

Estructura de Computadors

Grau en Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Tema I: El processador



Objectius

- Contextualitzar l'assignatura
 - ✓ Conéixer el context de l'arquitectura MIPS32
 - ✓ Conéixer los aspectes més generals de l'arquitectura MIPS32
 - ✓ Aprendre el cicle d'execució del processador
- Conéixer el disseny de la ruta de dades basada en multiplexors
- Conéixer el disseny de la unitat de control del processador

} Introduatori (Tema 0)

Contingut i bibliografia

- I – Arquitectura MIPS32

- ✓ Introducció i definició
- ✓ Característiques bàsiques
- ✓ Exemple d'execució

Introductori
(Tema 0)

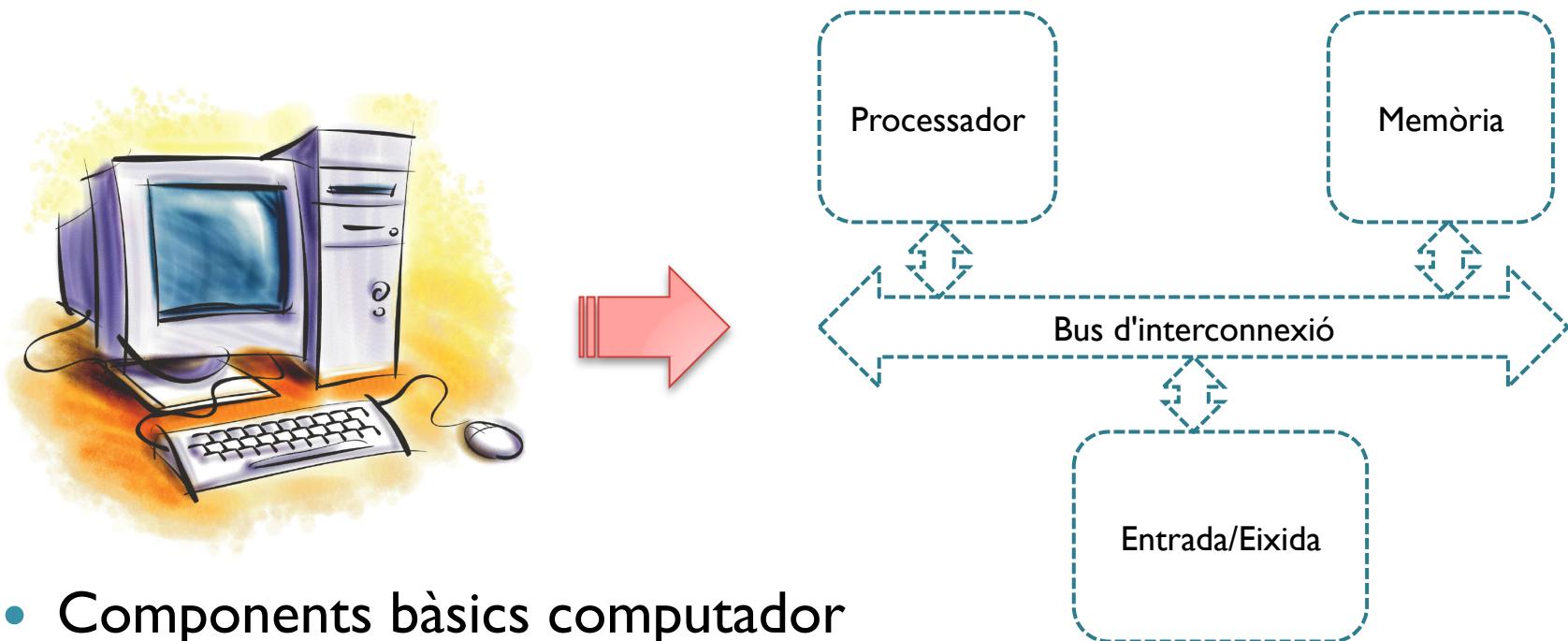


- 2 – La ruta de dades

- ✓ Etapes de búsqueda y decodificación
- ✓ Disseny de la ruta per a instruccions aritmètiques de tipus R
- ✓ Disseny de la ruta per a instruccions aritmètiques de tipus R+I
- ✓ Disseny de la ruta per a instruccions lw/sw
- ✓ Disseny de la ruta per a instruccions de salt beq/bne

Bibliografía: Patterson, D.A., Hennessy, J.L., “Estructura y diseño de computadores. La interfaz hardware-Software,” 4a edición, Ed. Reverté, 2011, Cap 4 (4.1 – 4.4)

Arquitectura MIPS32: introducció



- Components bàsics computador
 - ✓ Processador
 - ✓ Memòria
 - ✓ Entrada/Eixida
 - ✓ Interconnexió

Concepte de programa emmagatzemat:

- Les instruccions es representen com a nombres
- Els programes s'emmagatzemen en la memòria per tal de ser llegits o escrits també com a nombres
- El programa interactua amb el món exterior mitjançant dispositius d'Entrada/Eixida

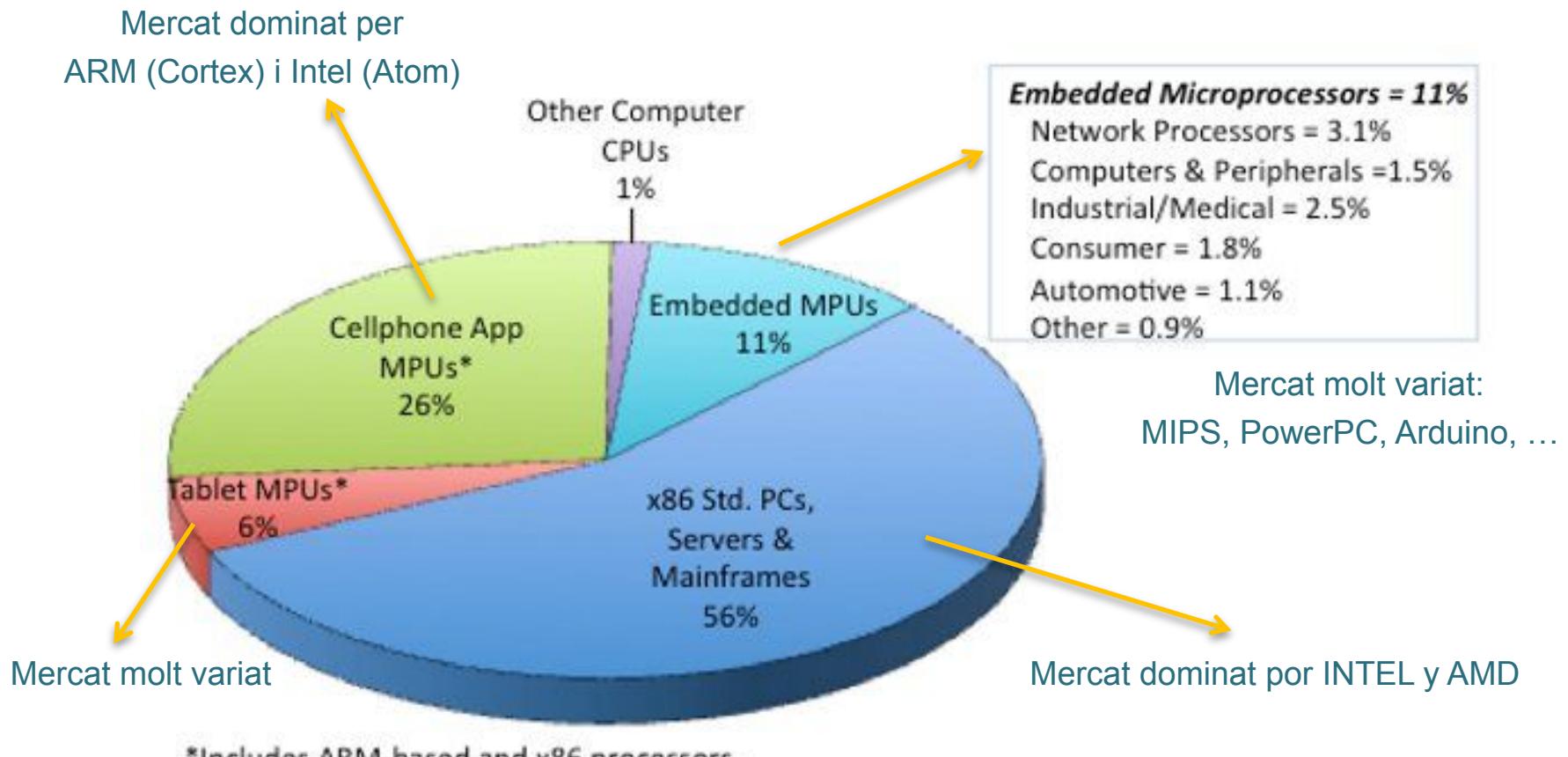
Arquitectura MIPS32 seleccionada com a base d'aprenentatge de l'assignatura

Introducció: Tipus de computadors

| Tipus | Potència | Ample paraula (bits) | Consum | Aplicacions |
|--|---------------|----------------------|------------|--|
| Mainframes i Supercomputers | Molt alta | 64, 128 | Molt alt | Gestió corporacions i governs. Computació massiva en centres d'investigació |
| Workstations | Alta | 32, 64 | Alt | Disseny, simulació, control: Empresses, Universitats |
| Desktops Laptops Netbooks | Alta-Mitjana | 32, 64 | Mitjà Baix | Informàtica en general Computadors personals |
| Tablet | Mitjana | 32,64 | Molt baix | Informàtica personal |
| Embedded | Mitjana-alta. | 16, 32, 64 | Mitjà Baix | Sistemes encastrats per a: Comunicacions (routers, switches). Electrodomèstics, automoció, perifèrics, sistemes industrials,... |
| Cellphones | Mitjana-baixa | 16, 32 | Molt baix | Telèfons mòbils |

Introducció: El mercat dels processadors

2015



Todo procesador se define de acuerdo con una **ARQUITECTURA**

Introducció: Arquitectura

- Arquitectura (ISA: *Instruction Set Architecture*)
 - ✓ Fa referència al repertori d'instruccions, registres, model d'excepcions, maneig de la memòria virtual, mapa d'adreces físiques i d'altres
 - ✓ Tot allò que el programador ha de saber de la màquina
- Exemples d'arquitectures:
 - ✓ x86, x86-64, IA-32, INTEL®64 -> AMD, INTEL
 - ✓ MIPS-32, MIPS-64 -> Mips Tech.
 - ✓ DEC Alpha Architecture -> Digital Equipment Corp.
 - ✓ Power , PowerPC -> Appel, IBM, Motorola
 - ✓ ARM Architecture -> ARM (Advanced RISC Machines Holdings)
- Les arquitectures poden evolucionar, donant lloc a versions de 16, 32 o 64 bits, sempre compatibles entre sí.

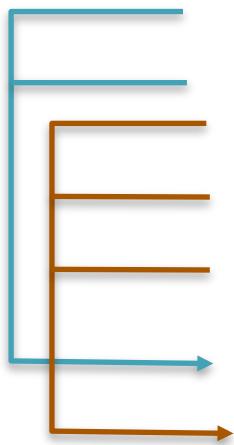
Introducció: Implementació

- **Implementació:** Hace referencia a las características concretas de los circuitos que conforman el diseño de un procesador que ejecuta una arquitectura.
 - A veces, con una implementación concreta se crean familias de modelos, como las ‘microarquitecturas’, de INTEL

| Arquitectura | Famílies / Models | | Fabricants |
|-----------------|-------------------------------------|---|------------|
| MIPS | R2000, R3000, R4000,...., R16000 | | MIPS Tech. |
| IA-32, Intel 64 | Microarquitectura: P5, P6 | Models: Pentium II, III, Pro | INTEL |
| | NetBurst | Pentium 4, Pentium D, Celeron D | |
| | Pentium M | Core Duo, Core Solo, Pentium M, Celeron M | |
| | Intel Core | Core 2 Duo, Quad, Extreme, Xeon 5xxx, 7xxx, Celeron dual-core | |
| | Atom | Atom | |
| | Core i7 | i3, i5, i7 | |
| X86, x86-64 | K5,K6,K8 | Athlon 64, Phenom, Turion, Sempron, Opteron | AMD |
| ALPHA | 22064, 21164, 22164, 23164 | | DEC |

Arquitectura MIPS

- **MIPS: Microprocessor without Interlocked Pipeline Stages)**
 - ✓ Processador RISC (**R**educed **I**nstruction **S**et) desenvolupat en la Universitat de Stanford per John L. Hennessy amb la ruta de dades segmentada.
 - ✓ Comercialitzat per MIPS Technologies (Silicon Graphics International)
 - ✓ Arquitectura senzilla → Muy utilitzada per les universitats per a docència
 - ✓ El disseny segmentat del MIPS és el precursor de la majoria dels processadors RISC posteriors



| Versió del joc d'instruccions (ISA) | Ample paraula | Processadors |
|-------------------------------------|---------------|---------------|
| MIPS I | 32 | R2000, R3000 |
| MIPS II | 32 | R6000 |
| MIPS III | 64 | R4000 |
| MIPS IV | 64 | R5000, R10000 |
| MIPS V | 64 | - |
| MIPS32 | 32 | 4K |
| MIPS64 | 64 | 5K |

Implementacions del MIPS

- ✓ Els diferents models del MIPS, R2000, R3000, ... R10000 funcionaren com a CPU dels computadors venuts per SGI, Olivetti, Siemens, etc..
 - Desaparegueren d'un mercat dominat per Intel i AMD.
- ✓ En la dècada dels 90 *MIPS Technologies* va licenciar el disseny del MIPS en forma de dues arquitectures: MIPS32 i MIPS64, i els fabricants els utilitzen en microcomputadors i sistemes encastrats.
 - En Xina hi ha el processador *Loongson* amb el què fan portàtils i supercomputadors
- ✓ En 2012, *Imagination Tech.* va adquirir *MIPS Tech.* Des d'aleshores els dissenys basats en MIPS s'han distribuit àmpliament en forma d'IP-cores en el mercat dels sistemes encastrats i SoC: Cisco, Sony, NEC, Microchip, Toshiba,....

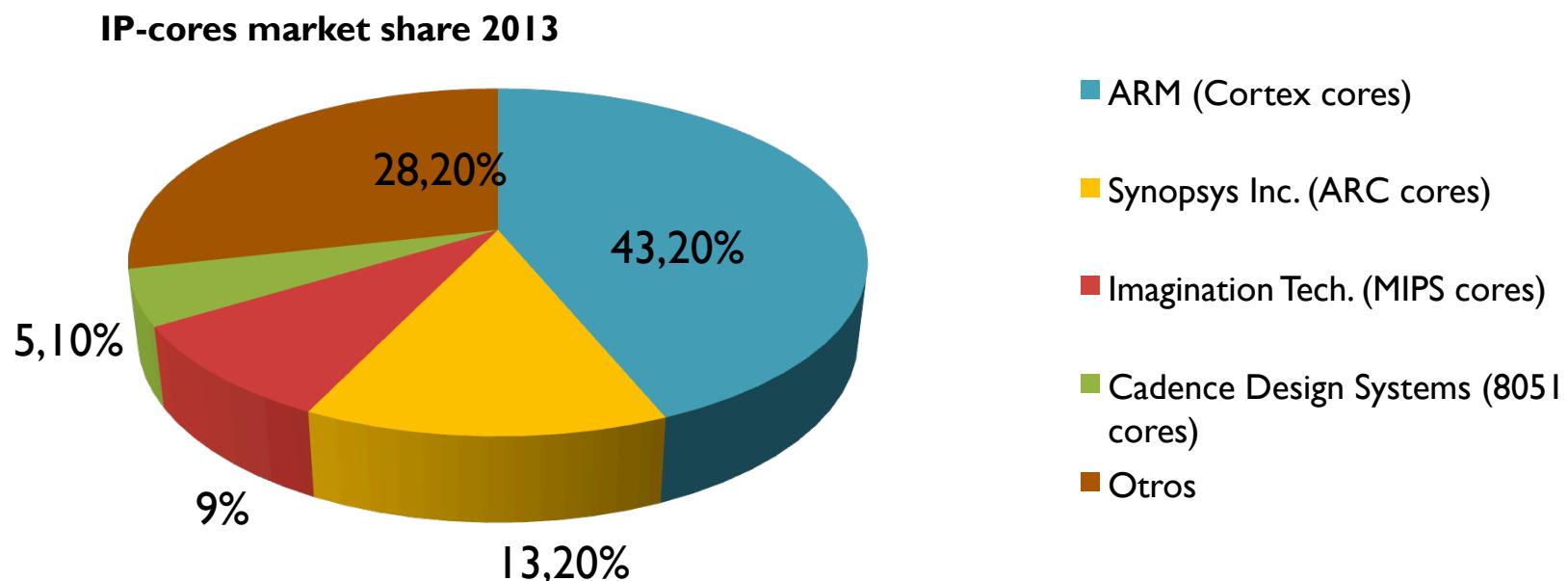
IP-core (Intellectual Property core): Bloc lògic o circuit que és propietat intel·lectual d'una companyia que el ven (el "llicència") als fabricants perquè l'incloguen en els seus dissenys FPGA (*field-programmable gate array*) o ASIC (Applicacion Specific Integrated Circuit)

Embedded System: Sistema computador encastrat en una màquina i dissenyat per a fer unes funcions específiques, usualment de temps real.

SoC (System On Chip): Circuit integrat que integra tots els components d'un computador o sistema electrònic en un únic xip.

Implementaciones del MIPS

- Los diferentes modelos (cores) del MIPS se pueden ver en:
 - ✓ https://en.wikipedia.org/wiki/List_of_MIPS_microarchitectures
- ✓ El mercado de los IP-cores en 2013 se distribuía como indica la figura:

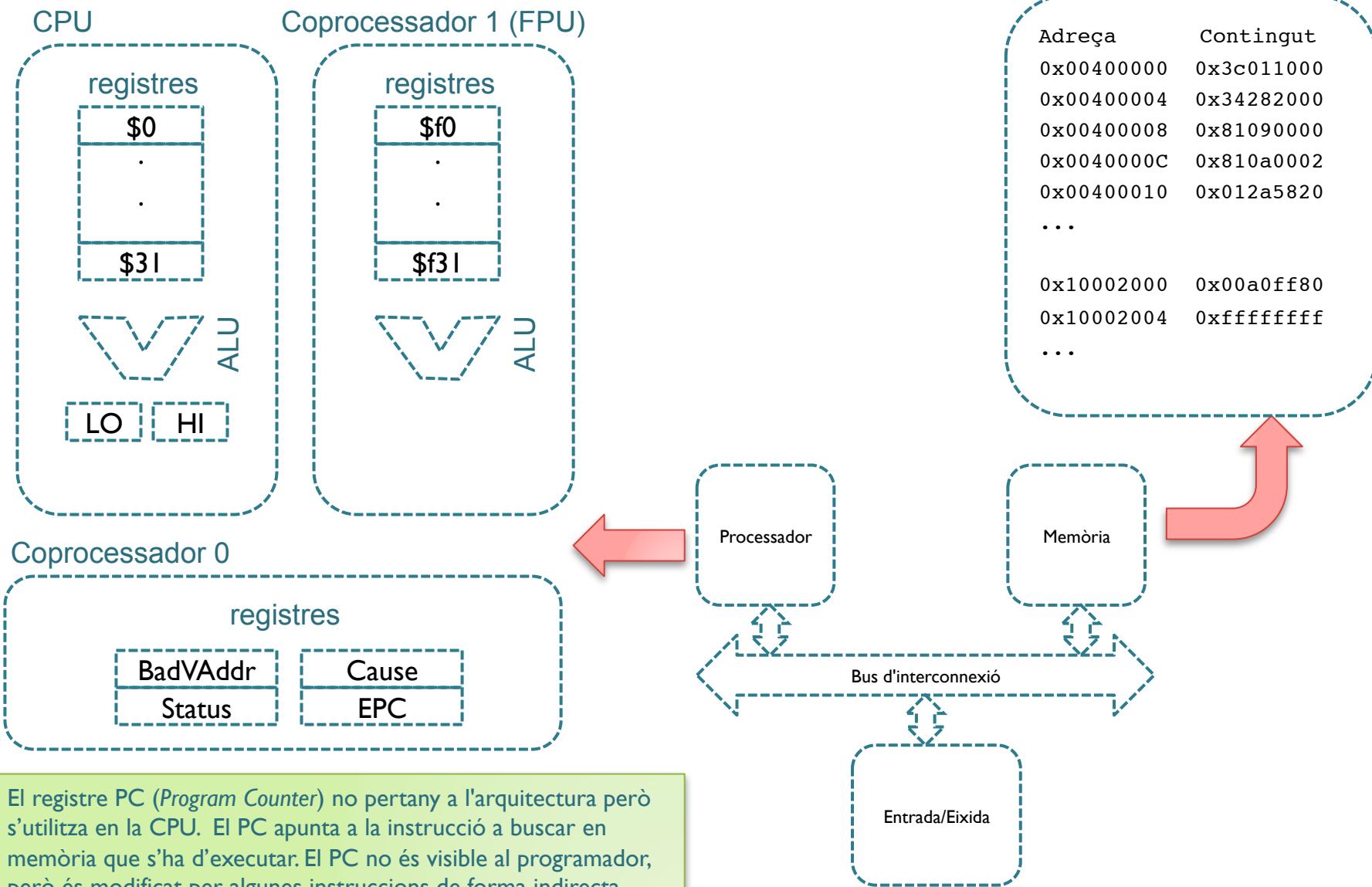


Arquitectura MIPS32: característiques bàsiques

- Màquina RISC (*Reduced Instruction Set Computer*)
- Amplària de paraula i grandària dels bussos de 32 bits
- Principals grandàries de dades en les instruccions:
 - ✓ Byte (B), halfword (H), word (W)
- Arquitectura de càrrega/emmagatzematge (*load/store architecture*)
 - ✓ Instruccions específiques de lectura (càrrega) i escriptura (emmagatzematge) en memòria
 - ✓ Tots els operands d'una instrucció aritmètica es carreguen inicialment en registres o bé estan dins la mateixa instrucció (constants)
 - ✓ Instruccions aritmètiques amb 3 operands de 32 bits en registres
- Modes de funcionament: usuari, nucli (*kernel*) i depuració

