Segmentació

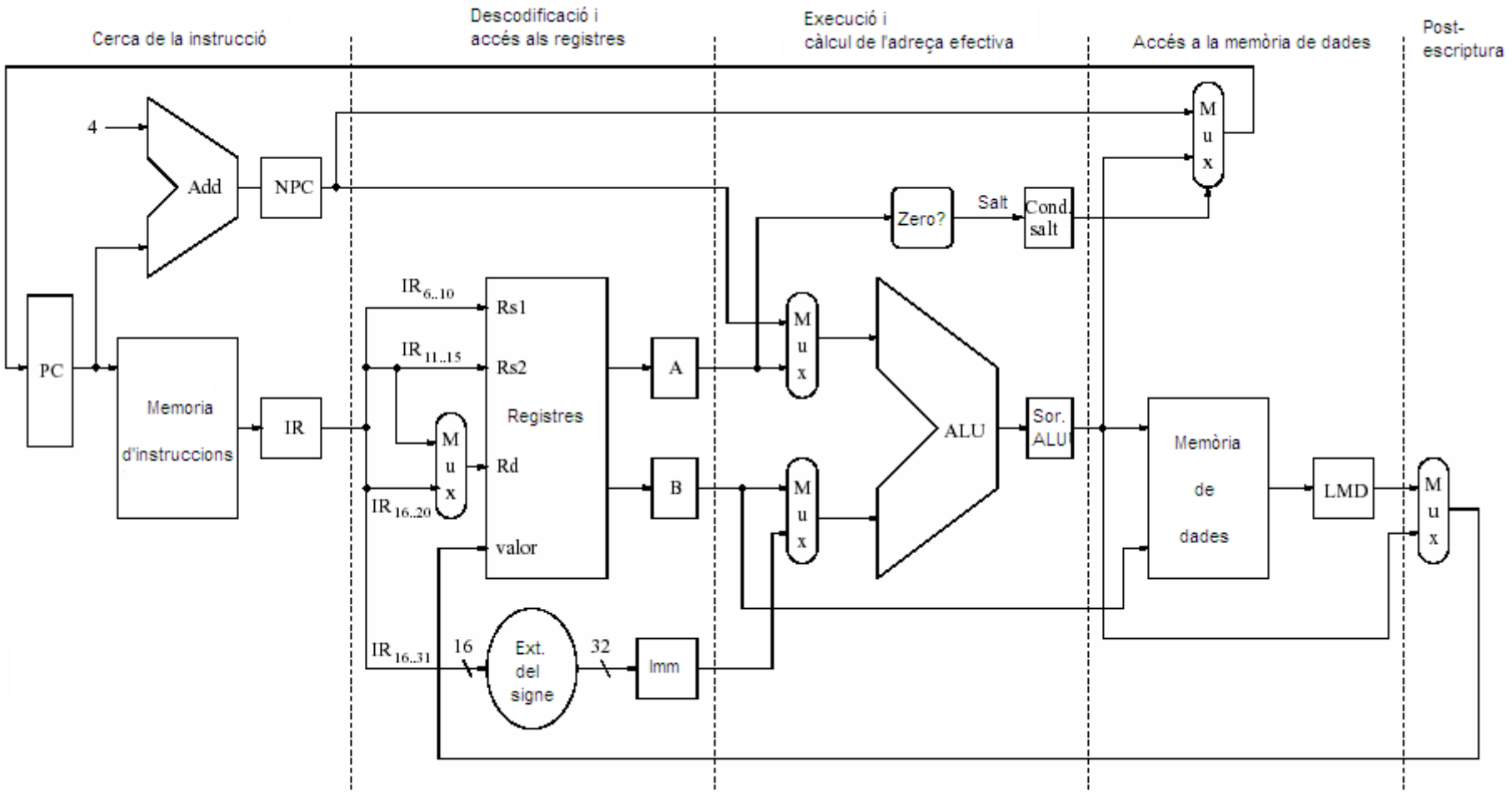
Introducció

Que és?

- La segmentació (pipelining) és una tècnica basada en la divisió de l'execució de les instruccions en etapes per poder-les solapar.
- Cadascuna de les etapes ha de completar les seves accions en un cicle de rellotge, passant els seus resultats a la següent etapa i rebent-les de l'anterior.
- Son necessaris uns registres intermedis per emmagatzemar les dades

Segmentació

Camí de dades:



Multicicle amb una estructura Harvard.

