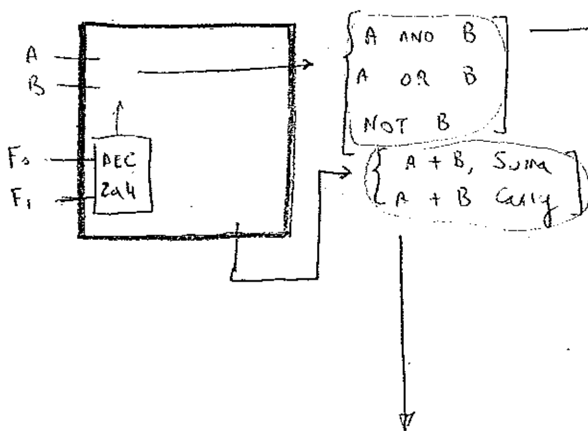
5. EL PROCESSADOR.

μP = microprocessador

5.1. ALU: Aritmetic-Logic Unit.

Fotocòpia Tanembaum, p. 111.

Circuit que admet 4 bits, i en genera 1 de sortida, (o 2)



→ Qualsevol expressió lògica genera una taula de veritat, expressable en forma de suma de MINTERMS, que tan sols necessiten els operadors lògics AND, OR, NOT.

Qualsevol operació lògica és avaluable en aquest circuit: exemple

$A \oplus B = (A \text{ OR } B) \text{ AND NOT } (A \text{ AND } B)$

Qualsevol operació aritmètica és convertible a sumes:

- SUMA → Ja ho és, sabem com fer un sumador de 8 bits.
- RESTA →
  - A − B es pot fer amb complement a 1 de B, i $A + C_1B$
  - Complement a 2 de B: NOT B + 1
  - En binari: e.g. B = 00010100 → 11101011 ← NOT B

    $$\begin{array}{r} 11101011 \\ +\quad\;\; 1 \\ \hline 11101100 \end{array}$$
    SUMA
- MULTIPLICACIÓ:
  - Sumes successives.
  - Hem de comptar el nº de vegades que es suma.
- DIVISIÓ: Restes successives. Hem de comptar el nº de vegades que es resta.

Qualsevol operació aritmètica és executable en aquest circuit.

Codis d'instrucció:

F_0	F_1	Operació que s'activa a la ALU
0	0	AB = A AND B
0	1	A+B = A OR B
1	0	NOT B
1	1	A+B, Suma i Carry.

→ Instruccions en "codi màquina" (tires de 1's i 0's)

ALU de 8 bits: pot fer AB, A+B, $\overline{B}$, A+B ~~suma i carry~~

amb operands de 8 bits. El codi F_0F_1 s'envia a les 8 ALUS d'1 bit simultàniament.

5.2 Unitat de Control.

En realitat són diferents subunitats, en general sincrones ~~totes elles~~, que tenen funcions com:

→ Llegir una instrucció emmagatzemada a la RAM, i portarla al DEC de l'ALU per que l'executi.

→ Llegir una dada que s'ha de sumar, des de RAM fins al lloc on es suma.

→ Calcular una adreça de RAM.

→ Emmagatzemar "status" d'operacions: "tot ok", "hi ha hagut un error", "encara no hem acabat", etc. (FLAGS) "Hi ha carry"

OF = Overflow;
ZF = Zero Flag.
CF = Carry Flag.
SF, Sign Flag

→ Atendre dispositius externs al μP que requereixen la seva atenció, o que han de fer cas al μP: "S'ha polsat una tecla", "s'ha d'escriure un caràcter en pantalla"... (interrupcions).

...

Fotocòpia ARQUITECTURA INTERNA del 8086.

5.3 Llenguatge màquina (VS) Ensamblador.

Ja hem vist que una instrucció és "una tira de 0 i 1" que es decodifica a la ALU, amb els codis d'instrucció F_0F_1 del nostre exemple.

Podem assignar un nom a cada instrucció, de manera que ens indiqui què fa la instrucció.

F_0 F_1	Operació	Mnemotècnic exemple:
0 0	AB	AND [Operands]
0 1	A + B	OR [Operands]
1 0	NOT B	NOT [Operand]
1 1	A Sumat a B, Suma i Carry	ADD [Operand]

- L'ordinador tan sols "executa" (~~és a dir, decodifica i activa~~ el circuit corresponent a la ALU) "tires de 1s i 0s". Es pot, però, posar nom a cada una d'aquestes "tires", de forma que ens doni una idea de què fa.
- El llenguatge de 1s i 0s és el <u>llenguatge màquina</u>.
- El llenguatge a base de noms mnemotècnics és <u>l'ensamblador</u>. (Assembler, llenguatge ASM).
- Les instruccions ASM tenen una estructura concreta, una <u>sintaxi</u>:

[etiqueta] nom [operand[s]] [comentari]

entre [ ], les parts opcionals.

- "nom" és, en el fons, una tira de 1s i 0s. Hi ha tans de noms (instruccions) com 2^n si n és el nº de bits que dedicam a codificar-los.
  → E.g. paraula de 16 bits ⇒ 8 bits nom + 8 bits operand ⇒ $2^8=256$ instruccions

Exemples d'instruccions ASM:

MOV Desti, Origen

S'activa un <u>registre de desplaçament</u> que "copia" els bits que hi ha a Origen, i els porta a Destí. Destí i Origen poden ser: <u>registres de la EU</u>, <u>posicions de memòria que tenen un nom</u>, <u>direccions de memòria</u>, <u>valors literals en hexadecimal</u>.

