Instrucciones Básicas

CI-0118 Lenguaje Ensamblador
GR-01
I ciclo 2020
Dr. Carlos Vargas
ECCI-UCR

Registros accesibles al programador

Registros de Propósito General

64 bits	32 bits	16 bits	8 bits	8 bits
RAX	EAX	AX	AH	AL
RBX	EBX	BX	BH	BL
RCX	ECX	CX	CH	CL
RDX	EDX	DX	DH	DL
RSI	ESI	SI	No disponible	
RDI	EDI	DI	No disponible	
RSP	ESP	SP	No disponible	
RBP	EBP	BP	No disponible	

Variables

1) Haga un diagrama del espacio de memoria asignado e inicializado por medio de las siguientes declaraciones:

```
.DATA
NUM1        DB      3
NUM2        DW      3
NUM3        DW      ?
NUM4        DD      3
LETRA1      DB      'A'
LETRA1      DW      'A'
CADENA1     DB      "ABC"
CADENA2     DW      "ABC"
VECTOR      DB      3 DUP ( 2, ? , 0 )
MATR        DB      3 DUP ( 4 DUP ( 2 ) )
```

Constantes

```
.DATA
Limite          EQU 100
ASCIIzero       EQU 30H
Mascara         EQU 00010000B
SimpleDecimal EQU 823D
```

Procs

El procedimiento (subrutina o **función**) es una parte importante de la arquitectura de cualquier sistema computacional. Un procedimiento es un grupo de instrucciones que por lo general desempeñan una tarea. Un procedimiento es una sección reutilizable del software que se almacena en memoria una vez, pero se utiliza tantas veces como sea necesario. Esto ahorra espacio en memoria y facilita el desarrollo del software. La única desventaja de un procedimiento es que la computadora requiere de una pequeña cantidad de tiempo para enlazarse al procedimiento y regresar de él. La instrucción CALL enlaza el procedimiento y la instrucción RET (**retorno**) regresa del procedimiento.

La pila almacena la dirección de retorno siempre que se llama a un procedimiento durante la ejecución de un programa. La instrucción CALL mete en la pila la dirección de la instrucción que va después de CALL (**dirección de retorno**). La instrucción RET saca una dirección de la pila, para que el programa regrese a la instrucción que va después de CALL.

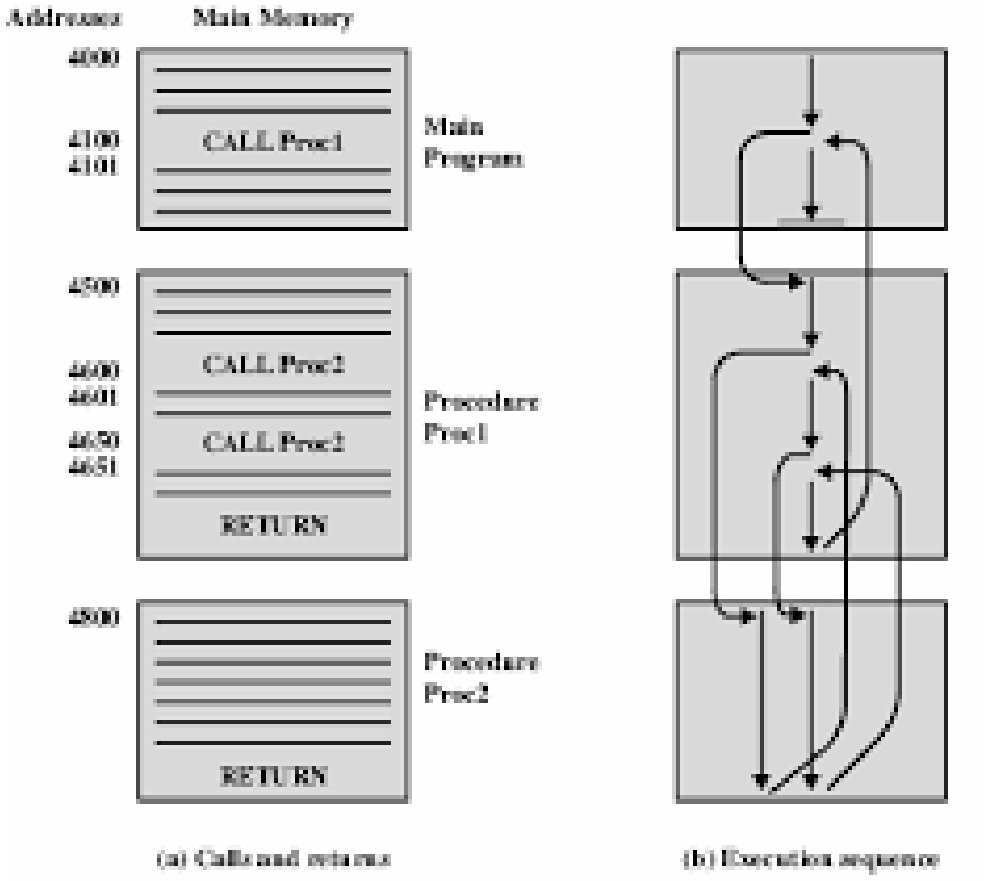
Procs

La pila almacena la dirección de retorno siempre que se llama a un procedimiento durante la ejecución de un programa. La instrucción CALL mete en la pila la dirección de la instrucción que va después de CALL (**dirección de retorno**). La instrucción RET saca una dirección de la pila, para que el programa regrese a la instrucción que va después de CALL.

EJEMPLO 6-14

```
0000                    SUMAS     PROC NEAR
0000  03  C3                     ADD  AX,BX
0002  03  C1                     ADD  AX,CX
0004  03  C2                     ADD  AX,DX
0006  C3                         RET
0007                    SUMAS     ENDP

0007                    SUMAS1    PROC FAR
0007  03  C3                     ADD  AX,BX
0009  03  C1                     ADD  AX,CX
000B 03  C2                      ADD  AX,DX
000D  CB                         RET
000E                    SUMAS1    ENDP
```

Procs


Addresses
Main Memory
4000
4100
4101
CALL Proc1
Main
Program
4500
4600
4601
4650
4651
CALL Proc2
CALL Proc2
RETURN
Procedure
Proc1
4800
RETURN
Procedure
Proc2
(a) Calls and returns
(b) Execution sequence

AND

FIGURA 5-3 (a) La tabla de verdad para la operación AND y (b) el símbolo lógico de una compuerta AND.

A	B	T
0	0	0
0	1	0
1	0	0
1	1	1

(a)

(b)

FIGURA 5-4 La operación de la función AND que muestra cómo se hacen cero los bits de un número.

	x x x x x x x x	Número desconocido
•	0 0 0 0 1 1 1 1	Máscara
	0 0 0 0 x x x x	Resultado

AND

EJEMPLO 5-25

```
0000  BB  3135          MOV    BX,3135H          ;carga el valor ASCII
0003  81  E3  0F0F      AND    BX,0F0FH          ;enmascara BX
```

TABLA 5-14 Ejemplos de instrucciones AND.

Lenguaje ensamblador	*Operación*
AND AL,BL	AL = AL and BL
AND CX,DX	CX = CX and DX
AND ECX,EDI	ECX = ECX and EDI
AND CL,33H	CL = CL and 33H
AND DI,4FFFH	DI = DI and 4FFFH
AND ESI, 34H	ESI = ESI and 34H
AND AX,[DI]	Se aplica un AND entre AX y el contenido tipo palabra de la posición de memoria del segmento de datos direccionada por DI.
AND ARREGLO[SI],AL	Se aplica un AND entre AL y el contenido tipo byte de la posición de memoria del segmento de datos direccionada por ARREGLO más SI.
AND [EAX],CL	Se aplica un AND entre CL y el contenido tipo byte de la posición de memoria del segmento de datos direccionada por ECX.

OR

FIGURA 5-5 (a) La tabla de verdad para la operación OR y (b) el símbolo lógico de una compuerta OR.

A	B	T
0	0	0
0	1	1
1	0	1
1	1	1

(a)

(b)

FIGURA 5-6 La operación de la función OR que muestra cómo los bits de un número se hacen uno.

```
  x x x x  x x x x   Número desconocido
+ 0 0 0 0  1 1 1 1   Máscara
-------------------
  x x x x  1 1 1 1   Resultado
```

OR

TABLA 5-15 Ejemplos de instrucciones OR.

Lenguaje ensamblador	*Operación*
OR AH,BL	AL = AL or BL
OR SI,DX	SI = SI or DX
OR EAX,EBX	EAX = EAX or EBX
OR DH,0A3H	DH = DH or 0A3H
OR SP,990DH	SP = SP or 990DH
OR EBP,10	EBP = EBP or 10
OR DX,[BX]	Se aplica un OR entre DX y el contenido tipo palabra de la posición de memoria del segmento de datos direccionada por BX.
OR FECHAS[DI+2],AL	Se aplica un OR entre AL y el contenido tipo byte de la posición de memoria del segmento de datos direccionada por DI más 2.

EJEMPLO 5-26

```
0000  B0  05          MOV     AL,5              ;carga los datos
0002  B3  07          MOV     BL,7
0004  F6  E3          MUL     BL
0006  D4  0A          AAM                       ;ajuste
0008  0D  3030        OR      AX,3030H          ;convierte en ASCII
```

XOR

FIGURA 5-7 (a) La tabla de verdad para la operación OR exclusivo y (b) el símbolo lógico de una compuerta OR exclusivo.

A	B	T
0	0	0
0	1	1
1	0	1
1	1	0

(a)

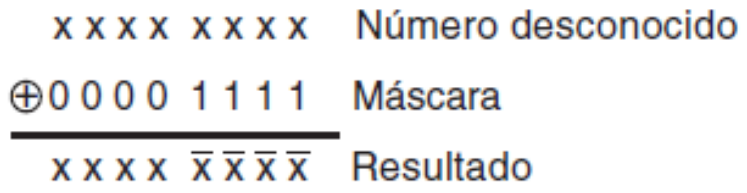
(b)

TABLA 5-16 Ejemplos de instrucciones OR exclusivo.

Lenguaje ensamblador	*Operación*
XOR CH,DL	CH = CH xor DL
XOR SI,BX	SI = SI xor BX
XOR EBX,EDI	EBX = EBX xor EDI
XOR AH,0EEH	AH = AH xor 0EEH
XOR DI,00DDH	DI = DI xor 00DDH
XOR ESI,100	ESI = ESI xor 100
XOR DX,[SI]	Se aplica un OR exclusivo entre DX y el contenido tipo palabra de la posición de memoria del segmento de datos direccionada por SI.
XOR TRATO[BP+2],AH	Se aplica un OR exclusivo entre AH y el contenido tipo byte de la posición de memoria del segmento de pila direccionada por BP más 2.

XOR

FIGURA 5-8 La operación de la función OR exclusivo que muestra cómo se invierten los bits de un número.



EJEMPLO 5-27

```
0000  81  C9  0600          OR      CX,0600H              ;activa los bits 9 y 10
0004  83  E1  FC            AND     CX,0FFFCH             ;borra los bits 0 y 1
0007  81  F1  1000          XOR     CX,1000H              ;invierte el bit 12
```

NOT y NEG

La instrucción NOT invierte todos los bits de un byte, una palabra o una doble palabra. La instrucción NEG complementa a dos un número, lo que significa que el signo aritmético de un número con signo cambia de positivo a negativo o viceversa. La función NOT se considera lógica; la función NEG se considera una operación aritmética.

