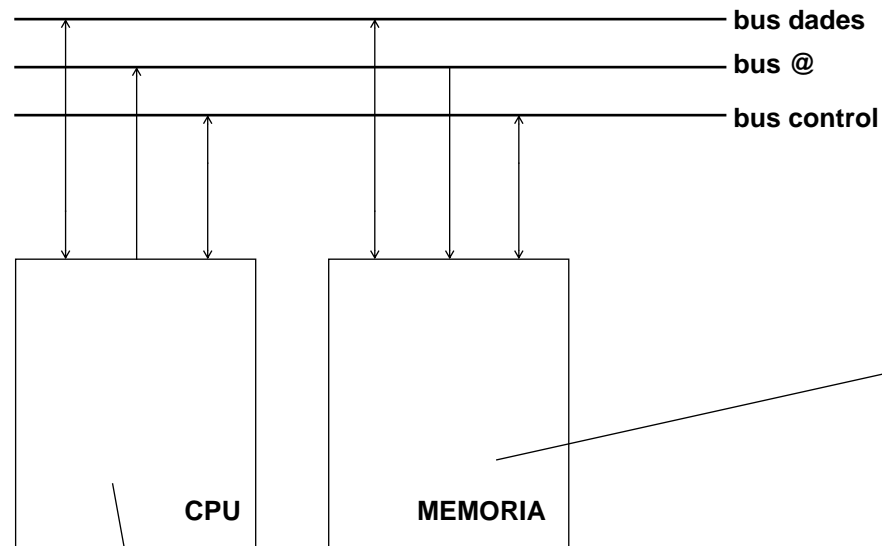


Part III: Interpretació del nivell de LM

Arquitectura de Computadors i Sistemes Operatius I
Curs 04/05



Com a resultat la memòria envia al processador:

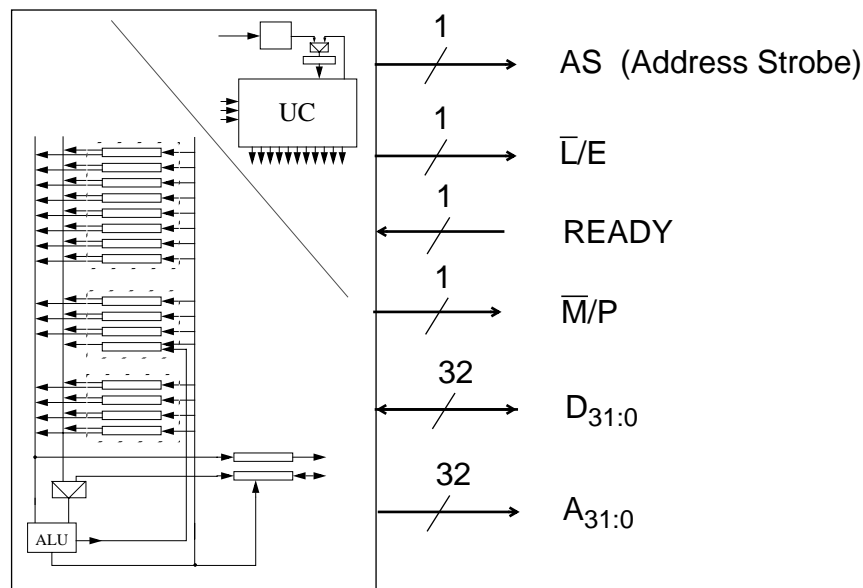
- la informació sol.licitada (en cas de lectura)
- indicació de que l'accés ha finalitzat

Per realitzar un accés a memòria, el processador ha de indicar:

- adreça on es vol accedir
- tipus d'accés que es vol fer (lectura o escriptura)
- mida de la informació (byte, word, longword)
- la informació (en cas d'escriptura)
- inici de l'accés