E.g.
```
MOV AX, 0H
MOV AX, BX
MOV AX, Factor1   → posició de memòria "amb nom".
```

ADD Sumand 1, Sumand 2

Sumand 1 és un registre de la EU.
Sumand 2 és qualsevol dels anteriors 4 casos.

→ El valor de la Suma queda enregistrat a Sumand 1
→ Si es genera Carry, el Flag CF es posa a 1.

E.g.
```
MOV AX, 0H     ; posam |00000000|00000000| dins AX
MOV AL  FFH    ; posam 11111111 dins AL
ADD AL  1H     ; sumam 00000001 a AL
```

dins CF s'ha posat un 1
dins AX tenim |00000001|00000000|
AH AL

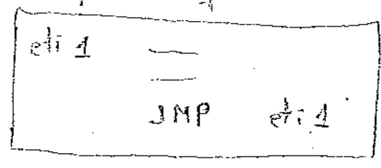
01D8 ADD → |0000000111011000| Interpretable pels processadors Intel per a PC.

NOT Operand | S'inverteixen els bits del registre o posició de memòria dins el mateix lloc

E.g.
```
MOV BL, FH   ; tenim 00001111 dins BL
NOT BL       ; tenim 11110000 dins BL
```

JMP etiqueta | JMP és "Jump", "salta"

S'interromp la seqüència que marca IP, (execució seqüencial) i es "salta" a la posició que indica el valor etiqueta.



CMP OP1, OP2 (Destí, Font) | Es comparen 2 valors de la mateixa longitud. Segons el resultat, els flags ZF i CF queden així.

	ZF	CF
Destí < Font	0	1
Destí = Font	1	0
Destí > Font	0	0

E.g.
```
MOV AX, 0H
MOV BX, 1H
CMP AX, BX
```
(Dins ZF queda 0, dins CF = 1). Destí < Font

Salts condicionals

Amb etiquetes, segons valors de Flags.

- JZ etiqueta
- JNZ etiqueta

} segons ZF = 0 o 1

NO → JB etiqueta | si CF = 1. (→ Destí ha resultat < que Font.)

NO → JA etiqueta | si CF=0, ZF=0. (→ destí ha resultat > que Font)

(N'hi ha més, segons altres Flags)

NO → JE etiqueta | JZ=1 (destí = font)

```
        MOV AX, 0H
        MOV BX, FH
seguir  ADD AX, 1H
        CMP BX, AX
        JNZ seguir   ; Sumam 1 a AX fins que arriba a valer F.
        -
        -
        -
```

Exemples de programes.

Programa Resta.

→ Sabem que per restar: Minvend - Substrahend, feiem complement a 2 del substrahend, i sumàvem. El C2 surt de invertir els seus bits (NOT) i sumar-li 1.

```
  24   →    1 1 0 0 0  }→  1 1 0 0 0 }  1 1 0 0 0
-  7      - 0 0 1 1 1  }   1 1 0 0 0 }  1 1 0 0 1
----                                   ----------
  17                                  1 1 0 0 0 1
                                      ↙ └───────┘
                                   Carry    17
```

Aleshores:

```
MOV AX, Minvend        ; posició identificada dins el DS.
MOV BX, Substrahend    ; posició identificada dins el DS.
NOT BX
ADD BX, 1              ;
ADD AX, BX             ; dins AX queda el valor Minvend - Substrahend
```

Programa Producte.

Tendrem factor1 i factor2, sumarem factor1 factor2 vegades.

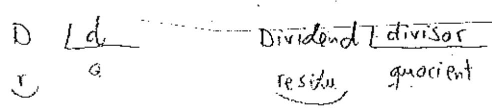
```
        MOV AX, Factor1     ; Comprovar si AX=0, si BX=0 (*)
        MOV BX, Factor2
        MOV CX, AX          ; per no perdre factor1 si modificam AX. Dins CX quedarà Factor1*Factor2
        MOV DX, 1           ; Valor inicial del numero de Sumes
        CMP BX, DX          ; si Factor2=1, aleshores ja hem acabat.
        JNZ final           ; ZF=1 => són iguals => hem acabat.
seguir  ADD CX, AX          ; Sumam una vegada AX
        ADD DX, 1           ; Sumam 1 al nº de sumes
        CMP DX, BX          ; comparam si el numero de sumes = factor2
        JZ  seguir
Final   ~~~                 ; el registre CX conté Factor1 * Factor2
```

Exercici: → (*)

```
           CMP CX, 0
           JNZ tenimzero
           CMP BX, 0
           JNZ tenimzero
           JMP nofactorzero
tenimzero  MOV CX, 0
           JMP Final
```

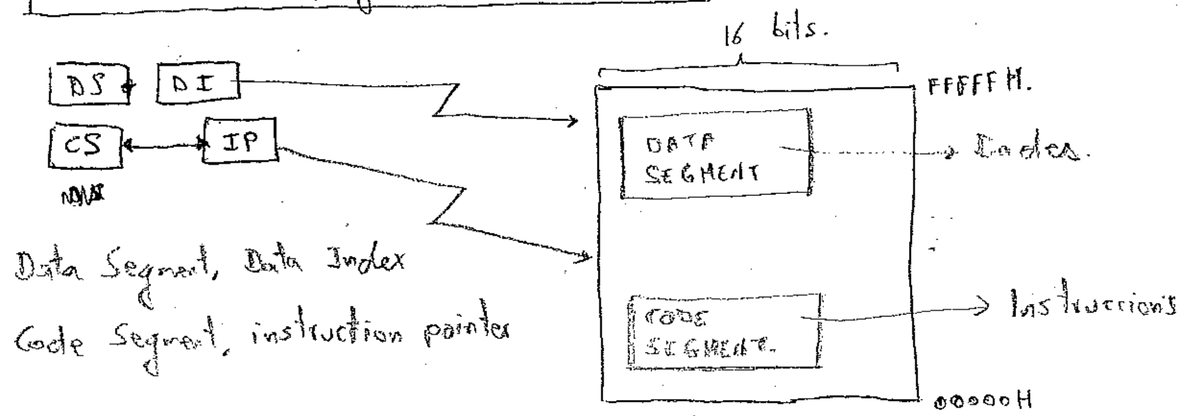
Exercici:

Programa Divisió.




```
        MOV AX, divisor    ; li restarem successius BX, i aquí quedarà el residu
        MOV BX, dividend
        MOV CX, 0H         ; aquí queda el quocient, li sumarem 1 per cada resta
        CMP BX, 0
        JNZ Final
Seguir  CMP AX, BX
        JB  Final
        MOV DX, BX         ; el complement  }  es pot calcular una sola
        NOT DX             ; a 2 es guarda  }  vegada, abans del CMP marcat
        ADD DX, 1H         ; dins DX        }  per Seguir
        ADD AX, DX
        ADD CX, 1H         ; sumem 1 al quocient
        JMP Seguir
Final   ~~~~~~             ; dins AX tenim el residu dins CX tenim el quocient.
```

Execució d'un programa en memòria.



* Un programa és una seqüència d'instruccions en codi màquina ubicades a posicions contigües de RAM dins el Code Segment.
* Cada vegada que s'executa una instrucció, IP incrementa el seu valor en 1, 2, 3 (mecanisme síncron), a no ser que s'executi una instrucció que alteri el fluxe seqüencial d'execució: JMP, JZ, JNZ ...

(segons el nº de bytes de l'instrucció)

Càlcul d'adreces. Segmentació

→ Si manipulam dades de 16 bits, com a màxim podem direccionar $2^{16} = 65.536$ posicions. $\frac{65.536}{1.024} = 64$ K bytes.

→ Aquest és el tamany màxim d'un segment.

→ Si tenim un espai direccionable de, p.e. 1 Mb., necessitam adreces més llargues, de 20 bits.

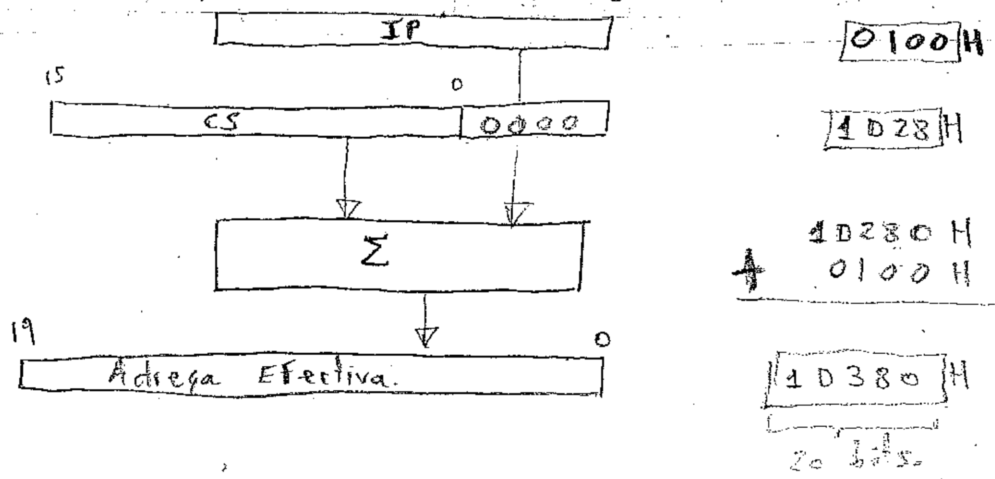
$2^{20} = 1.048.576$ bytes $= 1.024$ K bytes $= 1$ Mb.

16 bits = 4 digits hexa
20 bits = 5 digits hexa. → Posició { màxima = FFFFF H. ; mínima = 00000 H }

→ Dins CS tenim 16 bits, que indiquen l'inici del segment de codi. L'adreça real d'aquest inici és amb 4 zeros més:

→ Dins IP tenim 16 bits.

La forma d'obtenir una adreça d'instrucció és:



Interrupcions

Una interrupció és un esdeviment asincron, (no apareix en un moment determinat per la senyal CK, sinò en qualsevol moment) que reclama l'atenció del μP.

Exemples de interrupcions:

- Un perifèric realitza una operació d'input (Es polsa una tecla, i s'ha de guardar un byte a la RAM). Ctrl. Alt. Del
- Un programa reclama l'atenció d'un perifèric per a output.
  El software li diu al processador "escriu això a pantalla" "a disc"
  Instrucció INT de l'ASM, amb diferents codis de Funció. Exemple programa "escriure"
- Aparició d'una condició d'error, o especial
  - → Divisió per zero, causada pel software. (variables)
  - → Mode "Trace" (debug)
  - → Segment Overrun (hem escrit dins DS més del q. hi cap, probl. típic dels progrs. que manipulen molta info, video...).

...

Funcionament de la pila.

→ Cada programa en execució ("procés") té un Stack Segment. Si el processador deixa de fer-li cas durant un temps per passar a fer cas a un altre procés, els valors guardats

dins els registres CS, DS ... AX, BX ... IP, valor dels flags ...
s'han de "conservar" en algun lloc. Aquesta és la funció de
l'Stack Segment, on es guarden els regs del MP ordenadament.

