

# Estructura de Computadors

Grau d'Enginyeria Informàtica  
ETSINF

Tema 2: Segmentació Bàsica  
del Processador

# Objectius

- Conéixer el concepte de segmentació com a tècnica de millora de prestacions
- Conéixer la segmentació bàsica del processador MIPS
- Detectar conflictes i riscos senzills
- Conéixer algunes de les tècniques de resolució de conflictes que apareixen en la ruta de dades segmentada del MIPS
- Saber comparar prestacions d'una ruta de dades segmentada

# Continguts i Bibliografia

- El procés de segmentació
  - ✓ Concepte
  - ✓ Diagrames temporals
  - ✓ Prestacions
- Segmentació de la ruta de dades bàsica
  - ✓ Identificació de les fases d'execució de les instruccions
  - ✓ Disseny de les etapes
  - ✓ Control de la segmentació
- Conflictes i riscos
  - ✓ Conflictes de dades
  - ✓ Conflictes de control (flux)
- Perspectiva històrica. RISC-CISC i altres alternatives

---

**Bibliografia:** Patterson, D.A., Hennessy, J.L., “Estructura y diseño de computadores. La interfaz hardware/software,” 4a edició, Ed. Reverté, 2011, Cap 4 (4.5 – 4.8)

# Concepte de Segmentació

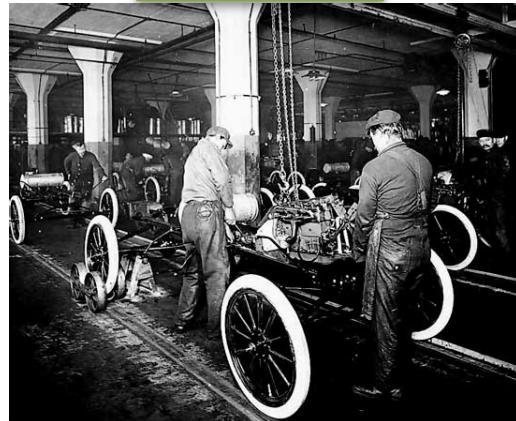
- Definició de segmentació
  - ✓ Descomposició d'un sistema que executa un determinat procés en diverses etapes, de manera que:
    - Cada etapa s'ocupa d'una part del procés global utilitzant recursos propis
    - El procés global requereix l'aplicació ordenada de totes les etapes
    - Totes les etapes treballen simultàniament (cadascuna en un procés distint)
- Objectiu: Augmentar el paral·lelisme (temporal) dels processos i per tant, augmentar la productivitat



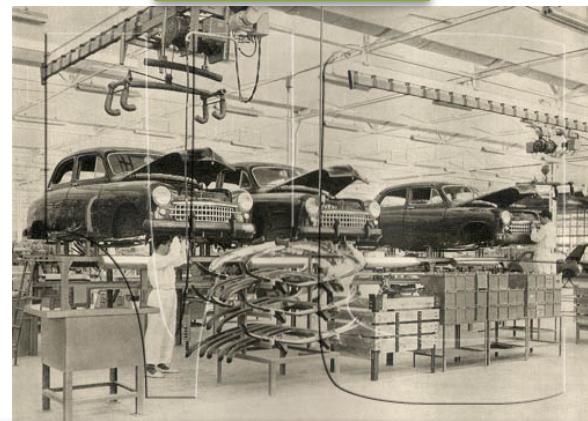
# Concepte de Segmentació

- Exemples

Ford 1907-1908



Seat Dècada 1950



Cadena de muntatge d'un cotxe

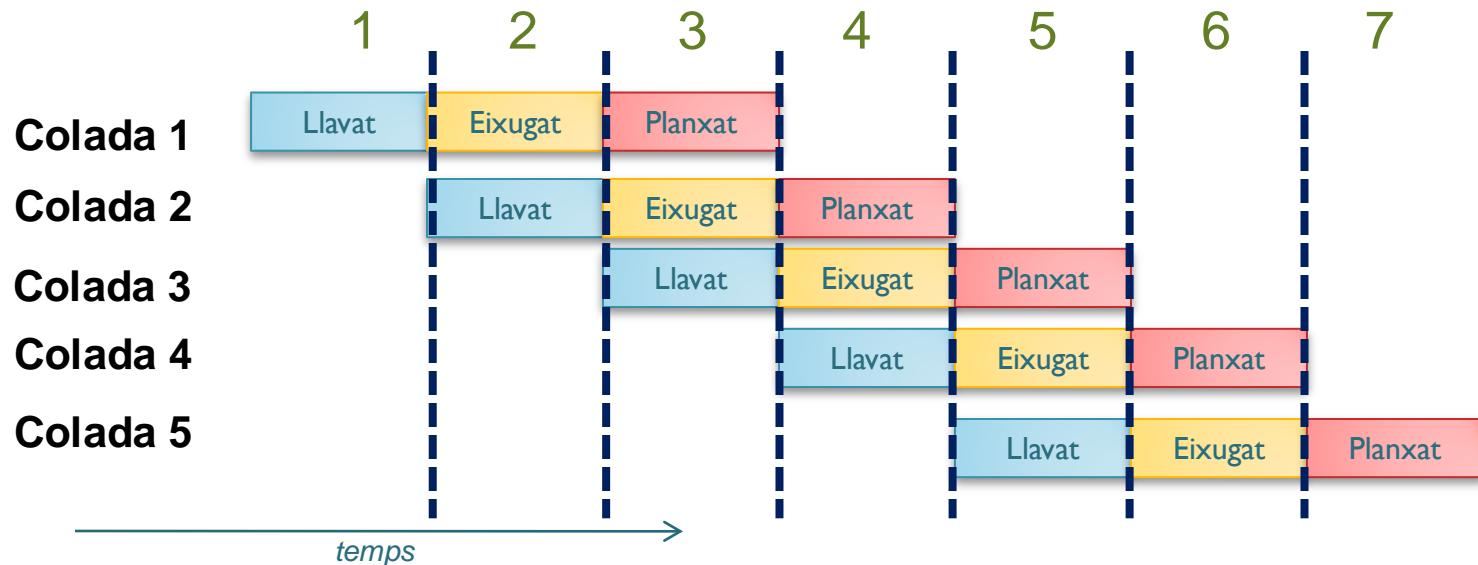
(muntatge motor, assemblatge interior, pintat, ...)

# Concepte de Segmentació

- Exemples



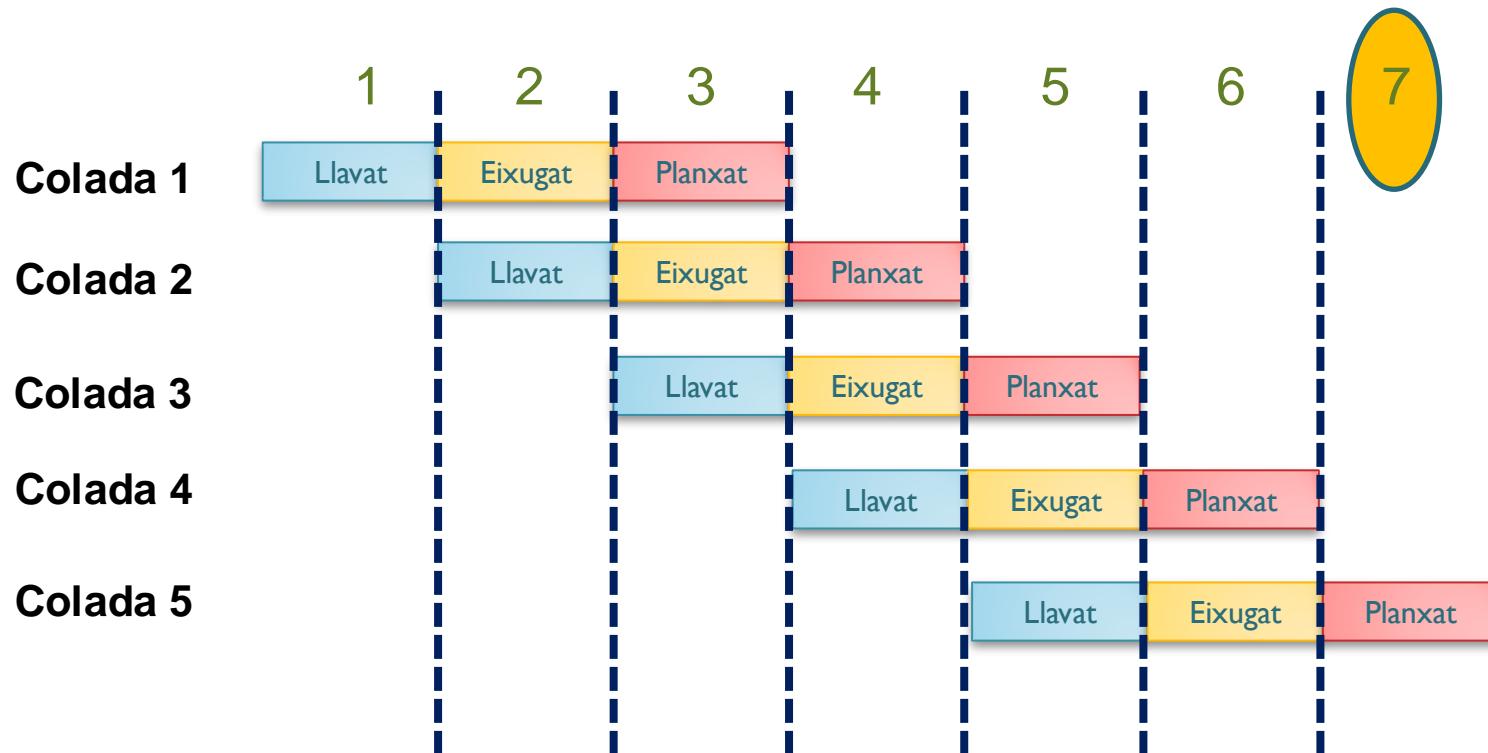
Bugaderia  
(llavat, eixugat, planxat)



# El símil de la lavandería

3 etapas  $K = 3$

5 coladas  $n = 5$



Fer 5 bugades demanarà un temps total de 7 etapes:

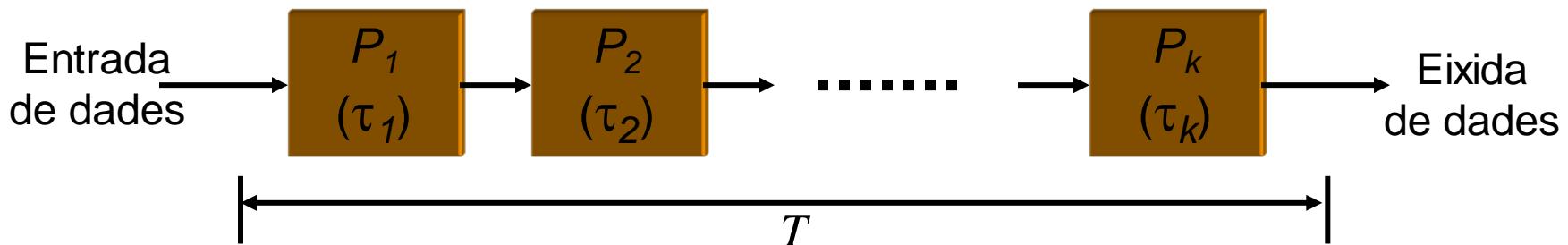
$$n + K - 1$$

# Procés de Segmentació

- Entrada

- ✓ Operador lògic P que opera sobre dades d'entrada
- ✓ P es considera constituït per k etapes  $P_i$  que operen de manera seqüencial sobre les dades
- ✓ Consideracions temporals:
  - Retard del circuit:  $T$
  - Cada etapa  $P_i$  té necessitats temporals (retard) distintes:  $\tau_i$

$$T = \sum_{i=1}^n \tau_i$$



# Procés de Segmentació

- Eixida (circuit segmentat)

- ✓ k etapes (profunditat de segmentació: k)

- Cada etapa va precedida d'un registre d'etapa (*staging latch*)

- Etapa i:

- Entrada: contingut del registre i

- Eixida: s'emmagatzema en el registre  $i+1$

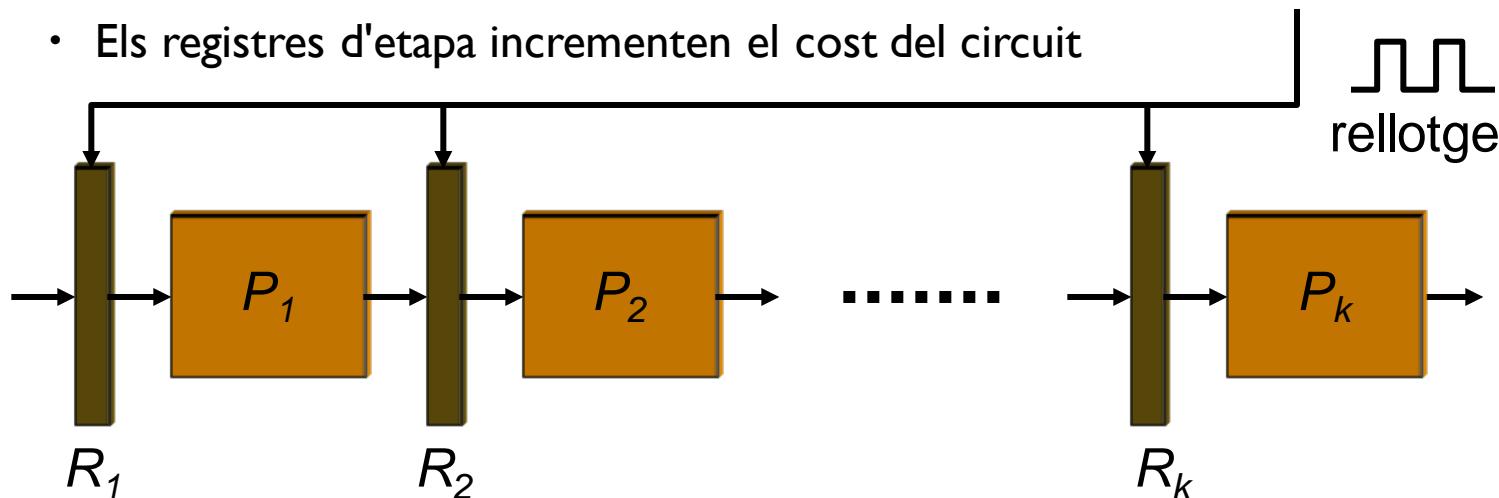
- registre d'etapa i:

- Recull les dades produïdes per l'etapa  $i-1$

- Subministra dades a l'etapa i

- Tots els registres d'etapa s'actualitzen en el mateix flanc de rellotge

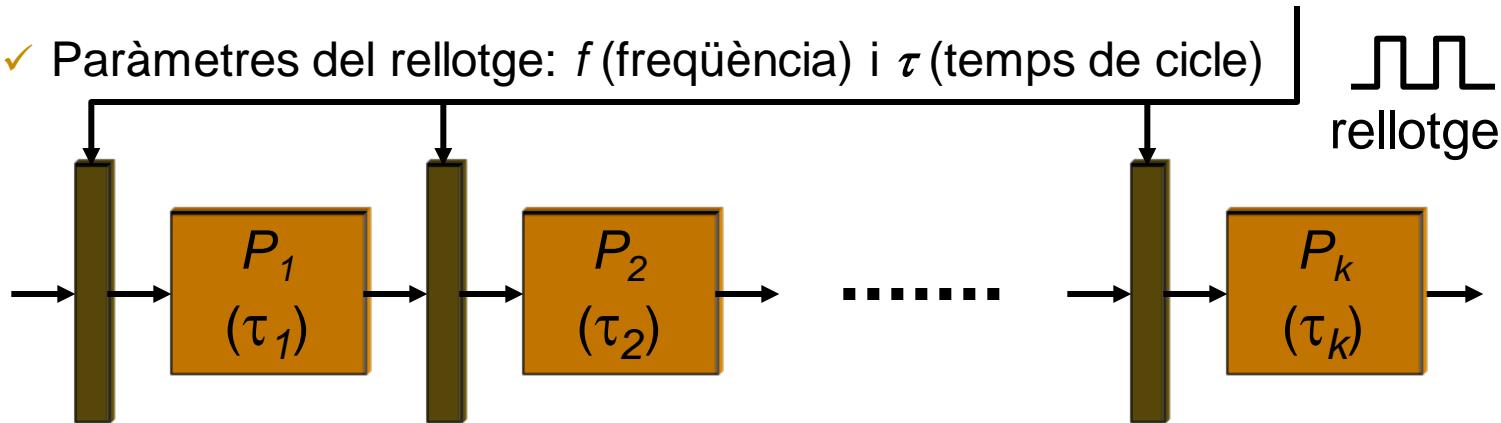
- Els registres d'etapa incrementen el cost del circuit



# Procés de Segmentació

- Sincronització del circuit

- ✓ Paràmetres del rellotge:  $f$  (freqüència) i  $\tau$  (temps de cicle)



- ✓ Consideracions temporals:

- El retard dels registres  $T_R$  s'ha de considerar en el temps de cicle:

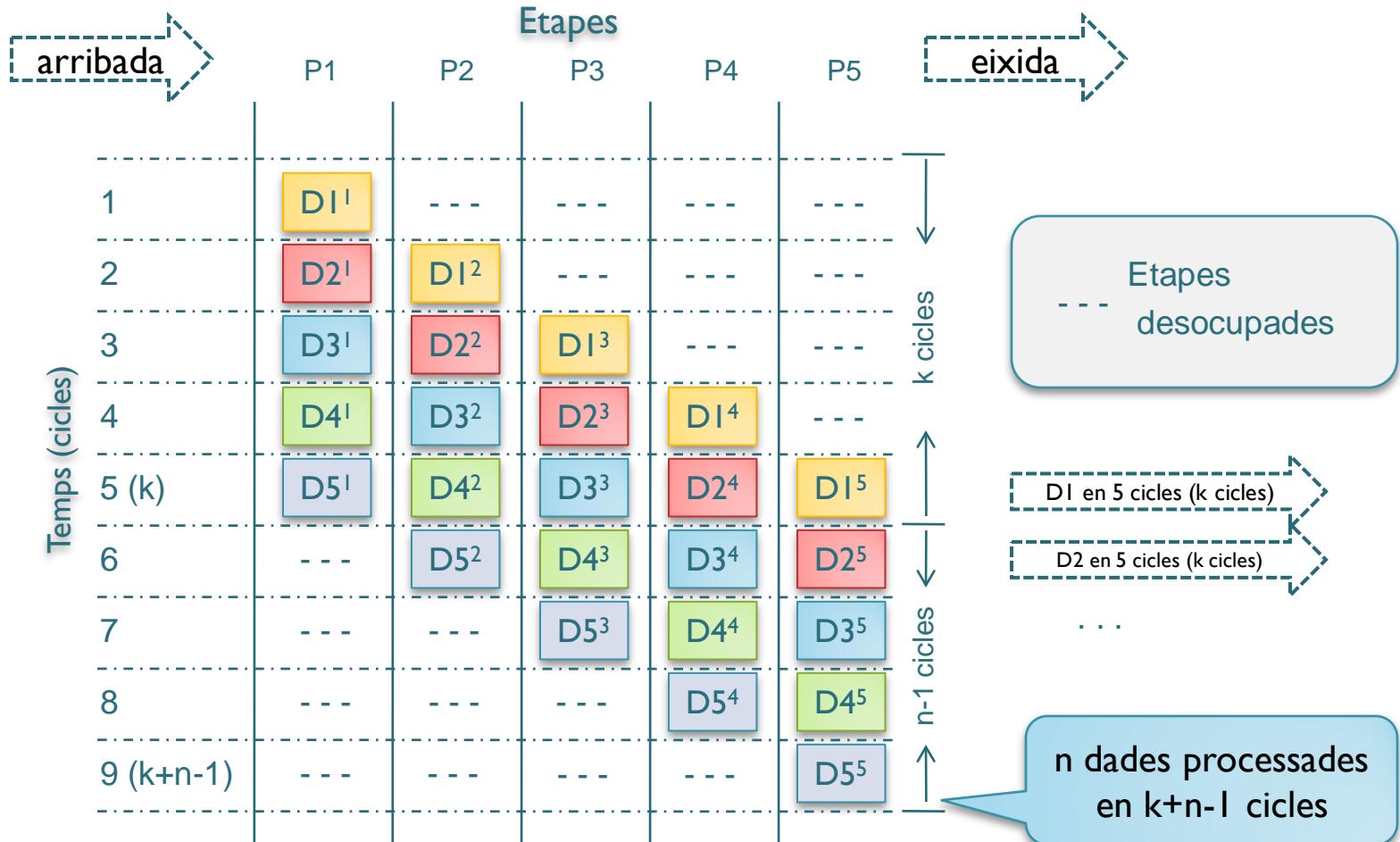
$$\tau = \max\{\tau_i\} + T_R \quad f = \frac{1}{\tau}$$

- La segmentació penalitza el temps de procés d'una operació individual

$$T \leq k \times \tau$$

# Diagrames Temporals: Diagrama Etapes/Temps

- Exemple de segmentació amb 5 etapes ( $k=5$ ) i 5 dades ( $n=5$ )

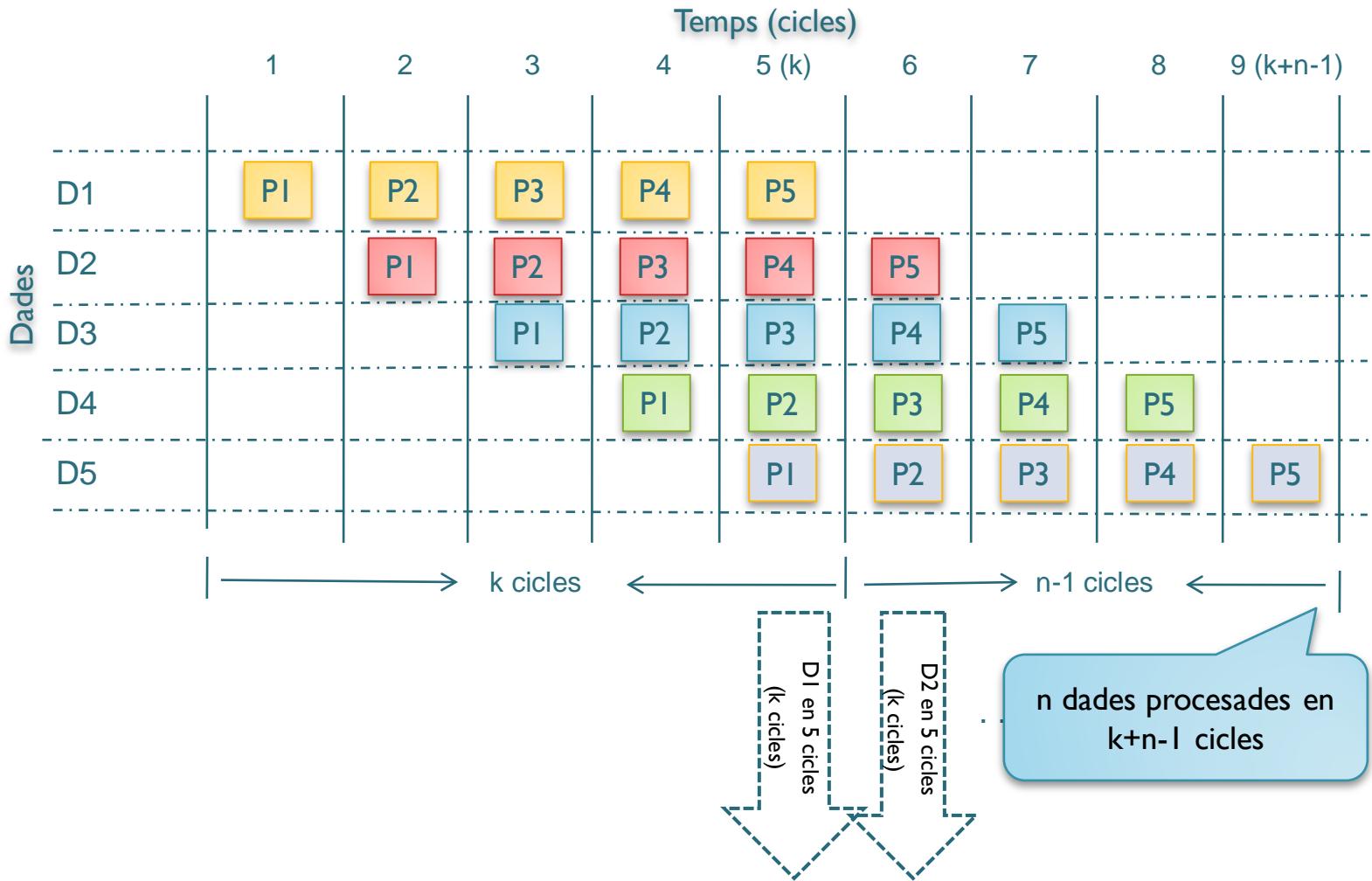


$DX^y$  representa Dada X en etapa i ( $DX^y \neq DX^z$ )

Per simplificitat, obviarem a partir d'ara les diferències de les dades entre etapes

# Diagrames Temporals: Diagrama Temps/Dades

- Exemple de segmentació amb 5 etapes ( $k=5$ ) i 5 dades ( $n=5$ )



# Prestacions de la Segmentació

- Mesures de prestacions (n dades)

- ✓ **Productivitat:** treball realitzat per unitat de temps

$$X(n) = \frac{n}{t_s(n)} = \frac{n}{(n+k-1) \times \tau} \text{ dades/temps}$$

- Incrementa:
  - Amb el nombre de dades
  - Amb la freqüència de rellotge
- Límit teòric (n suficientment gran):
  - Una operació per cicle

$$X(\infty) = \frac{1}{\tau} = f$$

- ✓ **Acceleració:** Guany de velocitat respecte al circuit no segmentat:

$$S(n) = \frac{t_{ns}(n)}{t_s(n)} = \frac{n \times T}{(n+k-1) \times \tau}$$

$$S(\infty) = \frac{T}{\tau}$$

# Prestacions de la Segmentació

- Conclusions

$$S(n) = \frac{t_{ns}(n)}{t_s(n)} = \frac{n \times T}{(n + k - 1) \times \tau}$$

- ✓ Com  $T \leq k \cdot \tau$ , si  $n=1$  aleshores  $S \leq 1$ : la segmentació no beneficia
- ✓ Com més gran és  $n$  (dades a processar), més beneficis ( $S$ )
  - Si  $n \rightarrow \infty$ , aleshores  $S \rightarrow T/\tau \leq k$
- ✓ **Guany teòric màxim**
  - Si  $T = k \cdot \tau$  (cas ideal), el guany seria  $S=k$ . Això s'acompliria només si totes les etapes tingueren igual retard ( $\tau_i$ ) i el retard dels registres de segmentació ( $T_R$ ) fóra zero
  - En principi, i amb suficients dades, com més xicotet siga  $\tau$  més acceleració obtindrem

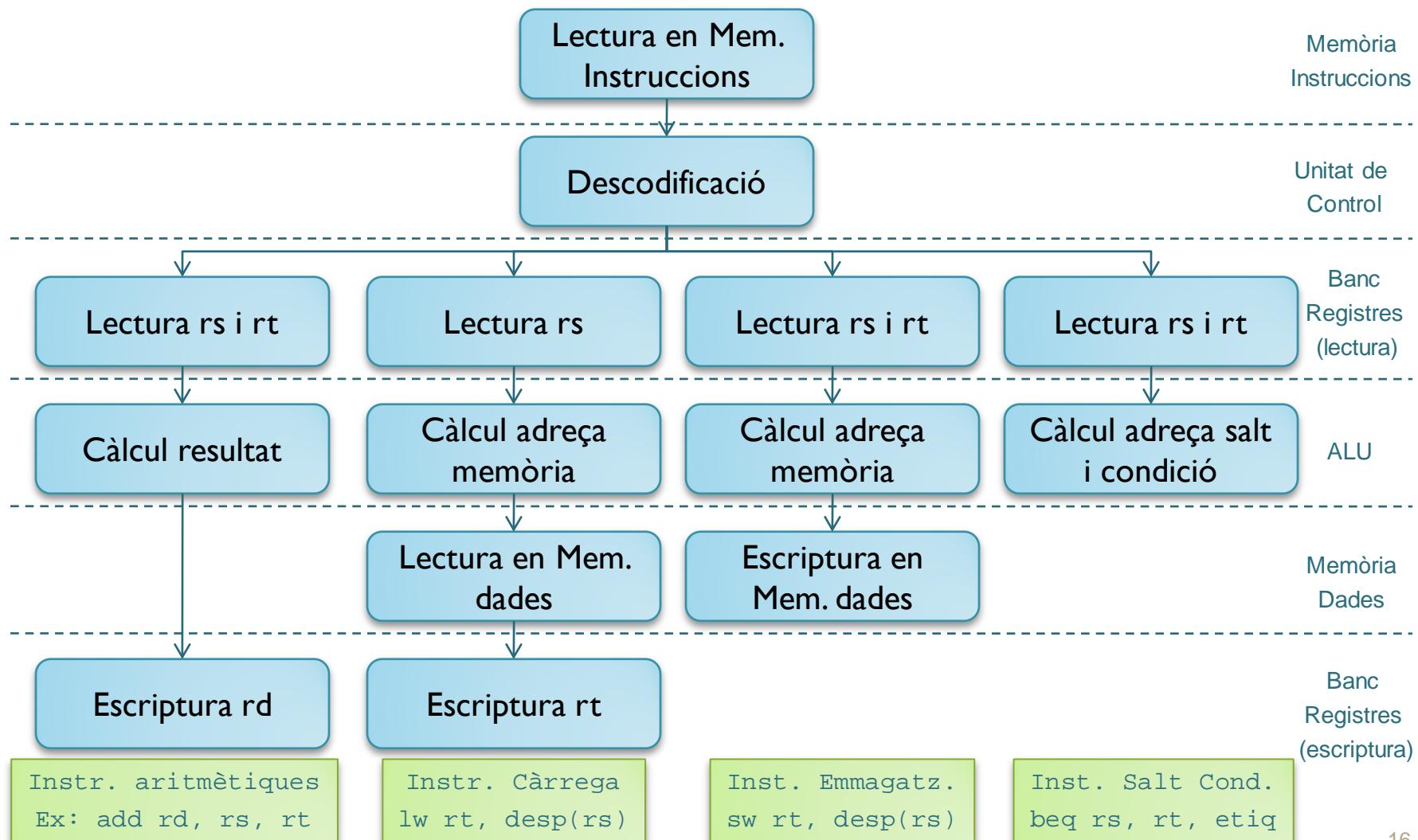
# Prestacions de la Segmentació

- Consideracions temporals

- ✓ El període mínim de rellotge és  $\tau = \max(\tau_i) + T_R$
- ✓ Per a incrementar la productivitat ( $1/\tau$ ) reduirem:
  - el retard màxim d'etapa (tot augmentant  $k$  i mantenint els retards equilibrats)
  - el retard dels registres

