JOC D'INSTRUCCIONS 80x86

DESCRIPCIÓ COMPLETA DE LES INSTRUCCIONS.

Nota: En l'efecte de les instruccions sobre el registre de estat s'utilitzarà la següent notació:

- bit no modificat
? desconegut o indefinit
X modificat segons el resultat de l'operació
1 posat sempre a 1
0 posat sempre a 0

INSTRUCCIONS DE CÀRREGA DE REGISTRES I ADRECES.

MOV (transferència)

Sintaxi: `MOV destí, origen`

Indicadors (flags):

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

Transfereix dades de longitud byte o paraula del operant origen al operant destí. Poden ser operant origen i operant destí qualsevol registre o posició de memòria adreçada de les formes ja vistes, amb la única condició de que origen i destí tinguin la mateixa dimensió. Existeixen certes limitacions, com que els registres de segment no admeten l'adreçat immediat: es incorrecte MOV DS,4000h; però no ho son per exemple MOV DS,AX o MOV DS,VARIABLE. No es possible, així mateix, utilitzar CS com destí (es incorrecte fer MOV CS,AX encara que ho pugui admetre algun assemblador). Al fer MOV cap a un registre de segment, les interrupcions queden inhibides fins després de executar-se la següent instrucció (8086/88 de 1983 i processadors posteriors).

Exemples:

```
mov     ds,ax
mov     bx,es:[si]
mov     si,offset dada
```

En el últim exemple, no es posa a SI el valor de la variable dada sinó la seva posició de memòria o desplaçament respecte al segment de dades. En altres paraules, SI es un apuntador a dada però no es dada. més endavant en la descripció de l'assemblador veurem com es declaren les variables.

XCHG (intercanviar)

Sintaxi: XCHG destí, origen

Indicadors (flags):

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

Intercanvia el contingut dels operants origen i destí. No poden utilitzar-se registres de segments com operants.

Exemple:

```
xchg    bl,ch
xchg    mem_pal,bx
```

XLAT (traducció)

Sintaxi: XLAT taula

Indicadors (flags):

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

S'utilitza per traduir un byte del registre AL a un byte prenent de la taula de traducció. Les dades es prenen des d'una adreça de la taula corresponent a BX + AL, on bx es un apuntador al començament de la taula i AL es un índex. Indicar taula al costat de **xlat** es només una redundància opcional.

Exemple:

```
mov     bx,offset taula
mov     al,4
xlat
```

LEA (carrega adreça efectiva)

Sintaxi: `LEA destí, origen`

Indicadors (flags):

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

Transfereix el desplaçament de operant font a operant destí. Altres instruccions poden a continuació utilitzar el registre com desplaçament per accedir a les dades que constitueixen l'objectiu. Operant destí no pot ser un registre de segment. En general, aquesta instrucció es equivalent a MOV destí, OFFSET font i de fet els bons assembladors (TASM) la codifiquen com MOV per economitzar un byte de memòria. Però LEA es en alguns casos més potent que MOV al permetre indicar registres de índex i desplaçament per calcular l'offset:

```
lea     dx,dades[si]
```

En el exemple de dalt, el valor dipositat a DX es l'offset de la etiqueta dades més el registre SI. Aquesta sola instrucció es equivalent a aquestes dues:

```
mov     dx,offset dades
add     dx,si
```

LDS (carrega un apuntador utilitzant DS)

Sintaxi: `LDS destí, origen`

Indicadors (flags):

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

Trasllada un apuntador de 32 bits (adreça complerta de memòria formada per segment i desplaçament), al destí indicat i a DS. A partir de l'adreça indicada per operant origen, el processador pren 4 bytes de la memòria: amb els dos primers forma una paraula que diposita a destí i, amb els altres dos, altra a DS.

Exemple:

```
punt      dd     12345678h
lds  si,punt
```

Com resultat d'aquesta instrucció, a DS:SI es fa referència a la posició de memòria 1234h:5678h; 'dd' serveix per definir una variable llarga de 4 bytes (denominada punt en el exemple) i serà explicat amb la descripció de l'assemblador.

LES (carrega un apuntador utilitzant ES)

Sintaxi: `LES destí, origen`

Aquesta instrucció es anàloga a LDS, però utilitzant ES en lloc de DS.

LAHF (carrega AH amb els indicadors)

Sintaxi: LAHF

Indicadors (flags):

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

Carrega els bits 7, 6, 4, 2 y 0 del registre AH amb el contingut dels indicadors SF, ZF, AF, PF Y CF respectivament. El contingut de la resta de bits queda sense definir.

SAHF (copia AH als indicadors)

Sintaxi: SAHF

Indicadors (flags):

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

Transfereix el contingut dels bits 7, 6, 4, 2 y 0 als indicadors SF, ZF, AF, PF i CF respectivament.

INSTRUCCIONS DE MANIPULACIÓ DEL REGISTRE D'ESTAT.

CLC (baixa l'indicador de carry)

Sintaxi: CLC

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
--	--	--	--	--	--	--	--	**0**

Esborra l'indicador de carry (CF) sense afectar a cap altra.

CLD (baixa l'indicador de direcció)

Sintaxi: `CLD`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
--	**0**	--	--	--	--	--	--	--

Posa a 0 l'indicador de direcció DF, per tant els registres SI i/o DI son autoincrementats en les operacions de cadenes, sense afectar a la resta dels indicadors. Es NECESARI posar-lo abans de les instruccions de manejament de cadenes si no es coneix amb seguretat el valor de DF. Veure STD.

CLI (baixa l'indicador d'interrupció)

Sintaxi: `CLI`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
--	--	**0**	--	--	--	--	--	--

Esborra l'indicador d'activació d'interrupcions IF, el que desactiva les interrupcions enmascarables. Es molt convenient fer això abans de modificar la parella SS:SP en els 8086/88 anteriors a 1983 (veure comentari a la instrucció MOV), o abans de canviar un vector d'interrupció sense l'ajut del DOS. Generalment les interrupcions només s'inhibeixen uns breus instants en moments crítics. Veure també STI.

CMC (complementa l'indicador de carry)

Sintaxi: `CMC`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
--	--	--	--	--	--	--	--	X

Complementa l'indicador de carry CF invertint el seu estat.

STC (posa a 1 l'indicador de carry)

Sintaxi: `STC`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
--	--	--	--	--	--	--	--	**1**

Posa a 1 l'indicador de carry CF sense afectar a cap altra indicador.

STD (posa a 1 l'indicador de direcció)

Sintaxi: `STD`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
--	**1**	--	--	--	--	--	--	--

Posa a 1 l'indicador de direcció DF, per tant els registres SI i/o DI s'autodecrementen a les operacions de cadenes, sense afectar a la resta dels indicadors. Es NECESSARI posar-lo abans de les instruccions de manejament de cadenes si no es coneix seguretat l'estat de DF. Veure també CLD.

STI (posa a 1 l'indicador d'interrupció)

Sintaxi: `STI`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
--	--	**1**	--	--	--	--	--	--

Posa a 1 la bandera de desactivar les interrupcions IF i activa les interrupcions enmascarables. Una interrupció pendent no es reconeguda, no obstant, fins després d'executar la instrucció que segueix a STI. Veure també CLI.

INSTRUCCIONS DE MANEGAMENT DE LA PILA.

POP (extreure de la pila)

Sintaxi: `POP destí`

Indicadors (flags):

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

Transfereix l'element paraula que es troba a la part més alta de la pila (apuntat per SP) a operant destí que ha de ser tipus paraula, i incrementa en dos el registre SP. La instrucció POP CS, poc útil, no funciona correctament en els 286 i superiors.

Exemples:

```
pop     ax
pop     pepe
```

PUSH (introdueix a la pila)

Sintaxi: `PUSH origen`

Indicadors (flags):

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

Decrementa l'apuntador de la pila (SP) en 2 i llavors transfereix la paraula especificada en el operant origen al top de la pila. El registre CS aquí sí que es pot especificar com origen, al contrari del que afirmen algunes publicacions.

Exemple:

```
push    cs
```

POPF (extreu els indicadors de la pila)

Sintaxi: `POPF`

Indicadors (flags):

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

Trasllada al registre dels indicadors la paraula emmagatzemada al top de la pila; a continuació l'apuntador de la pila SP s'incrementa en dos.

PUSHF (introdueix els indicadors a la pila)

Sintaxi: PUSHF

Indicadors (flags):

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

Decrementa en dos l'apuntador de la pila i trasllada al top de la pila el contingut dels indicadors.

INSTRUCCIONS DE TRANSFERENCIA DE CONTROL.