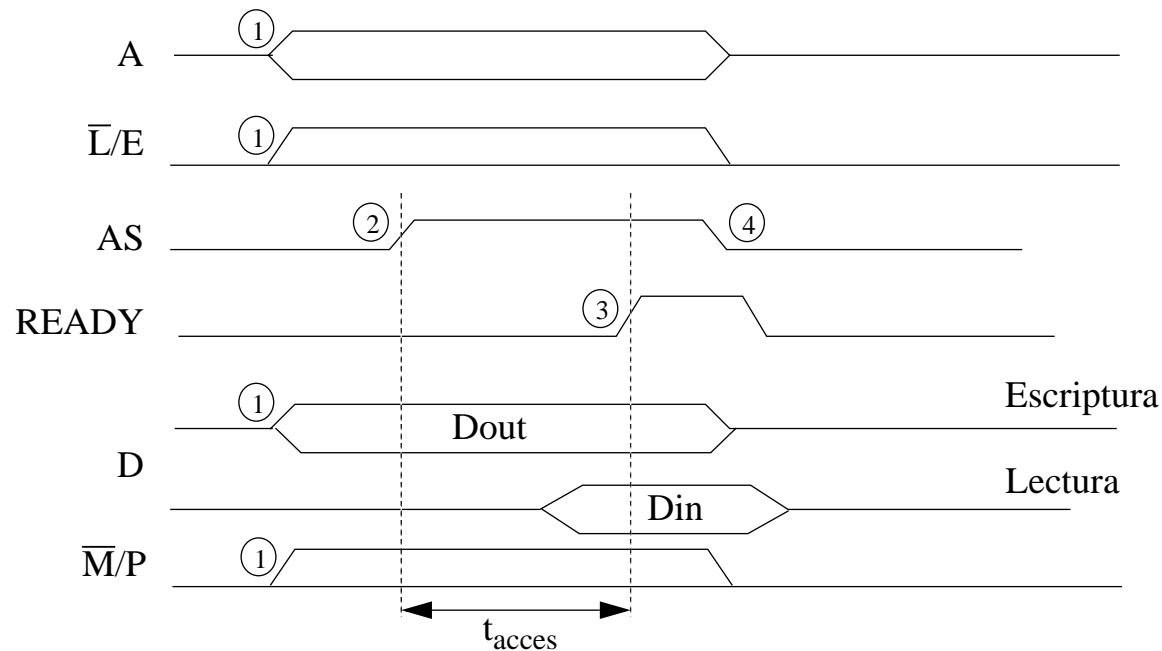
- Senyals de la CPU per suportar la transferència amb memòria:

PROCESSADOR



- $A_{31:0}$ codifiquen l'adreça que es vol accedir. Cada posició conté una paraula de 32 bits.
- $D_{31:0}$ és el bus bidireccional de dades.
- \bar{L}/E indica si l'accés és de lectura (0) o d'escriptura (1)
- AS : el processador l'activa a 1 per indicar que es pot iniciar l'accés a memòria
- READY : la memòria l'activa per indicar al processador que l'accés ha finalitzat
- \bar{M}/P : indica si l'accés és a l'espai de memòria (0) o a l'espai d'entrada/sortida (1)