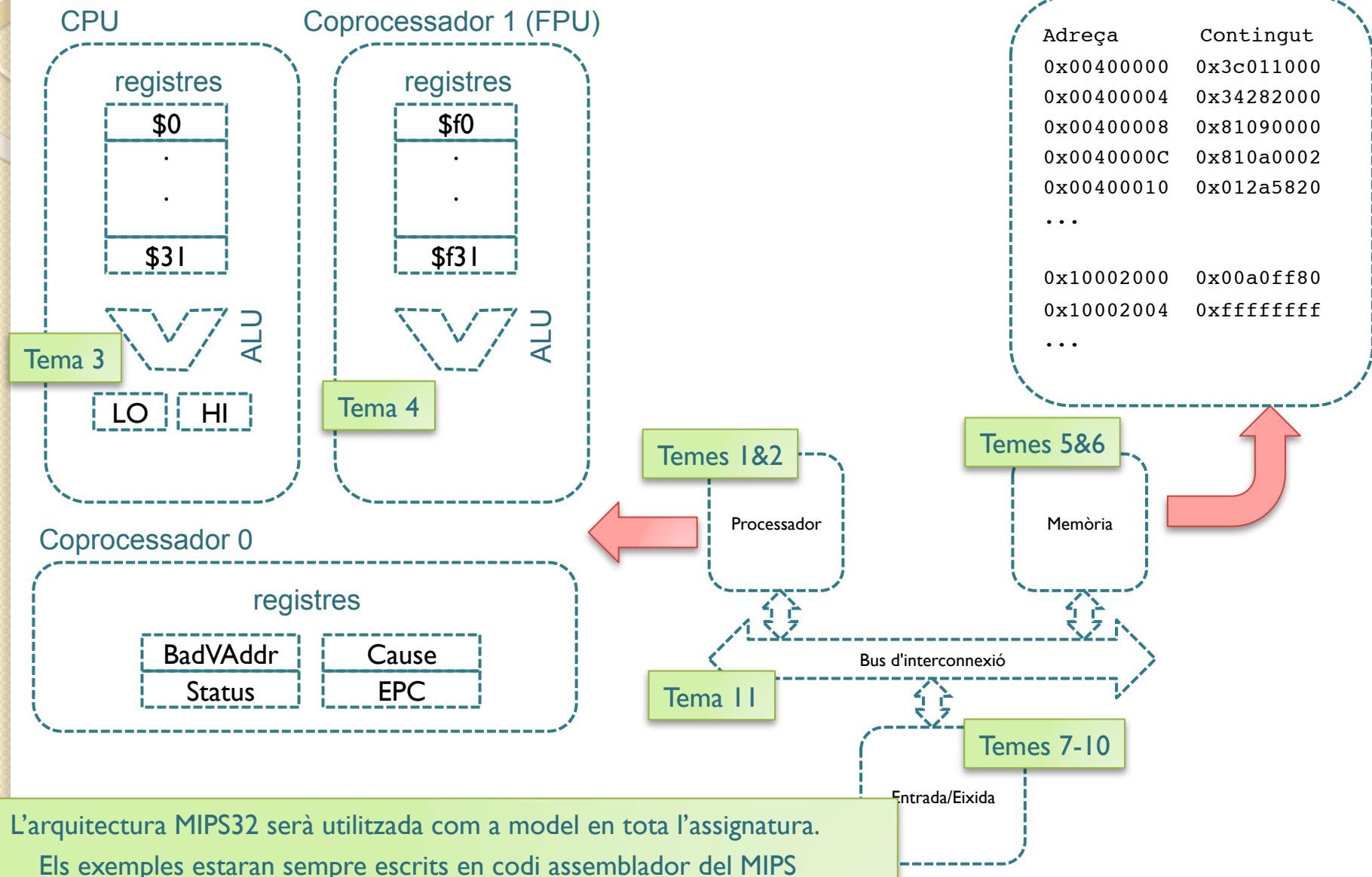
Arquitectura MIPS32: característiques bàsiques

Contingut memòria



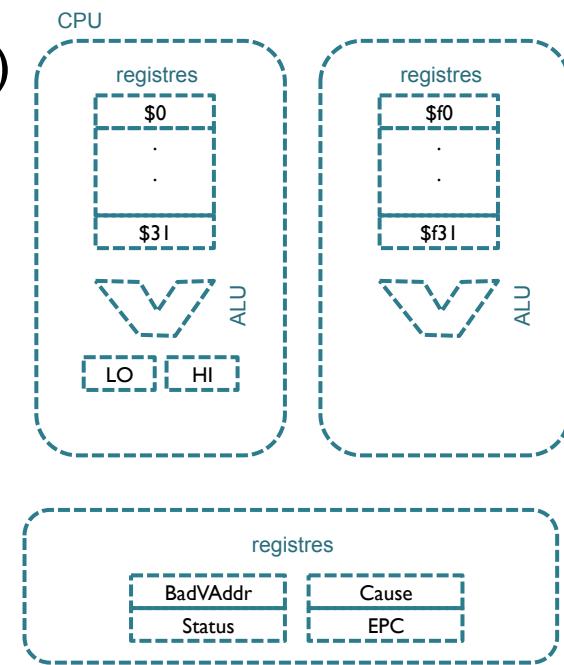
Arquitectura MIPS32: característiques bàsiques

Contingut memòria



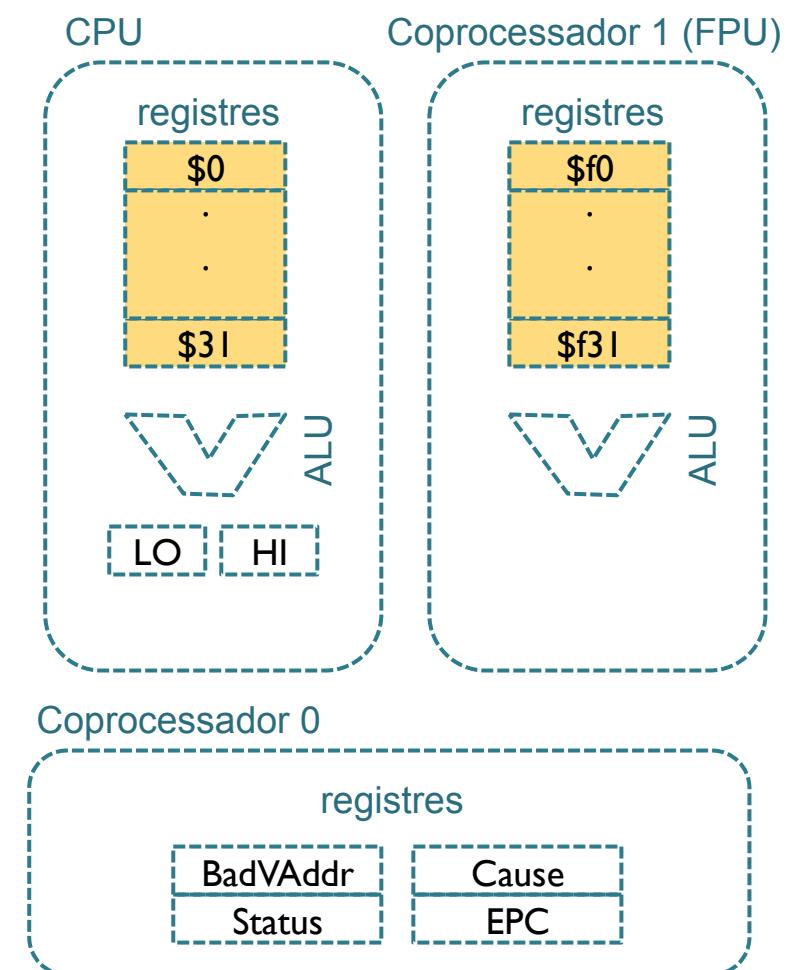
Arquitectura MIPS32: model de programació

- CPU
 - ✓ 32 registres de 32 bits (\$0..\$31)
 - ✓ Registres HI-LO
- Coprocessador 0
 - ✓ Control del sistema (jerarquia, excepcions, modes d'execució, ...)
 - ✓ 4 registres específics (n'hi ha més)
- FPU, *Floating Point Unit* (Coprocessador 1, opcional)
 - ✓ 32 registres de 32 bits (\$f0..\$f31)
 - ✓ Aritmètica coma flotant
 - Simple precisió (32 registres)
 - Doble precisió (16 parelles de registres)

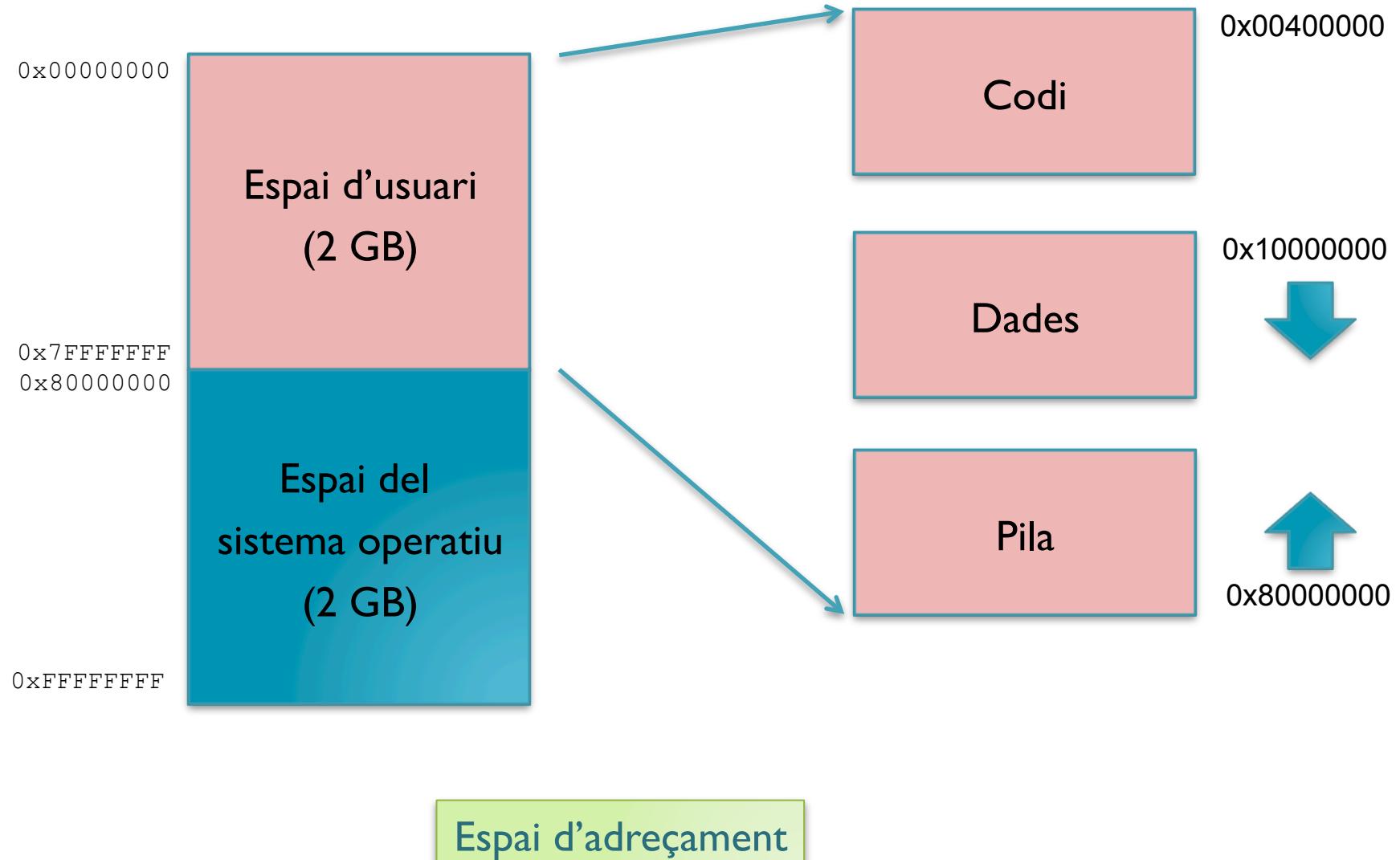


Arquitectura MIPS32: registres

| Nom | Registre | Ús |
|------------|-----------|------------------------------------|
| \$zero | \$0 | Constant 0 |
| \$at | \$1 | Registre temporal de l'assemblador |
| \$v0-\$v1 | \$2-\$3 | Retorn de funcions |
| \$a0-\$a3 | \$4-\$7 | Arguments de funcions |
| \$t0-\$t7 | \$8-\$15 | Registres temporals |
| \$s0-\$s7 | \$16-\$23 | Temporals salvats |
| \$t8-\$t9 | \$24-\$25 | Registres temporals |
| \$k0-\$k1 | \$26-\$27 | Utilitzats pel sistema operatiu |
| \$gp | \$28 | Punter global |
| \$sp | \$29 | Punter de pila |
| \$fp | \$30 | Punter de marc (<i>frame</i>) |
| \$ra | \$31 | Adreça de retorn |
| \$f0-\$f31 | \$0..\$31 | Registres de coma flotant |



Arquitectura MIPS32: memòria



Arquitectura MIPS32: memòria

- Directives de l'assemblador
 - ✓ Directives de segments de memòria
 - .data [adreça]
 - .text [adreça]
 - .end
 - ✓ Directives de reserva de memòria de dades
 - .space n
 - .byte b1 [, b2] ...
 - .half b1 [, b2] ...
 - .word b1 [, b2] ...
 - .ascii cadena1 [, cadena2] ...
 - .asciiz cadena1 [, cadena2]...

Les variables del programa es codifiquen com etiquetes de dades
A = 0x10000000
W = 0x10000004
...

```
.data 0x10000000
A:    .byte 2, 3, 4
W:    .word 33
V:    .space 100

.data 0x10004000
C1:   .ascii "hola"
C2:   .asciiz "hola"

.text 0x00400000
...
.end
```

Arquitectura MIPS32: memòria

- Adreçament

- ✓ Adreçament a nivell de byte
 - Compte! Tot byte de memòria té una adreça pròpia i única
- ✓ Suporta els modes *big-endian* i *little-endian*

- Alineament

- ✓ Byte en qualsevol adreça
- ✓ Mitja paraula (*half*) en adreça múltiple de 2
- ✓ Paraula (*word*) en adreça múltiple de 4
- ✓ Doble paraula (*dword*) en adreça múltiple de 8

- Directiva `.align N`

- ✓ L'objecte següent és alineat a partir de la primera adreça posterior múltiple de 2^N

```
.data 0x10000000
.byte 0x56 # En 0x10000000
.byte 0x34 # En 0x10000001
.align 2
.space 1 # En 0x10000004
```

Arquitectura MIPS32: instruccions

● Grups d'instruccions

- ✓ Transferència entre registres
- ✓ Càrrega/emmagatzematge
- ✓ Aritmètiques
- ✓ Lògiques
- ✓ Comparació
- ✓ Salt
- ✓ Coma flotant
- ✓ D'altres

● Sintaxi i codificació

- ✓ <http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html>
- ✓ http://en.wikipedia.org/wiki/MIPS_architecture

Les instruccions s'estudiaran amb més detall en les sessions de laboratori. L'apèndix conté alguns exemples d'ús.

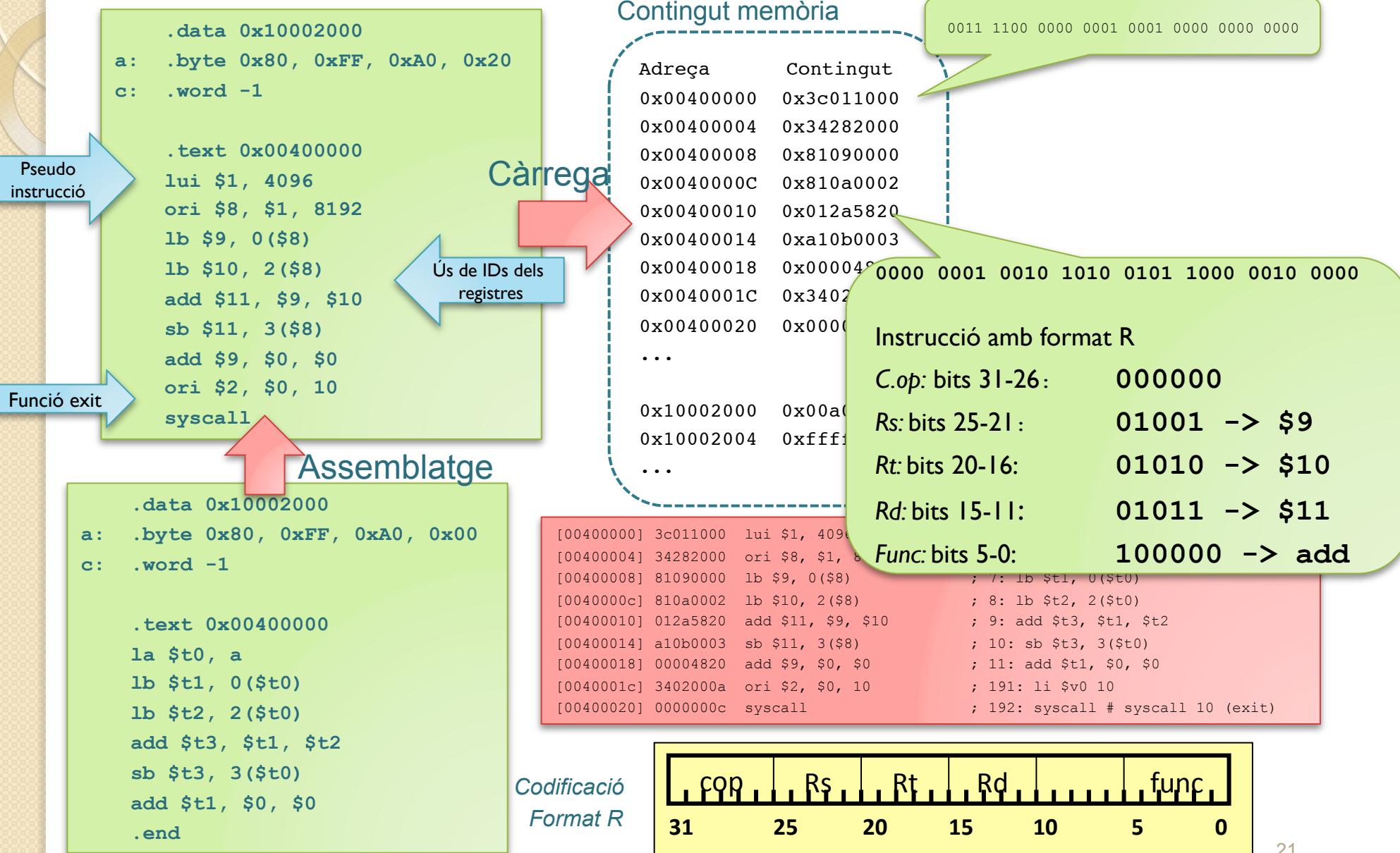
Instruccions vs pseudoinstruccions

Les instruccions són executades pel hardware mentre que les pseudoinstruccions es tradueixen pel programa assemblador en un conjunt d'instruccions equivalent

