

# Representació de la informació

## TIPUS D'INFORMACIÓ:

- Numèrica
- Text
- Imatges
- Sons

**CONTEXT:** aquella informació que li dona sentit a la informació.

**BASE:** és la quantitat de grafismes diferents distingibles individualment que disposes per crear un sistema de numeració.

*Exemple: guarismes base 10 {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, base 3 {0, 1, 2}*

Normalment quan es passa de base 10 s'utilitzen les lletres per representar més guarismes.

*Base 3: {000, 001, 002, 010, 011, 012, 020, 021, 022, 100, 101, 102 ...}*

**CAPACITAT DE REPRESENTACIÓ:** quantitat de valors diferents que pots escriure amb aquell nombre de xifres.

$$\text{Capacitat} = b^n$$

*Exemple: base 3 amb 3 xifres  $3^3=27$  podem aconseguir 27 xifres diferents*

Si b és petita necessitem més xifres per representar un valor concret.

## RANG:

$$0 \leq x \leq b^n - 1$$

## CANVI DE BASE

Representació posicional ponderada: és rellevant la posició de les xifres, no totes els xifres tenen el mateix valor. (centenes, desenes i unitats)

$$X = \sum_{i=0}^{n-1} a_i b^i$$

$$X = a_{n-1}b^{n-1} + \dots + a_1b^1 + a_0b^0$$

Exemples:

$$253_{10} = 2 \cdot 10^2 + 5 \cdot 10^1 + 3 \cdot 10^0 \rightarrow \text{base 10}$$

$$253_{16} = 2 \cdot 6^2 + 5 \cdot 6^1 + 3 \cdot 6^0 = 72 + 30 + 3 = 105 \rightarrow \text{passem de la base 6 a la 10}$$

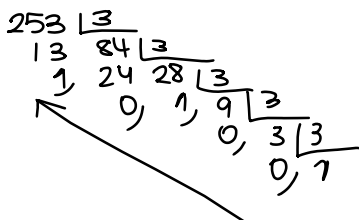
Passem de base 10 a 3:

$$253_{10} = 2 \cdot 10^2 + 12 \cdot 10^1 + 10_{10}$$

Com que la destinació es la base 3 utilitzem la base 3 quan sumem. En base 3:

$$2_{10} = 2_{16}, 5_{10} = 12_{16}, 3_{10} = 10_{16}$$

$$100101_{13} = 253_{10}$$



$$\frac{x}{d} = Q \cdot \frac{R}{d} \rightarrow X = Q \cdot d + R$$

$$X = a_{n-1}b^{n-1} + \dots + a_1b^1 + a_0b^0 = b(a_{n-1}b^{n-2} + \dots + a_1) + a_0$$

3                      Q                      R

## Sistema binari

La base binària dona una representació fàcil de la lògica booleana. Quan més grafismes hi ha (més gran és la base) més difícil és distingir els diferents grafismes i per tant s'ha de gastar més energia en distingir-los per això s'utilitza el binari.

La base de representació òptima és el número 2 (entre 2 i 3) però es va optar per 2 per representar la lògica booleana (sí-no, veritat-fals, blanc-negre).

## Sistema octal (base 8)

$$0_8 = 000_2$$

$$1_8 = 001_2$$

$$2_8 = 010_2$$

$$3_8 = 011_2$$

$$4_8 = 100_2$$

$$5_8 = 101_2$$

$$6_8 = 110_2$$

$$7_8 = 111_2$$

$$253_{10} = 010\ 101\ 011_2$$

L'octal és una forma compacte d'expressar el binari, s'agrupa en grups de 3

$$11000111101000_2 = 30750_8$$

## Sistema hexadecimal (base 16)

$$0_{16} = 0000_2$$

$$1_{16} = 0001_2$$

$$2_{16} = 0010_2$$

$$3_{16} = 0011_2$$

$$4_{16} = 0100_2$$

$$5_{16} = 0101_2$$

$$6_{16} = 0110_2$$

$$7_{16} = 0111_2$$

$$8_{16} = 1000_2$$

$$9_{16} = 1001_2$$

$$A_{16} = 1010_2$$

$$B_{16} = 1011_2$$

$$C_{16} = 1100_2$$

$$D_{16} = 1101_2$$

$$E_{16} = 1110_2$$

$$F_{16} = 1111_2$$

Serveix per compactar, igual que l'octal però agrupem d'en quatre en quatre.

Exemple: 0010 0101 0011<sub>2</sub> = 253<sub>16</sub>

11 0001 1110 1000<sub>2</sub> = 31E8<sub>16</sub>

## FRACCIONARIS

Si tenim un nombre fraccionari → 1011,01101<sub>2</sub>

Fariem un sumatori igual:

$$2^3 + 2^1 + 2^0 + 2^{-2} + 2^{-3} + 2^{-5} = 8 + 2 + 1 + \frac{1}{4} + \frac{1}{8} + \frac{1}{32} = 11 + \frac{(8 + 4 + 1)}{32} = 11 + \frac{13}{32}$$

Fem ara el camí contrari (de decimal a binari):

65,32<sub>10</sub> → X<sub>2</sub>

$$\begin{array}{r} 65 \overline{) 128} \\ 1, 32 \overline{) 64} \\ 0, 16 \overline{) 32} \\ 0, 8 \overline{) 16} \\ 0, 4 \overline{) 8} \\ 0, 2 \overline{) 4} \\ 0, 1 \end{array}$$

0,32<sub>10</sub> → X<sub>2</sub>

$$0,32 \cdot 2 = 0,64$$

$$0,64 \cdot 2 = 1,28$$

$$0,28 \cdot 2 = 0,56$$

$$0,56 \cdot 2 = 1,12$$

$$0,12 \cdot 2 = 0,24$$

$$0,24 \cdot 2 = 0,48$$

$$0,48 \cdot 2 = 0,96$$

65,32<sub>10</sub> → 1000001,01010001

$$a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \dots + a_1b^1 + a_0b^0$$

$xb = a_{n-1}b^n + a_{n-2}b^{n-1} + \dots + a_1b^2 + a_0b^1 \rightarrow$  Multipliquem per la base

EXEMPLE (ho fem per proximitat):

243 → La potència de dos més pròxima és  $2^7$  a la seva meitat per tant ocuparà la posició 8. Restem  $2^7$

$$\begin{array}{r} 243 \\ - 128 \\ \hline 115 \\ - 64 \\ \hline 51 \\ - 32 \\ \hline 19 \\ - 16 \\ \hline 3 \\ - 2 \\ \hline 1 \end{array} \quad \begin{array}{l} 2^7 \\ 2^6 \\ 2^5 \\ 2^4 \\ 2^1 \\ 2^0 \end{array}$$

Les potències de 2 ens indiquen les posicions que ocuparan els uns

243<sub>10</sub> = 11110011<sub>2</sub>

# ENTERS

En un computador només hi ha zeros i uns per això per representar els signes fem:

Positiu  $\rightarrow 0$

Negatiu  $\rightarrow 1$

T'han d'indicar si forma part del nombre o t'estan indicant el signe (és important el context), ja que no hi ha manera de diferenciar-ho.

$\left. \begin{array}{l} 001101 \\ 101101 \end{array} \right\}$  El context ens dirà si és el mateix nombre amb diferent signe o dos nombres completament diferents

Tenim tres maneres de representar un nombre amb signe:

- o Signe i magnitud
- o Complement a la base disminuïda
- o Complement a la base

## Signe i magnitud

S	Magnitud
---	----------

$+3_{10} \rightarrow 0011_{12}$  (4 bits)

$-7_{10} \rightarrow 10000111_{12}$  (8 bits)

La suma amb signe i magnitud és poc pràctica

## Complement a base disminuïda (complement a 1)

- o Si és positiu  $\rightarrow$  la representació és la mateixa que amb signe magnitud
- o Si és negatiu  $\rightarrow X = (2^n - 1) - |X|$  n (nombre de xifres)

EXEMPLE: Treballem amb 8 bits

Calculem primer per separat:

$$\left. \begin{array}{r} 2^8 \rightarrow 100000000 \\ - 1 \\ \hline 011111111 \end{array} \right\} (2^n - 1)$$

$$\left. \begin{array}{l} |-23| = 23 \\ 16 = 2^4 \quad 2 = 2^1 \\ 4 = 2^2 \quad 1 = 2^0 \end{array} \right\} |X| = |-23| = 23_{10} = 00010111_{12}$$

$$\left. \begin{array}{r} 11111111 \\ - 00010111 \\ \hline 11101000 \end{array} \right\} -23_{10} = 11101000_{12}$$

Observem que únicament s'ha d'invertir bit per bit  $\rightarrow \overline{X_1} + \overline{X_2} + \dots$

Provem de fer ara operacions:

$$\begin{array}{r}
 -23 \\
 +10 \\
 \hline
 -13
 \end{array}
 \quad
 \begin{array}{r}
 11101000 \\
 + 00001010 \\
 \hline
 11110101
 \end{array}
 \rightarrow \text{Està a CI ho passem a signe i magnitud}$$

$$|00001101| = 13$$

EXEMPLE (8 bits):

$$-14 \rightarrow |14| = 00001110$$

$$-45 \rightarrow |45| = 00101101$$

$$\begin{array}{r}
 -14 \quad 11110001 \\
 -45 \rightarrow 11010010 \\
 \hline
 -59 \quad 11000011 = X
 \end{array}
 \quad
 C \text{ (Carry)} = 1 \text{ (en portem una)}$$

$|X|$  (desfem el complement a 1)

$$|X| = 00111110 = 60 \rightarrow X = -60$$

Veiem que no ens dona exacte si desfem l'operació, a complement a 1 s'ha de corregir. S'ha de sumar sempre el carry.

$$\begin{array}{r}
 11000011 \\
 + C \\
 \hline
 11000100 \rightarrow -59
 \end{array}$$

És poc pràctic perquè ha de fer dos cops l'operació. Per tant recorrem al complement a 2 (C2):

## Complement a base (complement a 2)

- Si és positiu = Signe Magnitud (SM)
- Si és negatiu  $X = 2^n - |X|$

El C2 serà el mateix però més 1

EXEMPLE:

$$\begin{array}{r}
 -14 \rightarrow C1 (-14) = 11110001 \\
 \quad \quad \quad + 1 \\
 \hline
 \quad \quad \quad 11110010 = -14 \text{ (C2)}
 \end{array}$$

$$\begin{array}{r}
 -45 \rightarrow C1 (-45) = 11010010 \\
 \quad \quad \quad + 1 \\
 \hline
 \quad \quad \quad 11010011 = -45 \text{ (C2)}
 \end{array}$$

$$\begin{array}{r}
 -14 \quad 11110010 \\
 -45 \rightarrow 11010011 \\
 \hline
 -59 \quad 11000101 = X
 \end{array}
 \quad
 \text{(no es contempla el carry)}$$