TABLA 5-19 Ejemplos de instrucciones NOT y NEG.

Lenguaje ensamblador	*Operación*
NOT CH	CH se complementa a uno.
NEG CH	CH se complementa a dos.
NEG AX	AX se complementa a dos.
NOT EBX	EBX se complementa a uno.
NEG ECX	ECX se complementa a dos.
NOT TEMP	El contenido de la posición de memoria TEMP del segmento de datos se complementa a uno.
NOT BYTE PTR[BX]	El contenido tipo byte de la posición de memoria del segmento de datos direccionada por BX se complementa a uno.

IF

6.5.1 Instrucciones IF con estructura de bloque

En la mayoría de los lenguajes de alto nivel, una estructura IF implica que una expresión booleana debe ir seguida de dos listas de instrucciones: una que se realiza cuando la expresión es verdadera, y otra que se realiza cuando la expresión es falsa:

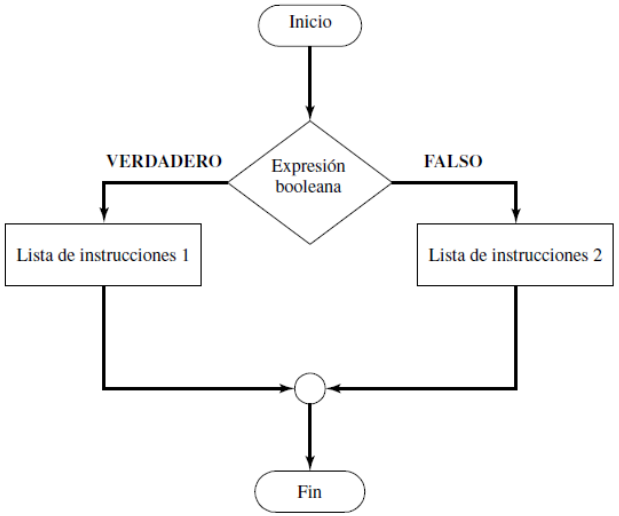
```
if( expresión )
  lista-instrucciones-1
else
  lista-instrucciones-2
```

La porción del **else** de la instrucción es opcional. El diagrama de flujo de la figura 6-1 muestra las dos rutas de bifurcación en una estructura IF condicional, las cuales se etiquetan como *verdadero* y *falso*.

Ejemplo 1 Usando la sintaxis de Java/C++, se ejecutan dos instrucciones de asignación si **op1** es igual a **op2**:

```
if( op1 == op2 )
{
      X = 1;
      Y = 2;
}
```

Diagrama de flujo de una estructura IF.



IF

La única manera de traducir esta instrucción IF en lenguaje ensamblador es utilizar una instrucción CMP, seguida de uno o más saltos condicionales. Como **op1** y **op2** son operandos de memoria, hay que mover uno de ellos a un registro antes de ejecutar la instrucción CMP. El siguiente código implementa la instrucción IF de la manera más eficiente posible, invirtiendo la condición de igualdad y usando la instrucción JNE:

```
        mov   eax,op1
        cmp   eax,op2                ; ¿op1 == op2?
        jne   L1                     ; no: salta la siguiente instrucción
        mov   X,1                    ; sí: asigna X y Y
        mov   Y,2
L1:
```

Si implementamos el operador == usando JE, el código resultante será menos compacto (seis instrucciones, en vez de cinco):

```
        mov   eax,op1
        cmp   eax,op2                ; ¿op1 == op2?
        je    L1                     ; sí: salta a L1
        jmp   L2                     ; no: salta las asignaciones
L1:     mov   X,1                    ; sí: asigna X y Y
        mov   Y,2
L2:
```

Comparación: CMP

La instrucción de comparación (CMP) es una resta que sólo cambia los bits de bandera; el operando de destino nunca cambia. Una comparación es útil para comprobar todo el contenido de un registro o de una posición de memoria sobre la base de otro valor. Por lo general, una instrucción CMP va seguida de una instrucción de salto condicional, que evalúa la condición de los bits de bandera.

El ejemplo 5-12 muestra una comparación seguida de una instrucción de salto condicional. En este ejemplo, el contenido de AL se compara con 10H. Las instrucciones de salto condicional que van por lo general después de la comparación son JA (**salto si es mayor**) y JB (**salto si es menor**). Si la instrucción JA va después de la comparación, el salto se realiza si el valor en AL es mayor de 10H. Si la instrucción JB va después de la comparación, el salto se realiza si el valor en AL es menor de 10H. En este ejemplo, la instrucción JAE va después de la comparación. Esta instrucción hace que el programa continúe en la posición de memoria SUBER si el valor en AL es 10H o mayor. También hay una instrucción JBE (**salto si es menor o igual**) que podría ir después de la comparación para realizar el salto si el resultado es menor o igual a 10H. En los siguientes capítulos se proporcionarán más detalles sobre las instrucciones de comparación y de salto condicional.

EJEMPLO 5-12

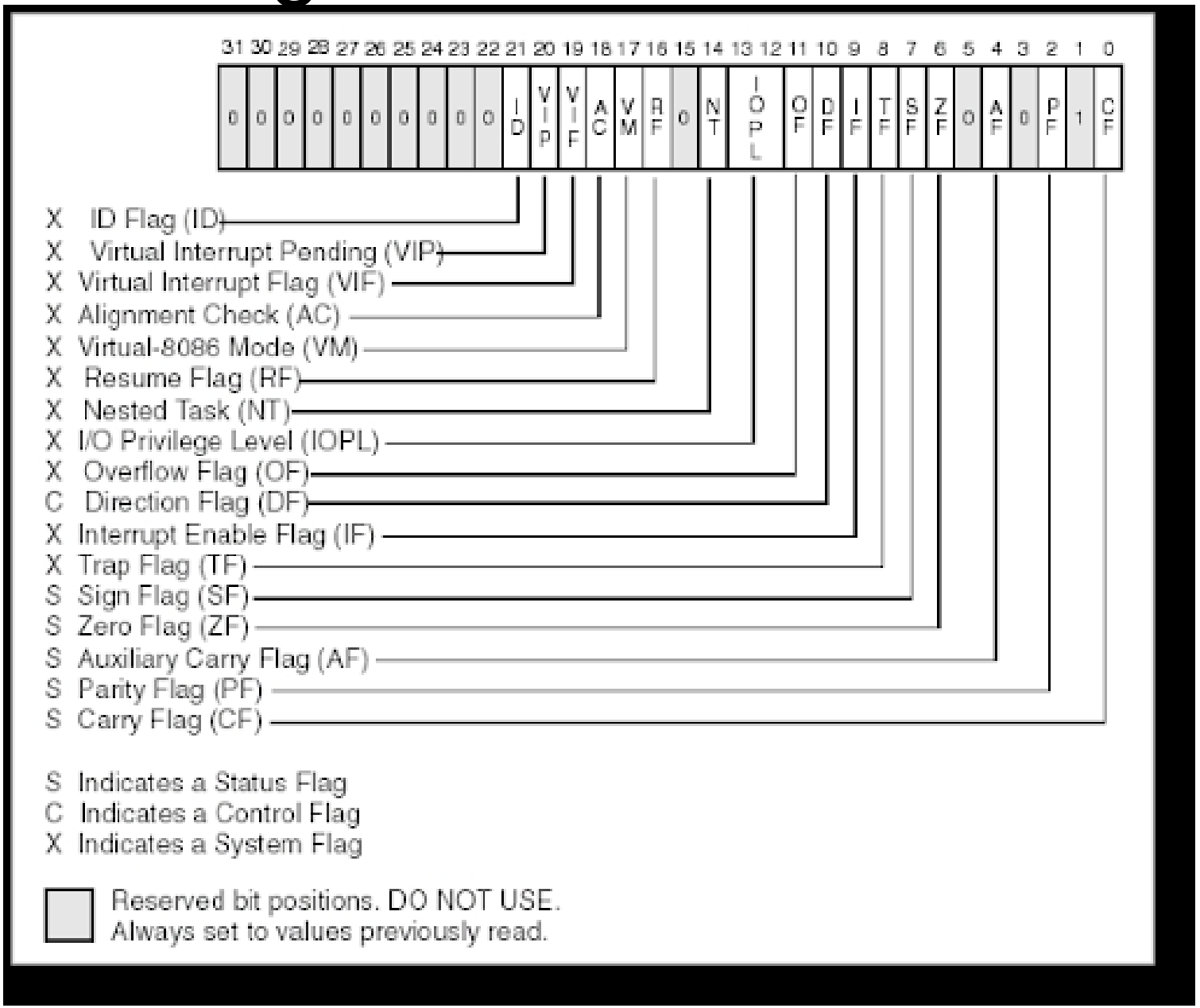
```
0000  3C  10        CMP     AL,10H          ;compara AL con 10H
0002  73  1C        JAE     SUBER           ;si AL es 10H o mayor
```

Comparación: CMP

TABLA 5-7 Ejemplos de instrucciones de comparación.

Lenguaje ensamblador	*Operación*
CMP CL,BL	CL – BL
CMP AX,SP	AX – SP
CMP EBP,ESI	EBP – ESI
CMP AX,2000H	AX – 2000H
CMP [DI],CH	CH se resta del contenido tipo byte de la posición de memoria del segmento de datos direccionada por DI.
CMP CL,[BP]	El contenido tipo byte de la posición de memoria del segmento de pila direccionada por BP se resta de CL.
CMP AH,TEMP	El contenido tipo byte de la posición de memoria TEMP del segmento de datos se resta de AH.
CMP DI,TEMP[BX]	El contenido tipo palabra de la posición de memoria del segmento de datos direccionada por TEMP más BX se resta de DI.
CMP AL,[EDI+ESI]	El contenido tipo byte de la posición de memoria del segmento de datos direccionada por EDI más ESI se resta de AL.

Registro de Banderas



Instrucciones de salto condicional

Las instrucciones de salto condicional evalúan los siguientes bits de bandera: signo (S), cero (Z), acarreo (C), paridad (P) y desbordamiento (0). Si la condición que se evalúa es verdadera, se produce una ramificación hacia la etiqueta asociada con la instrucción de salto. Si la condición es falsa, se ejecuta el siguiente paso secuencial en el programa. Por ejemplo, una instrucción JC realiza el salto si el bit de acarreo está activado.

Lenguaje ensamblador	*Condición a evaluar*	*Operación*
JA	Z = 0 y C = 0	Salta si está por encima.
JAE	C = 0	Salta si está por encima o si es igual.
JB	C = 1	Salta si está por debajo.
JBE	Z = 1 o C = 1	Salta si está por debajo o si es igual.
JC	C = 1	Salta si hay acarreo.
JE o JZ	Z = 1	Salta si es igual o salta si es cero.
JG	Z = 0 y S = 0	Salta si es mayor que.
JGE	S = 0	Salta si es mayor o igual que.
JL	S != O	Salta si es menor que.
JLE	Z = 1 o S != O	Salta si es menor o igual que.
JNC	C = 0	Salta si no hay acarreo.
JNE o JNZ	Z = 0	Salta si no es igual o salta si no es cero.
JNO	O = 0	Salta si no hay desbordamiento.
JNS	S = 0	Salta si no hay signo (positivo).
JNP o JPO	P = 0	Salta si no hay paridad o si la paridad es impar.
JO	O = 1	Salta si hay desbordamiento.
JP o JPE	P = 1	Salta si hay paridad o si la paridad es par.
JS	S = 1	Salta si hay signo (negativo).
JCXZ	CX = 0	Salta si CX es igual a cero.