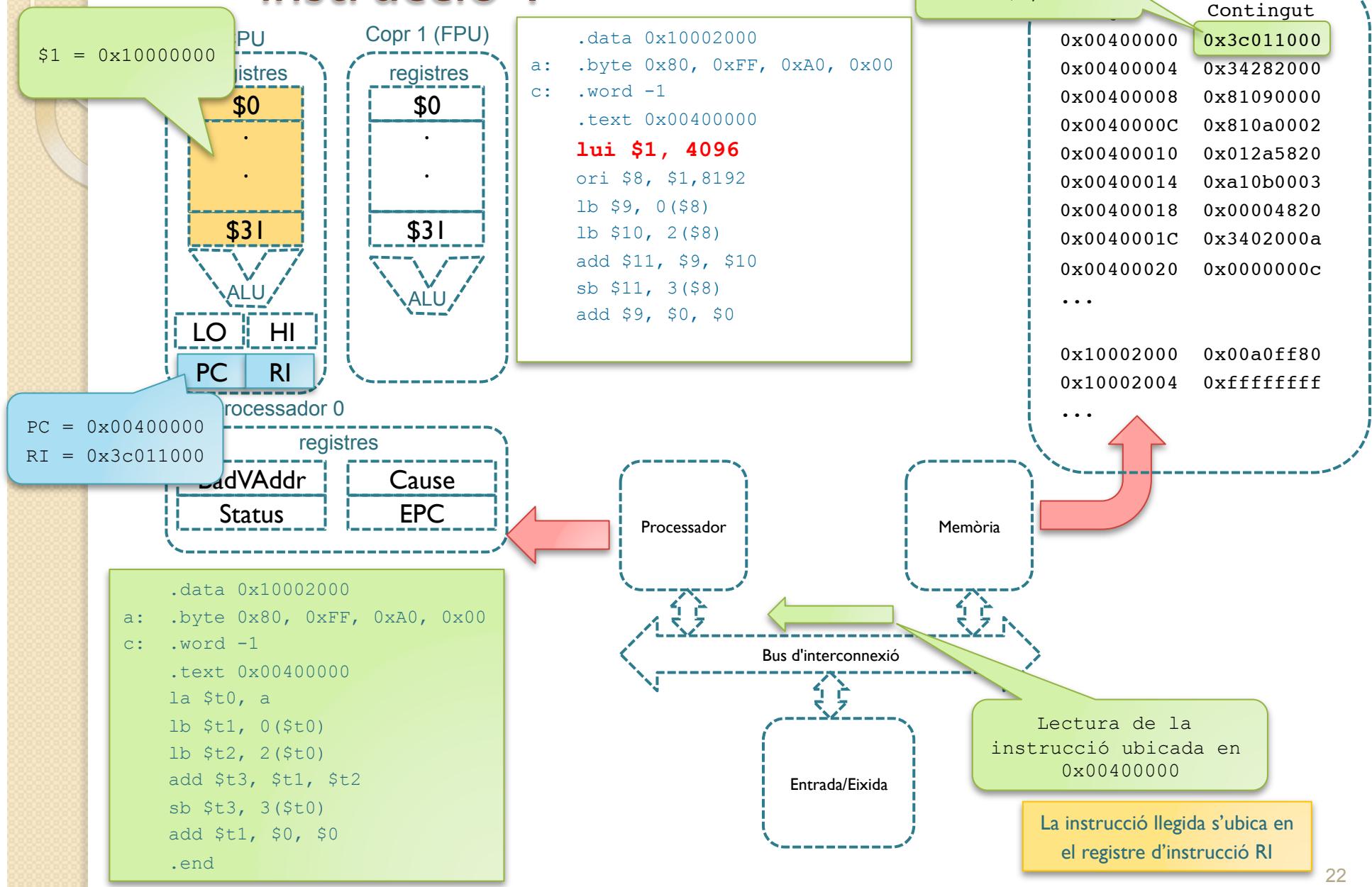
Exemple:

```
li $ta, 0x10002000 -> lui $t1, 0x1000  
                      ori $t1, $t1, 0x2000
```

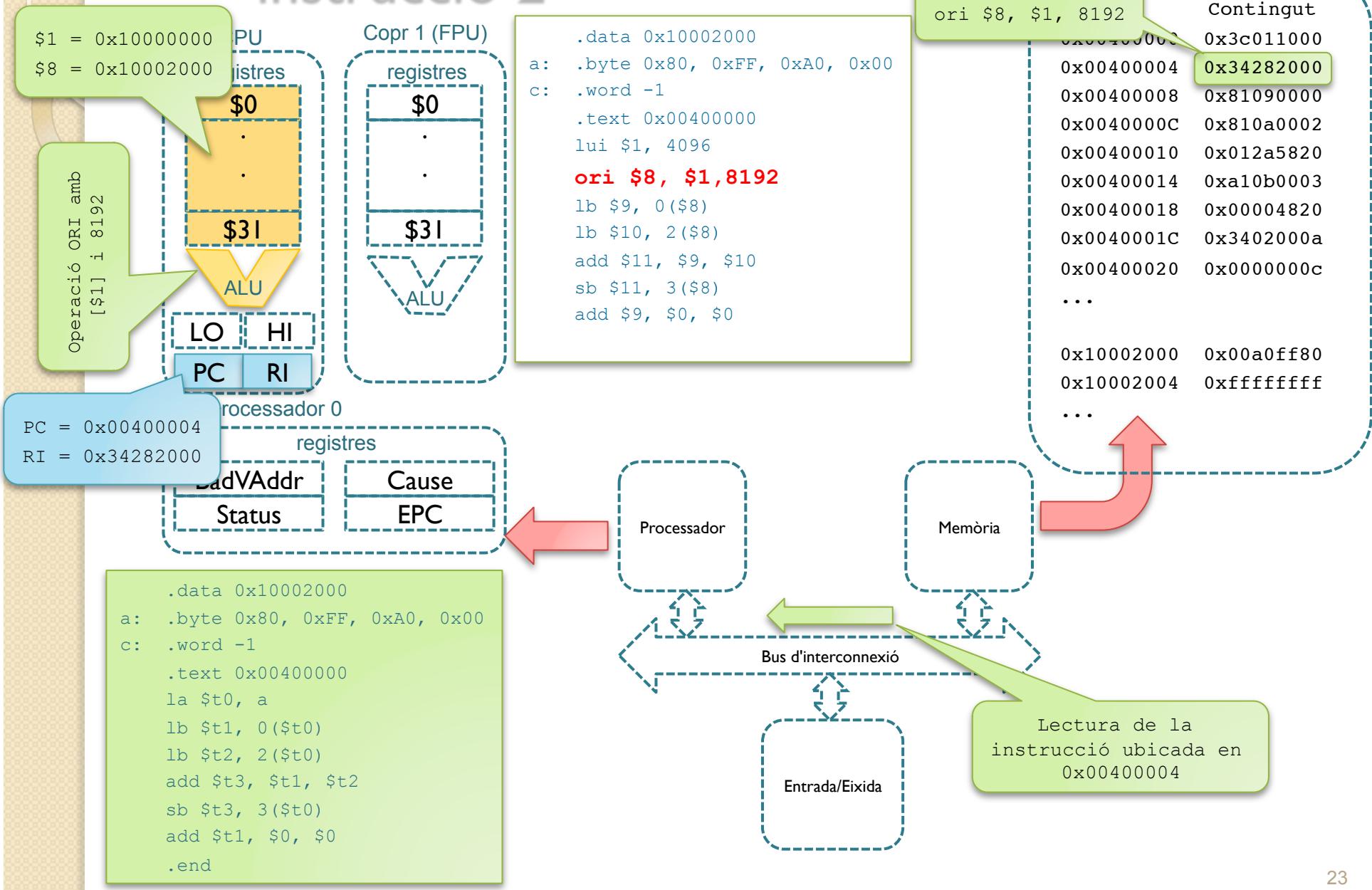
Exemple: codificació, assemblatge i càrrega



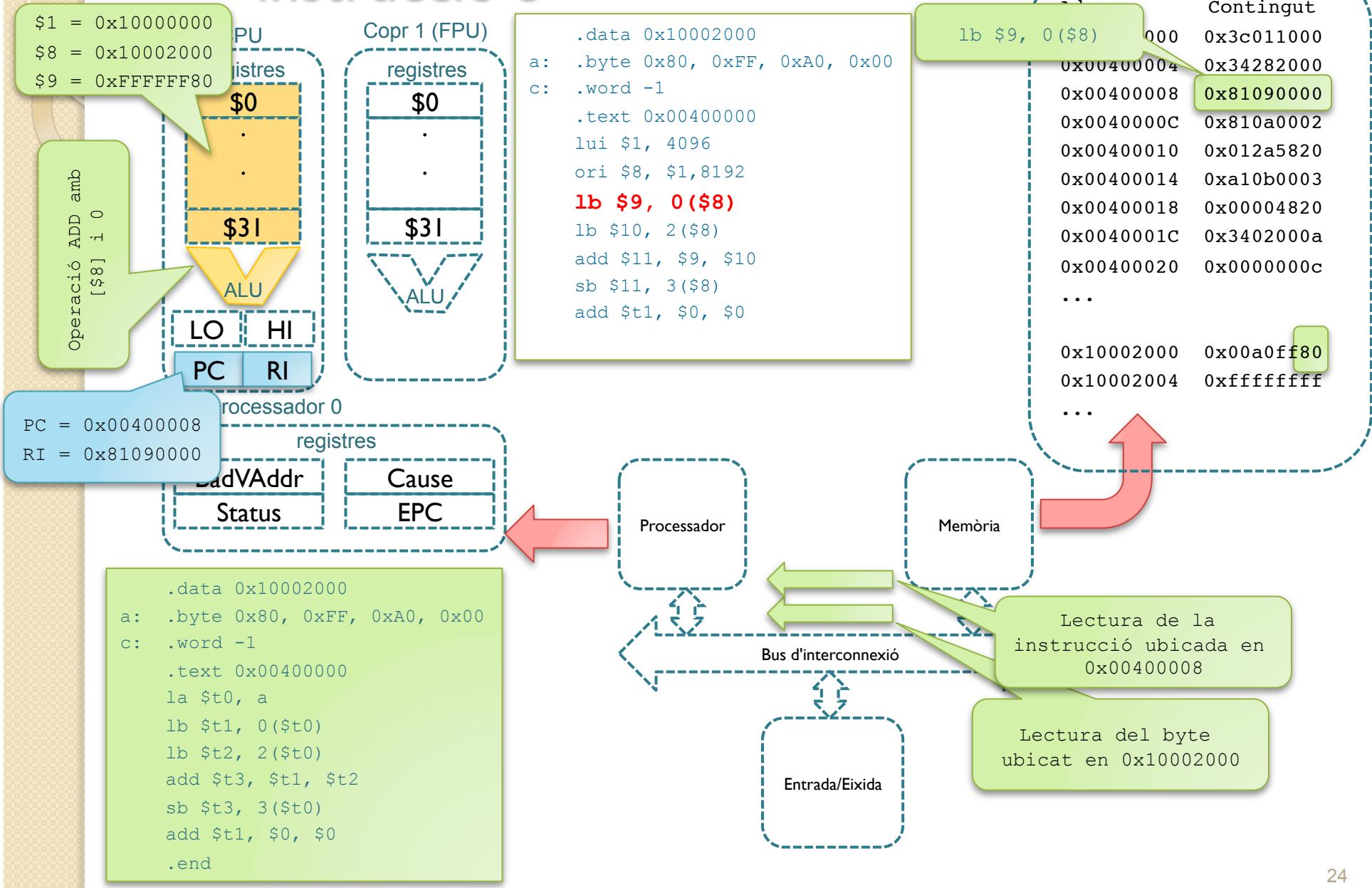
Exemple: execució instrucció I



Exemple: execució instrucció 2

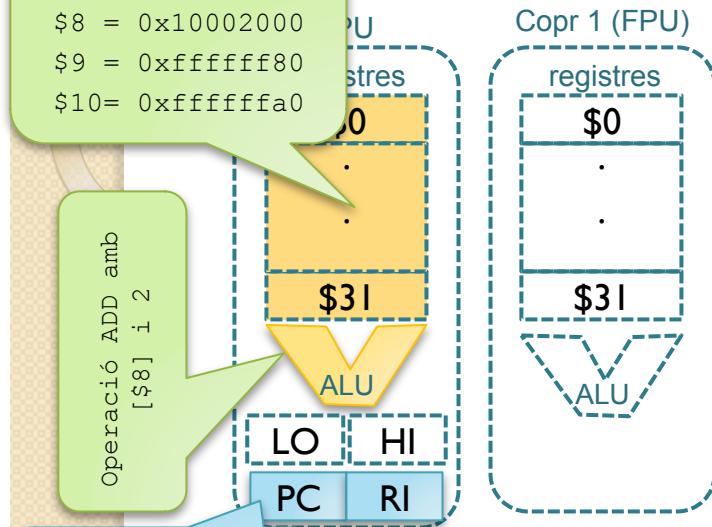


Exemple: execució instrucció 3

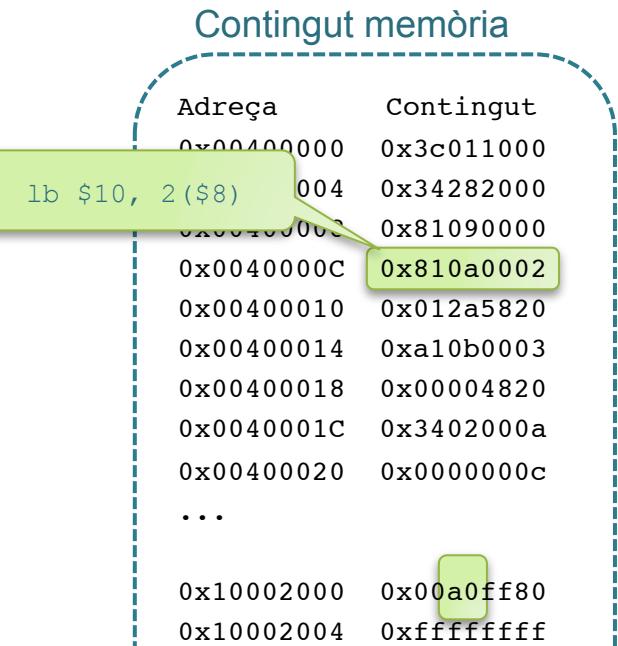


Exemple: execució instrucció 4

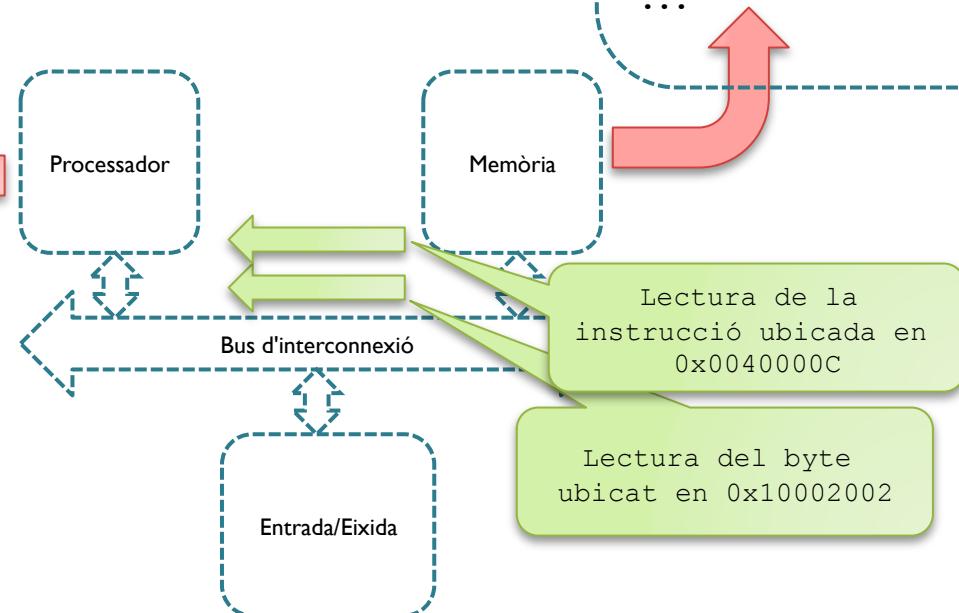
```
$1 = 0x10000000
$8 = 0x10002000
$9 = 0xfffffff80
$10= 0xffffffffa0
```



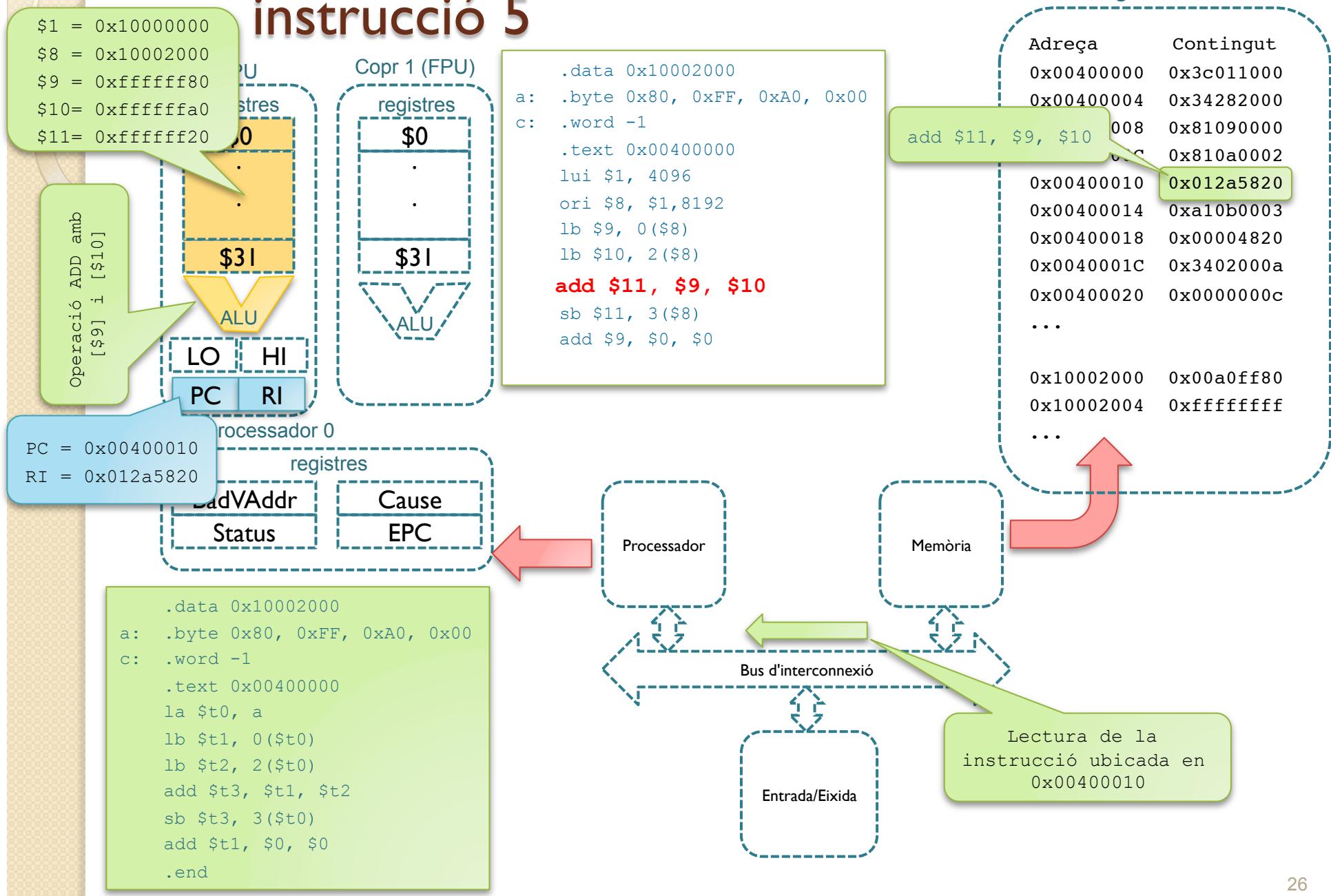
```
.data 0x10002000
a: .byte 0x80, 0xFF, 0xA0, 0x00
c: .word -1
.text 0x00400000
lui $1, 4096
ori $8, $1, 8192
lb $9, 0($8)
1b $10, 2($8)
add $11, $9, $10
sb $11, 3($8)
add $9, $0, $0
```



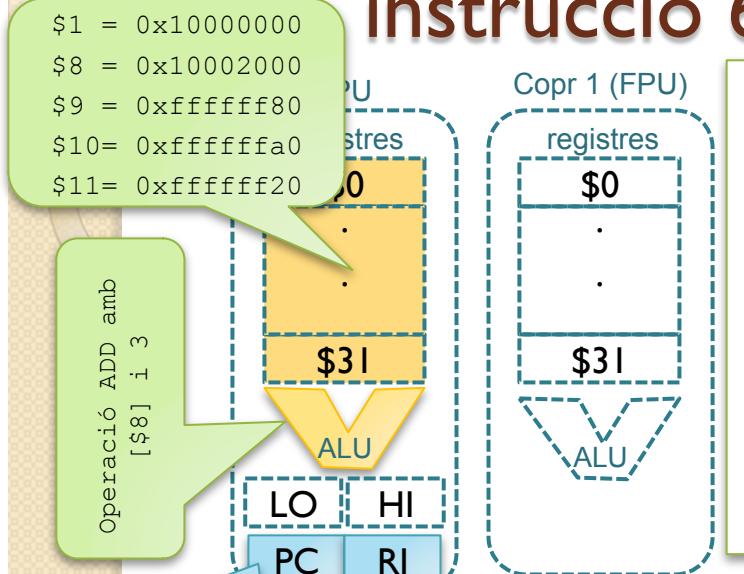
```
.data 0x10002000
a: .byte 0x80, 0xFF, 0xA0, 0x00
c: .word -1
.text 0x00400000
la $t0, a
lb $t1, 0($t0)
lb $t2, 2($t0)
add $t3, $t1, $t2
sb $t3, 3($t0)
add $t1, $0, $0
.end
```