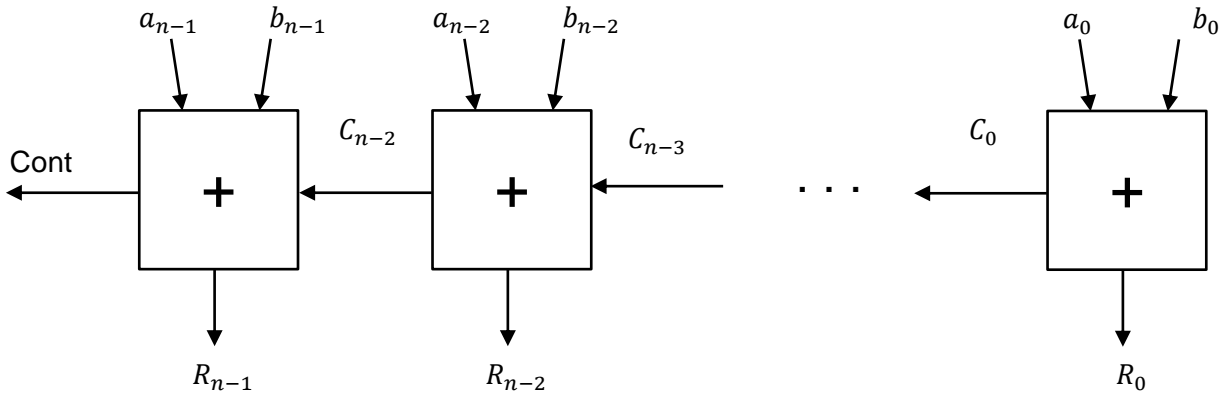
$|X|$  (desfem el C2)

- Resetem l  $\rightarrow$  11000100
- Invertim  $\rightarrow$  00111011 = 59

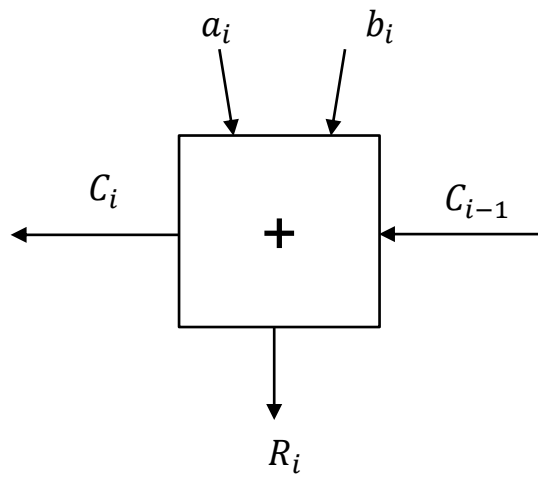
$$X = -59$$

Treballar a C1 és més lent.

Hardware necessari:



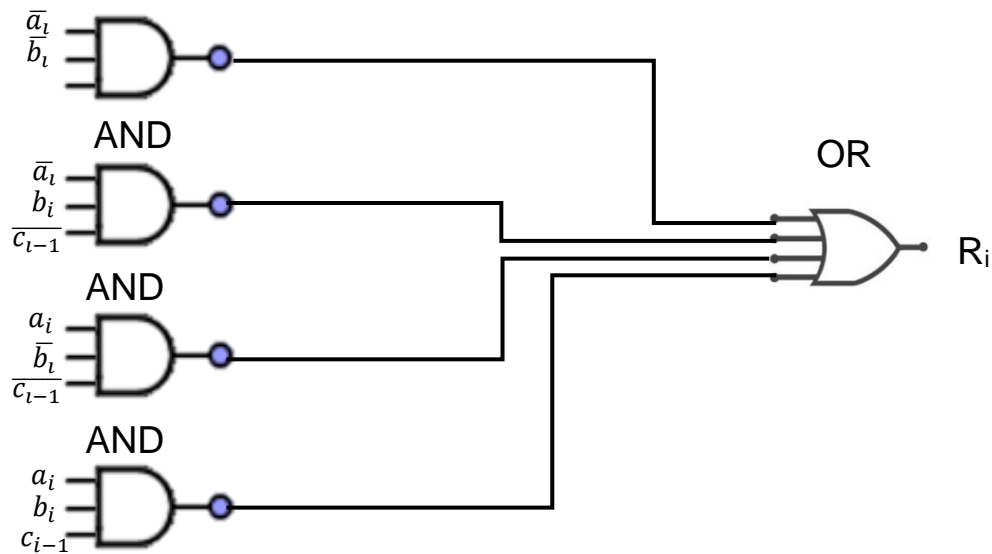
Cada bloc:



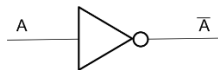
$a_i$	$b_i$	$c_{i-1}$	$R_i$	$C_i$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Per representar  $R_i$  agafem els valors que prenguin el valor 1 de la seva columna i neguem els zeros:

$$R_i = \bar{a}_i \bar{b}_i c_{i-1} + \bar{a}_i b_i c_{i-1} + a_i \bar{b}_i \bar{c}_{i-1} + a_i b_i c_{i-1}$$



Inversor:



## Overflow (desbordament)

$$\begin{array}{r} +73 \\ +61 \\ \hline 134 \end{array} \rightarrow \begin{array}{r} |73| = 01001001 \\ |61| = 00111101 \\ \hline 10000110 \end{array} \rightarrow \text{només tenim 7 bits i el signe}$$

Ens ha sortit un nombre negatiu com a suma de dos positius. Si representem més de 127 amb 7 bits passa això.

Obtenim un resultat erroni irrecuperable, hem d'afegir més bits i tornar a començar:

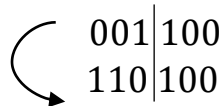
$$\begin{array}{r} 001001001 \\ 000111101 \\ \hline 010000110 \end{array}$$

Només passa si tenen el mateix signe, si són contraris no passarà

$$\begin{array}{r} -73 \quad 10110111 \\ -61 \rightarrow 11000011 \\ \hline -134 \quad 01111010 \rightarrow \text{overflow} \end{array}$$

Per passar a complement a 2 tenim dos formes:

$$\begin{array}{r} -12 \rightarrow |12| = 001100 \\ 110011 \\ +1 \\ \hline 110100 \end{array}$$



Comencem per la dreta, respectem els zeros i a partir del primer 1 els canviem tots

Com que hem tingut overflow afegim 4 bits:

000010110111  
000011000011

En l'exemple superior hem perdut el signe, és important fer l'extensió del signe (el signe ha d'ocupar tots els nous bits).

111110110111  
111111000011    El signe ocupa tots els nous bits  
000000111101

EXEMPLE:

110011    a C2

$|X| = 001101 \rightarrow -13 = 8+4+1$

Podem donar significació numèrica a 110011

$-2^5+2^4+2^1+2^0 = 32+16+2+1 = -13 \rightarrow$  sumatori general incloent el signe

## Rang de representació

	SM	C1	C2
0000	+0	0	0
0001	+1	+1	+1
0010	+2	+2	+2
0011	+3	+3	+3
0100	+4	+4	+4
0101	+5	+5	+5
0110	+6	+6	+6
0111	+7	+7	+7
1000	0	-111 = -7	-8
1001	-1	-110 = -6	-111 = -7
1010	-2	-101 = -5	-110 = -6
1011	-3	-100 = -4	-101 = -5
1100	-4	-011 = -3	-100 = -4
1101	-5	-010 = -2	-011 = -3
1110	-6	-001 = -1	-010 = -2
1111	-7	0	-1



RANGS:

$$SM \rightarrow -(2^{n-1} - 1) \leq X \leq +(2^{n-1} - 1) \text{ i } 2 \text{ zeros}$$

$$C1 \rightarrow -(2^{n-1} - 1) \leq X \leq +(2^{n-1} - 1) \text{ i } 2 \text{ zeros}$$

$$C2 \rightarrow -2^{n-1} \leq X \leq +(2^{n-1} - 1) \text{ i } 1 \text{ zero}$$

BCD: Decimal Codificat en Binari

$$32,46 = 3,246 \cdot 10^1$$

$$0,\underbrace{1\dots\dots}_{\text{manísta}} \cdot 2^x$$

Com que el primer nombre després del zero sempre és 1, no es posa i així estalviem un bit.

La mantisa es representa amb signe i magnitud

Amb 32 bits:

Signe	exponent 8 bits (C2)	23 bits
-------	----------------------	---------

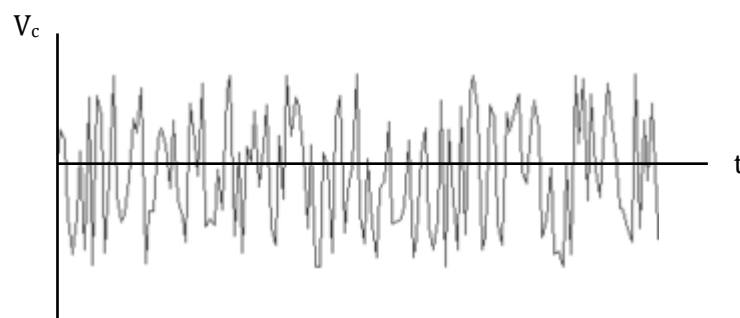
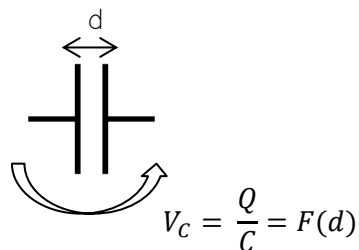
El número més gran que podem representar és  $10^{38}$  ( $2^{127} \rightarrow 10^x$ ), el més petit  $2^{-128}$

Amb 64 bits:

Signe	11 bits (C2)	52 bits
-------	--------------	---------

## SONS

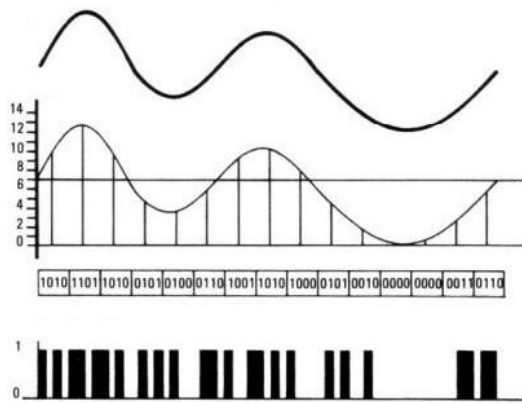
Un MICRÒFON és un condensador. Està format per una membrana que es modifica quan vibra l'aire i fa variar la tensió  $V_c$



Un ALTAVEU repeteix les ones, es tracta també d'un condensador.

Per passar a l'ordinador aquestes ones mostregem, agafem mostres de la representació. Quan més freqüent sigui el mostreig, més fidel a la realitat serà, però com a inconvenient haurem d'emmagatzemar més informació.

Transformem el so en nombres → digitalització



El tractament del so en digital és molt més fidel que el tractament en analògic ja que en aquest últim el so es veu més alterat



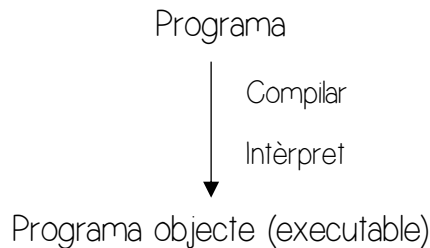
Cada píxel té una intensitat de llum i de color

Intensitat	Color
------------	-------

Depenent del nombre de bits que tingui el color tindrem una paleta més àmplia de colors o no.

# Computador

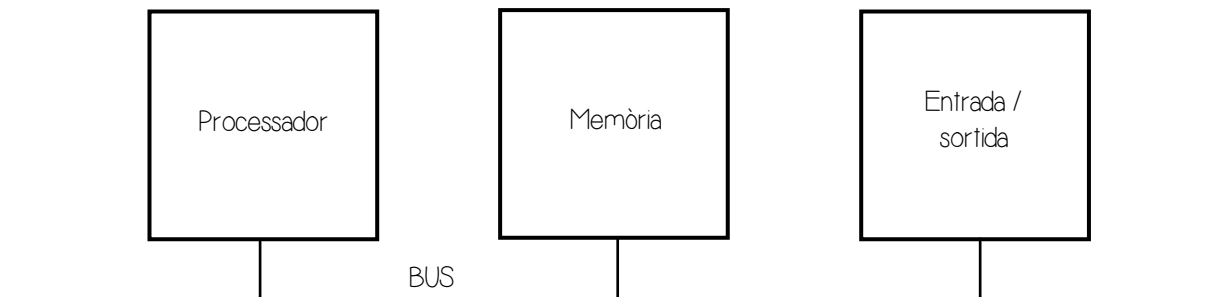
Un computador és una eina que serveix per processar i emmagatzemar informació.



## ELEMENTS PER CONSTRUIR UN COMPUTADOR

### MODEL DE VON NEUMANN:

- Memòria (programa, dades)
- Processador
- Entrada / sortida
- Busos (permeten enllaçar els elements anteriors)

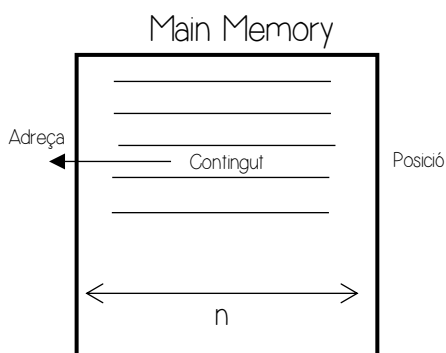
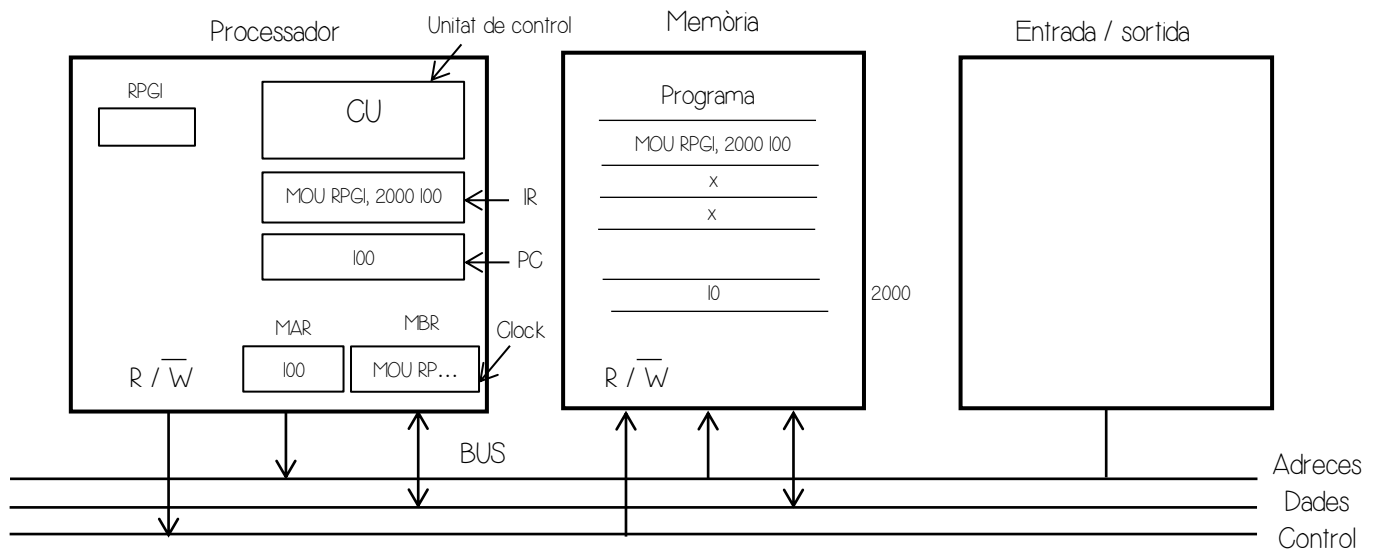


L'entrada i la sortida permet l'intercanvi d'informació entre l'entorn i el computador.

Programa resident: tot el programa sencer ha d'estar dins la memòria (està en llenguatge màquina). Per tant és directament comprensible pel computador.

El bus intercomunica les diferents parts.

Memòria: unitat passiva que permet emmagatzemar informació, tindrà el programa i totes les dades necessàries.



Està organitzada en posicions de memòria amb una adreça concreta

El contingut poden ser 0 o 1

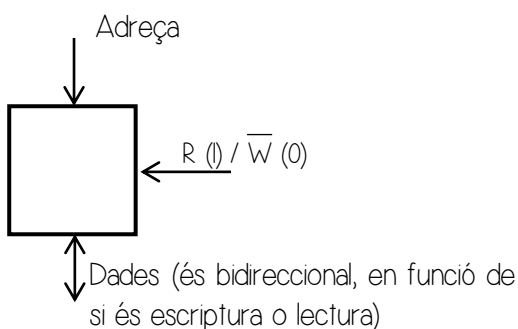
Podem fer dos operacions (lectura /read o escriptura/write)



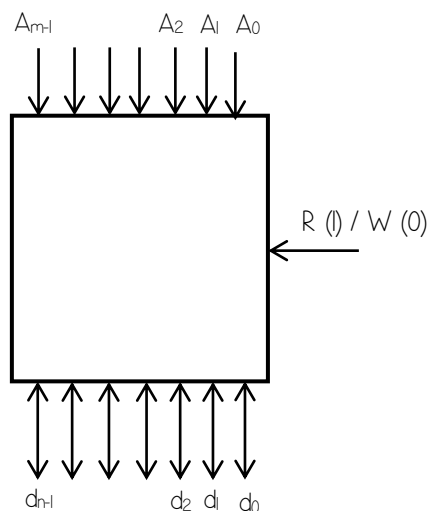
8 bits  $\rightarrow$  1 byte

16 bits  $\rightarrow$  2 bytes (o paraula)

32 bits  $\rightarrow$  4 bytes (o doble paraula)



$m$  = número de bits



Si la memòria fos de 1 GB llavors  $m = 30 \rightarrow 1 \text{ GB} = 2^{30} \text{ bits}$

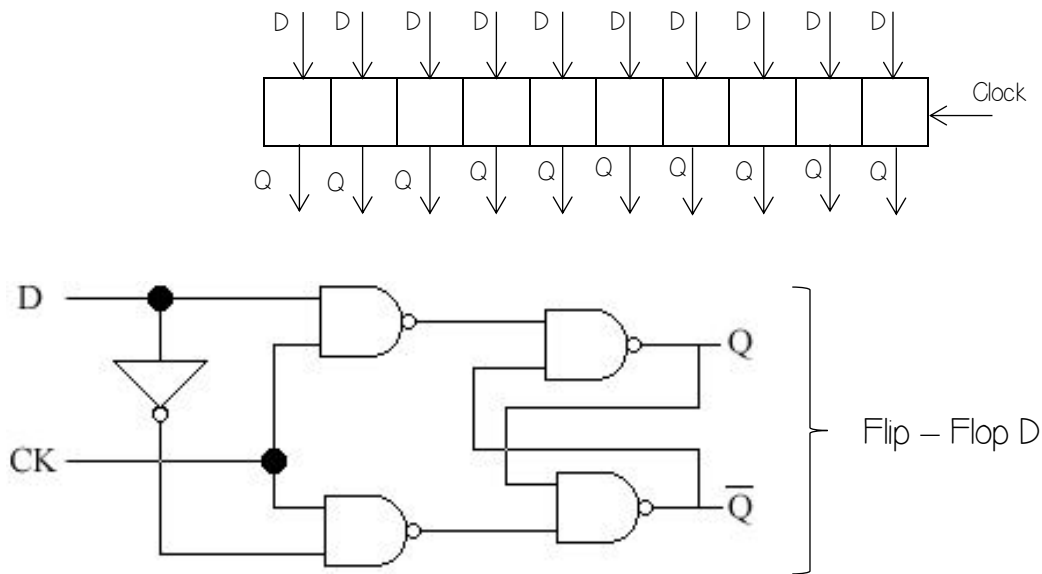
Si no es vol fer res es posa a lectura i així no s'altera res.

Temps d'accés: temps entre l'ordre de lectura fins que es produeix el resultat. És independent de la posició que ocupa i de si es tracta de lectura o escriptura

RAM → Random Access Memory (és una forma d'organització de la memòria)

L'escriptura és destructiva

Registre → dispositiu format per un conjunt de biestables.



Treballa de manera que si D val 0, Q val 0 i si D és 1, Q també, això per cada bit.

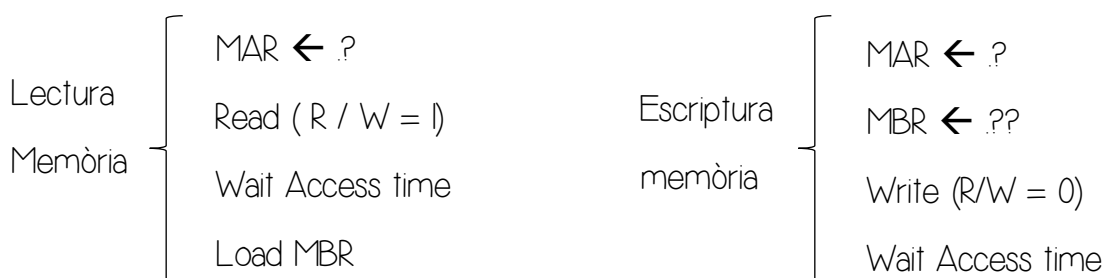
És un dispositiu com una memòria que permet fer una "instantània" del que tenim. No hi haurà canvis, tindrem una sortida estable. Controlem el moment en que entra la informació, es controla amb el clock.

El processador escriu les adreces i les rep la memòria

Unitat de control → autòmat, rep una informació del món exterior i en base d'això genera unes accions.

MAR → Memory Adress Register

MBR → Memory Buffer Register (registre en trànsit, temporal)



El llenguatge màquina són un conjunt d'instruccions que s'executen en un ordre determinat.

INSTRUCTION SET (Repertori d'instruccions)

Conjunt d'instruccions que pot fer una màquina

Les instruccions tenen el següent format:

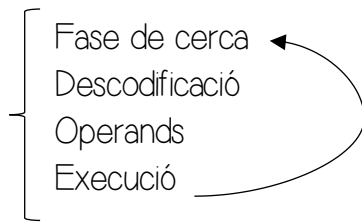


Op. Code → codi d'operació

Paràmetres → informació complementària

Operands → informació per fer allò que se li demana

## CICLE D'INSTRUCCIONS



En cada repetició el PC val un més per tant executada una darrera l'altra

### FASE DE CERCA:

MAR ← < PC > (ho fa CU)  
Read (R / W = I)  
Wait AT  
Load MBR  
IR ← < MBR >

PC → program counter: conté l'adreça de la posició de memòria de la pròxima acció a ser executada

< > → contingut de

IR → Registre d'instruccions

La fase de cerca es fa per totes les instruccions.

### FASE DE DESCODIFICACIÓ:

Mira primer el codi d'operació, després els seus paràmetres i finalment operarà amb els seus operands.

### FASE D'OPERANDS:

MAR ← < IR ><sup>(2000)</sup><sub>Ad</sub>  
Read  
Wait ΔT → ΔPC ( PC = < PC > + I )  
Load MBR

Quan acaba ha d'afegir a l'IR per executar els següents.

### FASE D'EXECUCIÓ:

RPGI ← < MBR >

## INSTRUCTION SET

L'instruction set és el menú que conté totes les instruccions que pot realitzar, que segueixen el format:

Codi d'operació	paràmetres	operands
-----------------	------------	----------

# Tipus d'instruccions

## TRANSFERÈNCIA:

Mouen una còpia de l'original a un altre lloc sense eliminar l'original. Aquesta transferència pot ser:

(STORE)	{	Registre → Memòria
MOV		Memòria → Registre
(LOAD)		Registre → Registre
		Memòria → Memòria (molt poc utilitzada)

MOV : s'utilitza per totes les transferències

LOAD ( Memòria → Registre )

STORE ( Registre → Memòria )

Per tractar transferències a la pila utilitzem:

- PSH : insereix informació a la pila
- PULL (O POP) : extreu informació de la pila

La pila es 'crea' a dins de la memòria RAM que la simulem:

PSH 2000	{	Posa al top de la pila la informació de 2000
Stack ← <2000>		
CPU → Reg <span style="border: 1px solid black; padding: 2px;">Stack pointer (SP)</span>		

## OPERATIVES

Les operatives es divideixen en aritmètiques i lògiques. Són operacions que es fan a la unitat aritmètica i que afecten als flags → N (negatiu), Z (zero), C (carry), V

Aquests flags són unitats lògiques sí / no (1 / 0)

### • ARITMÈTIQUES

SUM (ADD) – Sumar

SUB – Restar

MUL – Multiplicar

DIV – Dividir

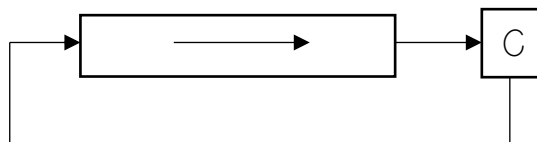
INC – Incrementar

DEC – Decreixement

CMP – Comparar

SHIFT – Desplaçament del contingut a la dreta o a l'esquerra d'un bit

ROT – Rotació



### • LÒGIQUES

- AND, NOT, OR, NAND, NOR, EXOR

RI 

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

OR RI, #00001000 → permet modificar un únic bit

## RUPTURA DE SEQUÈNCIA

Aquestes instruccions permeten fer bucles.

- JMP Adreça salt (on volem anar)

### ESTRUCTURA:

Fase de cerca

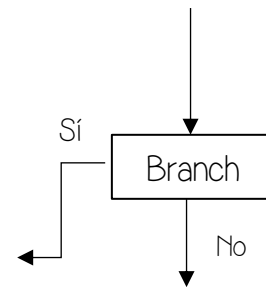
Descodificació

$PC \leftarrow < IR >_{Ad} (1000)$

- BRANCH  $\rightarrow$  BXX ( BON, BNN, BOZ, BNZ, BOC, BNC )  
 $N=1 \quad N=0 \quad Z=1 \quad Z=0 \quad C=1 \quad C=0$

El 'branch' es bifurca segons la pregunta en funció dels flags.

Continua i només salta si la condició es compleix



- CALL (salt a subrutina / funció)

El 'call' no és un simple jump perquè també ha de guardar el punt on ha de tornar després del salt (ha de guardar el PC)

### EXEMPLE:

I50 CALL 1000

Fase de cerca

Descodificació

Salvar PC ( I51 )

$PC \leftarrow < IR >_{Ad} (1000)$

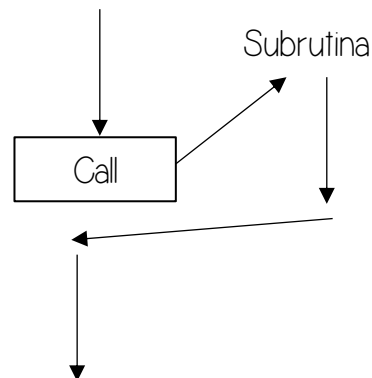
(Subrutina)

I327 RTN

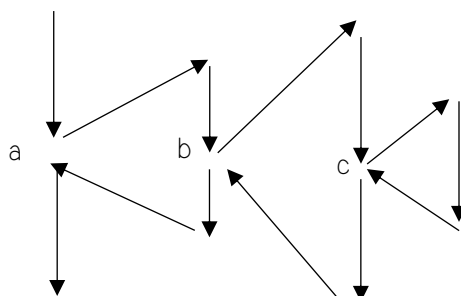
Fase de cerca

Descodificació

Restaurar PC ( I51 )



Per guardar el PC utilitzem una pila (stack). Va emmagatzemant el PC en la part alta i va accedint a elles per ordre.



Top
c
b
a

SP (stack pointer)  $\rightarrow$  apuntador de la pila Té la informació (adreça) del top de la pila

Per accedir a la pila utilitzem l'SP. La informació que enviarem al MAR també serà l'adreça SP.

Quan traiem el top de la línia, l'SP passa a una posició inferior.

x
y
z



Per extreure la informació de Z hem de posar SP apuntant a Z i enviar-lo al MAR, que el llegirà i la Z desapareixerà ja que la lectura de la pila és destructiva (un cop has fet el PULL l'SP ja passa de llarg, no hi podem accedir). L'SP per tant apunta la primera posició lliure de la pila

### PUSH RPG3

Fase cerca (searching phase)

MAR  $\leftarrow$  <PC>

READ

WAIT ACCESS TIME + PC  $\leftarrow$  <PC> + I

LOAD MBR

IR  $\leftarrow$  <MBR>

Fase de descodificació (decoding phase)

Fase d'operands (operand phase)

SP  $\leftarrow$  <SP> + I (va a la següent buida)

MAR  $\leftarrow$  <SP>

MBR  $\leftarrow$  <RPG3>

WRITE

WAIT ACCESS TIME

### PULL RPG3

Fase cerca (searching phase)

MAR  $\leftarrow$  <PC>

READ

WAIT ACCESS TIME + PC  $\leftarrow$  <PC> + I

LOAD MBR

IR  $\leftarrow$  <MBR>

Fase de descodificació (decoding phase)

Fase d'operands (operand phase)

MAR  $\leftarrow$  <SP>

READ

WAIT ACCESS TIME

SP  $\leftarrow$  <SP> - I

LOAD MBR

Fase d'execució (execution phase)

RPG3  $\leftarrow$  <MBR>

### CALL 4000 (jump + push)

Fase cerca (searching phase)

MAR  $\leftarrow$  <PC>

READ

WAIT ACCESS TIME + PC  $\leftarrow$  <PC> + I

LOAD MBR

$IR \leftarrow \langle MBR \rangle$

Fase de descodificació (decoding phase)

Fase d'operands (operand phase)

$SP \leftarrow \langle SP \rangle + 1$  (augmenta en 1 perquè va a la següent posició lliure)

$MAR \leftarrow \langle SP \rangle$  (En les següents línies guarda a la pila el PC abans de fer el salt)

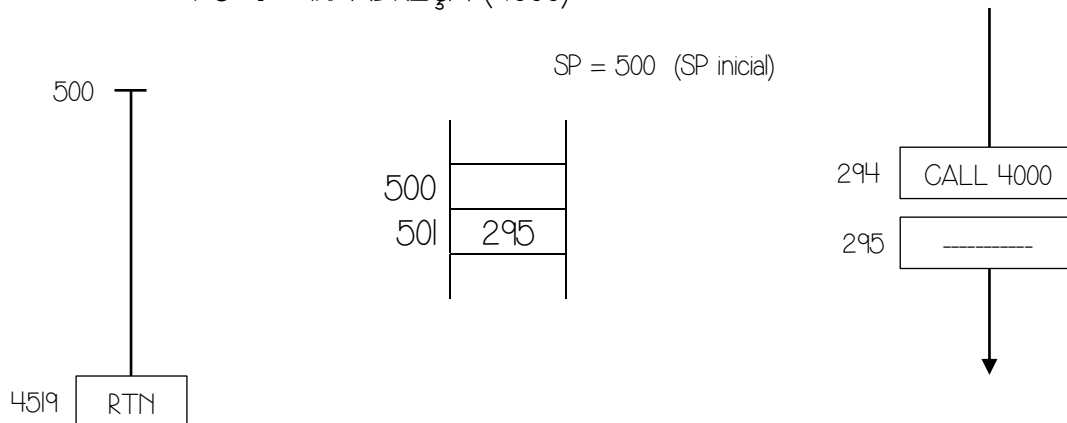
$MBR \leftarrow \langle PC \rangle$

WRITE

WAIT ACCESS TIME

Fase d'execució (execution phase)

$PC \leftarrow \langle IR \rangle \text{ADREÇA (4000)}$



RTN

Fase cerca (searching phase)

$MAR \leftarrow \langle PC \rangle$

READ

$WAIT ACCESS TIME + PC \leftarrow \langle PC \rangle + 1$

LOAD MBR

$IR \leftarrow \langle MBR \rangle$

Fase de descodificació (decoding phase)

Fase d'operands (operand phase)

$MAR \leftarrow \langle SP \rangle$

READ

WAIT ACCESS TIME

$SP \leftarrow \langle SP \rangle - 1$

LOAD MBR

Fase d'execució (execution phase)

$PC \leftarrow \langle MBR \rangle$

## EXEMPLES

while ( a <= b ) a++;      a → 1000 ; b → 2000

1 MOV RGI , 2000 RPG (registre de propòsit general)

2 CMP RGI , 1000

3 BON 6 (salt si N=1 a la línia 6)

```

4 INC 1000
5 JMP 1 (salt incondicional a la línia 1)
6 Continuar

```

Per comprovar si un nombre és major o menor es resta un menys l'altre i es comprova si el resultat és positiu o negatiu.

```

If ( a=b ) { a := 1 }
else { a:= 0 }          a → 1000 ; b → 2000

```

```

1 MOV RG7 , 1000
2 CMP RG7 , 2000
3 BOZ 6 (Salta si Z=1) → és un valor lògic pren el valor de 0 o 1
4 MOV 1000 , # 0 (és un operand, no una posició de memòria)
5 JMP 7 (salt incondicional)
6 MOV 1000 , #1
7 Continuar

```

## MODES D'ADREÇAMENT

L'adreça explícita és la que figura explícitament a la pròpia instrucció.

L'adreça efectiva (EA) és l'adreça on realment trobem l'operand

ADREÇAMENT: és una funció a partir de l'adreça explícita "m" de l'operand calcula la seva adreça efectiva

$$EA = F(m) \quad \text{Operand} = \langle EA \rangle$$

ADREÇAMENT DIRECTE a memòria i a registre

$$EA = m \quad \text{Operand} = \langle EA \rangle = \langle m \rangle$$

```
MOV 2000 , AX
```

Fase cerca (searching phase)

```

MAR ← <PC>
READ
WAIT ACCESS TIME + PC ← <PC> + 1
LOAD MBR
IR ← <MBR>

```

Fase de descodificació (decoding phase)

Fase d'operands (operand phase)

```

MAR ← <IR> adreça (2000)
MBR ← <AX>
WRITE
WAIT ACCESS TIME

```

No hi ha fase d'execució

### ADREÇAMENT IMMEDIAT

Operand = m

MOV 2000 , #123 (s'utilitza #)

Fase cerca (searching phase)

Fase de descodificació (decoding phase)

Fase d'operands (operand phase)

MAR  $\leftarrow$  <IR> adreça1 (2000)

MBR  $\leftarrow$  <IR> adreça2 (123)

WRITE

WAIT ACCESS TIME

No hi ha fase d'execució

### ADREÇAMENT A REGISTRE BASE

EA = m + <Reg Base>

MOV AX , (BX +4)

Inicialment registre base BX = 1500

Fase cerca (searching phase)

Fase de descodificació (decoding phase)

Fase d'operands (operand phase) m = 4

MAR  $\leftarrow$  <BX> + <IR> adreça (4) EA = 1504

READ

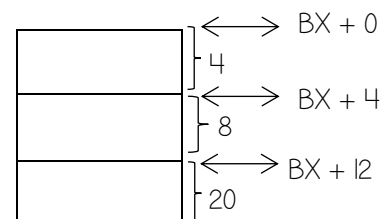
WAIT ACCESS TIME

LOAD MBR

Fase d'execució (execution phase)

AX  $\leftarrow$  <MBR>

Pensat per gestionar els diferents camps d'una estructura (registre r/w) que comença a la posició d'adreça 1500. El primer camp és a la posició relativa 0 (mida 4) el segon a la posició relativa 4 (mida 8), el tercer a la 12 .



### ADREÇAMENT RELATIU

EA = m + <PC>

JMP -100 (tira 100 posicions endarrere, si fos positiu aniria endavant)

Inicialment registre base PC = 38500

Fase cerca (searching phase)

Fase de descodificació (decoding phase)

Fase d'operands (operand phase)

PC  $\leftarrow$  <PC> + <IR> adreça (-100) PC = 38400

Adreçament relatiu al punt de memòria on som. D'aquesta manera independitza de la posició de memòria ja que el nostre programa no estarà sempre en les mateixes posicions.

ADREÇAMENT INDIRECTE A MEMÒRIA (apuntadors)      EA = < m >      Operand = < < m > >

MOV AX , < 1000 >

Mou a AX la posició 1000 amb adreçament indirecte ( <    > )

Fase cerca (searching phase)

Fase de descodificació (decoding phase)

Fase d'operands (operand phase)

MAR  $\leftarrow$  <IR> adreça (1000)

READ

WAIT ACCESS TIME

LOAD MBR

MAR  $\leftarrow$  < MBR >

READ

WAIT ACCESS TIME

LOAD MBR

Fase d'execució (execution phase)

AX  $\leftarrow$  <MBR>

15	300
300	1000

L'operand seria 15.

A dins la posició 1000 té l'adreça on hi ha l'operand

1000 és un apuntador

ADREÇAMENT INDIRECTE A REGISTRE (apuntadors)      EA = < m >

MOV AX , < BX >

Fase cerca (searching phase)

Fase de descodificació (decoding phase)

Fase d'operands (operand phase)

MAR  $\leftarrow$  < <IR> adreça (REG) > ( < BX > )

READ

WAIT ACCESS TIME

LOAD MBR

Fase d'execució (execution phase)

AX  $\leftarrow$  <MBR>

BX substitueix el 1000 de l'adreçament anterior. El tenim dins el registre BX, no fa falta accedir a dos accessos de memòria només a un

### ADREÇAMENT INDEXAT

EA = m + < Reg Índex >

MOV AX , 200 (X)

Fase cerca (searching phase)

Fase de descodificació (decoding phase)

Fase d'operands (operand phase)

MAR  $\leftarrow$  <IR> adreça (200) + < X >

READ

WAIT ACCESS TIME

LOAD MBR

Fase d'execució (execution phase)

AX  $\leftarrow$  <MBR>

Permet fer un conjunt d'instruccions de manera consecutiva. S'assembla a l'adreçament base. Instruccions INX (incrementar X) i DEX (decrementar X) permeten recorre còmodament llistes i arrays.

EXEMPLE: for ( i = 0 ; i < 100 ; i++) a[i] = b[i] + c[i];  
a[i] -> 1000 b[i] -> 2000 c[i] -> 3000 (primera posició de cada array)

En aquest programa sumem dos arrays i guardem el resultat en un tercer.

1 MOV X , #0 (x=0)

2 MOV RG0 , 2000 (X) (movem la posició 2000 al registre)

3 SUM RG0 , 3000 (X) (sumem al registre anterior la posició 3000)

4 MOV 1000 (X) , RG0 (movem el primer valor obtingut a la primera posició del vector resultat)

5 INX (incrementem la x)

6 CPX #100 (CMP X, #100) (comprovem si sortim del for, compara el valor de x amb 100)

7 BNZ 2 (Salta si Z=0)

8 Continuar

MOV X , # 100 Alternativa més curta

Comencem al revés i baixem i així estalviem el comparar.

1

2 MOV AX , 1999 (X)

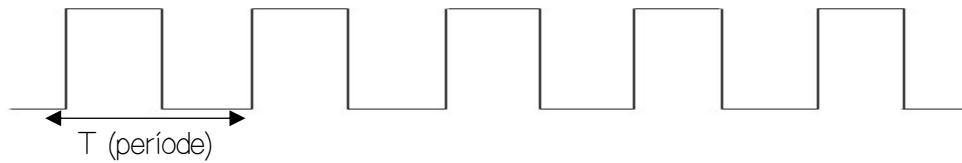
3 SUM AX , 2999 (X)

4 MOV 0999 (X) , AX

5 DEX

6 BNZ 2

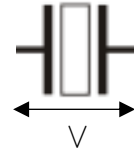
Totes les instruccions estan regides per la unitat de control



Ona quadrada que marca els moments en els que fer les coses.

El clock funciona amb un cristall de quars.

- Més prim el cristall → més alta freqüència
- Més ample el cristall → més baixa freqüència



## MEMÒRIA

### RAM

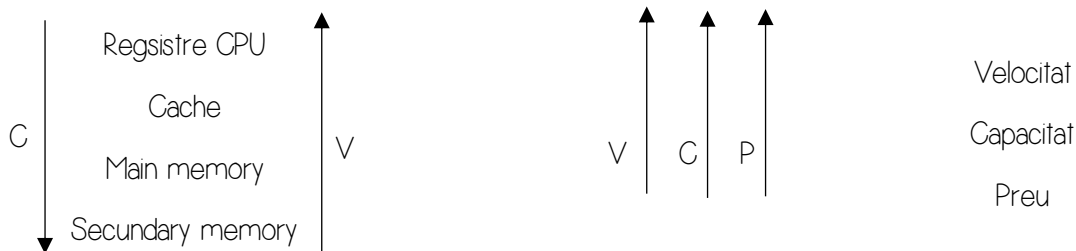
Les memòries RAM poden ser de dos, els dos són tecnologies volàtils, necessiten energia constantment:

- ESTÀTIC: estan creades per portes. Consumeixen més energia i ocupen més espai
- DINÀMIC: estan construïdes per condensadors. Carregat = 1, Descarregat = 0. Consumeixen menys i ocupen menys. L'inconvenient és que se'ls ha d'afegir circuits de refresc, s'ha de tornar a carregar aquells condensadors descarregats.

### NO VOLÀTILS

- ROM: read only memory, és un tipus de RAM. Les RAM poden ser volàtils o no volàtils. La RAM és una forma d'organització de la memòria, no es contraposa la RAM i la ROM. La ROM normalment té una organització RAM. La ROM no té valors aleatoris, té uns continguts, s'utilitza per contenir els programes d'engegada.
- PROM: ROM programada, és una ROM en blanc que pots afegir continguts. Serveixen sobretot per fer proves. Es produeixen a partir de fusibles, és permanent.
- EPROM: es poden esborrar i es pot modificar amb una llum ultravioleta
- EEPROM: PROM elèctricament esborrables. Tenen bon temps de lectura, però no tant d'escriptura (els USB i la memòria SSD són EEPROM)

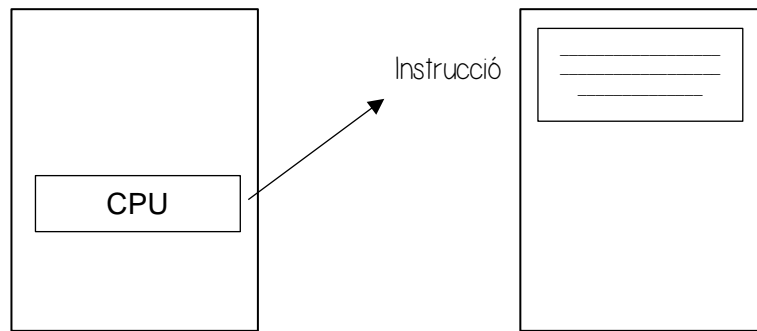
## JERARQUIA DE MEMÒRIA



MS → emmagatzematge permanent

Cache: està situada dins la CPU, és una memòria associada. S'hi accedeix per contingut.

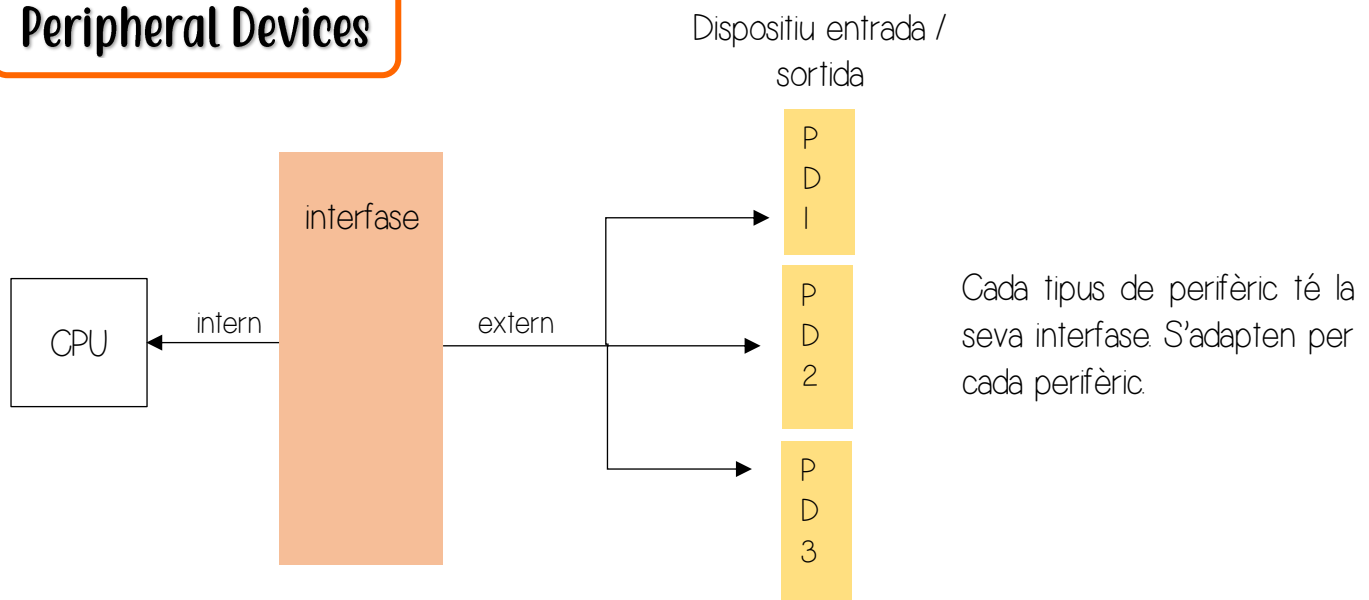
Buscarà el contingut a la maina memory i a la CPU i la situarà a caché. També agafa el contingut veí.



La idea és que si es torna a buscar aquesta informació sigui més accessible i ràpid ja que es troba a la CPU.

## ENTRADA / SORTIDA

### Peripheral Devices



Hi ha tres aspectes bàsics dels que depèn el tipus d'interfase :

- Elèctrics: ha de proporcionar compatibilitat elèctrica
- Velocitat: les entrades i sortides tenen diferent velocitat de la CPU
- Timing : conjunt de senyals que implica realitzar la lògica entrada/sortida

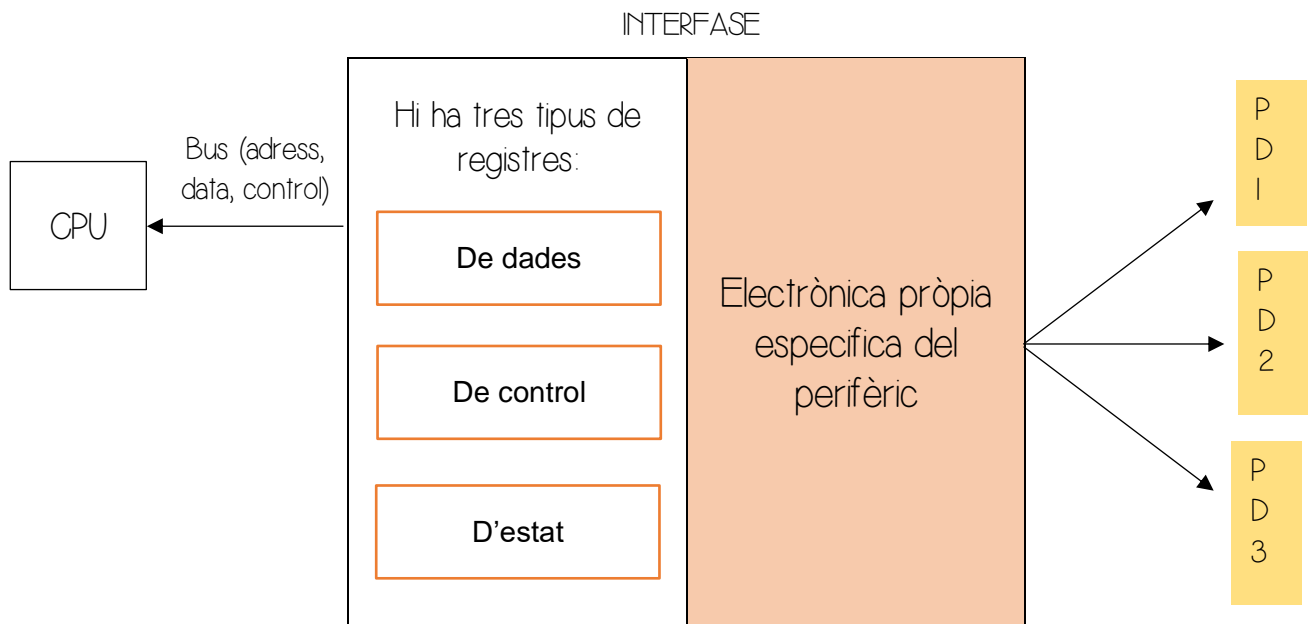
### Interfase

La interfase té dos parts:

- Controlador (Hardware) → ve instal·lada a l'ordinador
- Software (driver) → la proporciona el fabricant

El driver sap les adreces de cada registre, per tant sense el driver no funciona res, el driver té el control.





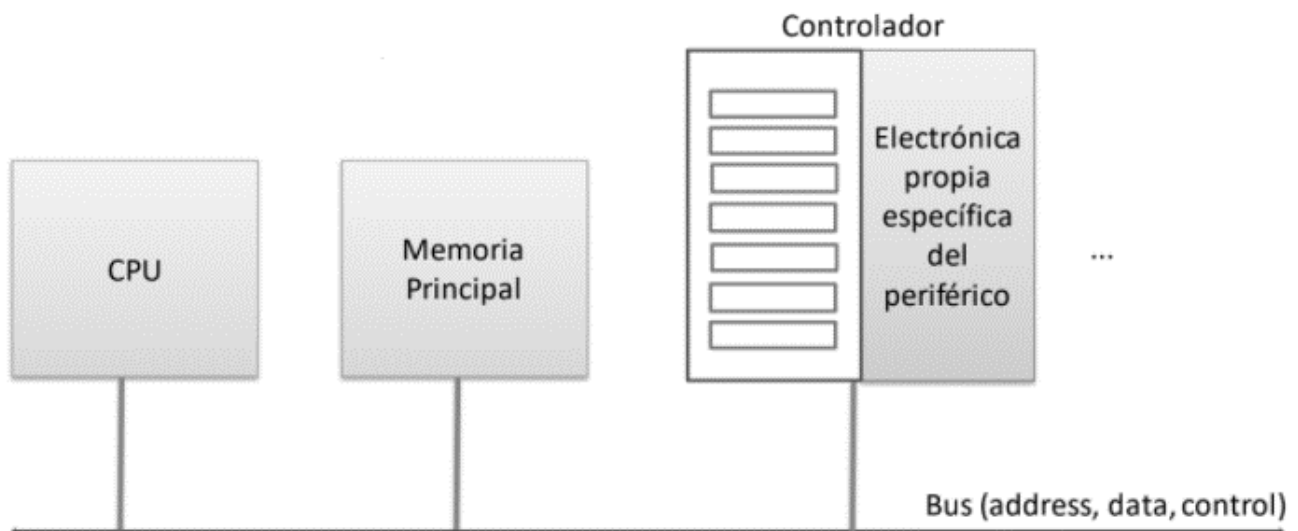
De dades: poden ser d'entrada i / o sortida S'anomenen Port → són de lectura i escriptura

De control o comandes: per escriure les ordres per gestionar els perifèrics → són d'escriptura

D'estat: informació de la condició en la que està el perifèric: ocupat, funcionant... → són de lectura

## Models per veure el registre d'estat

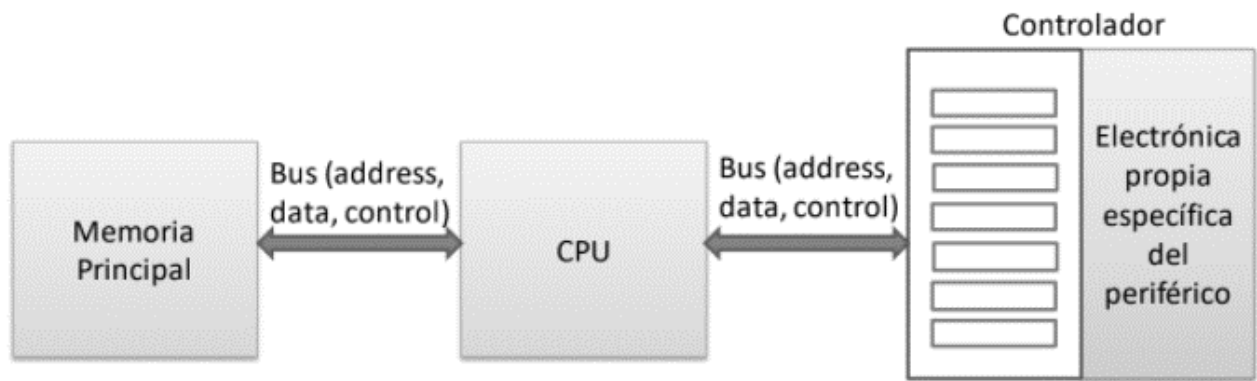
I / O MEMORY MAPPED:



Els registre són vistos com direccions de memòria No tenim instruccions específiques per fer l'entrada i la sortida S'utilitza el MOV. Hi ha un bus únic, els registres no formen part de la memòria

Hi ha un rang de memòria destinada pels controladors, hi ha d'haver una clara identificació del rang, per la CPU ha de ser transparent. Si tenim un registre de comandes a la posició FF34, aquesta adreça no pot existir a la memòria principal perquè seria un registre controlador.

## E / S EXPLÍCITA



MOVE RG24, FF26

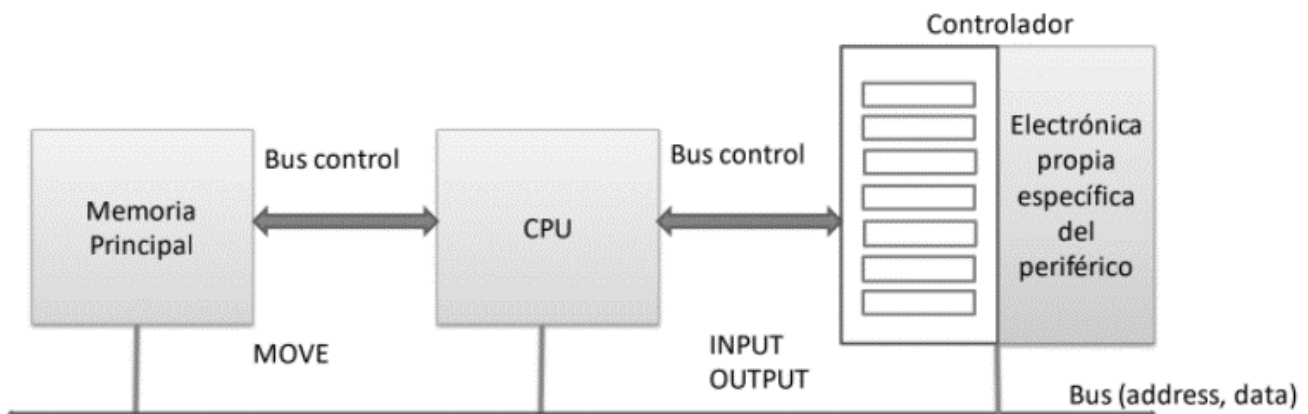
Entrada → INPUT RG2, 27

Sortida → OUTPUT RG3, 18

Aquí el 27 y 18 son  
registros del controlador

En aquest cas tenim instruccions d'entrada i sortida. Els registres són memòria

## E / S EXPLÍCITA HÍBRIDA

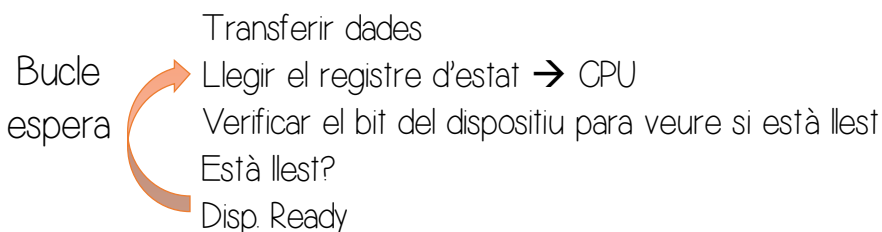


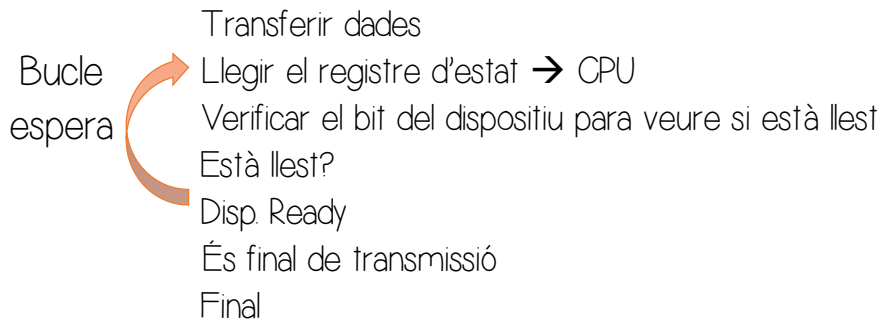
Els ports d' E / S ocupen direccions de memòria perquè el bus de direccions és comú a la memòria i als controladors.

## Software (driver)

Transferència programada de dades. Els passos a realitzar entre la CPU i un dispositiu són:

- Seleccionar (connectar) al dispositiu (activar el controlador)
- Determinar l'estat del dispositiu (sobre un registre d'estat)





Hi ha una espera improductiva, una espera on no fa res. És un bucle fins que se li dona la informació esperada. Utilitzem les interrupcions per passar a un altre programa mentre esperem i ens avisa que està disponible amb una senyal elèctrica flang. La CPU mira al final del cicle d'instrucció (quan acaba un procés) per mirar si hi ha alguna interrupció.

### EXEMPLE:

F583 = registre de control del dispositiu ( bit 3 selecció / actiu)

F584 = registre d'estat del dispositiu ( bit 6 ready 1 / busy 0)

F500 = registre de dades d'entrada ( va rebent seqüencialment les dades que arriben d'una xarxa, total MIDA )

Selecció / Activació 1 OR F583 , # %00001000

Monitorització 2 MOV RPG3 , F584

3 AND RPG3 , # %01000000

4 BOZ 2

Intercanvi 5 MOV RPG1 , F500

6 MOV I000 (X) , RPG1

Final? No, repetir a Monitorització 7 INX

8 CPX #MIDA

9 BNZ 2

Sí, continuar 10 CONTINUAR

Utilitzem lògica pel controlador:

0	0	0	0	1	0	0	0
7	6	5	4	3	2	1	0

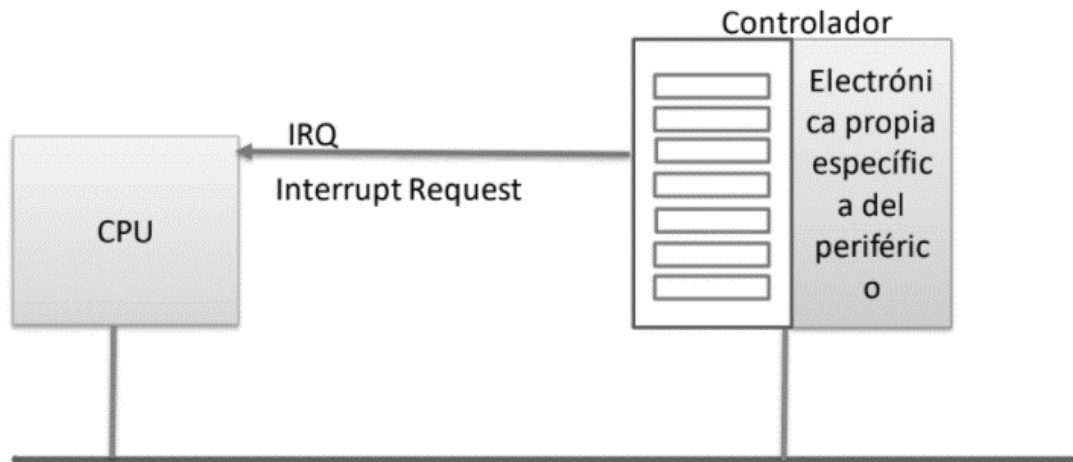
## INTERRUPCIÓ

Quan l'entrada o la sortida avisa que necessita fer la interrupció, la CPU es para i atén l'entrada o sortida

Per fer una interrupció fem:

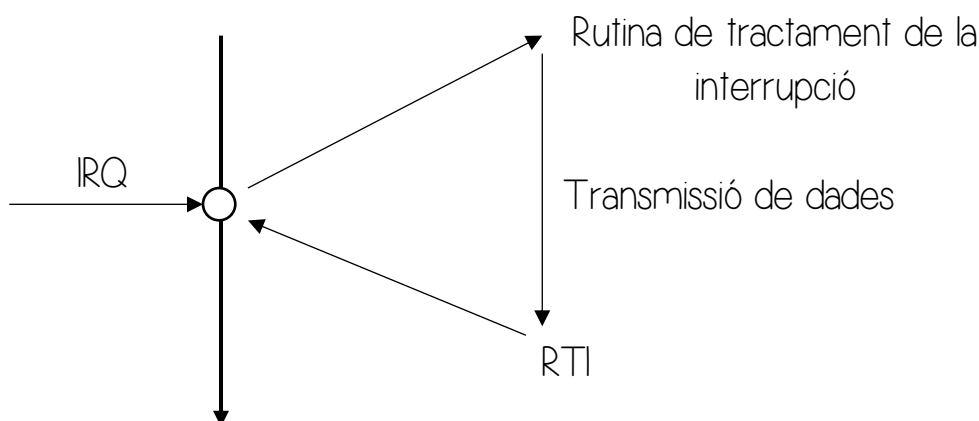
- Seleccionar el dispositiu
- Determinar l'estat del controlador

- Programar el sistema d'interrupcions del controlador
- Transmetre dada
- Fem un altra cosa
- Transmetre dada



### REBRE UNA INTERRUPCIÓ

- Quan rep una interrupció acaba el cicle de la instrucció que està fent
- Guarda el PC → Pila
- Guarda l'estat de la CPU → Pila
- Canvia el mode
- PC ← Vector d'interrupció (és l'adreça on comença la rutina d'interrupció). El controlador envia la interrupció + vector d'interrupció a través del bus de dades
- Cos de la rutina d'interrupció (transmetre dades). El context es gestiona similar a fer un CALL
- RTI, retorn de la interrupció, instrucció del repertori d'instruccions



RTI → retorn d'interrupció

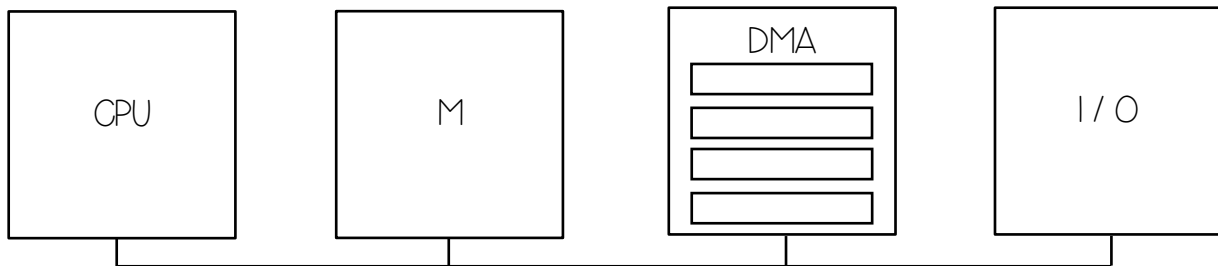
- Canvia de mode (a usuari)
- Restaura l'estat de la CPU
- Restaura la CPU

## CANVI DE MODE:

Al rebre un IRQ es realitza un canvi de mode. Hi ha dos modes de funcionament:

- Usuari: per executar programes d'usuari. L'espai de la memòria principal és limitat, està limitat també l'espai dels controladors i al repertori d'instruccions. No té accés als registres interns de la CPU.
- Supervisor: per executar el sistema operatiu. Es pot fer totes les accions.

## TRANSFERÈNCIA AUTÒNOMA



Controlador DMA → substitueix la CPU, vol la transferència sense haver de passar per la CPU.

La CPU programa la DMA i li dona la informació que necessita en quatre registres.

Els quatre registres són:

- Perifèric: dispositiu
- Adreça: on comença
- Quantitat: d'informació a transferir
- Entrada o sortida

Cada cop augmenta l'adreça i disminueix la quantitat fins que la quantitat a transferir és 0.

D'aquesta manera si la CPU no està fent servir el bus, la CPU es pot dedicar a executar un procés i mentrestant el DMA pot encarregar-se de l'entrada i sortida.

Cada Access Time hi ha una transferència amb el DMA en canvia la CPU tarda més perquè ha de fer la fase de cerca, descodificació...

Les interrupcions poden ser:

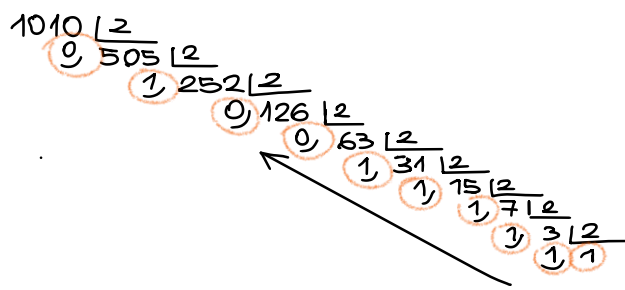
- Hardware: són síncrones i estan produïdes per un perifèric.
- Excepcions: situacions excepcionals que es produeixen en la CPU (overflow, dividir entre 0...)
- Software: interrupcions programades (asíncrones) que s'utilitzen en processos, saltem a una rutina d'interrupcions.

# Problemes Representació

BINARI	DECIMAL	OCTAL	HEXADECIMAL
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	8	8
1001	9		9
1010	10		A
1011	11		B
1100	12		C
1101	13		D
1110	14		E
1111	15		F

## EXEMPLES

1)  $1010_{10}$



$$1010_{10} = 1111110010_2$$

2)  $24,52_{10}$

Separem la part entera i la part decimal

Part decimal:

$$0,52 \cdot 2 = 1,04 - 1 = 0,04$$

$$0,04 \cdot 2 = 0,08$$

$$0,08 \cdot 2 = 0,16$$

$$0,16 \cdot 2 = 0,32$$

$$0,32 \cdot 2 = 0,64$$

$$0,64 \cdot 2 = 1,28 - 1 = 0,28$$

$$0,28 \cdot 2 = 0,56$$

$$0,56 \cdot 2 = 1,12 - 1 = 0,12$$

$$0,12 \cdot 2 = 0,24$$

La part decimal acaba quan s'arriba al zero i normalment es fa simplement una aproximació.

$$24,52_{10} = 11000.100001010 \dots_2$$

**PROBLEMA 1** Canvi de base numèric, passar de decimal a binari

c)  $45,24_{10}$

$$0,24 \cdot 2 = 0,48$$

$$0,48 \cdot 2 = 0,96$$

$$0,96 \cdot 2 = 1,92 - 1 = 0,92$$

$$0,92 \cdot 2 = 1,84 - 1 = 0,84$$

$$45,24_{10} = 101101.0011110 \dots_2$$

$$0,84 \cdot 2 = 1,68 - 1 = 0,68$$

$$0,68 \cdot 2 = 1,36 - 1 = 0,36$$

$$0,36 \cdot 2 = 0,72$$

**PROBLEMA 2** Representació de números amb signes

Número	Tipus de representació	
	Signe i magnitud	Complement a 2
$1523_8$	0000001101010011	00000011010011
$-1523_8$	1000001101010011	11111100101101
$1315_{10}$	0000010100100011	0000010100100011
$-1315_{10}$	1000010100100011	1111101011011101

Observem que entre el positiu i el negatiu a complement a 2 obtenim el mateix nombre però invertint els 0 i els 1 a partir del primer 1 començant per la dreta

$1523_8 \rightarrow$     1    5    2    3<sub>8</sub>    Base 8 agrupem d'en tres en tres  
                   001   101   010   011<sub>2</sub>    Complement a 2 d'un nombre positiu és el mateix  
                   0000 001   101   010   011<sub>2</sub>    (utilitzem 16 bits)

$$1315_{10} \rightarrow \begin{array}{r} 1315 \div 2 = 657 \text{ residu } 1 \\ 657 \div 2 = 328 \text{ residu } 1 \\ 328 \div 2 = 164 \text{ residu } 0 \\ 164 \div 2 = 82 \text{ residu } 0 \\ 82 \div 2 = 41 \text{ residu } 0 \\ 41 \div 2 = 20 \text{ residu } 1 \\ 20 \div 2 = 10 \text{ residu } 0 \\ 10 \div 2 = 5 \text{ residu } 0 \\ 5 \div 2 = 2 \text{ residu } 1 \\ 2 \div 2 = 1 \text{ residu } 0 \\ 1 \div 2 = 0 \text{ residu } 1 \end{array}$$

FA8 <sub>16</sub>	0000111110101000	0000111110101000
-FA8 <sub>16</sub>	1000111110101000	1111000001011000

FA8<sub>16</sub> → F      A      8      Base 16 agrupem d'en quatre en quatre  
           1111    1010    1000

**PROBLEMA 3** Determinar el valor en base 10 dels següents en complement a 2

- a) 1100101010111000<sub>2</sub>  
 0011010101001000<sub>2</sub> =  $2^3 + 2^6 + 2^8 + 2^{10} + 2^{11} + 2^{12} = -13640_{10}$   
 (el signe l'indica el primer caràcter, els valors de les potències de 2 corresponen a les posicions dels uns)

- b) 176102<sub>3</sub> = 1111110001000010<sub>2</sub> = -958<sub>10</sub>

1    7    6    1    0    2

001 111 110 001 000 010 ← té 18 bits i ens el demanen de 16 i el primer indicar el signe

$$C_2 = 00000011110111110 = 2^1 + 2^2 + 2^3 + 2^4 + 2^7 + 2^8 + 2^9 = -958$$

**PROBLEMA 4** Conversió de bases:

- a) 1A7<sub>16</sub> = 0001 1010 0111<sub>2</sub>  
 b) FA3,D5<sub>16</sub> = 1111 1010 0011 1101 0101<sub>2</sub>  
 c) 527<sub>10</sub> = 101 010 111<sub>2</sub>  
 d) 720,64<sub>10</sub> = 111 010 000, 110 100<sub>2</sub>  
 e) 915,25<sub>10</sub> = 1110010011,01<sub>2</sub>

$$0,25 \cdot 2 = 0,5$$

$$0,5 \cdot 2 = 1$$

$$915 \div 2 = 457 \text{ residu } 1 \\ 457 \div 2 = 228 \text{ residu } 1 \\ 228 \div 2 = 114 \text{ residu } 0 \\ 114 \div 2 = 57 \text{ residu } 0 \\ 57 \div 2 = 28 \text{ residu } 1 \\ 28 \div 2 = 14 \text{ residu } 0 \\ 14 \div 2 = 7 \text{ residu } 0 \\ 7 \div 2 = 3 \text{ residu } 1 \\ 3 \div 2 = 1 \text{ residu } 1$$

- f) 1110101,011<sub>2</sub> =  $6^6 + 6^5 + 6^4 + 6^2 + 6^0 + 6^{-2} + 6^{-3}$ <sub>6</sub>  
 g) 5AD,B7<sub>16</sub> =  $5 \cdot 16^2 + 10 \cdot 16^1 + 13 \cdot 16^0 + 11 \cdot 16^{-1} + 7 \cdot 16^{-2} = 1483,7148$ <sub>10</sub>



## PROBLEMA 5 Operacions

$$A = 1638_{10} = 000\ 001\ 110\ 011\ 101_{12}$$

$$B = 759_{10} = 0000\ 0010\ 1111\ 0111_{12}$$

$$C = 1011011110_{12}$$

$$C_2(-B) = 1111\ 1101\ 0000\ 1001$$

A - B:

$$\begin{array}{r} 0000\ 0011\ 1001\ 1101 \\ 1111\ 1101\ 0000\ 1001 \\ \hline 0000\ 0000\ 1010\ 0110 \end{array}$$

# Problemes Computadors

PROBLEMA 1 Escriviu un programa en ensamblador que implementi el mateix algorisme:

```
N = 4;
sum = 0;
partial_sum = 0;
i = 0;

while ( i < N ) {
    partial_sum = partial_sum + v[i];
    i = i + 1;
}

sum = partial_sum;
```

```
MOV R1, 4
MOV R2, 0
MOV R3, 0
MOV R4, 0
while: CMP R4, R1
      JGE fin
      ADD R3, V[+R4]
      ADD R4, 1
      JMP while
fin:   MOV R2, R3
```

PROBLEMA 2 A Escriviu un programa en ensamblador que implementi el mateix algorisme:

```
if ( A > B ) {
    C = C + 3;
else
    C = C - 1;
}
```

```
MOV R1, [A]
CMP R1, [B]
JLE else
ADD [C], 3
JMP fin
else: SUB [C], 1
fin:
```

PROBLEMA 2 B Escriviu un programa en ensamblador que implementi el mateix algorisme:

```
while ( A < B ) {
    C = C * C
    A = A + 1
}
```

```
MOV R1, [A]
MOV R2, [B]
MOV R3, [C]
while: CMP R1, R2
      JGE fin
      MUL R3, R3
      INC R1
      JMP while
      MOV [C], R3
MOV [A], R1
```

PROBLEMA 2 C Escriviu un programa en ensamblador que implementi el mateix algorisme:

```
num = 6;
count = 1;
fact = 1;
while ( count < num) {
    count = count + 1;
    fact = fact * count;
}
```

```
MOV R1, 6
MOV R2, 1
MOV R3, 2
while: CMP R2, R1
      JGE fin
      INC R2
      MUL R3, R2
      MOV [fact], R2
      JMP while
fin:
```

PROBLEMA 3 A Escriviu en RI un 1 si el contingut de R2 és més gran o igual que el de R3 i a més el de R4 és més petit que el de R5. Si no passa tot l'anterior, escriviu un 0.

```
MOV R1, 0
CMP R2, R3
JL fin
CMP R4, R5
JGE fin
MOV R1, 1
fin:
```

## PROBLEMES ADREÇAMENTS

Tipus d'adreçaments:

- Directe → MOV A, 2
- Immediat → MOV, #2
- Indexat → MOV A, 2[x]
- Indirecte → MOV A, (2)

PROBLEMA 1 A partir del següent fragment de programa i l'estat de la memòria, mostreu l'evolució dels registres i del contingut de la memòria a mesura que s'executa el programa.

```
0200 MOV A, (3)
0201 ADD A, #5
0202 MOV 6, A
0203 SUB A, 3[X]
0204 MOV (1), X
0205 MOV A, 1[X]
0206 ADD A, (1)
0207 MOV (7), A
```

Instrucció	Registre			Posició de la memòria							
	PC	A	X	0	1	2	3	4	5	6	7
Estat inicial	199	0	2	6	4	4	5	3	6	2	1
200 MOV A, (2)	201	3			(L1)		(L2)				
201 ADD A, #5	202	8									
202 MOV 6, A	203									8	
203 SUB A, 3 [x]	204	2					(L1)		(L2)		
204 MOV (1), x	205				(L)			2			
205 MOV A, 1 [x]	206	5			(L1)		(L2)				
206 ADD A, (1)	207	8			(L1)			(L2)			
207 MOV (7), A	208				8						(L1)

**PROBLEMA 2** A partir del següent fragment de programa i l'estat de la memòria, mostreu l'evolució dels registres i del contingut de la memòria a mesura que s'executa el programa

La màquina incorpora una pila amb la següent funcionalitat i instruccions

- PUSH [ $sp \leftarrow sp+1$ ,  $(sp) \leftarrow \text{valor}$ ]
- POP [ $(sp) \rightarrow \text{valor}$ ,  $sp \leftarrow sp-1$ ]
- CALL RutinaX [ $sp \leftarrow sp+1$ ,  $(sp) \leftarrow [\text{adreça actual (PC)} + 1]$ ]
- RET [ $\text{adreça actual (PC)} \leftarrow (sp)$ ,  $sp \leftarrow sp-1$ ]

0100 MOV A, #8	Rutina I:
0101 MOV X, #6	0120 PUSH A
0102 CALL RutinaI	0121 SUB A, X
0103 MOV 206, A	0122 POP X
0104 MOV 207, X	0123 RET

[illegible]