# Segmentació de la Ruta de Dades

- Especificació del cicle d'instrucció
  - ✓ Distribució en etapes dels components de la ruta



# Segmentació de la Ruta de Dades

- Definició de les etapes
  - ✓ Etapes comunes a totes les instruccions
    - **LI:** Etapa de lectura d'instrucció (i increment del PC)
    - **DI:** Etapa de decodificació d'instrucció (i lectura de registres)
  - ✓ Etapes que depenen del tipus d'instrucció
    - **EX:** Etapa d'execució
      - Instruccions de càlcul: càlcul del resultat
      - Load i Store: càlcul de l'adreça de memòria
      - Instruccions de salt: càlcul de l'adreça de salt i de la condició de salt
    - **M:** Etapa de memoria
      - Load i Store: accés a la memoria de dades
      - Instruccions de càlcul i salt: res
    - **ER:** Etapa d'escriptura de registre
      - Instruccions de càlcul i Load : escriptura del registre
      - Store i instruccions de salt: res

# Segmentació de la Ruta de Dades

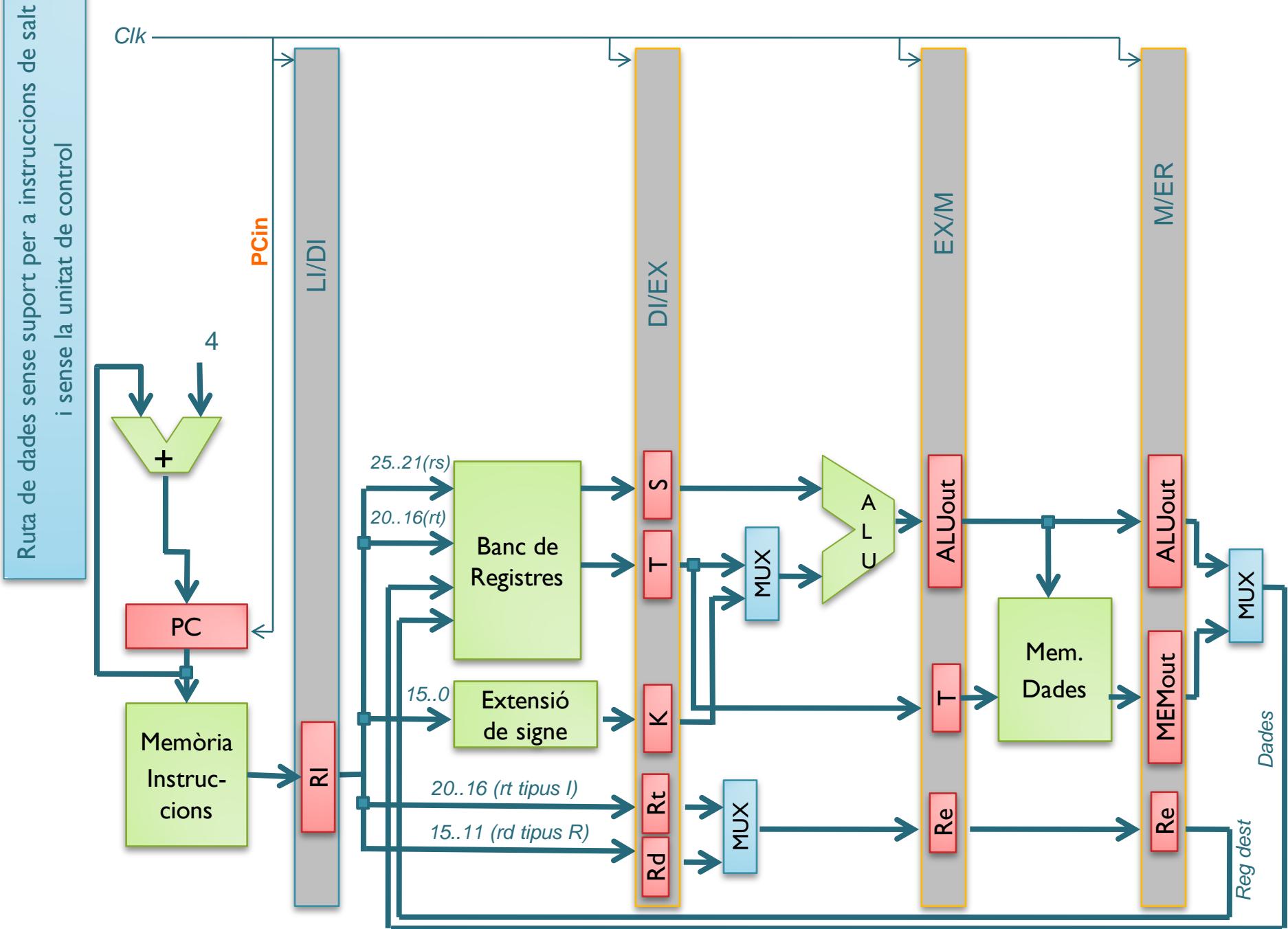
- Els registres de segmentació

- ✓ En calen quatre

- Cadascun està estructurat en subregistres

- ✓ Nomenclatura

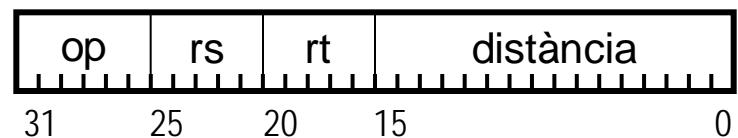
- Nom de cada registre d'etapa: el de les etapes que separa
      - Ex.: registre *LI/DI*
    - Per a referir-nos als subregistres: *reg.subreg*
      - Ex.: *LI/DI.RI* = registre d'instrucció de la instrucció que està en l'etapa DI
    - Per a referir-nos a un rang de bits d'un subregister: subíndexs
      - Ej.: *LI/DI.RI<sub>31..26</sub>* = codi d'operació de la instrucció continguda en DI o bé: *LI/DI.RI[Codop]*



# Suport Instruccions de Salt

- Instruccions de salt condicional en el MIPS
  - ✓ En el MIPS R2000 n'hi ha sis, totes del format I
  - ✓ Adreçament relatiu al PC
    - Com a distància de salt es codifica (en complement a 2) el nombre de paraules entre la instrucció següent i la instrucció de salt

instrucció	condició de salt
<i>beq rs,rt,etiq</i>	<i>rs=rt</i>
<i>bne rs,rt,etiq</i>	<i>rs&lt;&gt;rt</i>
<i>bgez rs,etiq</i>	<i>rs&gt;=0</i>
<i>bgtz rs,etiq</i>	<i>rs&gt;0</i>
<i>blez rs,etiq</i>	<i>rs&lt;=0</i>
<i>bltz rs,etiq</i>	<i>rs&lt;0</i>



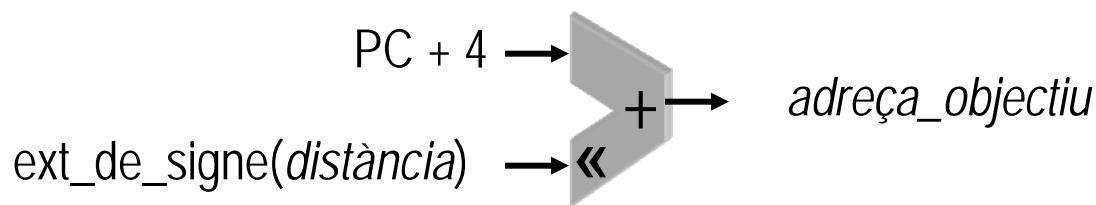
```
.text 0x00400000
beq $t0, $0, etiq
sll $t3, $t2, 3
xor $t4, $0, $0
ori $t4, $t3, $t1
etiq: add $t0, $0, $0
.end
```

Codifica un  
+3

$$\text{distància} = \frac{\text{adreça objectiu} - \text{adreça següent}}{4}$$

# Càlcul Adreça de Salt

- S'ha de calcular l'adreça absoluta de salt
  - ✓ adreça\_objectiu = (PC+ 4) + ext\_de\_signe(distància)\*4
  - ✓ El càlcul es pot fer en EX amb un sumador específic addicional
  - ✓ Necessitat de transmetre el valor PC + 4 des de LI fins EX
- Si la condició s'acompleix, l'etapa M escriu adreça\_objectiu en el PC



# Càlcul de la Condició de Salt

- Avaluació de la condició

- ✓ L'ALU ha de tenir l'operació identitat:  $\text{ALUout} = A$
- ✓ Ha de subministrar dos indicadors:
  - Z (resultat igual a zero)
  - S (bit de signe del resultat)

instrucció	condició	op.ALU	COND
beq	$a=b$	resta	Z
bne	$a \neq b$	resta	$\bar{Z}$
bgez	$a \geq 0$	identitat	$S$
bgtz	$a > 0$	identitat	$\bar{S} \cdot \bar{Z} = \overline{S + Z}$
blez	$a \leq 0$	identitat	$S + Z$
bltz	$a < 0$	identitat	S

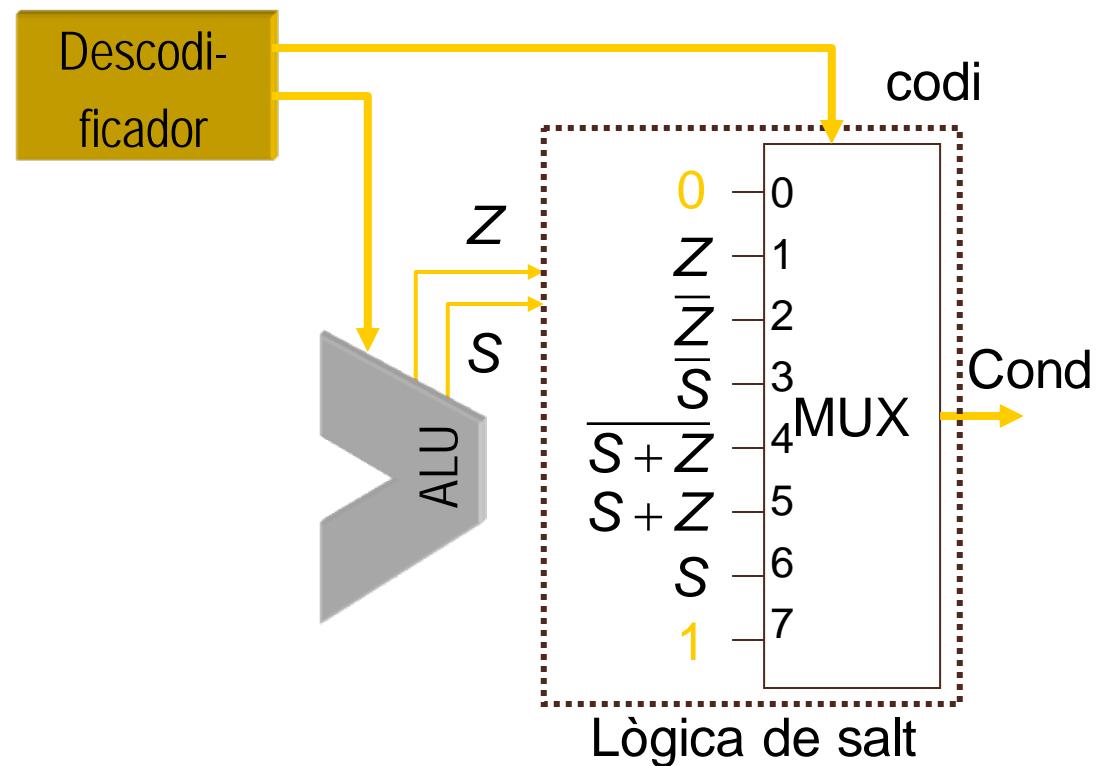
# Càlcul de la Condició de Salt

- Control bàsic de la bifurcació

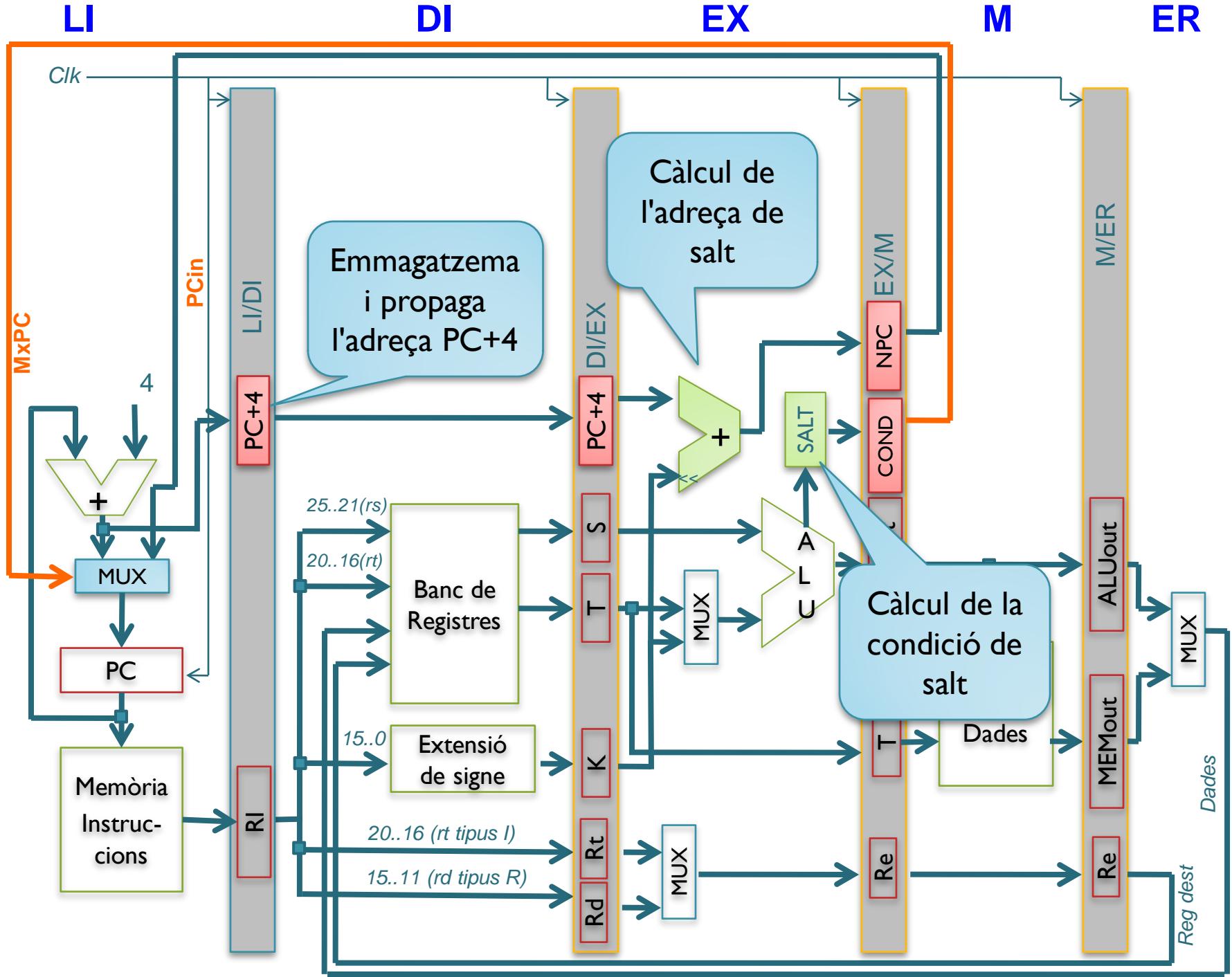
- ✓ Un nou registre EX/M.Cond (un bit): indicarà si hi ha bifurcació efectiva
- ✓ Implementació amb un MUX