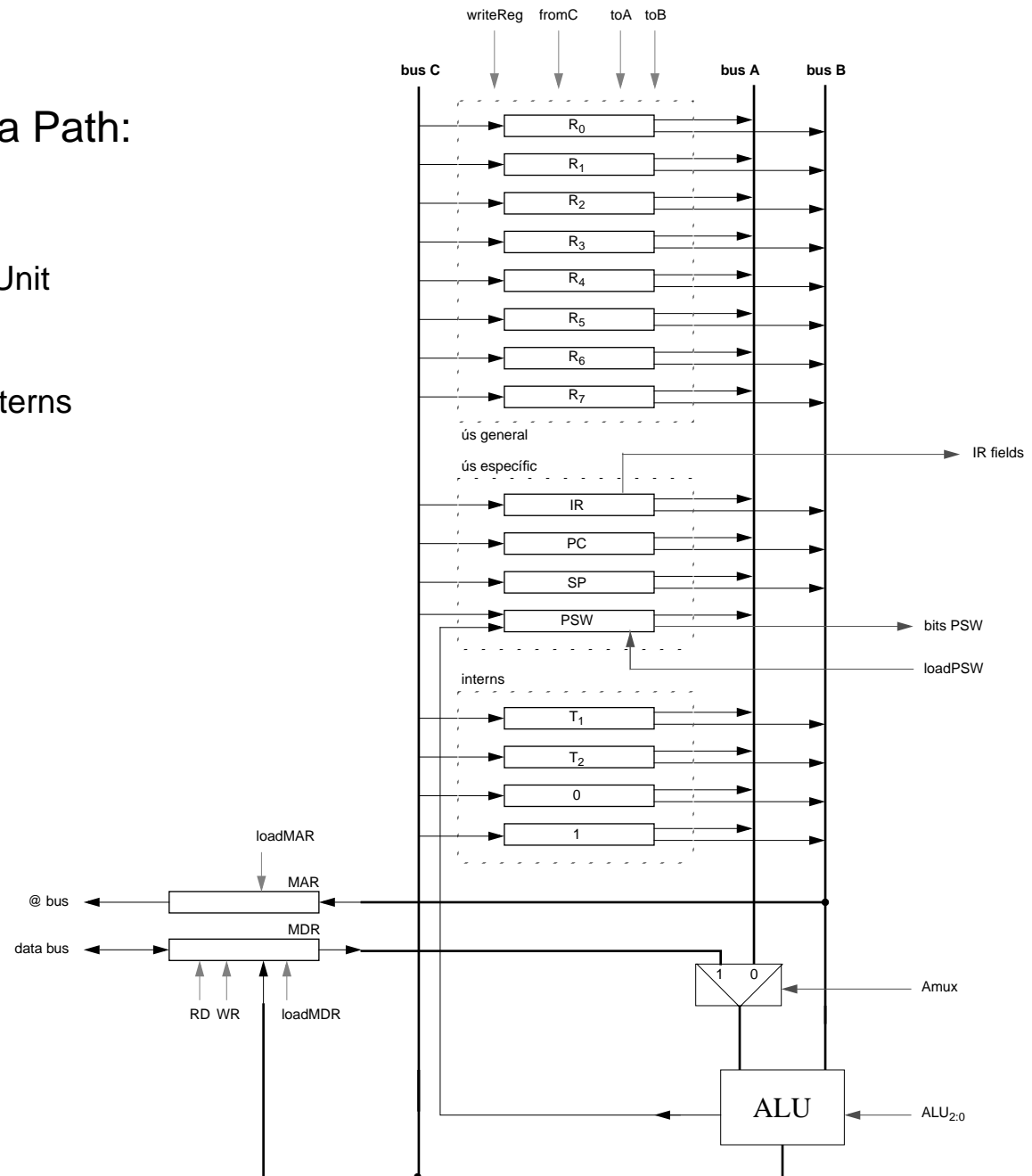
- Cronograma d'accés a memòria:



- ① Indicar l'adreça (A), tipus d'accés (\bar{L}/E), espai al que s'accedeix (\bar{M}/P) i dada (en cas d'escriptura).
- ② Activar AS a 1 per tal que s'iniciï l'accés a memòria.
- ③ En cas de lectura, esperar a que la memòria indiqui (READY=1) que la dada llegida ja està disponible al bus D.
En cas d'escriptura, esperar a que la memòria indiqui (READY=1) que la dada ja s'ha escrit a memòria.
- ④ Desactivar el senyal de petició d'accés (AS=0).