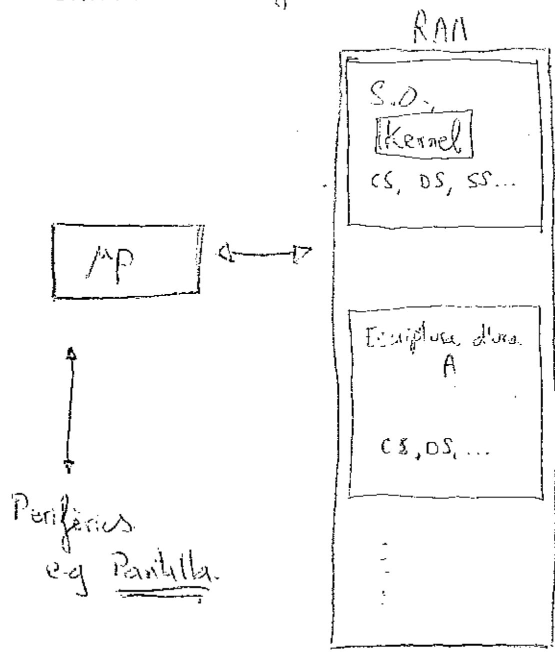
→ Manipulació manual de la pila: dues instruccions:

[PUSH [valor],] valor = registre, etiqueta del DS, valor literal.
Col·loca [valor] dins la pila.

[POP [destí]] extreu el que la pila apunta dins destí = registre, pos de RAM.

Posició RAM de la pila:
SS : SP. Stack Pointer : Stack Segment

⊛ Quan s'executa una interrupció, pot haver-hi una transferència de control, (passam d'un procés a un altre), i les piles respectives de cada procés entren en joc



Estat inicial.

1er) El S.O. és el programa "actiu". En el MP estan carregats les adreces del S.O. CS, DS, ..

2on) El S.O., quan se li demana que executi un programa, el busca en el disc, el col·loca a la RAM i li transfereix l'ús del MP.
Aquest programa, el 1er que fa és salvaguardar dins la seva pila les adreces RAM del DS : 0000 q. està utilitzant el S.O.

```
PUSH DS
sub AX, AX
PUSH AX
```
→ (Gestió manual de la pila)

I després carregar el seu DS :

```
MOV AX, DADES
MOV DS, AX
```

3er) Si el procés no necessita ajut de ningú, manipula registres i RAM i tot va bé.
Si el procés necessita manipular, per exemple, un dispositiu extern (pantalla), li dona el control temporalment al SO.
Això es fa amb la instrucció INT.

En l'exemple:

```
MOV AX, 2H ;   funció "escriu en pantalla"
MOV DL, lletra ;   què has d'escriure
INT 21h;   el kernel "desallotja" del MP al programa, l'ocupa,
```
[illegible]

- La instrucció que transferia el control al S.O. és la 21H

Altres són:
- 13H → accés a disquettes → identificació de sectors, nº de cap/pl, quantitat de bits a llegir/escriure...
- 25H → llegir del HD.
- 26H → escriure en el HD.
- 27H → TSR,
- 16H → Funcions extres de teclat.
- ⋮

} Algunes actuen sobre els circuits de la BIOS.

- Un tipus especial d'interrupció: DMA = Direct Memory Access.

→ És una tècnica que consisteix en transvassar grans quantitats de dades entre RAM i perifèric, sense que el μP se n'hagi de fer càrrec. Simplement, el perifèric avisa de "ara comença", "ara acaba" la transferència. Cada perifèric té un identificador numèric de DMA.

→ En el cas d'escriure A que coneixem, el codi ASCII de 'A' havia de escriure's en el μP. Amb DMA, poden enviar conjunts complets de dades al dispositiu de sortida sense que passin pel μP

→ Típicament, doncs, un perifèric té
- una línia de interrupció: IRQ
- un codi numèric DMA. línia DMA.
- un soft del S.O. que hi accedeix: driver.

→ Amb sistemes operatius antics, instal·lar un dispositiu nou (targeta de so, 2n disc, etc...) volia dir:
- a) Connexions físiques.
- b) Instal·lació al HD del programa driver.
- c) Comunicació al SO dels codis IRQ, DMA del dispositiu.

→ El SO Windows actual pretén fer b) i c) automàticament, en el procés conegut com Plug & Play. El driver es troba en el CD del propi SO, o en el q. acompanya al dispositiu nou. IRQ i DMA s'assignen segons les IRQ, DMA ja assignades.

Es poden veure a: Panel de Control / Sistema / Administrador de dispositivos...

Microprogramació

- Hi ha instruccions ensamblador que poden no existir, però que serien molt útils si existissin. Per exemple, SUB. AX, BX. Cada vegada que hem de restar sabem que podem executar

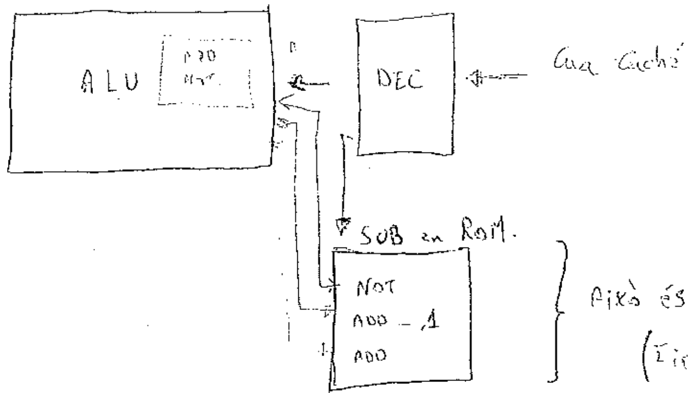
```
MOV AX, Minuend
MOV BX, Substrahend
NOT BX
ADD BX,1
ADD AX,BX
```

} 3 instruccions q. executen la resta efectiva.