- El descodificador d'instrucció, en l'etapa DI, calcula la posició del MUX per a cada instrucció

instrucció	codi
càcul, l/s, etc.	0
beq	1
bne	2
bgez	3
bgtz	4
blez	5
bltz	6
j	7



## Suport per a les instruccions de salt



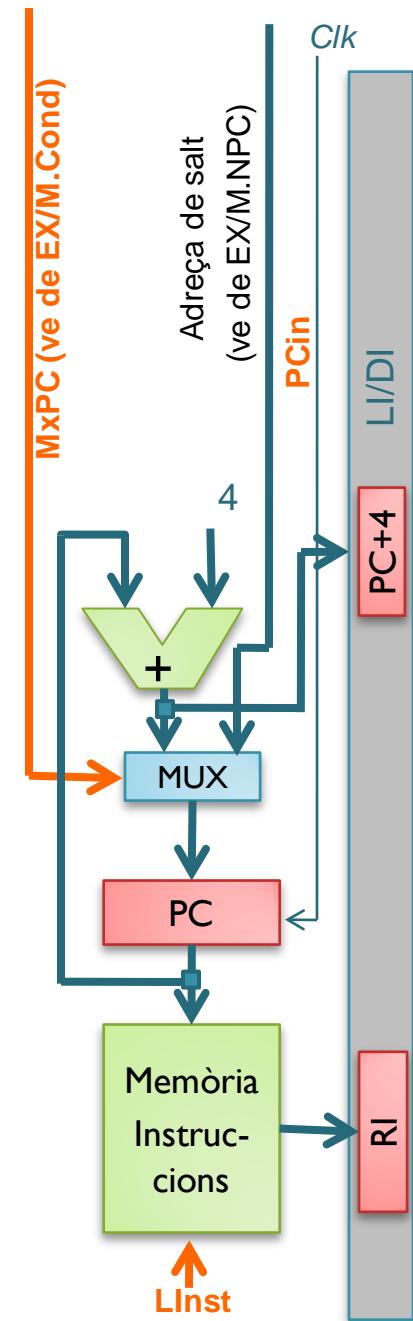
# Etapa de Lectura d'Instrucció

- Funcions

- ✓ Amb totes les instruccions:
  - Llegir la instrucció apuntada per el PC
  - Avanzar el PC per a que apunte a la instrucció següent
- ✓ Instrucció de salt condicional
  - Seleccionar la següent adreça d'instrucció, entre PC i l'adreça de salt (calculada en l'etapa EX)

- Senyals de control

- ✓ **PCin**: Carregar PC, activada per flanc del rellotge
- ✓ **LInst**: Llegir en la memòria d'instruccions
  - PCIn i LInst són constants (podem prescindir d'elles)
- ✓ **MxPC**: Calculada en l'etapa EX (reg EX/M.Cond)



# Etapa de Descodificació

- Funcions

- ✓ Amb totes les instruccions:

- Calcular els senyals de control per a les etapes posteriors
- Llegir els registres font
- Processar el camp de desp/imm

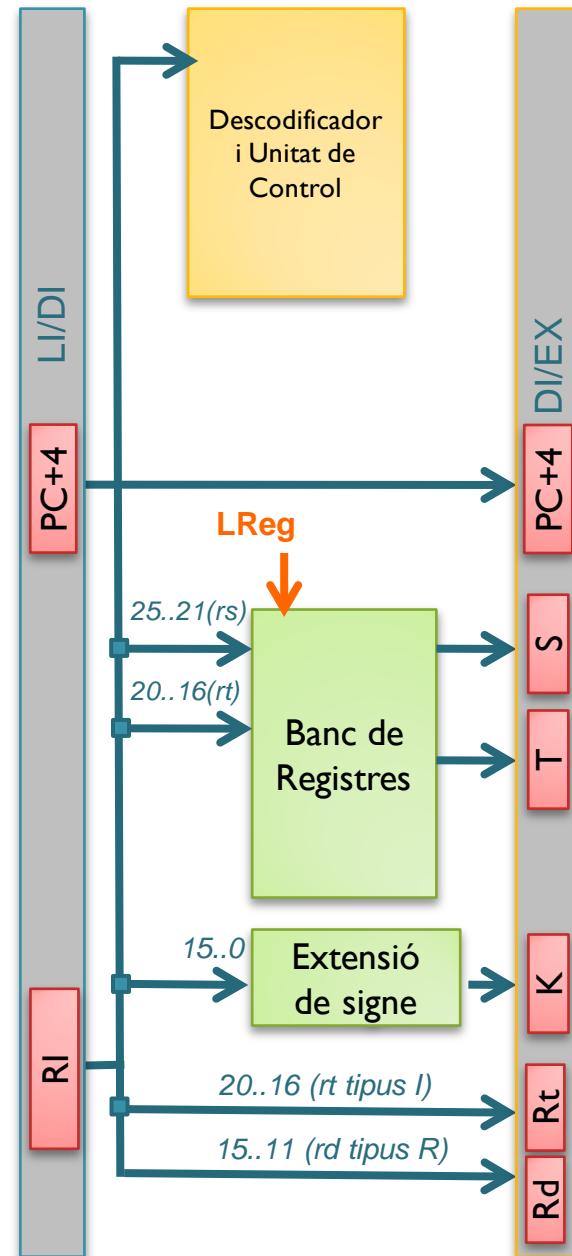
- Components

- ✓ Extensió de signe (combinacional)
- ✓ Banc de registres (ports de lectura)

- $S := \text{Reg}[Rl_{25..21}]$
- $T := \text{Reg}[Rl_{20..16}]$

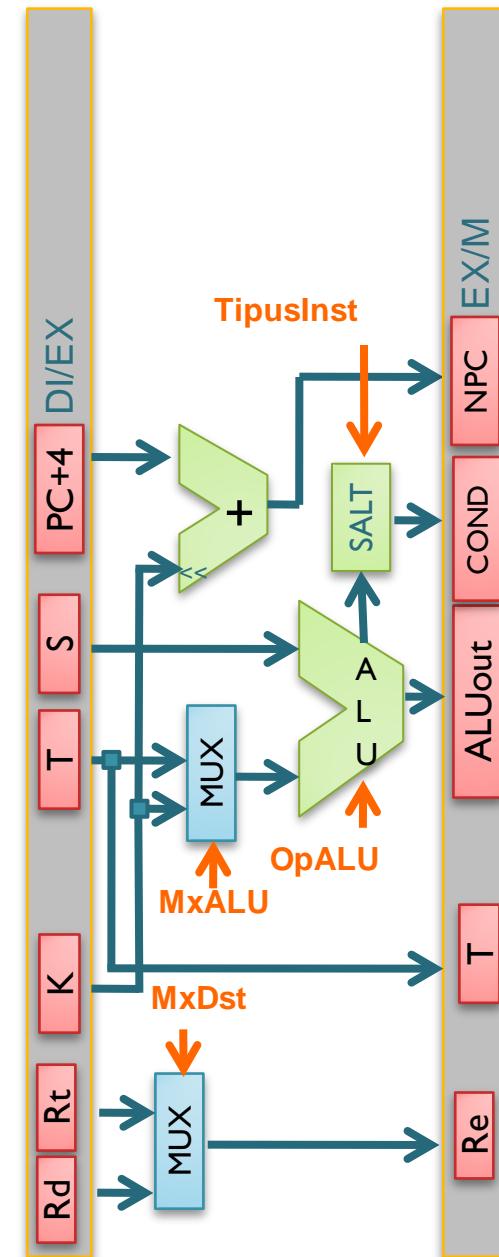
- Senyals de control

- ✓ **LReg**: Activa la lectura dels dos registres (podem prescindir d'aquest senyal)



# Etapa de Execució

- Funcions
  - ✓ Instr. de càlcul: obtenció del resultat
  - ✓ Instr. load/store: càlcul de l'adreça de memòria
  - ✓ Instr. salt: calcula l'adreça de salt i avalia la condició de salt
- Senyals de control
  - ✓ **MxALU**: Selecciona el segon operand de l'ALU
  - ✓ **MxDst**: Determina registre destinació (escriptura)
  - ✓ **OpALU**: Determina l'operació que farà l'ALU
  - ✓ **TipusInst**: Tipus d'instrucció



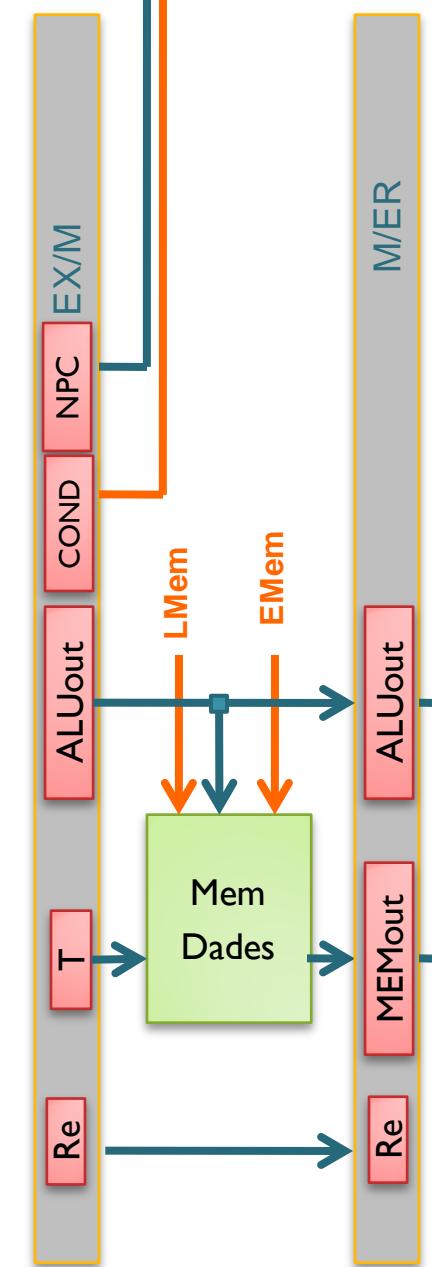
# Etapa d'Accés a Memòria

- Funcions

- ✓ Instruccions de càlcul: res
- ✓ Instruccions load/store: accés a memòria
- ✓ Instruccions de salt: selecciona següent adreça d'instrucció a llegir

- Senyals de control

- ✓ **LMem**: activa la lectura de la memòria
- ✓ **EMem**: activa l'escriptura de la memòria



# Etapa d'Escriptura en Registres

- Funcions

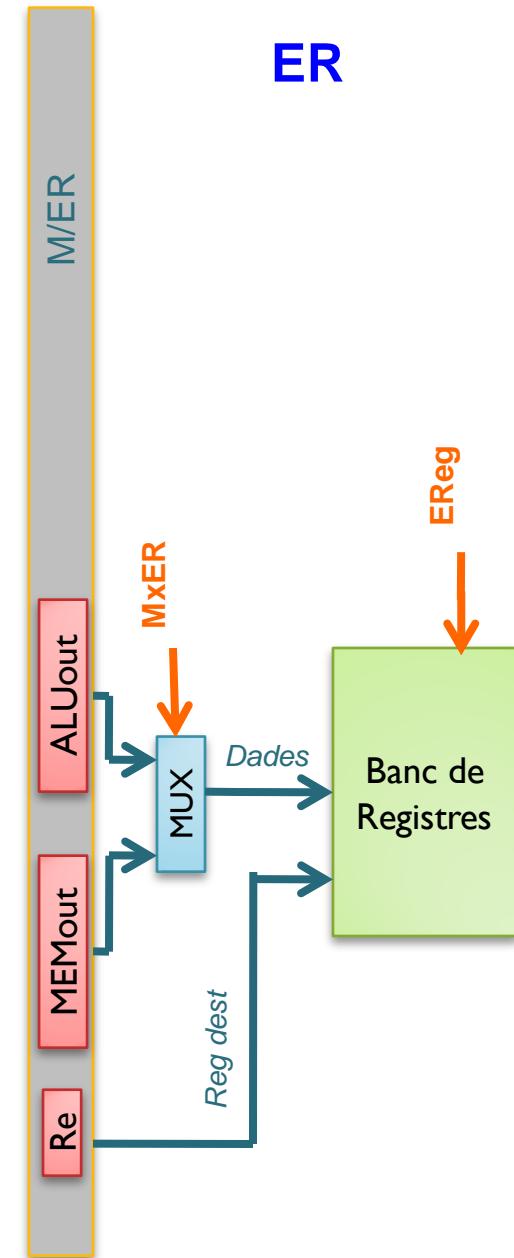
- ✓ Instruccions de càcul: escriptura de resultats
- ✓ Instruccions de load: escriptura del valor llegit de la memòria
- ✓ Instruccions de store i salt: res

- Característiques del banc de registres

- ✓ Ha de suportar dos accessos de lectura i un accés d'escriptura en cada cicle

- Senyals de control

- ✓ **MxER**: Selecciona el valor que s'hi escriu
- ✓ **EReg**: Activa l'escriptura en el banc de registres



# Segmentació de la Ruta de Dades

- La freqüència de rellotge
  - ✓ Temps de cicle de rellotge
    - $t_{seg} = \max(\tau_i) + t_R$
    - Caldrà considerar el temps de retard  $\tau_i$  de cada etapa i fixar-se en el més llarg
    - Caldrà considerar el retard  $t_R$  dels registres de segmentació
  - ✓ Exemple:
    - Si  $\tau_{LI} = \tau_M = 30$  ns,  $\tau_{DI} = \tau_{ER} = 20$  ns i  $\tau_{EX} = 25$  ns;  $t_R = 10$  ns
      - Període  $t_{seg} = 30 + 10 = 40$  ns
      - Freqüència de rellotge  $f = 1/40$  ns = 25 MHz