Incondicional

CALL (crida a subrutina)

Sintaxi: `CALL destí`

Indicadors (flags):

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

Transfereix el control del programa a un procediment, salvant prèviament a la pila l'adreça de la instrucció següent, per poder tornar-hi un cop executat el procediment. El procediment pot estar en el mateix segment (tipus NEAR) o en un altre segment (tipus FAR). Així mateix la crida pot ser directa a una etiqueta (especificant el tipus de crida NEAR - per defecte - o FAR) o indirecte, indicant l'adreça on es troba l'apuntador. Segons la crida sigui pròxima o llunyana, s'emmagatzema a la pila una adreça de retorn de 16 bits o dos paraules de 16 bits indicant en aquest últim cas tant l'offset (IP) com el segmento (CS) a on tornar.

Exemples:

```
        call    proc1
        dir     dd      0f000e987h
```

```
call     dword ptr dir
```

En el segon exemple, la variable dir emmagatzema l'adreça a on saltar. D'aquesta última manera –coneixent la seva adreça- pot cridar-se també a un vector d'interrupció, guardant prèviament els flags a la pila (PUSHF), perquè la rutina d'interrupció retornarà (amb IRET en lloc de amb RETF) treuen els flags de la pila.

JMP (salt)

Sintaxi: `JMP adreça o JMP SHORT adreça`

Indicadors (flags):

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

Transfereix el control incondicionalment a l'adreça indicada a operant. La bifurcació pot ser també directa o indirecta com hem vist anteriorment, però a més a més pot ser curta (tipus SHORT) amb un desplaçament compres entre -128 y 127; o llarg, amb un desplaçament de dos bytes amb signe. Si es fa un JMP SHORT i no arriba el salt (perquè esta massa allunyada de aquesta etiqueta) l'assemblador donarà error. Els bons assembladores (com TASM) quan donen dos passades col·loquen allà on es possible un salt curt, per economitzar memòria, sense que el programador tingui d'ocupar-se de posar short. Si el salt de dos bytes, que permet desplaçaments de 64 Kb a la memòria segueix següent insuficient, es pot indicar amb far que es llarg (salt a un altra segment).

Exemples:

```
jmp     etiqueta
jmp     far ptr etiqueta
```

RET / RETF (retorn de subrutina)

Sintaxi: `RET [valor] o RETF [valor]`

Indicadors (flags):

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

Retorna d'un procediment agafant de la pila posició de la següent adreça. S'extraurà el registre de segment i el desplaçament en un procediment de tipus FAR (dos paraules) i solament el desplaçament en un procediment NEAR (una paraula). si aquesta instrucció es col·locada dins d'un bloc PROC-ENDP (com es veurà més endavant) l'assemblador

sap el tipus de retorn que te de fer, segons el procediment sia NEAR o FAR. En qualsevol cas, es pot forçar que el retorn sia de tipus FAR amb la instrucció RETF. Valor, si es indicat permet sumar una quantitat valor en bytes a SP abans de retornar, lo que es freqüent en el codi generat por els compiladors per retornar d'una funció amb paràmetres. També es pot retornar d'una interrupció amb RETF 2, per que retorni el registre d'estat sense restaurar-lo de la pila.

Condicional

Les següents instruccions son de transferència condicional de control a la instrucció que es troba a la posició IP + desplaçament (desplaçament compres entre -128 y +127) si es compleix la condició. Algunes condicions es poden denotar de varies maneres.

Tots els salts son curts i si no arriben s'ha de fer com sigui. En negreta es remarquen les condicions més emprades. On intervé SF es consideren amb signe els operants implicats en l'última comparació o operació aritmetica-lógica, i s'indiquen a la taula com '±' (-128 a +127 ó -32768 a +32767); en els demés casos, indicats com '+', es consideren sense signe (0 a 255 ó 0 a 65535):

JA/JNBE	Salta si major (above), si no menor o igual (not below or equal), si CF=0 y ZF=0.	+
JAE/JNB	Salta si major o igual (above or equal), si no menor (not below), si CF=0.	+
JB/JNAE/JC	Salta si menor , si no ni superior ni igual (not above or equal), si carry, si CF=1.	+
JBE/JNA	Salta si no menor o igual, si no mayor (not above), si CF=1 ó ZF=1.	+
JCXZ	Salta si CX=0.	
JE/JZ	Salta si igual (equal), si cero (zero), si ZF=1.	
JG/JNLE	Salta si major (greater), si no menor ni igual (not less or equal), si ZF=0 i SF=0.	±
JGE/JNL	Salta si major o igual (greater or equal), si no menor (not less), si SF=0.	±
JL/JNGE	Salta si menor (less), si no major ni igual (not greater or equal), si SF<>OF.	±
JLE/JNG	Salta si menor o igual (less or equal), si no major (not greater), si ZF=0 i SF<>OF.	±
JNC	Salta si no carry, si CF=0.	
JNE/JNZ	Salta si no igual, si no cero, si ZF=0.	
JNO	Salta si no desbordament, si OF=0.	
JNP/JPO	Salta si no paritat, si paritat senar, si PF=0.	
JNS	Salta si no signe, si positiu, si SF=0.	

JO Salta si desbordament, si OF=1.

JP/JPE Salta si paritat, si paritat parell, si PF=1.

JS Salta si signe, si SF=1.

Gestió de bucle

LOOP (bucle)

Sintaxi: `LOOP desplaçament`

Indicadors (flags):

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

Decrementa el registre comptador CX; si CX es zero, executa la següent instrucció, en cas contrari transfereix el control a l'adreça resultant de sumar a IP + desplaçament. El desplaçament te d'estar compres entre -128 i +127.

Exemple:

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

Amb les mateixes característiques que la instrucció anterior:

LOOPE/LOOPZ Bucle si igual, si zero. Z=1 y CX<>0

LOOPNE/LOOPNZ Bucle si no igual, si no zero. Z=0 y CX<>0

INTERRUPCIONS

INT (interrupció)

Sintaxi: `INT` n (0 <= n <= 255)

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
--	--	**0**	**0**	--	--	--	--	--

Dona valors inicials a un procediment d'interrupció d'un tipus indicat a la instrucció. A la pila s'introdueix al cridar a una interrupció l'adreça de retorn formada por els registres CS i IP i l'estat dels indicadors. INT 3 es un cas especial de INT, al assemblar, l'assemblador genera un sol byte en lloc dels dos habituals; aquesta interrupció s'utilitza per posar punts de ruptura en els programes. Veure també IRET.

Exemple:

```
        int     21h
```

INTO (interrupció per desbordament)

Sintaxi: `INTO`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
--	--	**0**	**0**	--	--	--	--	--

Genera una interrupció de tipus 4 (INT 4) si existeix desbordament (OF=1). Per el contrari es continua amb la instrucció següent.

IRET (retorn d'interrupció)

Sintaxi: `IRET`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**

Retorna el control a la adreça de retorn salvada a la pila per una interrupció prèvia i restaura els indicadors que també es van introduir a la pila. En total, es treuen les 3 paraules que foren col·locades a la pila quan es va produir la interrupció. Veure també INT.

INSTRUCCIONS D'ENTRADA SORTIDA (E/S).

IN (entrada)

Sintaxi: `IN acumulador, port`

Indicadors (flags):

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

Transfereix dades des del port indicat fins el registre AL o AX, depenent de la longitud byte o paraula respectivament. El port es pot especificar mitjançant una constant (0 a 255) o a través del valor contingut a DX (0 a 65535).

Exemple:

```
in      ax,0fh
in      al,dx
```

OUT (sortida)

Sintaxi: `OUT port, acumulador`

Indicadors (flags):

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

Transfereix un byte o paraula del registre AL o AX a un port de sortida. El port es pot especificar amb un valor fix entre 0 i 255 ó a través del valor contingut en el registre DX (de 0 a 65535).

Exemple:

```
out     12h,ax
out     dx,al
```

INSTRUCCIONS ARITMÉTIQUES.

* * * S U M A * * *

AAA (ajust ASCII per la suma)

Sintaxi: AAA

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
?	--	--	--	**?**	**?**	**X**	**?**	**X**

Converteix el contingut del registre AL a un número BCD no empaquetat. Si els quatre bits menys significatius de AL son més grans que 9 ó si l'indicador AF esta a 1, se suma 6 a AL, 1 a AH, AF es posa a 1, CF s'iguala a AF i AL posa els seus quatre bits més significatius a 0.

Exemple:

```
add     al,bl
aaa
```

En el exemple, després de la suma de dos números BCD no empaquetats posats a AL i BL, el resultat (a traves de AAA) segueix següent un número BCD no empaquetat.