- Altres maneres de fer la resta: a) Via programa ASM, com ja hem fet.

b) Construir, dins l'ALU, un circuit que l'executi, per qualsevol dels mètodes possibles: a) Complement i suma b) Unitat Prestada, c) Altres.

c) Aprofitar les instruccions NOT i ADD ja construïdes a l'ALU, construir en ROM la seqüència que les executa i passar-li un codi d'instrucció màquina reconeixible:



En un processador, la quantitat d'instruccions que s'implementin directament a la ALU, o s'hagin de deixar fora, és important.

Processador RISC: Reduced Instruction Set Computer. Disposa de no moltes instruccions en codi màquina. Si és així, el seu codi binari és més curt, s'han de construir DECs més petits, i arribem a elles travessant menys portes lògiques. Aleshores, són molt ràpids. Probl: necessiten moltes coses en microprogrames. Es pot tenir un coprocessador q. amplii el IS, (Instruction Set)

Processador CISC: Complex Instruction Set Computers. És el cas contrari: més instruccions disponibles a l'ALU, però totes més lentes. Pentium és CISC.

Velocitat dels processadors.

Mhz → idea de "força bruta", nº de cicles per segon, traduïble a nsegons q. es tarda en fer coses sentilles. E.g. recuperar una dada de memòria. Però, quan es tarda en "fer algo amb ella"?

MIPS → Millions d'instruccions per segon, aquí ja es veu la diferència entre una màquina RISC i una màquina CISC. Problema, no totes les instruccions són "igualment sofisticades". SUB és més complexa que ADD, sigui quin sigui el mètode amb que s'han implementat.

MegaFLOPS Milions de Floating Point Operations per Second. És el mateix concepte que els MIPS, però restringint la mesura a operacions aritmètiques que operin sobre numeros en Coma Flotant.

El Sistema Operatiu.

N'hem parlat varies vegades, però ara en donarem una visió més rigorosa:

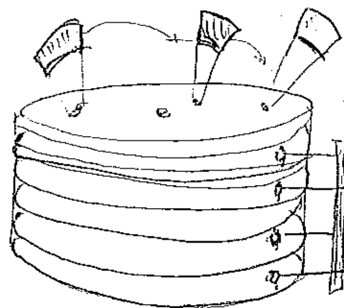
* El q. més coneixem és Windows, que és el q. fem servir com usuaris. Windows és una evolució del DOS, (Disk Operating System) de MicroSoft, MS DOS.

Què Fa? (2 coses bàsiques.) * **Funció bàsica:** traduir l'esquema de compartimentació dels discs floppys i discs durs en algo que puguem manipular: una estructura de fitxers

Taula de descripció de fitxers:

NOM	SECTOR INICIAL	altres
———	Posició en el S.	...
Longitud fitxer		

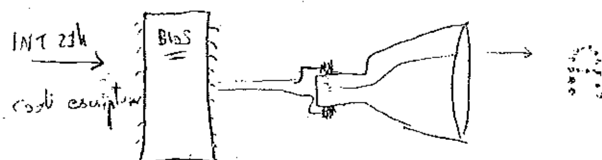
Així és la FAT = File Access Table



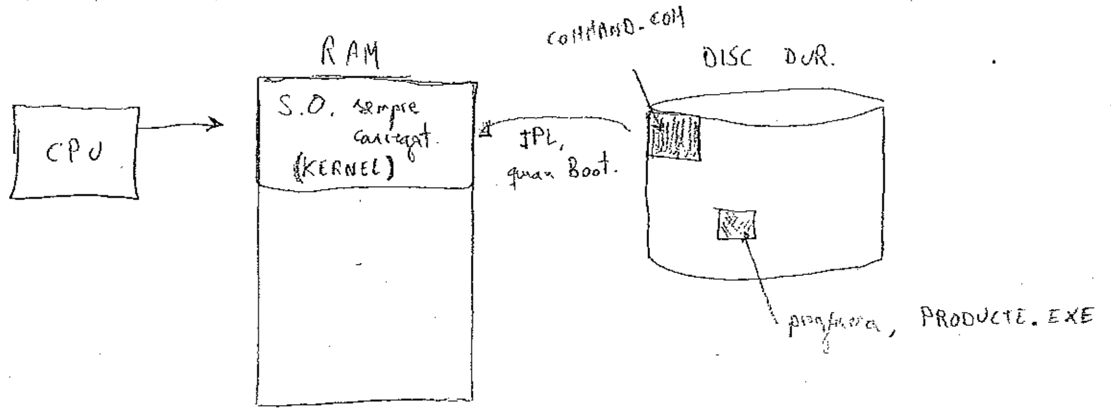
Sectors, o Clusters

Tamany mínim d'un fitxer.

* **Altra Funció bàsica:** Gestió de perifèrics, mitjançant IRQ, DMA, drivers i circuits implementats en xips específics: (BIOS. Basic Input/Output System).

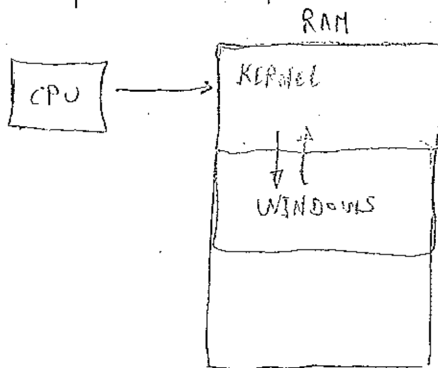


⊛ Ja hem vist que DOS treballa a base de rebre ordres escrites i executar-les.