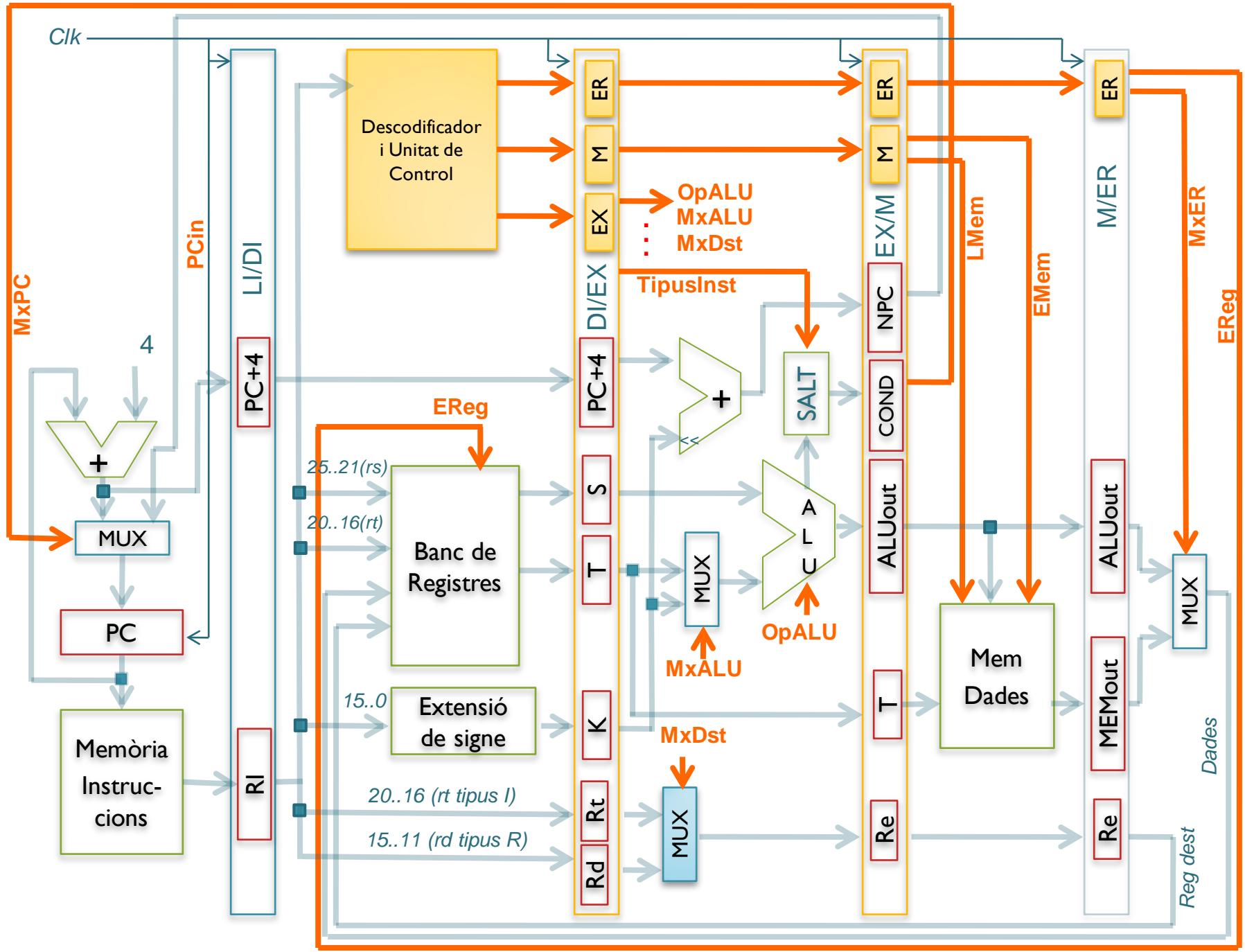
Comparació ruta no segmentada amb temps d'accés a memòria: 30 ns, a registres: 20 ns, ALU 25ns

Instrucció més lenta: lectura en memòria (load) amb 125 ns:  $F = 8$  MHz;  $S = 3.125$

(30 ns lect. instr. + 20 ns lect. registres + 25 ns ALU càlcul dir. + 30 ns lect. mem + 20 ns escript. en registre)

# Control del Procesador Segmentat

- Disseny bàsic del control
  - ✓ Objectiu general: Aconseguir que les cinc etapes funcionen ordenadament i puguen executar les instruccions escollides
  - ✓ Observacions
    - Cada etapa ha de ser autònoma
    - Les dues primeres etapes, LI i DI, processen instruccions no descodificades
      - Els seus senyals de control seran els mateixos durant tots els cicles
    - L'etapa DI s'encarregarà de calcular els senyals de control de les etapes posteriors i els transferirà a les següents etapes
      - Les etapes restants han de processar les instruccions en funció del codi d'operació
    - Els senyals de control coincideixen amb els de la ruta no segmentada. De fet, els senyals de control depenen exclusivament de la instrucció executada i no de la ruta de dades



# Representació Gràfica de la Segmentació

Diagrama  
instruccions/temps

Instrucció	1	2	3	4	5	6	7	8	9
lw \$1,a	LI	DI	EX	M	ER				
add \$3,\$2,\$0		LI	DI	EX	M	ER			
sub \$4,\$2,\$0			LI	DI	EX	M	ER		
and \$5,\$2,\$0				LI	DI	EX	M	ER	
or \$6,\$2,\$0					LI	DI	EX	M	ER

Cicle	LI	DI	EX	M	ER
1	lw	?	?	?	?
2	add	lw	?	?	?
3	sub	add	lw	?	?
4	and	sub	add	lw	?
5	or	and	sub	add	lw
6	?	or	and	sub	add
7	?	?	or	and	sub
8	?	?	?	or	and

Situació  
en el cicle 5

Diagrama  
temps/etapes

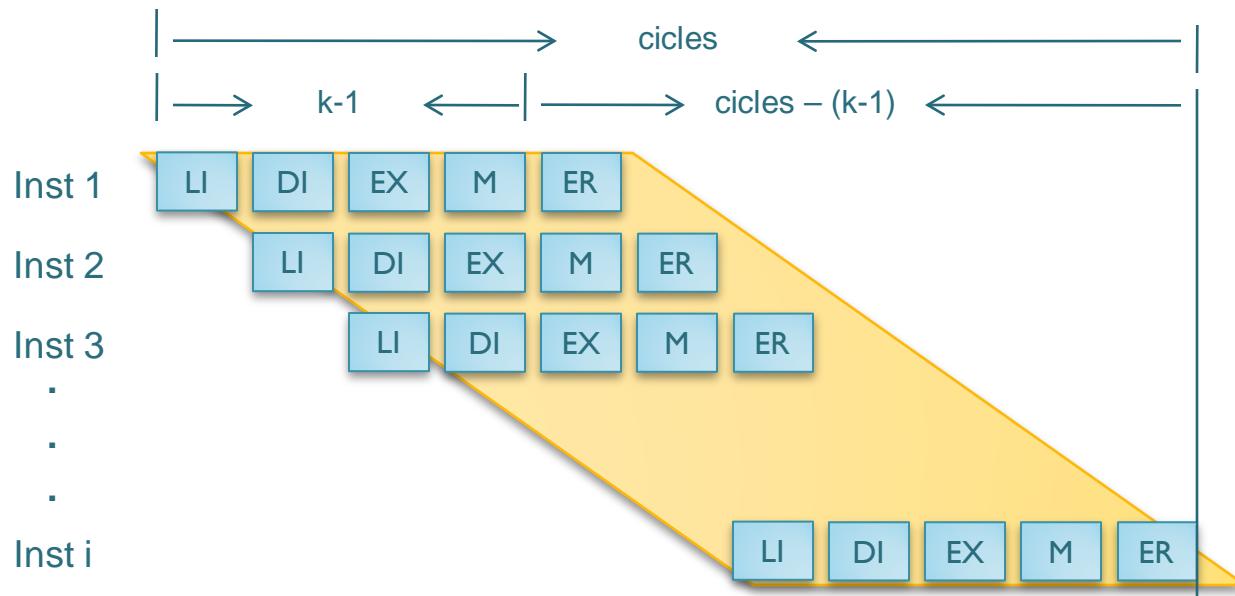
# Prestacions del Procesador Segmentat

- Generalitats
  - ✓ Equació del temps d'execució d'un programa en un processador en **règim estacionari** (no té en compte el temps d'omplir les etapes) :
$$T = I \times CPI \times t_C$$
  - ✓ T: temps d'execució total del programa
    - I: nombre d'instruccions que s'hi executen
    - CPI: nombre mitjà de cicles per instrucció
    - $t_C$ : temps de cicle del rellotge del processador
  - ✓ En general, convindrà minimitzar els tres factors I, CPI i  $t_C$
  - ✓ Ens preocuparem particularment dels factors I i CPI
  - ✓ Particularitzarem aquests conceptes al cas del processador segmentat

# Prestacions del Procesador Segmentat

## • CPI

- ✓ Índex típic que s'utilitza per a quantificar les prestacions del processador
- ✓ Representa el nombre mitjà de cicles per instrucció
- ✓ CPI>1 (el pipeline necessita k-1 cicles per a arribar a l'última etapa; cicles > I)



- ✓ Condicions ideals:
  - Infinites instruccions i zero cicles de parada
  - CPI ideal = 1 (cota inferior del CPI)

$$CPI = \frac{\text{cycles} - 4}{I}$$

# Introducció als Conflictes

## • Conflictes o riscos

- ✓ Són situacions produïdes per la segmentació del processador en què l'execució d'una o més instruccions no pot avançar
- ✓ Si el processador no estiguera segmentat, desapareixerien

### ✓ Tipus

- Estructurals (no els veiem perquè la nostra ruta no en té)
- De dades
- De control

### ✓ Solucions

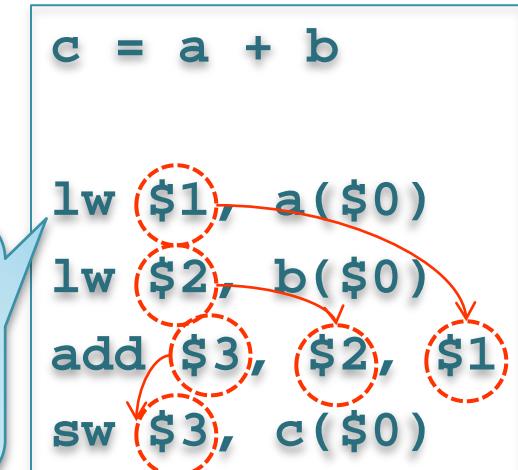
- Cicles de parada
- Modificació del software (instruccions NOP)

Els conflictes estructurals es produeixen en l'ús dels recursos per part de les instruccions l'execució de les quals se solapa. Per exemple, disposar d'una única memòria de codi i de dades.

# Els Conflictes de Dades

- Dependències de dades
  - ✓ Les instruccions estan fortament relacionades entre elles, resultat de la compilació de sentències escrites en un llenguatge de programació d'alt nivell
- Què són les dependències de dades?
  - ✓ Les instruccions es poden veure com a productores i consumidores de dades
  - ✓ Dependència de dades:
    - Instrucció que consumeix una dada produïda per una altra
  - ✓ Exemple:
    - Compilació de  $c = a + b$

Hi ha dependència entre la instrucció 1 i la 3 pel registre \$1



# Els Conflictes de Dades

- Conflictes de dades dins d'un processador segmentat
  - ✓ Apareixen quan una instrucció ha d'operar amb un valor que encara no ha subministrat una instrucció anterior:

instrucció	1	2	3	4	5	6	7	8
lw \$1 , a(\$0)	LI	DI	EX	M	ER \$1			
lw \$2 , b(\$0)		LI	DI	EX	M	ER \$2		
add \$3 , \$2 , \$1			LI	DI \$1	EX	M	ER \$3	
sw \$3 , c(\$0)				LI	DI \$2	EX	M	ER

- ✓ El valor de \$1 s'escriu en el banc en el cicle 5; però ha de ser llegit per add en el cicle 4
- ✓ El valor de \$2 s'escriu en el banc en el cicle 6; però ha de ser llegit per add en el cicle 4
- ✓ El valor de \$3 s'escriu en el banc en el cicle 7; però ha de ser llegit per sw en el cicle 5

# Els Conflictes de Dades

- Tècniques de resolució de conflictes
  - ✓ Tècniques d'urgència
    - Solució per software: inserció d'instruccions NOP en el codi
    - Solució per hardware: generació de cicles de parada

# Els Conflictes de Dades

- Solució d'urgència per software: inserció d'instruccions NOP
  - ✓ En generar el codi màquina, el compilador pot inserir instruccions NOP
  - ✓ Exemple: compilació de  $c = a + b;$

instrucció	1	2	3	4	5	6	7	8	9	10	11	12
lw \$1, a(\$0)	LI	DI	EX	M	ER	\$1						
lw \$2, b(\$0)	LI	DI	EX	M	ER	\$2						
nop		LI	DI	EX	M	ER						
nop			LI	DI	EX	M	ER					
add \$3, \$2, \$1			LI	DI	EX	M	ER	\$3				
nop				LI	DI	EX	M	ER				
nop					LI	DI	EX	M	ER			
sw \$3, c(\$0)						DI	EX	M	ER			

$$CPI = \frac{12 - 4}{8} = 1$$

$$T_{\text{programa}} = 8 \text{ instruccions} * 1 \text{ CPI} * T_{\text{cicle}}$$

# Els Conflictes de Dades

- Solució d'urgència per hardware: cicles de parada
  - ✓ El control del processador pot preveure les dependències de dades i generar cicles de parada per a resoldre-les

instrucció	1	2	3	4	5	6	7	8	9	10	11	12
lw \$1 , a(\$0)	LI	DI	EX	M	ER \$1	ER \$2						
lw \$2 , b(\$0)	LI	DI	EX	M	ER \$2	ER \$3						
add \$3 , \$2 , \$1	LI	DI	DI	DI	DI	EX	M	ER \$3				
sw \$3 , c(\$0)					LI	DI	DI	EX	M	ER		

Diagram illustrating data dependency conflicts (data races) between instructions. Red circles labeled 'ER' (Early Read) indicate dependencies where a value is read before it is written. Red arrows labeled 'DI' (Dependency) show the flow of data from one instruction's write to another's read. A red bracket below the timeline indicates four cycles of pipeline stalls (paradas) due to these conflicts.

$$CPI = \frac{12 - 4}{4} = 2$$

4 cicles de parada

- ✓ Podem calcular el CPI a partir del nombre de cicles de parada (P)

$$CPI = \frac{I + P}{I} = 1 + \frac{P}{I} = 1 + \frac{4}{4} = 2$$

$$T_{\text{programa}} = 4 \text{ instruccions} * 2 \text{ CPI} * T_{\text{cicle}}$$

# Solucions i prestacions

- Inserció de NOP

$$CPI = \frac{12 - 4}{8} = 1$$

El temps d'execució en régime estacionari:

$$T_{\text{programa}} = 8 \text{ instruccions} * 1 \text{ CPI} * T_{\text{cicle}}$$

- Detenció (cicles d'espera)

$$CPI = \frac{12 - 4}{4} = 2$$

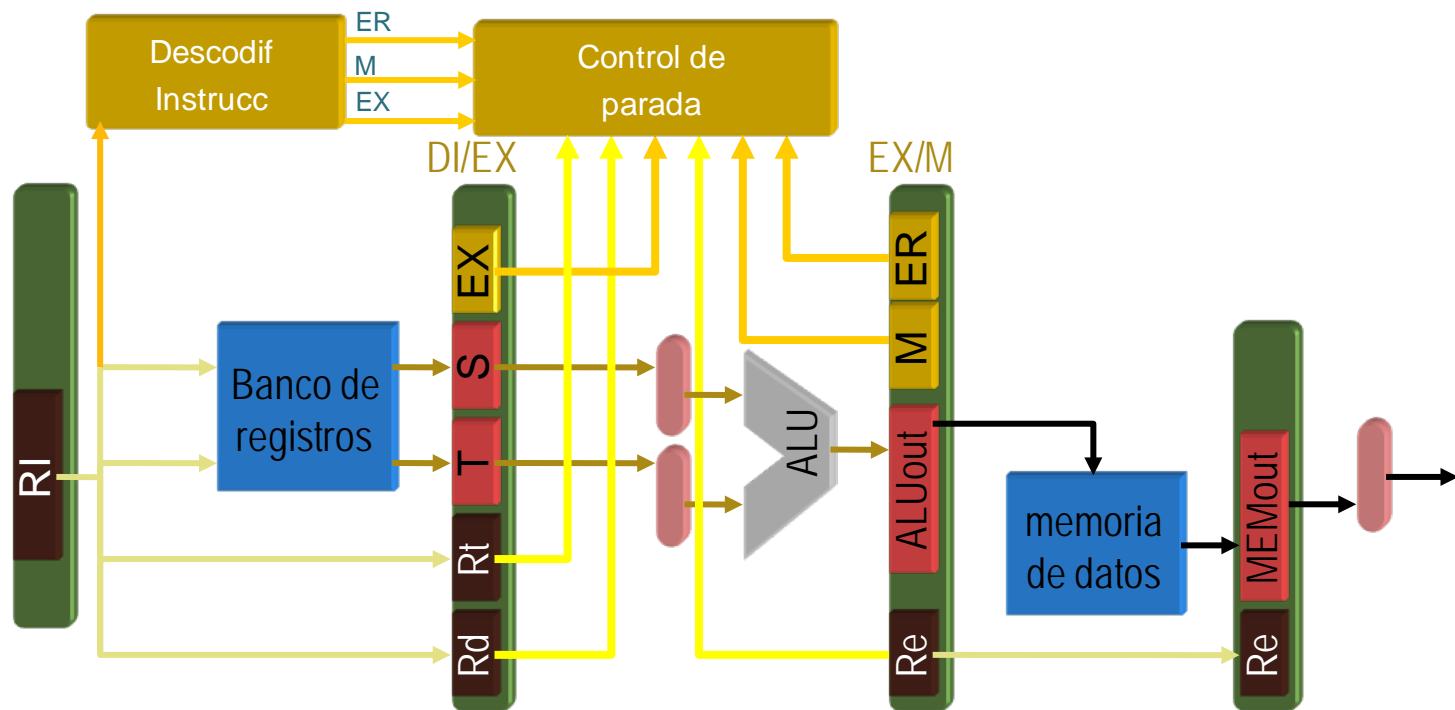
El temps d'execució en régime estacionari:

$$T_{\text{programa}} = 4 \text{ instruccions} * 2 \text{ CPI} * T_{\text{cicle}}$$

