

## **Repertorio de instrucciones x86-16bits**

### **Índice**

---

- 1. Generalidades**
- 2. Instrucciones de transferencia**
- 3. Instrucciones de proceso**
- 4. Instrucciones de bifurcación**
- 5. Otras instrucciones**

## **Repertorio de instrucciones x86-16bits**

### **1. Generalidades**

---

- No se pueden realizar operaciones donde ambos operandos residan en memoria**
- Las instrucciones de transferencia NO ACCEDEN al registro de estado**
- Las instrucciones de proceso ESCRIBEN el registro de estado**
- Las instrucciones de salto LEEN el registro de estado**

## Repertorio de instrucciones x86-16bits

### 1. Generalidades

---

- Existen 2 tipos de sintaxis
  - ➔ INTEL
  - ➔ ATT
- Nosotros usaremos la sintaxis INTEL
  - ➔ El primer operando es el DESTINO
  - ➔ El segundo operando es el FUENTE
- En las operaciones de proceso, el operando DESTINO pierde su valor inicial siendo reemplazado por el resultado

© Rafael Rico López

3/145

## Repertorio de instrucciones x86-16bits

### 2. Instrucciones de transferencia

---

#### 2. Instrucciones de transferencia

##### Índice

1. Movimiento de datos
2. Extensión de signo
3. Transfiriendo punteros
4. Transferencias con la pila
5. Entrada/salida

© Rafael Rico López

4/145

## Repertorio de instrucciones x86-16bits

### 2.1. Movimiento de datos

**MOV {reg/mem},{reg/mem/inmediato}**

- Transfiere un byte o una palabra desde el operando fuente al destino

**op. destino ← op. fuente**

- El operando fuente no se destruye
- Ambos operandos deben ser del mismo tamaño

© Rafael Rico López

5/145

## Repertorio de instrucciones x86-16bits

### 2.1. Movimiento de datos

- Ejemplo:

**MOV AX,FFFF ; AX = FFFF h**

**MOV BX,1234 ; BX = 1234 h**

**MOV AX,BX ; AX = 1234 h**

**; BX = 1234 h**

© Rafael Rico López

6/145

## Repertorio de instrucciones x86-16bits

### 2.1. Movimiento de datos

- **Restricciones:**

- ➔ No se pueden mover datos entre dos elementos de memoria; hay que utilizar un registro intermedio

```
MOV AX, mem1  
MOV mem2, AX
```

- ➔ No se puede mover un inmediato a un registro de segmento; hay que usar un registro intermedio

```
MOV AX, 1234h      ; AX = 1234h  
MOV DS, AX         ; DS = 1234h
```

- ➔ El registro de segmento CS no puede ser destino

© Rafael Rico López

7/145

## Repertorio de instrucciones x86-16bits

### 2.1. Movimiento de datos

**XCHG {reg/mem},{reg/mem}**

- Intercambia el contenido de los operandos
- Es útil para evitar el uso de una variable temporal
- Ejemplo:

```
MOV AX,FFFF      ; AX = FFFF h  
MOV BX,0         ; BX = 0000 h  
  
XCHG AX,BX       ; AX = 0000 h  
                ; BX = FFFF h
```

© Rafael Rico López

8/145

## Repertorio de instrucciones x86-16bits

### 2.1. Movimiento de datos

#### XLAT memoria

- Memoria es un desplazamiento sobre DS. El puntero a memoria será:

DS:memoria

prefijo: XLAT memoria

- Ahora el segmento viene dado por el prefijo. El puntero a memoria será:

prefijo:memoria

© Rafael Rico López

9/145

## Repertorio de instrucciones x86-16bits

### 2.1. Movimiento de datos

- La instrucción XLAT carga en AL el valor de una tabla de memoria
- Es útil para traducir entre sistemas de codificación
- La tabla debe ser de bytes y no puede tener más de 256 bytes; el primero tiene desplazamiento 0
- La base de la tabla se coloca en BX y el puntero en AL

$AL \leftarrow [BX+AL]$

© Rafael Rico López

10/145

## Repertorio de instrucciones x86-16bits

### 2.1. Movimiento de datos

- Ejemplo:

```
.DATA
TABLA DB 1,2,3,4,5,6,7 ;declaración

.CODE
MOV BX,OFFSET TABLA ;carga BX
MOV AL,4 ;5º valor
XLAT TABLA ;AL = 5
```

- Esto es equivalente a:

```
MOV AL,TABLA[4]
```

© Rafael Rico López

11/145

## Repertorio de instrucciones x86-16bits

### 2.1. Movimiento de datos

- Ejemplo:

➔ Esta tabla permite traducir  
códigos ASCII a EBCDIC

➔ El código ASCII sirve de  
índice

➔ Por ejemplo, el índice 32  
apunta al "2" en EBCDIC

dirección	valor
TABLA[0]	XX
TABLA[1]	XX
TABLA[30]	F0
TABLA[31]	F1
TABLA[32]	F2
TABLA[33]	F3
TABLA[34]	F4
TABLA[35]	F5

© Rafael Rico López

12/145

## Repertorio de instrucciones x86-16bits

### 2.1. Movimiento de datos

#### LAHF

- Carga los 8 bits más bajos del registro de estado (banderas de estado) en AH

$AH \leftarrow \text{banderas de estado (bits 0, 2, 4, 6, 7)}$

#### SAHF

- Recupera las banderas de estado desde AH

$\text{banderas de estado} \leftarrow AH$

© Rafael Rico López

13/145

## Repertorio de instrucciones x86-16bits

### 2.1. Movimiento de datos

- Las instrucciones de transferencia de las banderas de estado se suelen usar para mover el estado entre coprocesadores
- Para manejar el conjunto completo del registro de estado se deben usar instrucciones de transferencia con la pila

➡ PUSHF

➡ POPF

© Rafael Rico López

14/145

## Repertorio de instrucciones x86-16bits

### 2.2. Extensión de signo

- Antes de poder mover datos de diferente tamaño es necesario extender adecuadamente el signo
- El procedimiento es distinto si el número es considerado con signo o sin él, pero es el programador el que debe tenerlo en cuenta ya que la máquina no advierte la diferencia
  - ➔ Cuando el valor tiene signo se usa CBW o CWD
  - ➔ Cuando el valor es sin signo se rellena con ceros

© Rafael Rico López

15/145

## Repertorio de instrucciones x86-16bits

### 2.2. Extensión de signo

#### CBW

- Convertir byte en palabra
- Copia el bit 7 del registro AL en todo el registro AH

#### CWD

- Convertir palabra en doble palabra
- Copia el bit 15 del registro AX en el registro DX
  - ➔ Doble palabra → DX : AX

© Rafael Rico López

16/145



## Repertorio de instrucciones x86-16bits

### 2.2. Extensión de signo

- Ejemplo con signo:

```
.DATA
mem8    DB  -5           ;declaración
mem16   DW  -5           ;declaración

.CODE
MOV AL, mem8      ;carga AL = FBh
CBW              ;AX = FFFBh (-5)

MOV AX, mem16     ;carga AX = FFFBh
CWD              ;DX = FFFFh
              ;DX:AX = (-5)
```

© Rafael Rico López

17/145

## Repertorio de instrucciones x86-16bits

### 2.2. Extensión de signo

- Ejemplo sin signo:

```
.DATA
mem8    DB  251          ;declaración (FBh)
mem16   DW  251          ;declaración (FFFBh)

.CODE
MOV AL, mem8      ;carga AL = FBh(251)
XOR AH, AH        ;AX = 00FBh (251)

MOV AX, mem16     ;carga AX = FFFBh
XOR DX, DX        ;DX:AX = 0000 FFFBh
```

© Rafael Rico López

18/145

## Repertorio de instrucciones x86-16bits

### 2.3. Transfiriendo punteros

- Instrucciones para cargar punteros en registros
- Los punteros pueden ser
  - ➔ *Near* → dentro de un segmento; no exceden los 64KB
    - LEA
  - ➔ *Far* → entre segmentos; exceden los límites del segmento y se requiere una *base* y un *desplazamiento*
    - LES
    - LDS

© Rafael Rico López

19/145

## Repertorio de instrucciones x86-16bits

### 2.3. Transfiriendo punteros

LEA {reg/mem}

- Carga un puntero *near* en un registro; el puntero es la *dirección efectiva* de la posición de memoria especificada en el operando fuente
  - ➔ El operando destino puede ser cualquier registro de propósito general
  - ➔ No están permitidos los registros de segmento
  - ➔ El operando fuente es una posición de memoria especificada por cualquier modo de direccionamiento

© Rafael Rico López

20/145

## Repertorio de instrucciones x86-16bits

### 2.3. Transfiriendo punteros

- Ejemplo:
- Transfiere el desplazamiento del operando fuente al registro destino

```
LEA AX, 1234[SI]      ;si SI = 1000h
                      ;AX = 1234 + 1000
                      ;AX = 2234h
```

© Rafael Rico López

21/145

## Repertorio de instrucciones x86-16bits

### 2.3. Transfiriendo punteros

- Advertencias respecto a LEA:  

```
LEA DX,cadena
MOV DX,OFFSET cadena
```
- Dan el mismo resultado pero es más rápida la segunda ya que la posición de *cadena* en el área de datos es conocida en tiempo de ensamblado
- Usaremos LEA cuando queramos transferir un desplazamiento no conocido en tiempo de diseño:  

```
LEA DX,cadena[SI]
MOV DX,OFFSET cadena[SI] ;no funciona
```

© Rafael Rico López

22/145

## Repertorio de instrucciones x86-16bits

### 2.3. Transfiriendo punteros

**LES {reg/mem}**

- Copia un puntero *far* (32 bits) guardado en el operando fuente al registro especificado en el operando destino y al registro **ES**
  - ➔ **ES** → salva la base
  - ➔ Registro destino → salva el desplazamiento (no se aceptan los registros de segmento)
- El operando fuente es una posición de memoria de tamaño doble palabra (32 bits)