- Elements bàsics del Data Path:

- ✓ Banc de registres
- ✓ Arithmetic & Logic Unit
- ✓ Camins de dades
- ✓ Accés als busos externs



- Senyals de control del Data Path:

- ✓ **ALU_{2:0}** : selecció del tipus d'operació a realitzar a l'ALU

000	- out = A + B	- suma
001	- out = B - A	- resta
010	- out = A	- A
011	- out = A or B	- OR
100	- out = \overline{A} xor B	- XOR
101	- out = \overline{A}	- NOT
110	- out = A >>	- rshift
111	- out = << A	- lshift

- ✓ **loadPSW** : càrrega del registre paraula d'estat en funció del resultat de la ALU
- ✓ **toA** : selecció del registre que surt pel bus A
- ✓ **toB** : selecció del registre que surt pel bus B
- ✓ **fromC** : selecció del registre que es carrega amb el contingut del bus C
- ✓ **writeReg** : permetre l'escriptura al banc de registres
- ✓ **Amux** : seleccionar entre el contingut del bus A (Amux=0) o el contingut del MDR (Amux=1)
- ✓ **RD, WR** : control del MDR (RD=1, WR=0 vol dir data_bus -> MDR; RD=0, WR=1 vol dir MDR -> data_bus)
- ✓ **loadMDR** : càrrega del MDR amb el contingut del bus C
- ✓ **loadMAR** : càrrega del MAR amb el contingut del bus B

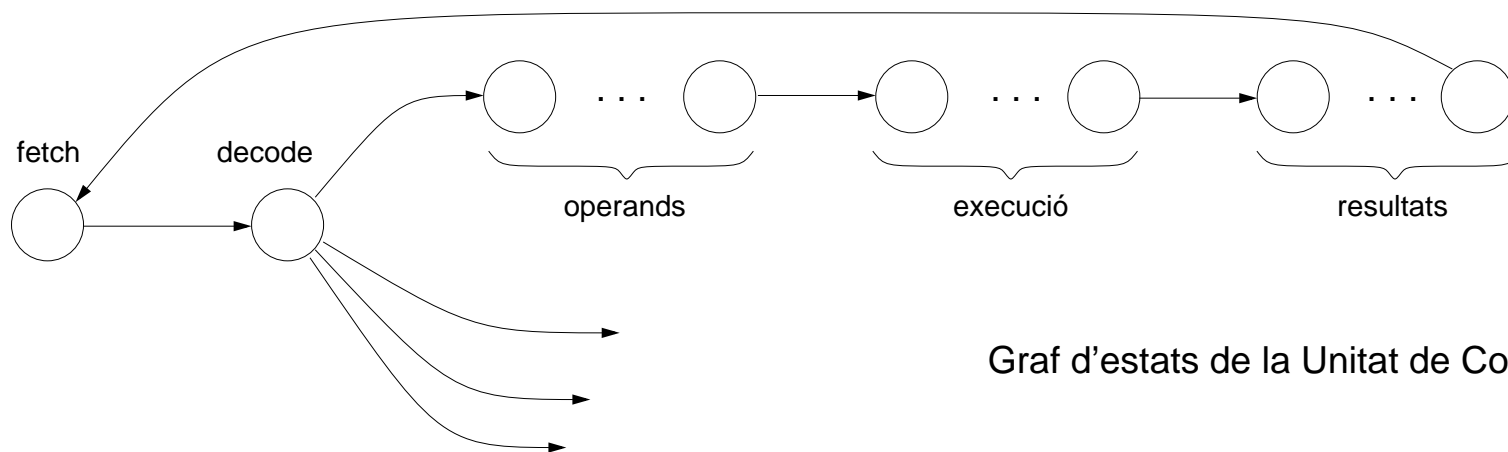
- Fases en l'execució d'una instrucció:

- ✓ Fetch: Anar a memòria a cercar la següent instrucció a executar:

$$IR \leftarrow \text{mem}[\text{PC}]$$

$$\text{PC} \leftarrow \text{PC} + 1$$

- ✓ Decode: esbrinar quina instrucció volem executar.
- ✓ Cerca d'operands: anar a memòria a cercar tots aquells operands que ens fan falta per executar la instrucció, amb els modes d'adreçament corresponents.
- ✓ Execució: Executar l'operació indicada a la instrucció, i activar els bits de la paraula d'estat.
- ✓ Escriptura de resultats: Escriure resultats a memòria, si és necessari.