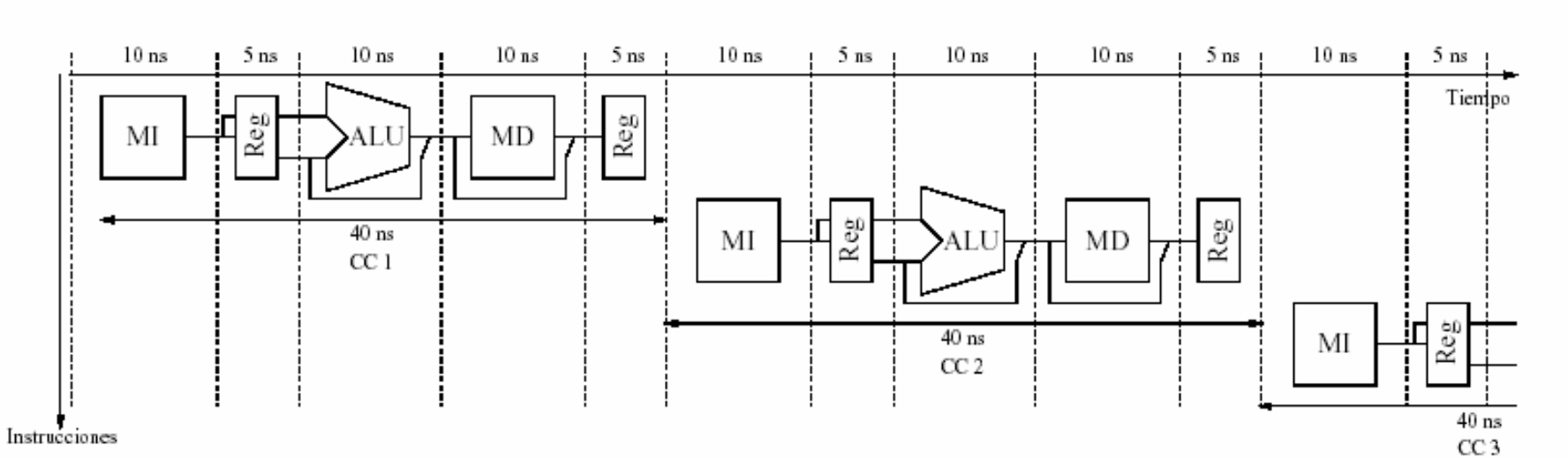
Segmentació

Camí de dades no segmentat:

Instrucció de càrrega d'una dada:

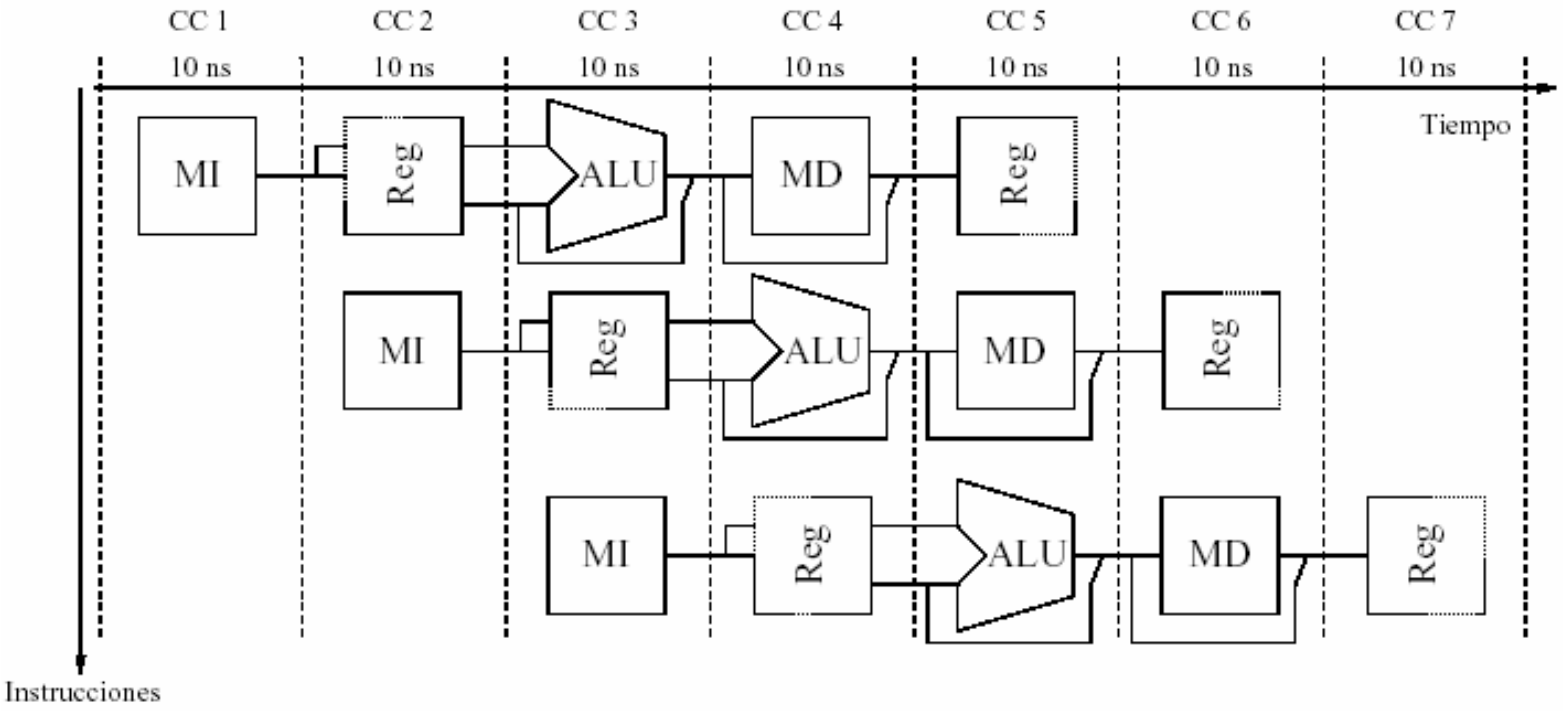
MI	– Cerca de la instrucció.	(10 ns)
Reg	– Lectura del registre.	(5 ns)
ALU	– Accés a l' ALU.	(10 ns)
MD	– Aconseguir la dada.	(10 ns)
Reg	– Escriptura en el registre.	(5 ns)

```
lw $t1,100($t2)
lw $t2,104($t2)
lw $t3,108($t2)
add  $t4, $t1, $t2
....
```