© Rafael Rico López

23/145

## Repertorio de instrucciones x86-16bits

### 2.3. Transfiriendo punteros

- Los punteros en memoria se salvan por este orden
  - ➔ Palabra de menor peso → desplazamiento
  - ➔ Palabra de mayor peso → base

- Ejemplo:

**PTR → 1234:5678**

**LES DI, PTR      ;DI = 5678h  
                     ;ES = 1234h**

dirección	valor
PTR[0]	78
PTR[1]	56
PTR[2]	34
PTR[3]	12

© Rafael Rico López

24/145

## Repertorio de instrucciones x86-16bits

### 2.3. Transfiriendo punteros

**LDS {reg/mem}**

- Copia un puntero *far* (32 bits) guardado en el operando fuente al registro especificado en el operando destino y al registro DS
  - ➔ DS → salva la base
  - ➔ Registro destino → salva el desplazamiento (no se aceptan los registros de segmento)
- El operando fuente es una posición de memoria de tamaño doble palabra (32 bits)

© Rafael Rico López

25/145

## Repertorio de instrucciones x86-16bits

### 2.3. Transfiriendo punteros

- Ejemplo:

**PTR → 1234:5678**

**LDS SI,PTR      ;SI = 5678h  
                     ;DS = 1234h**

dirección	valor
PTR[0]	78
PTR[1]	56
PTR[2]	34
PTR[3]	12

© Rafael Rico López

26/145

## Repertorio de instrucciones x86-16bits

### 2.3. Transfiriendo punteros

- Ejemplo:

```
.DATA
cadena    DB "Hola mundo"          ;cadena
pcadena   DD cadena                ;salvo el puntero
array     DB 100 DUP(?)            ;reservo array
parray    DD array

.CODE
LES DI,pcadena    ;ES:DI←cadena
LDS SI,parray     ;DS:SI←array
```

© Rafael Rico López

27/145

## Repertorio de instrucciones x86-16bits

### 2.4. Transferencias con la pila

- La pila es un área de memoria de acceso secuencial tipo LIFO gobernada por el puntero **SP** y usada para almacenar datos temporalmente:
  - ➔ Paso de parámetros a procedimientos
  - ➔ Variables locales a procedimientos
  - ➔ Salvaguarda de registros cuando deben ser utilizados por otra variable
- En el x86-16bits el **SP** comienza en las posiciones más altas y avanza hacia las más bajas

© Rafael Rico López

28/145

## Repertorio de instrucciones x86-16bits

### 2.4. Transferencias con la pila

- Las instrucciones que trabajan con la pila sólo especifican un operando ya que el otro es implícito (la cima de la pila referenciada por **SP**)
- Estas instrucciones también actúan de manera implícita sobre el puntero de pila (**SP**)
  - ➔ Decrementándolo cuando introducen datos
  - ➔ Incrementándolo cuando sacan datos
- Las transferencias con de tamaño palabra (16 bits)
  - ➔ El **SP** se actualiza de 2 en 2

© Rafael Rico López

29/145

## Repertorio de instrucciones x86-16bits

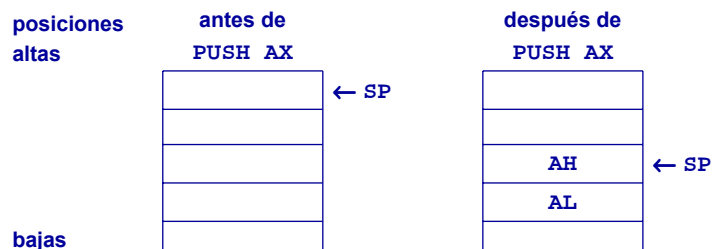
### 2.4. Transferencias con la pila

**PUSH {reg/mem}**

- Poner palabra en la pila
- Decrementa el **SP** en 2 y coloca el operando en la pila
  - ➔ El operando nunca puede ser **CS**

$SP \leftarrow SP - 2$

$SS : SP \leftarrow \text{operando}$



© Rafael Rico López

30/145

## Repertorio de instrucciones x86-16bits

### 2.4. Transferencias con la pila

- Ejemplo:

`PUSH AX` ;pone AX en la cima de la pila

- Es equivalente a:

`SUB SP,2` ;  $SP \leftarrow SP - 2$   
`MOV [SP],AX` ;  $SS:SP \leftarrow AX$

© Rafael Rico López

31/145

## Repertorio de instrucciones x86-16bits

### 2.4. Transferencias con la pila

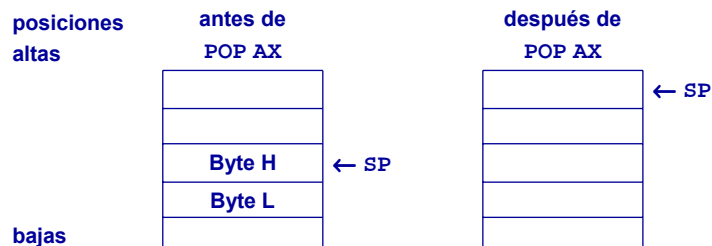
**POP {reg/mem}**

- Sacar palabra de la pila
- Copia el dato de la pila en el operando especificado e incrementa el SP en 2

➔ El operando nunca puede ser CS

$\text{operando} \leftarrow SS:SP$

$SP \leftarrow SP + 2$



© Rafael Rico López

32/145



## Repertorio de instrucciones x86-16bits

### 2.4. Transferencias con la pila

- Ejemplo:

`POP AX ;AX ← valor cima de la pila`

- Es equivalente a:

`MOV AX,[SP] ; AX ← SS:SP`  
`ADD SP,2 ; SP ← SP + 2`

© Rafael Rico López

33/145

## Repertorio de instrucciones x86-16bits

### 2.4. Transferencias con la pila

#### **PUSHF**

- Transfiere el registro de estado completo a la pila

$SP \leftarrow SP - 2$   
 $SS:SP \leftarrow \text{reg. estado}$

#### **POPF**

- Carga el registro de estado completo con el contenido de la cima de la pila

$\text{reg. estado} \leftarrow SS:SP$   
 $SP \leftarrow SP + 2$

© Rafael Rico López

34/145

## Repertorio de instrucciones x86-16bits

### 2.5. Entrada/salida

- Los mapas de memoria y de entrada/salida en máquinas x86-16bits son disjuntos
- Cuando se emite una dirección en el bus de direcciones es necesario especificar si es de memoria o de E/S
- Por esto contamos con instrucciones específicas de E/S
  - ➔ IN → señal  $\overline{IO/\overline{M}}$  = 1 → señal RD activa
  - ➔ OUT → señal  $\overline{IO/\overline{M}}$  = 1 → señal WD activa

© Rafael Rico López

35/145

## Repertorio de instrucciones x86-16bits

### 2.5. Entrada/salida

IN Acc, {puerto/DX}

- Carga el acumulador con un valor leído en un puerto de E/S especificado por el operando fuente
  - ➔ El puerto puede ser un inmediato de tamaño byte (puertos 0 – 255)
  - ➔ Por encima de este puerto hay que darlo como DX
- El tamaño de la transferencia viene dado por Acc:
  - ➔ Si es AX → tamaño palabra
  - ➔ Si es AL → tamaño byte

© Rafael Rico López

36/145

## Repertorio de instrucciones x86-16bits

### 2.5. Entrada/salida

**OUT {puerto/DX},Acc**

- **Escribe el contenido del acumulador en el puerto especificado**
  - ➔ El puerto puede ser un inmediato de tamaño byte (puertos 0 – 255)
  - ➔ Por encima de este puerto hay que darlo como DX
- **El tamaño de la transferencia viene dado por Acc:**
  - ➔ Si es AX → tamaño palabra
  - ➔ Si es AL → tamaño byte

© Rafael Rico López

37/145

## Repertorio de instrucciones x86-16bits

### 2.5. Entrada/salida

- **Normalmente las E/S se realizan mediante llamadas al S.O. (que a su vez realiza llamadas al BIOS)**
  - ➔ INT 21h → bajo D.O.S.
  - ➔ API → bajo Windows
- **Sin embargo, las instrucciones de E/S proporcionan un método para hacer E/S directamente**
  - ➔ ¡OJO! pueden provocar problemas de portabilidad

© Rafael Rico López

38/145

## Repertorio de instrucciones x86-16bits

### 3. Instrucciones de proceso

#### 3. Instrucciones de proceso

##### Índice

1. Operaciones lógicas
2. Desplazamientos y rotaciones
3. Suma
4. Resta
5. Multiplicación
6. División
7. Operaciones en BCD

© Rafael Rico López

39/145

## Repertorio de instrucciones x86-16bits

### 3.1. Operaciones lógicas

- Las instrucciones lógicas realizan operaciones booleanas sobre bits individuales
  - ➔ Cada resultado en el bit *i-ésimo* sólo depende de los bits *i-ésimos* de los operandos de entrada
    - ➔ Son las más rápidas
  - ➔ No conllevan propagaciones de acarreo, etc.
    - ➔ No son función del peso ( $\neq f(\text{peso})$ )

© Rafael Rico López

40/145

## Repertorio de instrucciones x86-16bits

### 3.1. Operaciones lógicas

- El repertorio del x86-16bits da soporte a las operaciones lógicas AND, OR, XOR entre dos operandos (diádicas) y a la NOT de un operando (monádica)

X	Y	diádicas			monádica
		X AND Y	X OR Y	X XOR Y	NOT X
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

© Rafael Rico López

41/145

## Repertorio de instrucciones x86-16bits

### 3.1. Operaciones lógicas

- Se suelen usar combinando un operando con una “máscara”
  - ➔ La “máscara” tiene diferentes formas dependiendo de la operación
  - ➔ Se utiliza para modificar o extraer información de unos bits y obviar la de otros
- Las instrucciones no han de confundirse con los operadores
  - ➔ Las primeras operan en tiempo de ejecución
  - ➔ Los segundos son órdenes para el ensamblador
  - ➔ Se distinguen por el contexto