Instrucciones de salto condicional

```
mov b,0       ;Inicialización de variables
mov e,0

call leecar   ;Procedimiento que lee un caracter

 cmp al,'s'   ;Compara el caracter leído con 's'
je prefin          ; si es igual termina

cmp e,1       ;Compara la variable e con 1
je menu1           ;Si es igual salta a menu1

mov v1,al     ;Almacena el valor leido en la variable v1

mov b,1       ;Switch estado de bandera propia
```

Instrucciones de salto condicional

Ejemplo 3 La siguiente instrucción IF-ELSE en seudocódigo tiene bifurcaciones alternativas:

```
if op1 > op2 then
     call Rutina1
else
     call Rutina2
end if
```

En la siguiente traducción de lenguaje ensamblador, asumimos que op1 y op2 son variables tipo doble palabra con signo. El operador > se implementa mejor usando JNG, el complemento de JG:

```
     mov   eax,op1
     cmp   eax,op2                ; ¿op1 > op2?
     jng   A1                     ; no: llama a Rutina2
     call  Rutina1                ; sí: llama a Rutina1
     jmp   A2
A1:  call  Rutina2
A2:
```

Instrucciones de salto condicional

Operador AND lógico

El lenguaje ensamblador implementa con facilidad las expresiones booleanas compuestas que contienen operadores AND. Considere el siguiente seudocódigo, en el que se asume que las variables son enteros sin signo:

```
if (al > bl) AND (bl > cl)
{
     X = 1
}
```

Evaluación de corto circuito La siguiente es una implementación directa que utiliza la evaluación de *corto circuito*, en la que la segunda expresión no se evalúa si la primera expresión es falsa:

```
     cmp   al,bl                ; primera expresión...
     ja    L1
     jmp   siguiente
L1:  cmp   bl,cl                ; segunda expresión...
     ja    L2
     jmp   siguiente
L2: mov    X,1                  ; ambas verdaderas: se establece X en 1
siguiente:
```

Podemos optimizar el código a cinco instrucciones, si cambiamos la instrucción JA inicial por JBE:

```
     cmp   al,bl                ; primera expresión...
     jbe   siguiente            ; termina si es falso
     cmp   bl,cl                ; segunda expresión
     jbe   siguiente            ; termina si es falso
     mov   X,1                  ; ambas son verdaderas
siguiente:
```

Instrucciones de salto condicional

Operador OR lógico

Cuando ocurren varias expresiones en una expresión compuesta que utiliza el operador lógico OR, la expresión es automáticamente verdadera, tan pronto como cualquiera de las expresiones sean verdaderas. Vamos a utilizar el siguiente seudocódigo como ejemplo:

```
if (al > bl) OR (bl > cl)
   X = 1
```

En la siguiente implementación, el código se bifurca a L1 si la primera expresión es verdadera; en caso contrario, pasa a la segunda instrucción CMP. La segunda expresión invierte el operador > y utiliza JBE en su defecto:

```
      cmp   al,bl                ; 1: compara AL con BL
      ja    L1                   ; si es verdadero, omite la segunda expresión
      cmp   bl,cl                ; 2: compara BL con CL
      jbe   siguiente            ; falso: omite la siguiente instrucción
L1:   mov   X,1                  ; verdadero: establece X = 1
siguiente:
```

Para una expresión compuesta dada, hay por lo menos varias formas en que ésta puede implementarse en lenguaje ensamblador.

Instrucciones de salto condicional

Ejemplo: instrucción IF anidada en un ciclo

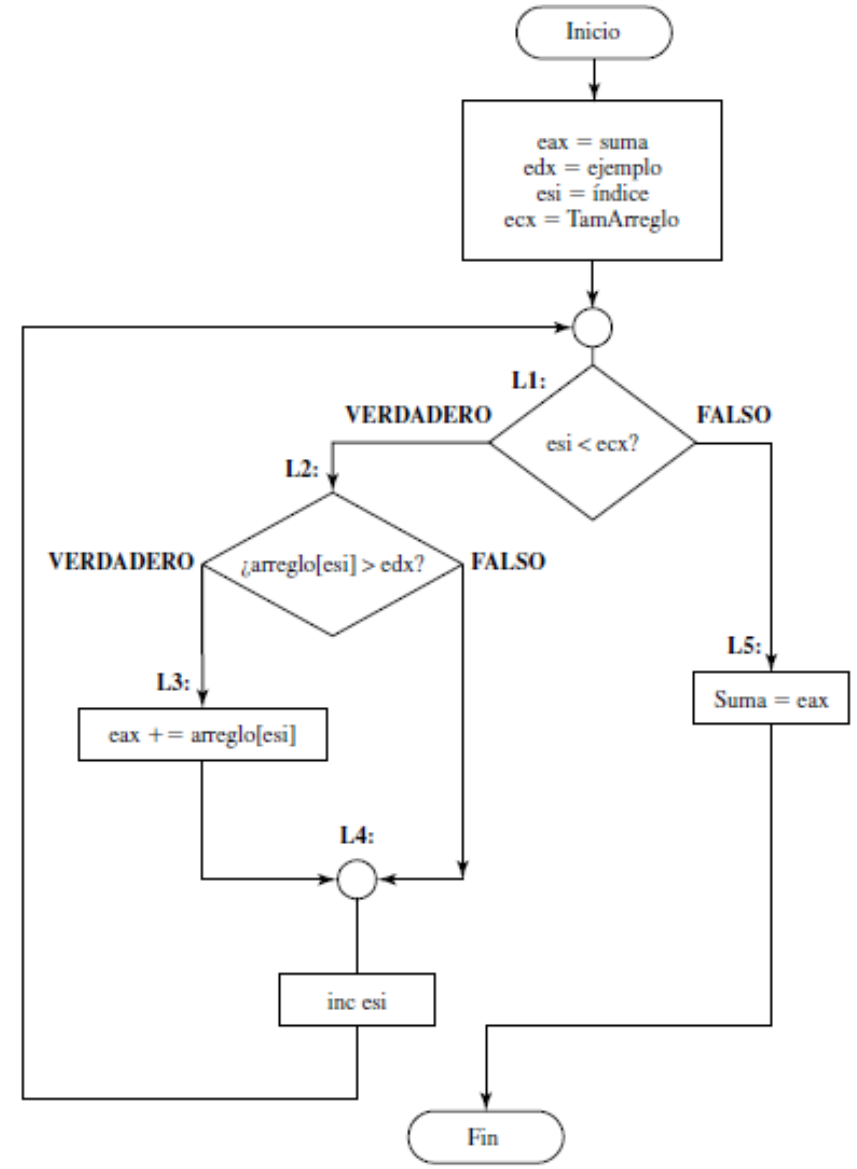
Los lenguajes estructurados de alto nivel son muy adecuados para representar estructuras de control anidadas. En el siguiente ejemplo en C++, una instrucción IF está anidada dentro de un ciclo WHILE. Esta instrucción calcula la suma de todos los elementos del arreglo que sean mayores que el valor en **ejemplo**:

```
int arreglo[] = {10,60,20,33,72,89,45,65,72,18};
int ejemplo = 50;
int TamArreglo = sizeof arreglo / sizeof ejemplo;
int indice = 0;
int suma = 0;
while( indice < TamArreglo )
{
   if( arreglo[indice] > ejemplo )
   {
     suma += arreglo[indice];
     indice++;
   }
}
```

Antes de codificar este ciclo en lenguaje ensamblador, utilizaremos el diagrama de flujo de la figura 6-2 para describir la lógica. Para simplificar la traducción y agilizar la ejecución reduciendo el número de accesos a memoria, hemos sustituido los registros por variables. EDX = ejemplo, EAX = suma, ESI = indice, y ECX = TamArreglo (una constante). Se agregaron nombres de etiquetas a las figuras.

Instrucciones de salto condicional

Ciclo que contiene una instrucción IF.




```
.data
suma DWORD 0
ejemplo DWORD 50
arreglo DWORD 10,60,20,33,72,89,45,65,72,18
TamArreglo = ($ - Arreglo) / TYPE arreglo

.code
main PROC
     mov   eax,0                        ; suma
     mov   edx,ejemplo
     mov   esi,0                        ; índice
     mov   ecx,TamArreglo

     L1:  cmp   esi,ecx
          jl    L2
          jmp   L5

     L2:  cmp   arreglo[esi*4], edx
          jg    L3
          jmp   L4
     L3:  add   eax,arreglo[esi*4]

     L4:  inc   esi
          jmp   L1

     L5:  mov   suma,eax
```

TEST

Instrucciones TEST y de prueba de bits

La **instrucción TEST** realiza la operación AND. La diferencia es que la instrucción AND modifica el operando de destino, mientras que TEST no. Una instrucción TEST sólo afecta la condición del registro de banderas, el cual indica el resultado de la prueba. La instrucción TEST usa los mismos modos de direccionamiento que la instrucción AND. La tabla 5-17 muestra algunas instrucciones TEST y sus operaciones.

La instrucción TEST funciona de la misma manera que una instrucción CMP. La diferencia es que la instrucción TEST por lo general evalúa un solo bit (o en algunas ocasiones varios bits), mientras que la instrucción CMP evalúa todo el byte, palabra o doble palabra. La bandera cero (Z) es un 1 lógico (lo que indica un resultado de cero) si el bit que se evalúa es un cero, y Z = 0 (lo que indica un resultado distinto de cero) si el bit que se evalúa es distinto de cero.

TEST

Por lo general, la instrucción TEST va seguida de la instrucción JZ (**salto si es cero**) o JNZ (**salto si es distinto de cero**). Lo común es evaluar el operando de destino y compararlo con datos inmediatos. El valor de los datos inmediatos es 1 para evaluar la posición del bit de más a la derecha, 2 para evaluar el siguiente bit, 4 para el siguiente y así sucesivamente.

El ejemplo 5-28 muestra un programa corto que evalúa las posiciones de los bits de más a la derecha y de más a la izquierda del registro AL. Aquí, 1 selecciona el bit de más a la derecha y 128 selecciona el bit de más a la izquierda. (Nota: Un 128 es un 80H.) La instrucción JNZ va después de cada prueba para saltar hacia distintas posiciones de memoria, dependiendo del resultado de las pruebas. La instrucción JNZ salta a la dirección del operando (DERECHA o IZQUIERDA en el ejemplo) si el bit que se evalúa es distinto de cero.

EJEMPLO 5-28

```
0000  A8  01          TEST    AL,1            ;evalúa el bit derecho
0002  75  1C          JNZ     DERECHA         ;si está activado
0004  A8  80          TEST    AL,128          ;evalúa el bit izquierdo
0006  75  38          JNZ     IZQUIERDA       ;si está activado
```

TEST DE BITS ALTERNATIVO

TABLA 5-18 Instrucciones de prueba de bits.