Segmentació

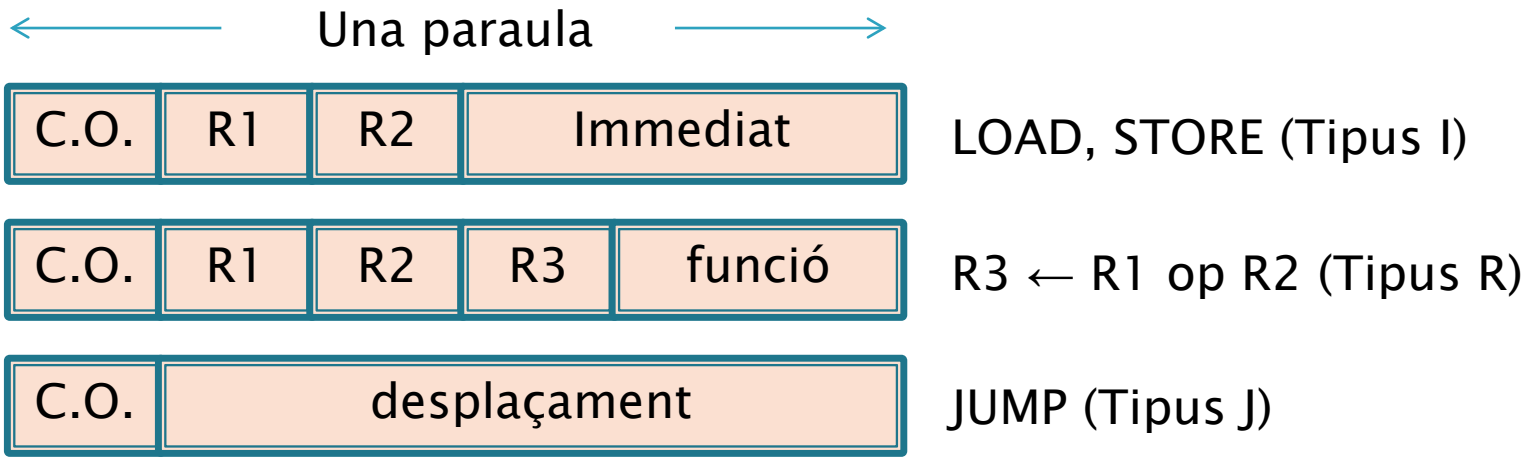
Camí de dades segmentat:



Segmentació de 5 etapes que permet executar 5 instruccions simultàniament. En aquest exemple, només 3.
El cicle de rellotge s'ha d'adaptar a la instrucció més lenta (10 ns). Les més ràpides s'han d'esperar.
Un recurs (Reg, ALU, memòria) només hi pot accedir un sol segment.

Segmentació

CAS D'ESTUDI: Model registre-registre



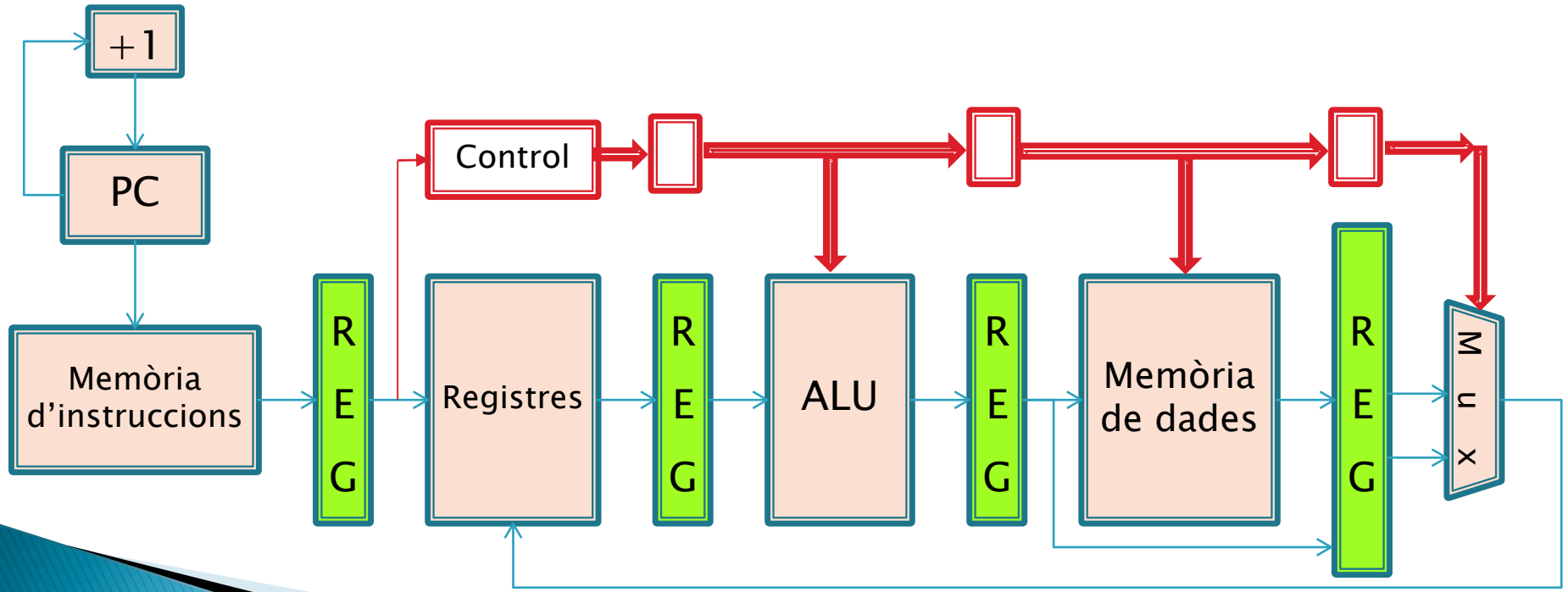
Fases:

1) BI: Fetch i increment del PC
2) DI: Descodificació i lectura dels registres
3) EX: Execució: Operació o càlcul d'adreces
4) MEM: Accés a la memòria de dades
5) PE: Escriptura als registres