Graf d'estats de la Unitat de Control

- Valor per defecte dels senyals de control

```
ALU <- X // loadPSW <- 0 // toA <- X // toB <- X // fromC <- X // writeReg <- 0 // Amux <- X //
```

```
RD <- 0 // WR <- 0 // loadMDR <- 0 // loadMAR <- 0 //
```

- Exemple: fase de fetch de qualsevol instrucció

```
toB <- PC // loadMAR <- 1 // toA <- 1 // Amux <- 0 // ALU <- suma // fromC <- PC // writeReg <- 1 ///
```

```
RD <- 1 ///
```

```
Amux <- 1 // ALU <- A // fromC <- IR // writeReg <- 1 ///
```

- Exemple: fase de cerca d'operand en modalitat indirecte MOV [R1], R0

```
toA <- R0 // Amux <- 0 // ALU <- A // loadPSW <- 1 // loadMDR <- 1 // toB <- R1 // loadMAR <- 1 ///
```

```
WR <- 1 ///
```

- Exemple: fase d'execució d'una instrucció del tipus ADD R2, R1

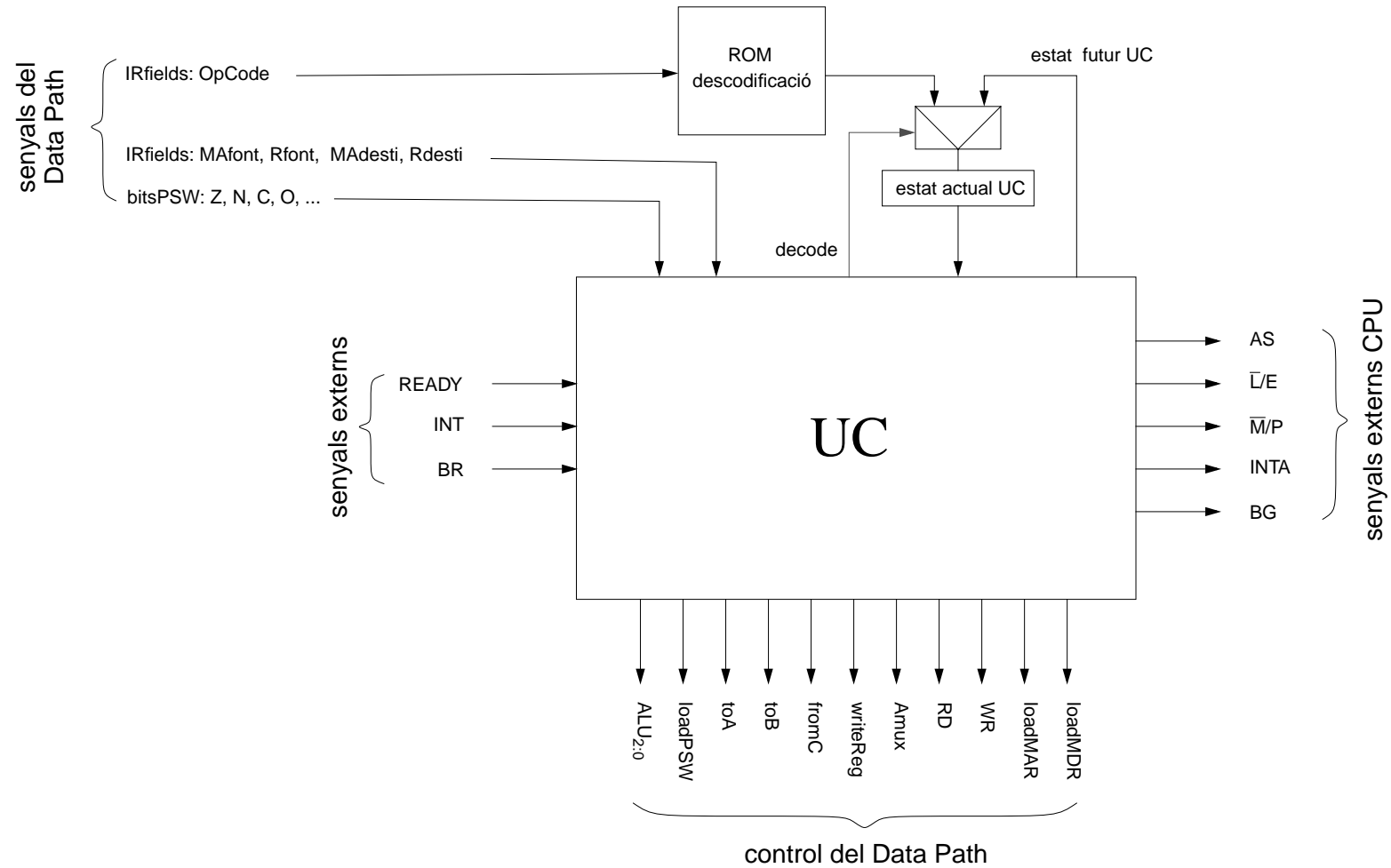
```
toA <- R1 // toB <- R2 // Amux <- 0 // ALU <- suma // loadPSW <- 1 // writeReg <- 1 // fromC <- R2 ///
```