Lenguaje ensamblador	*Operación*
BT	Evalúa un bit en el operando de destino, el cual se especifica mediante el operando de origen.
BTC	Evalúa y complementa un bit en el operando de destino, el cual se especifica mediante el operando de origen.
BTR	Evalúa y restablece un bit en el operando de destino, el cual se especifica mediante el operando de origen.
BTS	Evalúa y establece un bit en el operando de destino, el cual se especifica mediante el operando de origen.

El ejemplo 5-27 muestra una secuencia corta de instrucciones que borra los bits 0 y 1 de CX, activa los bits 9 y 10 de CX e invierte el bit 12 de CX. La instrucción OR se utiliza para activar bits, la instrucción AND se utiliza para borrar bits y la instrucción XOR invierte bits.

EJEMPLO 5-27

```
0000  81  C9  0600        OR      CX,0600H          ;activa los bits 9 y 10
0004  83  E1  FC          AND     CX,0FFFCH         ;borra los bits 0 y 1
0007  81  F1  1000        XOR     CX,1000H          ;invierte el bit 12
```

TEST DE BITS ALTERNATIVO

Los procesadores del 80386 al Pentium 4 contienen instrucciones de prueba adicionales que evalúan las posiciones de bits individuales. La tabla 5-18 lista las cuatro instrucciones de prueba de bits distintas que están disponibles para estos microprocesadores.

Las cuatro formas de la instrucción de prueba de bits evalúan la posición del bit en el operando de destino seleccionado por el operando de origen. Por ejemplo, la instrucción BT AX,4 evalúa la posición del bit 4 en AX. El resultado de la prueba se encuentra en el bit de la bandera de acarreo. Si la posición del bit 4 es 1, el acarreo está activado; si la posición del bit 4 es un 0, el acarreo está desactivado.

Las tres instrucciones de prueba de bits restantes también colocan el bit a evaluar en la bandera de acarreo y lo modifican después de la prueba. La instrucción BTC AX,4 complementa la posición del bit 4 después de evaluarla, la instrucción BTR AX,4 lo borra (0) después de la prueba y la instrucción BTS AX,4 lo activa (1) después de la prueba.

El ejemplo 5-29 repite la secuencia de instrucciones que se listan en el ejemplo 5-27. Aquí la instrucción BTR borra los bits en CX, BTS activa los bits en CX y BTC invierte los bits en CX.

EJEMPLO 5-29

```
0000  0F  BA  E9  09          BTS     CX,9          ;activa el bit 9
0004  0F  BA  E9  0A          BTS     CX,10         ;activa el bit 10
0008  0F  BA  F1  00          BTR     CX,0          ;borra el bit 0
000C  0F  BA  F1  01          BTR     CX,1          ;borra el bit 1
0010  0F  BA  F9  0C          BTC     CX,12         ;complementa el bit 12
```

Desplazamiento

DESPLAZAMIENTO (SHIFT) Y DESPLAZAMIENTO CÍCLICO (ROTATE)

Estas instrucciones manipulan números binarios a nivel de bit binario, como las instrucciones AND, OR, OR exclusivo y NOT. Los desplazamientos y los desplazamientos cíclicos se aplican con más frecuencia en el software de bajo nivel que se utiliza para controlar dispositivos de E/S. El microprocesador contiene un conjunto completo de instrucciones de desplazamiento y de desplazamiento cíclico que se utilizan para desplazar o desplazar en forma cíclica cualquier dato de memoria o registro.

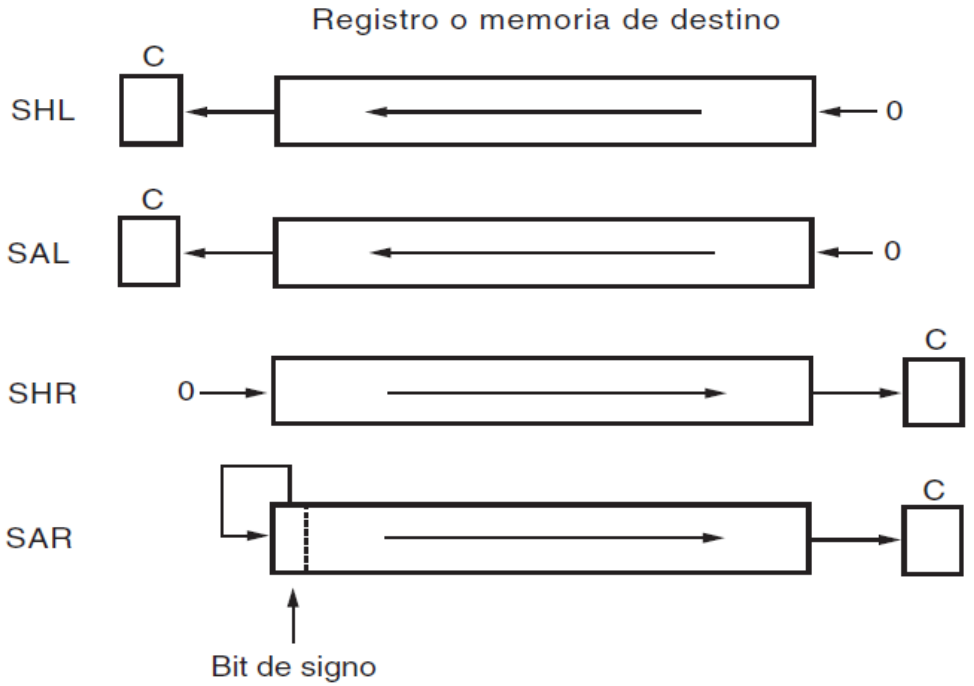
Desplazamiento

Estas instrucciones posicionan o mueven números a la izquierda o a la derecha dentro de un registro o posición de memoria. También realizan aritmética simple tal como la multiplicación por potencias de 2^{+n} (desplazamiento a la izquierda) y la división entre potencias de 2^{-n} (desplazamiento a la derecha). El conjunto de instrucciones del microprocesador contiene cuatro instrucciones de desplazamiento distintas: dos son desplazamientos lógicos y dos son desplazamientos aritméticos. En la figura 5-9 se muestran las cuatro operaciones de desplazamiento.

Desplazamiento cíclico. Estas instrucciones posicionan datos binarios mediante la rotación de la información en un registro o posición de memoria, ya sea de un extremo a otro o a través de la bandera de acarreo. A menudo se utilizan para desplazar o posicionar números que son mayores de 16 bits en los microprocesadores del 8086 al 80286, o mayores de 32 bits en los microprocesadores del 80386 al Pentium 4. En la figura 5-10 aparecen las cuatro instrucciones de desplazamiento cíclico disponibles.

Desplazamiento

FIGURA 5-9 Las instrucciones de desplazamiento que muestran la operación y la dirección del desplazamiento.



EJEMPLO 5-30

```
0000   C1   E2   0E          SHL     DX,14

                  o

0003   B1   0E              MOV     CL,14
0005   D3   E2              SHL     DX,CL
```

Desplazamiento

TABLA 5-20 Ejemplos de instrucciones de desplazamiento.

Lenguaje ensamblador	*Operación*
SHL AX,1	AX se desplaza en forma lógica 1 posición a la izquierda.
SHR BX,12	BX se desplaza en forma lógica 12 posiciones a la derecha.
SHR ECX,10	ECX se desplaza en forma lógica 10 posiciones a la derecha.
SAL DATOS1,CL	El contenido de la posición de memoria DATOS1 del segmento de datos se desplaza en forma aritmética a la izquierda, el número de espacios especificado por CL.
SAR SI,2	SI se desplaza en forma aritmética 2 posiciones a la derecha.
SAR EDX,14	EDX se desplaza en forma aritmética 14 posiciones a la derecha.

Multiplicación rápida SHL puede realizar multiplicaciones de alta velocidad, por potencias de 2. Si se desplaza cualquier operando a la izquierda por n bits, esto equivale a multiplicar el operando por 2^n. Por ejemplo, si se desplaza el número 5 a la izquierda por 1 bit, se produce el producto de 5 * 2:

```
mov  dl,5
shl  dl,1
```

Antes: 0 0 0 0 0 1 0 1 = 5

Después: 0 0 0 0 1 0 1 0 = 10

Si el número 10 decimal se desplaza 2 bits a la izquierda, el resultado es el mismo que si se multiplica el 10 por 2^2:

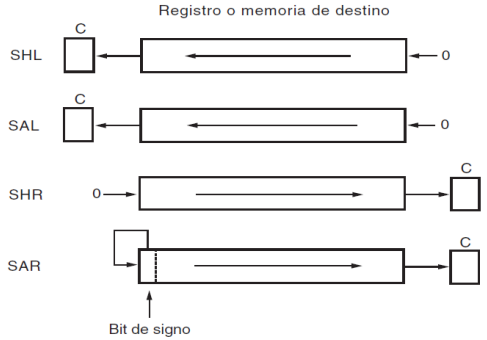
```
mov  dl,10
shl  dl,2                    ; (10 * 4) = 40
```

Desplazamiento

Suponga que el contenido de AX debe multiplicarse por 10, como se muestra en el ejemplo 5-31. Esto puede hacerse de dos formas: mediante la instrucción MUL o mediante desplazamientos y sumas. Un número se duplica cuando se desplaza una posición a la izquierda. Cuando un número se duplica y luego se suma al mismo número multiplicado por 8, el resultado es 10 veces ese número. El número 10 decimal es 1010 en binario. Un 1 lógico aparece en la posición 2 y 8. Si el número se multiplica por 2 y luego se suma al mismo número multiplicado por 8, el resultado es 10 veces ese número. Mediante esta técnica puede escribirse un programa para multiplicar por cualquier constante. Esta técnica se ejecuta por lo general más rápido que la instrucción de multiplicación que se incluye en las primeras versiones del microprocesador Intel.