Exemple: execució instrucció 5



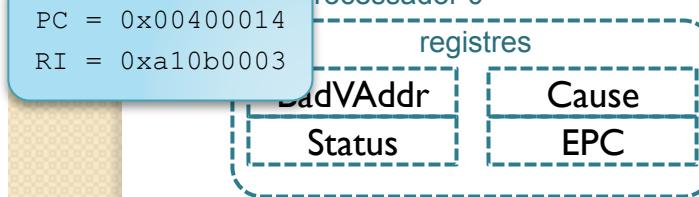
Exemple: execució instrucció 6



```

.data 0x10002000
a: .byte 0x80, 0xFF, 0xA0, 0x00
c: .word -1
.text 0x00400000
lui $1, 4096
ori $8, $1, 8192
lb $9, 0($8)
lb $10, 2($8)
add $11, $9, $10
sb $11, 3($8)
add $9, $0, $0

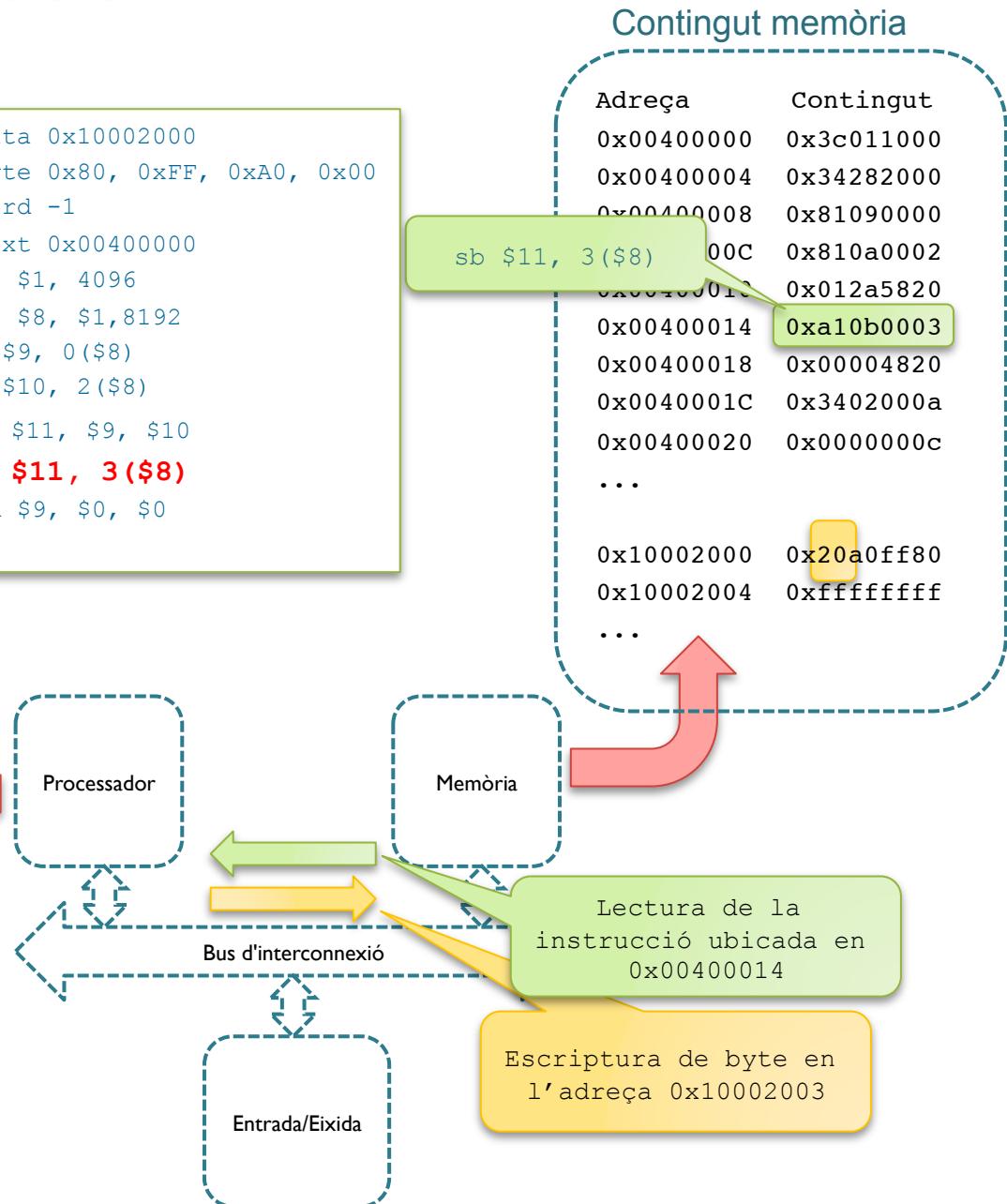
```



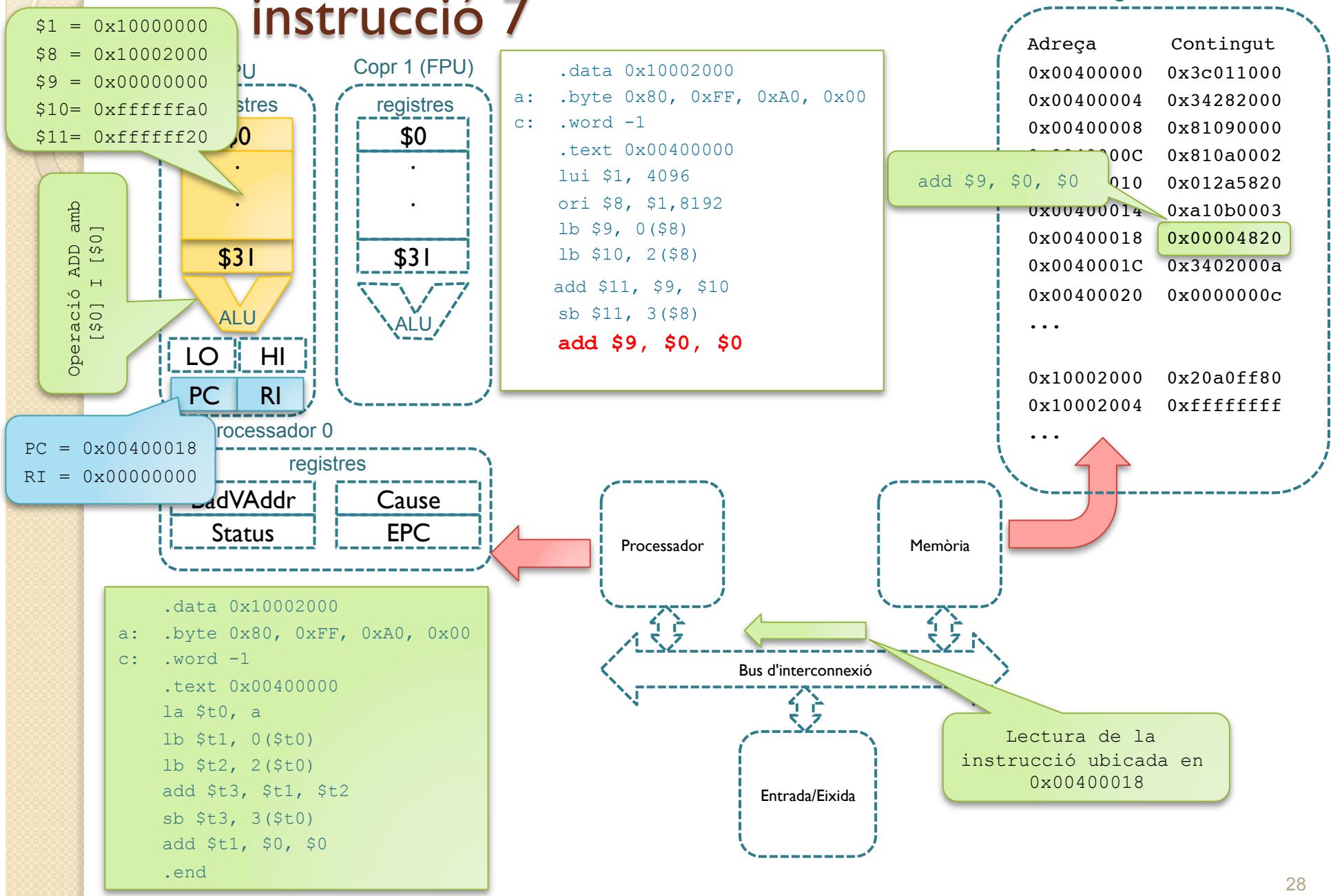
```

.data 0x10002000
a: .byte 0x80, 0xFF, 0xA0, 0x00
c: .word -1
.text 0x00400000
la $t0, a
lb $t1, 0($t0)
lb $t2, 2($t0)
add $t3, $t1, $t2
sb $t3, 3($t0)
add $t1, $0, $0
.end

```



Exemple: execució instrucció 7



Contingut i bibliografia

- I – Arquitectura MIPS32

- ✓ Introducció i definició
- ✓ Característiques bàsiques
- ✓ Exemple d'execució

Introductori
(Tema 0)



- **2 – La ruta de dades**

- ✓ Etapes de búsqueda y decodificación
- ✓ Disseny de la ruta per a instruccions aritmètiques de tipus R
- ✓ Disseny de la ruta per a instruccions aritmètiques de tipus R+I
- ✓ Disseny de la ruta per a instruccions lw/sw
- ✓ Disseny de la ruta per a instruccions de salt beq/bne

Bibliografía: Patterson, D.A., Hennessy, J.L., “Estructura y diseño de computadores. La interfaz hardware-Software,” 4a edición, Ed. Reverté, 2011, Cap 4 (4.1 – 4.4)

El processador

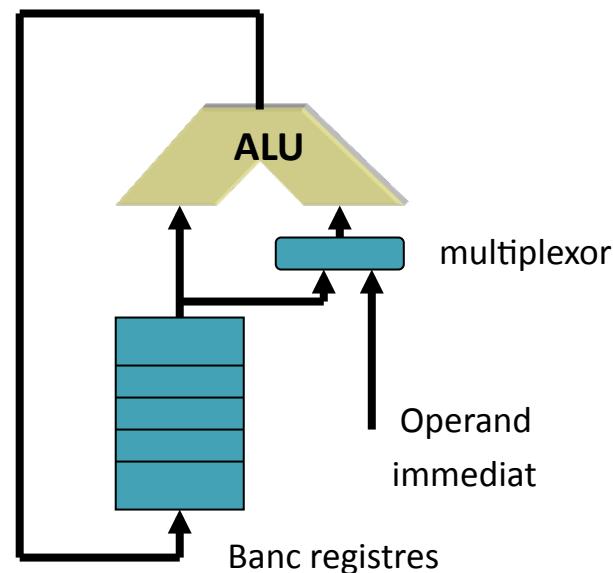
- Disseny del processador
 - ✓ Elecció del joc de instruccions a executar
 - Subconjunt del joc de instruccions del MIPS
 - Aritmètiques i comparació: add, sub, and, or, slt
 - Càrrega/emmagatzematge: lw, sw
 - Salt: beq, jump **(la instrucció jump serà afegida posteriorment)**
 - ✓ Cicle únic de rellotge
 - ✓ Combinació d'elements lògics senzills
 - Circuits combinacionals: portes, descodificadors, multiplexors, ALU, ...
 - Circuits sequencials: biestables, registres, banc de registres, memòria, ...
 - ✓ Les **prestacions** depenen del nombre d'instruccions que executa per unitat de temps i de la durada del cicle de rellotge
 - Objectiu inicial: executar una instrucció per cicle i reduir el cicle de rellotge

El processador

- Etapes en el disseny
 - ✓ Dissenyar la ruta de dades
 - Seleccionar els elements necessaris per a executar les instruccions
 - Interconnectar els elements
 - ✓ Dissenyar la unitat de control
 - Identificar els senyals de control necessaris
 - Dissenyar la lògica associada a cada senyal

La ruta de dades

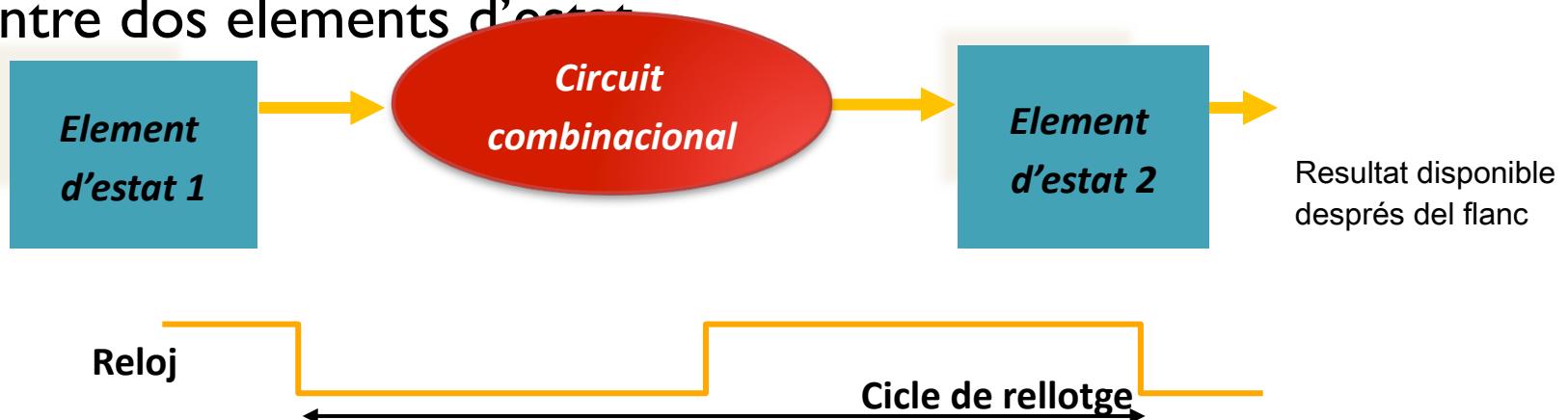
- Definició
 - ✓ Combinació d'elements combinacionals, sequencials i la seua interconnexió
 - ✓ Permet el flux de la informació que es transforma a conseqüència de l'execució de cada instrucció



Exemple de ruta de dades simple

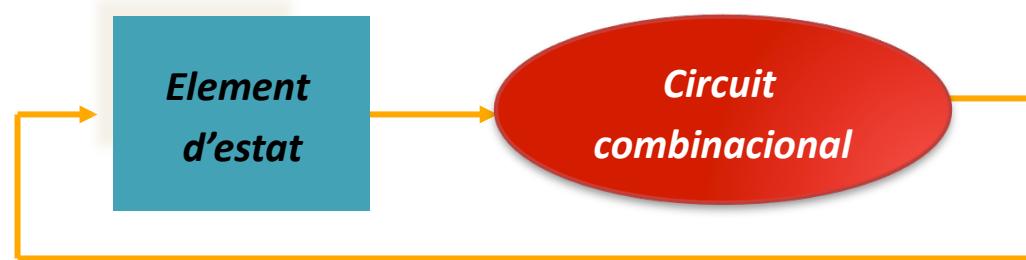
La ruta de dades: sincronització

- Sincronització: quan es pot llegir i escriure la informació en els elements d'emmagatzematge del processador
 - ✓ Quan i com s'han d'activar els senyals de control que en són responsables
- Assumirem sincronització **per flanc**
 - ✓ Qualsevol valor emmagatzemat sols s'actualitza en un flanc del rellotge
- **Cicle de rellotge:** defineix el temps de propagació necessari entre dos elements d'estat

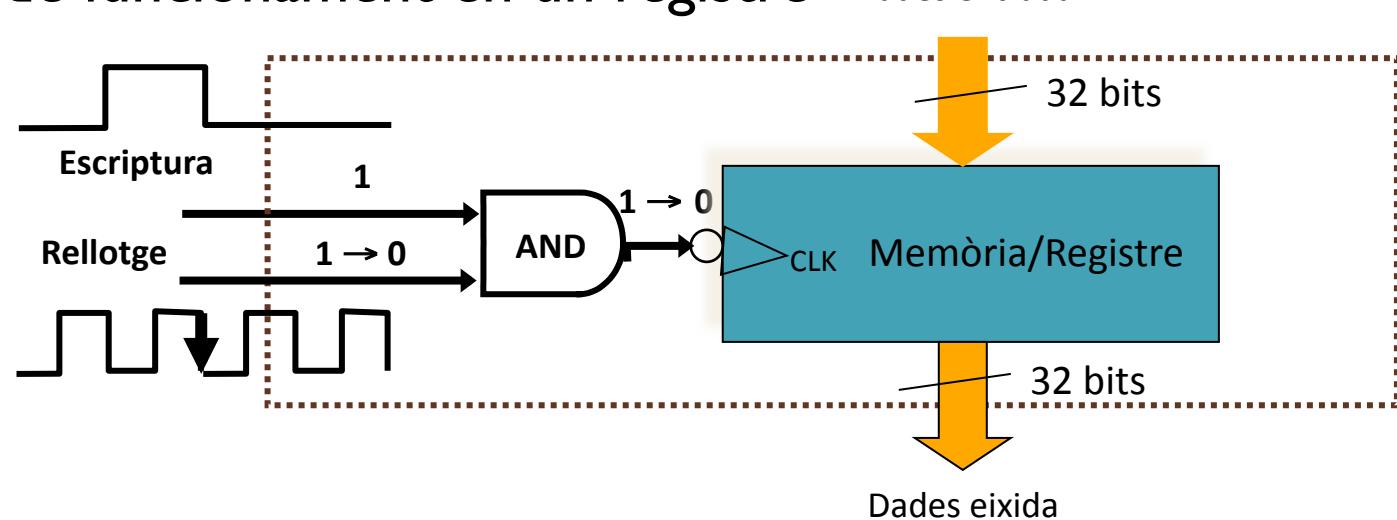


La ruta de dades: sincronització

- Un element es pot llegir i escriure en el mateix cicle de rellotge



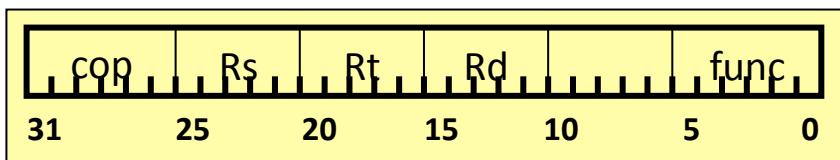
- Exemple de funcionament en un registre



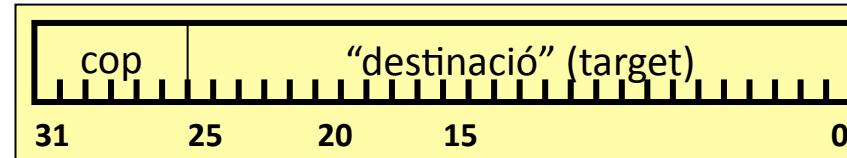
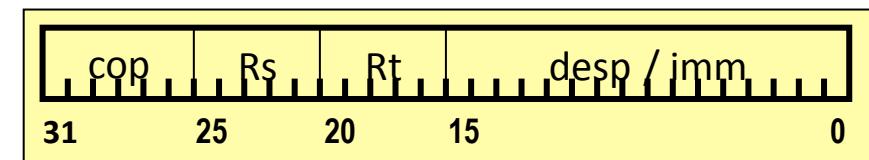
La ruta de dades: selecció d'instruccions

| Instrucció | Operació | Format | Codi Op. | Funció |
|------------------|--|--------|----------|--|
| add rd, rs, rt | $rd = rs + rt$ | R | 000000 | 100000 |
| sub rd, rs, rt | $rd = rs - rt$ | R | 000000 | 100010 |
| and rd, rs, rt | $rd = rs \wedge rt$ | R | 000000 | 100100 |
| or rd, rs, rt | $rd = rs \vee rt$ | R | 000000 | 100101 |
| lw rt, desp(rs) | $rt = \text{Mem}[rs+desp]$ | I | 100011 | |
| sw rt, desp(rs) | $\text{Mem}[rs+desp] = rt$ | I | 101011 | |
| beq rs, rs, etiq | Si $rs == rt$ aleshores $PC = etiq$ $(PC = PC + desp * 4)$ | I | 000100 | |
| j etiq | $PC = \text{etiqueta}$ $(PC_{27..0} = \text{destinació} * 4)$ | J | 000010 | S'implementarà en sessions de pràctiques |