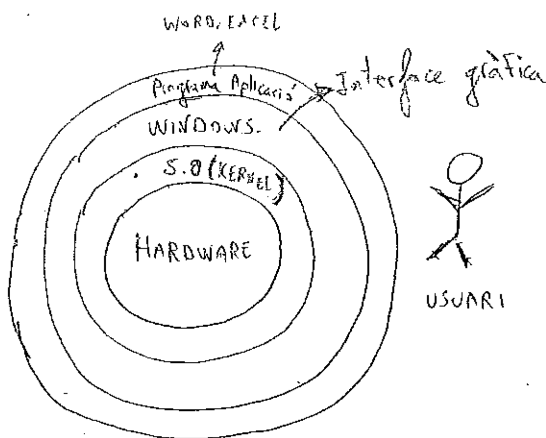


Kernel entén 2 tipus d'ordres:
- directament executables: DIR, TIME, CLS.
- ordres que ha d'anar a buscar al disc: execució de programes, que es traslladen a la RAM.

⊛ Windows va començar sent un "programa extra" que parlava amb el Kernel per un costat i per l'altre podia rebre les ordres de l'usuari en un format "més amistós"

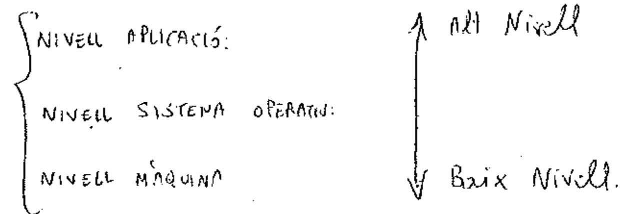


Això també es pot simbolitzar així:

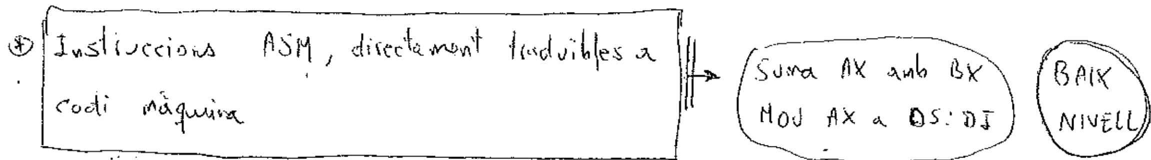


- Sense Windows: programes aplicatius sense interface gràfica. Exemple: EDIT.

Parlem de:



(*) En quant a llenguatges de programació, per a "construir fitxers .EXE"



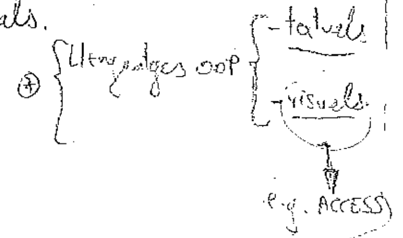
Assembler s'utilitza per: - Sistemes Operatius
- Drivers ...

(*) Instruccions Pascal, BASIC, C, ...
Ens permet plantejar problemes més "humans", i oblidar-nos una mica del hardware que hi ha baix.

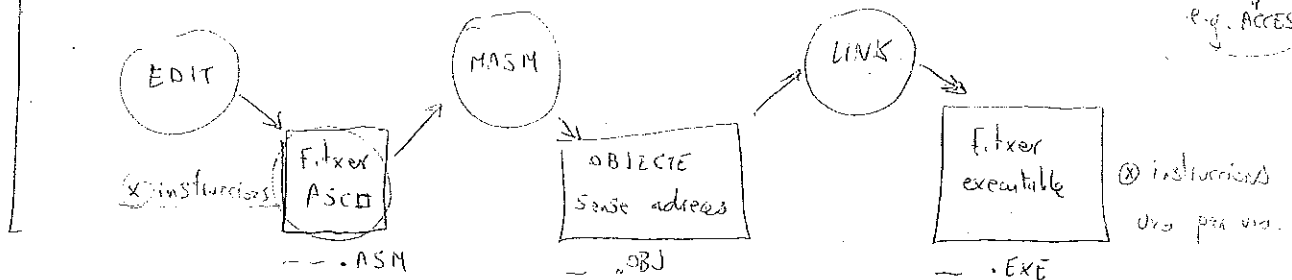
→ - Repeteix algo mentres no tal condició.
- Escriu això a pantalla

ALT NIVELL

Llenguatges q. s'utilitzen per a problemes més generals.



Ja hem vist que: en Assembler.



En Pascal veurem que:

En un mateix programa tenim:

- Editor ASCII, que permet obrir, guardar, guardar com.
- Traducció de les instruccions Pascal a .EXE directament, incloguent el pas de Linkatge, i generació d'un fitxer (.exe)

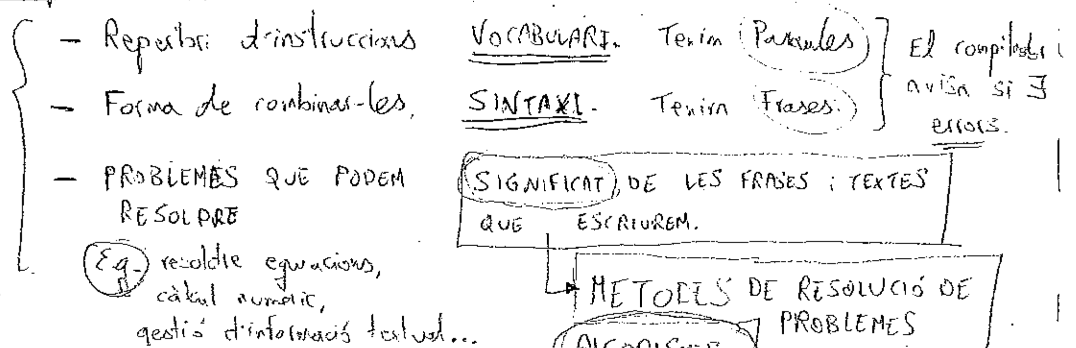
Aquest procés es diu compilació.