- La Unitat de Control és un sistema seqüencial que:
 - ✓ Controla el funcionament de tot el Data Path
 - ✓ Genera el seqüenciament de les diferents fases d'execució i de cadascun dels estats de que consta
 - ✓ Genera els senyals externs del processador: AS, \bar{L}/E , \bar{M}/P , INTA i BG
 - ✓ Tot l'anterior ho fa en funció de:
 - Informació procedent dels camps de la instrucció que hi ha al IR
 - Informació procedent dels bits de la paraula d'estat PSW
 - Senyals exteriors: READY, INT i BR

Instrucció (IR)

OpCode	OpFont	OpDesti
--------	--------	---------

- Senyals de la Unitat de Control:



- Exemple: Accés a memòria per escriure un resultat ADD [R2], R1

toB <- R2 // loadMAR <- 1 ///

x1: AS <- 1 // RD <- 1 // \overline{M}/P <- 0 // \overline{L}/E <- 0 // si READY = 0 llavors anar a x1 ///

toB <- R1 // Amux <- 1 // ALU <- suma // loadPSW <- 1 // loadMDR <- 1 ///

x2: AS <- 1 // WR <- 1 // \overline{M}/P <- 0 // \overline{L}/E <- 1 // si READY = 0 llavors anar a x2
sino anar fetch///

- Exemple: fase de descodificació del codi d'operació del IR

decode <- 1 ///

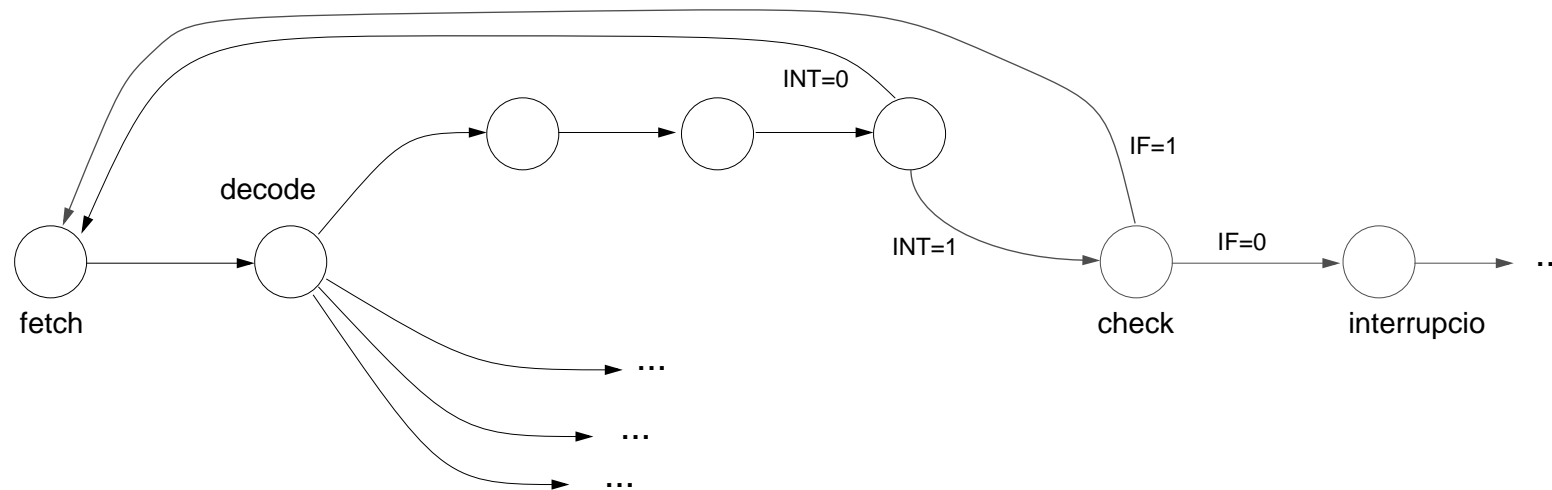
- Exemple: Accés a perifèric IN R1, [R2] (adreça del perifèric indicada a R2)