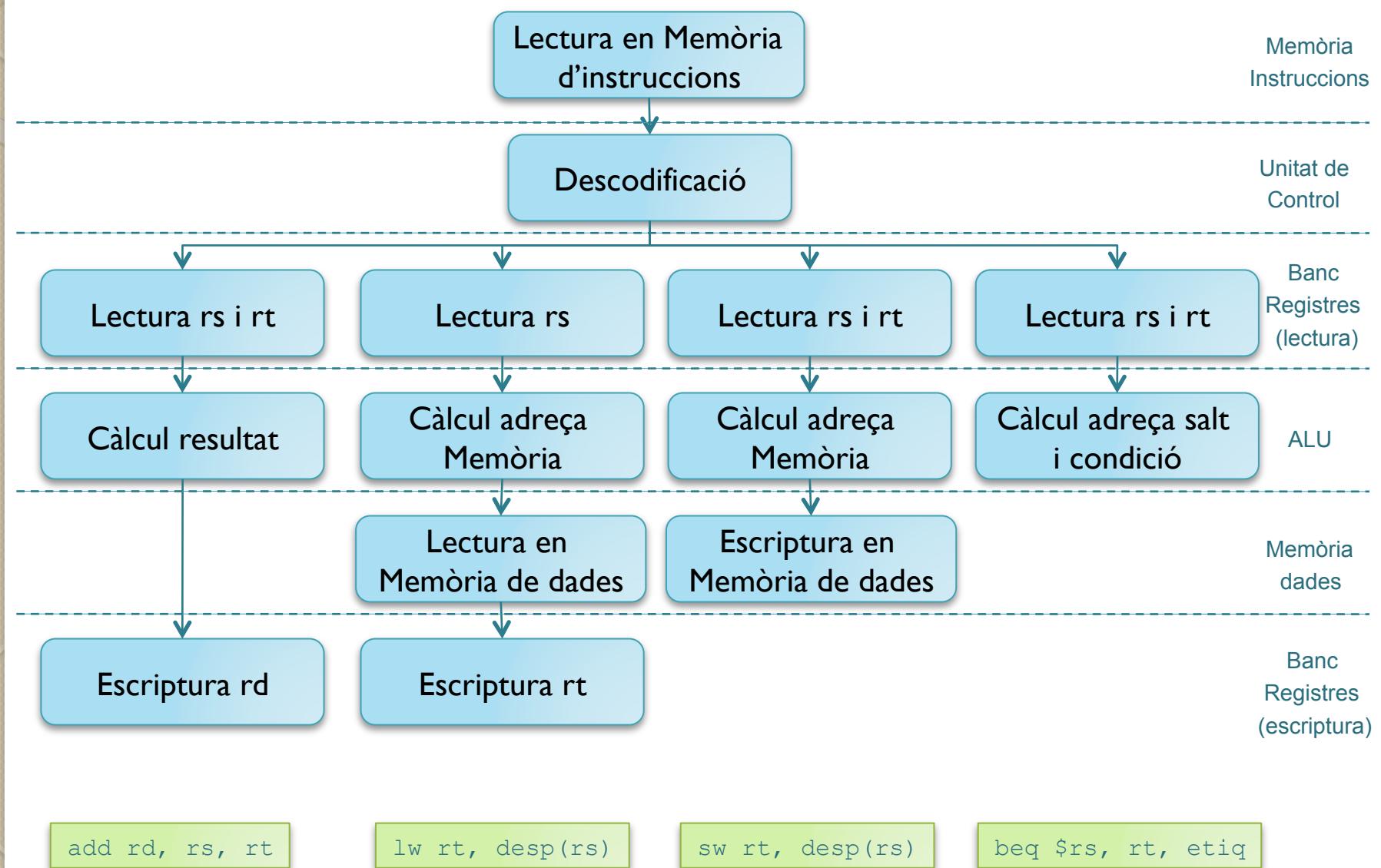
Format R



Format I



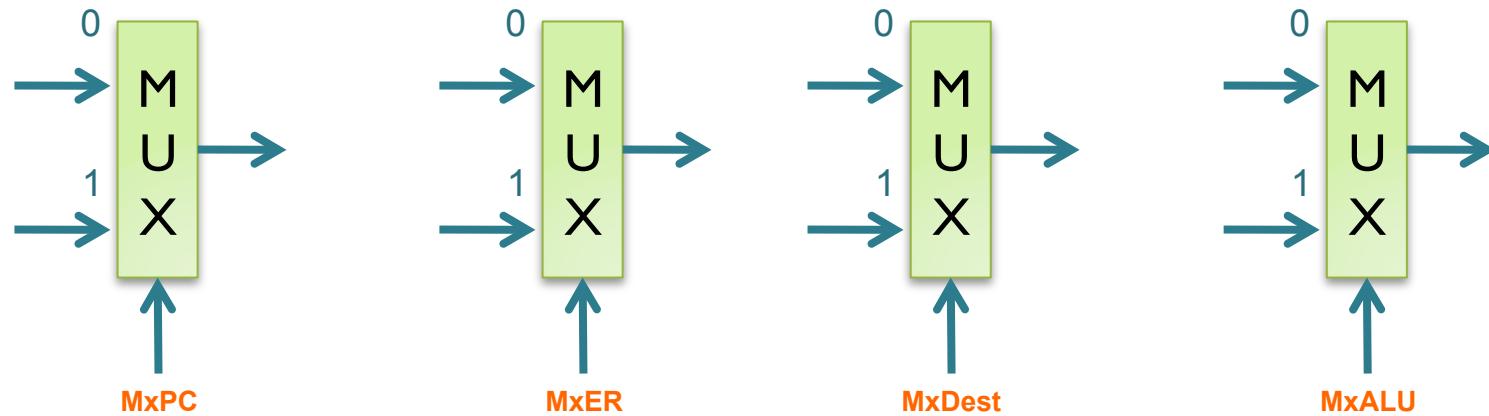
La ruta de dades: fases d'execució d'instruccions



La ruta de dades: selecció de recursos

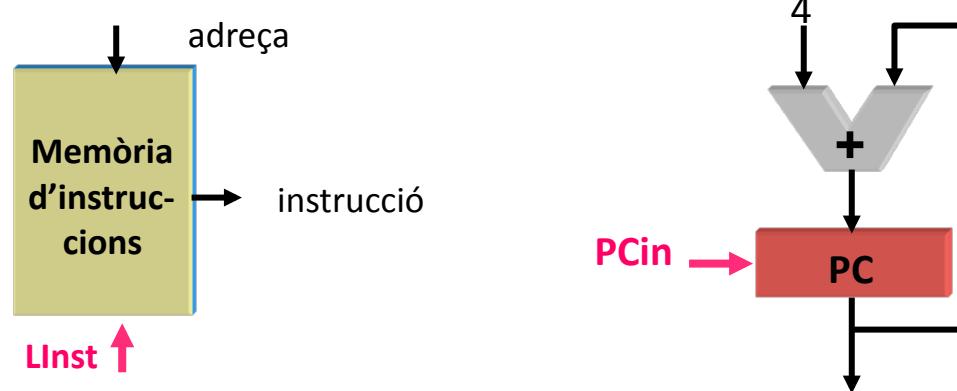
- Multiplexors

- ✓ L'ús dels recursos (memòries, ALUs, registres, ...) diferirà segons la instrucció que s'executa
- ✓ Farem servir quatre multiplexors, segons la instrucció, per a triar els camins dins la ruta de dades

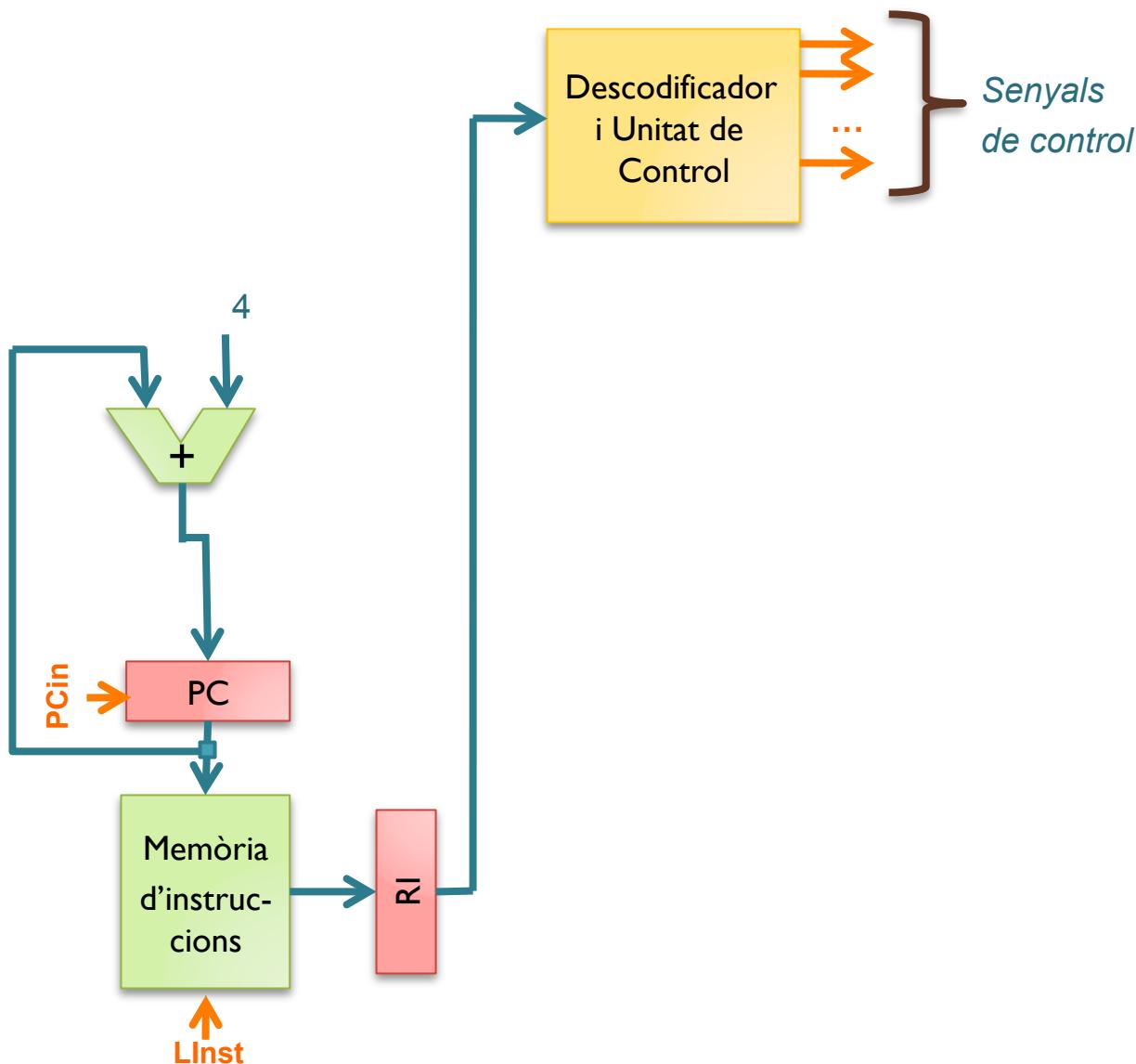


La ruta de dades: selecció de recursos

- Totes les instruccions tenen una fase en comú
 - ✓ Cerca de la instrucció i descodificació
 - ✓ Cal:
 - 1) Llegir la instrucció de la memòria d'instruccions
 - L'adreça la facilita el **Comptador de Programa (PC, Program Counter)**
 - 2) Incrementar el PC perquè apunte a la següent instrucció ($PC = PC + 4$)
 - 3) Descodificar la instrucció
 - Ho fa la unitat **de CONTROL**



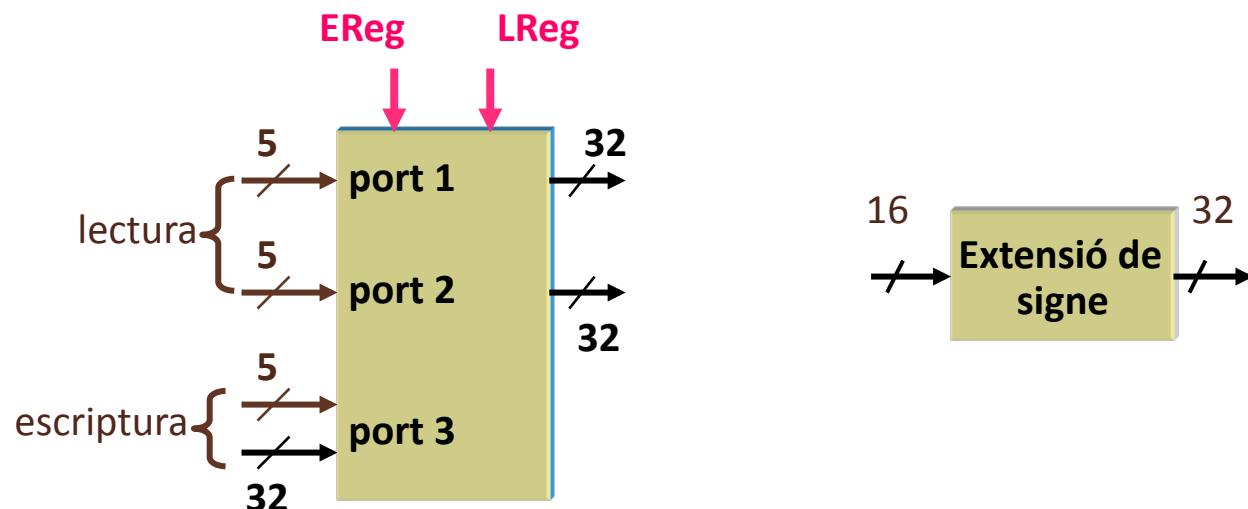
La ruta de dades: cerca i descodificació



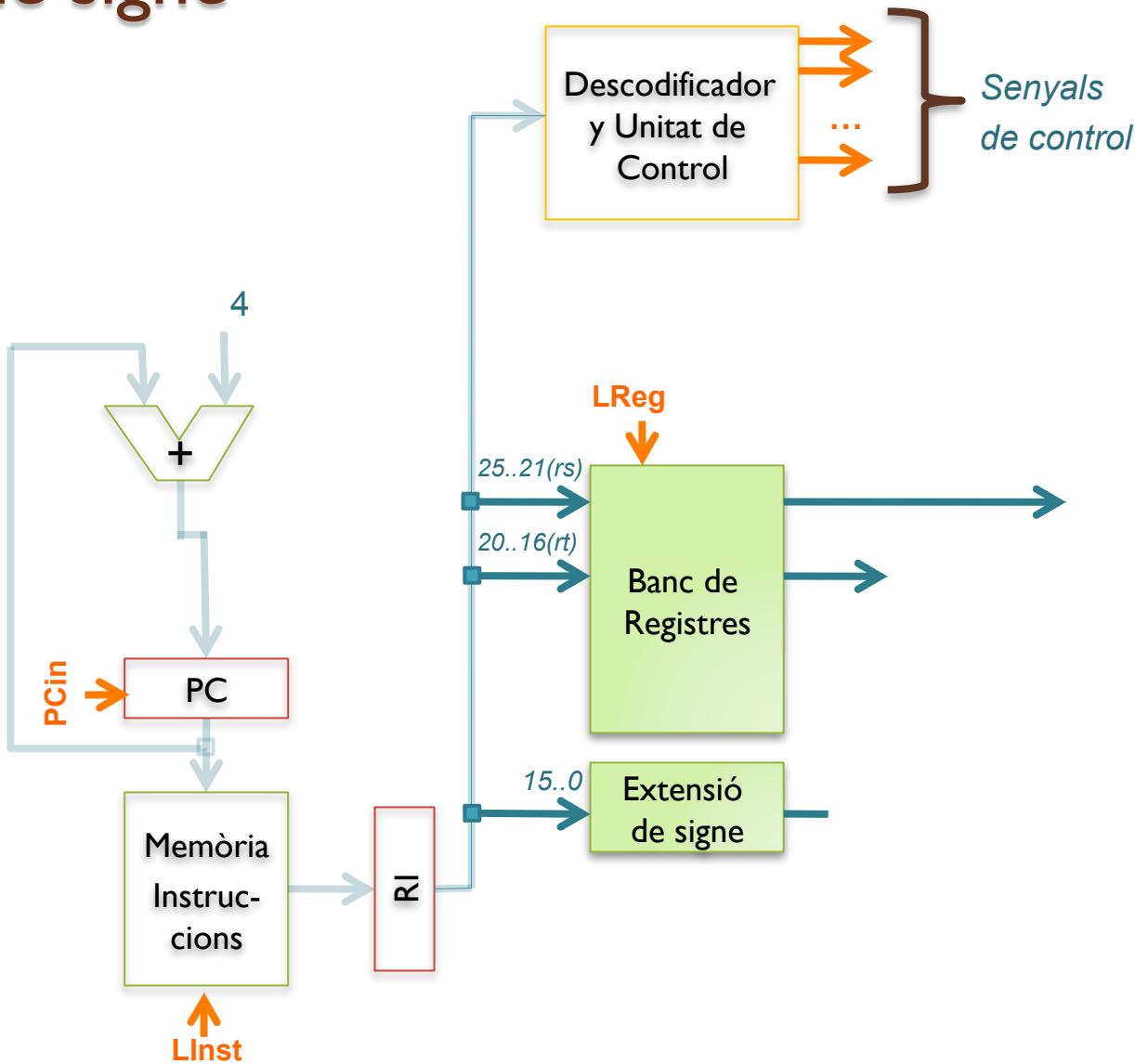
La ruta de dades: selecció de recursos

- Les instruccions utilitzen el banc de registres (\$0 a \$31)
 - ✓ Tenim 32 registres de 32 bits
 - ✓ Necesitem dos ports de lectura i un port d'escriptura
 - Els ports de lectura són activats simultàniament pel mateix senyal
- Extensió de signe necessària en algunes operacions

```
lb $t1, -2($t0)      # -2 està codificat en Ca2 de 16 bits i la
                      # suma -2+$t0 ha de fer-se en Ca2 de 32 bits
```