En un llenguatge disposam de:

- Repertori d'instruccions VOCABULARI. Tenim (Paraules)
- Forma de combinar-les, SINTAXI. Tenim (Frases)

El compilador avisa si ∃ errors.

- PROBLEMES QUE PODEM RESOLDRE
(Eg.) resoldre equacions, càlcul numèric, gestió d'informació textual...



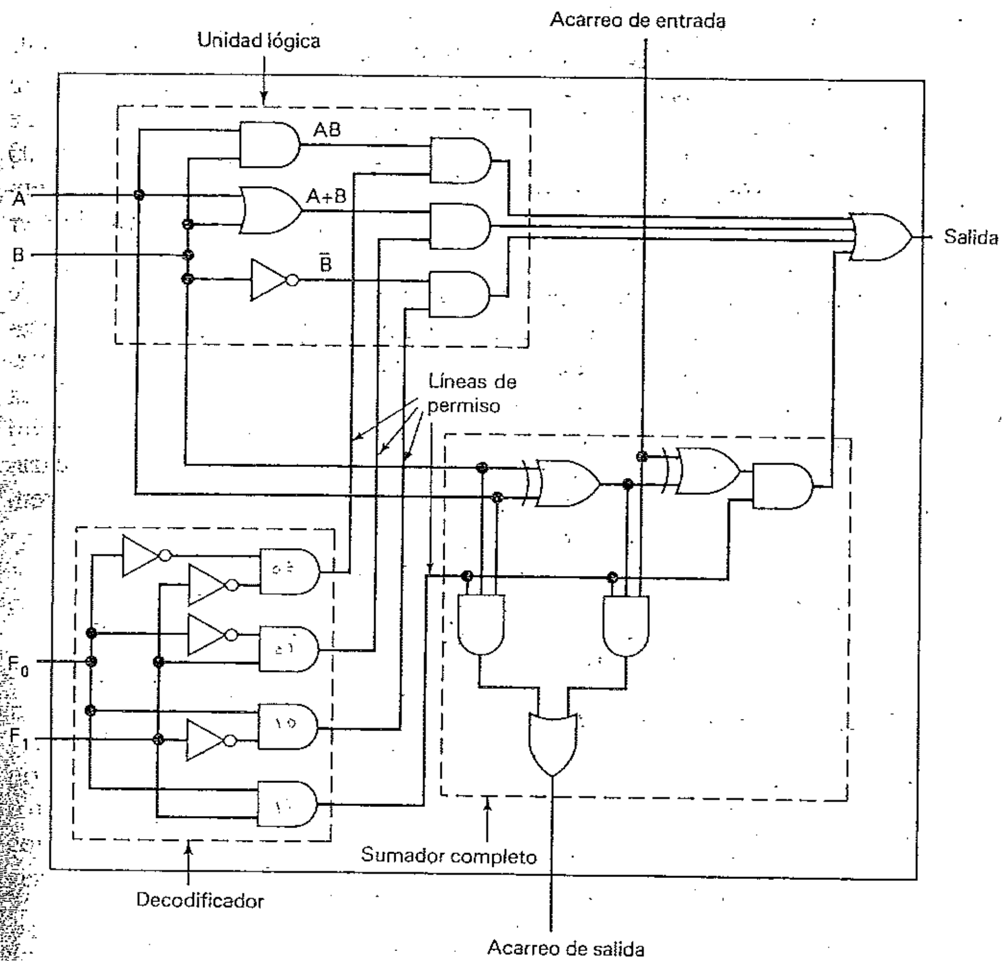


Fig. 3-20. ALU de un bit.