# Lògica de detecció de la dependència

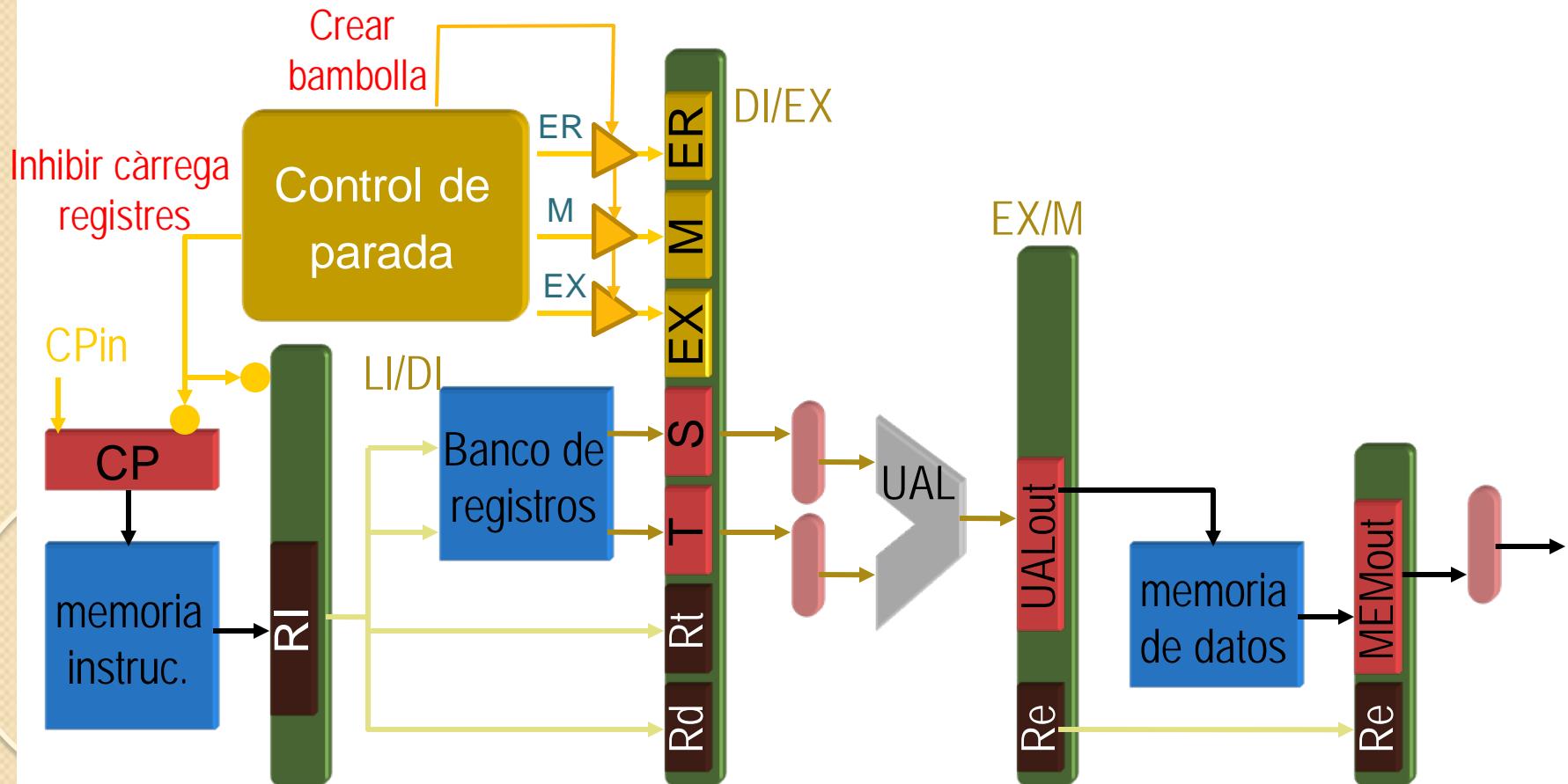
- Detecció:

- ✓  $\text{EX[LMem]}=0 \& \text{EX[EMem]}=0 \& \text{(la instrucció actual no es 'load' ni 'store')}$
- ✓  $(\text{DI/EX.Rd}=\text{RI.Rs} + (\text{DI/EX[MxALU2]}=1 \& \text{DI/EX.Rt}=\text{RI.Rs}))$
- ✓  $(\text{DI/EX.Rd}=\text{RI.Rt} + (\text{DI/EX[MxALU2]}=1 \& \text{DI/EX.Rt}=\text{RI.Rt}))$
- ✓  $(\text{EX/M.Re} = \text{RI.Rs} + (\text{EX[MxALU2]}=0 \& \text{EX/M.Re} = \text{RI.Rt}))$



# Lògica d'inserció del cicle de parada

- Les etapes LI i DI han de repetir instrucció
- Cal crear una bambolla en EX (LMem=0; EMem=0; EReg=0)



# Els Conflictes de Dades

- Valoració de les dues solucions vistes

- ✓ Semblances

- El nombre de cicles en temps d'execució és el mateix
      - En l'exemple: 12 cicles
    - El compilador pot ajudar a baixar el temps d'execució

- ✓ Diferències

- La inserció d'instruccions **nop** incrementa el factor I (nombre d'instruccions)
    - Els cicles de parada incrementen el factor CPI
    - La ruta de dades segmentada és **compatible binària** amb la no segmentada (és a dir, l'execució del mateix programa en ambdós processadors produiria els **mateixos resultats**)
    - La complexitat de la lògica d'inserció de cicles de parada podria allargar el retard de les etapes i caldria baixar la freqüència del rellotge

# Conflictes de Control

- Les instruccions de salt
  - Trenquen la seqüència lineal d'execució dels programes
  - Tipus
    - Salt incondicional (*jump*)
    - Salt condicional o bifurcació (*branch*)
    - Crida i retorn de subprograma (*call/jump&link i return*)
  - Modes d'adreçament
    - Absolut
    - Relatiu a CP
    - Indirecte
  - Freqüència d'aparició de les instruccions de salt: 10 al 20% del codi
    - Depén del tipus de processador, de les tècniques de compilació i del programa en concret

# Conflictes de Control

- Instruccions de salt condicional en el MIPS
  - ✓ Exemples

```
if (x>y)
/*then*/ z=x
else z=y;
```



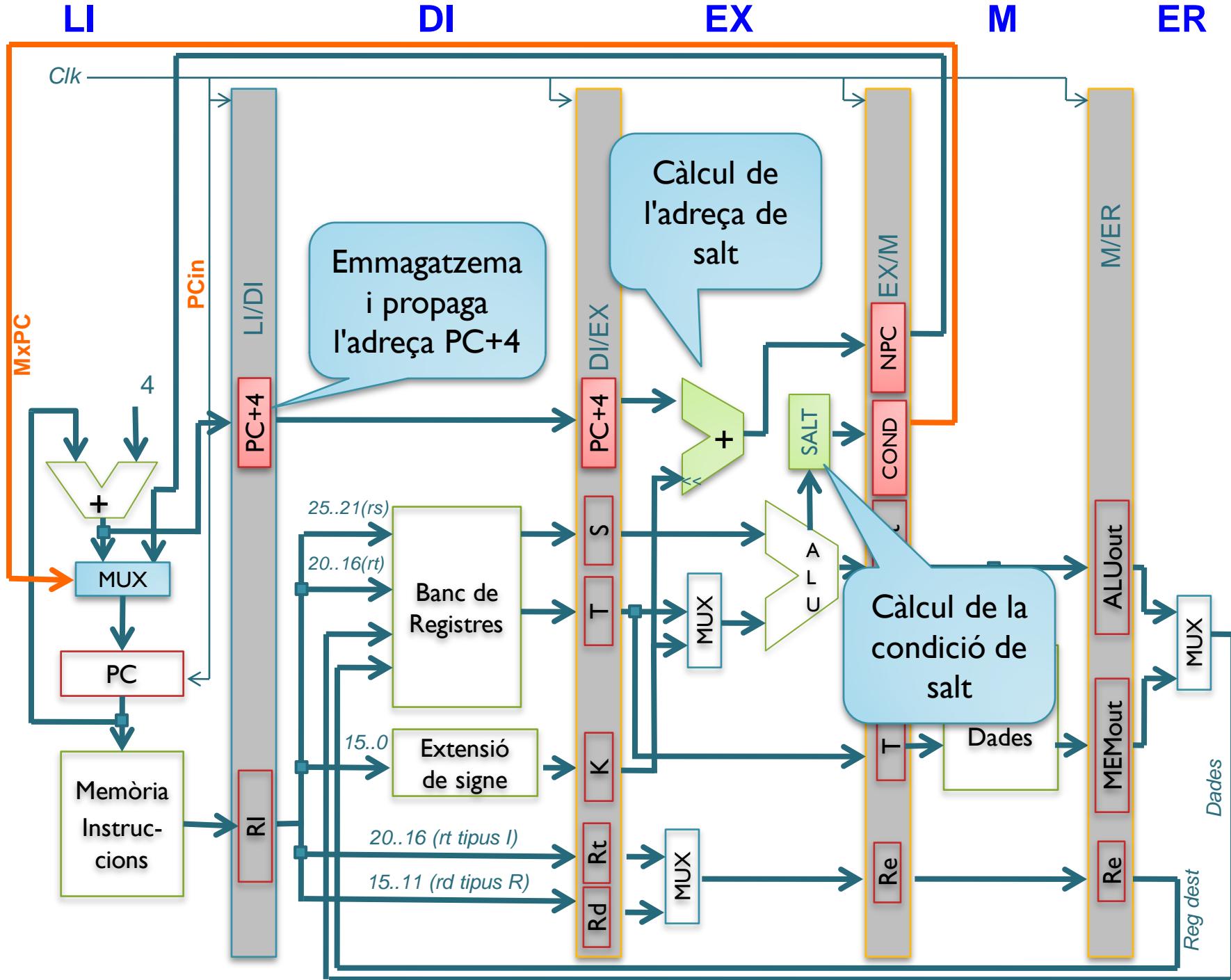
```
if:    lw $t0,x($0)
       lw $t1,y($0)
       sub $t2,$t1,$t0
       bgez $t2,else
then:   sw $t0,z($0)
        j endif
else:   sw $t1,z($0)
endif:
```

```
z=0;
do      z=z+y;
        x=x-1;
while  (x!=0)
```



```
add $t2,$zero,$zero
lw $t0,x($0)
lw $t1,y($0)
do:    add $t2,$t2,$t1
       addi $t0,$t0,-1
while: bne $t0,$zero,do
enddw: sw $t2,z($0)
```

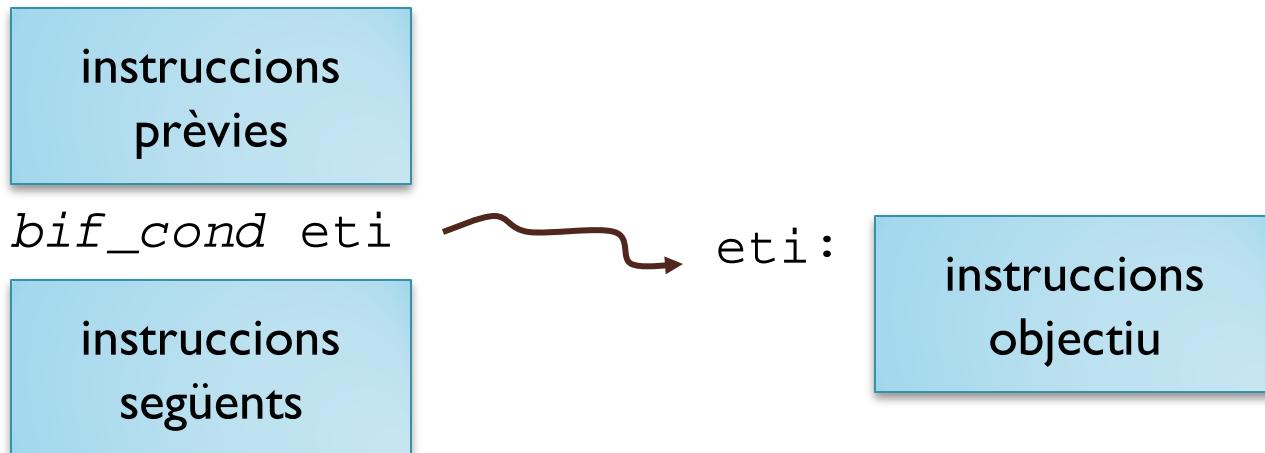
## Suport per a les instruccions de salt



# Conflictes de Control

## • Vocabulari

- ✓ Instrucció objectiu (*target*): instrucció destinatària del salt
- ✓ Les bifurcations salten si s'acompleix una condició
  - Si bifurquen, direm que el salt és efectiu (*taken*)
  - En cas contrari, direm que el salt és no efectiu (*not taken*)
- ✓ Una instrucció de salt condicional relaciona tres grups d'instruccions:



# Conflictes de Control

- Anàlisi del conflicte

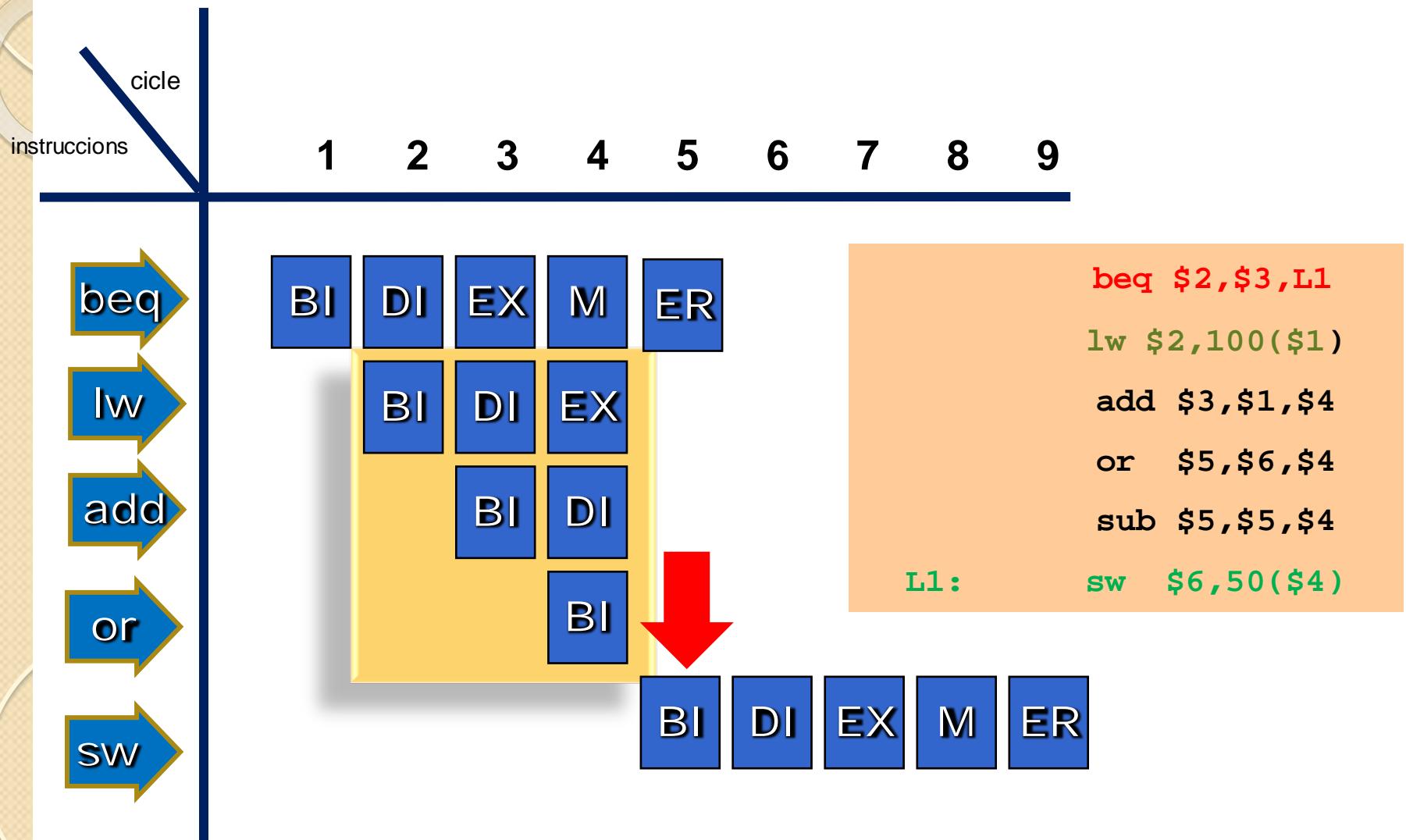
- ✓ Quan el salt és efectiu en l'etapa M el processador té tres instruccions següents en procés. En l'etapa ER s'escriu el nou PC

Instrucció	1	2	3	4	5	6	7	8	9
prèvia	LI	DI	EX	M	ER				
bifurcació		LI	DI	EX	M	ER			
següent 1			LI	DI	EX	M	ER		
següent 2				LI	DI	EX	M	ER	
següent 3					LI	DI	EX	M	ER
objectiu						LI	DI	EX	M

Veure ruta

cicle	LI	DI	EX	M	ER
4	següent2	següent1	bifurcació	prèvia	?
5	següent3	següent2	següent1	bifurcació	prèvia
6	objectiu	següent3	següent2	següent1	bifurcació

# Anàlisi del conflicte



# Conflictos de Control

## • Anàlisi del conflicte

- ✓ Latència de salt: nombre de cicles que necessita el processador per a executar una instrucció de salt

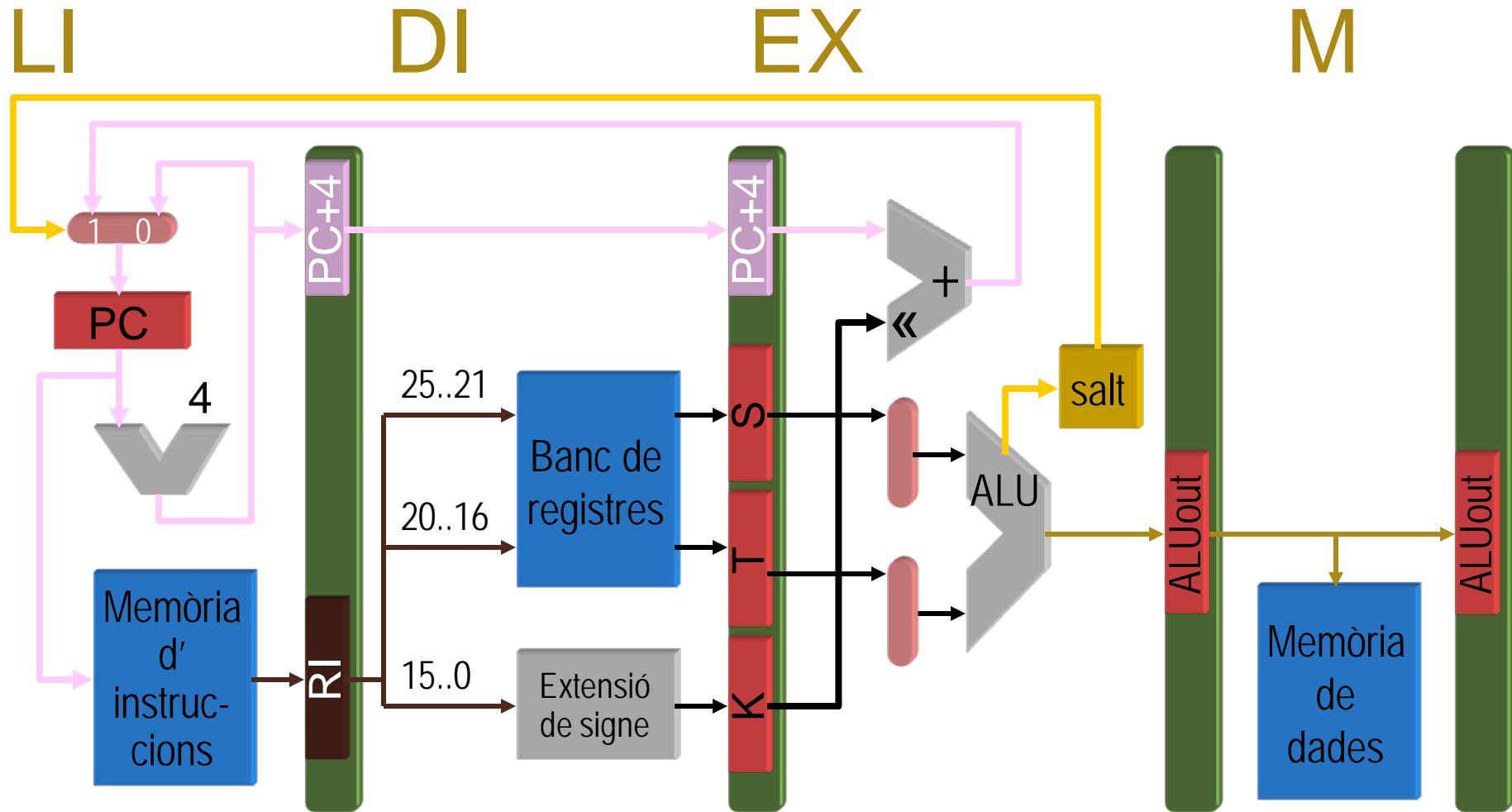
- En el cas d'un processador segmentat linealment, tenim:

latència de salt = nombre d'instruccions següents afectades  
= número d'etapa en què el salt és efectiu – 1

- En el cas anterior la **latència de salt = 3**
  - Com més primerenca siga l'etapa en què s'escriu el PC, menor penalització
  - Per a reduir el conflicte, es pot modificar el disseny de la ruta de dades i avançar el moment de l'escriptura del PC. Veure esquema de la següent transparència.

# Latència del Salt

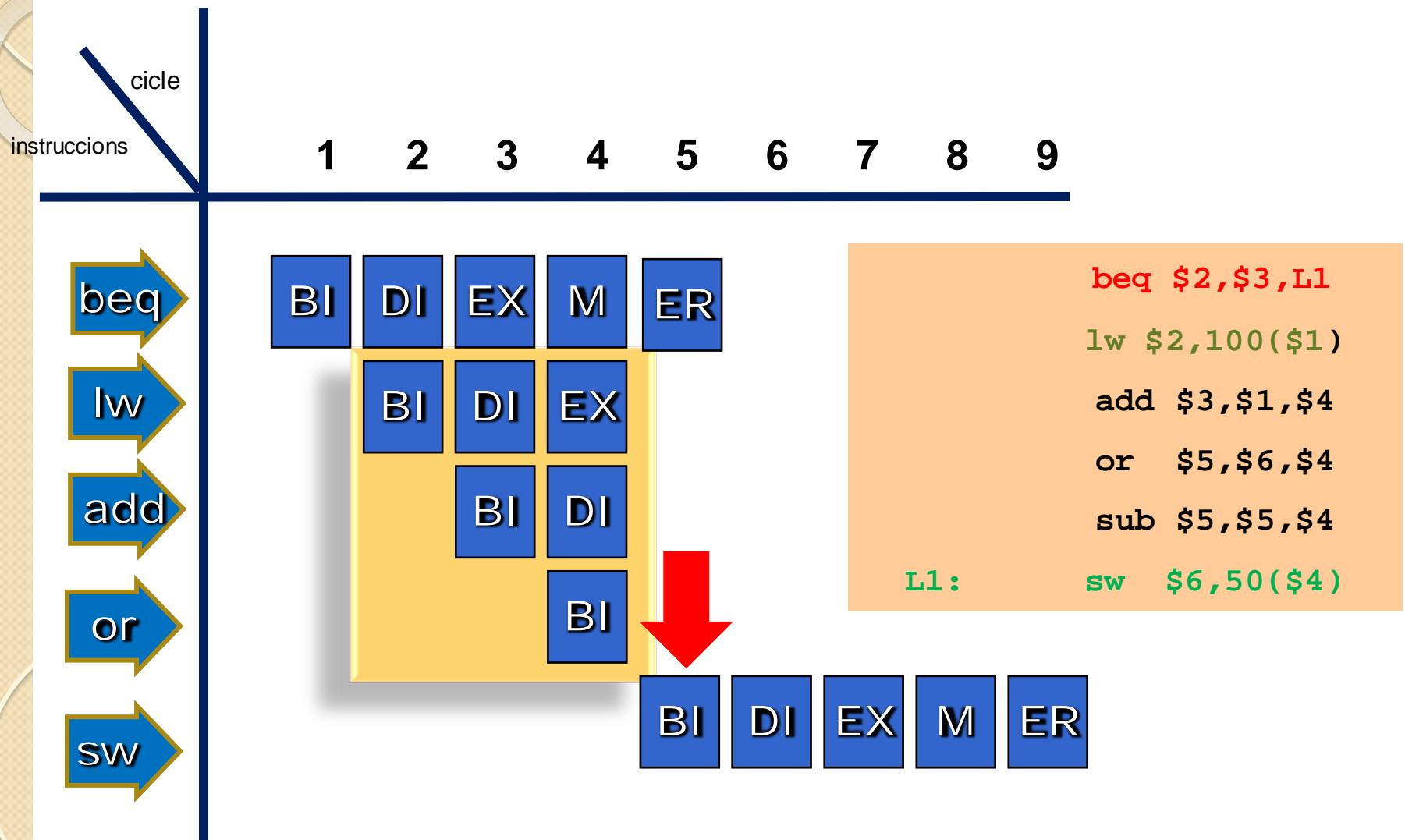
- Ruta de dades amb latència de salt = 2



# Tractament dels Conflictes de Control

- Primeres solucions al conflicte
  - ✓ Solucions d'urgència:
    - Hardware
      - El descodificador insereix cicles de parada en detectar un salt
    - Software: salt retardat
      - El compilador reordena el codi o insereix instruccions **nop**
  - ✓ Solucions avançades: predicció
    - Predicció fixa:
      - *predict-not taken (always)*: Intel i486
      - *predict taken*: Sun SuperSparc

# Anàlisi del conflicte



# Prestacions



Les solucions conservatives (inserció d'instruccions NOP o cicles d'espera) no permeten millorar el rendiment del processador.

Para no tenir cicles improductius el compilador pot trobar instruccions útils en lloc de NOP. Això suposa re-organitzar el codi.

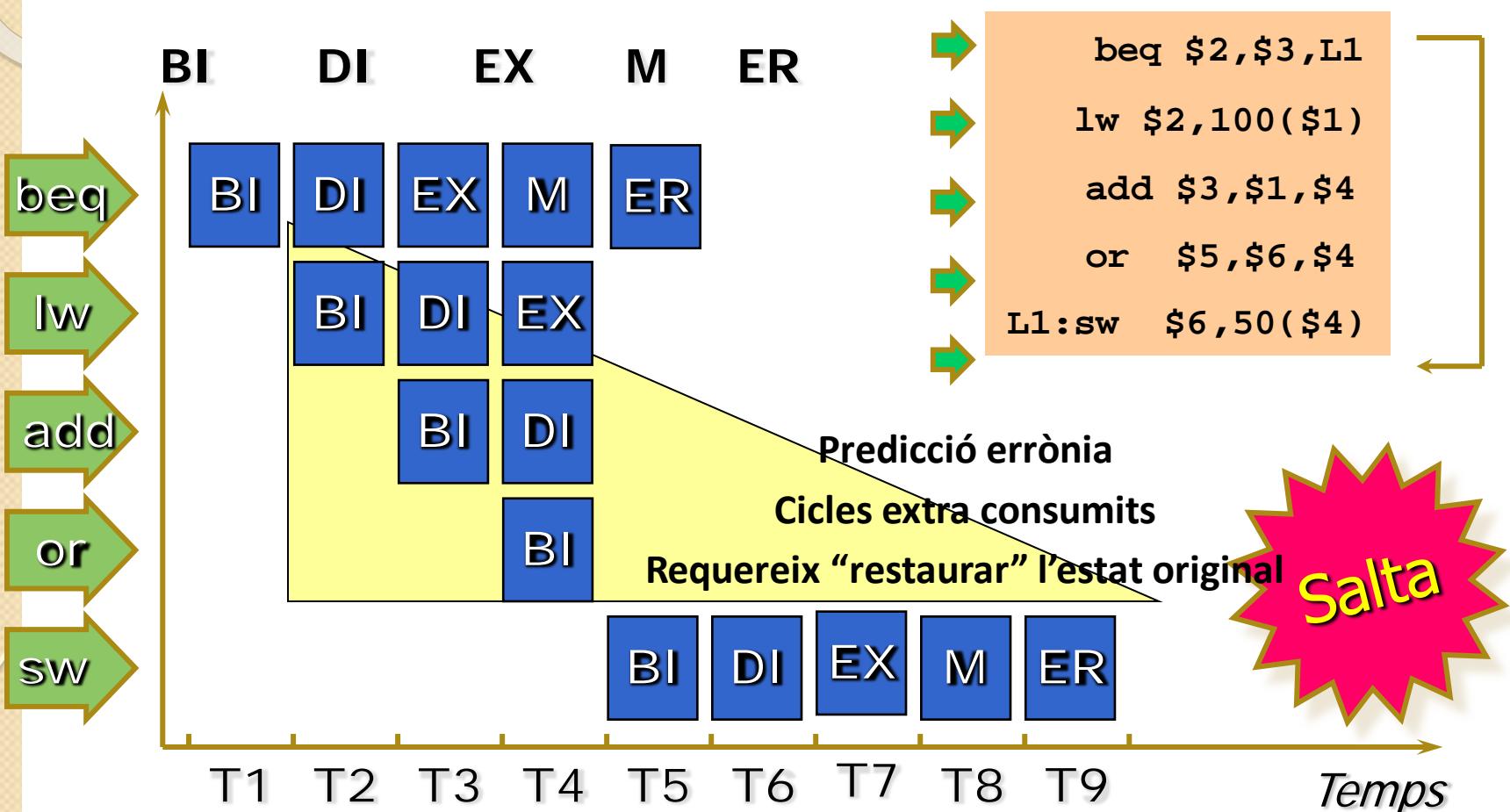
# Alternativa: Predicció de salt



- Predicció estàtica
  - Salt efectiu
  - Salto No efectiu
- Predicció dinàmica  
(té en compter el històric)