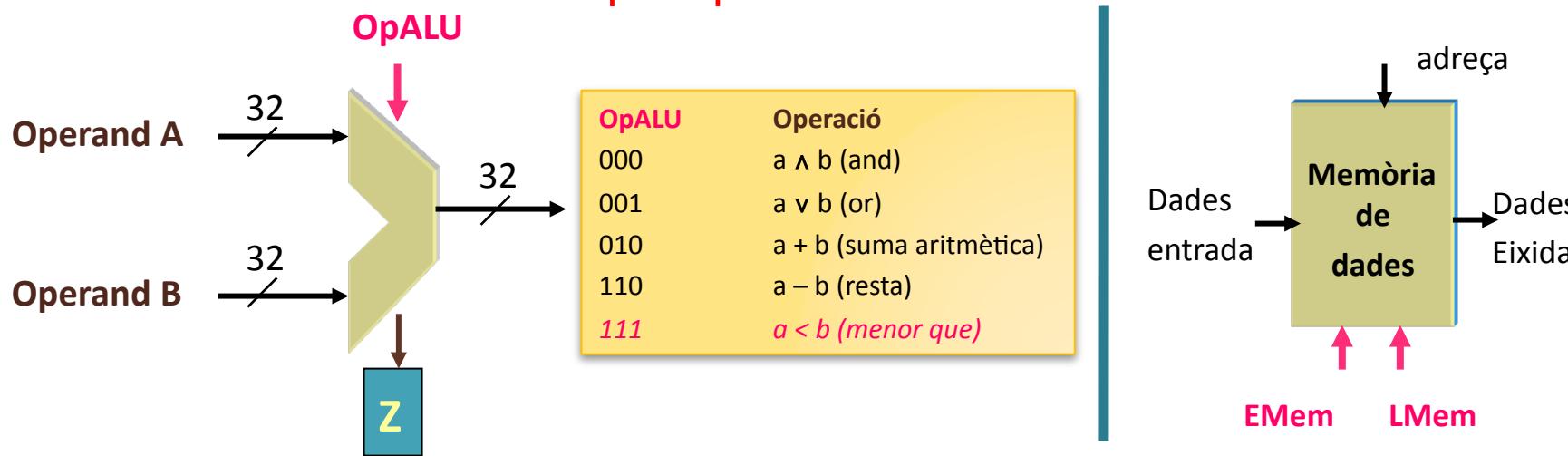


La ruta de dades: lectura d'operands i extensió de signe

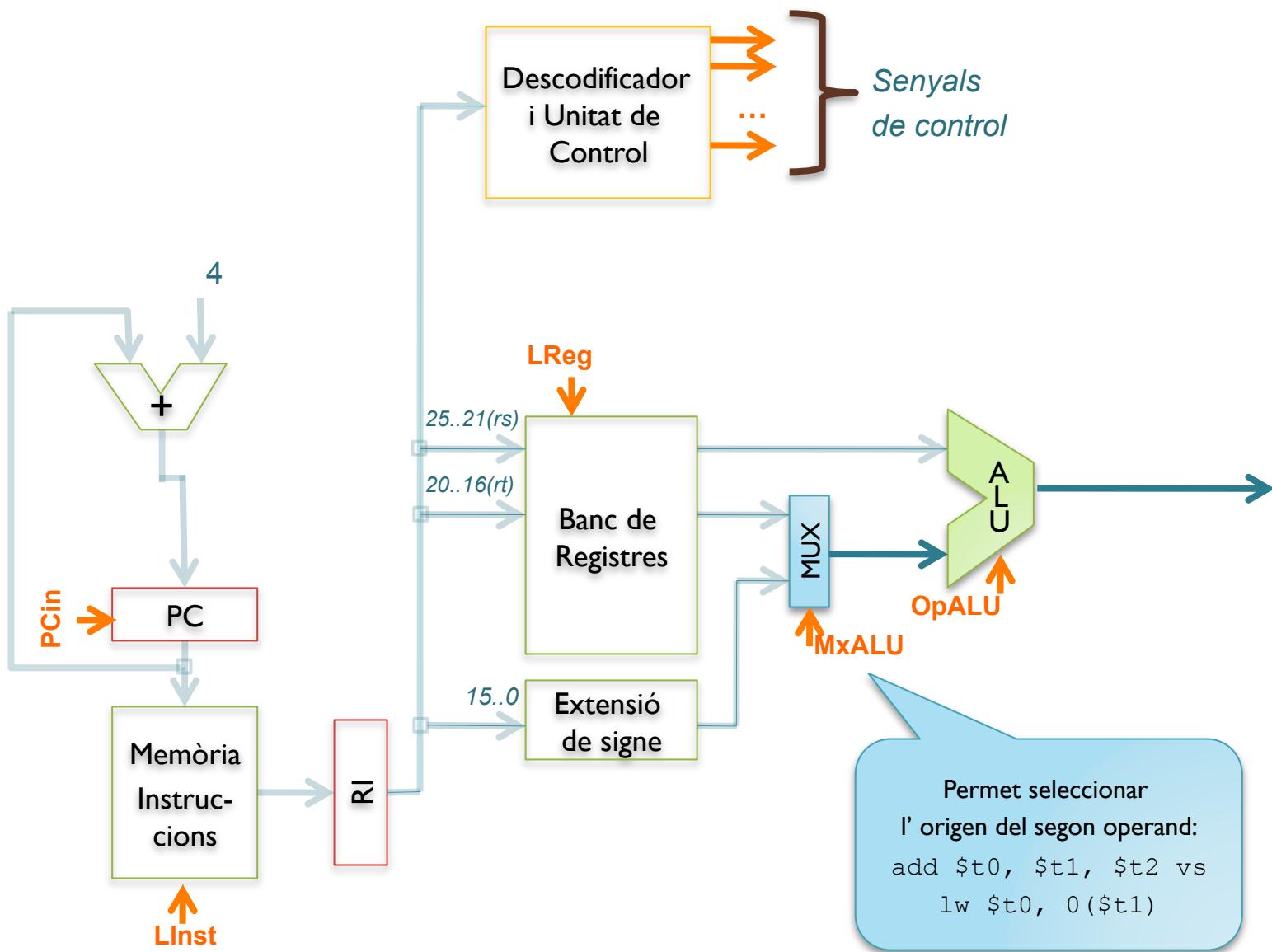


La ruta de dades: selecció de recursos

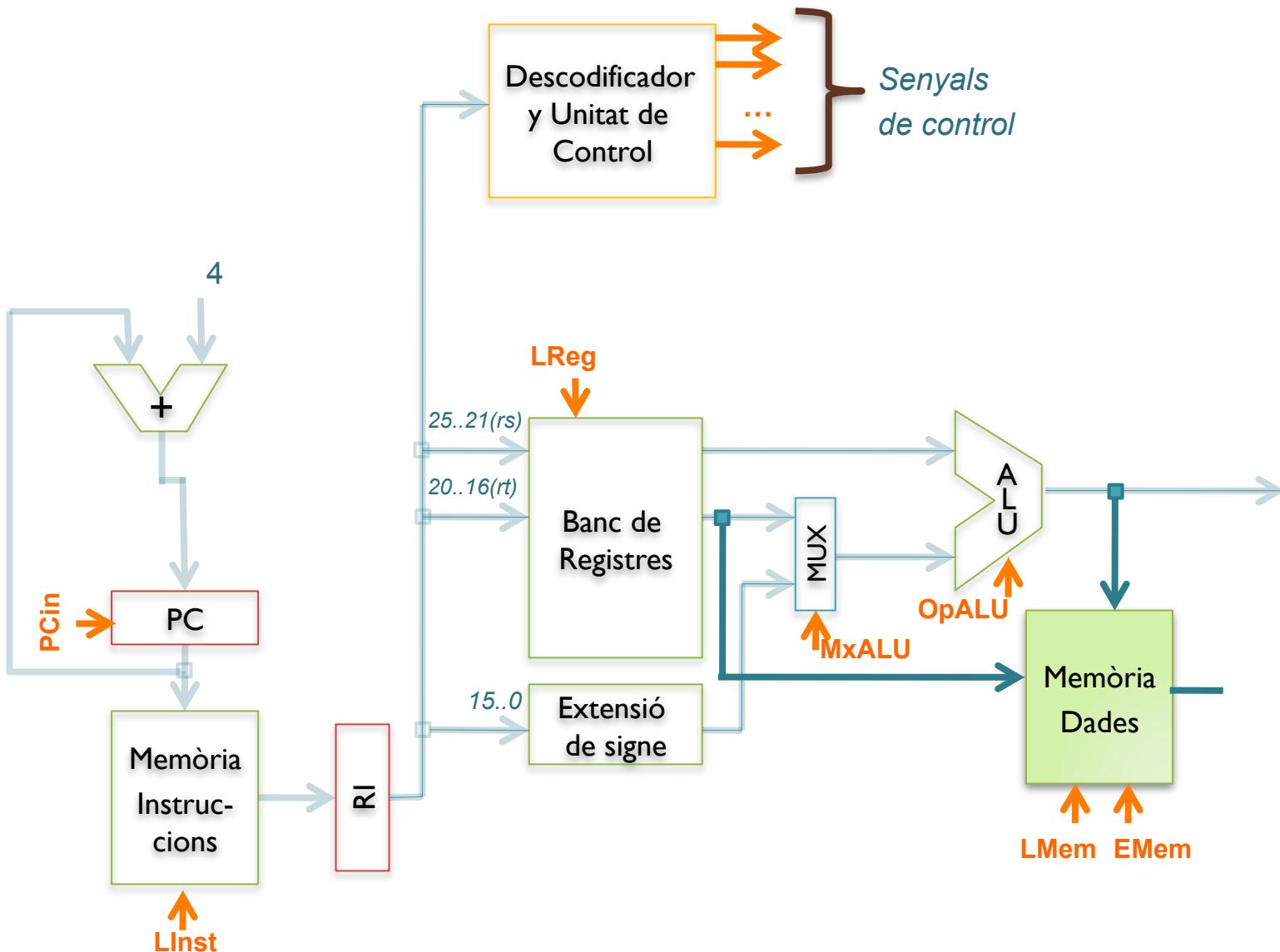
- Quasi totes les instruccions utilitzen un operador aritmèticològic
 - ✓ Emprem una ALU per a suportar les operacions necessàries per al joc d'instruccions
 - Disposa d'un indicador de zero (Z)
- Algunes instruccions accedeixen a memòria per a llegir/escriure dades
 - ✓ Utilitzem una memòria de dades
 - ✓ En futurs temes veurem que aquesta memòria és una memòria cau OpALU



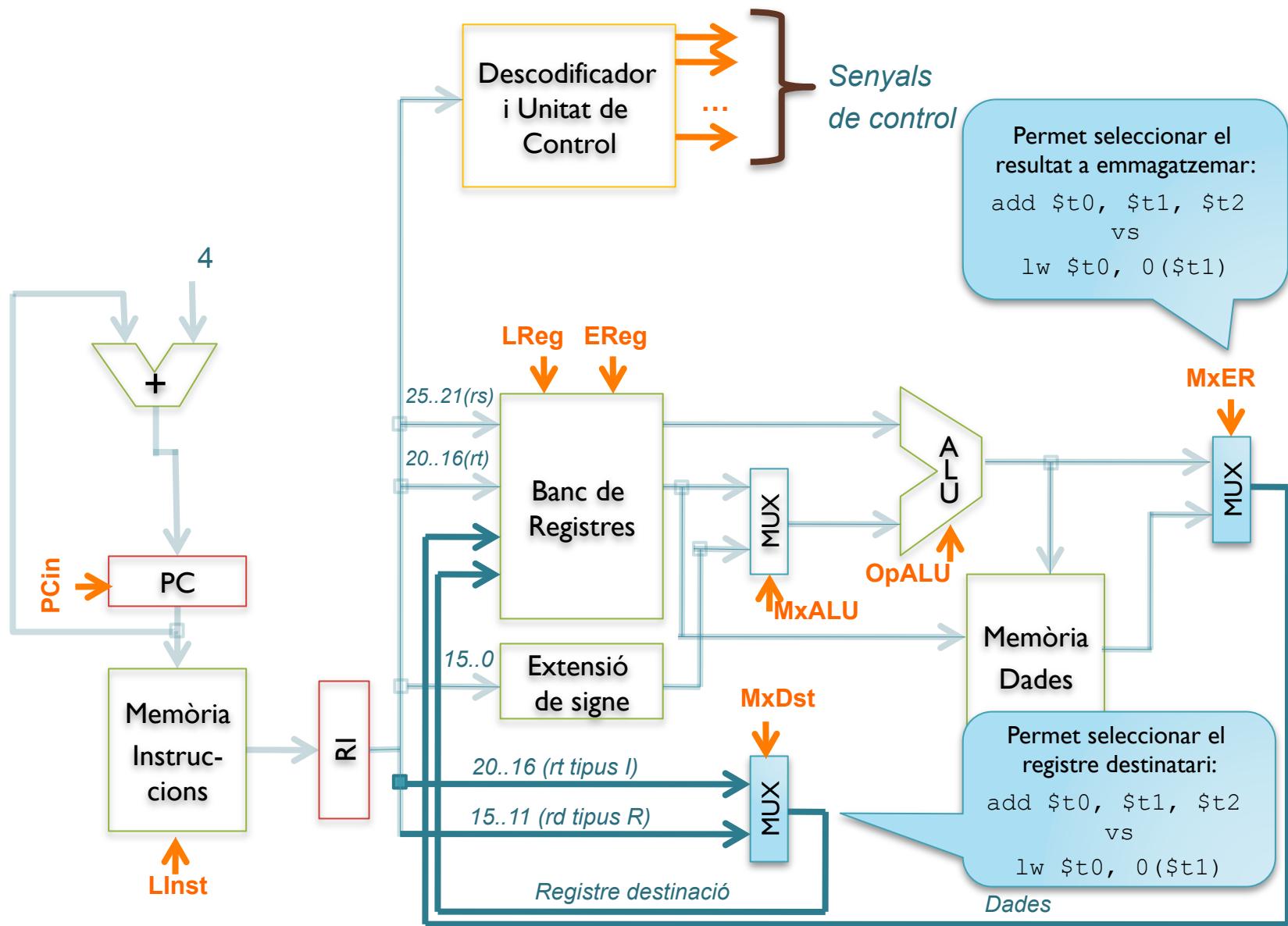
La ruta de dades: operació amb la ALU



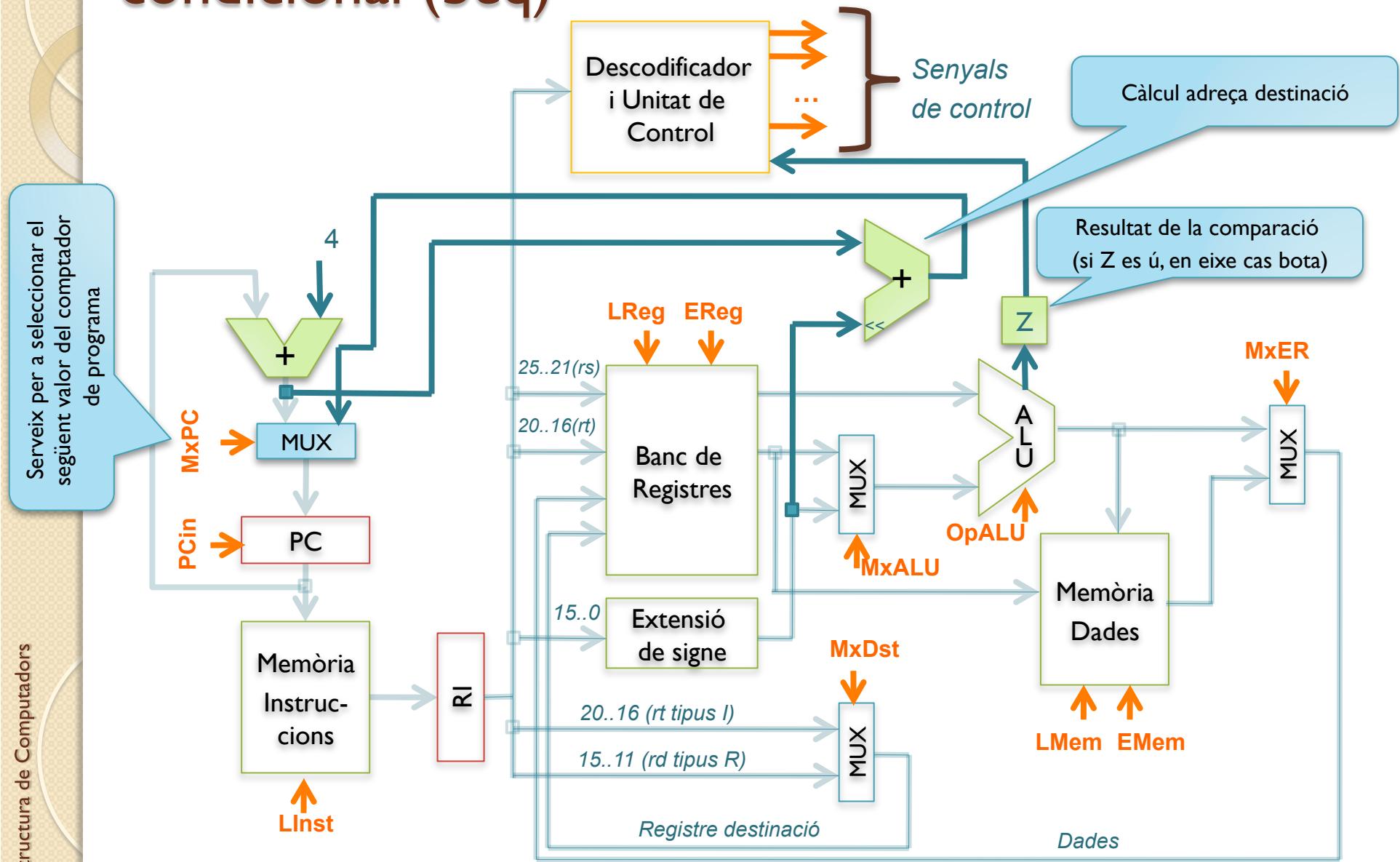
La ruta de dades: accés a memòria de dades



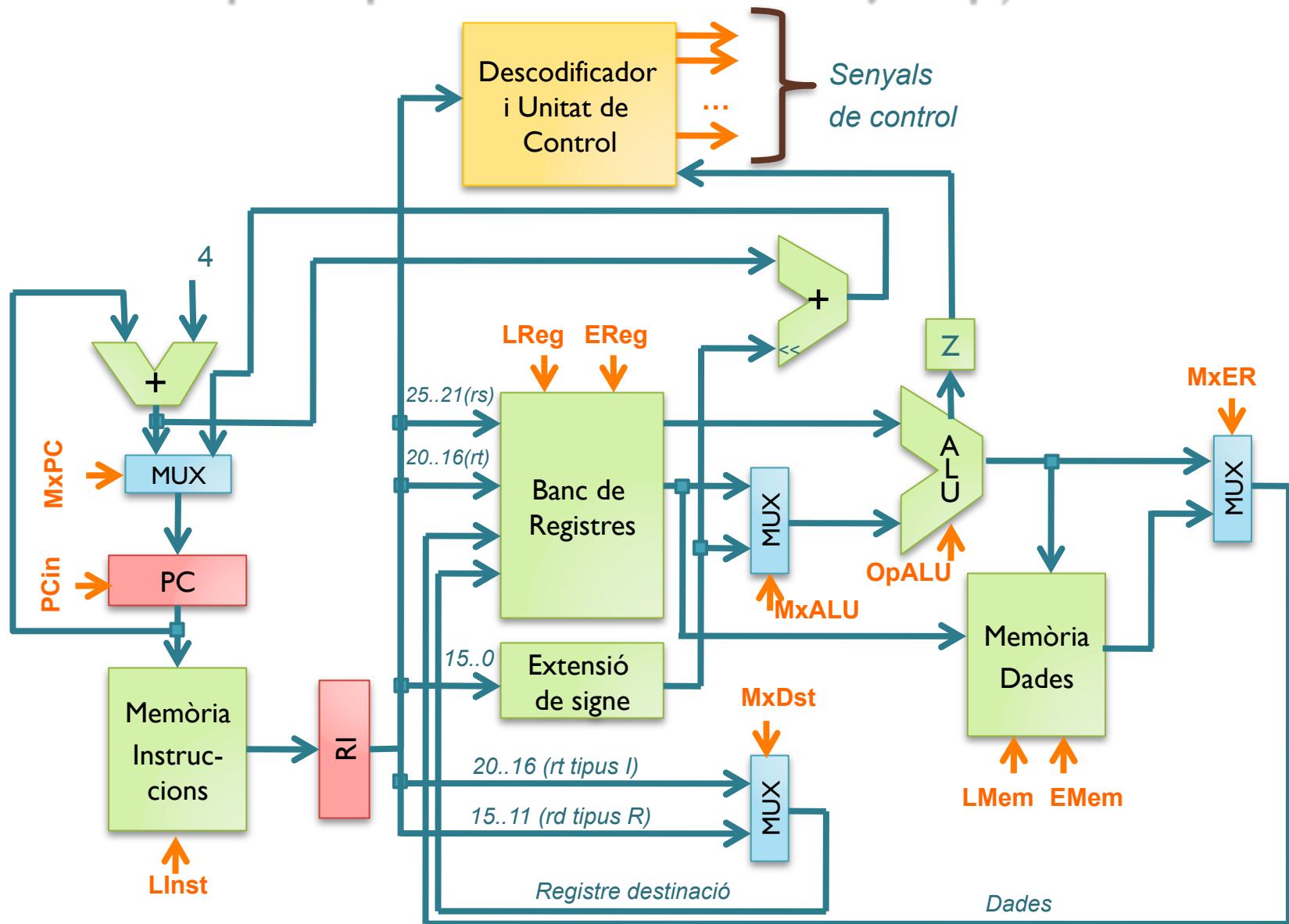
La ruta de dades: emmagatzemar el resultat



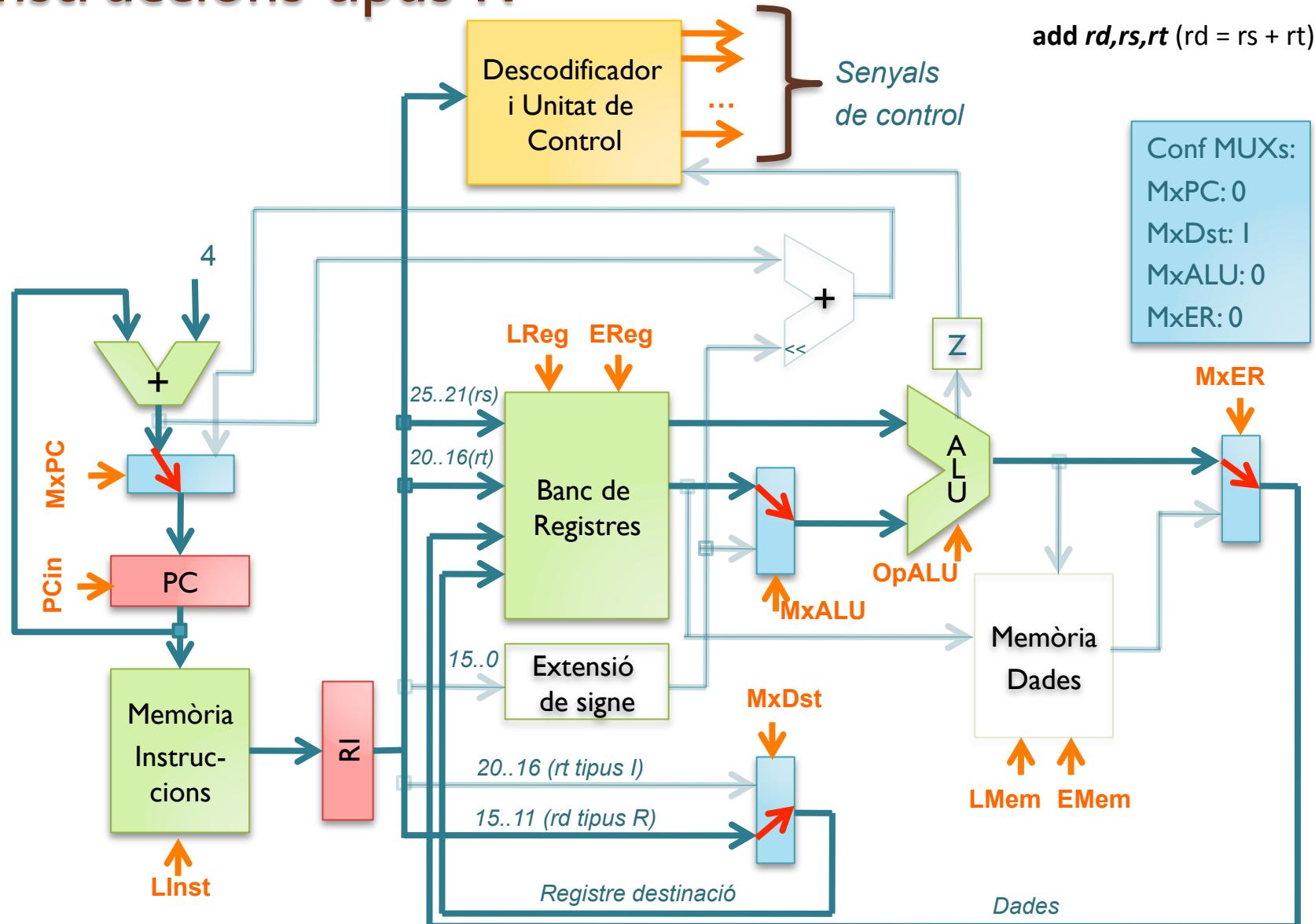
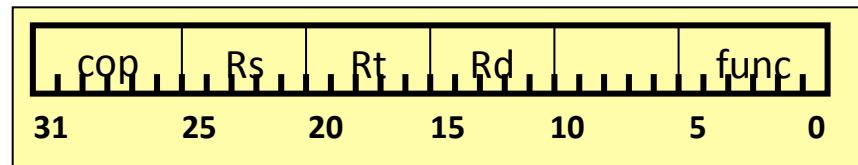
La ruta de dades: suport instrucció de bot condicional (beq)



