededor para entrar en el bit desocupado más a la derecha poco la posición del destino. Esta "rotación de bit", continúa el número de veces especificado en operando. (Otra forma de ver esto es considerar la bandera de acarreo en el bit de orden más elevado de la palabra en rotación.)



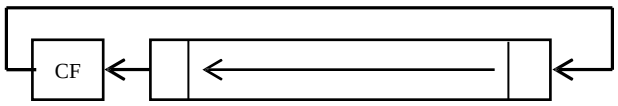
Sintaxis: **RCL** oper1, opr2

a) RCL reg/mem, CL/01

Ejemplo:

```
RCL DX, CL
RCL BX, 01
```

RCR Cambia la palabra o byte en el destino hacia la derecha por el número de posiciones de bit que se especifican en el segundo operando. El bit de más a la derecha (orden bajo) final del destino entra en la bandera de acarreo, y la bandera de acarreo desplazados gira alrededor para entrar en la posición vacante dejado más bits de destino. Esta "rotación de bit", continúa el número de veces especificado en el operando. (Otra forma de ver esto es considerar la bandera de acarreo en el bit de orden más bajo de la palabra en rotación.)



Sintaxis: **RCR** oper1, opr2

a) RCR reg/mem, CL/01

Ejemplo:

```
RCR AX, CL
RCR DX, 01
```

3.4. TRANSFERENCIA DE PROGRAMA

Las instrucciones de transferencia de programa son aquellas instrucciones que sirven para modificar el orden en que se llevan a cabo las instrucciones, unas refieren a saltos y otras a llamados a procedimientos.

Procedimientos ***(subrutinas)****:* Una *subrutina* es un conjunto de instrucciones que llevan a cabo una tarea específica y puede ser utilizado (llamado) por otro programa en el momento en que se requiera.

En el 8086/88 se tienen dos clases de subrutinas:

- ***Cercanas*** ***(near).*** Son aquellas que se encuentran definidas en el mismo segmento que el programa que los llama.
- ***Lejanas*** ***(far).*** Son aquellas que se encuentran definidas en un segmento distinto al programa que las llama. Por lo tanto, existen dos tipos de llamados y dos tipos de regreso.

  Los llamados son:

  a) **CALL NEAR** y
  b) **CALL FAR**

  Los retornos son:

  a) RET NEAR y
  b) RET FAR

JMP Realiza un ***salto incondicional***, esto es, cambia la secuencia de ejecución. Al registro IP
se le asigna la dirección del argumento de la instrucción JMP.

Sintaxis: **JMP** oper

a) JMP dir/regp1/regind/mem

Ejemplo:

```
JMP 120
JMP [BX]
JMP SI
```

J*xy* Realiza un ***salto condicional***. (Se verá en la sección correspondiente a Manejo de

Banderas) **CALL** Realiza llamado a subprograma o subrutina, cambia temporalmente la

secuencia de
ejecución para llevar a cabo la tarea de la subrutina.

Sintaxis: **CALL** dir

Ejemplo:

```
CALL 100
```

RET Realiza retorno de subrutina. Esto es regresa al lugar donde la rutina fue llamada.

Sintaxis: **RET**

Ejemplo:

```
        CALL lee
        MOV BH, AL
            …
```

Donde lee es:

```
lee: MOV  AH, 1
     INT 21h
     RET
```

3.5. MANEJO DE BANDERAS

Las banderas del procesador indican el estado de este y del resultado de las operaciones, por medio de 0 (apagadas) o 1 (encendidas). Este estado puede ser verificado para la realización de una tarea determinada y puede ser modificado de manera directa, utilizando instrucciones.

J*xy* Realiza un ***salto condicional***, esto es, cambia la secuencia de ejecución bajo el cumplimiento de una determinada condición. Al registro IP se le asigna la dirección del argumento.

Sintaxis: **Jxy**

dir/regp1/regind/mem

JA *o* ***JNBE*** Salta si la bandera del carry (CF) es 0 (no carry) **y** la bandera de cero (ZF) es 0 (no cero).

JAE *o* ***JNB*** Salta si la bandera del carry (CF) es 0 (no carry).

JBE *o* ***JNA*** Salta si la bandera del carry (CF) es 1 **o** la bandera de carry auxiliar (AF) es 1.

JB *o* ***JNAE*** Salta si la bandera del carry (CF) es 1.

JC Salta si la bandera del carry (CF) es 1 (idéntico a **JB** o **JNAE**)

JCXZ Salta si el registro CX es cero.

JE *o* ***JZ*** Salta si la bandera de cero (ZF) es 1.

JG *o* ***JNLE*** Salta si la bandera de cero (ZF) es 0 y la bandera de signo (SF) es igual a la bandera de overflow (OF) (ambas 0 o ambas 1). Salta si es mayor o si no es menor o igual.

JGE *o* ***JNL*** Salta si la bandera del signo (SF) es igual a la de overflow (OF) (ambas 0 o ambas 1). Salta si es mayor o igual o si no es menor.