Segmentació

Camí de dades segmentat:

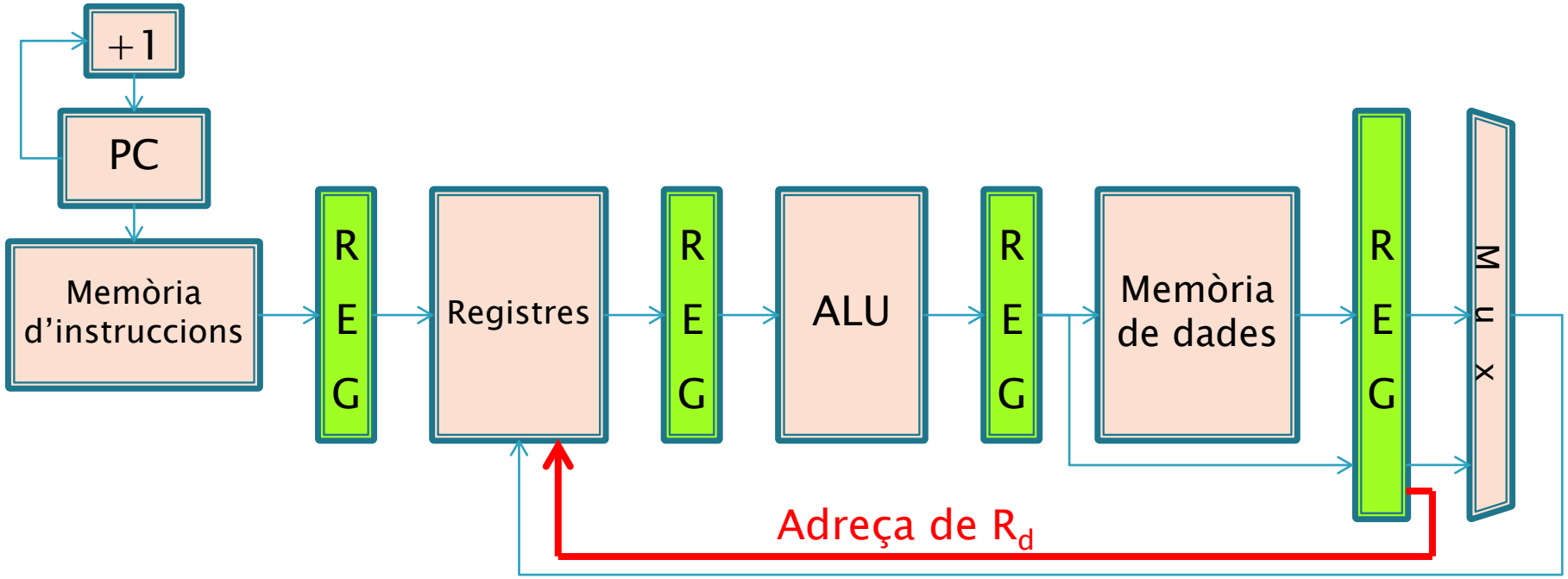
Es pot observar en tot moment si hi ha conflictes.
Problema 1: El codi de la instrucció i els registres es necessiten durant tota l'execució de la instrucció i ja que son compartits entre tots els segments al passar d'una fase a un altre es perdrien i per tant s'han d'afegir registres entre cada fase, que ens permetin desplaçar els valors de fase a fase.



Segmentació

Camí de dades segmentat:

Càrrega d'una dada: de memòria a registre. S'ha de modificar l'estructura introduint un camí que permeti passar dels darrers registres de segmentació l'adreça del registre on s'ha d'emmagatzemar.



Segmentació

Camí de dades segmentat: LOAD

1. Cerca d'instrucció: S'agafarà el contingut del comptador de programa i es buscarà la instrucció corresponent a la memòria d'instruccions. Tant el comptador de programa com la instrucció es guardaran en el registres de segmentació per poder ser utilitzats, si es necessari, en les següents etapes de la segmentació. Finalment s'incrementarà el contingut de comptador de programa.
2. Descodifica la instrucció i accedeix al registre rt i rs (Base)
3. Càlcul de l'adreça efectiva. Es calcula a l'ALU l'adreça de memòria que s'ha d'accedir i es guarda als registres de segmentació per a la fase següent tota la informació de l'adreça de memòria i els valors dels registres de l'ALU, així com el registre sobre el que s'ha de fer l'escriptura rt posteriorment.
4. Accés a memòria. Es va a buscar la dada a l'adreça de memòria i es guarda al registre de segment juntament amb l'adreça del registre a on s'ha de guardar.
5. Escriptura en el registre. Finalment s'ha de passar el contingut del registre de segmentació la lectura de la memòria i l'adreça del registre per poder fer l'escriptura de la dada en el registre corresponent.