ADC (suma amb carry)

Sintaxi: `ADC destí, origen`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	X	**X**	**X**	**X**	**X**

Suma els operants origen, destí i el valor del indicador de carry (0 ó 1) i el resultat s'emmagatzema a operant destí. S'utilitza normalment per sumar números grans, de més de 16 bits, en varis passos, considerant el que portem (el carry) de la suma anterior.

Exemple:

```
adc     ax,bx
```

ADD (suma)

Sintaxi: `ADD destí, origen`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF

--	--	--	--	--	--	--	--	--

Suma els operants origen i destí emmagatzemant el resultat a operant destí. S'activa el carry si es desborda el registre destí durant la suma.

Exemples:

```
add     ax,bx
add     cl,dh
```

DAA (ajust decimal per la suma)

Sintaxi: `DAA`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
?	--	--	--	**X**	**X**	**X**	**X**	**X**

Converteix el contingut del registre AL en un parell de valors BCD: si els quatre bits menys significatius de AL son un número major que 9, l'indicador AF es posa a 1 i se suma 6 a AL. De igual forma, si els quatre bits més significatius de AL després de la operació anterior resulta un número major de 9, l'indicador CF es posa a 1 i se suma 60h a AL.

Exemple:

```
add     al,cl
daa
```

En el exemple anterior, si AL y CL contenien dos números BCD empaquetats, DAA fa que el resultat de la suma (a AL) segueixi sent també un BCD empaquetat.

INC (incrementar)

Sintaxi: `INC destí`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	**X**	**X**	**X**	**X**	--

Incrementa l'operant destí. Operant destí pot ser byte o paraula. Cal observar que aquesta instrucció no modifica el bit de carry (CF) i no es possible detectar un desbordament per aquest procediment (cal usar ZF).

Exemples:

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

* * * R E S T A * * *

AAS (ajust ASCII per la resta)

Sintaxi: AAS

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
?	--	--	--	?	?	X	?	X

Converteix el resultat de la resta de dos operants BCD no empaquetats per que segueixi sent un número BCD no empaquetat. Si el nibble inferior de AL te un valor major que 9, a AL es resta 6, es decrementa AH, AF es posa a 1 i CF s'iguala a AF. El resultat es guarda a AL amb els bits del 4 al 7 posats a 0.

Exemple:

```
sub     al,bl
aas
```

En el exemple, després de la resta de dos números BCD no empaquetats posats a AL y BL, el resultat (a traves de AAS) segueix sent un número BCD no empaquetat.

CMP (comparació)

Sintaxi: `CMP destí, origen`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	X	X	X	X	X

Resta origen de destí sense retornar cap resultat. Els operants resten sense canvis, però els indicadors poden ser consultats mitjançant instruccions de bifurcació condicional. Els operants poden ser de tipus byte o paraula però ambdós de la mateixa dimensió.

Exemple:

```
cmp     bx, mem_pal
cmp   ch,cl
```

DAS (ajust decimal per la resta)

Sintaxi: `DAS`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
--	--	--	--	X	X	X	X	X

Corregeix el resultat a AL de la resta de dos números BCD empaquetats, convertint-lo també en un valor BCD empaquetat. Si el nibble inferior te un valor major de 9 o AF es 1,

a AL se li resta 6, AF es posa a 1. Si el nibble més significatiu es major de 9 ó CF esta a 1, llavors es resta 60h a AL i s'activa després CF.

Exemple:

```
sub     al,bl
das
```

En el exemple anterior, si AL i BL contenien dos números BCD empaquetats, DAS fa que el resultat de la resta (a AL) sigui sent també un BCD empaquetat.

DEC (decrementar)

Sintaxi: `DEC destí`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	**X**	**X**	**X**	**X**	--

Resta una unitat de operant destí. Operant pot ser byte o paraula. Cal observar que aquesta instrucció no modifica el bit de carry (CF) i no es possible detectar un desbordament per aquest procediment (cal usar ZF).

Exemple:

```
dec     ax
dec     mem_byte
```

NEG (negació)

Sintaxi: `NEG destí`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	**X**	**X**	**X**	**X**	**X**

Calcula el valor negatiu en complement a dos del operant i retorna el resultat en el mateix operant.

Exemple:

```
neg     al
```

SBB (resta amb carry)

Sintaxis: SBB `destí, origen`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	X	X	X	X	X

Resta operant origen del operant destí i el resultat el guarda a operant destí. Si esta a 1 l'indicador de carry i resta una unitat mes. Els operants poden ser de tipus byte o paraula. S'utilitza normalment per restar números grans, de més de 16 bits, en varis passos, considerant el que portem (el carry) de la resta anterior.

Exemple:

```
sbb     ax,ax
sbb     ch,dh
```

SUB (resta)

Sintaxis: SUB `destí, origen`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	X	X	X	X	X

Resta operant destí al operant origen, deixant el resultat a operant destí. Els operants poden tenir o no signe, següent necessari que siguin del mateix tipus, byte o paraula.

Exemples:

```
sub     al,bl
sub     dx,dx
```

* * * M U L T I P L I C A C I O * * *

AAM (ajust ASCII per la multiplicació)

Sintaxi: AAM

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
?	--	--	--	X	X	?	X	?

Corregeix el resultat a AX del producte de dos números BCD no empaquetats, convertint-lo en un valor BCD també no empaquetat. A AH queda el quocient de AL/10 queden a AL la resta d'aquesta operació.

Exemple:

```
mul     bl.
ama
```

En el exemple, després del producte de dos números BCD no empaquetats situats a AL i BL, el resultat (a traves de AAM) segueix sent, a AX, un número BCD no empaquetat.

IMUL (multiplicació entera amb signe)

Sintaxi: `IMUL origen` (origen no pot ser operant immediat a 8086, sí a 286)

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	**?**	**?**	**?**	**?**	**X**

Multiplica un operant origen amb signe de longitud byte o paraula per AL o AX respectivament. Si origen es un byte el resultat es guarda a AH (byte més significatiu) i a AL (menys significatiu), si origen es una paraula el resultat es diposita a DX (part alta) i AX (part baixa). Si les meitats més significatives son distintes de zero, independentment del signe, CF y OF son activats.

Exemple:

```
imul     bx
imul     ch
```

MUL (multiplicació sense signe)

Sintaxi: `MUL origen` (origen no pot ser operant immediat)

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	**?**	**?**	**?**	**?**	**X**

Multiplica el contingut sense signe del acumulador per operant origen. Si operant destí es un byte l'acumulador es AL guardant el resultat a AH i AL, si el contingut de AH es diferent de 0 activa els indicadors CF y OF. Quan operant origen es de longitud paraula l'acumulador es AX quedant el resultat sobre DX y AX, si el valor de DX es diferent de zero els indicadors CF y OF s'activen.

Exemple:

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

* * * D I V I S I O * * *

AAD (ajust ASCII per la divisió)

Sintaxi: AAD

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
?	--	--	--	**X**	**X**	**?**	**X**	**?**

Converteix dos números BCD no empaquetats continguts a AH i AL en un dividend d'un byte que queda emmagatzemat a AL. Després la operació AH queda a zero. Aquesta instrucció es necessària ABANS de l'operació de dividir, al contrari que AAM.

Exemple:

```
aad
div     bl.
```

En el exemple, després de convertir els dos números BCD no empaquetats (a AX) en un dividend vàlid, la instrucció de dividir genera un resultat correcte.

DIV (divisió sense signe)

Sintaxi: `DIV origen` (origen no pot ser operant immediat)

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
?	--	--	--	**?**	**?**	**?**	**?**	**?**

Divideix, sense considerar el signe, un número contingut a l'acumulador i la seva extensió (AH, AL si el operant es del tipus byte o DX, AX si operant es paraula) entre operant font. El quocient es guarda a AL o AX i la resta a AH o DX segons operant sigui byte o paraula respectivament. DX o AH tenen d'estar a 0 abans de l'operació. Quan el quocient es major que el resultat màxim que es pot emmagatzemar, quocient i resta queden indefinides produint-se una interrupció 0. En cas de que les parts més significatives del quocient tinguin un valor diferent de zero s'activen els indicadors CF i OF.

Exemple:

```
div     bl.
div     mem_pal
```

IDIV (divisió entera)

Sintaxi: `IDIV origen` (origen no pot ser operant immediat)

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
?	--	--	--	**?**	**?**	**?**	**?**	**?**

Divideix, considerant el signe, un número contingut a l'acumulador i la seva extensió entre operant font. El quocient s'emmagatzema a AL o AX segons operant sia byte o paraula i de igual manera la resta a AH o DX. DX o AH tenen que ser zero abans de l'operació. Quan el quocient es positiu i superior al valor màxim que es pot emmagatzemar (7fh ó

7fffh), o quan el quocient es negatiu i inferior al valor mínim que es pot emmagatzemar (81h o 8001h) llavors quocient i resta queden indefinits, generant-se una interrupció 0, el que també succeeix si el divisor es 0.

Exemple:

```
        idiv    bl.
        idiv    bx
```

* * * C O N V E R S I O N S * * *

CBW (conversió de byte a paraula)

Sintaxi: `CBW`

Indicadors (flags):

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

Copia el bit 7 del registre AL a tots els bits del registro AH, es a dir, expandeix el signe de AL a AX com pas previ a una operació de 16 bits.

CWD (conversió de paraula a doble paraula)

Sintaxi: `CWD`

Indicadors (flags):

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

Expandeix el signe del registre AX sobre el registre DX, copiant el bit més significatiu de AH a tot DX.

INSTRUCCIONS DE MANIPULACIÓ DE CADENES.

CMPS/CMPSB/CMPSW (compara cadenes)

Sintaxi:

```
CMPS        cadena_desti, cadena_origen
CMPSB      (bytes)
CMPSW     (paraules)
```

Indicadors (flags):

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

Compara dos cadenes restant al origen el destí. Cap dels operants s'alteren, preo els indicadors resulten afectats. La cadena origen s'adreça amb el registre SI sobre el segment de dades DS i la cadena destí s'adreça amb el registre DI sobre el segment extra ES. Els registres DI i SI s'autoincrementen o autodecrementen segons el valor de l'indicador DF (veure CLD y STD) en una o dos unitats, depenent de si es treballa amb bytes o amb paraules. Cadena origen y cadena destí son dos operants redundants que només indiquen el tipus del dada (byte o paraula) a comparar, es més còmode col·locar aquest CMPSB o CMPSW per indicar bytes/paraules. Si s'indica un registre de segment, aquest substituirà a la cadena origen al DS ordinari.

Exemple:

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

LODS/LODSB/LODSW (carregar cadena)

Sintaxi:

```
LODS      cadena_origen
LODSB     (bytes)
LODSW     (palabras)
```

Indicadors (flags):

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

Copia a AL o AX una cadena de longitud byte o paraula adreçada sobre el segment de dades (DS) amb el registre SI. Després de la transferència, SI s'incrementa o decrementa segons l'indicador DF (veure CLD i STD) en una o dos unitats, segons s'estiguin manegant bytes o paraules. Cadena_origen es un operant redundant que només indica el tipus de dada (byte o paraula) que s'ha de carregar, es més còmode col·locar LODSB o LODSW per indicar bytes/paraules.

Exemple:

```
cld
lea     si,origen
lodsb
```

MOVS/MOVSB/MOVSW (moure cadena)

Sintaxi:

```
MOVS     cadena_desti, cadena_origen
MOVSB     (bytes)
MOVSW     (paraules)
```

Indicadors (flags):

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

Transfereix un byte o una paraula de la cadena origen adreçada per DS:SI a la cadena destí adreçada per ES:DI, incrementant o decrementant a continuació els registres SI y DI segons el valor de DF (veure CLD i STD) en una o dos unitats, depenent de si es treballa amb bytes o amb paraules. Cadena origen i cadena destí son dos operants redundants que només indiquen el tipus del dada (byte o paraula) a comparar, es més còmode col·locar MOVSB o MOVSW per indicar bytes/paraules. Si s'indica un registre de segment, aquest substituirà a la cadena origen al DS ordinari.

Exemple:

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

SCAS/SCASB/SCASW (explorar cadena)

Sintaxi:

```
SCAS      cadena_destino
SCASB     (bytes)
SCASW     (palabras)
```

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	**X**	**X**	**X**	**X**	**X**

Resta de AX o AL una cadena destí adreçada per el registre DI sobre el segment extra. Cap dels valors es alterat preo els indicadors es veuen afectats. DI s'incrementa o decrementa segons el valor de DF (veure CLD y STD) en una o dos unitats - segons s'estigui treballant amb bytes o paraules - per apuntar al següent element de la cadena. Cadena_destí es un operant redundant que solament indica el tipus de dada (byte o paraula), es més còmode col·locar SCASB o SCASW per indicar bytes/paraules.

Exemple:

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

STOS/STOSB/STOSW (emmagatzema cadena)

Sintaxi:

```
STOS      cadena_desti
STOSB     (bytes)
STOSW     (paraules)
```

Indicadors (flags):

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

Transfereix operant origen emmagatzemat a AX o AL, al destí adreçat per el registre DI sobre el segment extra. Després de la operació, DI s'incrementa o decrementa segons l'indicador DF (veure CLD y STD) per apuntar al següent element de la cadena. Cadena_destí es un operant redundant que només indica el tipus del dada (byte o paraula) a carregar, es més còmode col·locar STOSB o STOSW per indicar bytes/paraules.

Exemple:

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

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

REP	repetir operació de cadena
REPE/REPZ	repetir operació de cadena si igual/si zero
REPNE/REPNZ	repetir operació de cadena si no igual (si no 0)

Aquestes instruccions es poden col·locar com prefix de una altra instrucció de manejament de cadenes, amb l'objecta de que la mateixa es repeteixi un nombre determinat de vegades incondicionalment o fins que es verifiqui alguna condició. El nombre de vegades s'indica a CX. Per sentit comú només es tenen d'utilitzar les següents combinacions:

Prefix	Funció	Instruccions
REP	Repetir CX vegades	MOVS, STOS
REPE/REPZ	Repetir CX vegades mentre ZF=1	CMPS, SCAS
REPNE/REPNZ	Repetir CX vegades mentre ZF=0	CMPS, SCAS

Exemples:

1) Cercar el byte 69 entre les 200 primeres posicions de taula (se suposa taula en el segment ES):

```
LEA       DI,taula
MOV       CX,200
MOV            AL,69
CLD
REPNE     SCASB
```

```
JE        trobat
```

2) Omplir de zeros 5000 bytes d'una taula col·locada a dades (se suposa dades en el segment ES):

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

3) Copiar la memòria de pantalla de text (adaptador de color) d'un PC a un buffer (se suposa buffer en el segment ES):

```
MOV       CX,0B800h ; segment de pantalla
MOV       DS,CX     ; a DS
LEA       DI,buffer ; destí a ES:DI
MOV       SI,0           ; copiar des de DS:0
MOV       CX,2000   ; 2000 paraules
CLD                 ; cap endavant
REP       MOVSW     ; copiar CX paraules
```

INSTRUCCIONS D'OPERACIONS LÓGIQUES A NIVELL DE BIT.

AND (i lògic)

Sintaxi: AND destí, origen

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
0	--	--	--	**X**	**X**	**?**	**X**	**0**

Realitza una operació de I lògic entre operant origen i destí queden el resultat en el destino. Son vàlids operants byte o paraula, preo amdos del mateix tipus.

Exemples:

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

NOT (no lògic)

Sintaxi: `NOT destí`

Indicadors (flags):

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

Realitza el complement a 1 de operant destí, invertint cadascun dels seus bits. Els indicadors no resulten afectats.

Exemple:

```
not     ax
```

OR (O lògic)

Sintaxi: `OR destí, origen`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
0	--	--	--	**X**	**X**	**?**	**X**	**0**

Realitza una operació O lògic a nivell de bits entre els dos operants, emmagatzemant-se després el resultat en el operant destí.

Exemple:

```
or      ax,bx
```

TEST (comparació lògica)

Sintaxi: `TEST destí, origen`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
0	--	--	--	**X**	**X**	**?**	**X**	**0**

Realitza una operació I lògica entre els dos operants però sense emmagatzemar el resultat. Els indicadors son afectats amb l'operació.

Exemple:

```
test     al,bh
```

XOR (O exclusiva)

Sintaxi: `XOR destí, origen`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
0	--	--	--	**X**	**X**	**?**	**X**	**0**

Operació OR exclusiva a nivell de bits entre los operants origen i destí emmagatzemant el resultat en aquest últim.

Exemple:

```
xor      di,ax
```

INSTRUCCIONS DE CONTROL DEL PROCESSADOR.

NOP (operació nul·la)

Sintaxi: `NOP`

Indicadors (flags):

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

Realitza una operació nul·la, es a dir, el microprocessador descodifica la instrucció i passa a la següent. Realment es tracta de la instrucció XCHG AX,AX.

ESC (sortida a un coprocessador)

Sintaxi: `ESC codi_operació, origen`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF

--	--	--	--	--	--	--	--	--

S'utilitza en combinació amb processadors externs, com els coprocessadors de coma flotant o de E/S, i obre al dispositiu extern el accés a les adreces i operants requerits. Al mnemònic ESC el segueixen els codis d'operació apropiats per el coprocessador així com la instrucció i l'adreça del operant necessari.

Exemple:

```
esc     21,ax
```

HLT (parada fins interrupció o reset)

Sintaxi: `HLT`

Indicadors (flags):

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

El processador es para fins que se restaura el sistema o es rep una interrupció. Com en els PC es produeixen normalment 18,2 interrupcions de tipus 8 per segon (del temporitzador) alguns programadors utilitzen HLT per fer pauses i bucles de retard. De totes formes, el mètode no es precís i pot fallar amb certs controladors de memòria.

LOCK (bloqueja els bussos)

Sintaxi: `LOCK`

Indicadors (flags):

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

Es una instrucció que s'utilitza en aplicacions de recursos compartits per assegurar que no accedeix simultàniament a la memòria més de un processador. Quan una instrucció va precedida per LOCK, el processador bloqueja immediatament el bus, introduint un senyal per el pin LOCK.

WAIT (espera)

Sintaxi: `WAIT`

Indicadors (flags):

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

Provoca l'espera del processador fins que es detecta un senyal en el pin TEST. Passa, per exemple, quan el coprocessador ha acabat una operació i indica la seva fi. Sol precedir a ESC per sincronitzar les accions del processador i coprocessador.

INSTRUCCIONS DE ROTACIÓ I DESPLAÇAMENT.

RCL (rotació a l'esquerra amb carry)

Sintaxi: `RCL destí, comptador`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	--	--	--	--	**X**

Rodar a l'esquerra els bits del operant destí junt amb l'indicador de carry CF el número de bits especificat en el segon operant. Si el número de bits a desplaçar es 1, es pot especificar directament, en cas contrari el valor deu carregar-se a CL i especificar CL com segon operant. No es convenient que CL sigui major de 7, en bytes; ó 15, en paraules.

Exemples:

```
        rcl     ax,1
        rcl     al,cl
        rcl     di,1
```

RCR (rotació a la dreta amb carry)

Sintaxi: `RCR destí, comptador`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	--	--	--	--	**X**

Rodar a la dreta els bits del operant destí juntament amb l'indicador de carry CF el número de bits especificat en el segon operant. Si el número de bits es 1 es pot especificar directament; en cas contrari el seu valor es te de carregar a CL i especificar CL com segon operant:

Exemples:

```
rcr     bx,cl
rcr     bx,1
```

ROL (rotació a l'esquerra)

Sintaxi: `ROL destí, comptador`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	--	--	--	--	**X**

Roda a l'esquerra els bits del operant destí el número de bits especificat en el segon operant, que pot ser 1 ó CL prèviament carregat amb el valor del número de vegades.

Exemples:

```
rol     dx,cl
rol     ah,1
```

ROR (rotació a la dreta)

Sintaxi: `ROR destí, comptador`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	--	--	--	--	**X**

Roda a la dreta els bits del operant destí el número de bits especificat en el segon operant. Si el número de bits es 1 es pot posar directament, en cas contrari es te de posar a través de CL.

Exemples:

```
ror     cl,1
ror     ax,cl
```

SAL/SHL (desplaçament aritmètic a l'esquerra)

Sintaxi: `SAL/SHL destí, comptador`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	**X**	**X**	**?**	**X**	**X**

Desplaça a l'esquerra els bits del operant el número de bits especificat en el segon operant que te de ser CL si es major que 1 els bits desplaçats.

SAR (desplaçament aritmètic a la dreta)

Sintaxi: `SAR destí, comptador`

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	**X**	**X**	**?**	**X**	**X**

Desplaça a la dreta els bits del operant destí el número de bits especificat en el segon operant. Els bits de l'esquerra s'emplenen amb el bit de signe del primer operant. Si el número de bits a desplaçar es 1 es pot especificar directament, si es major s'especifica a través de CL.

Exemples:

```
sar     ax,cl
sar     bp,1
```

SHR (desplaçament lògic a la dreta)

Sintaxi: SHR destí, comptador

Indicadors (flags):

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	--	--	--	X	X	?	X	X

Desplaça a la dreta els bits del operant destí el número de bits especificats en el segon operant. Els bits de la esquerra s'emplenen amb zero. Si el número de bits a desplaçar es 1 es pot directament en el cas en que no passi es posa el valor a CL:

Exemples:

```
        shr     ax,cl
        shr     cl,1
```

RESUMEN ALFABÉTIC DE LES INSTRUCCIONS I BANDERINS. ÍNDEX.

Nota: en el efecte de les instruccions sobre el registre d'estat s'utilitzarà la següent notació:

- \- bit no modificat
- ? desconegut o indefinit
- x modificat segons el resultat de la operació
- 1 posat sempre a 1
- 0 posat sempre a 0

Instrucció	Sintaxi	Efecte sobre els flags								
		OF	DF	IF	TF	SF	ZF	AF	PF	CF
AAA	AAA	?	-	-	-	?	?	x	?	x
AAD	AAD	?	-	-	-	x	x	?	x	?
AAM	AAM	?	-	-	-	x	x	?	x	?
AAS	AAS	?	-	-	-	?	?	x	?	x
ADC dst,fnt	ADC dst,fnt	x	-	-	-	x	x	x	x	x
ADD dst,fnt	ADD dst,fnt	x	-	-	-	x	x	x	x	x
AND dst,fnt	AND dst,fnt	0	-	-	-	x	x	?	x	0
CALL dsp	CALL dsp	-	-	-	-	-	-	-	-	-
CBW	CBW	-	-	-	-	-	-	-	-	-
CLC	CLC	-	-	-	-	-	-	-	-	0
CLD	CLD	-	0	-	-	-	-	-	-	-
CLI	CLI	-	-	0	-	-	-	-	-	-
CMC	CMC	-	-	-	-	-	-	-	-	x
CMP dst,fnt	CMP dst,fnt	x	-	-	-	x	x	x	x	x
CMPS/CMPSB										

CMPSW cdst,cfnt	CMPS cdst,cfnt	x	-	-	-	x	x	x	x	x
CWD	CWD	-	-	-	-	-	-	-	-	-
DAA	DAA	?	-	-	-	x	x	x	x	x
DAS	DAS	-	-	-	-	x	x	x	x	x
DEC dst	DEC dst	x	-	-	-	x	x	x	x	-
DIV fnt	DIV dst	?	-	-	-	?	?	?	?	?
ESC opcode,fnt	ESC opcode,fnt	-	-	-	-	-	-	-	-	-
HLT	HLT	-	-	-	-	-	-	-	-	-
IDIV fnt	IDIV fnt	?	-	-	-	?	?	?	?	?
IMUL fnt	IMUL fnt	x	-	-	-	?	?	?	?	x
IN acum,port	IN acum,port	-	-	-	-	-	-	-	-	-
INC dst	INC dst	x	-	-	-	x	x	x	x	-
INT interrup	INT interrup	-	-	0	0	-	-	-	-	-
INTO	INTO	-	-	0	0	-	-	-	-	-
IRET	IRET	x	x	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	-	-	-	-	-	-	-	-	-

Instrucció	Sintaxi	Efecte sobre els flags								
		OF	DF	IF	TF	SF	ZF	AF	PF	CF
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	-	-	-	-	-	-	-	-	-
MUL fnt	MUL fnt	x	-	-	-	?	?	?	?	x
NEG dst	NEG fnt	x	-	-	-	x	x	x	x	x
NOP	NOP	-	-	-	-	-	-	-	-	-
NOT dst	NOT dst	-	-	-	-	-	-	-	-	-
OR dst,fnt	OR dst,fnt	0	-	-	-	x	x	?	x	0
OUT port,acum	OUT port,acum	-	-	-	-	-	-	-	-	-
POP dst	POP dst	-	-	-	-	-	-	-	-	-
POPF	POPF	x	x	x	x	x	x	x	x	x
PUSH dst	PUSH dst	-	-	-	-	-	-	-	-	-
PUSHF	PUSHF	-	-	-	-	-	-	-	-	-
RCL dst,cnt	RCL dst,cnt	x	-	-	-	-	-	-	-	x
RCR dst,cnt	RCR dst,cnt	x	-	-	-	-	-	-	-	x
REP/REPE/REPZ/										
REPNE/REPNZ	REP	-	-	-	-	-	-	-	-	-
RET [val]	RET [val]	-	-	-	-	-	-	-	-	-
RETF [val]	RETF [val]	-	-	-	-	-	-	-	-	-
ROL dst,cnt	ROL dst,cnt	x	-	-	-	-	-	-	-	x
ROR dst,cnt	ROR dst,cnt	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	?	x	x
STC	STC		-	-	-	-	-	-	-	-	1
STD	STD		-	1	-	-	-	-	-	-	-
STI	STI		-	-	1	-	-	-	-	-	-
STOS/STOSB/ STOSW cdst	STOS	cdst	-	-	-	-	-	-	-	-	-
SUB dst,fnt	SUB	dst,fnt	x	-	-	-	x	x	x	x	x
TEST dst,fnt	TEST	dst,fnt	0	-	-	-	x	x	?	x	0
WAIT	WAIT		-	-	-	-	-	-	-	-	-
XCHG dst,fnt	XCHG	dst,fnt	-	-	-	-	-	-	-	-	-
XLAT tfnt	XLAT	tfnt	-	-	-	-	-	-	-	-	-
XOR dst,fnt	XOR	dst,fnt	0	-	-	-	x	x	?	x	0

INSTRUCCIONS ESPECIFIQUES DEL 286, 386 y 486 EN MODUS REAL.

DIFERENCIES EN EL COMPORTAMENT GLOBAL RESPECTE AL 8086.

- Excepcions de divisió:
Les excepcions INT 0, degudes a una divisió per zero o a un quocient excessivament gran, provoquen que a la pila s'emmagatzemi el valor de CS:IP per la següent instrucció en el 8086. En el 286 i superiores s'emmagatzema el CS:IP de la pròpia instrucció que causa la excepció.

- Codis d'operació indefinits.
En el 286 i superiors es produeix una excepció 6 (INT 6) o, si es una
Instrucció amb sentit per aquests processadors, s'executa. El 8086 s'estavella.

- Valor de PUSH SP.
El valor que introdueix a la pila en el 286 i superiors es el de SP abans del PUSH; en el 8086 es el de SP després del PUSH (dos unitats menys).

- Desplaçaments i rotacions.
El valor de desplaçament en las operacions de manipulació de bits del 8086 es una constant de 8 bits (indicada a CL); en el 286 i superiors es pren el mòdul 32 (només es consideren els 5 bits menys significatius).

- Prefixes redundants.
Las instruccions tenen una longitud il·limitada en el 8086; en el 286 i superiors no poden excedir de 15 bytes. Per tant, els prefixes redundants poden produir excepcions de codi d'operació no vàlid.

- Accessos al límit del segment.
Un accés de 16 bits en el offset 0FFFFh en el 8086 provoca un accés als bytes ubicats a las posicions 0FFFFh i 0 (es dona la volta al voltant del segment). En el 286 i superiors, es produeix una excepció de violació de límits. En el 386 i superiors es produeix també a accessos de 32 bits a las posicions 0FFFDh a la 0FFFFh. Això es compleix tant per accessos a dades a memòria com per instruccions del programa en aquests punts crítics.

LOCK.

Aquesta instrucció no esta limitada de cap manera en el 8086 i en el 286. En el 386 i superiores la utilització esta restringida a determinades instruccions.

- Execució pas a pas.
La prioritat de la excepció pas a pas en el 286 i superiors es més alta que la d'una interrupció externa; per tant, les interrupcions externes no es poden traçar.

- Registre de FLAGS.
Difereix quelcom en els bits 12 al 15 a tots els processadors; el 386 disposa a més a més d'un registre de flags de 32 bits.

- Interrupció NMI.
Des del 286 i superiors, una NMI no pot interrompre una rutina de tractament NMI.

- Error del coprocessador.
En el 286 i superiors s'utilitza el vector 16; en el 8086 qualsevol vector.

- Prefixes de les instruccions del coprocessador.
Al produir-se una excepció d'error de coprocessador, en el 8086 s'emmagatzema un CS:IP que no inclou prefixes - si n'hi havia -, al contrari que en el 286 i superiors.

- Límit del primer megabyte.
En el 8086 la memòria es circular; al final del primer megabyte es torna a començar per les posicions més baixes de la memòria. En el 286 i superiors, s'accedeix a la memòria estesa (un artifici hardware en els PC ho impedeix al forçar A20 a estat baix, però es pot resoldre).

- Instruccions de cadena repetitives.
El CS:IP gravat en el 8086 no inclou el prefix, si existeix; en el 286 i superiors sí.

INSTRUCCIONS ESPECIFIQUES DEL 286.

A continuació es descriuen les instruccions addicionals que incorporen els 286 en mode real, que també poden ser considerades quan treballem amb els microprocessadors compatibles V20 y V30, així com amb els processadors superiors al 286. Les instruccions del mode protegit es dirigeixen especialment a la multiprogramació i el temps compartit, següent específiques de la commutació de processos i tractament de la memòria virtual i no poden emprar-se directament sota DOS.

BOUND r16, mem16: Comprova si el registre de 16 bits indicat com primer operant esta dins dels límits d'una matriu. Els límits de la matriu els defineixen dos paraules consecutives a la memòria apuntades per mem16. Si esta fora dels límits, es produeix una interrupció 5 a la que el IP apilat queda apuntant a la instrucció BOUND (¡no s'incrementa!).

ENTER crea una estructura de pila per un procediment d'alt nivell.

Les instruccions PUSH permeten posar valors immediats a la pila: es vàlid fer PUSH 40h.

IMUL pot multiplicar qualsevol registre de 16 bits per una constant immediata, tornant un resultat paraula (CF=1 si no cap en 16 bits); per exemple, es vàlid IMUL CX,25. També s'admeten tres operants: IMUL r1, r2, imm. En aquest cas, es multiplica r2 per el valor immediat (8/16 bits) i el resultat s'emmagatzema a r1. Tant r1 com r2 han de ser de 16 bits.

LEAVE abandona els procediments d'alt nivell (equival a MOV SP,BP / POP BP).

PUSHA/POPA: Introdueix a la pila i en aquest ordre els registres AX, CX, DX, BX, SP, BP, SI i DI - i els treu en ordre invers -. Ideal en el manejament d'interrupcions i molt usada a les BIOS de 286 y 386.

OUTS (sortida de cadenes) i **INS** (entrada de cadenes) repetitives (equivalent a MOVS i LODS).

RCR/RCL, ROR/ROL, SAL/SAR i SHL/SHR que admeten una constant de rotació diferent de 1.

INSTRUCCIONS PROPIES DEL 386 Y 486.

A més a més de totes les possibilitats addicionals del 286, el 386 i el 486 permeten utilitzar qualsevol registre de 32 bits de propòsit general en tots els modes de funcionament, inclòs el mode real, com EAX, EBX, ECX, EDX, ESI, EDI, EBP. De totes formes no s'ha d'intentar fer adreçats per sobre dels 64K. En altres paraules, se poden utilitzar per accelerar les operacions però no per accedir a més memòria. Per exemple, si EBX > 0FFFFh, la instrucció MOV AX,[EBX] tindria un resultat imprevist. I, aquests processadors tenen dos segments mes: a més de DS, ES, CS y SS es poden emprar també FS i GS. Avis: sembla ser que en alguns 386 fallen ocasionalment les instruccions de multiplicar de 32 bits.
Nota: No es del tot cert que el 386 i el 486 no permetin accedir a més de 64 Kb en mode real: més endavant veurem un exemple.

Els modes d'adreçat augmenten considerablement la seva flexibilitat en el 386 i superiors. Amb els registres de 16 bits només estan disponibles els modes tradicionals. En canvi, amb els de 32 es pot utilitzar en l'adreçat indirecte qualsevol registre: es vàlid, per exemple, una instrucció del tipus MOV AX,[ECX] o MOV EDX,[EAX]. Els desplaçaments en l'adreçat indexat amb registres de 32 bits poden ser de 8 i també de 32 bits. Quan dos registres es tenen de sumar per calcular l'adreça efectiva, el segon pot estar multiplicat per 2, 4 u 8; per exemple, es vàlida la instrucció MOV AL,[EDX+EAX*8]. Per suposat, sota DOS ens hem d'assegurar sempre que el resultat de totes les operacions que determinen l'adreça efectiva no excedeixi de 0FFFFh (0FFFEh si s'accedeix a paraules i 0FFFCh en accessos a dobles paraules a memòria).

BOUND r32, mem32: S'admeten ara operants de 32 bits.

BSF/BSR: Exploració de bits cap endavant i enrera, respectivament. La sintaxi es:

BSF reg, reg ó BSF reg, [memòria]
BSR reg, reg ó BSR reg, [memòria]

On reg pot ser de 16 ó 32 bits. Es comença a explorar per el bit 0 (BSF) o per el més significatiu (BSR) del segon operant: si no apareix cap bit actiu (a 1) l'indicador ZF s'activa; en cas contrari s'emmagatzema en el primer operant la posició relativa de aquest bit:

```
MOV     AX,8
BSF     BX,AX
JZ     ax_es_0     ; no es saltarà, a més a més BX = 3
```

BT/BTC/BTR/BTS: Operacions sobre bits: comprovació, comprovació i complement, comprovació i posta a 0, comprovació i posta a 1. Sintaxi (exemple sobre BT):

BT reg, reg ó BT reg, imm8

On reg pot ser de 16 ó 32 bits, el operant immediat es necessàriament de 8. Aquestes instruccions copien el número de bit del primer operant que indiqui el segon operant (entre 0 y 31) al carry. A continuació no l'hi fan res a aquest bit (BT), el complementen (BTC), l'esborren (BTR) o l'activen (BTS). Exemple:

```
MOV     AX,16
BTC      AX,4          ; resultat: CF = 1  y  AX = 0
```

CDQ: Similar a CWD, estén el signe de EAX a EDX:EAX.

CMPSD: Similar a CMPSW però emprant ESI, EDI, ECX i comparen dades de 32 bits. Es pot emprar sota DOS sempre que ESI y EDI (utilitzant REP també ECX) no excedeixin de 0FFFFh.

CWDE: Estén el signe de AX a EAX.

IMUL: Ara s'admet un adreçat a memòria en el segon operant: IMUL CX,[dato]

INSD: Similar a INSW però emprant ESI, EDI, ECX i llegint dades de 32 bits. Es pot emprar sota DOS sempre que ESI i EDI (utilitzant REP també ECX) no excedeixin de 0FFFFh.

JCC: Els salts condicionals ara poden ser de ¡32 bits!. Molta cura amb la directiva .386 en els programes en que es desitgi mantenir la compatibilitat amb processadors anteriors. JECXZ s'utilitza en lloc de JCXZ (mateix codi d'operació).

LODSD: Similar a LODSW però emprant ESI, EDI y ECX i carregant dades de 32 bits a EAX. Es pot emprar sota DOS sempre que ESI i EDI (utilitzant REP també ECX) no excedeixin de 0FFFFh.

LSS, LFS, LGS: similar a LDS o LES però amb aquests registres de segment.

MOV CRx,reg / MOV DRx,reg i els recíprocs: accés a registres de control i depuració.

MOVSD: Similar a MOVSW però emprant ESI, EDI, ECX i movent dades de 32 bits. Es pot emprar sota DOS per accelerar les transferències sempre que ESI i EDI (utilitzant **REP** també ECX) no excedeixin de 0FFFFh. Operant sobre la memòria de vídeo només s'obté avantatja si la targeta es realment de 32 bits.

MOVSX / MOVZX: carrega amb extensió de signe o zero. Pren el segon operant, l'estén adequadament el signe (o posa a zero la part alta) fins que sigui tan gran com el primer operant i llavors el carrega en el primer operant. Si el primer operant es de 16 bits, el segon només pot ser de 8; si el primer es de 32 bits el segon pot ser de 8 ó 16. El primer operant te de ser un registre, el segon pot ser un registre o operant a memòria (mai immediat):

```
MOV     EAX,0FFFFFFFFh
MOV     AX,7FFFh          ; resultat: EAX = 0FFFF7FFFh
MOVSX   EAX,AX            ; resultat: EAX = 000007FFFh
```

OUTSD: Similar a OUTSW però emprant ESI, EDI, ECX i enviant dades de 32 bits. Es pot emprar sota DOS sempre que ESI i EDI (utilitzant REP també ECX) no sobrepassin de 0FFFFh.

Prefixes FS: y GS: en els accessos a memòria, referència a aquests segments.

PUSHAD / POPAD: Similars a PUSHA i POPA però amb els registres de 32 bits. La instrucció POPAD falla en la majoria dels 386, inclosos els de AMD. Per resoldre el fallo (que consisteix en que EAX no es restaura correctament) basta col·locar un NOP immediatament després de POPAD.

PUSHFD/POPFD introdueixen i treuen de la pila els flags de 32 bits.

SCASD: Similar a SCASW però emprant ESI, EDI, ECX i buscant dades de 32 bits. Es pot emprar sota DOS sempre que ESI i EDI (utilitzant REP també ECX) no sobrepassin 0FFFFh.

SETcc reg8 ó mem8: Si es compleix la condició cc, es posa a 1 el byte de memòria o registre de 8 bits indicat (si no, a 0). Per exemple, amb el carry actiu, SETC AL posa a 1 el registre AL.

SHLD / SHRD: Desplaçament de doble precisió a l'esquerra/dreta. La sintaxi es (exemple sobre SHLD):

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

On regmem es un registre o operant a memòria, indistintament, de la mida indicada. En el caso de SHLD, es desplaça el primer operant a l'esquerra tan com indica el tercer operant (comptador). Un cop desplaçat, els bits menys significatius s'emplenen amb els més significatius del segon operant, que no resulta alterat. SHRD es semblant però al revés.

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

STOSD: Similar a STOSW però emprant ESI, EDI, ECX i emmagatzemant EAX. Es pot emprar sota DOS sempre que ESI i EDI (utilitzant REP també ECX) no sobrepassin de 0FFFFh.

DETECCIÓ D'UN SISTEMA AT O SUPERIOR.

Hi ha casos en els que es necessari determinar si una màquina es AT o superior: no de cara a emprar instruccions pròpies del 286 en mode real (també disponibles en els V20/V30 y 80188/80186) sinó degut a la necessitat d'accedir a certs chips (per exemple, el segon controlador d'interrupcions) que ja se sap que només porten les màquines AT o superiors. Es important per tan determinar la presencia d'un AT, de cara a evitar certes instruccions que podrien bloquejar un PC o XT. No s'ha de comprovar en aquests casos els bytes de la ROM que identifica l'equip: a vegades no son correctes i, a mes, l'evolució futura que tinguin no la sabem. Lo ideal es verificar directament si esta instal·lat un 286 o superior.

```
PUSHF
POP     AX       ; AX = flags
AND     AH,0Fh   ; esborrar nibble més significatiu
PUSH    AX
POPF         ; intentar posar a 0 els 4 bits més significatius dels flags
PUSHF
POP     AX
AND     AH,0F0h  ; seguiran valent 1 excepte en un 80286 o superior
CMP     AH,0F0h
JE      no_es_AT
JMP     si_es_AT ; es 286 o superior
```

AVALUACIÓ EXACTA DEL MICROPROCESSADOR INSTAL·LAT.

Sobra dir que les instruccions avançades tenen que ser utilitzades amb la prèvia comprovació del tipus de processador, encara que només sigui per dir a l'usuari que es compri una màquina més potent abans de avortar l'execució del programa. Per esbrinar el processador d'un ordinador pot emprar-se el següent programa d'utilitat, basat en el procediment processador que retorna a AX un codi numèric entre 0 y 8 distingint entre els 9 processadors més difícils d'identificar dels ordinadors compatibles. Nota: el 486 no te de tenir coprocessador necessàriament (el 486sx no el porta).

Algunes versions de processador 486 i tots els processadors posteriors suporten la instrucció CPUID que permet identificar la CPU. Basta comprovar un bit del registre d'estat per saber si esta suportada i, en aquest cas, poder emprar aquesta instrucció. D'aquest forma, resulta trivial detectar el Pentium o qualsevol processador posterior que aparegui.