toB <- R2 // loadMAR <- 1 ///

x: AS <- 1 // RD <- 1 // \overline{M}/P <- 1 // \overline{L}/E <- 0 // si READY = 0 llavors anar a x ///

Amux <- 1 // ALU <- A // fromC <- R1 // writeReg <- 1 // anar a fetch ///

- Detectar existència d'interrupció ... (punt de control de les interrupcions)

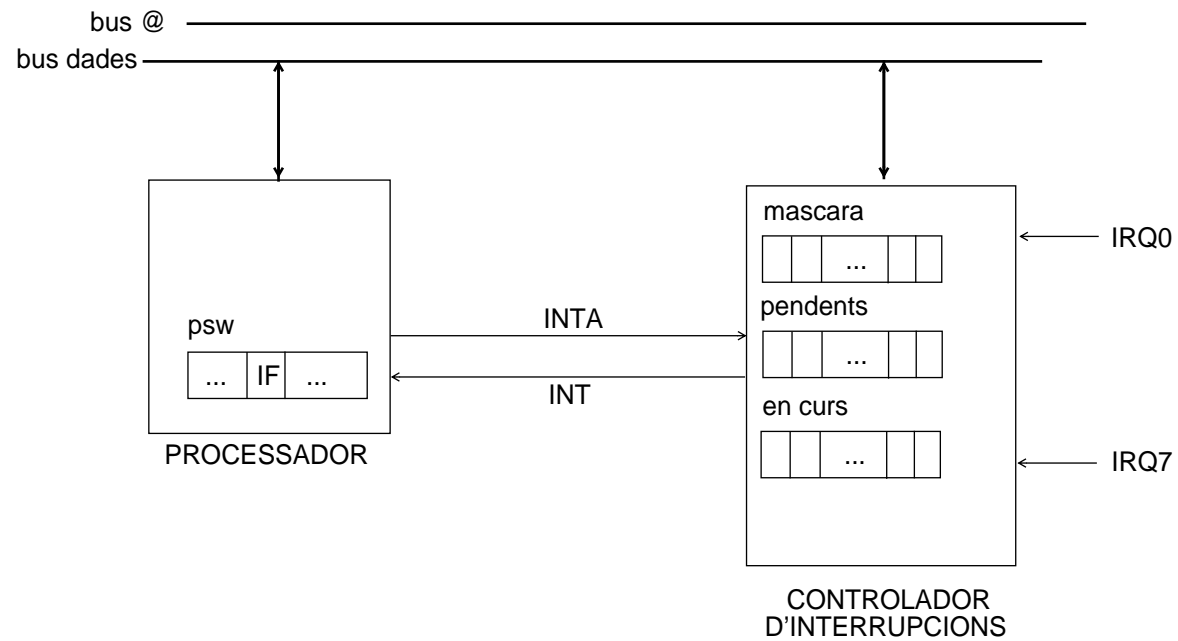


Totes les instruccions ara acaben amb un "si $INT=0$ llavors anar a fetch"