LW rt, Desp(Base)

rt = Mem[(Base) + Desp]

Segmentació

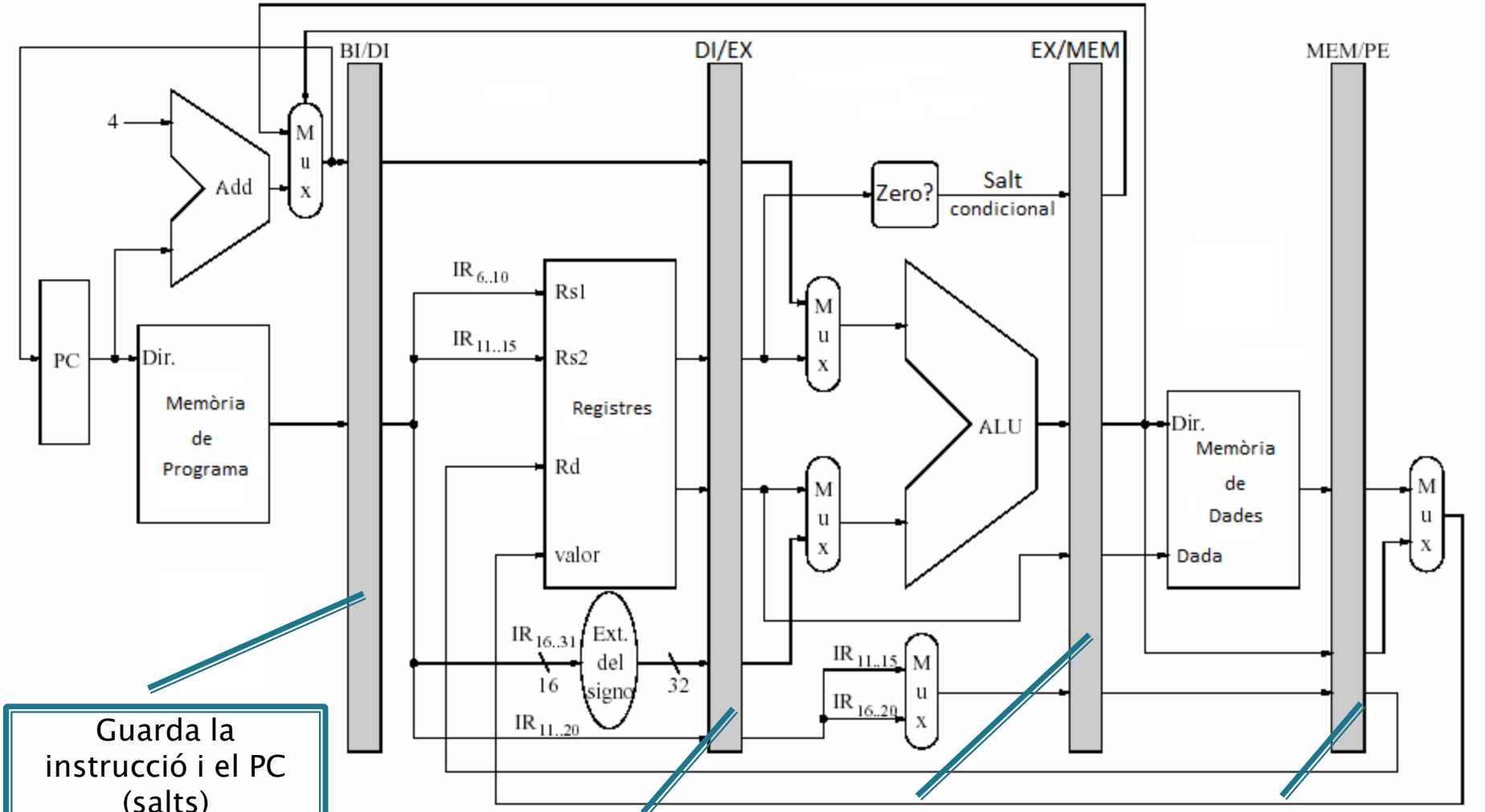
Camí de dades segmentat: STORE

1. Cerca d'instrucció: S'agafarà el contingut del comptador d'instruccions i es buscarà la instrucció corresponent a la memòria d'instruccions. Tant el comptador d'instrucció com la instrucció es guardaran en el registres de segmentació per poder ser utilitzats, si així s'escau, en les següents etapes de la segmentació. Finalment s'incrementarà el contingut de comptador d'instruccions.
2. Descodificació de la instrucció i accés al registre rs i rt.
3. Càlcul de l'adreça efectiva: (Base) + Desp. Es calcula a l'ALU l'adreça de memòria que s'ha d'accedir i es guarda al registre de segmentació per a la fase següent tota la informació de l'adreça de memòria i els valors dels registres de la CPU, així com el contingut del registre rt.
4. Accés a memòria. Es guarda a l'adreça de memòria la dada del registre (rt).
5. Escriptura en el registre. No ha de fer res.

SW rt, Desp(base)

Mem[(Base) + Desp] = rt

Segmentació: Camí de dades



Guarda la instrucció i el PC (salts)

Guarda el valor RS1, RS2, el número de registre destí (NRd) i el PC

Guarda la sortida de l'ALU, la condició de salt, el valor a guardar a memòria (Store) i el NRd