© Rafael Rico López

42/145

## Repertorio de instrucciones x86-16bits

### 3.1. Operaciones lógicas

**AND {reg/mem},{reg/mem/inmediato}**

- Realiza la operación lógica AND
- Se puede usar para poner a cero un bit independientemente de su valor actual
  - ➔ La máscara contendrá un 0 allá donde queramos colocar un cero y un 1 donde queremos dejar intacto el bit original

© Rafael Rico López

43/145

## Repertorio de instrucciones x86-16bits

### 3.1. Operaciones lógicas

- Ejemplo:

```
MOV AX, 035h      ; 0011 0101
AND AX, 0FBh      ;and 1111 1011
                  ; 0011 0001
AND AX, 0F8h      ; 0011 0001
                  ;and 1111 1000
                  ; 0011 0000

MOV AH, 7         ;servicio 7 de INT 21h
INT 21h           ;entrada carácter sin eco
AND AL, 1101 1111b ;convierte a Mayúsculas
CMP AL, 'Y'       ;¿es 'Y'?
JE YES            ;si es 'Y' salta a la rutina
                  ;si no es 'Y' continúa
YES:              : : : ;rutina
```

© Rafael Rico López

44/145

## Repertorio de instrucciones x86-16bits

### 3.1. Operaciones lógicas

OR {reg/mem}, {reg/mem/inmediato}

- Realiza la operación lógica OR
- Se puede usar para poner a uno un bit independientemente de su valor actual
  - ➔ La máscara contendrá un 1 allá donde queramos colocar un 1 y un 0 donde queremos dejar intacto el bit original
- También se usa para comparar con 0

© Rafael Rico López

45/145

## Repertorio de instrucciones x86-16bits

### 3.1. Operaciones lógicas

- Ejemplo:

```
MOV AX, 035h      ; 0011 0101
OR AX, 08h         ;or 0000 1000
                   ; 0011 1101
                   ; 0011 1101
OR AX, 07h         ;or 0000 0111
                   ; 0011 1111

OR BX, BX          ;¿es BX=0?
                   ;ocupa 2B tarda 2 ciclos
JG                 ;BX positivo
JL                 ;BX negativo
                   ;BX cero

CMP BX, 0           ;¿es BX=0?
                   ;ocupa 3B tarda 3 ciclos
```

es una resta que no salva el resultado

© Rafael Rico López

46/145

## Repertorio de instrucciones x86-16bits

### 3.1. Operaciones lógicas

`XOR {reg/mem},{reg/mem/inmediato}`

- Realiza la operación lógica XOR
- Se puede usar para conmutar el valor de bits específicos
  - ➔ La máscara tendrá 1 allá donde quieras conmutar
- También se usa para poner a cero un registro

© Rafael Rico López

47/145

## Repertorio de instrucciones x86-16bits

### 3.1. Operaciones lógicas

- Ejemplo:

<code>MOV AX, 035h</code>	<code>;</code>	<code>0011 0101</code>
<code>XOR AX, 08h</code>	<code>;xor</code>	<code>0000 1000</code>
	<code>;</code>	<code>0011 1101</code>
	<code>;</code>	<code>0011 1101</code>
<code>XOR AX, 07h</code>	<code>;or</code>	<code>0000 0111</code>
	<code>;</code>	<code>0011 1010</code>
<code>XOR CX, CX</code>	<code>;ocupa 2B tarda 3 ciclos</code>	
	<code>;actualiza el estado</code>	
<code>MOV CX, 0</code>	<code>;ocupa 3B tarda 4 ciclos</code>	
	<code>;NO actualiza el estado</code>	
<code>SUB CX, CX</code>	<code>;ocupa 2B tarda 3 ciclos</code>	
	<code>;actualiza el estado</code>	

© Rafael Rico López

48/145



## Repertorio de instrucciones x86-16bits

### 3.1. Operaciones lógicas

**NOT {reg/mem}**

- Complementa todos los bits del operando
- Un uso típico es invertir el significado de una máscara

© Rafael Rico López

49/145

## Repertorio de instrucciones x86-16bits

### 3.1. Operaciones lógicas


- Ejemplo:

```
.DATA
mascara    DB 0001 0000b
.CODE

MOV AX, 0D743    ;AL = 0100 0011
OR AL, mascara   ;or   0001 0000
                  ;    0101 0011

NOT mascara      ;invierte significado

AND AH, mascara  ;AH = 1101 0111
                  ;and  1110 1111
                  ;    1100 1111
```



© Rafael Rico López

50/145

## Repertorio de instrucciones x86-16bits

### 3.2. Desplazamientos y rotaciones

SHL {reg/mem}, {CL/1}	ROL {reg/mem}, {CL/1}
SHR {reg/mem}, {CL/1}	ROR {reg/mem}, {CL/1}
SAL {reg/mem}, {CL/1}	RCL {reg/mem}, {CL/1}
SAR {reg/mem}, {CL/1}	RCR {reg/mem}, {CL/1}

- Conjunto de operaciones para desplazar y rotar bits a derecha e izquierda

- ➔ Algunas conservan el signo si es necesario
- ➔ Pasan por el acarreo

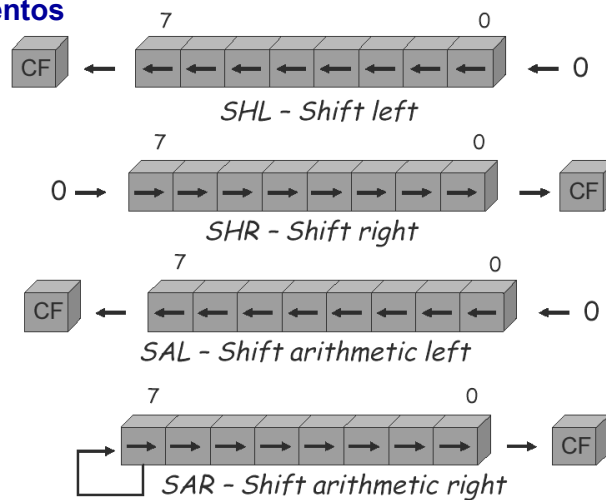
© Rafael Rico López

51/145

## Repertorio de instrucciones x86-16bits

### 3.2. Desplazamientos y rotaciones

#### Desplazamientos



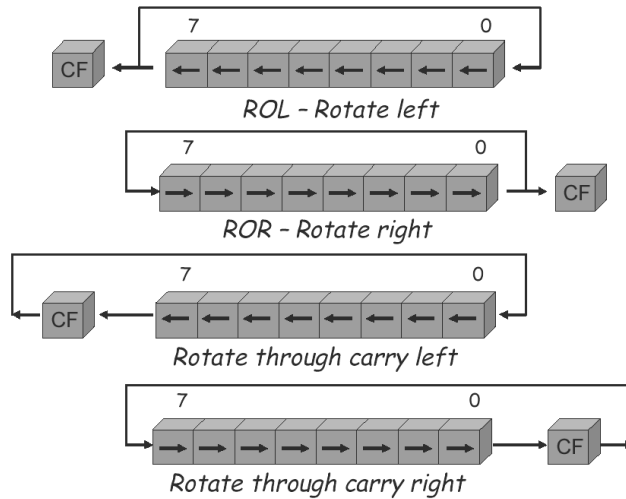
© Rafael Rico López

52/145

## Repertorio de instrucciones x86-16bits

### 3.2. Desplazamientos y rotaciones

#### Rotaciones



© Rafael Rico López

53/145

## Repertorio de instrucciones x86-16bits

### 3.2. Desplazamientos y rotaciones

- **Multiplicación y división por constantes**

- ➔ Desplazar un bit a la derecha es igual a dividir entre 2; 2 bits es dividir entre 4; etc.
- ➔ Desplazar un bit a la izquierda es igual a multiplicar por 2; 2 bits es multiplicar por 4; etc.
- ➔ Este hecho se puede aprovechar para hacer algunas operaciones más rápidas

© Rafael Rico López

54/145

## Repertorio de instrucciones x86-16bits

### 3.2. Desplazamientos y rotaciones

- SHR divide números sin signo
- SAR divide números con signo
  - ➔ La división redondea por defecto
  - ➔ IDIV redondea por exceso
- SHL y SAL funcionan igual para números con y sin signo

© Rafael Rico López

55/145

## Repertorio de instrucciones x86-16bits

### 3.2. Desplazamientos y rotaciones

- Ejemplo:

```
XOR AH,AH      ;0 en AH
SHL AX,1        ;multiplico por 2
                ;4 ciclos en 8086

MOV BL,2
MUL BL          ;74 a 81 ciclos en 8086
                ;15 ciclos en 80286
                ;11 a 16 ciclos en 80386
```

© Rafael Rico López

56/145

## Repertorio de instrucciones x86-16bits

### 3.2. Desplazamientos y rotaciones

- Desplazamientos multipalabra (I)

- ➔ Se usan con variables de tamaño arbitrario (*big numbers*)
- ➔ Cuando la variable a desplazar es demasiado grande para un registro hay que usar varias ubicaciones pasando a través del *flag* de acarreo

© Rafael Rico López

57/145

## Repertorio de instrucciones x86-16bits

### 3.2. Desplazamientos y rotaciones

- Desplazamientos multipalabra (II)
- Ejemplo:

```
.DATA
mem32 DD 500000
.CODE
;dividir 32 bits sin signo entre 16

MOV CX,4 ;4 iteraciones en el bucle
;(dividir entre 2 en cada pasada)

nuevo: SHR WORD PTR mem32[2],1 ;desplazo a través de CF
RCR WORD PTR mem32[0],1 ;el CF entra por la izq.
LOOP nuevo
```

