

JUEGO DE INSTRUCCIONES 80x86

4.1. - DESCRIPCIÓN COMPLETA DE LAS INSTRUCCIONES.

Nota: en el efecto de las instrucciones sobre el registro de estado se utilizará la siguiente notación:

- bit no modificado
- ? desconocido o indefinido
- x modificado según el resultado de la operación
- 1 puesto siempre a 1
- 0 puesto siempre a 0

4.1.1. - INSTRUCCIONES DE CARGA DE REGISTROS Y DIRECCIONES.

- MOV (transferencia)

Sintaxis: MOV dest, origen.

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Transfiere datos de longitud byte o palabra del operando origen al operando destino. Pueden ser operando origen y operando destino cualquier registro o posición de memoria direccionada de las formas ya vistas, con la única condición de que origen y destino tengan la misma dimensión. Existen ciertas limitaciones, como que los registros de segmento no admiten el direccionamiento inmediato: es incorrecto MOV DS,4000h; pero no lo es por ejemplo MOV DS,AX o MOV DS,VARIABLE. No es posible, así mismo, utilizar CS como destino (es incorrecto hacer MOV CS,AX aunque pueda admitirlo algún ensamblador). Al hacer MOV hacia un registro de segmento, las interrupciones quedan inhibidas hasta **después** de ejecutarse la siguiente instrucción (8086/88 de 1983 y procesadores posteriores).

Ejemplos:	mov	ds,ax
	mov	bx,es:[si]
	mov	si,offset dato

En el último ejemplo, no se coloca en SI el valor de la variable *dato* sino su dirección de memoria o desplazamiento respecto al segmento de datos. En otras palabras, SI es un puntero a *dato* pero no es *dato*. En el próximo capítulo se verá cómo se declaran las variables.

- XCHG (intercambiar)

Sintaxis: XCHG destino, origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Intercambia el contenido de los operandos origen y destino. No pueden utilizarse registros de segmentos como operandos.

Ejemplo:	xchg	bl, ch
	xchg	mem_pal, bx

- XLAT (traducción)

Sintaxis: XLAT tabla

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Se utiliza para traducir un byte del registro AL a un byte tomado de la tabla de traducción. Los datos se toman desde una dirección de la tabla correspondiente a BX + AL, donde bx es un puntero a el comienzo de la tabla y AL es un índice. Indicar *tabla* al lado de xlat es sólo una redundancia opcional.

Ejemplo:	mov	bx, offset tabla
	mov	al, 4
	xlat	

- LEA (carga dirección efectiva)

Sintaxis: LEA destino, origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Transfiere el desplazamiento del operando fuente al operando destino. Otras instrucciones pueden a continuación utilizar el registro como desplazamiento para acceder a los datos que constituyen el objetivo. El operando destino no puede ser un registro de segmento. En general, esta instrucción es equivalente a *MOV destino, OFFSET fuente* y de hecho los buenos ensambladores (TASM) la codifican como MOV para economizar un byte de memoria. Sin embargo, LEA es en algunos casos más potente que MOV al permitir indicar registros de índice y desplazamiento para calcular el offset:

```
lea    dx, datos[si]
```

En el ejemplo de arriba, el valor depositado en DX es el offset de la etiqueta *datos* más el registro SI. Esa sola instrucción es equivalente a estas dos:

```
mov    dx, offset datos
```


Transfiere el contenido de los bits 7, 6, 4, 2 y 0 a los indicadores SF, ZF, AF, PF y CF respectivamente.

4.1.2. - INSTRUCCIONES DE MANIPULACIÓN DEL REGISTRO DE ESTADO.

- CLC (baja el indicador de acarreo)

Sintaxis: CLC

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	0

Borra el indicador de acarreo (CF) sin afectar a ninguno otro.

- CLD (baja el indicador de dirección)

Sintaxis: CLD

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	0	-	-	-	-	-	-	-

Pone a 0 el indicador de dirección DF, por lo que los registros SI y/o DI se autoincrementan en las operaciones de cadenas, sin afectar al resto de los indicadores. Es NECESARIO colocarlo antes de las instrucciones de manejo de cadenas si no se conoce con seguridad el valor de DF. Véase STD.

- CLI (baja indicador de interrupción)

Sintaxis: CLI

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	0	-	-	-	-	-	-

Borra el indicador de activación de interrupciones IF, lo que desactiva las interrupciones enmascarables. Es muy conveniente hacer esto antes de modificar la pareja SS:SP en los 8086/88 anteriores a 1983 (véase comentario en la instrucción MOV), o antes de cambiar un vector de interrupción sin el apoyo del DOS. Generalmente las interrupciones sólo se inhiben por breves instantes en momentos críticos. Véase también STI.

- CMC (complementa el indicador de acarreo)

Sintaxis: CMC

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	x

Complementa el indicador de acarreo CF invirtiendo su estado.

- STC (pone a uno el indicador de acarreo)

Sintaxis: STC

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	1

Pone a 1 el indicador de acarreo CF sin afectar a ningún otro indicador.

- STD (pone a uno el indicador de dirección)

Sintaxis: STD

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	1	-	-	-	-	-	-	-

Pone a 1 el indicador de dirección DF, por lo que los registros SI y/o DI se autodecrementan en las operaciones de cadenas, sin afectar al resto de los indicadores. Es NECESARIO colocarlo antes de las instrucciones de manejo de cadenas si no se conoce con seguridad el estado de DF. Véase también CLD.

- STI (pone a uno el indicador de interrupción)

Sintaxis: STI

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	1	-	-	-	-	-	-

Pone a 1 la bandera de desactivación de interrupciones IF y activa las interrupciones enmascarables. Una interrupción pendiente no es reconocida, sin embargo, hasta después de ejecutar la instrucción que sigue a STI. Véase también CLI.

4.1.3. - INSTRUCCIONES DE MANEJO DE LA PILA.

- POP (extraer de la pila)

Sintaxis: POP destino

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Transfiere el elemento palabra que se encuentra en lo alto de la pila (apuntado por SP) al operando destino que a de ser tipo palabra, e incrementa en dos el registro SP. La instrucción POP CS, poco útil, no funciona correctamente en los 286 y superiores.

Ejemplos:	pop	ax
	pop	pepe

- PUSH (introduce en la pila)

Sintaxis: PUSH origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Decrementa el puntero de pila (SP) en 2 y luego transfiere la palabra especificada en el operando origen a la cima de la pila. El registro CS aquí sí se puede especificar como origen, al contrario de lo que afirman algunas publicaciones.

Ejemplo:	push	cs
----------	------	----

- POPF (extrae los indicadores de la pila)

Sintaxis: POPF

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	x	x	x	x	x	x	x	x	x

Traslada al registro de los indicadores la palabra almacenada en la cima de la pila; a continuación el puntero de pila SP se incrementa en dos.

- PUSHF (introduce los indicadores en la pila)

Sintaxis: PUSHF

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Decrementa en dos el puntero de pila y traslada a la cima de la pila el contenido de los indicadores.

4.1.4. - INSTRUCCIONES DE TRANSFERENCIA DE CONTROL.

Incondicional

- CALL (llamada a subrutina)

Sintaxis: CALL destino

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Transfiere el control del programa a un procedimiento, salvando previamente en la pila la dirección de la instrucción siguiente, para poder volver a ella una vez ejecutado el procedimiento. El procedimiento puede estar en el mismo segmento (tipo NEAR) o en otro segmento (tipo FAR). A su vez la llamada puede ser directa a una etiqueta (especificando el tipo de llamada NEAR -por defecto- o FAR) o indirecta, indicando la dirección donde se encuentra el puntero. Según la llamada sea cercana o lejana, se almacena en la pila una dirección de retorno de 16 bits o dos palabras de 16 bits indicando en este último caso tanto el offset (IP) como el segmento (CS) a donde volver.

Ejemplos:	call	procl
	dir	dd 0f000e987h
	call	dword ptr dir

En el segundo ejemplo, la variable dir almacena la dirección a donde saltar. De esta última manera -conociendo su dirección- puede llamarse también a un vector de interrupción, guardando previamente los flags en la pila (PUSHF), porque la rutina de interrupción retornará (con IRET en vez de con RETF) sacándolos.

- JMP (salto)

Sintaxis: JMP dirección o JMP SHORT dirección

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Transfiere el control incondicionalmente a la dirección indicada en el operando. La bifurcación puede ser también directa o indirecta como anteriormente vimos, pero además puede ser corta (tipo SHORT) con un desplazamiento comprendido entre -128 y 127; o larga, con un desplazamiento de dos bytes con signo. Si se hace un JMP SHORT y no llega el salto (porque está demasiado alejada esa etiqueta) el ensamblador dará error. Los buenos ensambladores (como TASM) cuando dan dos pasadas colocan allí donde es posible un salto corto, para economizar memoria, sin que

el programador tenga que ocuparse de poner *short*. Si el salto de dos bytes, que permite desplazamientos de 64 Kb en la memoria sigue siendo insuficiente, se puede indicar con *far* que es largo (salto a otro segmento).

Ejemplos: jmp etiqueta
 jmp far ptr etiqueta

- RET / RETF (retorno de subrutina)

Sintaxis: RET [valor] o RETF [valor]

Indicadores: OF DF IF TF SF ZF AF PF CF
 - - - - - - - - -

Retorna de un procedimiento extrayendo de la pila la dirección de la siguiente dirección. Se extraerá el registro de segmento y el desplazamiento en un procedimiento de tipo FAR (dos palabras) y solo el desplazamiento en un procedimiento NEAR (una palabra). si esta instrucción es colocada dentro de un bloque PROC-ENDP (como se verá en el siguiente capítulo) el ensamblador sabe el tipo de retorno que debe hacer, según el procedimiento sea NEAR o FAR. En cualquier caso, se puede forzar que el retorno sea de tipo FAR con la instrucción RETF. *Valor*, si es indicado permite sumar una cantidad *valor* en bytes a SP antes de retornar, lo que es frecuente en el código generado por los compiladores para retornar de una función con parámetros. También se puede retornar de una interrupción con RETF 2, para que devuelva el registro de estado sin restaurarlo de la pila.

Condicional

Las siguientes instrucciones son de transferencia condicional de control a la instrucción que se encuentra en la posición IP+desplazamiento (desplazamiento comprendido entre -128 y +127) si se cumple la condición. Algunas condiciones se pueden denotar de varias maneras. Todos los saltos son cortos y *si no alcanza* hay que apañárselas como sea. En **negrita** se realizan las condiciones más empleadas. Donde interviene SF se consideran con signo los operandos implicados en la última comparación u operación aritmético-lógica, y se indican en la tabla como '±' (-128 a +127 ó -32768 a +32767); en los demás casos, indicados como '+', se consideran sin signo (0 a 255 ó 0 a 65535):

JA/JNBE	Salto si mayor (above), si no menor o igual (not below or equal), si CF=0 y ZF=0.	+
JAE/JNB	Salto si mayor o igual (above or equal), si no menor (not below), si CF=0.	+
JB/JNAE/JC	Salto si menor (below), si no superior ni igual (not above or equal), si acarreo, si CF=1.	+
JBE/JNA	Salto si menor o igual (not below or equal), si no mayor (not above), si CF=1 ó ZF=1.	+
JCXZ	Salto si CX=0.	
JE/JZ	Salto si igual (equal), si cero (zero), si ZF=1.	
JG/JNLE	Salto si mayor (greater), si no menor ni igual (not less or equal), si ZF=0 y SF=0.	±

JGE/JNL	Salto si mayor o igual (greater or equal), si no menor (not less), si SF=0.	±
JL/JNGE	Salto si menor (less), si no mayor ni igual (not greater or equal), si SF<>OF.	±
JLE/JNG	Salto si menor o igual (less or equal), si no mayor (not greater), si ZF=0 y SF<>OF.	±
JNC	Salto si no acarreo, si CF=0.	
JNE/JNZ	Salto si no igual, si no cero, si ZF=0.	
JNO	Salto si no desbordamiento, si OF=0.	
JNP/JPO	Salto si no paridad, si paridad impar, si PF=0.	
JNS	Salto si no signo, si positivo, si SF=0.	
JO	Salto si desbordamiento, si OF=1.	
JP/JPE	Salto si paridad, si paridad par, si PF=1.	
JS	Salto si signo, si SF=1.	

Gestión de bucle

- LOOP (bucle)

Sintaxis: LOOP desplazamiento

Indicadores: OF DF IF TF SF ZF AF PF CF
 - - - - - - - - -

Decrementa el registro contador CX; si CX es cero, ejecuta la siguiente instrucción, en caso contrario transfiere el control a la dirección resultante de sumar a IP + desplazamiento. El desplazamiento debe estar comprendido entre -128 y +127.

Ejemplo:

```

                mov  cx,10
bucle:  ....
                ....
                loop bucle

```

Con las mismas características que la instrucción anterior:

- LOOPE/LOOPZ Bucle si igual, si cero. Z=1 y CX<>0
- LOOPNE/LOOPNZ Bucle si no igual, si no cero. Z=0 y CX<>0

Interrupciones

- INT (interrupción)

Sintaxis: INT n (0 <= n <= 255)

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	0	0	-	-	-	-	-

Inicializa un procedimiento de interrupción de un tipo indicado en la instrucción. En la pila se introduce al llamar a una interrupción la dirección de retorno formada por los registros CS e IP y el estado de los indicadores. INT 3 es un caso especial de INT, al ensamblarla el ensamblador genera un sólo byte en vez de los dos habituales; esta interrupción se utiliza para poner *puntos de ruptura* en los programas. Véase también IRET y el apartado 1 del capítulo VII.

Ejemplo: int 21h

- INTO (interrupción por desbordamiento)

Sintaxis: INTO

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	0	0	-	-	-	-	-

Genera una interrupción de tipo 4 (INT 4) si existe desbordamiento (OF=1). De lo contrario se continúa con la instrucción siguiente.

- IRET (retorno de interrupción)

Sintaxis: IRET

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	x	x	x	x	x	x	x	x	x

Devuelve el control a la dirección de retorno salvada en la pila por una interrupción previa y restaura los indicadores que también se introdujeron en la pila. En total, se sacan las 3 palabras que fueron colocadas en la pila cuando se produjo la interrupción. Véase también INT.

4.1.5. - INSTRUCCIONES DE ENTRADA SALIDA (E/S).

- IN (entrada)

Sintaxis: IN acumulador, puerto.

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Transfiere datos desde el puerto indicado hasta el registro AL o AX, dependiendo de la longitud byte o palabra respectivamente. El puerto puede especificarse mediante una constante (0 a 255) o a través del valor contenido en DX (0 a 65535).

Ejemplo: in ax,0fh
 in al,dx

- OUT (salida)

Sintaxis: OUT puerto, acumulador

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Transfiere un byte o palabra del registro AL o AX a un puerto de salida. El puerto puede especificarse con un valor fijo entre 0 y 255 ó a través del valor contenido en el registro DX (de 0 a 65535).

Ejemplo: out 12h,ax
 out dx,al

4.1.6. - INSTRUCCIONES ARITMÉTICAS.

* * * S U M A * * *

- AAA (ajuste ASCII para la suma)

Sintaxis: AAA

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	?	-	-	-	?	?	x	?	x

Convierte el contenido del registro AL en un número BCD no empaquetado. Si los cuatro bits menos significativos de AL son mayores que 9 ó si el indicador AF está a 1, se suma 6 a AL, 1 a AH, AF se pone a 1, CF se iguala a AF y AL pone sus cuatro bits más significativos a 0.

Ejemplo: add al,b1
 aaa

En el ejemplo, tras la suma de dos números BCD no empaquetados colocados en AL y BL, el resultado (por medio de AAA) sigue siendo un número BCD no empaquetado.

- ADC (suma con acarreo)

Sintaxis: ADC destino, origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	x	-	-	-	x	x	x	x	x

Suma los operandos origen, destino y el valor del indicador de acarreo (0 ó 1) y el resultado lo almacena en el operando destino. Se utiliza normalmente para sumar números grandes, de más de 16 bits, en varios pasos, considerando *lo que nos llevamos* (el acarreo) de la suma anterior.

Ejemplo: adc ax, bx

- ADD (suma)

Sintaxis: ADD destino, origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	x	-	-	-	x	x	x	x	x

Suma los operandos origen y destino almacenando el resultado en el operando destino. Se activa el acarreo si se desborda el registro destino durante la suma.

Ejemplos: add ax, bx
 add cl, dh

- DAA (ajuste decimal para la suma)

Sintaxis: DAA

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	?	-	-	-	x	x	x	x	x

Convierte el contenido del registro AL en un par de valores BCD: si los cuatro bits menos significativos de AL son un número mayor que 9, el indicador AF se pone a 1 y se suma 6 a AL. De igual forma, si los cuatro bits más significativos de AL tras la operación anterior son un número mayor que 9, el indicador CF se pone a 1 y se suma 60h a AL.

Ejemplo: add al, cl
 daa

En el ejemplo anterior, si AL y CL contenían dos números BCD empaquetados, DAA hace que el resultado de la suma (en AL) siga siendo también un BCD empaquetado.

- INC (incrementar)

Sintaxis: INC destino

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	x	-	-	-	x	x	x	x	-

Incrementa el operando destino. El operando destino puede ser byte o palabra. Obsérvese que esta instrucción no modifica el bit de acarreo (CF) y no es posible detectar un desbordamiento por este procedimiento (utilícese ZF).

Ejemplos:	inc	al
	inc	es:[di]
	inc	ss:[bp+4]
	inc	word ptr cs:[bx+di+7]

* * * R E S T A * * *

- AAS (ajuste ASCII para la resta)

Sintaxis: AAS

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	?	-	-	-	?	?	x	?	x

Convierte el resultado de la sustracción de dos operandos BCD no empaquetados para que siga siendo un número BCD no empaquetado. Si el nibble inferior de AL tiene un valor mayor que 9, de AL se resta 6, se decrementa AH, AF se pone a 1 y CF se iguala a AF. El resultado se guarda en AL con los bits de 4 a 7 puestos a 0.

Ejemplo:	sub	al,bl
	aas	

En el ejemplo, tras la resta de dos números BCD no empaquetados colocados en AL y BL, el resultado (por medio de AAS) sigue siendo un número BCD no empaquetado.

- CMP (comparación)

Sintaxis: CMP destino, origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	x	-	-	-	x	x	x	x	x

Calcula el valor negativo en complemento a dos del operando y devuelve el resultado en el mismo operando.

Ejemplo: neg al

- SBB (resta con acarreo)

Sintaxis: SBB destino, origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	x	-	-	-	x	x	x	x	x

Resta el operando origen del operando destino y el resultado lo almacena en el operando destino. Si está a 1 el indicador de acarreo además resta una unidad más. Los operandos pueden ser de tipo byte o palabra. Se utiliza normalmente para restar números grandes, de más de 16 bits, en varios pasos, considerando *lo que nos llevamos* (el acarreo) de la resta anterior.

Ejemplo: sbb ax, ax
 sbb ch, dh

- SUB (resta)

Sintaxis: SUB destino, origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	x	-	-	-	x	x	x	x	x

Resta el operando destino al operando origen, colocando el resultado en el operando destino. Los operandos pueden tener o no signo, siendo necesario que sean del mismo tipo, byte o palabra.

Ejemplos: sub al, bl
 sub dx, dx

*** MULTIPLICACION ***

- AAM (ajuste ASCII para la multiplicación)

Sintaxis: AAM

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	?	-	-	-	x	x	?	x	?

Corrige el resultado en AX del producto de dos números BCD no empaquetados, convirtiéndolo en un valor BCD también no empaquetado. En AH sitúa el cociente de AL/10 quedando en AL el resto de dicha operación.

```
Ejemplo:      mul    bl
              aam
```

En el ejemplo, tras el producto de dos números BCD no empaquetados colocados en AL y BL, el resultado (por medio de AAA) sigue siendo, en AX, un número BCD no empaquetado.

- IMUL (multiplicación entera con signo)

Sintaxis: IMUL origen (*origen* no puede ser operando inmediato en 8086, sí en 286)

```
Indicadores:  OF  DF  IF  TF  SF  ZF  AF  PF  CF
              x   -   -   -   ?   ?   ?   ?   x
```

Multiplica un operando origen con signo de longitud byte o palabra por AL o AX respectivamente. Si *origen* es un byte el resultado se guarda en AH (byte más significativo) y en AL (menos significativo), si *origen* es una palabra el resultado es devuelto en DX (parte alta) y AX (parte baja). Si las mitades más significativas son distintas de cero, independientemente del signo, CF y OF son activados.

```
Ejemplo:      imul   bx
              imul   ch
```

- MUL (multiplicación sin signo)

Sintaxis: MUL origen (*origen* no puede ser operando inmediato)

```
Indicadores:  OF  DF  IF  TF  SF  ZF  AF  PF  CF
              x   -   -   -   ?   ?   ?   ?   x
```

Multiplica el contenido sin signo del acumulador por el operando origen. Si el operando destino es un byte el acumulador es AL guardando el resultado en AH y AL, si el contenido de AH es distinto de 0 activa los indicadores CF y OF. Cuando el operando origen es de longitud palabra el acumulador es AX quedando el resultado sobre DX y AX, si el valor de DX es distinto de cero los indicadores CF y OF se activan.

```
Ejemplo:      mul    byte ptr ds:[di]
              mul    dx
              mul    cl
```

***** DIVISION *****

- AAD (ajuste ASCII para la división)

Sintaxis: AAD

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	?	-	-	-	x	x	?	x	?

Convierte dos números BCD no empaquetados contenidos en AH y AL en un dividendo de un byte que queda almacenado en AL. Tras la operación AH queda a cero. Esta instrucción es necesaria ANTES de la operación de dividir, al contrario que AAM.

Ejemplo: aad
 div bl

En el ejemplo, tras convertir los dos números BCD no empaquetados (en AX) en un dividendo válido, la instrucción de dividir genera un resultado correcto.

- DIV (división sin signo)

Sintaxis: DIV origen (*origen* no puede ser operando inmediato)

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	?	-	-	-	?	?	?	?	?

Divide, sin considerar el signo, un número contenido en el acumulador y su extensión (AH, AL si el operando es de tipo byte o DX, AX si el operando es palabra) entre el operando fuente. El cociente se guarda en AL o AX y el resto en AH o DX según el operando sea byte o palabra respectivamente. DX o AH deben ser cero antes de la operación. Cuando el cociente es mayor que el resultado máximo que puede almacenar, cociente y resto quedan indefinidos produciéndose una interrupción 0. En caso de que las partes más significativas del cociente tengan un valor distinto de cero se activan los indicadores CF y OF.

Ejemplo: div bl
 div mem_pal

- IDIV (división entera)

Sintaxis: IDIV origen (*origen* no puede ser operando inmediato)

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	?	-	-	-	?	?	?	?	?

Divide, considerando el signo, un número contenido en el acumulador y su extensión entre el operando fuente. El cociente se almacena en AL o AX según el operando sea byte o palabra y de igual manera el resto en AH o DX. DX o AH deben ser cero antes de la operación. Cuando el cociente es positivo y superior al valor máximo que puede almacenarse (7fh ó 7fffh), o cuando el cociente es negativo e

inferior al valor mínimo que puede almacenarse (81h u 8001h) entonces cociente y resto quedan indefinidos, generándose una interrupción 0, lo que también sucede si el divisor es 0.

Ejemplo: idiv bl
 idiv bx

***** CONVERSIONES *****

- CBW (conversión de byte en palabra)

Sintaxis: CBW

Indicadores: OF DF IF TF SF ZF AF PF CF
 - - - - - - - - -

Copia el bit 7 del registro AL en todos los bits del registro AH, es decir, expande el signo de AL a AX como paso previo a una operación de 16 bits.

- CWD (conversión de palabra a doble palabra)

Sintaxis: CWD

Indicadores: OF DF IF TF SF ZF AF PF CF
 - - - - - - - - -

Expande el signo del registro AX sobre el registro DX, copiando el bit más significativo de AH en todo DX.

4.1.7. - INSTRUCCIONES DE MANIPULACIÓN DE CADENAS.

- CMPS/CMPSB/CMPSW (compara cadenas)

Sintaxis: CMPS cadena_destino, cadena_origen
 CMPSB (bytes)
 CMPSW (palabras)

Indicadores: OF DF IF TF SF ZF AF PF CF
 x - - - x x x x x

Compara dos cadenas restando al origen el destino. Ninguno de los operandos se alteran, pero los indicadores resultan afectados. La cadena origen se direcciona con

registro SI sobre el segmento de datos DS y la cadena destino se direcciona con el registro DI sobre el segmento extra ES. Los registros DI y SI se autoincrementan o autodecrementan según el valor del indicador DF (véanse CLD y STD) en una o dos unidades, dependiendo de si se trabaja con bytes o con palabras. *Cadena origen* y *cadena destino* son dos operandos redundantes que sólo indican el tipo del dato (byte o palabra) a comparar, es más cómodo colocar CMPSB o CMPSW para indicar bytes/palabras. Si se indica un registro de segmento, éste sustituirá en la cadena origen al DS ordinario. Ejemplo:

```
lea    si,origen
lea    di,destino
cmpsb
```

- LODS/LODSB/LODSW (cargar cadena)

Sintaxis: LODS cadena_origen
LODSB (bytes)
LODSW (palabras)

Indicadores: OF DF IF TF SF ZF AF PF CF
 - - - - - - - - -

Copia en AL o AX una cadena de longitud byte o palabra direccionada sobre el segmento de datos (DS) con el registro SI. Tras la transferencia, SI se incrementa o decrementa según el indicador DF (véanse CLD y STD) en una o dos unidades, según se estén manejando bytes o palabras. *Cadena origen* es un operando redundante que sólo indica el tipo del dato (byte o palabra) a cargar, es más cómodo colocar LODSB o LODSW para indicar bytes/palabras.

Ejemplo: cld
 lea si,origen
 lodsrb

- MOVS/MOVSb/MOVSW (mover cadena)

Sintaxis: MOVS cadena_destino, cadena_origen
MOVSb (bytes)
MOVSW (palabras)

Indicadores: OF DF IF TF SF ZF AF PF CF
 - - - - - - - - -

Transfiere un byte o una palabra de la cadena origen direccionada por DS:SI a la cadena destino direccionada por ES:DI, incrementando o decrementando a continuación los registros SI y DI según el valor de DF (véanse CLD y STD) en una o dos unidades, dependiendo de si se trabaja con bytes o con palabras. *Cadena origen* y *cadena destino* son dos operandos redundantes que sólo indican el tipo del dato (byte o palabra) a comparar, es más cómodo colocar MOVSb o MOVSW para indicar bytes/palabras. Si se indica un registro de segmento, éste sustituirá en la cadena origen al DS ordinario.

```
Ejemplo:      lea    si,origen
               lea    di,destino
               movsw
```

- SCAS/SCASB/SCASW (explorar cadena)

Sintaxis: SCAS cadena_destino
 SCASB (bytes)
 SCASW (palabras)

```
Indicadores:  OF  DF  IF  TF  SF  ZF  AF  PF  CF
               x  -  -  -  x  x  x  x  x
```

Resta de AX o AL una cadena destino direccionada por el registro DI sobre el segmento extra. Ninguno de los valores es alterado pero los indicadores se ven afectados. DI se incrementa o decrementa según el valor de DF (véanse CLD y STD) en una o dos unidades -según se esté trabajando con bytes o palabras- para apuntar al siguiente elemento de la cadena. *Cadena_destino* es un operando redundante que sólo indica el tipo del dato (byte o palabra), es más cómodo colocar SCASB o SCASW para indicar bytes/palabras.

```
Ejemplo:      lea    di,destino
               mov    al,50
               scasb
```

- STOS/STOSB/STOSW (almacena cadena)

Sintaxis: STOS cadena_destino
 STOSB (bytes)
 STOSW (palabras)

```
Indicadores:  OF  DF  IF  TF  SF  ZF  AF  PF  CF
               -  -  -  -  -  -  -  -  -
```

Transfiere el operando origen almacenado en AX o AL, al destino direccionado por el registro DI sobre el segmento extra. Tras la operación, DI se incrementa o decrementa según el indicador DF (véanse CLD y STD) para apuntar al siguiente elemento de la cadena. *Cadena_destino* es un operando redundante que sólo indica el tipo del dato (byte o palabra) a cargar, es más cómodo colocar STOSB o STOSW para indicar bytes/palabras.

```
Ejemplo:      lea    di,destino
               mov    ax,1991
               stosw
```

- REP/REPE/REPZ/REPNE/REPZ (repetir)

REP repetir operación de cadena
 REPE/REPZ repetir operación de cadena si igual/si cero
 REPNE/REPZ repetir operación de cadena si no igual (si no 0)

Estas instrucciones se pueden colocar como prefijo de otra instrucción de manejo de cadenas, con objeto de que la misma se repita un número determinado de veces incondicionalmente o hasta que se verifique alguna condición. El número de veces se indica en CX. Por sentido común sólo deben utilizarse las siguientes combinaciones:

Instrucciones	Prefijo	Función
--	REP	Repetir CX veces
	REPE/REPZ	Repetir CX veces mientras ZF=1
	REPNE/REPZ	Repetir CX veces mientras ZF=0
		MOVS, STOS
		CMPS, SCAS
		CMPS, SCAS

Ejemplos:

1) Buscar el byte 69 entre las 200 primeras posiciones de *tabla* (se supone *tabla* en el segmento ES):

```
LEA    DI,tabla
MOV    CX,200
MOV    AL,69
CLD
REPNE  SCASB
JE     encontrado
```

2) Rellenar de ceros 5000 bytes de una tabla colocada en *datos* (se supone *datos* en el segmento ES):

```
LEA    DI,datos
MOV    AX,0
MOV    CX,2500
CLD
REP    STOSW
```

3) Copiar la memoria de pantalla de texto (adaptador de color) de un PC en un buffer (se supone *buffer* en el segmento ES):

```
MOV    CX,0B800h    ; segmento de pantalla
MOV    DS,CX        ; en DS
LEA    DI,buffer    ; destino en ES:DI
MOV    SI,0         ; copiar desde DS:0
MOV    CX,2000     ; 2000 palabras
CLD                ; hacia adelante
REP    MOVSW        ; copiar CX palabras
```

4.1.8. - INSTRUCCIONES DE OPERACIONES LÓGICAS A NIVEL DE BIT.

- AND (y lógico)

Sintaxis: AND destino, origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	0	-	-	-	x	x	?	x	0

Realiza una operación de Y lógico entre el operando origen y destino quedando el resultado en el destino. Son válidos operandos byte o palabra, pero ambos del mismo tipo.

Ejemplos: and ax,bx
 and bl,byte ptr es:[si+10h]

- NOT (no lógico)

Sintaxis: NOT destino

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Realiza el complemento a uno del operando destino, invirtiendo cada uno de sus bits. Los indicadores no resultan afectados.

Ejemplo: not ax

- OR (O lógico)

Sintaxis: OR destino, origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	0	-	-	-	x	x	?	x	0

Realiza una operación O lógico a nivel de bits entre los dos operandos, almacenándose después el resultado en el operando destino.

Ejemplo: or ax,bx

- TEST (comparación lógica)

Sintaxis: TEST destino, origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	0	-	-	-	x	x	?	x	0

Realiza una operación Y lógica entre los dos operandos pero sin almacenar el resultado. Los indicadores son afectados con la operación.

Ejemplo: test al,bh

- XOR (O exclusivo)

Sintaxis: XOR destino, origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	0	-	-	-	x	x	?	x	0

Operación OR exclusivo a nivel de bits entre los operandos origen y destino almacenándose el resultado en este último.

Ejemplo: xor di,ax

4.1.9. - INSTRUCCIONES DE CONTROL DEL PROCESADOR.

- NOP (operación nula)

Sintaxis: NOP

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Realiza una operación nula, es decir, el microprocesador decodifica la instrucción y pasa a la siguiente. Realmente se trata de la instrucción XCHG AX,AX.

- ESC (salida a un coprocesador)

Sintaxis: ESC código_operación, origen

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

Se utiliza en combinación con procesadores externos, tales como los coprocesadores de coma flotante o de E/S, y abre al dispositivo externo el acceso a las direcciones y operandos requeridos. Al mnemónico ESC le siguen los códigos de operación apropiados para el coprocesador así como la instrucción y la dirección del operando necesario.

Ejemplo: esc 21,ax

- HLT (parada hasta interrupción o reset)

Sintaxis: HLT

Indicadores:	OF	DF	IF	TF	SF	ZF	AF	PF	CF
	-	-	-	-	-	-	-	-	-

El procesador se detiene hasta que se restaura el sistema o se recibe una interrupción. Como en los PC se producen normalmente 18,2 interrupciones de tipo 8 por segundo (del temporizador) algunos programadores utilizan HLT para hacer pausas y bucles de retardo. Sin embargo, el método no es preciso y puede fallar con ciertos controladores de memoria.

- LOCK (bloquea los buses)

Sintaxis: LOCK

Indicadores: OF DF IF TF SF ZF AF PF CF
 - - - - - - - - -

Es una instrucción que se utiliza en aplicaciones de recursos compartidos para asegurar que no accede simultáneamente a la memoria más de un procesador. Cuando una instrucción va precedida por LOCK, el procesador bloquea inmediatamente el bus, introduciendo una señal por la patilla LOCK.

- WAIT (espera)

Sintaxis: WAIT

Indicadores: OF DF IF TF SF ZF AF PF CF
 - - - - - - - - -

Provoca la espera del procesador hasta que se detecta una señal en la patilla TEST. Ocurre, por ejemplo, cuando el copro ha terminado una operación e indica su finalización. Suele preceder a ESC para sincronizar las acciones del procesador y coprocesador.

4.1.10. - INSTRUCCIONES DE ROTACIÓN Y DESPLAZAMIENTO.

- RCL (rotación a la izquierda con acarreo)

Sintaxis: RCL destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF
 x - - - - - - - x

Rotar a la izquierda los bits del operando destino junto con el indicador de acarreo CF el número de bits especificado en el segundo operando. Si el número de bits a desplazar es 1, se puede especificar directamente, en caso contrario el valor debe cargarse en CL y especificar CL como segundo operando. No es conveniente que CL sea mayor de 7, en bytes; ó 15, en palabras.



Ejemplos: rcl ax,1
 rcl al,cl
 rcl di,1

- RCR (rotación a la derecha con acarreo)

Sintaxis: RCR destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF
 x - - - - - - - - x

Rotar a la derecha los bits del operando destino junto con el indicador de acarreo CF el número de bits especificado en el segundo operando. Si el número de bits es 1 se puede especificar directamente; en caso contrario su valor debe cargarse en CL y especificar CL como segundo operando:



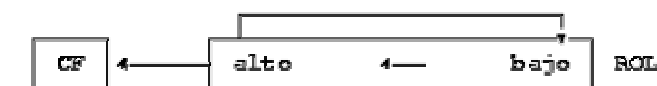
Ejemplos: rcr bx,cl
 rcr bx,1

- ROL (rotación a la izquierda)

Sintaxis: ROL destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF
 x - - - - - - - - x

Rota a la izquierda los bits del operando destino el número de bits especificado en el segundo operando, que puede ser 1 ó CL previamente cargado con el valor del número de veces.



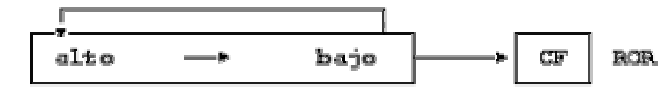
Ejemplos: rol dx,cl
 rol ah,1

- ROR (rotación a la derecha)

Sintaxis: ROR destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF
 x - - - - - - - - x

Rota a la derecha los bits del operando destino el número de bits especificado en el segundo operando. Si el número de bits es 1 se puede poner directamente, en caso contrario debe ponerse a través de CL.



Ejemplos: ror cl,1
 ror ax,cl

- SAL/SHL (desplazamiento aritmético a la izquierda)

Sintaxis: SAL/SHL destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF
 x - - - x x ? x x

Desplaza a la izquierda los bits del operando el número de bits especificado en el segundo operando que debe ser CL si es mayor que 1 los bits desplazados.

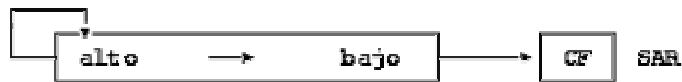


- SAR (desplazamiento aritmético a la derecha)

Sintaxis: SAR destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF
 x - - - x x ? x x

Desplaza a la derecha los bits del operando destino el número de bits especificado en el segundo operando. Los bits de la izquierda se rellenan con el bit de signo del primer operando. Si el número de bits a desplazar es 1 se puede especificar directamente, si es mayor se especifica a través de CL.



Ejemplos: sar ax,cl
 sar bp,1

- SHR (desplazamiento lógico a la derecha)

Sintaxis: SHR destino, contador

Indicadores: OF DF IF TF SF ZF AF PF CF
 x - - - x x ? x x

Desplaza a la derecha los bits del operando destino el número de los bits especificados en el segundo operando. Los bits de la izquierda se llena con cero. Si el número de bits a desplazar es 1 se puede especificar directamente en el caso en que no ocurra se pone el valor en CL:



Ejemplos: shr ax,cl

x	-	DEC dst	DEC	dst	x	-	-	-	x	x	x
?	?	DIV fnt	DIV	dst	?	-	-	-	?	?	?
-	-	ESC opcode, fnt	ESC	opcode, fnt	-	-	-	-	-	-	-
-	-	HLT	HLT		-	-	-	-	-	-	-
?	?	IDIV fnt	IDIV	fnt	?	-	-	-	?	?	?
?	x	IMUL fnt	IMUL	fnt	x	-	-	-	?	?	?
-	-	IN acum, port	IN	acum, port	-	-	-	-	-	-	-
x	-	INC dst	INC	dst	x	-	-	-	x	x	x
-	-	INT interrup	INT	interrup	-	-	0	0	-	-	-
-	-	INTO	INTO		-	-	0	0	-	-	-
x	x	IRET	IRET		x	x	x	x	x	x	x
-	-	Jcc (JA, JBE...)	Jcc	dsp	-	-	-	-	-	-	-
-	-	JMP	JMP	dsp	-	-	-	-	-	-	-
-	-	JCXZ dsp	JCXZ	dsp	-	-	-	-	-	-	-
-	-	LAHF	LAHF		-	-	-	-	-	-	-
-	-	LDS dst, fnt	LDS	dst, fnt	-	-	-	-	-	-	-
-	-	LEA dst, fnt	LEA	dst, fnt	-	-	-	-	-	-	-
-	-	LES dst, fnt	LES	dst, fnt	-	-	-	-	-	-	-
-	-	LOCK	LOCK		-	-	-	-	-	-	-
-	-	LODS/LODSB/ LODSW cfnt	LODS	mem	-	-	-	-	-	-	-
-	-	LOOP	LOOP	dsp	-	-	-	-	-	-	-
-	-	LOOPcc (LOOPE...)	LOOPcc	dsp	-	-	-	-	-	-	-
-	-	MOV dst, fnt	MOV	dst, fnt	-	-	-	-	-	-	-
-	-	MOVS/MOVSb/ MOVSW cdst, cfnt	MOVS	cdst, cfnt	-	-	-	-	-	-	-
?	x	MUL fnt	MUL	fnt	x	-	-	-	?	?	?
x	x	NEG dst	NEG	fnt	x	-	-	-	x	x	x
-	-	NOP	NOP		-	-	-	-	-	-	-
-	-	NOT dst	NOT	dst	-	-	-	-	-	-	-
x	0	OR dst, fnt	OR	dst, fnt	0	-	-	-	x	x	?

-	-	OUT port,acum	OUT	port,acum	-	-	-	-	-	-	-
-	-	POP dst	POP	dst	-	-	-	-	-	-	-
x	x	POPF	POPF		x	x	x	x	x	x	x
-	-	PUSH dst	PUSH	dst	-	-	-	-	-	-	-
-	-	PUSHF	PUSHF		-	-	-	-	-	-	-
-	x	RCL dst,cnt	RCL	dst,cnt	x	-	-	-	-	-	-
-	x	RCR dst,cnt	RCR	dst,cnt	x	-	-	-	-	-	-
-	-	REP/REPE/REPZ/ REPNE/REPZ	REP		-	-	-	-	-	-	-
-	-	RET [val]	RET	[val]	-	-	-	-	-	-	-
-	-	RETF [val]	RETF	[val]	-	-	-	-	-	-	-
-	x	ROL dst,cnt	ROL	dst,cnt	x	-	-	-	-	-	-
-	x	ROR dst,cnt	ROR	dst,cnt	x	-	-	-	-	-	-
x	x	SAHF	SAHF		-	-	-	-	x	x	x
x	x	SAL/SHL dst,cnt	SAL	dst,cnt	x	-	-	-	x	x	?
x	x	SAR dst,cnt	SAR	dst,cnt	x	-	-	-	x	x	?
x	x	SBB dst,fnt	SBB	dst,fnt	x	-	-	-	x	x	x
x	x	SCAS/SCASB/ SCASW cdst	SCAS	cdst	x	-	-	-	x	x	x
x	x	SHR dst,cnt	SHR	dst,cnt	x	-	-	-	x	x	?
-	1	STC	STC		-	-	-	-	-	-	-
-	-	STD	STD		-	1	-	-	-	-	-
-	-	STI	STI		-	-	1	-	-	-	-
-	-	STOS/STOSB/ STOSW cdst	STOS	cdst	-	-	-	-	-	-	-
x	x	SUB dst,fnt	SUB	dst,fnt	x	-	-	-	x	x	x
x	0	TEST dst,fnt	TEST	dst,fnt	0	-	-	-	x	x	?
-	-	WAIT	WAIT		-	-	-	-	-	-	-
-	-	XCHG dst,fnt	XCHG	dst,fnt	-	-	-	-	-	-	-
-	-	XLAT tfnt	XLAT	tfnt	-	-	-	-	-	-	-
x	0	XOR dst,fnt	XOR	dst,fnt	0	-	-	-	x	x	?

4.3. - INSTRUCCIONES ESPECIFICAS DEL 286, 386 y 486 EN MODO REAL.

4.3.1. - DIFERENCIAS EN EL COMPORTAMIENTO GLOBAL RESPECTO AL 8086.

- Excepciones de división:

Las excepciones INT 0, debidas a una división por cero o a un cociente excesivamente grande, provocan que en la pila se almacene el valor de CS:IP para la siguiente instrucción en el 8086. En el 286 y superiores se almacena el CS:IP de la propia instrucción que causa la excepción.

- Códigos de operación indefinidos.

En el 286 y superiores se produce una excepción 6 (INT 6) o, si es una instrucción con sentido para estos procesadores, se ejecuta. El 8086 se estrella.

- Valor de PUSH SP.

El valor que introduce en la pila en el 286 y superiores es el de SP antes del PUSH; en el 8086 es el de SP después del PUSH (dos unidades menos).

- Desplazamientos y rotaciones.

El valor de desplazamiento en las operaciones de manipulación de bits del 8086 es una constante de 8 bits (indicada en CL); en el 286 y superiores se toma módulo 32 (sólo se consideran los 5 bits menos significativos).

- Prefijos redundantes.

Las instrucciones tienen una longitud ilimitada en el 8086; en el 286 y superiores no pueden exceder de 15 bytes. Por tanto, los prefijos redundantes pueden producir excepciones de código de operación no válido.

- Accesos al límite del segmento.

Un acceso de 16 bits en el offset 0FFFFh en el 8086 provoca un acceso a los bytes ubicados en las posiciones 0FFFFh y 0 (se da la vuelta alrededor del segmento). En el 286 y superiores, se produce una excepción de violación de límites. En el 386 y superiores se produce también en accesos de 32 bits en las posiciones 0FFFDh a la 0FFFFh. Esto se cumple tanto para accesos a datos en memoria como a instrucciones del programa en esos puntos críticos.

- LOCK.

Esta instrucción no está limitada de ninguna manera en el 8086 y en el 286. En el 386 y superiores su uso está restringido a determinadas instrucciones.

- Ejecución paso a paso.

La prioridad de la excepción paso a paso en el 286 y superiores es más alta que la de una interrupción externa; por tanto, las interrupciones externas no pueden ser traceadas.

- Registro de FLAGS.

Difiere algo en los bits 12 al 15 en todos los procesadores; el 386 dispone además de un registro de flags de 32 bits.

- Interrupción NMI.

Desde el 286 y superiores, una NMI no puede interrumpir una rutina de tratamiento NMI.

- Error del coprocesador.

En el 286 y superiores se utiliza el vector 16; en el 8086 cualquier vector.

- Prefijos de las instrucciones del coprocesador.

Al producirse una excepción de error de coprocesador, en el 8086 se almacena un CS:IP que no incluye prefijos -si los había-, al contrario que en el 286 y superiores.

- Límite del primer megabyte.

En el 8086 la memoria es circular; al final del primer megabyte se vuelve a comenzar por las posiciones más bajas de la memoria. En el 286 y superiores, se accede a la memoria extendida (un artificio hardware en los PC lo impide al forzar A20 a estado bajo, pero puede ser solventado).

- Instrucciones de cadena repetitivas.

El CS:IP grabado en el 8086 no incluye el prefijo, si existe; en el 286 y superiores sí.

4.3.2. - INSTRUCCIONES ESPECIFICAS DEL 286.

A continuación se describen las instrucciones adicionales que incorporan los 286 en modo real, que también pueden ser consideradas cuando trabajamos con los microprocesadores compatibles V20 y V30, así como con los procesadores superiores al 286. Las instrucciones del modo protegido se dirigen especialmente a la multiprogramación y el tiempo compartido, siendo específicas de la conmutación de procesos y tratamiento de la memoria virtual y no pueden emplearse directamente bajo DOS.

- BOUND r16, mem16: Comprueba si el registro de 16 bits indicado como primer operando está dentro de los límites de una matriz. Los límites de la matriz los definen dos palabras consecutivas en la memoria apuntadas por mem16. Si está fuera de los límites, se produce una interrupción 5 en la que el IP apilado queda apuntando a la instrucción BOUND (¡no se incrementa!).
- ENTER crea una estructura de pila para un procedimiento de alto nivel.
- Las instrucciones PUSH permiten meter valores inmediatos a la pila: es válido hacer PUSH 40h.
- IMUL puede multiplicar cualquier registro de 16 bits por una constante inmediata, devolviendo un resultado palabra (CF=1 si no cabe en 16 bits); por ejemplo, es válido

IMUL CX,25. También se admiten tres operandos: IMUL r1, r2, imm. En este caso, se multiplica r2 por el valor inmediato (8/16 bits) y el resultado se almacena en r1. Tanto r1 como r2 han de ser de 16 bits.

- LEAVE abandona los procedimientos de alto nivel (equivale a MOV SP,BP / POP BP).

- PUSH/POPA: Introduce en la pila y en este orden los registros AX, CX, DX, BX, SP, BP, SI y DI -o los saca en orden inverso-. Ideal en el manejo de interrupciones y muy usada en las BIOS de 286 y 386.

- OUTS (salida de cadenas) e INS (entrada de cadenas) repetitivas (equivalente a MOVS y LODS).

- RCR/RCL, ROR/ROL, SAL/SAR y SHL/SHR admiten una constante de rotación distinta de 1.

4.3.3. - INSTRUCCIONES PROPIAS DEL 386 Y 486.

- Además de todas las posibilidades adicionales del 286, el 386 y el 486 permiten utilizar cualquier registro de 32 bits de propósito general en todos los modos de funcionamiento, incluido el modo real, tales como EAX, EBX, ECX, EDX, ESI, EDI, EBP. Sin embargo no deben intentarse direccionamientos por encima de los 64K. En otras palabras, se pueden utilizar para acelerar las operaciones pero no para acceder a más memoria. Por ejemplo, si $EBX > 0FFFFh$, la instrucción `MOV AX,[EBX]` tendría un resultado impredecible. Además, estos procesadores cuentan con dos segmentos más: además de DS, ES, CS y SS se pueden emplear también FS y GS. Aviso: parece ser que en algunos 386 fallan ocasionalmente las instrucciones de multiplicar de 32 bits.

Nota: No es del todo cierto que el 386 y el 486 no permitan acceder a más de 64 Kb en modo real: en la sección 4.3.6 hay un ejemplo de ello.

- Los modos de direccionamiento aumentan notablemente su flexibilidad en el 386 y superiores. Con los registros de 16 bits sólo están disponibles los modos tradicionales. En cambio, con los de 32 se puede utilizar en el direccionamiento indirecto cualquier registro: es válida, por ejemplo, una instrucción del tipo `MOV AX,[ECX]` o `MOV EDX,[EAX]`. Los desplazamientos en el direccionamiento indexado con registros de 32 bits pueden ser de 8 y también de 32 bits. Cuando dos registros deben sumarse para calcular la dirección efectiva, el segundo puede estar multiplicado por 2, 4 u 8; por ejemplo, es válida la instrucción `MOV AL,[EDX+EAX*8]`. Por supuesto, bajo DOS hay que asegurarse siempre que el resultado de todas las operaciones que determinan la dirección efectiva no excede de $0FFFFh$ ($0FFFEh$ si se accede a palabras y $0FFFCh$ en

accesos a dobles palabras en memoria).

- BOUND r32, mem32: Se admiten ahora operandos de 32 bits.
- BSF/BSR: Exploración de bits hacia adelante y atrás, respectivamente. La sintaxis es:

```
BSF reg, reg      ó      BSF reg, [memoria]
BSR reg, reg      ó      BSR reg, [memoria]
```

Donde *reg* puede ser de 16 ó 32 bits. Se comienza a explorar por el bit 0 (BSF) o por el más significativo (BSR) del segundo operando: si no aparece ningún bit activo (a 1) el indicador ZF se activa; en caso contrario se almacena en el primer operando la posición relativa de ese bit:

```
MOV     AX, 8
BSF     BX, AX
JZ      ax_es_0      ; no se saltará,
```

además BX = 3

- BT/BTC/BTR/BTS: Operaciones sobre bits: comprobación, comprobación y complementación, comprobación y puesta a 0, comprobación y puesta a 1. Sintaxis (ejemplo sobre BT):

```
BT reg, reg      ó BT reg, imm8
```

Donde *reg* puede ser de 16 ó 32 bits, el operando inmediato es necesariamente de 8. Estas instrucciones copian el número de bit del primer operando que indique el segundo operando (entre 0 y 31) en el acarreo. A continuación no le hacen nada a ese bit (BT), lo complementan (BTC), lo borran (BTR) o lo activan (BTS). Ejemplo:

```
MOV     AX, 16
BTC     AX, 4      ; resultado: CF =
```

1 y AX = 0

- CDQ: Similar a CWD, extiende el signo de EAX a EDX:EAX.
- CMPSD: Similar a CMPSW pero empleando ESI, EDI, ECX y comparando datos de 32 bits. Se puede emplear bajo DOS siempre que ESI y EDI (utilizando REP también ECX) no excedan de 0FFFFh.
- CWDE: Extiende el signo de AX a EAX.

- **IMUL:** Ahora se admite un direccionamiento a memoria en el 2º operando:
IMUL CX,[dato]
- **INSD:** Similar a INSW pero empleando ESI, EDI, ECX y leyendo datos de 32 bits. Se puede emplear bajo DOS siempre que ESI y EDI (utilizando REP también ECX) no excedan de 0FFFFh.
- **Jcc:** Los saltos condicionales ahora pueden ser de ¡32 bits!. Mucho cuidado con la directiva .386 en los programas en que se desee mantener la compatibilidad con procesadores anteriores. JECXZ se utiliza en vez de JCXZ (mismo código de operación).
- **LODSD:** Similar a LODSW pero empleando ESI, EDI y ECX y cargando datos de 32 bits en EAX. Se puede emplear bajo DOS siempre que ESI y EDI (utilizando REP también ECX) no excedan de 0FFFFh.
- **LSS, LFS, LGS:** similar a LDS o LES pero con esos registros de segmento.
- **MOV CRx,reg / MOV DRx,reg y los recíprocos:** acceso a registros de control y depuración.
- **MOVSD:** Similar a MOVSW pero empleando ESI, EDI, ECX y moviendo datos de 32 bits. Se puede emplear bajo DOS para acelerar las transferencias siempre que ESI y EDI (utilizando REP también ECX) no excedan de 0FFFFh. Operando sobre la memoria de vídeo sólo se obtiene ventaja si la tarjeta es realmente de 32 bits.
- **MOVSX / MOVZX:** carga con extensión de signo o cero. Toma el segundo operando, le extiende adecuadamente el signo (o le pone a cero la parte alta) hasta que sea tan grande como el primer operando y luego lo carga en el primer operando. Si el primer operando es de 16 bits, el segundo sólo puede ser de 8; si el primero es de 32 bits el segundo puede ser de 8 ó 16. El primer operando debe ser un registro, el segundo puede ser un registro u operando en memoria (nunca inmediato):

```

MOV     EAX,0FFFFFFFFh
MOV     AX,7FFFh           ; resultado: EAX =
0FFFF7FFFh
MOV     EAX,AX             ; resultado: EAX =
000007FFFh

```

- **OUTSD:** Similar a OUTSW pero empleando ESI, EDI, ECX y enviando datos de 32 bits. Se puede emplear bajo DOS siempre que ESI y EDI (usando REP también ECX)

no rebasen 0FFFFh.

- Prefijos FS: y GS: en los accesos a memoria, referenciando a esos segmentos.
- PUSHAD / POPAD: Similares a PUSHA y POPA pero con los registro de 32 bits. La instrucción POPAD falla en la mayoría de los 386, incluidos los de AMD. Para solventar el fallo (que consiste en que EAX no se restaura correctamente) basta colocar un NOP inmediatamente detrás de POPAD.
- PUSHFD/POPFD introducen y sacan de la pila los flags de 32 bits.
- SCASD: Similar a SCASW pero empleando ESI, EDI, ECX y buscando datos de 32 bits. Se puede emplear bajo DOS siempre que ESI y EDI (usando REP también ECX) no rebasen 0FFFFh.
- SETcc reg8 ó mem8: Si se cumple la condición *cc*, se pone a 1 el byte de memoria o registro de 8 bits indicado (si no, a 0). Por ejemplo, con el acarreo activo, SETC AL pone a 1 el registro AL.
- SHLD / SHRD: Desplazamiento de doble precisión a la izquierda/derecha. La sintaxis es (ejemplo sobre SHLD):

```
reg16, C          SHLD regmem16, reg16, imm8    ó   SHLD regmem16,
reg32, CL        SHLD regmem32, reg32, imm8    ó   SHLD regmem32,
```

Donde *regmem* es un registro u operando en memoria, indistintamente, del tamaño indicado. En el caso de SHLD, se desplaza el primer operando a la izquierda tanto como indique el tercer operando (contador). Una vez desplazado, los bits menos significativos se rellenan con los más significativos del segundo operando, que no resulta alterado. SHRD es análogo pero al revés.

```
MOV     AX, 1234h
MOV     BX, 5678h
SHLD   AX, BX, 4      ; resultado: AX=2345h,
BX=5678h
```

- STOSD: Similar a STOSW pero empleando ESI, EDI, ECX y almacenando EAX. Se puede emplear bajo DOS siempre que ESI y EDI (utilizando REP también ECX) no excedan de 0FFFFh.

4.3.4. - DETECCIÓN DE UN SISTEMA AT O SUPERIOR.

Hay casos en los que es necesario determinar si una máquina es AT o superior: no ya de cara a emplear instrucciones propias del 286 en modo real (también disponibles en los V20/V30 y 80188/80186) sino debido a la necesidad de acceder a ciertos chips (por ejemplo, el segundo controlador de interrupciones) que de antemano se sabe que sólo equipan máquinas AT o superiores. Es importante por tanto determinar la presencia de un AT, de cara a evitar ciertas instrucciones que podrían bloquear un PC o XT. No se debe en estos casos comprobar los bytes de la ROM que identifican el equipo: a veces no son correctos y, además, la evolución futura que tengan es impredecible. Lo ideal es verificar directamente si está instalado un 286 o superior.

```

PUSHF
POP     AX           ; AX = flags
AND     AH,0Fh      ; borrar nibble más significativo
PUSH    AX
POPF                    ; intentar poner a 0 los 4 bits más
significativos de los flags
PUSHF
POP     AX
AND     AH,0F0h     ; seguirán valiendo 1 excepto en un 80286 o
superior
CMP     AH,0F0h
JE      no_es_AT
JMP     si_es_AT    ; es 286 o superior

```

4.3.5. - EVALUACIÓN EXACTA DEL MICROPROCESADOR INSTALADO.

Sobra decir que las instrucciones avanzadas deben ser utilizadas con la previa comprobación del tipo de procesador, aunque sólo sea para decir al usuario que se compre una máquina más potente antes de abortar la ejecución del programa. Para averiguar el procesador de un ordenador puede emplearse el siguiente programa de utilidad, basado en el procedimiento **procesador?** que devuelve en AX un código numérico entro 0 y 8 distinguiendo entre los 9 procesadores más difíciles de identificar de los ordenadores compatibles. Nota: el 486 no tiene que tener coprocesador necesariamente (el 486sx carece de él).

Algunas versiones de procesador 486 y todos los procesadores posteriores soportan la instrucción CUID que permite identificar la CPU. Basta comprobar un bit del registro de estado para saber si está soportada y, en ese caso, poder emplear dicha instrucción. De este modo, resulta trivial detectar el Pentium o cualquier procesador posterior que aparezca. Esta instrucción está documentada, por ejemplo en alguno de los ficheros que acompañan al Interrupt List. Para los propósitos de este libro no es preciso en general detectar más allá del 386.

Es normal que el lector recién iniciado en el ensamblador no entienda absolutamente nada de este programa, ya que hasta los siguientes capítulos no será explicada la sintaxis del lenguaje. En ese caso, puede saltarse este ejemplo y continuar en el capítulo siguiente, máxime si no tiene previsto trabajar con otras instrucciones que no sean las del 8086. Por último, recordar que las instrucciones específicas del 286 en modo real también están disponibles en los V20/V30 de NEC y la serie 80188/80186 de Intel.

; *****

```

; *
; * CPU v2.2 (c) Septiembre 1992 CiriSOFT *
; * (c) Grupo Universitario de Informática - Valladolid *
; *
; * Este programa determina el tipo de microprocesador del equipo *
; * y devuelve un código ERRORLEVEL indicándolo: *
; *
; * 0-8088, 1-8086, 2-NEC V20, 3-NEC V30, *
; * 4-80188, 5-80186, 6-286, 7-386, 8-486 *
; *
; * Aviso: Utilizar TASM 2.0 o compatible exclusivamente. *
; *
; *****

```

```

cpu          SEGMENT
             ASSUME CS:cpu, DS:cpu

             .386

inicio:      ORG    100h

             LEA   DX,texto_ini      ; texto de saludo
             MOV   AH,9
             INT   21h                ; imprimirlo
             CALL  procesador?       ; tipo de procesador en AX
             PUSH  AX                 ; guardarlo para el final
             LEA   BX,cpus_indice-2  ; tabla de nombres-2
             MOV   CX,0FFFFh         ; número de iteración-1
otro_proc:   INC   CX
             ADD   BX,2
             MOV   DX,[BX]           ; nombre del primer procesador
             CALL  print
             CMP   CX,AX              ; ¿procesador del equipo?
             JNE   no_es_este
             LEA   DX,apuntador_txt  ; sí lo es: indicarlo
             CALL  print
no_es_este:  LEA   DX,separador_txt
             CALL  print
             CMP   CX,7                ; número de CPUs tratadas-1
             JBE   otro_proc
             LEA   DX,texto_fin      ; últimos caracteres
             CALL  print
             MOV   AH,4Ch             ; retornar código errorlevel AL
             INT   21h                ; fin de programa

procesador?  PROC                ; devolver el tipo de microprocesador en AX
             PUSHF
             PUSH  DS
             PUSH  ES
             PUSH  CX
             PUSH  DX
             PUSH  DI
             PUSH  SI
             MOV   AX,CS
             MOV   DS,AX              ; durante la rutina se guardará
             MOV   ES,AX              ; el tipo de procesador en DL:
             MOV   DL,6                ; supuesto un 286 (DL=6) ...
             PUSHF
             POP   AX                  ; AX = flags

```

```

AND    AX,0FFFh      ; borrar nibble más significativo
PUSH   AX
POPF                   ; intentar poner a 0 los 4 bits más
PUSHF                   ; significativos de los flags
POP    AX
AND    AX,0F000h     ; seguirán valiendo 1 excepto en
CMP    AX,0F000h     ; un 80286 o superior
JE     ni286ni_super
PUSHF                   ; es 286 o superior
POP    AX
OR     AX,7000h      ; intentar activar bit 12, 13 ó 14
PUSH   AX
POPF
PUSHF
POP    AX
AND    AX,7000h      ; 286 pone bits 12, 13 y 14 a cero
JZ     cpu_hallada    ; es un 286 (DL=6)
INC    DL             ; es un 386 (DL=7) ... de momento
PUSH   DX
CLI                                         ; momento crítico
MOV    EDX,ESP        ; preservar ESP en EDX
AND    ESP,0FFFFh     ; borrar parte alta de ESP
AND    ESP,0FFFCh     ; forzar ESP a múltiplo de 4
PUSHFD
POP    EAX            ; recuperar flags en EAX
MOV    ECX,EAX
XOR    EAX,40000h     ; conmutar bit 18
PUSH   EAX
POPF                    ; intentar cambiar este bit
PUSHFD
POP    EAX            ; ECX conserva el bit inicial
XOR    EAX,ECX        ; bit 18 de EAX a 1 si cambió
SHR    EAX,12h        ; mover bit 18 a bit 0
AND    EAX,1          ; dejar sólo ese bit
PUSH   ECX
POPF                    ; restaurar bit 18 de los flags
MOV    ESP,EDX        ; restaurar ESP
STI                                         ; permitir interrupciones de nuevo
POP    DX              ; recuperar tipo de CPU en DL
CMP    AX,0
JE     cpu_hallada    ; es 386: DL=7 (bit 18 no cambió)
INC    DL              ; es 486: DL=8 (bit 18 cambió)
JMP    cpu_hallada
ni286ni_super: MOV    DL,4          ; supuesto un 80188 ...
MOV    AX,0FFFFh
MOV    CL,33
SHL    AX,CL          ; (80188/80186 toman CL mod 32)
JNZ    tipo_bus_proc ; ... lo es, calcular bus (188/186)
MOV    DL,2           ; no lo es, supuesto un V20 ...
MOV    CX,0FFFFh
STI
DB     0F3h,26h,0ACh ; opcode de REPZ LODSB ES:
JCXZ   tipo_bus_proc ; ... lo es, calcular bus (V20/V30)
XOR    DL,DL          ; ya sólo puede ser un 8088/8086
tipo_bus_proc: STD                    ; transferencias hacia arriba
LEA    DI,tipo_bus_dest
MOV    AL,BYTE PTR DS:tipo_bus_byte ; opcode de STI
MOV    CX,3
CLI
REP    STOSB          ; transferir tres bytes
CLD

```

```

NOP                ; el INC CX (1 byte) será machacado
NOP                ; con STOSB pero aún se ejecutará
NOP                ; en un 8086/80186/V30 (y no en un
tipo_bus_byte: STI  ; 8088/80188/V20) porque está en la
tipo_bus_dest: STI  ; cola de lectura adelantada.
JCXZ  cpu_hallada  ; el bus ya era supuesto de 8 bits
INC   DL           ; resulta que es de 16
cpu_hallada: MOV   AL,DL
XOR   AH,AH
POP   SI
POP   DI
POP   DX
POP   CX
POP   ES
POP   DS
POPF
RET   ; AX = CPU: 0/1-8088/86, 2/3-NEC V20/V30
procesador? ENDP  ; 4/5-80188/186, 6-286, 7-386, 8-486

print PROC
PUSH  AX
PUSH  BX
PUSH  CX
MOV   AH,9
INT   21h
POP   CX
POP   BX
POP   AX
RET
print ENDP

cpus_indice DW   i88,i86,v20,v30,i188,i186,i286,i386,i486
i88        DB   "Intel 8088 $"
i86        DB   "Intel 8086 $"
v20        DB   " NEC V20  $"
v30        DB   " NEC V30  $"
i188       DB   "Intel 80188$"
i186       DB   "Intel 80186$"
i286       DB   "Intel 80286$"
i386       DB   "Intel 80386$"
i486       DB   "Intel 80486$"

apuntador_txt DB   " <---$"

texto_ini LABEL BYTE
DB   13,10,"CPU Test v2.2  "
DB   "(c) Septiembre 1992 Ciriaco García de Celis."
DB   13,10," El microprocesador de este "
DB   "equipo es compatible:",10
separador_txt DB  13,10,9,9,9,"$"
texto_fin DB   13,10,"$"

cpu ENDS
END inicio

```

4.3.6. - MODO PLANO (FLAT) DEL 386 Y SUPERIORES.

Como ya se comentó, no es estrictamente cierto que no se pueda rebasar el límite de 64 Kb en los segmentos en modo real. El problema es que al encender el ordenador, el 386 tiene definidos por defecto dichos límites de 64 Kb. Sin embargo, se puede pasar un momento a modo protegido, ampliar el límite y volver a modo real. Entonces se consigue el llamado modo *flat* o plano. No solo es factible de este modo saltar la restricción de 64 Kb, sino que además se puede acceder directamente, desde el modo real, a toda la memoria por encima del primer megabyte.

El problema es que pasar a modo protegido no es sencillo cuando la máquina ya está en modo protegido emulando al modo real (el conocido como modo virtual 86). Por tanto, el siguiente programa de ejemplo no funciona si está cargado un controlador de memoria expandida (EMM386, QEMM) o dentro de Windows 3.x. Arrancando sin controlador de memoria (excepto HIMEM) no habrá problema alguno. El programa de ejemplo se limita a llenar la pantalla de texto (empleando ahora la dirección absoluta 0B8000h a través de EBX) de letras 'A'.

Otra restricción de este programa de ejemplo es que no activa la línea A20 de direcciones; dicho de otro modo, el bit 21º (de los 32 bits de la dirección de memoria) suele estar forzado a 0 por defecto al arrancar. Para acceder a la memoria de vídeo esto no es problema, pero por encima del primer megabyte podría haber problemas según a qué dirección se pretenda acceder. De todos modos, sería relativamente sencillo habilitar la línea A20 directamente o a través de una función del controlador XMS.

Naturalmente, se sale de los objetivos de este libro describir el modo protegido o explicar los pasos que realiza esta rutina de demostración. Consúltese al efecto la [bibliografía](#) recomendada del apéndice.

```

; +-----+
; | Rutina para activar el modo flat del 386 y superiores (acceso |
; | a 4 Gb en modo real). |
; | |
; | TASM flat386 /m5 |
; | TLINK flat386 /t /32 |
; +-----+

                .386p                ; sólo para 386 o superior

segmento        SEGMENT USE16
                ASSUME CS:segmento, DS:segmento

                ORG    100h

prueba:
                CALL   flat386        ; activar modo flat
                XOR    AX,AX
                MOV    DS,AX
                MOV    EBX,0B8000h    ; dirección de vídeo absoluta
                MOV    CX,2000

llena_pant:    MOV    BYTE PTR [EBX], 'A'
                INC    EBX
                MOV    BYTE PTR [EBX], 15
                INC    EBX
                LOOP   llena_pant
                INT    20h            ; fin de programa

; ----- Esta rutina pasa momentáneamente a modo protegido de
;          manera directa (necesita la CPU en modo real). No se

```

```

; activa la línea A20 (necesario hacerlo directamente
; o a través de algún servicio XMS antes de acceder a
; las áreas de memoria extendida afectadas).

flat386 PROC
PUSH DS
PUSH ES
PUSH EAX
PUSH BX
PUSH CX
MOV CX,SS
XOR EAX,EAX
MOV AX,CS
SHL EAX,4 ; dirección lineal de segmento

CS
ADD EAX,OFFSET gdt ; desplazamiento de GDT
MOV CS:[gd2],EAX ; guardar dirección lineal de

GDT
CLI
LGDT CS:[gdtr] ; cargar tabla global de

descriptores
MOV EAX,CRO
OR AL,1 ; bit de modo protegido
MOV CRO,EAX ; pasar a modo protegido
JMP SHORT $+2 ; borrar cola de prebúsqueda
MOV BX,gcodl ; índice de descriptor en BX
MOV DS,BX ; cargar registro de segmento

DS
MOV ES,BX ; ES
MOV SS,BX ; SS
MOV FS,BX ; FS
MOV GS,BX ; GS
AND AL,11111110b
MOV CRO,EAX ; volver a modo real
JMP SHORT $+2 ; borrar cola de prebúsqueda
MOV SS,CX
STI
POP CX
POP BX
POP EAX
POP ES
POP DS
RET

gdtr LABEL QWORD ; datos para cargar en GDTR
gd1 DW gdt1-1
gd2 DD ?

gdt DB 0,0,0,0,0,0,0,0 ; GDT
gcod DB 0ffh,0ffh,0,0,0,9fh,0cfh,0
gcodl EQU $-OFFSET gdt
gdat DB 0ffh,0ffh,0,0,0,93h,0cfh,0
gdt1 EQU $-OFFSET gdt

flat386 ENDP

segmento ENDS
END prueba

```