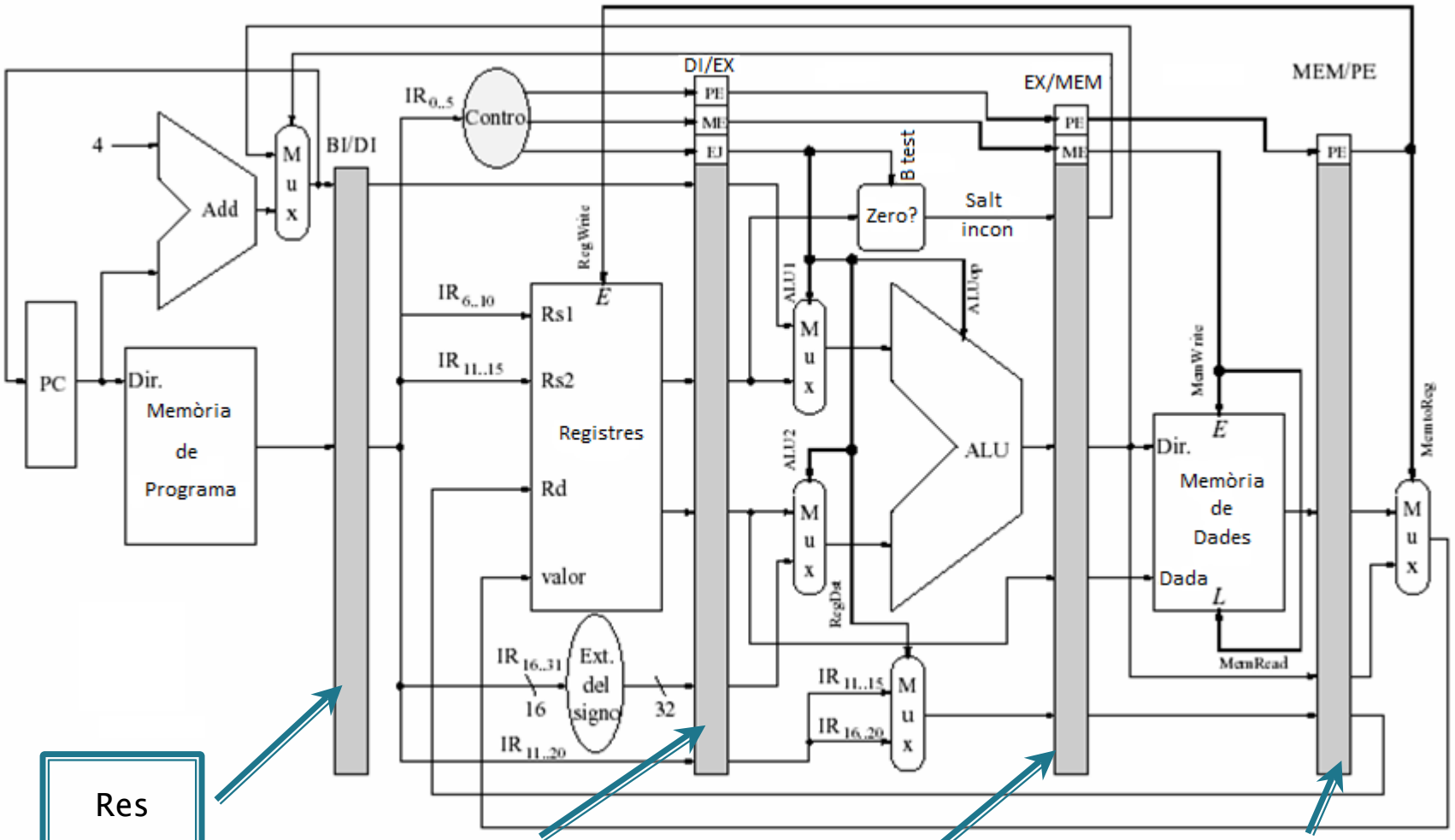
Guarda la sortida de memòria, la sortida de l'ALU i NRd

Segmentació

Registres de la segmentació:

- Necessaris per mantenir els valors que es transfereixen entre les diferents etapes
- BI/DI: Guarda la instrucció i el PC (salts)
- DI/EX: Guarda el valor RS1, RS2, el número de registre destí NRd i el PC
- EX/MEM: Guarda la sortida de l'ALU, la condició de salt, el valor a guardar a memòria (Store) i el NRd
- MEM/PE: Guarda la sortida de memòria, la sortida de l'ALU i NRd

Segmentació: Control



Res

Control de multiplexors d'entrada a l'ALU, operació de l'ALU, selecció del destí i control del salt

Selecció de lectura o escriptura

Selecció del valor a escriure i activació d'escriptura en registres

Segmentació

Senyals de control del sistema segmentat

- Senyals de control generades en l'etapa DI (etapes BI i DI no requereixen senyals de control especifiques respecte l'operació a realitzar)
- Els registres de segmentació transfereixen les senyals de control entre les diferents etapes
- Etapa EX: Control de multiplexors d'entrada a l'ALU, operació de l'ALU, selecció del destí i control del salt
- Etapa MEM: Selecció de lectura o escriptura
- Etapa PE: Selecció del valor a escriure i activació d'escriptura en registres.