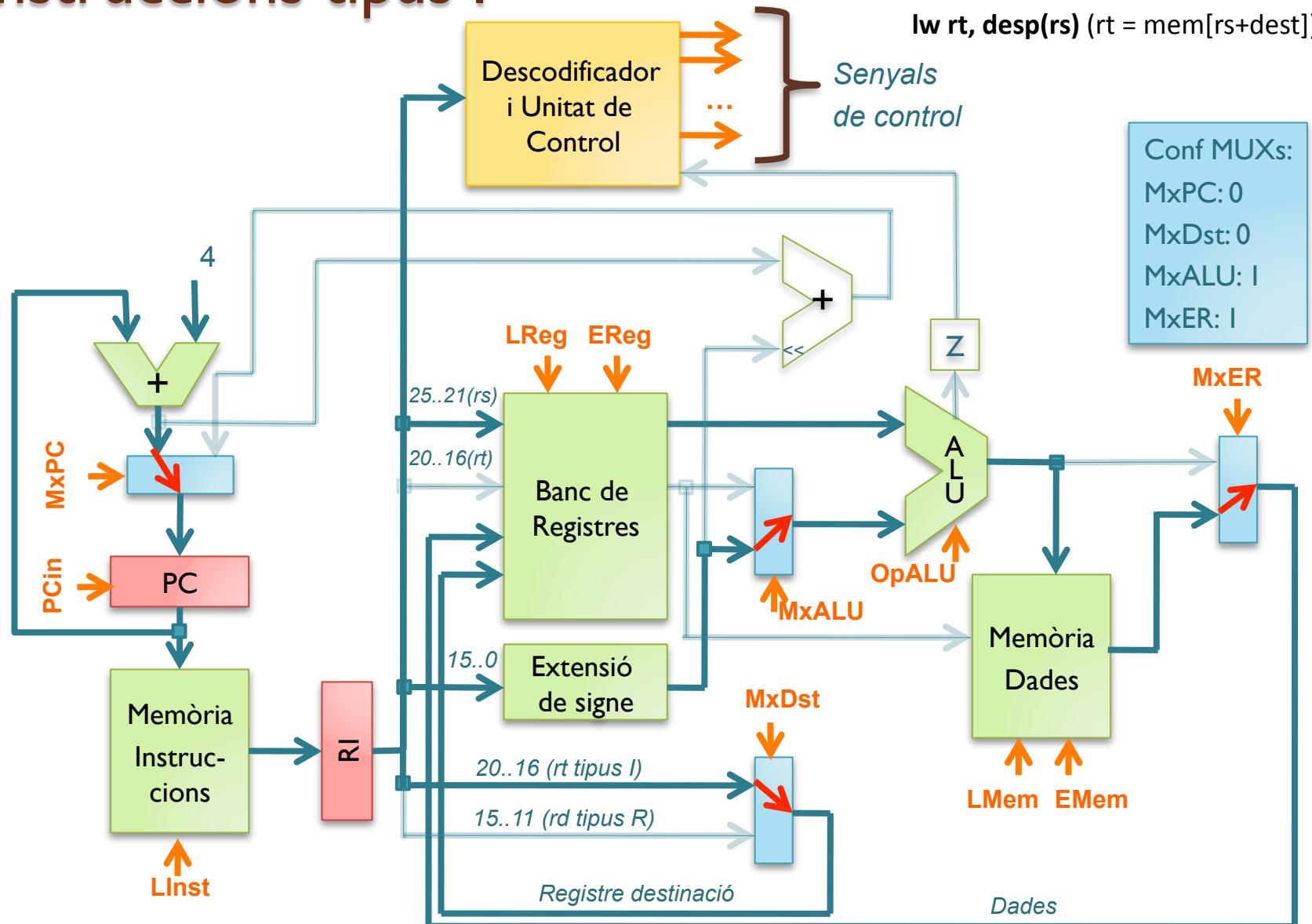
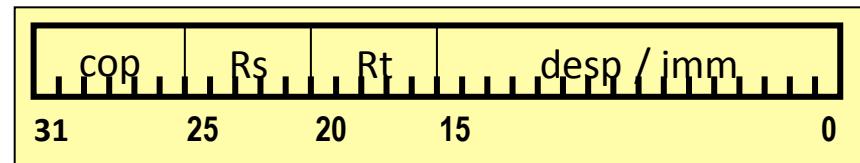
Ruta de dades completa (sense suport per a la instrucció jump)



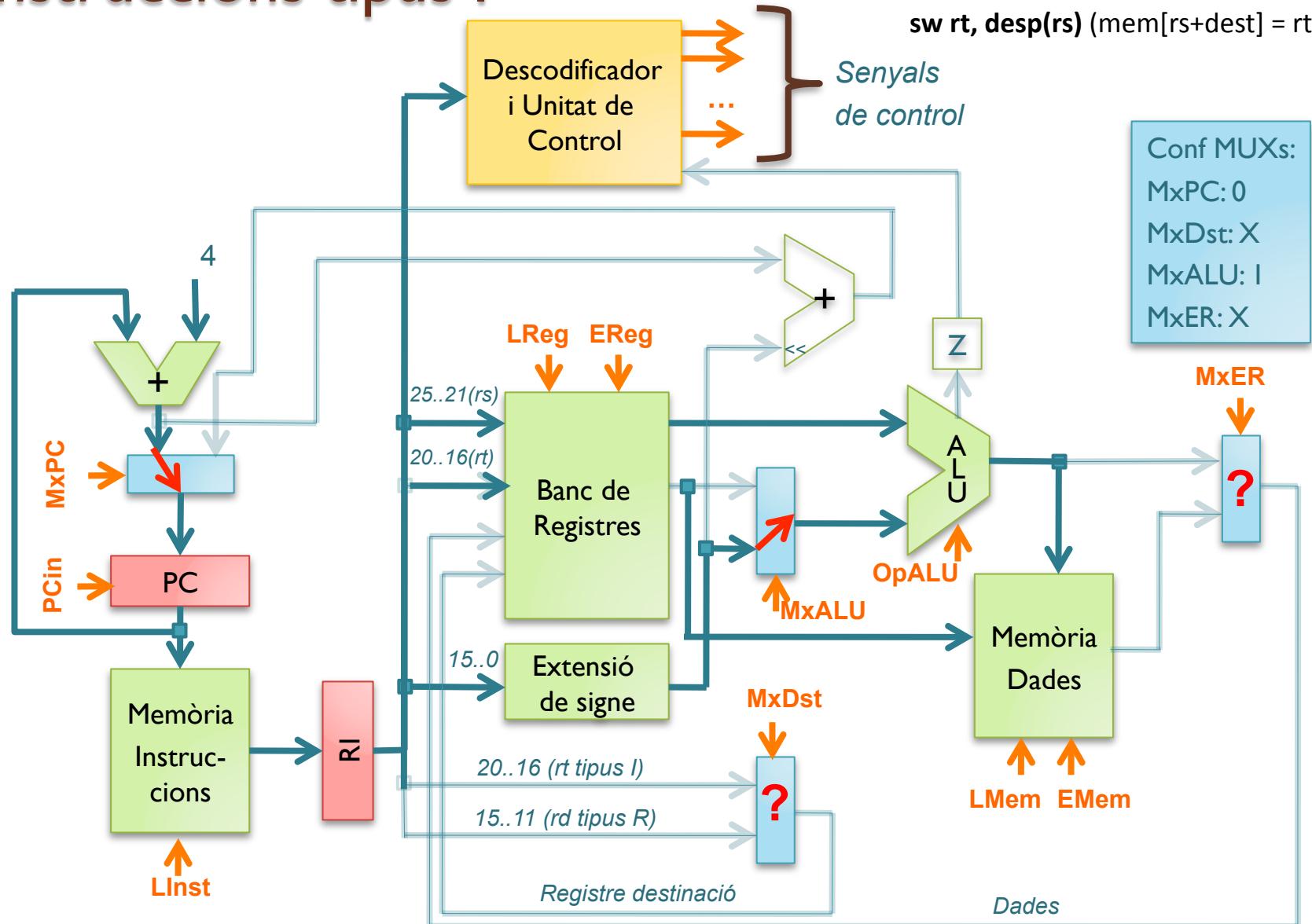
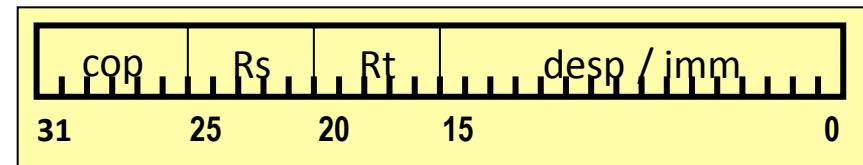
Ruta de dades per a instruccions tipus R



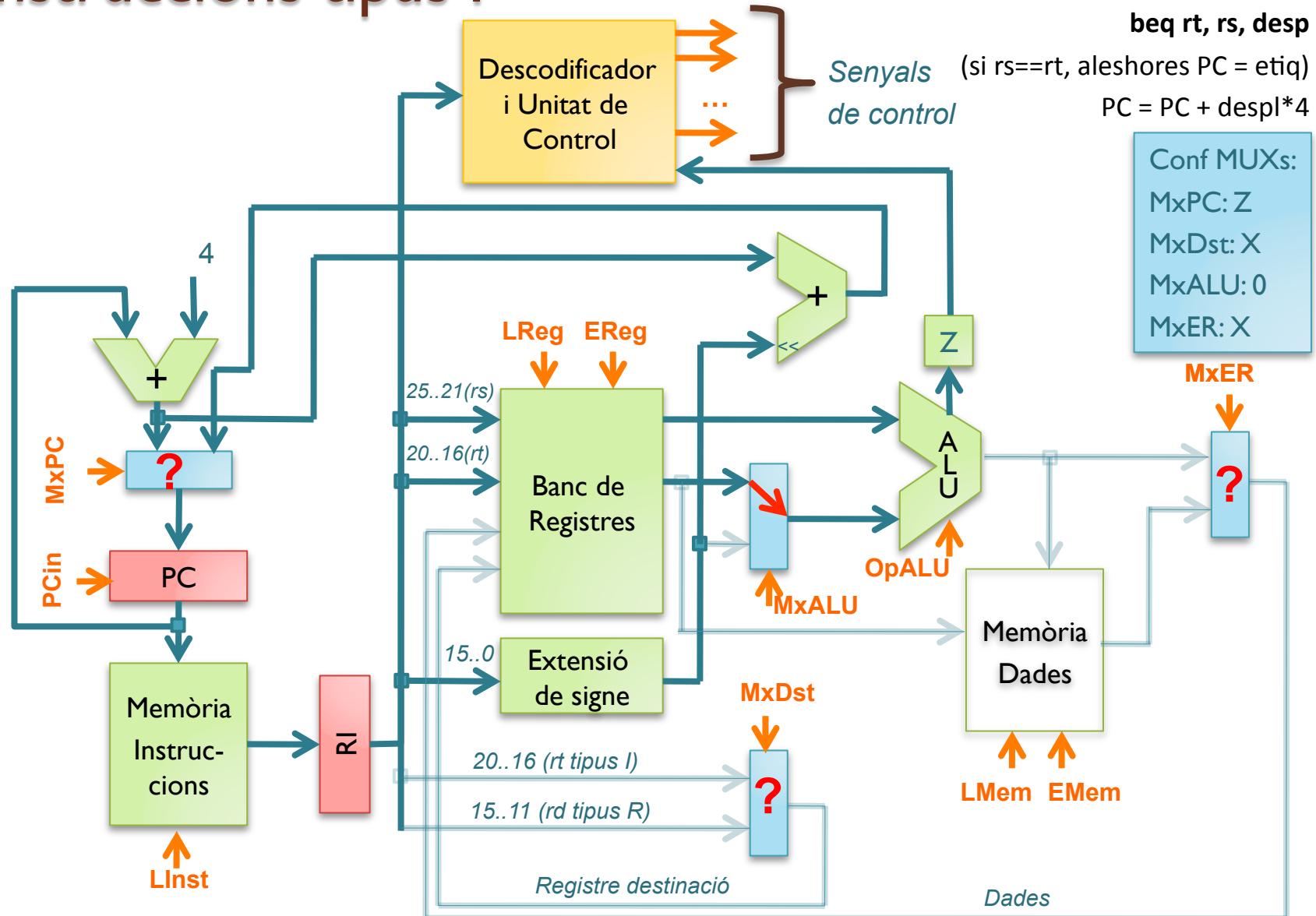
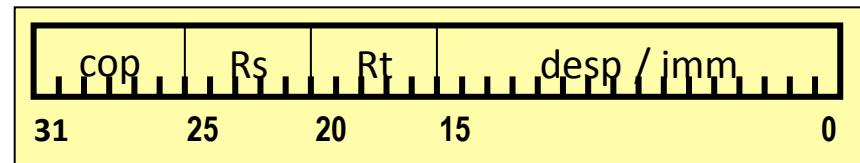
Ruta de dades per a instruccions tipus I



Ruta de dades per a instruccions tipus I



Ruta de dades per a instruccions tipus I



L'unitat de control

- Objectiu
 - ✓ Diu a la ruta de dades quines accions s'han de fer al llarg del cicle d'instrucció per a executar cada instrucció
- Passos
 - 1) Identificar els senyals de control que han d'actuar sobre la ruta de dades
 - 2) Relacionar l'activació de cadascun dels senyals amb cada instrucció
 - 3) Obtindre l'expressió lògica que defineix cada senyal de control
 - 4) Implementar el circuit

Unitat de Control: Senyals de control

| Instrucció | Form | Codi | Funció | PCin | LInst | LReg | EReg | OpALU | LMem | EMem | Multiplexors Configuració Ruta de Dades | | | | | | |
|------------------|------|--------|--------|------|-------|------|------|-------|------|------|---|-------------------|-----|------------|------|-------|-------|
| | | | | | | | | | | | Mem. Instr. | Banc de Registres | ALU | Mem. Dades | MxPC | MxALU | MxDst |
| add rd, rs, rt | R | 000000 | 100000 | 1 | 1 | 1 | 1 | 010 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | | |
| sub rd, rs, rt | R | 000000 | 100010 | 1 | 1 | 1 | 1 | 110 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | | |
| and rd, rs, rt | R | 000000 | 100100 | 1 | 1 | 1 | 1 | 000 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | | |
| or rd, rs, rt | R | 000000 | 100101 | 1 | 1 | 1 | 1 | 001 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | | |
| lw rt, desp(rs) | I | 100011 | | | 1 | 1 | 1 | 1 | 010 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| sw rt, desp(rs) | I | 101011 | | | 1 | 1 | 1 | 0 | 010 | 0 | 1 | 0 | 1 | X | X | | |
| beq rs, rs, etiq | I | 000100 | | | 1 | 1 | 1 | 0 | 110 | 0 | 0 | Z | 0 | X | X | | |



L'implementació d'aquesta taula permet la correcta execució de les instruccions en un cicle de rellotge

| | |
|-------|---------------------------|
| OpALU | Operació |
| 000 | $a \wedge b$ (and) |
| 001 | $a \vee b$ (or) |
| 010 | $a + b$ (suma aritmètica) |
| 110 | $a - b$ (resta) |
| 111 | $a < b$ (menor que) |

El processador: prestacions ruta de dades

- Ruta de dades amb els següents retards
 - ✓ Lectura/Escriptura en memòria de dades/instruccions: 2 ns
 - ✓ Lectura/Escriptura en banc de registres: 1 ns
 - ✓ Operació en la ALU: 2 ns
 - ✓ Resta de retards menyspreable
- Temps d'execució de les instruccions:
 - ✓ Aritmeticològiques: add \$t2, \$t1, \$t0
 - ✓ Lectura de memòria: lw \$t1, 0(\$t0)
 - ✓ Escriptura en memòria: sw \$t1, 0(\$t0)
 - ✓ Bot condicional: beq \$t0, \$0, bucle
- Calcula el temps de cicle i la freqüència de la ruta de dades

La taxa d'execució és d'una instrucció per cicle, però el temps de cicle resulta molt elevat (cas pitjor).

En el tema 2 utilitzarem aquesta ruta com a base per a millorar les prestacions amb la segmentació

add: 2ns + 1ns + 2ns + 1ns = 6 ns
lw: 2ns + 1ns + 2ns + 2ns + 1ns = 8 ns
sw: 2ns + 1ns + 2ns + 2ns = 7 ns
beq: 2ns + 1ns + 2ns = 5 ns

Tcicle = 8 ns;
Freqüència = 125 MHz

Apèndix: Instruccions de transferència

- Instruccions

- ✓ move, clear, mfhi, mflo, la, li, lui, li.s, li.d, mfc0, mtc0, mfc1, mtc1

```

.data 0x10000000
a: .byte 0
.text 0x00400000

move $t1, $t0      # $t1 = $t0
clear $t0          # $t0 = 0

li $t0, 23          # $t0 = 23
li.s $f0, 2.45e34   # $f0 = 2.45e34
li.d $f2, -2.33e11  # $f0|$f1 = -2.33e11

lui $t0, 0xAABB    # $t0 = 0xAABB0000

li $t0, 2            # $t0 = 2
li $t1, 4            # $t1 = 4
mult $t0, $t1        # hi|lo = 8 (2x4)
mflo $t0              # $t0 = 8 (part baixa resultat)
mfhi $t1              # $t1 = 0 (part alta resultat)

li $t0, 45            # $t0 = 45
la $t0, a             # $t0 = 0x10000000 (adr de a)

li $t0, 0x33440000   # $t0 = 0x33440000
li.s $f0, 1.0          # $f0 = 1.0 (0x3F800000)
mfc1 $t0, $f0          # $t0 = 0x3F800000 (!= 1!!!!)

li $t0, 1            # $t0 = 1
mtc1 $t0, $f0          # $f0 = 0x00000001 (!= 1.0!!!!)

mfc0 $t0, $12          # $t0 = registre d'estat
li $t0, 0            # $t0 = 0
mtc0 $t0, $12          # registre d'estat = 0

```

pseudo-instruccions en roig

Apèndix: Instruccions de càrrega/ emmagatzematge

● Instruccions

- ✓ Lectura de memòria: lb, lbu, lhu, lw, lwcl
- ✓ Escriptura en memòria: sb, sh, sw, swcl

```
.data 0x10000000
a: .word 0xFFFFFFF # a val -2
f: .float 2.45e-3
d: .double -2.34e450

.text 0x00400000
la $t0, a          # $t0 = 0x10000000
lb $t1, 0($t0)    # $t1 = 0xFFFFFFF
lbu $t2, 0($t0)   # $t2 = 0x000000FE
lh $t3, 2($t0)    # $t3 = 0xFFFFFFFF
lhu $t4, 2($t0)   # $t4 = 0x0000FFFF
lw $t5, 0($t0)    # $t5 = 0xFFFFFFFF

la $t0, a          # $t0 = 0x10000000
li $t1, 0x00        # $t1 = 0x00000000
sb $t1, 0($t0)    # a = 0xFFFFF000
sh $t1, 0($t0)    # a = 0xFFFF0000
sw $t1, 0($t0)    # a = 0x00000000
```

```
la $t0, f          # $t0 = 0x10000004
lwcl $f0, 0($t0)  # $f0 = 2.45e-3
la $t0, d          # $t0 = 0x10000008
lwcl $f0, 0($t3)  #
lwcl $f1, 4($t3)  # $f0|$f1 = -2.32e450

la $t0, f          # $t0 = 0x10000004
li.s $f0, 87.5    # $f0 = 87.5
swcl $f0, 0($t0)  # f = 87.5

la $t0, d          # $t0 = 0x10000008
li.d $f0, 345.3e-4 # $f0|$f1 = 345.3e-4
swcl $f0, 0($t0)  #
swcl $f1, 4($t0)  # d = 345.3e-4
```

Apèndix: Instruccions aritmètiques

● Instruccions

- ✓ add, addi, addu, addiu, sub, subu
- ✓ mult, multu, div, divu, mul, div, rem

```
.text 0x00400000
    li $t0, 3          # $t0 = 3
    li $t1, 6          # $t1 = 6
    add $t2, $t0, $t1 # $t2 = 9
    addi $t3, $t0, -4 # $t3 = -1
    addiu $t4, $t0, -4 # $t4 = 3 + 0x0000FFFC = 65535
    li $t0, 3          # $t0 = 3
    li $t1, -4         # $t1 = -4
    addu $t5, $t0, $t1 # $t5 = 3 + 0x0000FFFC = 65535

    sub $t6, $t0, $t1 # $t6 = 7
    subu $t7, $t0, $t1 # $t7 = 7 (desbordament)

    li $t0, 6          # $t0 = 6
    li $t1, 4          # $t1 = 4
    mult $t0, $t1      # hi|lo = 24
    div $t0, $t1       # lo = 1 (6/4), hi = 2 (6%4)

    mul $t2, $t0, $t1 # $t2 = 24
    div $t2, $t0, $t1 # $t2 = 1 (6/4)
    rem $t2, $t0, $t1 # $t2 = 2 (6%4)
```

Apèndix: Instruccions lògiques

● Instruccions

- ✓ and, andi, or, ori, xor, xori, nor
- ✓ sll, sllv, srl, sra, srlv

```
.text 0x00400000
li $t0, 0x00FF00FF          # $t0 = 0x00FF00FF
li $t1, 0xFF00FF00          # $t1 = 0xFF00FF00

and $t2, $t0, $t1           # $t2 = 0x00000000
andi $t2, $t0, 0xFFFF        # $t2 = 0x000000FF
or $t2, $t0, $t1             # $t2 = 0xFFFFFFFF
ori $t2, $t0, 0xFFFF         # $t2 = 0x00FFFFFF
xor $t2, $t0, $t1            # $t2 = 0xFFFFFFFF
xori $t2, $t0, 0x00FF        # $t2 = 0x00FF0000

sll $t2, $t0, 1              # $t2 = 0x01FE01FE

li $t2, 4                   # $t2 = 4
sllv $t3, $t0, $t2           # $t3 = 0x0FF00FF0

srl $t2, $t1, 1              # $t2 = 0x7F807F80
sra $t2, $t1, 1              # $t2 = 0xFF807F80

li $t2, 4                   # $t2 = 4
srlv $t3, $t1, $t2           # $t3 = 0x0FF00FF0
```