EJEMPLO 5-31

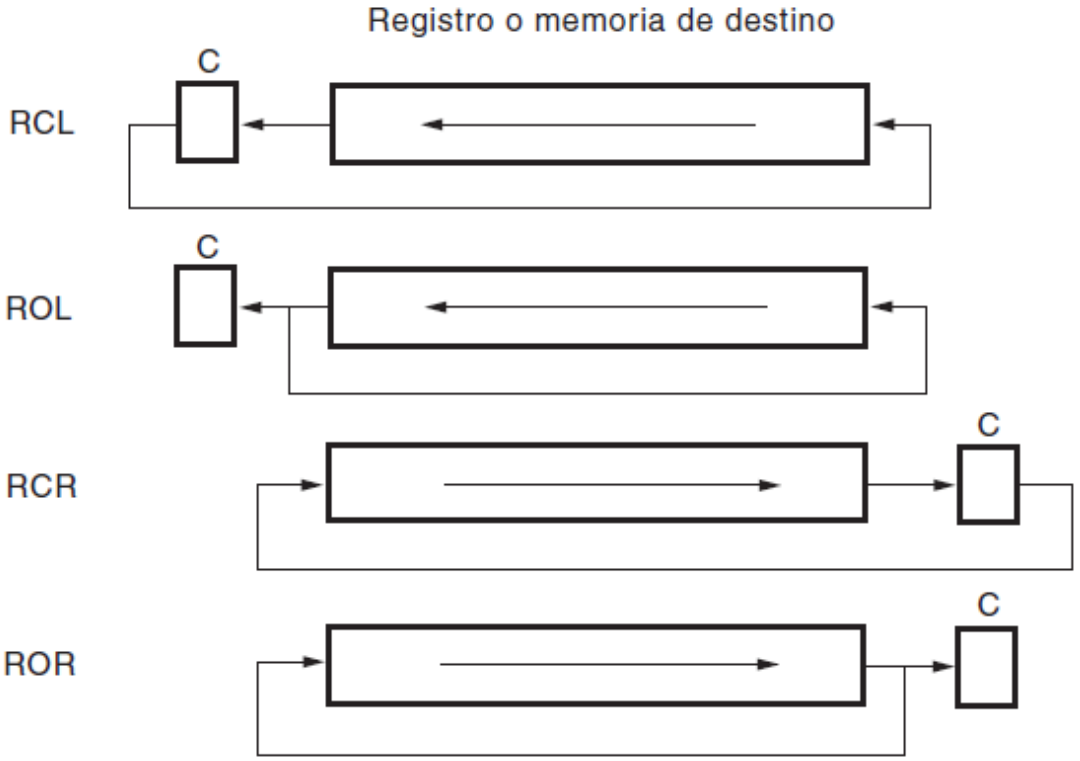
```
                        ;Multiplica AX por 10 (1010)
                        ;
0000  D1  E0            SHL     AX,1            ;AX por 2
0002  8B  D8            MOV     BX,AX
0004  C1  E0  02        SHL     AX,2            ;AX por 8
0007  03  C3            ADD     AX,BX           ;AX por 10
                        ;
                        ;Multiplica AX por 18 (10010)
                        ;
0009  D1  E0            SHL     AX,1            ;AX por 2
000B  8B  D8            MOV     BX,AX
000D  C1  E0  03        SHL     AX,3            ;AX por 16
0010  03  C3            ADD     AX,BX           ;AX por 18
                        ;
                        ;Multiplica AX por 5 (101)
                        ;
0012  8B  D8            MOV     BX,AX
0014  C1  E0  02        SHL     AH,2            ;AX por 4
0017  03  C3            ADD     AX,BX           ;AX por 5
```



Rotaciones

Desplazamiento cíclico. Estas instrucciones posicionan datos binarios mediante la rotación de la información en un registro o posición de memoria, ya sea de un extremo a otro o a través de la bandera de acarreo. A menudo se utilizan para desplazar o posicionar números que son mayores de 16 bits en los microprocesadores del 8086 al 80286, o mayores de 32 bits en los microprocesadores del 80386 al Pentium 4. En la figura 5-10 aparecen las cuatro instrucciones de desplazamiento cíclico disponibles.

FIGURA 5-10
Las instrucciones de desplazamiento cíclico que muestran la dirección y la operación de cada desplazamiento cíclico.



Rotaciónes

Las instrucciones de desplazamiento cíclico se utilizan a menudo para desplazar números grandes a la izquierda o a la derecha. El programa que se lista en el ejemplo 5-32 desplaza el número de 48 bits que está en los registros DX, BX y AX una posición binaria a la izquierda. Observe que los 16 bits menos significativos (AX) se desplazan primero a la izquierda. Esto hace que se mueva el bit de más a la izquierda de AX hacia el bit de bandera de acarreo. A continuación, la instrucción de desplazamiento cíclico de BX desplaza en forma cíclica el acarreo hacia BX, y su bit de más a la izquierda se mueve hacia el acarreo. La última instrucción desplaza el acarreo en forma cíclica hacia DX, y con esto termina el desplazamiento.

EJEMPLO 5-32

```
0000  D1  E0              SHL     AX,1
0002  D1  D3              RCL     BX,1
0004  D1  D2              RCL     DX,1
```

Rotaciónes

TABLA 5-21 Ejemplos de instrucciones de desplazamiento cíclico.

Lenguaje ensamblador	*Operación*
ROL SI,14	SI se desplaza en forma cíclica 14 posiciones a la izquierda.
RCL BL,6	BL se desplaza en forma cíclica 6 posiciones a la izquierda, a través del acarreo.
ROL ECX,18	ECX se desplaza en forma cíclica 18 posiciones a la izquierda
RCR AH,CL	AH se desplaza en forma cíclica a través del acarreo el número de posiciones que especifique CL.
ROR WORD PTR[BP],2	El contenido tipo palabra de la posición de memoria del segmento de pila direccionada por BP se desplaza en forma cíclica 2 lugares a la derecha.

Multiplicación: MUL

El operando individual es el multiplicador. La tabla 7-2 muestra el multiplicando predeterminado y el producto, dependiendo del tamaño del multiplicador. Como el operando de destino es del doble del tamaño del multiplicando y del multiplicador, no puede ocurrir un desbordamiento. MUL activa las banderas Acarreo y Desbordamiento si la mitad superior del producto no es igual a cero. Por lo general, la bandera Acarreo se utiliza para la aritmética sin signo, por lo que aquí nos enfocaremos en eso. Por ejemplo, cuando AX se multiplica por un operando de 16 bits, el producto se almacena en DX:AX. La bandera Acarreo se activa si DX no es igual a cero.

Tabla 7-2 Operandos de MUL.

Multiplicando	Multiplicador	Producto
AL	r/m8	AX
AX	r/m16	DX:AX
EAX	r/m32	EDX:EAX

Una buena razón para comprobar la bandera Acarreo después de ejecutar MUL, es para saber si la mitad superior del producto puede ignorarse sin riesgos.

Multiplicación: Ejemplos

Las siguientes instrucciones multiplican AL por BL, almacenando el producto en AX. La bandera Acarreo se borra (CF = 0) debido a que AH (la mitad superior del producto) es igual a cero:

```
mov  al,5h
mov  bl,10h
mul  bl                          ; AX = 50h, CF = 0
```

Las siguientes instrucciones multiplican el valor de 16 bits 2000h por 100h. CF = 1, ya que la parte superior del producto en DX no es igual a cero:

```
.data
val1  WORD  2000h
val2  WORD  0100h
.code
mov  ax,val1                     ; AX = 2000h
mul  val2                        ; DX:AX = 00200000h, CF = 1
```

Las siguientes instrucciones multiplican 12345h por 1000h, y el producto es de 64 bits. CF = 0, ya que EDX es igual a cero:

```
mov  eax,12345h
mov  ebx,1000h
mul  ebx                         ; EDX:EAX = 0000000012345000h, CF = 0
```

Multiplicación: IMUL

La instrucción IMUL (multiplicación con signo) realiza la multiplicación de enteros con signo, preservando el signo del producto. El conjunto de instrucciones IA-32 soporta tres formatos para esta instrucción: un operando, dos operandos y tres operandos. En el formato de un operando, el multiplicador y el multiplicando son del mismo tamaño y el producto es del doble de su tamaño. (Los procesadores 8086/8088 sólo soportan el formato de un operando).

Formatos de un operando Los formatos de un operando almacenan el producto en el acumulador (AX, DX:AX o EDX:EAX):

```
IMUL r/m8                ; AX = AL * r/m byte
IMUL r/m16               ; DX:AX = AX * r/m palabra
IMUL r/m32               ; EDX:EAX = EAX * r/m doble palabra
```

Como en el caso de MUL, el tamaño de almacenamiento del producto hace que el desbordamiento sea imposible en la instrucción IMUL de un operando. Las banderas Acarreo y Desbordamiento se activan si la mitad superior del producto no es una extensión del signo de la mitad inferior. Puede utilizar esta información para decidir si debe ignorar o no la mitad superior del producto.

Multiplicación: Ejemplos IMUL

Las siguientes instrucciones multiplican 48 por 4, produciendo +192 en AX. En el producto, AH no es una extensión del signo de AL, por lo que ocurre un desbordamiento con signo:

```
mov   al,48
mov   bl,4
imul  bl                          ; AX = 00C0h, OF = 1
```

Las siguientes instrucciones multiplican −4 por 4, produciendo −16 en AX. AH es una extensión del signo de AL en el producto, por lo que la bandera Desbordamiento se borra:

```
mov   al,-4
mov   bl,4
imul  bl                          ; AX = FFF0h, OF = 0
```

Las siguientes instrucciones multiplican 48 por 4, produciendo +192 en DX:AX. DX es una extensión del signo de AX, por lo que no hay desbordamiento con signo:

```
mov   ax,48
mov   bx,4
imul  bx                          ; DX:AX = 000000C0h, OF = 0
```

Las siguientes instrucciones realizan una multiplicación de 32 bits con signo (4823424 * −423), produciendo −2,040,308,352 en EDX:EAX. EDX es una extensión del signo de EAX, por lo que la bandera Desbordamiento se borra:

```
mov   eax,+482424
mov   ebx,4
imul  ebx                         ; EDX:EAX = FFFFFFF86635D80h, OF = 0
```

Multiplicación: MUL VRS SHL

```
mult_por_desplazamiento PROC
;
; Multiplica EAX por 36 usando SHL, CUENTA_CICLO veces.

     mov    ecx,CUENTA_CICLO
L1: push    eax                     ; guarda el valor original de EAX
     mov    ebx,eax
     shl    eax,5
     shl    ebx,2
     add    eax,ebx
     pop    eax                     ; restaura EAX
     loop   L1

     ret
mult_por_desplazamiento ENDP

mult_por_MUL PROC
;
; Multiplica EAX por 36 usando MUL, CUENTA_CICLO veces.

     mov    ecx,CUENTA_CICLO
L1: push    eax                     ; guarda EAX original
     mov    ebx,36
     mul    ebx
     pop    eax                     ; restaura EAX
     loop   L1

     ret
mult_por_MUL ENDP
```

Multiplicación: MUL VRS SHL

Vamos a llamar a **mult_por_desplazamiento** un gran número de veces y vamos a registrar el tiempo de ejecución:

```
.data
CUENTA_CICLO = 0FFFFFFFFh
.data
valInt DWORD 5
tiempoInicio DWORD ?
.code


call GetMseconds                    ; obtiene tiempo inicial
mov  tiempoInicial,eax

mov  eax,valInt
call mult_por_desplazamiento        ; multiplica ahora

call GetMseconds                    ; obtiene tiempo final
sub  eax,tiempoInicial
call WriteDec                       ; muestra el tiempo transcurrido
```

Suponiendo que llamamos a **mult_por_MUL** de la misma forma, los tiempos resultantes en un Pentium 4 de 4 GHz son evidentes: El método con SHL se ejecuta en 6.078 segundos y el método con MUL se ejecuta en 20.718 segundos. En otras palabras, ¡el uso de la instrucción MUL hace que el cálculo sea un 241 por ciento más lento! (Vea el programa *CompararMult.asm*).

División: DIV

La instrucción DIV (división sin signo) realiza la división de enteros con signo de 8 bits, 16 bits y 32 bits. El registro individual u operando de memoria es el divisor. Los formatos son:

```
DIV  r/m8
DIV  r/m16
DIV  r/m32
```

La siguiente tabla muestra la relación entre el dividendo, el divisor, el cociente y el residuo:

Dividendo	Divisor	Cociente	Residuo
AX	r/m8	AL	AH
DX:AX	r/m16	AX	DX
EDX:EAX	r/m32	EAX	EDX

La división de enteros con signo es casi idéntica a la división sin signo, con una importante diferencia: el dividendo implicado debe tener una extensión completa del signo antes de realizar la división. Primero veremos las instrucciones para la extensión del signo. Después las aplicaremos a la instrucción de división de enteros con signo, IDIV.