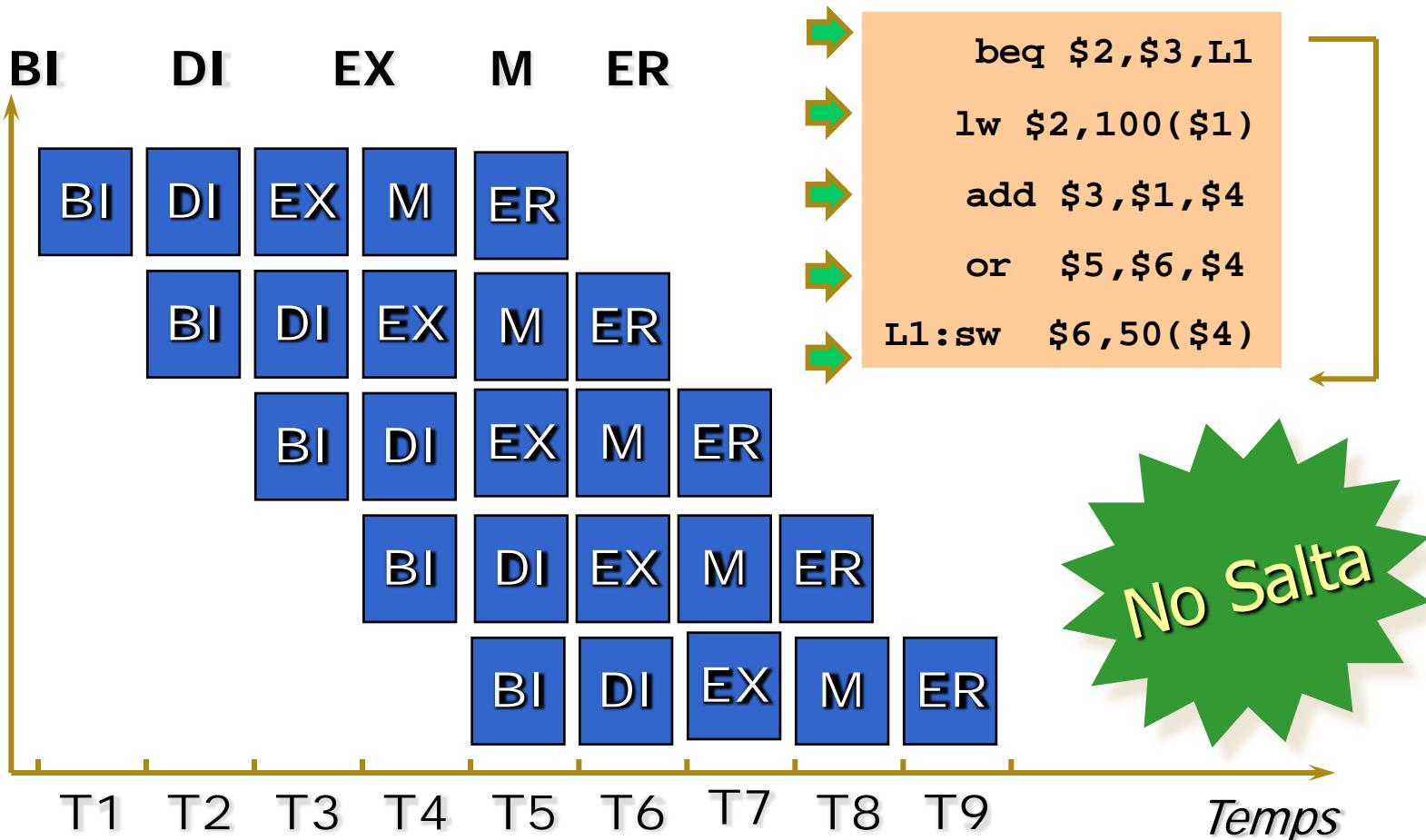
## Predictió de salt NO efectiu

### Predictió de salt no efectiu



## Predicción de salto NO efectiu

Predicción de  
salto no efectiu



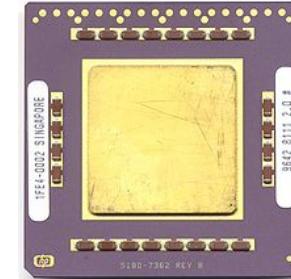
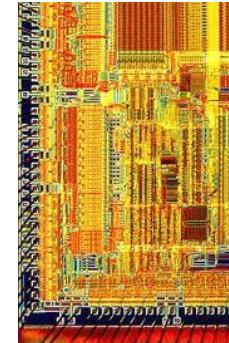
# Conclusions



- Les instruccions de salt (o de control de flux) suposen el trencament de la seqüència d'execució d'un programa i per tant una pèrdua de rendiment de la segmentació
- Les solucions conservatives es limiten a resoldre el problema per a garantir la correcta seqüenciació del programa
- Per a millorar el rendiment convé aprofitar les instruccions que ja han entrat en el processador.
- Les tècniques de predicció de salt van adreçades a això

# Exemples de Segmentació Bàsica

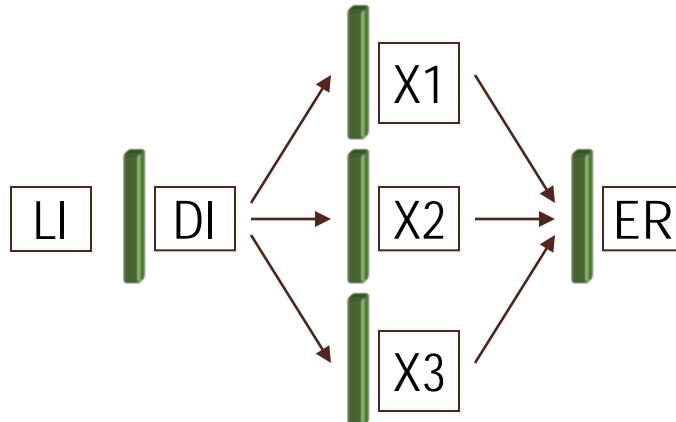
- La segmentació bàsica en els processadors comercials
  - ✓ El model de segmentació en cinc etapes és la base del disseny dels processadors moderns (excepte la família Intel x86)
  - ✓ Exemples:
    - MIPS R2000
    - Sun SPARC V7
    - Hewlett Packard – Precision Architecture (HP-PA)
  - ✓ Família Intel x86
    - 80486: també estava segmentat en 5 etapes, però amb diferent organització



# Altres Models de Segmentació

- Segmentació amb etapes multicicle
  - ✓ Permet l'execució d'operacions de durada variable (1 o més cicles)
    - aritmètica de coma flotant, multimèdia, etc.
  - ✓ Disposa d'una col·lecció d'operadors diversos, cadascun amb latència L i taxa de repetició R específics
    - accés a memòria (típicament  $L \geq 1$ ,  $R=1$ )
    - aritmètica entera ( $L=1$ ,  $R=1$ )
    - aritmètica de CF ( $L>1$ ,  $R>=1$ )

Exemple:



unitat	L	R
X1 (mem)	2	1
X2 (enters)	1	1
X3 (CF)	3	3

# Altres Models de Segmentació

- Segmentació amb etapes multicicle (cont.)
  - ✓ Cal una lògica d'emissió (*issue*) que distribuïsca el treball entre els diversos operadors
  - ✓ L'ordre d'acabament d'execució de les instruccions pot ser distint a l'ordre en què han sigut descodificades

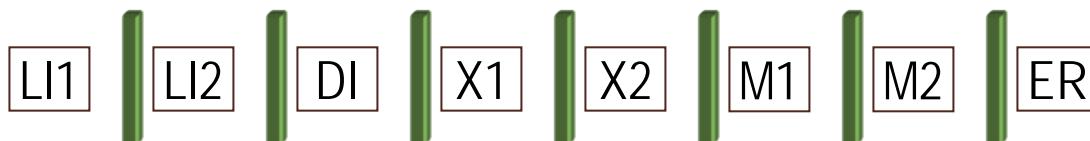
Instrucció	1	2	3	4	5	6	7
lw	LI	DI	X1	X1	ER		
add.s (CF)		LI	DI	X3	X3	X3	ER
add (enters)			LI	DI	X2	ER	

- ✓ La ruta de dades pot patir conflictes estructurals
  - Exemple: ¿què passa si van seguides dues instruccions de CF?

# Altres Models de Segmentació

## • Supersegmentació

- ✓ La tasca a realitzar en alguna/es etapes del model bàsic es reparteix entre diverses etapes
  - Augmenta el nombre d'etapes
  - Objectiu: incrementar la freqüència de rellotge (etapes més curtes)
  - CPI base = 1
- ✓ Exemple: MIPS 4000: 8 etapes LI1,LI2,DI,EX1,EX2,M1,M2,ER
  - LI1 i LI2 fan la lectura d'instrucció
  - M1 i M2 fan l'accés a memòria de dades



## ✓ Altres exemples

- SPARC-V8 (9 etapes)
- Alpha 21064 (7 etapes)

# Altres Models de Segmentació

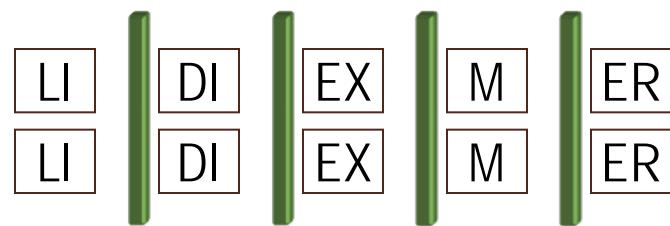
- Processadors superescalars

- ✓ Cada etapa pot processar  $n$  instruccions

- Objectiu: reduir CPI ideal a  $1/n$  tot mantenint la freqüència de rellotge
    - S'utilitza el IPC (Instruccions Per Cicle) =  $n$

- ✓ Exemple amb  $n=2$  (CPI ideal = 0.5, IPC ideal = 2)

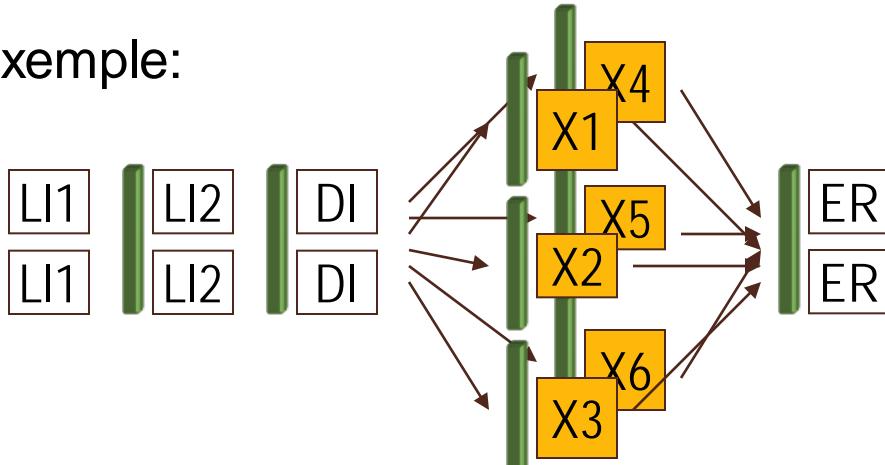
Instr.	1	2	3	4	5	6	7	
i1		LI	DI	EX	M	ER		
i2		LI	DI	EX	M	ER		
i3			LI	DI	EX	M	ER	
i4			LI	DI	EX	M	ER	
i5				LI	DI	EX	M	ER
i6				LI	DI	EX	M	ER



# Altres Models de Segmentació

- Combinacions
  - ✓ Els processadors actuals combinen diversos models de segmentació
  - ✓ Paràmetres significatius:
    - Nombre d'etapes
    - Ample d'escalaritat = nombre d'instruccions que es poden llegir o descodificar per cicle
    - Assortiment d'unitats d'execució i característiques (L, R) de cadascuna

Exemple:



Supersegmentat, superescalar de grau = 2

Unitat	L	R
X1 (l/s memòria)	3	1
X2 (s/r enters)	1	1
X3 (p/d enters)	4	3
X4 (s/r CF)	5	3
X5 (p/d CF)	15	15
X6 (multimèdia)	3	1

# RISC vs CISC: Dues Estratègies de Disseny

CISC: Complex Instruction Set Computer	RISC: Reduced Instruction Set Computer
Joc d'instruccions gran i complex	Joc d'instruccions reduït i senzill
Format d'instruccions variable (heterogeni) amb diferents longituds	Format d'instruccions fix (homogeni) amb la mateixa longitud
Molts modes d'adreçament i complexos	Pocs modes d'adreçament i senzills
Nombre reduït de registres	Nombre gran de registres
Moltes instruccions poden accedir a memòria	Arquitectura de càrrega/emmagatzemament
Unitat de control complexa (microprogramada)	Unitat de control senzilla (cablejada)
Objectiu: instruccions complexes, properes als llenguatges d'alt nivell	Objectiu: acabar d'executar cada instrucció en un únic cicle
Menor complexitat en el compilador	Major complexitat en el compilador
Major dificultat per a la segmentació	Major facilitat en la segmentació
Arquitectura típica dels anys 70 i 80	Arquitectura popular en els anys 90
Intel x86, Celeron, Pentium, AMD (Duron, Athlon)	PowerPC, DEC Alpha, MIPS, ARM, Sparc, HP
En l'actualitat no hi ha una clara diferència entre CISC-RISC. Els processadors no s'emmarquen dins d'una taxonomia tan estricta. Els processadores Intel es consideren CISC-in RISC-out	

# RISC vs CISC: Dues Estratègies de Disseny

- Estudis dels anys 80 demostraren:
  - ✓ Els processadors (CISC) dedicaven el 80% del temps en executar només un 20% de les instruccions disponibles en el joc d'instruccions
  - ✓ Algunes instruccions no s'empraven mai i d'altres poques voltes
- En reduir el joc d'instruccions i homogeneïtzar el format:
  - ✓ Les instruccions s'executen més ràpidament
  - ✓ La unitat de control és més simple, ràpida i ocupa menys espai
  - ✓ Hi ha més espai del xip per a incloure més registres i memòria cau
  - ✓ La segmentació pot ser més eficient
- CISC redueix l a costa d'incrementar el CPI
- RISC redueix el CPI tot augmentant l

$$\frac{\text{Temps}}{\text{programa}} = T_{\text{cicle}} * \frac{\text{cicles}}{\text{instrucció}} * \frac{\text{instruccions}}{\text{programa}}$$

# CISC vs RISC: Dues Estratègies de Disseny

- Multiplicació de dos nombres que es troben ubicats en memòria; el producte es deixa en l'adreça del primer operand:
  - ✓ **adr1**: adreça de memòria de l'operand 1
  - ✓ **adr2**: adreça de memòria de l'operand 2

CISC

**MULT adr1, adr2**



Poc treball per al compilador  
Baixa ocupació de la memòria  
Instruccions complexes  
Temps alt d'execució

RISC

```
li $t0, adr1  
li $t1, adr2  
lw $t3, 0($t0)  
lw $t4, 0($t1)  
mul $t3, $t4  
mflo $t5  
sw $t5, 0($t0)
```



El compilador ha de resoldre molts aspectes  
Més ocupació de la memòria de codi  
Més instruccions però més simples  
Menor temps d'execució per instrucció