Es normal que el acabat d'iniciar en el assemblador no entengui absolutament res d'aquest programa, ja que fins una mica més endavant no serà explicada la sintaxi del llenguatge. Cal recordar que les instruccions específiques del 286 en mode real també estan disponibles en els V20/V30 de NEC i la sèrie 80188/80186 de Intel.

```
; ********************************************************************
; *                                                                  *
; *   Aquest programa determina el tipus de microprocessador         *
; *   del equip i retorna un codi ERRORLEVEL que l'indica:           *
; *                                                                  *
; *              0-8088, 1-8086, 2-NEC V20, 3-NEC V30,               *
; *              4-80188, 5-80186, 6-286, 7-386, 8-486               *
; *                                                                  *
; *   Avis: Cal utilitzar TASM 2.0 o compatible exclusivament.       *
; *                                                                  *
; ********************************************************************

cpu             SEGMENT
                ASSUME CS:cpu, DS:cpu

                .386

                ORG   100h
inici:
                LEA   DX,texte_ini      ; text de salutació
                MOV   AH,9
                INT   21h               ; visualitzar
                CALL  procesador?       ; tipus de processador a AX
                PUSH  AX                ; guardar per el final
                LEA   BX,cpus_index-2   ; taula de noms-2
                MOV   CX,0FFFFh         ; número de iteració-1
altra_proc:     INC   CX
                ADD   BX,2
                MOV   DX,[BX]           ; nom del primer processador
                CALL  print
                CMP   CX,AX             ; ¿processador de l'equip?
                JNE   no_es_aquest
                LEA   DX,apuntador_txt  ; sí ho es: indicar-ho
                CALL  print
no_es_aquest:   LEA   DX,separador_txt
                CALL  print
                CMP   CX,7              ; número de CPUs tractades-1
                JBE   altra_proc
                LEA   DX,texte_fin      ; últims caràcters
                CALL  print
                MOV   AH,4Ch            ; retornar codi errorlevel AL
                INT   21h               ; fi de programa
```

```
procesador?    PROC       ; retornar el tipus de microprocessador a AX
               PUSHF
               PUSH  DS
               PUSH  ES
               PUSH  CX
               PUSH  DX
               PUSH  DI
               PUSH  SI
               MOV   AX,CS
               MOV   DS,AX          ; durant la rutina es guardarà
               MOV   ES,AX          ; el tipus de processador a DL:
               MOV   DL,6           ; suposat un 286 (DL=6) ...
               PUSHF
               POP   AX             ; AX = flags
               AND   AX,0FFFh       ; esborrar nibble més significatiu
               PUSH  AX
               POPF                 ; intentar posar a 0 els 4 bits mes
               PUSHF                ; significatius dels flags
               POP   AX
               AND   AX,0F000h      ; seguiran valen 1 excepte a
               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 posa bits 12, 13 y 14 a zero
               JZ    cpu_trobada    ; es un 286 (DL=6)
               INC   DL             ; es un 386 (DL=7) ... de moment
               PUSH  DX
               CLI                  ; moment crític
               MOV   EDX,ESP        ; preservar ESP en EDX
               AND   ESP,0FFFFh     ; esborrar part alta de ESP
               AND   ESP,0FFFCh     ; forçar ESP a múltiple de 4
               PUSHFD               ; guardar flags a la pila (32 bits)
               POP   EAX            ; recuperar flags a EAX
               MOV   ECX,EAX
               XOR   EAX,40000h     ; commutar bit 18
               PUSH  EAX
               POPFD                ; intentar canviar aquest bit
               PUSHFD
               POP   EAX            ; ECX conserva el bit inicial
               XOR   EAX,ECX        ; bit 18 de EAX a 1 si ha canviat
               SHR   EAX,12h        ; moure bit 18 a bit 0
               AND   EAX,1          ; deixar només aquest bit
               PUSH  ECX
               POPFD                ; restaurar bit 18 dels flags
               MOV   ESP,EDX        ; restaurar ESP
               STI                  ; permetre interrupcions un altra cop
               POP   DX             ; recuperar tipus de CPU a DL
               CMP   AX,0
               JE    cpu_trobada    ; es 386: DL=7 (bit 18 no ha canviat)
               INC   DL             ; es 486: DL=8 (bit 18 ha canviat)
               JMP   cpu_trobada
ni286ni_super: MOV   DL,4           ; suposem un 80188 ...
               MOV   AX,0FFFFh
               MOV   CL,33
               SHL   AX,CL          ; (80188/80186 prenen  CL mod 32)
               JNZ   tipus_bus_proc ; ... ho es, calcular bus (188/186)
               MOV   DL,2           ; no ho es, suposem un V20 ...
```

```
                MOV   CX,0FFFFh
                STI
                DB    0F3h,26h,0ACh   ; opcode de REPZ  LODSB ES:
                JCXZ  tipus_bus_proc  ;... ho es, calcular bus (V20/V30)
                XOR   DL,DL           ; ja només pot ser un 8088/8086
tipus_bus_proc: STD                   ; transferències cap a dalt
                LEA   DI,tipus_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) serà sobreescrit
                NOP                 ; amb STOSB però encara s'executarà
                NOP                 ; en un 8086/80186/V30 (i no en un
                INC   CX            ; 8088/80188/V20) perquè esta a la
tipus_bus_byte: STI                 ; cua de lectura avançada.
tipus_bus_dest: STI
                JCXZ  cpu_hallada   ; el bus ja era de 8 bits
                INC   DL            ; resulta que es de 16
cpu_trobada:    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_index      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    " <---$"

texte_ini        LABEL BYTE
                 DB    13,10,"CPU Test v2.2  "
                 DB    13,10,"  El microprocessador d'aquest "
                 DB    "equip es compatible:",10
separador_txt    DB    13,10,9,9,9,"$"
```

```
texte_fin       DB    13,10,"$"

cpu             ENDS
                END   inici
```

MODE PLA (FLAT) DEL 386 I SUPERIORS.

Com ja s'ha comentat, no es estrictament cert que no es pugui sobrepassar el límit de 64 Kb en els segments en mode real. El problema es que al engegar l'ordinador, el 386 te

definits per defecte aquests límits de 64 Kb. De totes maneres, es pot passar un moment a mode protegit, ampliar el límit i tornar a mode real. Llavors s'aconsegueix l'anomenat mode flat o pla. No només es factible d'aquesta manera saltar la restricció de 64 Kb, sinó que també es pot accedir directament, des de el mode real, a tota la memòria per sobre del primer megabyte.

El problema es que passar a mode protegit no es senzill quan la màquina ja esta en mode protegit emulant al mode real (el conegut com mode virtual 86). Per tant, el següent programa d'exemple no funciona si esta carregat un controlador de memòria expandida (EMM386, QEMM) o dins de Windows 3.x.
Arrancant sense controlador de memòria (excepte HIMEM) no hi haurà cap problema.
El programa de exemple es limita a emplenar la pantalla de text (emprant ara l'adreça absoluta 0B8000h a través de EBX) de lletres 'A'.

Una altra restricció d'aquest programa de exemple es que no activa la línia A20 d'adreces; dit d'una altra manera, el bit 21º (dels 32 bits de l'adreça de memòria) sol estar forçat a 0 per defecte al arrancar. Per accedir a la memòria de vídeo això no es problema, però per sobre del primer megabyte podria haver-hi problemes segons a quina adreça es vulgui accedir. De totes maneres, seria relativament senzill habilitar la línia A20 directament o a través d'una funció del controlador XMS.

Naturalment, se surt dels objectius descriure el mode protegit o explicar les passes que realitza aquesta rutina de demostració.

```
; +-----------------------------------------------------------------+
; | Rutina para activar el mode flat del 386 i superiors (accés     |
; | a 4 Gb en mode real).                                           |
; |                                                                 |
; |      TASM  flat386 /m5                                          |
; |      TLINK flat386 /t /32                                       |
; +-----------------------------------------------------------------+

                .386p                   ; només per 386 o superior

segment0        SEGMENT USE16
                ASSUME CS:segment0, DS:segment0

                ORG   100h
prova:
                CALL  flat386           ; activar mode flat
                XOR   AX,AX
                MOV   DS,AX
                MOV   EBX,0B8000h       ; adreça de vídeo absoluta
                MOV   CX,2000
ompla_pant:     MOV   BYTE PTR [EBX],'A'
                INC   EBX
                MOV   BYTE PTR [EBX],15
                INC   EBX
                LOOP  ompla_pant
                INT   20h               ; fi de programa

; ---------- Aquesta rutina passa momentàniament a mode protegit de
;            manera directa(necessita la CPU en mode real). No s'activa
;            la línia A20  (necessari fer-ho directament
;            o a través d'algun servei XMS  abans d'accedir a
;            les àrees de memòria estesa afectades).
```

```
flat386         PROC
                PUSH  DS
                PUSH  ES
                PUSH  EAX
                PUSH  BX
                PUSH  CX
                MOV   CX,SS
                XOR   EAX,EAX
                MOV   AX,CS
                SHL   EAX,4             ; adreça lineal de segment CS
                ADD   EAX,OFFSET gdt    ; desplaçament de GDT
                MOV   CS:[gd2],EAX      ; guardar adreça lineal de GDT
                CLI
                LGDT  CS:[gdtr]         ; carregar taula global de
descriptors
                MOV   EAX,CR0
                OR    AL,1              ; bit de mode protegit
                MOV   CR0,EAX           ; passar a mode protegit
                JMP   SHORT $+2         ; esborrar cua de precerca
                MOV   BX,gcodl          ; índex de descriptor a BX
                MOV   DS,BX             ; carregar registre de segment DS
                MOV   ES,BX             ; ES
                MOV   SS,BX             ; SS
                MOV   FS,BX             ; FS
                MOV   GS,BX             ; GS
                AND   AL,11111110b
                MOV   CR0,EAX           ; tornar a mode real
                JMP   SHORT $+2         ; esborrar cua de precerca
                MOV   SS,CX
                STI
                POP   CX
                POP   BX
                POP   EAX
                POP   ES
                POP   DS
                RET

gdtr            LABEL QWORD             ; dades per carregar en GDTR
gd1             DW    gdtl-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
gdtl            EQU   $-OFFSET gdt

flat386         ENDP


segment0        ENDS
                END   prova
```