División: Ejemplos con DIV

Las siguientes instrucciones realizan una división sin signo de 8 bits (83h/2), produciendo un cociente de 41h y un residuo de 1:

```
mov  ax,0083h                    ; dividendo
mov  bl,2                        ; divisor
div  bl                          ; AL = 41h, AH = 01h
```

Las siguientes instrucciones realizan una división sin signo de 16 bits (8003h/100h), produciendo un cociente de 80h y un residuo de 3. DX contiene la parte superior del dividendo, por lo que debe borrarse antes de ejecutar la instrucción DIV:

```
mov  dx,0                        ; borra dividendo, superior
mov  ax,8003h                    ; dividendo, inferior
mov  cx,100h                     ; divisor
div  cx                          ; AX = 0080h, DX = 0003h
```

La siguiente instrucción realiza una división sin signo de 32 bits, usando un operando de memoria como divisor:

```
.data
dividendo QWORD 0000000800300020h
divisor   DWORD 00000100h
.code
mov  edx,DWORD PTR dividendo + 4    ; doble palabra superior
mov  eax,DWORD PTR dividendo        ; doble palabra inferior
div  divisor                        ; EAX = 08003000h, EDX = 00000020h
```

División: Extención del signo

A menudo, se debe extender el signo de los dividendos de las instrucciones de división de enteros con signo para poder realizar la división (en la sección 4.1.5 se explicó la extensión del signo). Intel proporciona tres instrucciones útiles de extensión de signo: CBW, CWD y CDQ. La instrucción CBW (convertir byte a palabra) extiende el bit de signo de AL hacia AH, preservando el signo del número. En el siguiente ejemplo, 9Bh (en AL) y FF9Bh (en AX) son ambos iguales a −101:

```
.data
valByte SBYTE -101              ; 9Bh
.code
mov  al,valByte                 ; AL = 9Bh
cbw                             ; AX = FF9Bh
```

La instrucción CWD (convertir palabra a doble palabra) extiende el bit de signo de AX hacia DX:

```
.data
valWord SWORD -101              ; FF9Bh
.code
mov  ax,valWord                 ; AX = FF9Bh
cwd                             ; DX:AX = FFFFFF9Bh
```

La instrucción CDQ (convertir doble palabra a palabra cuádruple) extiende el bit de signo de EAX hacia EDX:

```
.data
valDword SDWORD -101            ; FFFFFF9Bh
.code
mov  eax,valDword
cdq                             ; EDX:EAX = FFFFFFFFFFFFFF9Bh
```

División: Extención del signo

La instrucción IDIV (división con signo) realiza una división de enteros con signo, usando los mismos operandos que DIV. Antes de ejecutar la división de 8 bits, se debe extender por completo el signo del dividendo (AX). El residuo siempre tiene el mismo signo que el dividendo.

Ejemplo 1 Las siguientes instrucciones dividen −48 entre 5. Después de ejecutar IDIV, el cociente en AL es −9 y el residuo en AH es −3:

```
.data
valByte SBYTE -48
.code
mov  al,valByte                ; dividendo
cbw                            ; extiende AL hacia AH
mov  bl,+5                     ; divisor
idiv bl                        ; AL = -9, AH = -3
```

División: Extención del signo

Ejemplo 2 La división de 16 bits requiere que se extienda el signo de AX hacia DX. El siguiente ejemplo divide −5000 entre 256:

```
.data
valWord SWORD -5000
.code
mov  ax,valWord                    ; dividendo, inferior
cwd                                ; extiende AX hacia DX

mov  bx,+256                        ; divisor
idiv bx                            ; cociente AX = -19, res DX = -136
```

Ejemplo 3 La división de 32 bits requiere que se extienda el signo de EAX hacia EDX. El siguiente ejemplo divide −5000 entre 256:

```
.data
valDword SDWORD + 50000
.code
mov  eax,valDword                  ; dividendo, inferior
cdq                                ; extiende EAX hacia EDX
mov  ebx,-256                      ; divisor
idiv ebx                           ; cociente EAX = -195, res EDX = +80
```

Todos los valores de las banderas de estado aritméticas quedan indefinidos después de ejecutar DIV e IDIV.

División

EJEMPLO 5-14

```
0000  A0  0000  R        MOV    AL,NUMERO          ;obtiene NUMERO
0003  B4  00             MOV    AH,0               ;extiende con ceros
0005  F6  36  0002  R    DIV    NUMERO1            ;divide entre NUMERO1
0009  A2  0003  R        MOV    RESPC,AL           ;almacena el cociente
000C  88  26  0004  R    MOV    RESPR,AH           ;almacena el residuo
```

División

División de 16 bits. La división de dieciséis bits es similar a la división de 8 bits, sólo que en vez de dividir entre AX, el número de 16 bits se divide entre DX-AX, un dividendo de 32 bits. Después de una división de 16 bits, el cociente aparece en AX y el residuo aparece en DX. La tabla 5-12 lista algunas de las instrucciones de división de 16 bits.

Al igual que en la división de 8 bits, los números deben convertirse con frecuencia a la forma apropiada para el dividendo. Si se coloca un número de 16 bits sin signo en AX, DX debe volverse cero. En los microprocesadores 80386 y superiores el número se extiende con ceros mediante el uso de la instrucción MOVZX. Si AX es un número de 16 bits con signo, la instrucción CWD (**convierte palabra en doble palabra**) se extiende con signo a un número de 32 bits con signo. Si está disponible el microprocesador 80386 o superior, también puede usarse la instrucción MOVSX para extender con signo un número.

EJEMPLO 5-15

```
0000  B8  FF9C          MOV    AX,-100        ;carga un -100
0003  B9  0009          MOV    CX,9           ;carga +9
0006  99                CWD                   ;extiende con signo
0007  F7  F9            IDIV   CX
```

El ejemplo 5-15 muestra la división de dos números con signo de 16 bits. Aquí, el –100 en AX se divide entre el +9 en CX. La instrucción CWD convierte el –100 en AX a 100 en DX-AX antes de la división. Después de la división, los resultados aparecen en DX-AX como un cociente de –11 en AX y un residuo de –1 en DX.

División

EJEMPLO 5-15

```
0000  B8  FF9C          MOV     AX,-100         ;carga un -100
0003  B9  0009          MOV     CX,9            ;carga +9
0006  99                CWD                     ;extiende con signo
0007  F7  F9            IDIV    CX
```

El ejemplo 5-15 muestra la división de dos números con signo de 16 bits. Aquí, el –100 en AX se divide entre el +9 en CX. La instrucción CWD convierte el –100 en AX a 100 en DX-AX antes de la división. Después de la división, los resultados aparecen en DX-AX como un cociente de –11 en AX y un residuo de –1 en DX.

División de 32 bits. Los procesadores del 80386 al Pentium 4 realizan la división de 32 bits en números con o sin signo. El contenido de 64 bits de EDX-EAX se divide entre el operando especificado

TABLA 5-12 Ejemplos de instrucciones de división de 16 bits.

Lenguaje ensamblador	*Operación*
DIV CX	DX-AX se divide entre CX; el cociente sin signo está en AX y el residuo sin signo está en DX.
IDIV SI	DX-AX se divide entre SI; el cociente con signo está en AX y el residuo con signo está en DX.
DIV NUMERO	DX-AX se divide entre el contenido tipo palabra de la posición de memoria NUMERO del segmento de datos; el cociente sin signo está en AX y el residuo sin signo está en DX.

División

División de 16 bits. La división de dieciséis bits es similar a la división de 8 bits, sólo que en vez de dividir entre AX, el número de 16 bits se divide entre DX-AX, un dividendo de 32 bits. Después de una división de 16 bits, el cociente aparece en AX y el residuo aparece en DX. La tabla 5-12 lista algunas de las instrucciones de división de 16 bits.

Al igual que en la división de 8 bits, los números deben convertirse con frecuencia a la forma apropiada para el dividendo. Si se coloca un número de 16 bits sin signo en AX, DX debe volverse cero. En los microprocesadores 80386 y superiores el número se extiende con ceros mediante el uso de la instrucción MOVZX. Si AX es un número de 16 bits con signo, la instrucción CWD (**convierte palabra en doble palabra**) se extiende con signo a un número de 32 bits con signo. Si está disponible el microprocesador 80386 o superior, también puede usarse la instrucción MOVSX para extender con signo un número.

EJEMPLO 5-15

```
0000  B8  FF9C          MOV     AX,-100         ;carga un -100
0003  B9  0009          MOV     CX,9            ;carga +9
0006  99                CWD                     ;extiende con signo
0007  F7  F9            IDIV    CX
```

El ejemplo 5-15 muestra la división de dos números con signo de 16 bits. Aquí, el –100 en AX se divide entre el +9 en CX. La instrucción CWD convierte el –100 en AX a 100 en DX-AX antes de la división. Después de la división, los resultados aparecen en DX-AX como un cociente de –11 en AX y un residuo de –1 en DX.

Gracias

MANEJO DE HILERAS

Instrucciones de manejo de Hileras

El conjunto de instrucciones IA-32 tiene cinco grupos de instrucciones para procesar arreglos de bytes, palabras y dobles palabras. Aunque se llaman *primitivas de cadenas*, no se limitan a los arreglos de cadenas. Cada instrucción en la tabla 9-1 utiliza en forma implícita a ESI, EDI o ambos registros para direccionar la memoria. Las referencias al acumulador implican el uso de AL, AX o EAX, dependiendo del tamaño de los datos de la instrucción. Las primitivas de cadenas se ejecutan con eficiencia, ya que se repiten e incrementan los índices de los arreglos de manera automática.

Tabla 9-1 Instrucciones primitivas de cadenas.

Instrucción	Descripción
MOVSB, MOVSW, MOVSD	**Mover datos de cadena:** copia los datos de la memoria direccionada por ESI a la memoria direccionada por EDI
CMPSB, CMPSW, CMPSD	**Comparar cadenas:** compara el contenido de dos ubicaciones de memoria direccionadas por ESI y EDI
SCASB, SCASW, SCASD	**Explorar cadena:** compara el acumulador (AL, AX o EAX) con el contenido de la memoria direccionada por EDI
STOSB, STOSW, STOSD	**Almacenar datos de cadena:** almacena el contenido del acumulador en la memoria direccionada por EDI
LODSB, LODSW, LODSD	**Cargar acumulador desde cadena:** carga la memoria direccionada por ESI al acumulador

Instrucciones de manejo de Hileras: Modo protegido vrs modo real

En los programas en modo protegido, ESI es de manera automática un desplazamiento en el segmento direccionado por DS; y EDI es de manera automática un desplazamiento en el segmento diseccionado por ES. DS y ES siempre se establecen con el mismo valor, y no se pueden cambiar. Por otro lado, en el modo de direccionamiento real, los programadores de ASM manipulan con frecuencia a ES y DS.

En el modo de direccionamiento real, las primitivas de cadenas usan los registros SI y DI para direccionar la memoria. SI es un desplazamiento desde DS, y DI es un desplazamiento desde ES. Por lo general, ES se establece al mismo valor de segmento que DS, al principio de main:

```
main PROC
     mov  ax,@data        ; obtiene direccionamiento del seg de datos
     mov  ds,ax           ; inicializa DS
     mov  es,ax           ; inicializa ES
```

Instrucciones de manejo de Hileras: Prefijo de repetición

Uso de un prefijo de repetición Por sí sola, una instrucción de primitiva de cadena sólo procesa un solo valor de memoria o un par de valores. Si agregamos un *prefijo de repetición*, la instrucción se repite usando a ECX como contador. El prefijo de repetición nos permite procesar un arreglo completo mediante una sola instrucción. Se utilizan los siguientes prefijos de repetición:

REP	Repite mientras que ECX > 0
REPZ, REPE	Repite mientras la bandera Cero esté en uno y ECX > 0
REPNZ, REPNE	Repite mientras la bandera Cero esté en cero y ECX > 0

Ejemplo: copiar una cadena En el siguiente ejemplo, MOVSB se mueve 10 bytes a partir de **cadena1**, hacia **cadena2**. El prefijo de repetición primero evalúa ECX > 0 antes de ejecutar la instrucción MOVSB.

Si ECX = 0, la instrucción se ignora y el control pasa a la siguiente línea en el programa. Si ECX > 0, ECX se decrementa y la instrucción se repite:

```
cld                                ; borra la bandera Dirección
mov  esi,OFFSET cadena1            ; ESI apunta al origen
mov  edi,OFFSET cadena2            ; EDI apunta al destino
mov  ecx,10                        ; establece el contador a 10
rep  movsb                         ; se mueve 10 bytes
```

ESI y EDI se incrementan de manera automática cuando MOVSB se repite. Este comportamiento se controla mediante la bandera Dirección de la CPU.

Instrucciones de manejo de Hileras: Bandera de dirección

Bandera Dirección Las instrucciones de primitiva de cadenas incrementan o decrementan a ESI y EDI, según el estado de la bandera Dirección (vea la tabla 9-2). Esta bandera puede modificarse en forma explícita, usando las instrucciones CLD y STD:

```
CLD         ; borra la bandera Dirección (dirección de avance)
STD         ; activa la bandera Dirección (dirección de retroceso)
```

Si olvidamos activar la bandera Dirección antes de una instrucción primitiva de cadena, podemos tener grandes problemas. El código resultante se ejecuta de manera inconsistente, según el estado arbitrario de la bandera Dirección.

TABLA 9-2 Uso de la bandera Dirección en instrucciones primitivas de cadena.

Valor de la bandera Dirección	Efecto sobre ESI y EDI	Secuencia de direcciones
Cero	Se incrementa	Bajo-alto
Uno	Se decrementa	Alto-bajo

Instrucciones de manejo de Hileras: MOVER HILERA

Las instrucciones MOVSB, MOVSW y MOVSD copian datos de la ubicación de memoria a la que apunta ESI, hasta la ubicación de memoria a la que apunta EDI. Los dos registros se incrementan o decrementan en forma automática (según el valor de la bandera Dirección):

MOVSB	Mueve (copia) bytes
MOVSW	Mueve (copia) palabras
MOVSD	Mueve (copia) dobles palabras

Puede utilizar un prefijo de repetición con MOVSB, MOVSW y MOVSD. La bandera Dirección determina si ESI y EDI se van a incrementar o a decrementar. El tamaño del incremento o decremento se muestra en la siguiente tabla:

Instrucción	Valor que se agrega o se resta a ESI y EDI
MOVSB	1
MOVSW	2
MOVSD	4

Instrucciones de manejo de Hileras: Ejemplo MOVER HILERA

Ejemplo: copiar arreglo de dobles palabras Suponga que queremos copiar 20 enteros tipo doble palabra, de **origen** a **destino**. Una vez que se copia el arreglo, ESI y EDI apuntan una posición (4 bytes) más lejos del final de cada arreglo:

```
.data
origen  DWORD 20 DUP(0FFFFFFFFh)
destino DWORD 20 DUP(?)
.code
cld                                    ; dirección = avance
mov  ecx,LENGTHOF origen               ; establece contador REP
mov  esi,OFFSET origen                 ; ESI apunta al origen
mov  edi,OFFSET destino                ; EDI apunta al destino
rep  movsd                             ; copia dobles palabras
```

Instrucciones de manejo de Hileras: COMPARAR HILERA

Las instrucciones CMPSB, CMPSW y CMPSD comparan un operando de memoria al que apunta ESI, con un operando de memoria al que apunta EDI:

CMPSB	Compara bytes
CMPSW	Compara palabras
CMPSD	Compara dobles palabras

Puede usar un prefijo de repetición con CMPSB, CMPSW y CMPSD. La bandera Dirección determina el incremento o decremento de ESI y EDI.

Forma explícita de CMPS: en otra forma de la instrucción de comparación de cadenas llamada *forma explícita*, se suministran dos operandos indirectos. El operando PTR aclara los tamaños de los operandos. Por ejemplo,

```
cmps DWORD PTR [esi],[edi]
```

Pero CMPS es engañoso, ya que el ensamblador nos permite suministrar operandos erróneos:

```
cmps DWORD PTR [eax],[ebx]
```

Sin importar qué operandos se utilicen, CMPS compara el contenido de la memoria a la que apunta ESI con la memoria a la que apunta EDI. Observe que el orden de los operandos en CMPS es opuesto a la instrucción CMPS, más conocida:

```
CMP destino,origen
CMPS origen,destino
```

He aquí otra forma de recordar la diferencia: CMP implica restar el *origen* del *destino*. CMPS implica restar el *destino* del *origen*. C Se sugiere evitar el uso de CMPS y utilizar las versiones específicas (CMPSB, CMPSW, CMPSD).

Instrucciones de manejo de Hileras: COMPARAR HILERA

Ejemplo: comparación de dobles palabras Suponga que desea comparar un par de dobles palabras mediante el uso de CMPSD. En el siguiente ejemplo, **origen** tiene un valor menor que **destino**. Cuando se ejecuta JA, el salto condicional no se lleva a cabo; en vez de ello se ejecuta la instrucción JMP:

```
.data
origen  DWORD 1234h
destino DWORD 5678h
.code
mov  esi,OFFSET origen
mov  edi,OFFSET destino
cmpsd                              ; compara dobles palabras
ja   L1                            ; salta si origen > destino
jmp  L2                            ; salta, ya que origen <= destino
```

Para comparar varias dobles palabras, borre la bandera Dirección (dirección de avance), inicialice ECX como contador y utilice un prefijo repetido con CMPSD:

```
mov  esi,OFFSET origen
mov  edi,OFFSET destino
cld                                ; dirección = avance
mov  ecx,LENGTHOF origen           ; contador de repetición
repe cmpsd                         ; repite mientras sea igual
```

El prefijo REPE repite la comparación e incrementa a ESI y EDI de manera automática hasta que ECX sea igual a cero, o hasta que un par de dobles palabras sea distinto.

Instrucciones de manejo de Hileras: Ejemplo COMPARA dos HILERAs

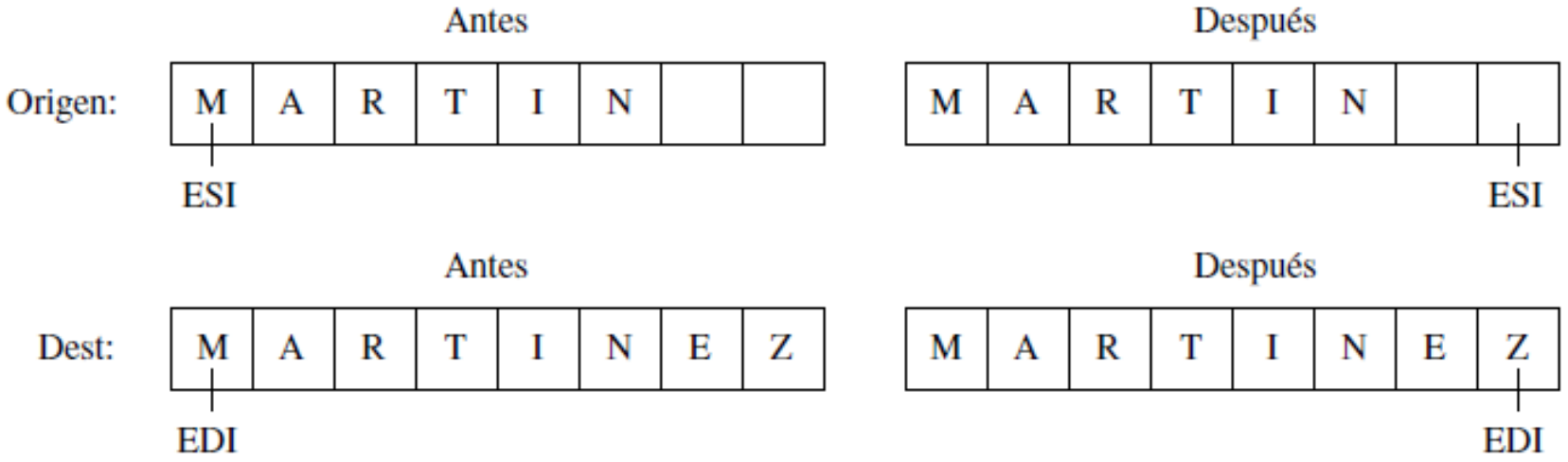
El siguiente programa utiliza CMPSB para comparar dos cadenas de igual longitud. El prefijo REPE hace que CMPSB continúe incrementando a ESI y EDI, y que compare los caracteres uno a uno hasta encontrar una diferencia entre las dos cadenas:

```
TITLE Comparación de cadenas(Cmpsb.asm)

; Este programa utiliza a CMPSB para comparar dos cadenas
; de la misma longitud.

INCLUDE Irvine32.inc

.data
origen BYTE "MARTIN  "
dest   BYTE "MARTINEZ"
cad1   BYTE "La cadena de origen es mas chica",0dh,0ah,0
cad2   BYTE "La cadena de origen no es mas chica",0dh,0ah,0

.code
main PROC
     cld                              ; dirección = avance
     mov   esi,OFFSET origen
     mov   edi,OFFSET dest
     mov   ecx,LENGTHOF origen
     repe  cmpsb
     jb    origen_mas_chico
     mov   edx,OFFSET Cad2
     jmp   listo
origen_mas_chico:
     mov   edx,OFFSET cad1
listo:
     call  WriteString
     exit
main ENDP
END main
```

Instrucciones de manejo de Hileras: Ejemplo COMPARA dos HILERAs

Al utilizar los datos de prueba proporcionados, aparece el mensaje “La cadena de origen es mas chica”. En la figura 9-1, ESI y EDI quedan apuntando una posición más lejos del punto en el que se encontró que las dos cadenas diferían. Si las cadenas hubieran sido idénticas, ESI y EDI hubieran quedado apuntando una posición más lejos del final de sus respectivas cadenas.

FIGURA 9–1 Comparación de dos cadenas mediante el uso de CMPSB.



Instrucciones de manejo de Hileras: ESCUDRIÑA HILERA

Las instrucciones SCASB, SCASW y SCASD comparan un valor en AL/AX/EAX con un byte, palabra o doble palabra, respectivamente, la cual está direccionada por EDI. Las instrucciones son útiles cuando se busca un valor individual en una cadena o arreglo. Si se combinan con el prefijo REPE (o REPZ), la cadena o arreglo se explora mientras ECX $>$ 0, y el valor en AL/AX/EAX coincida con cada valor subsiguiente en memoria. El prefijo REPNE explora hasta que AL/AX/EAX coincida con un valor en memoria, o cuando ECX $=$ 0.