Apèndix : Instruccions de comparació i bot

- Instruccions

- ✓ Comparació: slt, slti, sltiu, sltu
- ✓ Bot: beq, bne, bgez, bgtz, blez, bltz, bge, bgt, ble, blt, bgtu, j, jal, jr, b, bgezal, bltzal, bal

```
.text 0x00400000
li $t0, 45          # $t0 = 45
li $t1, 0           # $t1 = 0
beq $t0, $t1, eti1 # no bota (!45==0)
bne $t0, $t1, eti2 # bota (45!=0)
bgez $t0, eti3     # bota (45>=0)
bgtz $t0, eti4     # bota (45>0)
bltz $t1, eti5     # no bota (!0<0)
blez $t1, eti5     # bota (0<=0)

li $t0, 4           # $t0 = 4
li $t1, 45          # $t1 = 45
bge $t0, $t1, eti6 # no bota (!4>=45)
bgt $t1, $t0, eti7 # bota (45>4)
ble $t0, $t1, eti8 # bota (4<=45)
blt $t1, $t0, eti9 # no bota (!45<4)

li $t0, -1          # $t0 = 0xFFFFFFFF
li $t1, 1           # $t1 = 0x00000001
bgtu $t0, $t1, eti10 # bota (0xFFFFFFFF>0x00000001)
```

```
j eti11          # bota
jal sub           # bota a sub
add $t1, $t2, $t3
...
sub:...
jr $31           # bota a l'adreça instr add

b eti11          # bota (beq $0, $0, eti11)

bal $t0, eti12    # bota a eti12
```

Apèndix: Altres instruccions

- Instruccions:

- ✓ nop, syscall, break

```
.text 0x00400000

nop          # no fa res ☺

li $v0, 10
syscall      # crida a la funció "exit" del SO

break        # punt de ruptura del programa (per a depuració)
```

Apèndix : Instruccions de Coma Flotant

- Instruccions

- ✓ add.s, sub.s, mul.s, div.s, add.d, sub.d, mul.d, div.d
- ✓ cvt.s.w, cvt.d.w, cvt.d.s, cvt.w.s, cvt.s.d, cvt.w.d
- ✓ c.lt.s, c.lt.d, bc1t, bc1f

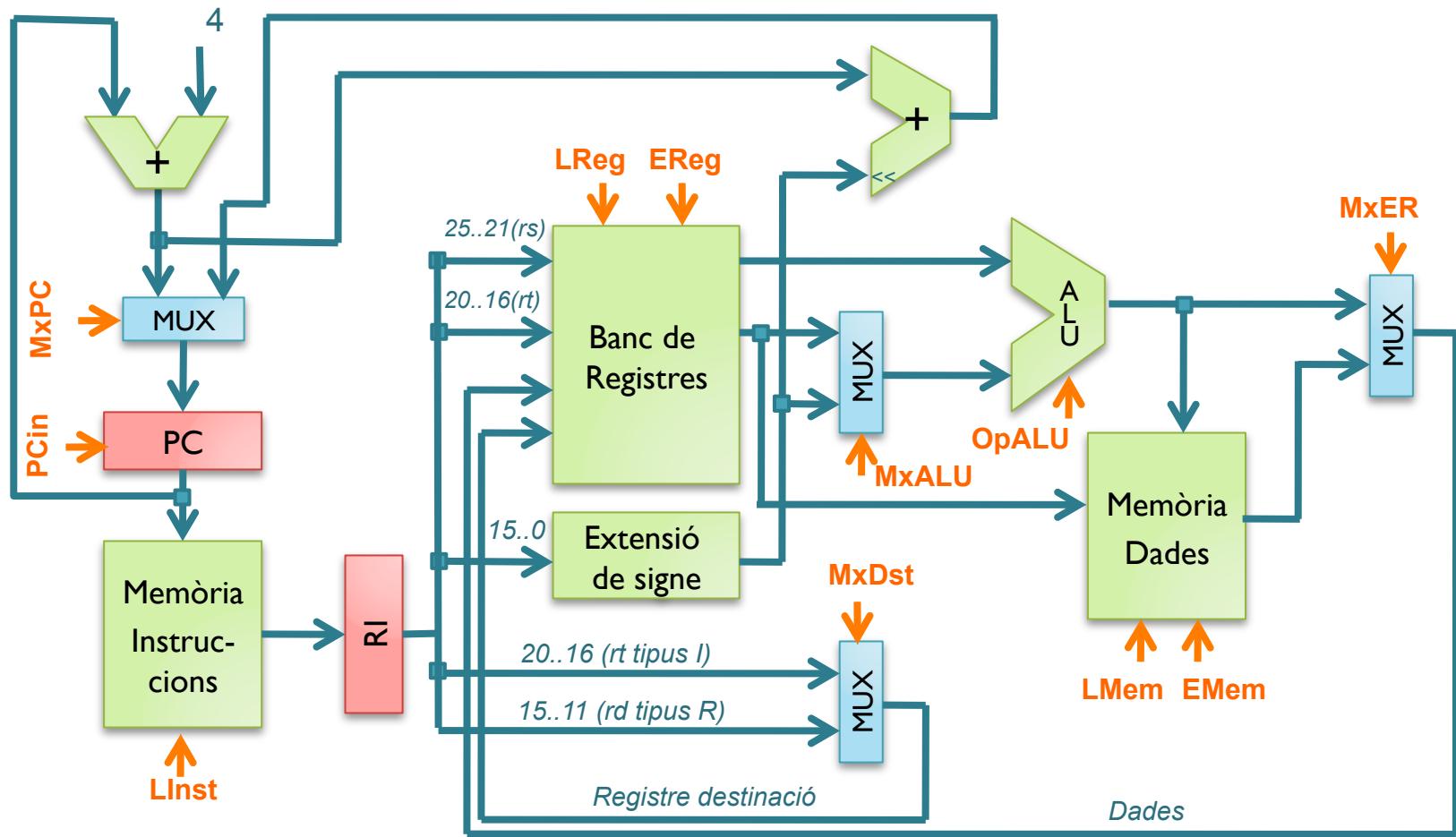
```
.text 0x00400000

li.s $f0, 2.3      # $f0 = 2.3
li.s $f1, 3.4      # $f1 = 3.4
add.s $f2, $f0, $f1 # $f2 = 5.7

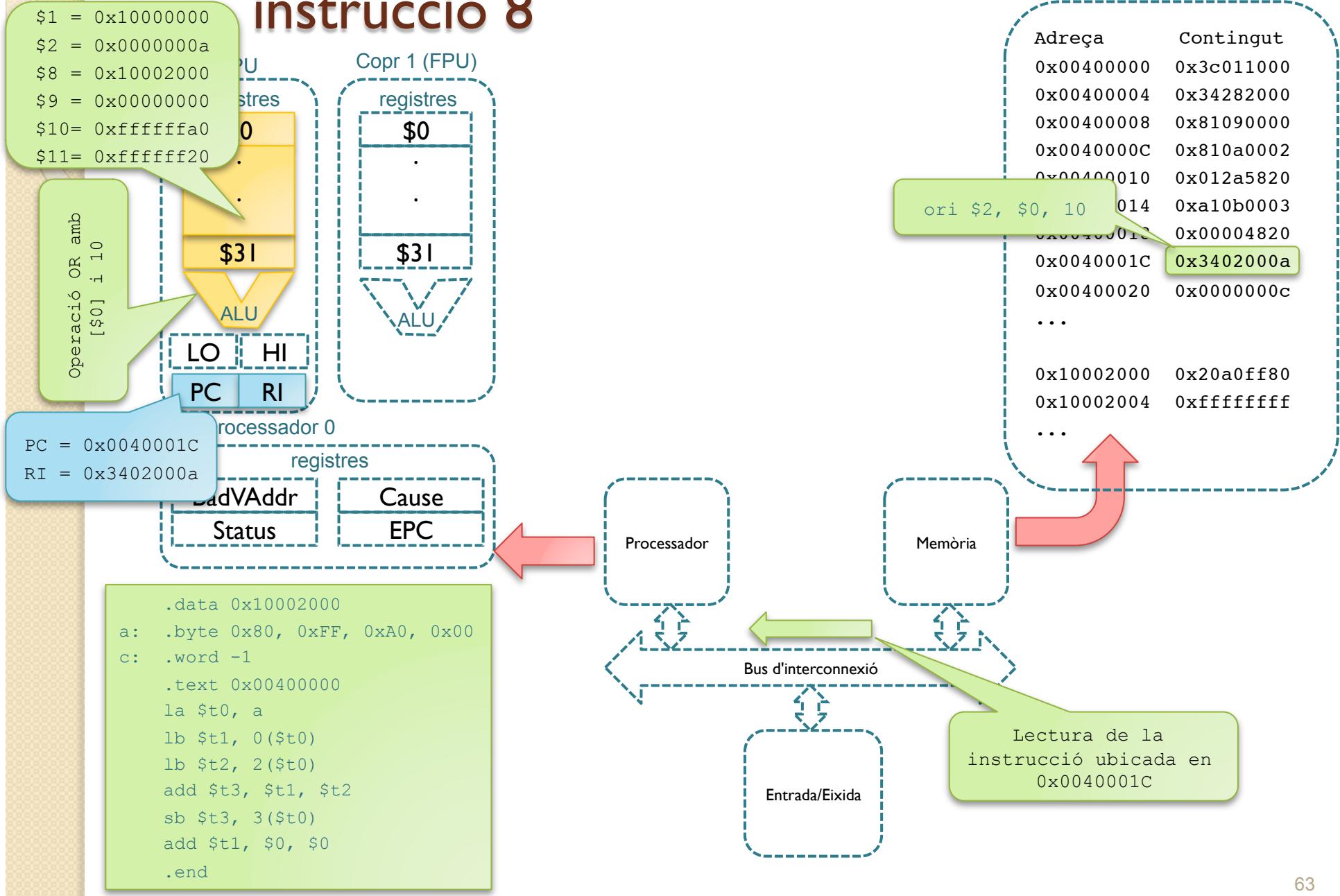
mul.s $f3, $f0, $f1 # $f3 = 7.82

li $t0, 1          # $t0 = 1
mtc1 $t0, $f0       # $f0 = 0x00000001
c.s.w $f0, $f0       # $f0 = 0x3F800000 = 1.0

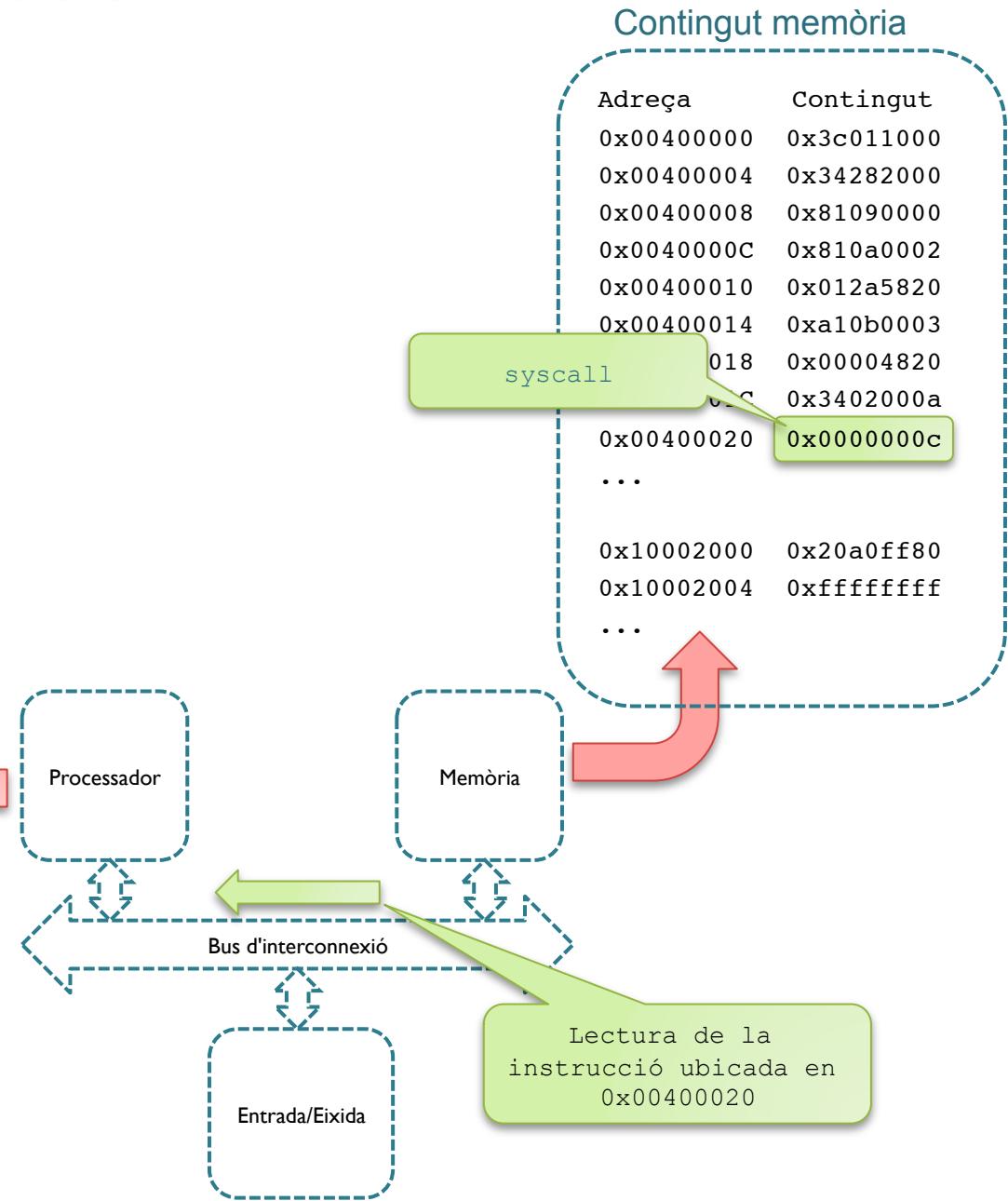
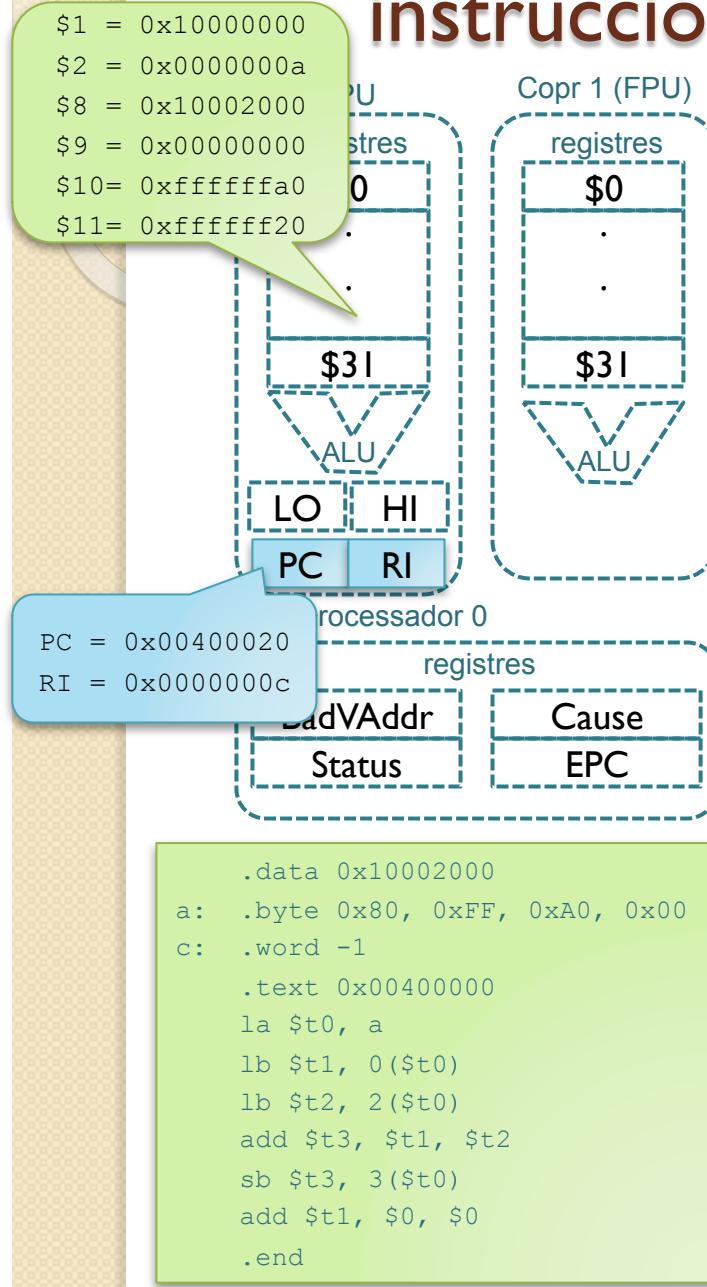
li.s $f0, 2.0      # $f0 = 2.0
li.s $f1, 3.0      # $f1 = 3.0
c.lt.s $f0, $f1      # compara $f0 i $f1
bc1t eti1          # bota (2.0<3.0)
```



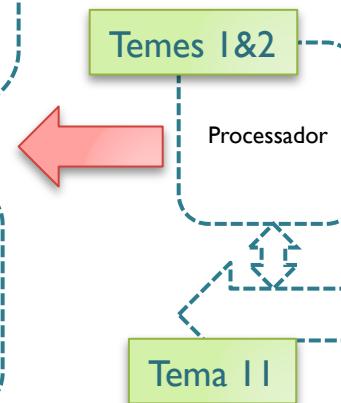
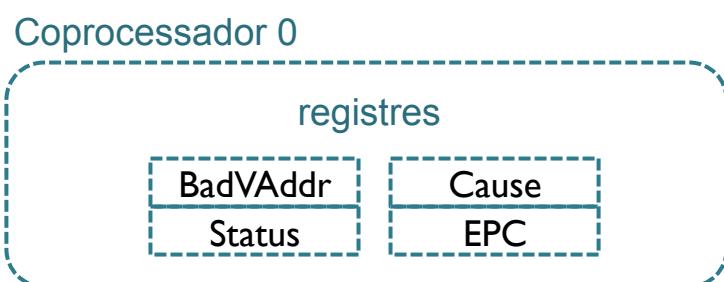
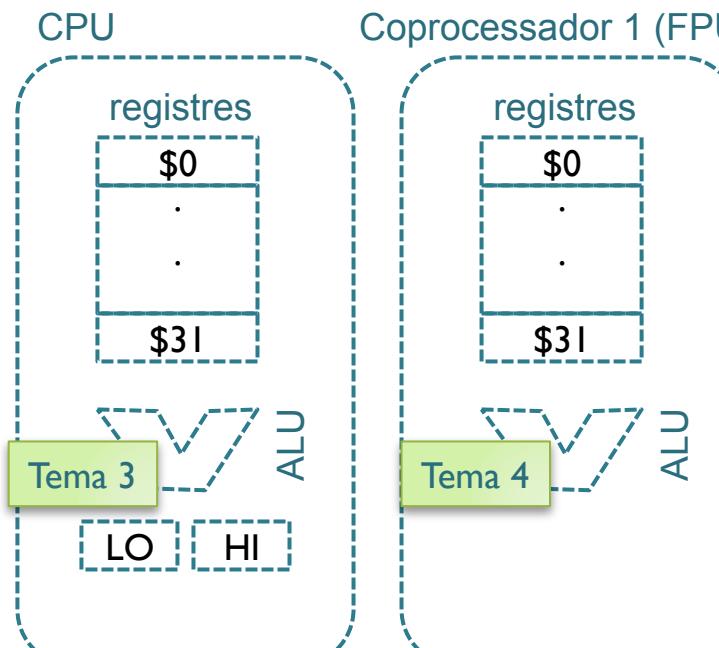
Exemple: execució instrucció 8



Exemple: execució instrucció 9



Arquitectura MIPS32 en l'assignatura



| Contingut memòria | |
|-------------------|------------|
| Adreça | Contingut |
| 0x00400000 | 0x3c011000 |
| 0x00400004 | 0x34282000 |
| 0x00400008 | 0x81090000 |
| 0x0040000C | 0x810a0002 |
| 0x00400010 | 0x012a5820 |
| ... | |
| 0x10002000 | 0x00a0ff80 |
| 0x10002004 | 0xfffffff |
| ... | |

L'arquitectura MIPS32 s'empra com exemple en tota l'assignatura mitjançant exemples programes en codi assemblador

Exemple: codificació, assemblatge i càrrega

Definició d'etiquetes de dades:

a: 0x10002000

c: 0x10002004

Definició de segment de dades, amb inici en adreça 0x10002000

```
.data 0x10002000
a: .byte 0x80, 0xFF, 0xA0, 0x00
c: .word -1
```

Reserva i inicialització de posicions de memòria
 0x10002000 – 0x10002007
 0x10002000: 0x80
 0x10002001: 0xFF
 0x10002002: 0xA0
 0x10002003: 0x00
 0x10002004: 0xFFFFFFFF

Suma de \$t1 i \$t2 i deixa el resultat en \$t3

Emmagatzema el byte de menor pes de \$t3 en memòria, en l'adreça resultada de sumar \$t0 i 3

A més a més posa a zero el registre \$t1

Definició de segment de codi amb inici en adreça 0x00400000

```
.text 0x00400000
la $t0, a
lb $t1, 0($t0)
lb $t2, 2($t0)
add $t3, $t1, $t2
sb $t3, 3($t0)
add $t1, $0, $0
.end
```

Càrrega del registre \$t0 amb valor etiqueta a (0x10002000)

Lectura de 2 bytes de memòria en adreces 0+\$t0 i 2+\$t0 i càrrega en \$t1 i \$t2

Directiva de fi del segment de codi