© Rafael Rico López

58/145

## Repertorio de instrucciones x86-16bits

### 3.3. Suma

- Realizan sumas:

ADD {reg/mem},{reg/mem/inmediato}

ADC {reg/mem},{reg/mem/inmediato}

INC {reg/mem}

- ➔ Por sí mismas suman valores de tamaño byte o palabra
- ➔ En conjunción pueden realizar sumas de 32 bits (o más)
- ➔ Con las instrucciones de ajuste ASCII pueden operar con números BCD

© Rafael Rico López

59/145

## Repertorio de instrucciones x86-16bits

### 3.3. Suma

ADD {reg/mem},{reg/mem/inmediato}

- Suma los operandos y salva el resultado en el destino
- Las banderas afectadas son: CF, PF, AF, ZF, SF y OF
- La operación puede ser interpretada tanto sin signo como con signo
  - ➔ ES RESPONSABILIDAD DEL PROGRAMADOR interpretar correctamente el resultado
    - ➔ Con signo → hay desbordamiento si OF = 1
    - ➔ Sin signo → hay desbordamiento si CF = 1

© Rafael Rico López

60/145

## Repertorio de instrucciones x86-16bits

### 3.3. Suma

- Ejemplo:

programa	AL	OF	CF	sin signo	con signo
.DATA					
mem8 DB 39					
.CODE					
MOV AL, 26	1A=0001 1010	NV 0	NC 0	26	26
INC AL	1B=0001 1011	NV 0	NC 0	27	27
ADD AL, 76	67=0110 0111	NV 0	NC 0	103	103
ADD AL, mem8	8E=1000 1110	OV 1	NC 0	142	-114+desb.
MOV AH, AL					
ADD AL, AH	1C=0001 1100	OV 1	CY 1	28+carry	

© Rafael Rico López

61/145

## Repertorio de instrucciones x86-16bits

### 3.3. Suma

- En el ejemplo anterior el programador puede gobernar la secuencia del programa en función del estado que considere oportuno

➡ Usará el salto condicional que lea el estado deseado

➡ Desbordamiento si opera con signo

➡ Acarreo si opera sin signo

© Rafael Rico López

62/145

## Repertorio de instrucciones x86-16bits

### 3.3. Suma

`ADC {reg/mem},{reg/mem/inmediato}`

- Suma igual que ADD pero incluye en la suma la bandera de acarreo CF
- Las banderas afectadas son: CF, PF, AF, ZF, SF y OF
- Sirve para realizar sumas de tamaño 32 bits (o más)
  - ➔ Utilizaremos dos registros
  - ➔ Es conveniente que sean DX:AX
- La suma de menor peso se realiza con ADD
- La suma de mayor peso se realiza con ADC

© Rafael Rico López

63/145

## Repertorio de instrucciones x86-16bits

### 3.3. Suma

- Ejemplo:

```
mem32    .DATA
          DD 316423
          .CODE
```

DX:AX	=	43.981
	+	316.423
		<hr/>
		360.404

```
MOV AX, 43981
XOR DX, DX
ADD AX, WORD PTR mem32[0]
ADC DX, WORD PTR mem32[2]
```

© Rafael Rico López

64/145



## Repertorio de instrucciones x86-16bits

### 3.3. Suma

`INC {reg/mem}`

- Las banderas afectadas son: PF, AF, ZF, SF y OF

➔ No modifica la bandera de acarreo

➔ Funciona como un contador no saturado

## Repertorio de instrucciones x86-16bits

### 3.4. Resta

- Realizan restas:

`SUB {reg/mem}, {reg/mem/inmediato}`

`SBB {reg/mem}, {reg/mem/inmediato}`

`DEC {reg/mem}`

`NEG {reg/mem}`

➔ Por si mismas restan valores de tamaño byte o palabra

➔ En conjunción pueden realizar restas sobre 32 bits (o más)

➔ Con las instrucciones de ajuste ASCII pueden operar con números BCD

## Repertorio de instrucciones x86-16bits

### 3.4. Resta

**SUB {reg/mem},{reg/mem/inmediato}**

- Resta los operandos y salva el resultado en el destino  
 $\text{op. destino} \leftarrow \text{op. destino} - \text{op. fuente}$
- Las banderas afectadas son: CF, PF, AF, ZF, SF y OF
- La operación puede ser interpretada tanto sin signo como con signo
  - ➔ **ES RESPONSABILIDAD DEL PROGRAMADOR** interpretar correctamente el resultado
    - ➔ Con signo → hay desbordamiento si **OF** = 1
    - ➔ Sin signo → hay desbordamiento si **SF** = 1

© Rafael Rico López

67/145

## Repertorio de instrucciones x86-16bits

### 3.4. Resta

- **Ejemplo:**

programa	AL	OF	SF	sin signo	con signo
.DATA					
mem8 DB 122					
.CODE					
MOV AL, 95	5F=0101 1111	NV 0	PL 0	95	95
DEC AL	5E=0101 1110	NV 0	PL 0	94	94
SUB AL, 23	47=0100 0111	NV 0	PL 0	71	71
SUB AL, mem8	CD=1100 1101	NV 0	NG 1	205+signo	-51
MOV AH, 119					
SUB AL, AH	56=0101 0110	OV 1	NG 1		86+desb.

© Rafael Rico López

68/145

## Repertorio de instrucciones x86-16bits

### 3.4. Resta

- En el ejemplo anterior el programador puede gobernar la secuencia del programa en función del estado que considere oportuno

➔ Usará el salto condicional que lea el estado deseado

➔ Desbordamiento si opera con signo

➔ Signo si opera sin signo

© Rafael Rico López

69/145

## Repertorio de instrucciones x86-16bits

### 3.4. Resta

**SBB {reg/mem}, {reg/mem/inmediato}**

- Resta igual que SUB pero incluye en la suma la bandera de acarreo CF
- Las banderas afectadas son: CF, PF, AF, ZF, SF y OF
- Sirve para realizar restas de tamaño 32 bits (0 más)
  - ➔ Utilizaremos dos registros
  - ➔ Es conveniente que sean DX:AX
- La resta de menor peso se realiza con SUB
- La resta de mayor peso se realiza con SBB

© Rafael Rico López

70/145

## Repertorio de instrucciones x86-16bits

### 3.4. Resta

- Ejemplo:

```
.DATA
mem32a DD 316423
mem32b DD 156739
.CODE
```

$\begin{array}{r} \text{DX:AX} = 316.423 \\ + 156.739 \\ \hline 159.684 \end{array}$
--

```
MOV AX, WORD PTR mem32a[0]
MOV DX, WORD PTR mem32a[2]
SUB AX, WORD PTR mem32b[0]
SBB DX, WORD PTR mem32b[2]
```

© Rafael Rico López

71/145

## Repertorio de instrucciones x86-16bits

### 3.4. Resta

**DEC {reg/mem}**

- Las banderas afectadas son: PF, AF, ZF, SF y OF

- ➔ No modifica la bandera de acarreo

- ➔ Funciona como un contador no saturado

© Rafael Rico López

72/145

## Repertorio de instrucciones x86-16bits

### 3.4. Resta

**NEG {reg/mem}**

- Calcula el negativo del operando en C-2
- Es equivalente a:

**NOT {reg/mem}**

**INC {reg/mem}**

- Afecta a todas las banderas de estado

© Rafael Rico López

73/145

## Repertorio de instrucciones x86-16bits

### 3.5. Multiplicación

- Realizan multiplicaciones:

**MUL {reg/mem}**

**IMUL {reg/mem}**

➔ **MUL** → números sin signo

➔ **IMUL** → números con signo

**AX ← operando fuente x AL**

**DX:AX ← operando fuente x AX**

➔ Si la mitad superior es no cero se indica con CF y OF

© Rafael Rico López

74/145

## Repertorio de instrucciones x86-16bits

### 3.5. Multiplicación

- Ejemplo:

```
.DATA
mem16 DW -30000
.CODE

;multiplicación 8 bits sin signo
MOV AL,23 ;carga AL 23
MOV BL,24 ;carga BL 24
MUL BL ;AX = 552 cf, of
```

Excede AL

```
;multiplicación 16 bits con signo
MOV AX,50 ;carga AX 50
IMUL mem16 ; * -30000
;DX:AX -150000 cf, of
```

Excede AX

© Rafael Rico López

75/145

## Repertorio de instrucciones x86-16bits

### 3.5. Multiplicación

- Multiplicar es una operación muy lenta
- Muchas veces es conveniente...
  - ➔ ... sustituir por desplazamientos a izquierda si el factor es potencia de 2
  - ➔ ... evaluar si el multiplicador es 0 ó 1 y saltar:
    - ➔ 0 → sustituir por cero
    - ➔ 1 → sustituir por el multiplicando

© Rafael Rico López

76/145

## Repertorio de instrucciones x86-16bits

### 3.6. División

- Realizan divisiones:

`DIV {reg/mem}`

`IDIV {reg/mem}`

➔ `DIV` → números sin signo

➔ `IDIV` → números con signo

© Rafael Rico López

77/145

## Repertorio de instrucciones x86-16bits

### 3.6. División

- Operandos:

➔ Dividendo → `AX` o `DX:AX`

➔ Divisor → operando fuente (excepto `AX` y/o `DX`)

- Operaciones de 16 bits entre 8 bits:

$AL(\text{cociente})/AH(\text{resto}) \leftarrow AX \div \text{operando fuente}$

- Operaciones de 32 bits entre 16 bits:

$AX(\text{cociente})/DX(\text{resto}) \leftarrow DX:AX \div \text{operando fuente}$

© Rafael Rico López

78/145

## Repertorio de instrucciones x86-16bits

### 3.6. División

- Operaciones de 16 bits entre 16 bits:
  - ➔ Es necesario extender el dividendo a 32 bits (con signo o sin signo)
- Si el divisor es 0 o el cociente excede el registro acumulador el procesador invoca la interrupción INT 0
  - ➔ Bajo D.O.S. → por defecto el programa termina y devuelve el control al D.O.S.
- ➔ Soluciones:
  - ➔ Comprobar si es cero antes de operar
  - ➔ Reescribir la rutina de atención a la INT 0

© Rafael Rico López

79/145

## Repertorio de instrucciones x86-16bits

### 3.6. División

- Ejemplo:

```
.DATA
mem16 DW      -2000
mem36 DD      500000
.CODE

MOV AX,700      ;divide 16 bits sin signo entre 8 bits
MOV BL,36       ;carga el dividendo
DIV BL          ;carga el divisor
               ;divide entre BL (700/36)
               ;cociente AL=19 resto AH=16

               ;divide 32 bits con signo entre 16 bits
MOV AX,WORD PTR mem32[0] ;carga dividendo en DX:AX
MOV DX,WORD PTR mem32[2] ;
IDIV mem16      ;divide 500000/-2000
               ;cociente AX=-250 resto DX=0
```

© Rafael Rico López

80/145



## Repertorio de instrucciones x86-16bits

### 3.6. División

- Ejemplo (continuación):

```
.DATA
mem16 DW      -2000
mem36 DD      500000
.CODE

;divide 16 bits con signo entre 16 bits
MOV AX,WORD PTR mem16 ;carga dividendo en AX
CWD                    ;extiende a DX:AX
MOV BX,-421            ;carga el divisor
IDIV BX                ;divide entre BX -2000/-421
                        ;cociente AX=4 resto DX=-316
```

© Rafael Rico López

81/145

## Repertorio de instrucciones x86-16bits

### 3.6. División

- Dividir es una operación muy lenta
- Muchas veces es conveniente...
  - ➔ ... sustituir por desplazamientos a derecha si el factor es potencia de 2
  - ➔ ... evaluar si divisor es 1 para sustituir por el dividendo
  - ➔ ... cambiar divisiones por multiplicaciones si es posible

$$X/5 = X * 0,2$$

© Rafael Rico López

82/145

## Repertorio de instrucciones x86-16bits

### 3.7. Operaciones en BCD

- Dos tipos de instrucciones:
  - ➔ Ajuste ASCII → BCD desempquetado
    - ➔ Suma
    - ➔ Resta
    - ➔ Multiplicación
    - ➔ División
  - ➔ Ajuste decimal → BCD empaquetado
    - ➔ Suma
    - ➔ Resta

© Rafael Rico López

83/145

## Repertorio de instrucciones x86-16bits

### 3.7. Operaciones en BCD

- AAA: ajusta el resultado después de una suma

➔ Ejemplo:

```
MOV AX, 9
MOV BX, 3
ADD AL, BL ;AL=0Ch
AAA        ;AL=02h AH=01h; cf
```

Resultado en AL

- AAS: ajusta el resultado después de una resta

➔ Ejemplo:

```
MOV AX, 103h
MOV BX, 4
SUB AL, BL ;AL=0FFh (-1)
AAS        ;AL=09h AH=0h; cf
```

© Rafael Rico López

84/145

## Repertorio de instrucciones x86-16bits

### 3.7. Operaciones en BCD

- **AAM:** ajusta el resultado después de multiplicación

➔ Ejemplo:

```
MOV AX,903h
MUL AH      ;AL=1Bh
AAM         ;AL=02h AH=07h
```

NO usar IMUL

- **AAD:** ajusta el resultado ANTES de una división

➔ Ejemplo:

```
MOV AX,205h ;BCD desempquetado
MOV BL,2    ;divisor
AAD         ;AX=19h

DIV BL      ;AL=0Ch AH=01h
AAM         ;AX=0102h resto perdido
```

85/145

© Rafael Rico López

## Repertorio de instrucciones x86-16bits

### 3.7. Operaciones en BCD

- **DAA:** ajusta el resultado después de suma

➔ Ejemplo:

```
MOV AX,8833h
ADD AL,AH   ;AL=0BBh
DAA         ;121 → CF=1 AL=21h
```

- **DAS:** ajusta el resultado después de resta

➔ Ejemplo:

```
MOV AX,3883h
SUB AL,AH   ;AL=04Bh
DAS         ;45 → CF=0 AL=45h
```

86/145

© Rafael Rico López

## Repertorio de instrucciones x86-16bits

### 4. Instrucciones de bifurcación

---

#### 4. Instrucciones de bifurcación

##### Índice

1. Bifurcaciones
  1. Incondicional
  2. Condicionales
2. Bucles
3. Procedimientos
4. Interrupciones

© Rafael Rico López

87/145

## Repertorio de instrucciones x86-16bits

### 4.1. Bifurcaciones

---

- Alteran el flujo de control del programa
- Actúan sobre IP (y CS, a veces)
- Los saltos pueden ser:
  - ➔ *Short* → rango de  $\pm 128B$  (modifican IP)
  - ➔ *Near* → rango de  $\pm 32KB$  (modifican IP)
  - ➔ *Far* → otro segmento (modifican CS e IP)

© Rafael Rico López

88/145

## Repertorio de instrucciones x86-16bits

### 4.1.1. Incondicional

**JMP {reg/mem}**

- Salta siempre
- La actualización del puntero IP se realiza:
  - ➔ *Short* →  $IP = IP + \text{desplazamiento (byte)}$
  - ➔ *Near* →  $IP = IP + \text{desplazamiento (2 bytes)}$
  - ➔ *Far* →  $IP = \text{desplazamiento CS} = \text{segmento}$
- Dos modos:
  - ➔ Directo: etiqueta de memoria resuelta en tiempo de compilación o ensamblado
  - ➔ Indirecto: se da un puntero que tiene la dirección de salto (se resuelve en tiempo de ejecución)

© Rafael Rico López

89/145

## Repertorio de instrucciones x86-16bits

### 4.1.1. Incondicional

- Tamaños del formato (modo directo)
  - ➔ *Short* → 2 bytes      1110 1011 (EBh) despla\_8
  - ➔ *Near* → 3 bytes      1110 1001 (E9h) despla\_16
  - ➔ *Far* → 5 bytes      1110 1010 (EAh) despla\_16:base
- Tamaños del formato (modo indirecto)
  - ➔ *Short* → no existe
  - ➔ *Near* → 2 bytes      1111 1111 (FFh) mod 100 r/m
  - ➔ *Far* → 2 bytes      1111 1111 (FFh) mod 101 r/m
    - ➔ (Si *far* el operando NO puede ser un registro ya que no caben los 32 bits de *base:desplazamiento*)
  - ➔ El modo indirecto permite convertir incondicionales en condicionales

© Rafael Rico López

90/145

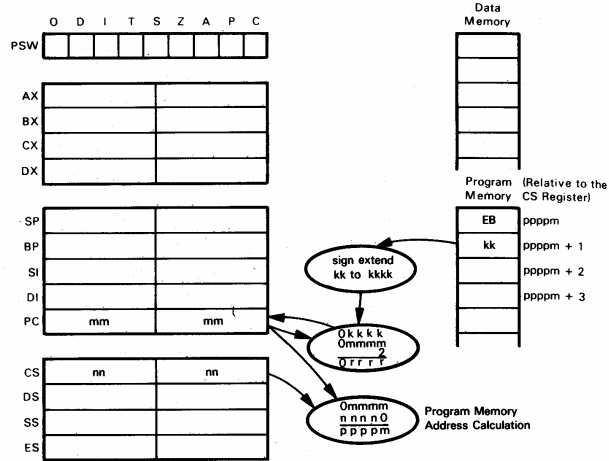
## Repertorio de instrucciones x86-16bits