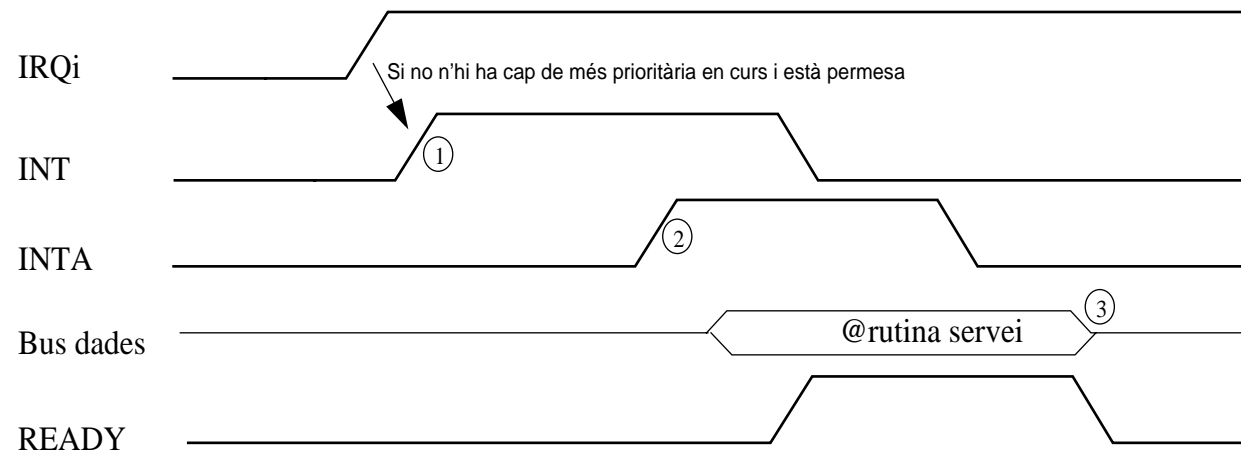
... si $INT = 0$ llavors anar a fetch sino anar a check///

Gestió de l'entrada/sortida

- Interrupcions



- Si IF=1 aleshores "interrupcions inhibides"
- Si IF=0 aleshores "interrupcions permeses"



- ① Si $IF=0$ aleshores salvar a pila PC, PSW
 $IF=1$
- ② Identificació del perifèric:
 - activació INTA
 - llegir bus de dades @rutina de servei
- ③ En fer l'IRET:
 - restaurar PSW, PC
 - anar a fetch

- ... Activar INTA, llegir @ rutina de servei i saltar:

```
z3:      INTA <- 1 // RD <- 1 // si READY = 0 llavors anar a z3 ///
        Amux <- 1 // ALU <- A // fromC <- PC // writeReg <- 1 // INTA <- 0 // anar a fetch ///
```

- Com es fa internament l'execució del IRET?:

```
iret:   toB <- SP // loadMAR <- 1 // toA <- 1 // ALU <- suma // fromC <- SP // writeReg <- 1 // Amux <- 0 ///
z11:    RD <- 1 // si READY = 0 llavors anar a z11 //  $\bar{L}/E$  <- 0 //  $\bar{M}/P$  <- 0 // AS <- 1 ///
        Amux <- 1 // ALU <- A // fromC <- PSW // writeReg <- 1 ///
        toB <- SP // loadMAR <- 1 // toA <- 1 // ALU <- suma // fromC <- SP // writeReg <- 1 // Amux <- 0 ///
z21:    RD <- 1 // si READY = 0 llavors anar a z21 //  $\bar{L}/E$  <- 0 //  $\bar{M}/P$  <- 0 // AS <- 1 ///
        Amux <- 1 // ALU <- A // fromC <- PC // writeReg <- 1 // anar a fetch ///
```

- Connexió dins del Sistema:

