

*notebook*

GOOD MONDAYS PAPER

## Sections

1 *Introducció*2 *Assemblador i tipus de dades*3 *Traducció de programes*4 *Matrícies*5 *Aritmètica enters i coma flotant*6 *Memòria cache*7 *Memòria virtual*

8

9

10

11

12

T

1

2

3

4

5

6

7

8

9

10

11

12

•  
•  
•

T

1

2

3

4

5

6

7

8

9

10

11

12

## *Section One*

### *Introducción*

---

[BACK TO INDEX](#)

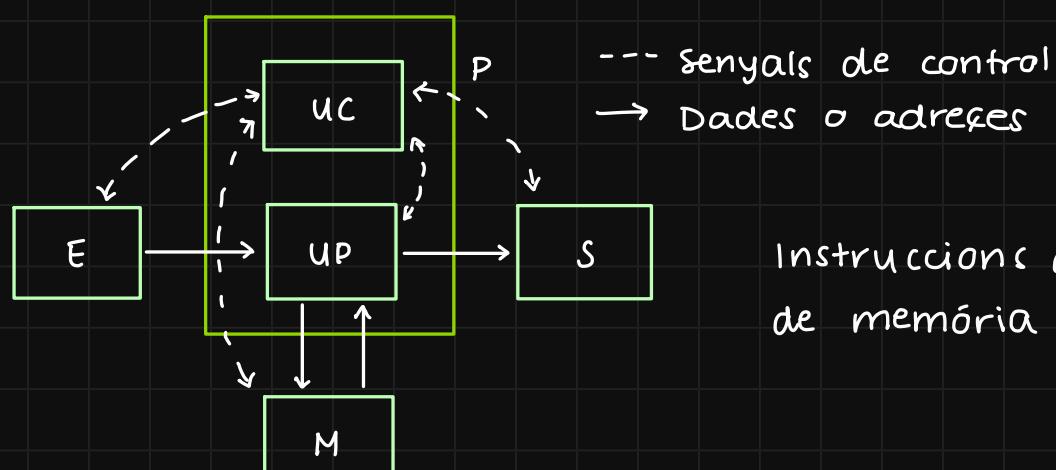
# DESCRIPCIÓN GERARQUICA DEL COMPUTADOR

1. Hardware
  2. Llenguatge màquina
  3. Llenguatge d'alt nivell
  4. Llenguatge d'usuari

Instructon Set Architecture (ISA) especificació que descriu els aspectes del processador viables a un programador de llenguatge màquina (o assemblador) : instruccions, registres...

Application Binary Interface (ABI) especificació que descriu la interface de baix nivell entre dos mòduls d'un programa (convenis de crides i retorn de funcions).

## 1. PARTS D'un COMPUTADOR VON NEUMANN



Instruccions del programa en el mateix espai d'adreces de memòria on es guarden les dades.

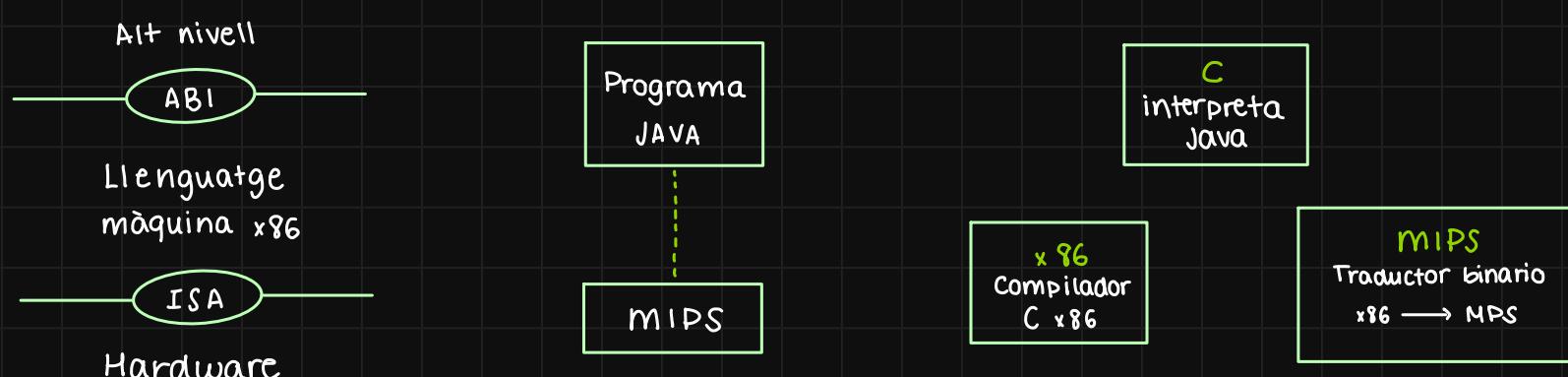
exemple : computador mips

Com executem un programa en java si tenim compiladors de C a x86 (escrito en x86)

## Interprete de Java

compiladores de C a x86 (escrito en x86)

## Traductor binari a x86 a MIPS



•  
•  
•  
T  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12

## *Section Two*

### *Assemblador i tipus de dades*

---

[BACK TO INDEX](#)

## ASSEMBLADOR

MIPS → ISA de tipus RISC, que és l'arquitectura de tipus RISC?  
 ↗ microprocessor without Interlocked Pipeline

- Reduced Instruction Set Computer (RISC):
  - Instruccions de mida fixa, pocs modes d'adreçament, accés a memòria load i store.
  - → Poques instruccions.
  - ARM, PowerPC, MIPS, Alpha, SPARC, etc.

## Arquitectura de MIPS

- 32 bits (1 word = 32 bits = 4 bytes)
- Instruccions de 32bits, adreces de 32 bits, 32 registres de 32 bits
- Memòria unificada per a instruccions i dades

## TIPUS DE DADES

- Palabras = words (32 bits)



## VARIABLES GLOBALES

Són accessibles des de qualsevol funció i sempre estan a la mateixa @.

## VARIABLES LOCALS

Accessibles al bloc declarat, deixen d'exsistir al sortir del bloc, es reserva espai a la memòria de manera dinàmica.

ex:

En C,

```
int a = 0x44332211
int main(void) {
    int i = 7;
```

}

En MIPS;

```
.data
a: .word 0x44332211
.text
main:
    instrucció load int
    li $t0, 7
```

Registre R0

## TIPUS DE VARIABLES

TAMANY	C	MIPS
1 byte	char/unsigned char	.byte
2 bytes	short	.half
4 bytes	int	.word
8 bytes	long long int	dword

- Las variables tenen que estar declaradas en adreces multiples de la seva mida.

ex : char a = 0xFF  
 char b = 0xEE  
 char c = 0xDD  
 unsigned long long d = 0x7766554433221100  
 short e = 0xABCD  
 unsigned int f = 0x40302010

a: .byte 0xFF  
 b: .byte 0xEE  
 c: .byte 0xDD  
 d: .dword 0x7766554433221100  
 e: half 0xABCD  
 f: word 0x40302010

0	FF	9	11
1	EE	A	22
2	DD	B	33
3	-	C	44
4	-	D	55
5	-	E	66
6	-	F	77
7	-	(0)	CD
8	00	(1)	AB

## VECTORES

- Inicialitzats  $\Rightarrow$  v. .half 2,-1,5,0,3
- Sense inicialització :
  - .data
  - a: .byte 'D'
  - .align 2 (dejas  $2^n$  espacios)
  - v: space 8 (dejas n espacios)

## MODOS DE DIRECCIONAMIENTO

Maneras en las que se especifica un operando en un registro en una instrucción.

### 1. MODO REGISTRO

- Operando está en un registro (suma: addu rd, rs, rt  $\Rightarrow$  rd = rs + rt)
- Registros temporales \$t0 - \$t9
- Registros de 32 bits

### 2. MODO INMEDIATO

- El operando se codifica en la propia instrucción  $\rightarrow$  valor inmediat 16 bits
- 16 bits en ca2  $\rightarrow$  necesitaremos extensión de signo (addiu rt, rs, imm16  $\Rightarrow$  rt = rs + signExt)
- Se puede hacer extensión de signo o extensión de 0. (lui rt, imm16)

ex : f = (g+h) - (i-100) tenim f:\$t0, g:\$t1, h:\$t2, i:\$t3  
 addu \$t4, \$t1, \$t2 = g+h  
 addiu \$t5, \$t3, -100 = i-100  
 subu \$t0, \$t4, \$t5 = (g+h) - (i-100)

## 3. MODO MEMORIA

- o Solo load y store
  - Load  $\Rightarrow$  leer memoria ( $lw rt, off16(rs)$   $rt = M_w[rs + \text{signExt}(off16)]$ )
  - Store  $\Rightarrow$  escribir en memoria ( $sw rt, off16(rs)$   $M_w[rs + \text{signExt}(off16)] = rt$ )

ex.  $\$t2 = 1001\ 0000$ 

@

0x 1001 0000	0x 11
1	0x 22
2	0x CC
3	0x DD

- $lb \$t1, 1(\$t2) \Rightarrow \$t1 = 0x 0000\ 0022$
- $lb \$t1, 2(\$t2) \Rightarrow \$t1 = 0x FFFFFFCC$
- $lh \$t1, 0(\$t2) \Rightarrow \$t1 = 0x 0000\ 2211$
- $lh \$t1, 2(\$t2) \Rightarrow \$t1 = 0x FFFF\ DDCC$
- $sb \$t1, 1^{(2)}(\$t2) \Rightarrow M_b[\$t2+1] = 0x CC \Rightarrow \$t2 = 0x 11CCCCDD$

## 3.1. EXTENSIÓN DE O EN L Y S

- o Loadhalf unsigned :  $(lh u rt, off16(rs))$ 
  - copia un halfword (2 bytes) de la memoria als 16 bits de menor pes de  $rt$ .
- o Loadbyte unsigned :  $(lb u rt, off16(rs))$ 
  - copia 1 byte en  $rt$

## REPRESENTACIÓN DE NATURALES

	$a_2$	$a_1$	
4	0100	0100	
-3	1101	1100	$\rightarrow 2^3 - 1 = 7$ (y el signo)
Rang	$[-2^{n-1}, 2^{n-1}+1]$	$[-2^{n-1}+1, 2^{n-1}-1]$	

 $a_1$ 

- o Si el decimal és positiu  $\Rightarrow$  normal
- o Si el decimal és negatiu  $\Rightarrow 2^n - 1$
- o 2 interpretacions pel : 0000 / 1111

## SIGNE I MAGNITUD

- o Positiu  $\Rightarrow$  normal  $a_2$
- o Negatiu  $\Rightarrow$  el número en positiu en binari i ú canvies el signe ( $1011 = -3$ )
- o Rang  $[2^{n-1}+1, 2^{n-1}-1]$

## EXCES K

$$K = 7$$

$$4 \Rightarrow 1011 \quad (4+7=11)$$

$$-3 \Rightarrow 0100 \quad (-3+7=4)$$

$$\text{Rang } [-K, 2^n - K - 1]$$

## CÓDIGO ASCII

codificació de caracters

- o ASCII de 7 bits = 128 símbols d'ordre alfabètic mayus → minus

$$'A' = 'a' - 32$$

## FORMAT INSTRUCCIONS MIPS

Instruccions de 32 bits

Format	6bits	5bits	5bits	5bits	5bits	6 bits
Registres	opcode	rs	rt	rd	shamt	funct.
Immediate	opcode	rs	rt		imm16	
Jump	opcode			target		

ex:

- addu \$t4, \$t3, \$t5
- addiu \$t7, \$t6, 25
- lw \$t3, 0(\$t2)

## PUNTEROS

- o Variable (word) que conté una direcció de memoria
- o Si p té la direcció de memoria de v → p apunta a v

## DECLARACIÓ

```
.data
p1: .word 0
p2: .word 0
p3: .word 0
```

## INICIATZACIÓ

```
C;
char a = 'A'
char b = 'B'
char *p1 = &a;
void func() {
    char *p2 = &a;
    p1 = &b;
}
```

MIPS;

```
.data
a: .byte 'A'
b: .byte 'B'
p1: .word a
func:
    la $t0, a      $t0 = @a
    la $t1, b      $t1 = @b
    la $t2, p1      $t2 = @p1
    sw $t1, 0($t2)  p1 = @b
```

## ACCÉS A PUNTERS

Per accedir a la direcció de memòria que apunta el punter

```
C;
char a = 'A'
char *p1 = &a
void func() {
    char tmp = *p1 // tmp = 'A'
}
```

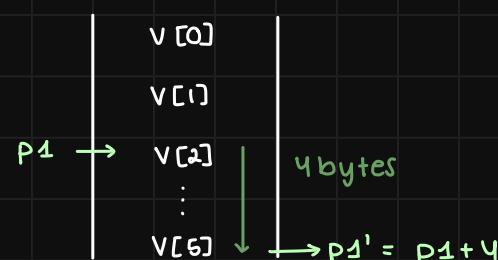
MIPS;

```
.data
a: .byte 'A'
p1: .word a
func:
    la $t0, p1      $t0 = @p1
    lw $t1, 0($t0)  $t1 = p1 = @a
    lb $t2, 0($t1)  tmp = *p1 = 'A'
```

## ARITMÉTICA DE PUNTERS

Suma d'un punter  $p + N$  = un altre punter  $q$ . On  $q$  apunta a lo que apuntava  $p$  però  $N$  elements delante.

`int * p1 = @vec[2]`



## VECTORS

En MIPS s'han de respectar les alineacions de les dades.

C:

`short vec[5] = {0,-1,2,-3,4}`

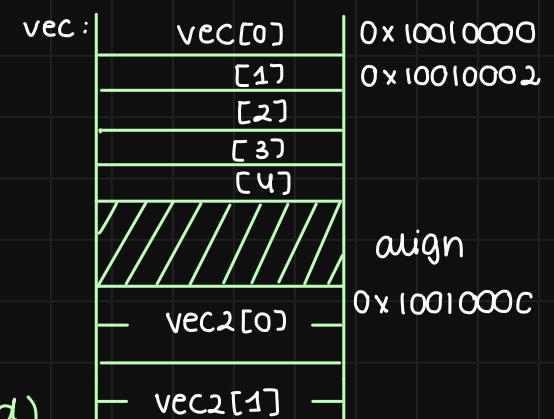
`int vec2[100]`

MIPS:

`vec: . half 0,-1,2,-3,4`

`. align 2  $2^2 = 4$`

`vec2: . space 400 100 int són 400 bits (word)`



## LITTLE ENDIAN

Norma que diu que en la memòria primer es guarden els bits de menor pes

## ACCÉS ALEATORI A UN ELEMENT D'UN VECTOR

Element de  $T$  bytes

Fórmula:  $[@vec[i]] = @vec[0] + (i \cdot T)$  o calcular la direcció de memòria de  $v[i]$

ex: `int val[100], vec[100]; int elem`

① `elem = val[5] + vec[10]`

```
. data
.align 2
val: . space 400
vec: . space 400
elem:.word 400
.text
la $t0, val+20      // val[5]
lw $t1, 0($t0)       // $t1 = *val[5]
la $t2, vec          // vec[10]
lw $t3, 40($t2)      // $t3 = *vec[10]
addu $t1, $t1, $t3
la $t2, elem
sw $t1, 0($t2)
```

② `elem = vec[10 + val[5]]`

```
. data
.align 2
val: . space 400
vec: . space 400
elem:.word 400
.text
la $t0, val+20
lw $t1, 0($t0)
addiu $t1, $t1, 10   $t1 = 10 + *val[5]
sll $t1, $t1, 2      $t1 = $t1 * 4
la $t0, vec
addu $t0, $t0, $t1    @vec[$t1]
```

o Un vector és un punter al primer element de vec.  $\rightarrow p = \text{vec}, p[i], * \text{vec} \Rightarrow *(p+i) = 0 \Rightarrow p[i] = 0$

## STRING

Últim caracter del string té que ser  $\emptyset$  (= "\0" en ASCII) centinella

## DECLARACIÓ

`.ascii "Hello world"` }  $\Rightarrow$  `.asciz "Hello world"`

## ACCÉS

$[@string[i]] = @string[0] + i$

•  
•  
•  
T  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12

### *Section Three*

## *Traduccio de programes*

---

[BACK TO INDEX](#)

## INSTRUCCIONS MIPS

### DESPLAÇAMENTS LÒGICS

- o shift left logical // shift right logical → desplaça a la dreta / esquerra el valor immediat en shamt  
sll // srl rd, rt, shamt

### DESPLAÇAMENTS ARITMÉTICS

- o Extensió de signe al resultat  
sra rd, rt, shamt

### DESPLAÇAMENT INDICAT EN REGISTRE

- o Cada cop que desplaça a la dreta multiplica per  $2^n$ .
- o Cada cop que desplaça a la esquerra divideixes per  $2^n$

sllv rd, rt, rs  
 sriv rd, rt, rs  
 sra v rd, rt, rs } ⇒ només s'utilitzen els 5 bits de menys pes de rs.

- o CONVERSIÓ  $ca2 \rightarrow can$  → si  $x$  negatiu : resta 1  
 si  $x$  positiu : igual

ex:  $a = (a \ll b) \gg 2$ ;  $a = \$t0, b = \$t1$   
 sllv \$t2, \$t0, \$t1  
 sra \$t0, \$t4, 2

### OPERACIONS C vs MIPS

- o AND:  $c = a \& b$ ; and \$t2, \$t0, \$t1 (andi)
- o OR:  $c = a | b$ ; or \$t2, \$t0, \$t1 (ori)
- o XOR:  $c = a \oplus b$ ; xor \$t2, \$t0, \$t1 (xori)
- o NOR:  $c = \sim a$ ; nor \$t2, \$t0, \$zero
- o NOT:  $b \neq a$ ; not \$t2, \$t0, \$zero

ex:  $a \sim (a \& b)$

and \$t2, \$t0, \$t1  
 nor \$t2, \$t2, \$zero

### COMPARACIÓN

- o  $\leq$  slt rd, rs, rt / slti rd, rs, imm16 }
- o  $\geq$  sltu rd, rs, rt / sltiu rd, rs, imm16 } ( $rd = rs < rt / imm16$ )
- o AND → lògica  $\&$   $\Rightarrow 0/1$   
 bit a bit  $\&$   $\Rightarrow tot$
- o OR → lògica  $\|$   $\Rightarrow 0/1$   
 bit a bit  $\|$   $\Rightarrow tot$

### SALTOS

- o beg rs, rt, label    si  $rs == rt$      $PC = PC_{up} + sext(offset16 \cdot 4)$
- o bne rs, rt, label    si  $rs \neq rt$
- o MACRO: b label     $PC = PC_{up} + sext(offset16 \cdot 4)$

direcció de la següent instrucció

distància a saltar des de la PC

- o MACRO : blt / bltu → si rs < rt ("less than")
- o MACRO : bgt / bgtu → si rs > rt ("greater than")
- o MACRO : bge / bgeu → si rs ≥ rt
- o MACRO : ble / bleu → si rs ≤ rt

### MULTIPLICACIÓ i DIVISIÓ

- o mult rs, rt ⇒ \$hi : \$lo → On es guarda rs \* rt (el mateix passa quan dividim, en un d'aquests registres es guarda el resultat i a l'altre el residu)
- o mflo rd ⇒ \$rd ← \$lo
- o mfhi rd ⇒ \$rd ← \$hi

### SALTOS INCONDICIONALS

- o j target
- o jr rs
- o jal target
- o jalr rd, rs (PC = rs)

### SENTENCIAS ALTERNATIVA

- o if (condició) → si la part de l'esquerra determina el resultat no cal evaluar el de la dreta.
- o then
- o else
- o while → evaluar condició, saltar a fiwhile si es falsa (blt), operació si és certa, b while, fiwhile.
- o for → es tradueix com un while

### SUBRUTINES

- o Estructura de programació (acció // funció)
- o Amb retorn de resultat (excepte voids)

### TRUCADA i RETURN

- o Hi ha que utilitzar les funcions jal // jr

ex: aux :

    jr \$ra   salta a @ guardada a \$ra

main :

    jal aux   salta a aux y \$ra = PC+4

- o Els paràmetres pasen en els registres \$a0 - \$a3

- o El resultat es retorna al registre \$v0

### SUBRUTINAS MULTINIVELL

- o Subrutines que criden a altres subrutines
- o Dades necessàries → Paràmetres \$a0 - \$a3
  - Direcció retorn \$ra
  - Punter pila \$sp
  - calculs intermitjos (registres o pila)

Quan una subrutina s'acaba cal restaurar els valors dels registres segurs (\$s0 - \$s3).

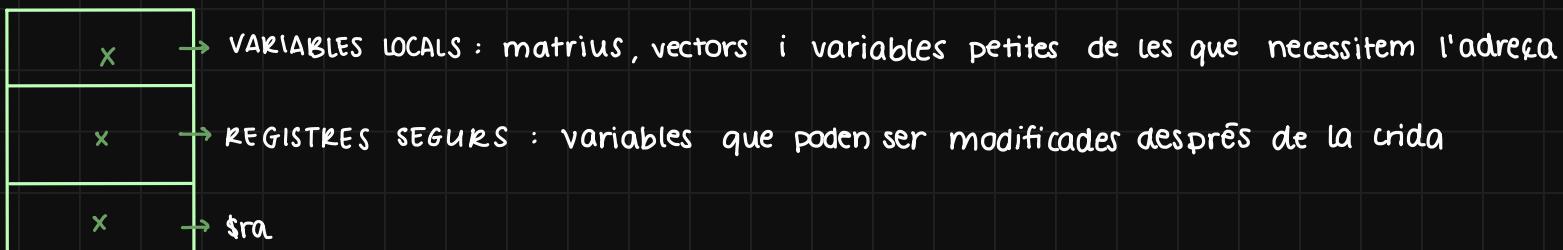
- o Has de saber quines dades voldràs salvar abans de realitzar la subrutina i s'utilitzaran després.
  - 1) Salvas al inici de la subrutina (a la pila)
  - 2) Restaures al final de la subrutina

**LA PILA****ESTRUCTURA BLOC D'ACTIVACIÓ**

- o variables locals
- o variables iniciais de registres segurs → només es guarden els necessaris

**NORMAS BLOC D'ACTIVACIÓ**

- o Posició: variables locals, registres segurs
- o Ordenació: variables locals en ordre de declaració
- o Alineació: variables locals respecten l'alineació dels registres segurs alineats en @ múltiples de 4.  
El tamany de la BD és múltiple de 4 sempre

**CONSTRUCCIO**

```
addiu $sp, $sp, -Tamany # reserves espai  
sw ... # per guardar el valor actual dels registres segurs
```

**RESTAURACIÓ**

```
lw... # per recuperar els valors dels registres  
addiu $sp, $sp, Tamany # restaura l'espai
```

•  
•  
•

T

1

2

3

4

5

6

7

8

9

10

11

12

## *Section Four*

---

[BACK TO INDEX](#)

*Matrius*

## MATRÍUS

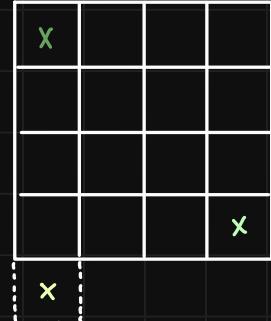
- Les matrius es guarden d'esquerra a dreta per files

- Tipus d'accisos:

$$M[i][j] = \&M + i \cdot NC \cdot Tam + j \cdot Tam$$

↑ num columnes

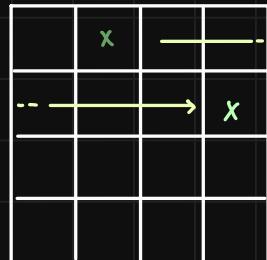
- Accés aleatori: calcula la formula de la pos. de la qual vols l'adreça
- Accés seqüencial (necessites una manera de parar: A) Últim ELEMENT DE N  
B) L'ELEMENT SEGUENT DEL FINAL C) AMB LA i )  $\Rightarrow$  amb for's o do while's.



- Primera pos. a la que vull accedir (no ha de ser  $M[0][0]$ )
- Últim element
- Element seguent  
Utilitzem el stride per moure's per N.

## STRIDE

- STRIDE: ex: tenim  $M[i][2i+1]$ , int  $M[4][4]$



```
for (int i=0; i<4; i++)
    o Primera iteració i = 0
         $M[i][2i+1] \Rightarrow M[0][1]$ 
    o Segona iteració i = 1
         $M[1][3]$ 
```

$$\frac{M[1][3] - M[0][1]}{2}$$

★ STRIDE:  $\rightarrow M[1][2] \leftarrow$

El calculem:

$$M[0][1] = \&M + 0 \cdot 4 \cdot 4 + 1 \cdot 4 = \&M + 4$$

$$M[1][3] = \&M + 1 \cdot 4 \cdot 4 + 3 \cdot 4 = \&M + 16 + 12 = \&M + 28$$

- Si quiero ir de  $\rightarrow$

la \$t0, M # \$t0 = @M[0][0]

addiu \$t1, \$t0, 4 # \$t1 = @M[0][1]

addiu \$t1, \$t1, 24 # \$t1 = @M[1][3]

$$\frac{\&M + 28 - \&M + 4}{24} \rightarrow \text{STRIDE}$$

•  
•  
•  
T  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12

## *Section Five*

### *Memoria cache*

---

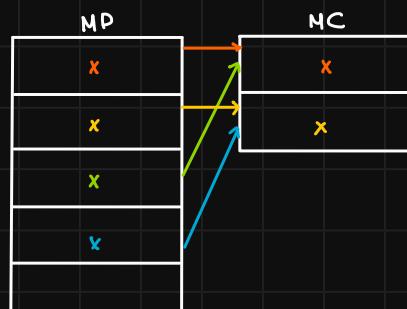
[BACK TO INDEX](#)

## MEMORIA CACHE

- \* TAG / etiqueta → matrícula per identificar la dada
- \* Quan sol·licitem una dada es copiarà a la NC tot el bloc a la que aquesta pertanyi.
- \* El número de bloc de la MP =  $\frac{\text{adreça}}{\text{TAM BLOC}}$   $\rightarrow 2^t$  on t es els bits del offset

## TIPUS D'ASSOCIATIVITAT

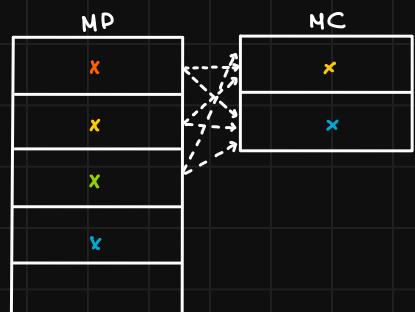
## CORRESPONDÈNCIA DIRECTA



TAG | # lineaNC | offset  
 \* Per saber on es trova una dada a la NC  
 $\Rightarrow \text{Línia NC} = \text{Líneas MP} (\text{mod Líneas NC})$

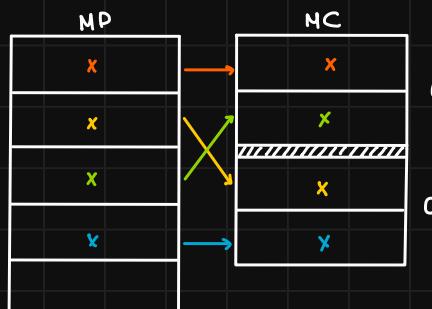
\*  $2^l$  líneas de N → els l bits de menys pes determinen el bloc a MP

## COMPLETAMENT ASSOCIATIVA



\* Cualsevol bloc de la MP pot anar a qualsevol bloc de la NC.  
 TAG \* Byte

## ASSOCIATIVA DE N-VIAS



C1 \* La NC es divideix en conjunts de x-blocs  
 TAG \* conjunt \* Byte

C2

## ALGORISMS DE REEMPLACAMENT

- \* Aleatori
- \* FIFO : firs input firs output , surt el primer que ha entrat
- \* LRU : last recent used , surt el que porta més temps sense ser usat.

## ESCRITURA i LECTURA

		Escriptura immediata sense assignació	Escriptura immediata amb assignació	Escriptura retardada amb assignació
Lectura	Encert	lecMC(byte)	lecMC(byte)	lecMC(byte)
	Fallada	blocMP->MC i lecMC(byte)	blocMP->MC i lecMC(byte)	si DBremp = 0 blocMP->MC i lecMC(byte) si DBremp = 1 bloc_reempMC->MP i blocMP->MC i lecMC(byte) i DB = 0
Escriptura	Encert	escMC(byte) i escMP(byte)	escMC(byte) i escMP(byte)	escMC(byte) i DB = 1
	Fallada	escMP(byte)	blocMP->MC i escMC(byte) i escMP(byte)	si DBremp = 0 blocMP->MC i escMC(byte) i DB = 1 si DBremp = 1 bloc_reempMC->MP i blocMP->MC i escMC(byte) i DB = 1

## TEMPS

- Hit ratio  $h = \frac{\text{num\_encerts}}{\text{num\_acessos}}$
- Miss ratio  $m = 1 - h = \frac{\text{num\_fallades}}{\text{num\_acessos}}$
- $t_{acces} = th + tp$

\* On el tp es  $\downarrow$  (temps de pena útzació)

tp		Escriptura immediata sense assignació	Escriptura immediata amb assignació	Escriptura retardada amb assignació
Lectura	Encert	0	0	0
	Fallada	tblock + th	tblock + th	si DBremp = 0 tblock + th si DBremp = 1 2*tblock + th
Escriptura	Encert	0*	0*	0
	Fallada	0*	tblock + th	si DBremp = 0 tblock + th si DBremp = 1 2*tblock + th

•  
•  
•

T

1

2

3

4

5

6

7

8

9

10

11

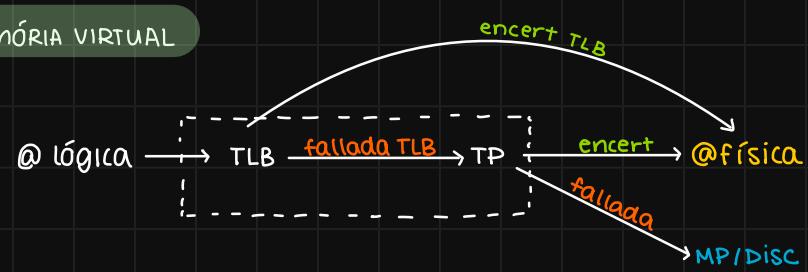
12

## *Section Six*

# *Memoria virtual*

---

[BACK TO INDEX](#)

**MEMÓRIA VIRTUAL**

\* TP representa una mini cache del disc

\* VPN virtual page number

\* PPN phisical page number

@lógica      **VPN**      offset

@física      **PPN**      offset

\* La TP té tantas entrades com VPN ens representen la MP