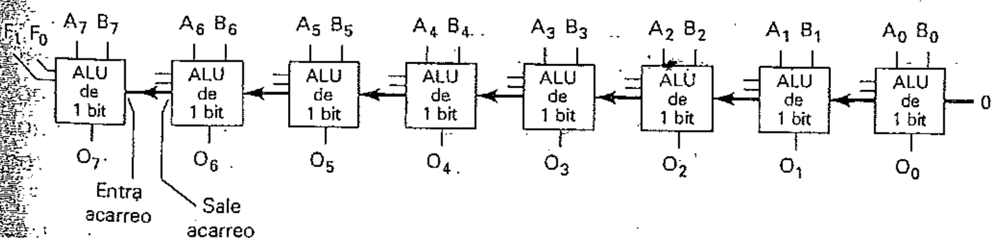
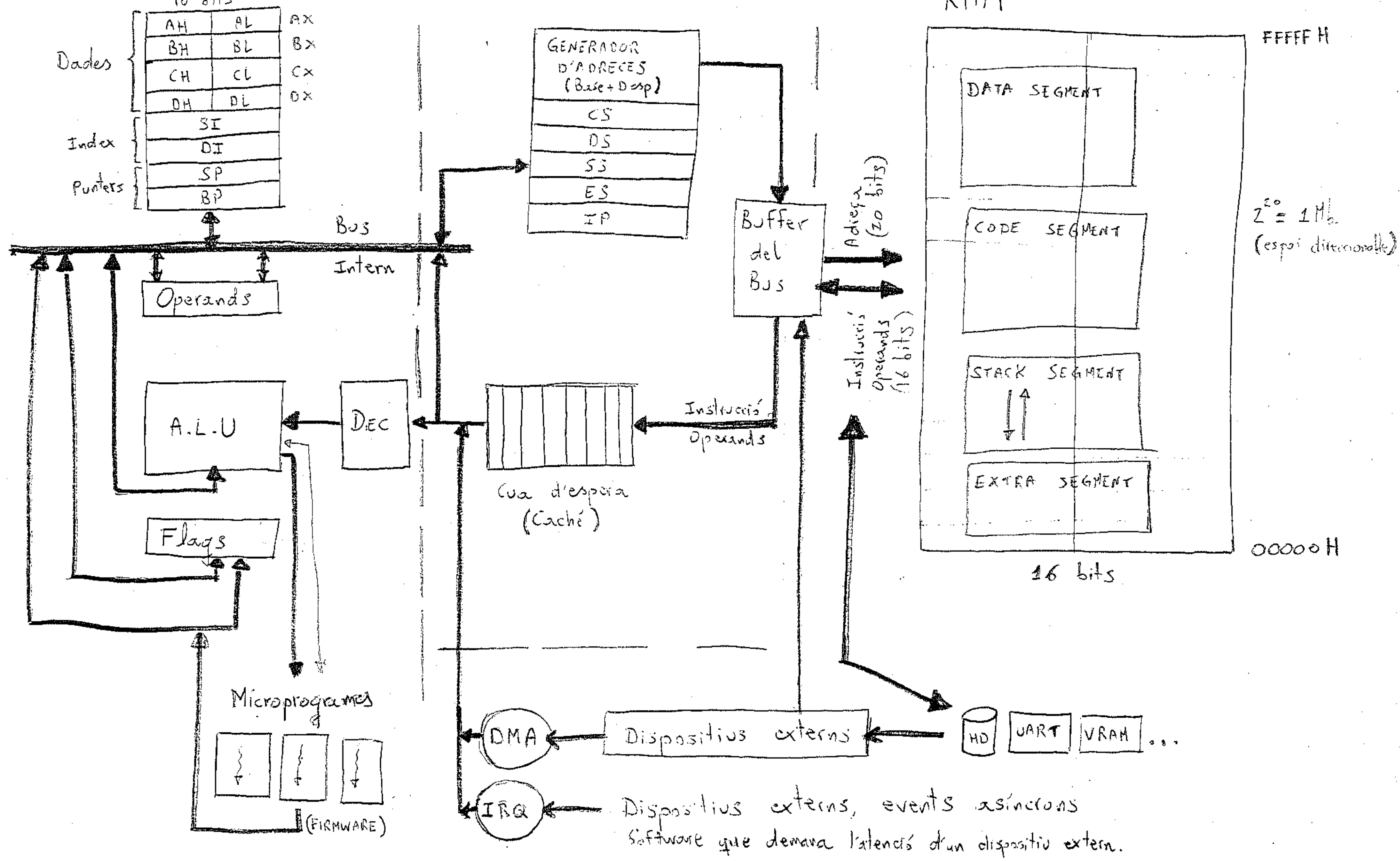


Fig. 3-21. Ocho ALU de 1 bit conectadas para formar una ALU de 8 bits.


ARQUITECTURA INTERNA DEL µP 8086.
EU = Execution Unit
BIU = Bus Interface Unit
16 bits
AH
AL
AX
BH
BL
BX
CH
CL
CX
DH
DL
DX
Dades
SI
DI
Index
SP
BP
Punters
Bus
Intern
Operands
A.L.U
Dec
Flags
Microprogrames
(FIRMWARE)
GENERADOR D'ADRECES (Base + Desp)
CS
DS
SS
ES
IP
Buffer del Bus
Adreça (20 bits)
Instrucció Operands (16 bits)
Instrucció Operands
Cua d'espera (Caché)
DMA
IRQ
Dispositius externs
HD
UART
VRAM
Dispositius externs, events asíncrons
Software que demana l'atenció d'un dispositiu extern.
RAM
FFFFF H
DATA SEGMENT
CODE SEGMENT
STACK SEGMENT
EXTRA SEGMENT
00000 H
16 bits
(espai direccionable)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                   ;;
;;        Producte de dos factors    ;;
;;                                   ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DADES  SEGMENT PARA PUBLIC 'DADES'

factor1        dw 7h      ; factor1 en hexadecimal
factor2        dw 7h      ; factor1 en hexadecimal

DADES  ENDS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

PILA   SEGMENT PARA STACK 'PILA'

          DB  128 DUP('P')

PILA   ENDS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CODI   SEGMENT PARA PUBLIC 'CODI'

INICI     proc far ;;;;;;;;;;;;;;;;INICIALITZACIONS;;;;;;;;;;;;;;;;;;;;;

          assume cs:CODI,ds:DADES,ss:PILA

          push ds                  ;PER A TORNAR AL DOS
          sub ax,ax                ;GUARDA   DS:00  A LA PILA
          push ax

          mov ax,DADES             ;CARRREGA EL SEG. DE DADES
          mov ds,ax

          call CALCULA              ;CRIDA EL PROCEDIMENT DE CODI

          ret
INICI     endp;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CALCULA PROC;;;;;producte de dos factors;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        MOV AX, factor1    ;
        MOV BX, factor2    ;
        MOV CX, AX        ; per no perdre factor1, aqu¡ queda el resultat
        MOV DX, 1h        ; valor inicial del n§ de sumes
        CMP BX, DX        ; si factor2=1 aleshores ja hem acabat
        JZ final         ; si ZF=1 => comparacio d'abans ok => acabam!
seguir: ADD CX, AX        ; suma del factor1 una vegada
        ADD DX, 1h        ; increment del comptador resultat
        CMP DX, BX        ; comparam si el n§ de sumes = factor2
        JNZ seguir         ; si ZF=0, no son iguals, hem de  seguir!
final:  RET               ;

CALCULA ENDP;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

CODI   ENDS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

       END    INICI
```