### 4.1.1. Incondicional

#### • Ejemplo: **JMP desp8**

➔ 15 ciclos

el desplazamiento  
es con signo



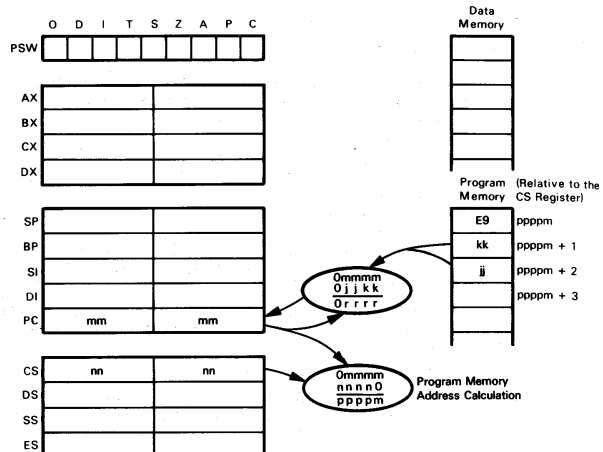
91/145

## Repertorio de instrucciones x86-16bits

### 4.1.1. Incondicional

#### • Ejemplo: **JMP desp16**

➔ 15 ciclos



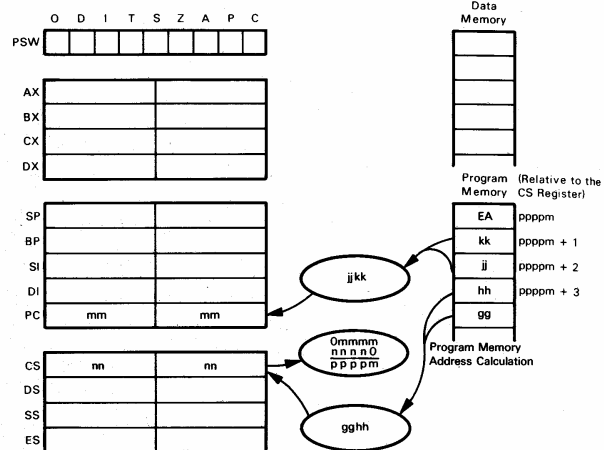
92/145

## Repertorio de instrucciones x86-16bits

### 4.1.1. Incondicional

- Ejemplo: **JMP dirección**

➔ 15 ciclos



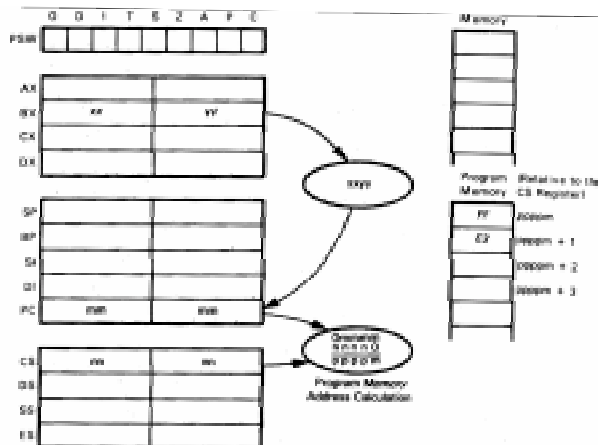
93/145

## Repertorio de instrucciones x86-16bits

### 4.1.1. Incondicional

- Ejemplo: **JMP BX**

➔ 11 ciclos



➔ **JMP [BX]**

➔ 16 + cálculo EA ciclos

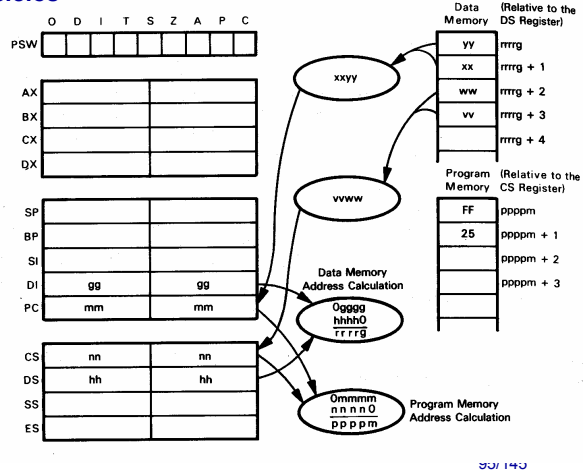
94/145

## Repertorio de instrucciones x86-16bits

### 4.1.1. Incondicional

- Ejemplo: **JMP far ptr[DI]**

➔ 24 + cálculo EA ciclos



## Repertorio de instrucciones x86-16bits

### 4.1.1. Incondicional

- Código en lenguaje C

```
/* captura variable B */
switch (B)
{
    case 0:
        /* rutina 0 */
        break;

    case 1:
        /* rutina 1 */
        break;

    case 2:
        /* rutina 2 */
        break;
}
```

- Código en ensamblador

```
.CODE
JMP start ;saltar datos
cases LABEL WORD
DW caso0 ;puntero
DW caso1 ;puntero
DW caso2 ;puntero
start : BX ← valor ;capturo valor

SHL BX,1 ;multiplico x2
JMP cases[bx] ;salto al caso

caso0: : :
JMP seguir ;break
caso1: : :
JMP seguir ;break
caso2: : :
JMP seguir ;break
seguir: : : ;fin switch
```

96/145



## Repertorio de instrucciones x86-16bits

### 4.1.1. Incondicional

#### Referencias a etiquetas adelantadas

```
JMP etiqueta
:::
:::
etiqueta: :::
```

- El ensamblador procesa los ficheros fuente secuencialmente en varias pasadas
- Cuando encuentra 'etiqueta' no sabe a qué distancia se encuentra (no puede determinar qué espacio reservar para el desplazamiento)
- Reserva por defecto espacio para una palabra (salto *near*) y espera a la segunda pasada para colocar el valor
- El código resultante puede ser incorrecto (si *far*) o ineficiente (si *short*)
  - ➔ Solución: marcar la longitud del salto (si no se acierta se emite un mensaje de error o de aviso)

97/145

© Rafael Rico López

## Repertorio de instrucciones x86-16bits

### 4.1.2. Condicionales

#### Jcc desplazamiento

- Salta o no en función del cumplimiento de la condición
  - ➔ Primero evalúa la condición
  - ➔ Segundo actualiza o no el IP
    - ➔ cc = true → salto
    - ➔ cc = false → no salto
- El desplazamiento se da como un etiqueta que se resuelve en tiempo de compilación
- Los saltos condicionales siempre son *short* ( $\pm 128$  bytes); si queremos saltar más lejos hemos de usar JMP

98/145

© Rafael Rico López

## Repertorio de instrucciones x86-16bits

### 4.1.2. Condicionales

- **Ejemplo:**

- ➔ Construcción de un salto condicional mayor de  $\pm 128$ Bytes

```
CMP AX, 7
JE cerca
: : : ;código si AX≠7
: : :

cerca: JMP lejos ;salto >  $\pm 128$ B

lejos: : : : ;código si AX=7
```

© Rafael Rico López

99/145

## Repertorio de instrucciones x86-16bits

### 4.1.2. Condicionales

- Las condiciones se expresan sobre el registro de estado excepto en el caso de JCXZ que se hace sobre el registro contador CX (buscando un cero)
- Los mnemónicos son muy variados pero la funcionalidad se repite:
  - ➔ Mnemónicos basados en *flags*
  - ➔ Mnemónicos basados en comparaciones

© Rafael Rico López

100/145

## Repertorio de instrucciones x86-16bits

### 4.1.2. Condicionales

- **Mnemónicos basados en *flags*:**

Instrucción	Función
JO	salta si OF = 1
JNO	salta si OF = 0
JC	salta si CF = 1 (= JB)
JNC	salta si CF = 0 (= JAE)
JZ	salta si ZF = 1 (= JE)
JNZ	salta si ZF = 0 (= JNE)
JS	salta si SF = 1
JNS	salta si SF = 0
JP	salta si PF = 1 (= JPE)
JNP	salta si PF = 0 (= JPO)
JPE	salta si PF = 1 (paridad par)
JPO	salta si PF = 0 (paridad impar)
JCXZ	salta si CX = 0

© Rafael Rico López

101/145

## Repertorio de instrucciones x86-16bits

### 4.1.2. Condicionales

- **Ejemplos:**

```
ADD AX, BX
JO desbordamiento
: : :
: : :
: : :

desbordamiento:

SUB AX, DX
JNZ seguir
CALL caso_cero
: : :
: : :

seguir:
```

© Rafael Rico López

102/145

## Repertorio de instrucciones x86-16bits

### 4.1.2. Condicionales

- **Mnemónicos basados en comparaciones:**

➔ Se usan después de una comparación

➔ **CMP** → SUB sin salvar resultado

➔ **TEST** → AND sin salvar resultado

**CMP {reg|mem},{reg|mem|inmediato}**

**TEST {reg|mem},{reg|mem|inmediato}**

© Rafael Rico López

103/145

## Repertorio de instrucciones x86-16bits

### 4.1.2. Condicionales

- **Mnemónicos basados en comparaciones:**

Letra	Significado
G	mayor que (números con signo)
L	menor que (números con signo)
A	por encima (números sin signo)
B	por debajo (números sin signo)
E	igual
N	no

Condición		CMP signo	Salta si	CMP sin signo	Salta si
Igual	=	JE	ZF=1	JE	ZF=1
Distinto	≠	JNE	ZF=0	JNE	ZF=0
Mayor que	>	JG o JNLE	ZF=0 and SF=OF	JA or JNBE	CF=0 and ZF=0
Menor o igual	≤	JLE o JNG	ZF=1 and SF≠OF	JBE or JNA	CF=1 or ZF=1
Menor que	<	JL o JNGE	SF≠OF	JB or JNAE	CF=1
Mayor o igual	≥	JGE o JNL	SF=OF	JAE or JNB	CF=0

© Rafael Rico López

104/145

## Repertorio de instrucciones x86-16bits

### 4.1.2. Condicionales

#### • Ejemplo:

- ➔ Si CX es menor que -20 entonces DX = 30 si no DX = 20

```

CMP CX, -20 ;if
JL menor ;then
MOV DX,20 ;else
JMP seguir
menor: MOV DX,30
seguir: : : :

```

- ➔ DX = 20 a no ser que CX sea menor que -20 en cuyo caso DX = 30

```

MOV DX,20 ;then
CMP CX,-20 ;if
JGE mayorque ;else
MOV Dx,30
mayorque: : : :

```

- ➔ Si CX es mayor o igual que -20 entonces DX = 20 si no DX = 30

```

CMP CX,-20 ;if
JNL nomenor ;else
MOV DX,30 ;then
JMP seguir
nomenor: MOV DX,20
seguir: : : :

```

- ➔ Este es el código más eficiente
- ➔ Evita el JMP

© Rafael Rico López

105/145

## Repertorio de instrucciones x86-16bits

### 4.1.2. Condicionales

#### • Ejemplo:

```

.DATA
bits DB ?
.CODE
;si bit 2 o bit 4 están a 1 entonces procA

TEST bits,10100b
JZ seguir
CALL procA
seguir:

;si bit 2 y bit 4 están a 0 entonces procB

TEST bits,10100b
JNZ continuar
CALL procB
continuar:

```

© Rafael Rico López

106/145

## Repertorio de instrucciones x86-16bits

### 4.2. Bucles

- Instrucciones para crear bucles:

- ➔ Bucles por contador

- ➔ `LOOP etiqueta` → for

- ➔ Bucles por condición

- ➔ `LOOPE/LOOPZ etiqueta` → while

- ➔ `LOOPNE/LOOPNZ etiqueta` → while

- ➔ Otros

- ➔ `JCXZ etiqueta`

© Rafael Rico López

107/145

## Repertorio de instrucciones x86-16bits

### 4.2. Bucles

- Funcionalidad:

- ➔ Bucles por contador

- ➔ `LOOP etiqueta`

contador = contador – 1 (sin afectar a los *flags*)

IF contador ≠ 0 THEN GOTO etiqueta

- ➔ Bucles por condición

- ➔ `LOOPE/LOOPZ etiqueta`

contador = contador – 1

IF contador ≠ 0 AND **ZF**=1 THEN GOTO etiqueta

- ➔ `LOOPNE/LOOPNZ etiqueta`

contador = contador – 1

IF contador ≠ 0 AND **ZF**=0 THEN GOTO etiqueta

© Rafael Rico López

108/145

## Repertorio de instrucciones x86-16bits

### 4.2. Bucles

- Operandos:

- ➔ Actualizan el contador CX automáticamente

- ➔ LOOP etiqueta → CX, salto *short*

- ➔ LOOPE/LOOPZ etiqueta → CX, ZF, salto *short*

- ➔ LOOPNE/LOOPNE etiqueta → CX, ZF, salto *short*

- ➔ No actualiza el contador CX automáticamente

- ➔ JCXZ etiqueta → CX, salto *short*

© Rafael Rico López

109/145

## Repertorio de instrucciones x86-16bits

### 4.2. Bucles

- Ejemplo:

```
MOV CX,200
lazo: . . . .
      . . . .
      . . . .
      LOOP lazo
```

```
MOV CX,200
lazo: . . . .
      . . . .
      . . . .
      DEC CX
      JNE lazo
```

- El primero es más compacto pero el segundo es necesario para comprobar varias condiciones
- Hoy se usa más el segundo ya que evita el uso dedicado de CX

© Rafael Rico López

110/145

## Repertorio de instrucciones x86-16bits

### 4.2. Bucles

- **Tiempos:**

- ➔ **LOOP**

- ➔ Se toma el salto → 17 ciclos
- ➔ No se toma el salto → 5 ciclos

- ➔ **LOOPE**

- ➔ Se toma el salto → 18 ciclos
- ➔ No se toma el salto → 6 ciclos

- ➔ **LOOPNE**

- ➔ Se toma el salto → 19 ciclos
- ➔ No se toma el salto → 5 ciclos

© Rafael Rico López

111/145

## Repertorio de instrucciones x86-16bits

### 4.3. Procedimientos

#### **CALL**

- ➔ Salva la dirección de retorno (IP o CS : IP en curso) en la pila y salta al procedimiento

#### **RET**

- ➔ Recupera la dirección de retorno y vuelve al programa principal

© Rafael Rico López

112/145



## Repertorio de instrucciones x86-16bits

### 4.3. Procedimientos

**CALL {reg/mem}**

- Salva en la pila la dirección de la siguiente instrucción
- Salta a la dirección especificada

[SP = SP - 2]  
[CS → pila]  
[CS = nuevo CS]

SP = SP - 2  
IP → pila  
IP = nuevo IP

© Rafael Rico López

113/145

## Repertorio de instrucciones x86-16bits

### 4.3. Procedimientos

**CALL {reg/mem}**

- La dirección de salto se puede dar de forma:
  - ➔ Directa → etiqueta
  - ➔ Indirecta → puntero en registro o en memoria
- Los saltos pueden ser:
  - ➔ Near → sólo involucra IP
  - ➔ Far → la dirección se da como CS : IP

Ejemplo:

CALL far ptr TAREA → salto far

problemática de las  
etiquetas adelantadas

© Rafael Rico López

114/145

## Repertorio de instrucciones x86-16bits

### 4.3. Procedimientos

#### • Definición de procedimientos:

1 etiqueta PROC [NEAR|FAR]  
 : : : ;código  
 RET [constante]  
 etiqueta ENDP

2 etiqueta:  
 : : : ;código  
 RETN [constante]

3 etiqueta: LABEL FAR  
 : : : ;código  
 RETF [constante]

© Rafael Rico López

115/145

## Repertorio de instrucciones x86-16bits

### 4.3. Procedimientos

RET [constante]

- Devuelve el control al proceso llamador sacando de la pila CS:IP o IP dependiendo de si es *far* o *near*
- Como operando opcional tenemos una constante cuyo significado es el número de bytes a sumar a SP

RET (far)	RET (near)	RETN 3
POP IP SP+2	POP IP SP+2	POP IP SP+2
POP CS SP+2		SP+3

© Rafael Rico López

116/145

## Repertorio de instrucciones x86-16bits

### 4.4. Interrupciones

#### INT número

- ➔ Interrupción es un procedimiento solicitado por número
- ➔ número está comprendido entre 0 y 255

#### IRET

- ➔ Recupera la dirección de retorno y el estado y devuelve el control al proceso llamador

© Rafael Rico López

117/145

## Repertorio de instrucciones x86-16bits

### 4.4. Interrupciones

#### INT número

- ➔ Cuando se llama a una interrupción se siguen estos pasos:
  1. PUSH *flags*
  2. TF = 0 e IF = 0
  3. Búsqueda del vector en la tabla →  $n \times 4$
  4. PUSH CS e IP
  5. Salta a la rutina de atención
  6. Ejecuta el código hasta encontrar un IRET

SP=SP-2  
*flags* → pila  
IF=0  
TF=0  
SP=SP-2  
CS → pila  
CS= $n \times 4 + 2$   
SP=SP-2  
IP → pila  
IP= $n \times 4$

© Rafael Rico López

118/145

## Repertorio de instrucciones x86-16bits

### 4.4. Interrupciones

#### IRET

- Devuelve el control al proceso llamador y restaura el estado:

1. POP IP
2. POP CS
3. POP *flags*

IP ← pila  
SP=SP+2  
CS ← pila  
SP=SP+2  
*flags* ← pila  
SP=SP+2

## Repertorio de instrucciones x86-16bits

### 4.4. Interrupciones

- Definición de rutinas de interrupción:

```
etiqueta    PROC FAR  
  
            : : :      ;código  
  
            IRET  
etiqueta    ENDP
```

## Repertorio de instrucciones x86-16bits

### 4.4. Interrupciones

#### INTO

- Equivalente a:

JO rutina	JO rutina
. . . .	. . . .
. . . .	. . . .
rutina: INT 4	rutina: ;código si OF

#### CLI

- IF = 0 → no se atienden interrupciones *hardware*

#### STI

- IF = 1 → se atienden las interrupciones *hardware*

© Rafael Rico López

121/145

## Repertorio de instrucciones x86-16bits

### 5. Otras instrucciones

#### 5. Otras instrucciones

##### Índice

##### 1. Procesamiento de cadenas

1. Configuración
2. Movimiento de cadenas
3. Búsquedas
4. Comparaciones
5. Paso de caracteres a cadenas
6. Lectura de caracteres desde cadenas
7. E/S con cadenas

##### 2. Instrucciones de control

© Rafael Rico López

122/145

## Repertorio de instrucciones x86-16bits

### 5.1. Procesamiento de cadenas

- El repertorio x86-16bits ofrece un conjunto de instrucciones para la manipulación de cadenas de caracteres
  - ➔ Están pensadas para disminuir el salto existente entre los lenguajes de alto nivel y el ensamblador
  - ➔ También permiten disminuir el tamaño de los ejecutables
  - ➔ Pueden usarse sin restricciones para estructuras de datos que no sean precisamente cadenas (bytes o palabras)

© Rafael Rico López

123/145

## Repertorio de instrucciones x86-16bits

### 5.1. Procesamiento de cadenas

- Las instrucciones de manejo de cadenas son:

Instrucción	Descripción
MOVS	Copia un carácter de una cadena a otro lugar
SCAS	Busca un carácter en una cadena
CMPS	Compara un carácter de una cadena con los de otra
LODS	Carga un carácter de una cadena en Acc
STOS	Transfiere un carácter del Acc a una cadena
INS	Transfiere un carácter desde un puerto de E/S a una cadena
OUTS	Transfiere un carácter desde una cadena a un puerto de E/S

- Tienen una sintaxis similar
- Pueden utilizar prefijos de repetición para trabajar como bucles

© Rafael Rico López

124/145

## Repertorio de instrucciones x86-16bits

### 5.1. Procesamiento de cadenas

- Los prefijos de repetición son:

➔ REP, REPE, REPNE, REPZ y REPNZ

Instrucción	Descripción
REP	Repite un número de iteraciones especificado en CX
REPE y REPZ	Repite mientras igual. CX indica el máximo nº de iteraciones
REPNE y REPNZ	Repite mientras no igual. CX indica el máximo nº de iteraciones

## Repertorio de instrucciones x86-16bits

### 5.1. Procesamiento de cadenas

- Sintaxis (I):

[prefijo]instrucción[ES:[destino]], [[reg\_seg:]fuente]

- ➔ Las instrucciones de manejo de cadenas pueden especificar el operando fuente y/o el destino aunque no es obligatorio
  - ➔ El tamaño del (los) operando(s) indica el tamaño de los objetos a procesar
  - ➔ Si no se especifican operandos, se toman los señalados por los punteros DS:SI para el fuente y ES:DI para el destino
  - ➔ El puntero destino siempre tiene como base ES
- ➔ Normalmente se usa prefijo de repetición para completar el procesamiento de la cadena de manera iterativa (≈ bucle)

## Repertorio de instrucciones x86-16bits

### 5.1. Procesamiento de cadenas

- **Sintaxis (II):**

[prefijo] instrucciónB

[prefijo] instrucciónW

- Las letras B o W identifican el tamaño de los operandos

➡ Con esta sintaxis no se permite especificar operandos

© Rafael Rico López

127/145

## Repertorio de instrucciones x86-16bits

### 5.1. Procesamiento de cadenas

- **Todo junto: instrucciones, prefijos y operandos**

Instrucción	Prefijo	Fuente/Destino	Puntero
MOVS	REP	ambos	DS:SI, ES:DI
SCAS	REPE/REPNE	destino	ES:SI
CMPS	REPE/REPNE	ambos	DS:SI, ES:DI
STOS	REP	destino	ES:DI
LODS	-----	fuentes	DS:SI
INS	REP	destino	ES:DI
OUTS	REP	fuentes	DS:SI

© Rafael Rico López

128/145



## Repertorio de instrucciones x86-16bits

### 5.5.1. Configuración

1. **Dar el valor deseado al *flag* de dirección**
  - **DF** = 0 → las cadenas se procesan hacia arriba (direcciones bajas a altas); instrucción **CLD**
  - **DF** = 1 → las cadenas se procesan hacia abajo (direcciones altas a bajas); instrucción **STD**
2. **Cargar el registro contador CX con el número de caracteres a procesar** (si se quiere un bucle implícito)
3. **Cargar la dirección de comienzo de la cadena fuente en DS:SI y la de la cadena destino en ES:DI**
  - La cadena fuente puede no estar en el segmento DS
  - La cadena destino siempre debe estar en ES
4. **Seleccionar el prefijo de repetición adecuado** (si bucle)
5. **Seleccionar la instrucción deseada y colocarla junto al prefijo** (si bucle)

© Rafael Rico López

129/145

## Repertorio de instrucciones x86-16bits

### 5.5.2. Movimiento de cadenas

#### MOVS

➡ Copia datos de un área de memoria a otra

[REP] MOVS [ES:]destino,[reg\_seg:]fuente

[REP] MOVB

[REP] MOVW

© Rafael Rico López

130/145


## Repertorio de instrucciones x86-16bits

### 5.5.2. Movimiento de cadenas

- Ejemplo:

```
.MODEL small
.DATA
fuente DB 10 DUP ('0123456789')
destino DB 100 DUP (?)
.CODE
MOV AX, @DATA
MOV DS, AX      ;solapo los 2      : : :
MOV ES, AX      ;segtos. de datos : : :
: : :
: : :
CLD
MOV CX, 100
MOV SI, OFFSET fuente
MOV DI, OFFSET destino
REP MOVSB

repetir: MOV AL, [SI]
        MOV [DI], AL
        INC SI
        INC DI
        LOOP repetir
```



131/145

© Rafael Rico López

## Repertorio de instrucciones x86-16bits

### 5.5.2. Movimiento de cadenas

- A veces es más rápido mover palabras en lugar de bytes:

➡ OJO con números impares de bytes (no completa palabra)

```
MOV CX, contador
SHR CX, 1      ;dividir entre 2 -> CF = 1 si es impar

REP MOVSW      ;copiamos palabras (más eficiente)
RCR CX, 1      ;si CF = 1 (impar) -> CX = 1
REP MOVSB      ;copia el último byte si lo hay
```

132/145

© Rafael Rico López

## Repertorio de instrucciones x86-16bits

### 5.5.3. Búsquedas

#### SCAS

- ➔ Explora una cadena buscando un carácter específico
- ➔ El carácter o valor se almacena en AL o AX
- ➔ Cada vez que se encuentra un emparejamiento el *flag* de cero se pone a 1 (IF [ES:DI]=Acc THEN ZF = 1)
- ➔ Sólo tiene sentido utilizar los prefijos que evalúan ZF

[REPE|REPNE] SCAS [ES:]destino

[REPE|REPNE] SCASB

[REPE|REPNE] SCASW

© Rafael Rico López

133/145

## Repertorio de instrucciones x86-16bits

### 5.5.3. Búsquedas

#### • Ejemplo:

```
.DATA
Cadena      DB "Más vale pájaro en mano que ciento volando"
Longitud     EQU $ - cadena
Puntero     DD cadena
.CODE
: : :
: : :
CLD          ;dirección a cero (hacia arriba)
MOV CX, longitud ;longitud cadena al contador
LES DI, puntero ;inicializo ES:DI
MOV AL, 'v'    ;valor a buscar (ASCII de la 'v')
REPNE SCASB   ;repetir mientras no sea igual
JNZ noesta    ;tratar el caso de que no esté
: : :
noesta:      : : :
```

© Rafael Rico López

134/145

## Repertorio de instrucciones x86-16bits

### 5.5.4. Comparaciones

#### CMPS

- ➔ Compara dos cadenas y apunta a la dirección donde se produce un emparejamiento o un desemparejamiento
- ➔ Compara carácter a carácter y si los valores son iguales  $ZF = 1$
- ➔ Sólo tiene sentido utilizar los prefijos que evalúan  $ZF$
- ➔ Los punteros  $SI$  y  $DI$  apunta al siguiente carácter
- ➔ Si finaliza la cuenta sin emparejamiento con  $REPNE$  CMPS entonces  $ZF = 0$
- ➔ Si finaliza la cuenta sin desemparejamiento con  $REPE$  CMPS entonces  $ZF = 1$

[REPE|REPNE]CMPS[reg\_seg:fuelle],[ES:]destino

[REPE|REPNE]CMPSB

[REPE|REPNE]CMPSW

es la única instrucción que escribe los operandos al revés (aunque no tiene mayor importancia)

135/145

© Rafael Rico López

## Repertorio de instrucciones x86-16bits

### 5.5.4. Comparaciones

#### • Ejemplo (I):

```
.MODEL large
.DATA
cadena1 DB "Más vale pájaro en mano que ciento volando"
.FARDATA
cadena2 DB "Más vale pájaro en mano que ciento saltando"
longitud EQU $ - cadena2

.CODE
MOV AX, @data           ;carga los segmentos
MOV DS, AX
MOV AX, @fardata
MOV ES, AX
: : :
: : :
```

136/145

© Rafael Rico López

## Repertorio de instrucciones x86-16bits

### 5.5.4. Comparaciones

- Ejemplo (II):

```

: : :
: : :
CLD                                ;dirección hacia arriba
MOV CX, longitud                   ;longitud al contador
MOV SI, OFFSET cadena1            ;inicializo SI
MOV DI, OFFSET cadena2            ;inicializo DI
REPE CMPSB                         ;repetir mientras igual
JZ todos=                          ;caso de que todo igual
DEC SI                            ;caso de carácter no igual
DEC DI
: : :
todos=: : :

```

también JCXZ; si CX = 0 he llegado  
al final sin desemparejamiento

© Rafael Rico López

137/145

## Repertorio de instrucciones x86-16bits

### 5.5.5. Paso de caracteres a cadenas

#### STOS

- ➔ Almacena un valor en cada posición de una cadena destino
- ➔ El valor a almacenar debe estar en Acc
- ➔ El prefijo a utilizar es REP

[REP] STOS [ES:]destino

[REP] STOB

[REP] STOW

© Rafael Rico López

138/145

## Repertorio de instrucciones x86-16bits

### 5.5.5. Paso de caracteres a cadenas

- Ejemplo:

```
.MODEL small
.DATA
destino DB 100 dup ?
.CODE
: : :

CLD
MOV AX, 'aa'
MOV CX, 50
MOV DI, OFFSET destino
REP STOSW
```

© Rafael Rico López

139/145

## Repertorio de instrucciones x86-16bits

### 5.5.6. Lectura de caracteres desde cadenas

#### LODS

- ➔ Carga un valor de una cadena en el Acc
- ➔ No se usa con prefijos de repetición

```
LODS [reg_seg:]fuente
LODSB
LODSW
```

© Rafael Rico López

140/145

## Repertorio de instrucciones x86-16bits

### 5.5.6. Lectura de caracteres desde cadenas

- Ejemplo:

```
.DATA
cadena DB 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
.CODE
: : :
CLD
MOV CX,10
MOV SI,OFFSET cadena
MOV AH,2          ;servicio 2 de la INT 21h
otro: LODSB
ADD AL,48          ;sumar 30h pasar a ASCII
MOV DL,AL          ;parámetro del servicio
INT 21h
LOOP otro
```

141/145

© Rafael Rico López

## Repertorio de instrucciones x86-16bits

### 5.5.7. E/S con cadenas

#### INS y OUTS

- ➔ INS lee un puerto y escribe en una cadena
- ➔ OUTS lee una cadena y la escribe en un puerto
- ➔ El puerto se da en DX (nunca inmediato) ya sea implícita o explícitamente

```
INS [ES:]destino,DX    OUTS DX,[reg_seg:]fuente
INSB                   OUTSB
INSW                   OUTSW
```

142/145

© Rafael Rico López

## Repertorio de instrucciones x86-16bits

### 5.5.7. E/S con cadenas

- Ejemplo:

```
contador    .DATA
            EQU 100
buffer      DB contador DUP (?)
            .CODE
            : : :
            CLD
            MOV CX,contador
            MOV DI,OFFSET buffer
            MOV DX,puerto
            REP INSB          ;transfiere la cadena
                               ;desde el puerto
```

© Rafael Rico López

143/145

## Repertorio de instrucciones x86-16bits

### 5.2. Instrucciones de control

- De manejo de las banderas

- ➔ CLC, CLD, CLI, CMC: borrar bandera de acarreo, dirección, interrupción o complementar acarreo
- ➔ STC, STD, STI: poner a 1 la bandera de acarreo, dirección o interrupción

© Rafael Rico López

144/145



## Repertorio de instrucciones x86-16bits

### 5.2. Instrucciones de control

- **De sincronización con coprocesadores (fp, E/S)**
  - ➔ **ESC**: indica al coprocesador que comience una tarea; el procesador principal no se detiene
  - ➔ **WAIT**: detiene al procesador principal hasta que, por ejemplo, se finalice el trabajo del coprocesador
  - ➔ **HLT**: detiene el procesador hasta que:
    - ➔ Se resetea el sistema
    - ➔ Se recibe una NMI o IRQ (si permitidas)
  - ➔ **NOP**: no hace nada; se pierden 3 ciclos y pasa a la siguiente instrucción (en realidad es **XCHG AX, AX**)