JL o JNGE Salta si la bandera del signo (SF) es diferente a la bandera de overflow (OF). Salta si es menor o salta si es mayor o igual.

JLE o JNG Salta si la bandera del cero (ZF) es 1 o si la bandera del signo (SF) es diferente a la bandera de overflow (OF). Salta si es menor o igual o salta si no es mayor que.

JNC Salta si la bandera del carry (CF) es 0 (no carry) (idéntico a **JAE** o **JNB**)

JNE o JNZ Salta si la bandera de cero (ZF) es igual a 0. Salta si no es igual o salta si no es cero.

JNO *Salta si la bandera del overflow (OF) es 0 (no overflow).*

JNP o JPO Salta si la bandera de paridad (PF) es 0 (o paridad impar).

JNS Salta si la bandera de signo (SF) es 0 (positivo).

JO Salta si la bandera del overflow (OF) es 1 (overflow puesto).

JP o JPE Salta si la bandera de paridad (PF) es 1 (paridad par).

JS Salta si la bandera de signo (SF) es 1 (negativo).

3.6. INSTRUCCIONES PARA MODIFICACIÓN DE BANDERAS

CLC Limpia la bandera de carry (CF = 0).

Sintaxis: **CLC**

STC Pone en 1 en la bandera de carrey (CF = 1)

Sintaxis: **STC**

CLD Limpia la bandera de dirección (DF = 0)

Sintaxis: **CLD**

STD. Pone 1 en la bandera de dirección (DF = 1)

Sintaxis: **STD**

CLI Limpia la bandera de interrupción (IF = 0)

Sintaxis: **CLI**

STI Pone 1 en la bandera de interrupción (IF = 1)

Sintaxis: **STI**

CMC Complementa la bandera de acarreo

Sintaxis: **CMC**

3.6. INSTRUCCIONES DE CICLOS

LOOP Repite la ejecución de un bloque de instrucciones CX veces, por lo tanto, es necesario un valor inicial en el registro CX. En cada iteración del ciclo (LOOP) de forma automática decrementa en 1 al registro CX. El ciclo termina hasta que CX es cero.

Sintaxis. **LOOP dirección**

Ejemplo:

```
        MOV AX,0
        MOV CX,10
ciclo:  INC AX
        LOOP ciclo
```

3.7. INSTRUCCIONES PARA MANEJO DE CADENAS

MOVSB/W Copia el byte o Word apuntado por DS:SI dentro de la localidad apuntada por ES:DI. Si DF=0 se incrementa, sino se decrementa.

REP Es un prefijo que puede ser utilizado antes de cualquier instrucción de cadena (CMPS, LODS, MOVS, SCAS, STOS). REP repite CX veces, es decir, mientras CX sea diferente de cero (C F) es 0, después de ejecutar la instrucción de cadena.

Se revisa si CX=0 antes de ejecutar la instrucción, la bandera de cero (C F) es verificada después de ejecutar la instrucción de cadena.

REP = **REPE** = **REPZ**

REPNE ; Mientras no sea igual.

Ejemplo: Mover 10 bytes de la dirección 150 a la dirección 170

```
CLD
LEA SI,[150]
LEA DI,[170]
MOV CX,10
REP MOVSB
```

SCASB/W Compara el acumulador (AL o AX) con el contenido de lo apuntado por DI.

Ejemplo: Busca en un bloque de 100 bytes la letra „A‟, pone la bandera de cero a 1 si encuentra al menos una, 0 en caso contrario.

```
        MOV AX, DS          ; esto no es necesario en debug
        MOV ES, AX          ; esto no es necesario en debug
        CLD
        MOV AL, “A‟
        MOV CX,100
        LEA DI, [150]
        REPNE SCASB
        JE encontrada
        MOV DL, 0
        JMP fin
encontrada: MOV DL,1
fin:    NOP
```

STOSB/W Copia lo contenido en AL o AX en ES:DI.

Ejemplo: Iniciar un buffer a cero.

```
MOV AX, DS
MOV ES, AX
MOV AL, 0
LEA DI,
[150]
MOV CX,
100
CLD
REPE STOSB
```

LODSB/W. Transfiere el valor apuntado por DS:SI dentro de AX o AL.

Ejemplo: Imprimir 20 caracteres de un buffer.

```
        MOV CX, 20
        LES SI, [150]
ciclo: LODSB
        MOV DL, AL
        MOV AH, 02
        INT 21
        LOOP ciclo
```

CMPSB/W Compara DS:SI con DS:DI

Ejemplo: Compara dos buffers de 10 elementos.

```
            CLD
            MOV CX, 10
            LEA SI, [150]
            LEA DI, [160]
            REPE CMPSB
            JNE
            diferentes
            MOV DL,1     ; iguales
            JMP fin
diferentes: MOV DL,0
      fin:  NOP
```