Explorar en busca de un carácter que coincida En el siguiente ejemplo buscamos la letra F en la cadena **alfa**. Si se encuentra la letra, EDI apunta una posición más allá del carácter que coincidió. Si no se encuentra la letra, JNZ termina el programa:

```
.data
alfa  BYTE "ABCDEFGH",0
.code
mov   edi,OFFSET alfa              ; EDI apunta a la cadena
mov   al,'F'                       ; busca la letra F
mov   ecx,LENGTHOF alfa            ; establece la cuenta de búsqueda
cld                                ; dirección = avance
repne scasb                        ; repite mientras no sea igual
jnz   salir                        ; termina si no se encontró la letra
dec   edi                          ; se encontró: retrocede EDI
```

JNZ se agregó después del ciclo para evaluar la posibilidad de que el ciclo se detuviera debido a ECX $=$ 0, y que no se encontrara el carácter en AL.

Instrucciones de manejo de Hileras: ALMACENA HILERA

Las instrucciones STOSB, STOSW y STOSD almacenan en memoria el contenido de AL/AX/EAX, respectivamente, en el desplazamiento al que apunta EDI. EDI se incrementa o decrementa con base en el estado de la bandera Dirección. Cuando se utilizan con el prefijo REP, estas instrucciones son útiles para rellenar todos los elementos de una cadena o arreglo con un solo valor. Por ejemplo, el siguiente código inicializa cada byte en **cadena1** con 0FFh:

```
.data
Cuenta = 100
Cadena1  BYTE Cuenta DUP(?)
.code

mov  al,0FFh                    ; valor a guardar
mov  edi,OFFSET cadena1         ; EDI apunta al destino
mov  ecx,Cuenta                 ; cuenta de caracteres
cld                             ; dirección = avance
rep  stosb                      ; llena con el contenido de AL
```

Instrucciones de manejo de Hileras: CARGAR HILERA

Las instrucciones LODSB, LODSW y LODSD cargan un byte o palabra de la memoria en ESI, hacia AL/AX/EAX, respectivamente. ESI se incrementa o decrementa según el estado de la bandera Dirección. El prefijo REP se utiliza raras veces con LODS, ya que cada nuevo valor que se carga en el acumulador sobrescribe su contenido anterior. En vez de ello, LODS se utiliza para cargar un solo valor. En el siguiente ejemplo, LODSB sustituye a las dos instrucciones siguientes (suponiendo que la bandera Dirección esté en cero):

```
mov  al,[esi]                    ; mueve byte hacia AL
inc  esi                         ; apunta al siguiente byte
```

Ejemplo de multiplicación de arreglos El siguiente programa multiplica cada elemento de un arreglo de dobles palabras por un valor constante. LODSD y STOSD trabajan en conjunto:

```
TITLE Multiplicación de un arreglo          (Mult.asm)

; Este programa multiplica cada elemento de un arreglo
; de enteros de 32 bits por un valor constante.

INCLUDE Irvine32.inc
.data
arreglo DWORD 1,2,3,4,5,6,7,8,9,10          ; datos de prueba
multiplicador DWORD 10                      ; datos de prueba

.code
main PROC
     cld                                    ; dirección = avance
     mov   esi,OFFSET arreglo               ; índice de origen
     mov   edi,esi                          ; índice de destino
     mov   ecx,LENGTHOF arreglo             ; contador de ciclo

L1:  lodsd                                  ; copia [ESI] hacia EAX
     mul   multiplicador                    ; multiplica por un valor
     stosd                                  ; guarda EAX en [EDI]
     loop  L1

     exit
main ENDP
END main
```

Gracias

Modos de Direccionamiento

Modos de Direccionamiento

Tipo	Instrucción	Origen	Generación de direcciones	Destino
De registro	MOV AX,BX	De registro BX		De registro AX
Inmediato	MOV CH,3AH	Datos 3AH		De registro CH
Directo	MOV [1234H],AX	De registro AX	DS × 10H + DISP 10000H + 1234H	Dirección de memoria 11234H
De registro indirecto	MOV [BX],CL	De registro CL	DS × 10H + BX 10000H + 0300H	Dirección de memoria 10300H
De base más índice	MOV [BX+SI],BP	De registro SP	DS × 10H + BX + SI 10000H + 0300H + 0200H	Dirección de memoria 10500H
De registro relativo	MOV CL,[BX+4]	Dirección de memoria 10304H	DS × 10H + BX + 4 10000H + 0300H + 4	De registro CL
De base relativa más índice	MOV ARRAY[BX+SI],DX	De registro DX	DS × 10H + ARRAY + BX + SI 10000H + 1000H + 0300H + 0200H	Dirección de memoria 11500H
De índice escalado	MOV [EBX+2 × ESI],AX	De registro AX	DS × 10H + EBX + 2 × ESI 10000H + 00000300H + 00000400H	Dirección de memoria 10700H

Observaciones: EBX = 00000300H, ESI = 00000200H, ARRAY = 1000H, y DS = 1000H

Modos de Direccionamiento

Direccionamiento de registro	El direccionamiento de registro transfiere una copia de un byte o palabra del registro de origen, o el contenido de una posición de memoria al registro de destino o posición de memoria. (Ejemplo: la instrucción MOV CX,DX copia el contenido del tamaño de una palabra del registro DX al registro CX.) En los microprocesadores 80386 y superiores, una doble palabra puede transferirse desde el registro o posición de memoria de origen hasta el registro o posición de memoria de destino. (Ejemplo: la instrucción MOV ECX,EDX copia el contenido del tamaño de una doble palabra del registro EDX al registro ECX.)

Modos de Direccionamiento

Direccionamiento inmediato	Este modo de direccionamiento transfiere el origen (datos inmediatos tipo byte o palabra) al registro o posición de memoria de destino. (Ejemplo: la instrucción MOV AL,22H copia el número 22H del tamaño de un byte al registro AL.) En los microprocesadores 80386 y superiores puede transferirse una doble palabra de datos inmediatos hacia un registro o una posición de memoria. (Ejemplo: la instrucción MOV EBX, 12345678H copia el número 12345678H del tamaño de una doble palabra al registro EBX de 32 bits.)

Modos de Direccionamiento

Direccionamiento directo	Este modo de direccionamiento mueve un byte o palabra entre una posición de memoria y un registro. El conjunto de instrucciones no soporta una transferencia de memoria a memoria, excepto con la instrucción MOVS. (Ejemplo: la instrucción MOV CX,LISTA copia el contenido del tamaño de una palabra de la posición de memoria LISTA al registro CX.) En los microprocesadores 80386 y superiores también puede direccionarse una posición de memoria del tamaño de una doble palabra. (Ejemplo: la instrucción MOV ESI,LISTA copia un número de 32 bits, almacenado en cuatro bytes consecutivos de memoria, de la posición LIST al registro ESI.)

Modos de Direccionamiento

Direccionamiento de registro indirecto

Este modo de direccionamiento transfiere un byte o una palabra entre un registro y una posición de memoria direccionados por un registro índice o base. Los registros índice y base son BP, BX, DI y SI. (Ejemplo: la instrucción MOV AX,[BX] copia los datos del tamaño de una palabra desde la dirección de desplazamiento del segmento de datos indizado por BX, hacia el registro AX.) En los microprocesadores 80386 y superiores se transfiere un byte, una palabra o una doble palabra entre un registro y una posición de memoria direccionada por cualquier registro: EAX, EBX, ECX, EDX, EBP, EDI o ESI. (Ejemplo: la instrucción MOV AL,[ECX] carga el registro AL de la dirección de desplazamiento del segmento de datos seleccionada por el contenido de ECX.)

Modos de Direccionamiento

Direccionamiento de base más índice Este modo de direccionamiento transfiere un byte o una palabra entre un registro y la posición de memoria direccionada por un registro base (BP o BX) más un registro índice (DI o SI). (Ejemplo: la instrucción MOV [BX+DI],CL copia el contenido del tamaño de un byte del registro CL en la posición de memoria del segmento de datos direccionada por BX más DL.) En los microprocesadores 80386 y superiores pueden combinarse dos registros cualesquiera (EAX, EBX, ECX, EDX, EBP, EDI o ESI) para generar la dirección de memoria. (Ejemplo: la instrucción MOV [EAX+EBX],CL copia el contenido del tamaño de un byte del registro CL en la posición de memoria del segmento de datos direccionada por EAX más EBX.)

Modos de Direccionamiento

Direccionamiento de registro relativo	Este modo de direccionamiento mueve un byte o una palabra entre un registro y la posición de memoria direccionada por un registro índice o base más un desplazamiento. (Ejemplo: MOV AX,[BX+4] o MOV AX,ARRAY[BX]. La primera instrucción carga AX en base a la dirección del segmento de datos formada por BX más 4. La segunda instrucción carga AX en base a la posición de memoria del segmento de datos en ARRAY más el contenido de BX.) Para direccionar memoria, los microprocesadores 80386 y superiores utilizan cualquier registro de 32 bits excepto ESP. (Ejemplo: MOV AX,[ECX+4] o MOV AX,ARRAY[EBX]. La primera instrucción carga AX en base a la dirección del segmento de datos formada por ECX más 4. La segunda instrucción carga AX en base a la posición de memoria ARRAY del segmento de datos más el contenido de EBX.)

Modos de Direccionamiento

Direccionamiento de base relativa más índice	Este modo de direccionamiento transfiere un byte o una palabra entre un registro y la posición de memoria direccionada por un registro base y un registro índice más un desplazamiento. (Ejemplo: MOV AX, ARRAY[BX+DI] o MOV AX,[BX+DI+4]. Estas instrucciones cargan AX en base a la posición de memoria de un segmento de datos. La primera instrucción utiliza una dirección formada mediante la suma de ARRAY, BX y DI, y la segunda mediante la suma de BX, DI y 4.) En los microprocesadores 80386 y superiores, MOV EAX,ARRAY[EBX+ECX] carga EAX en base a la posición de memoria del segmento de datos a la que se accede mediante la suma de ARRAY, EBX y ECX.

Modos de Direccionamiento

Direccionamiento de índice escalado

Este modo de direccionamiento sólo está disponible en los microprocesadores del 80386 al Pentium 4. El segundo registro de un par de registros se modifica mediante el factor de escala de 2×, 4× u 8× para generar la dirección de memoria del operando. (Ejemplo: una instrucción MOV EDX,[EAX+4*EBX] carga EDX en base a la posición de memoria del segmento de datos direccionada por EAX más cuatro veces EBX.) El uso de una escala permite el acceso a los datos de un arreglo de memoria de tipo palabra (2×), doble palabra (4×) o palabra cuádruple (8×). También existe un factor de escala de 1×, pero por lo general es implícito y no aparece explícitamente en la instrucción. La instrucción MOV AL,[EBX+ECX] es un ejemplo en el que el factor de escala es uno. Esta instrucción puede escribirse de manera alterna como MOV AL,[EBX+1*ECX]. Otro ejemplo es la instrucción MOV AL,[2*EBX], que utiliza sólo un registro escalado para direccionar memoria.