Segmentació

Senyals de control del sistema segmentat

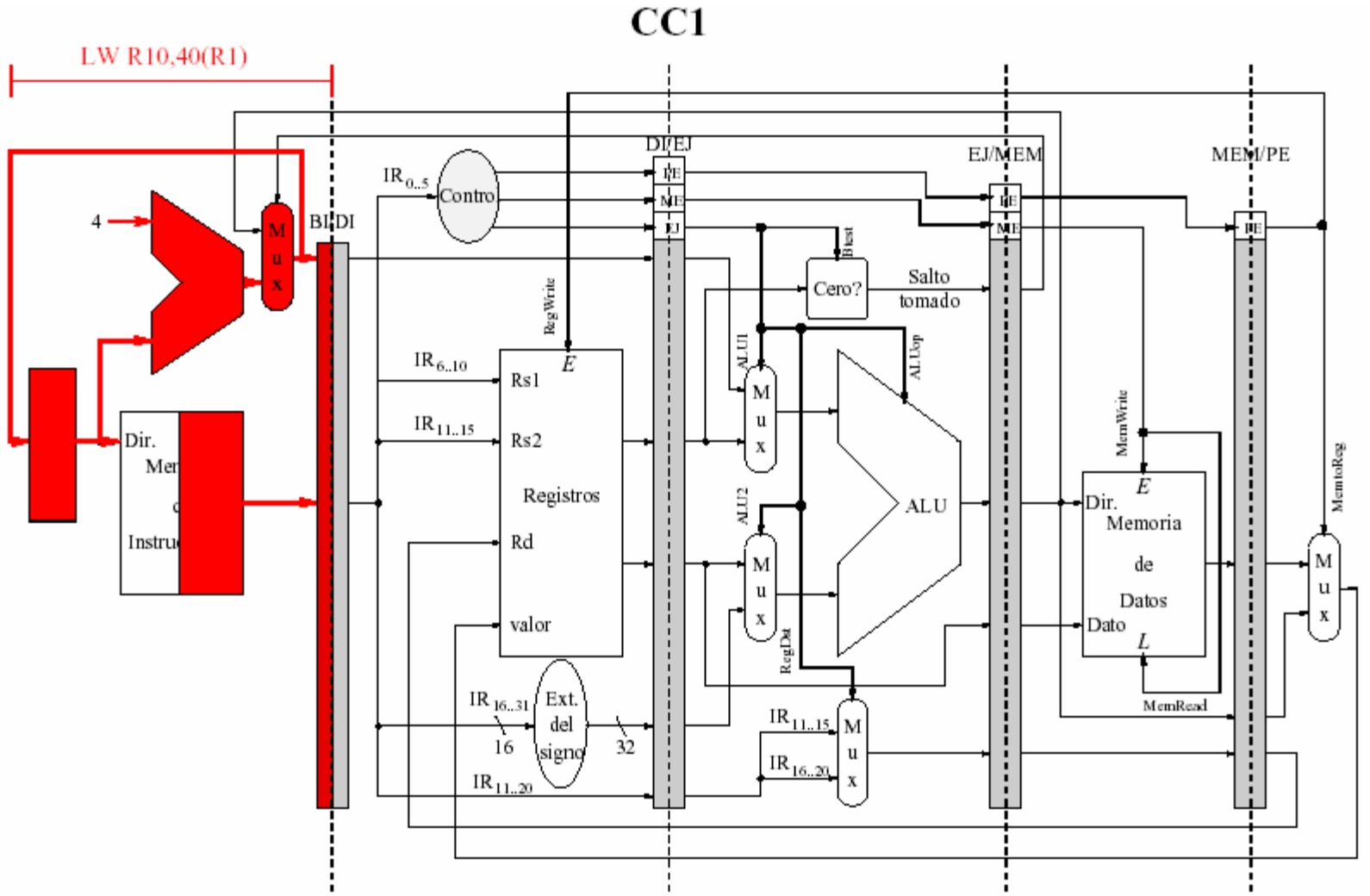
	Senyal	Efecte si val 0	Efecte si val 1
EX	ALU1 ALU2 RegDst Btest ALUop	PC -> ALU(1) Reg -> ALU(2) Reg.destí = $IR_{11..15}$ Res Suma	Reg -> ALU(1) Ext. Signe -> ALU(2) Reg.destí = $IR_{16..20}$ Comprovar el salt Resta
ME	MemWrite MemRead	Res Res	Escriu a memòria Llegeix de memòria
PE	MemtoReg RegWrite	Mem -> Reg Res	ALU -> Reg Escriu registre

Segmentació

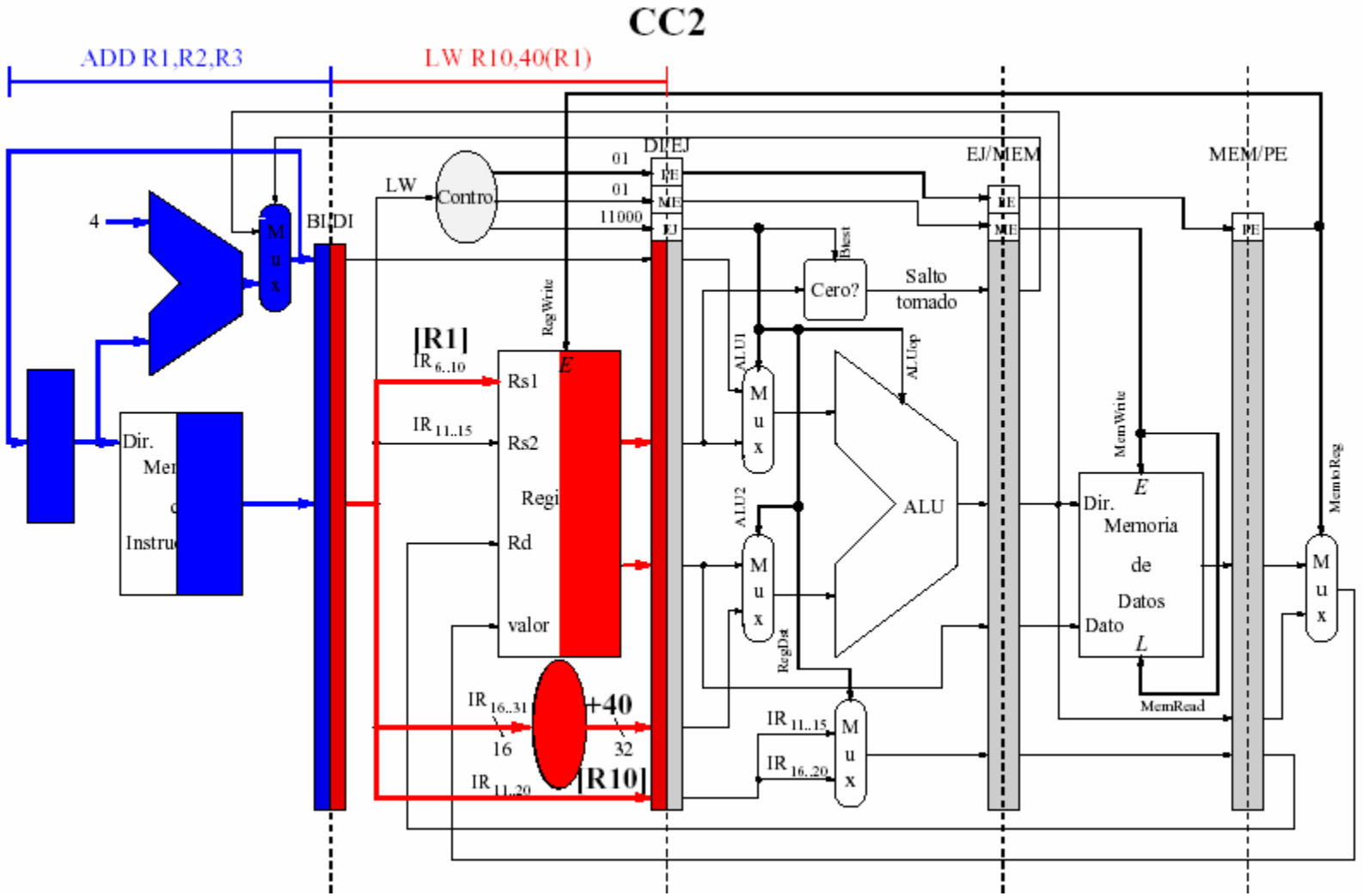
Exemple: Evolució de la segmentació en el camí de dades:

- LW R10, 40(R1) ; R10 = Mem[(R1)+40]
- ADD R1, R2, R3 ; R1 = R2 + R3
- SW R5, 0(R4) ; Mem[(R1)+40] = R5
- SUB R2, R2, R6 ; R2 = R2 – R6
- BEQ R1,R5, dest ; Si (R1=R5) → PC = PC + dest
- SW R10, 40(R1) ; Mem[(R1)+40] = R10

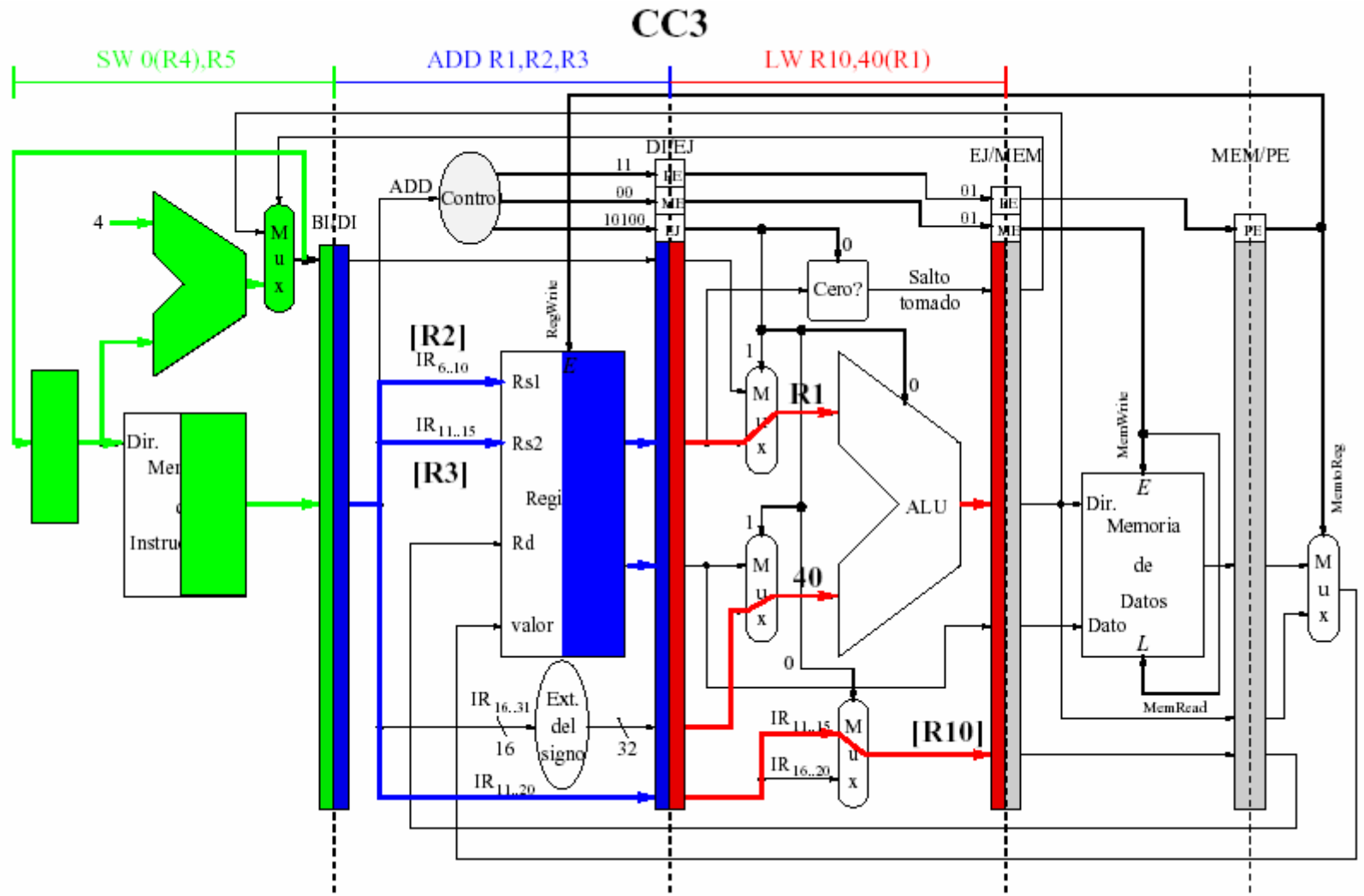
Segmentació



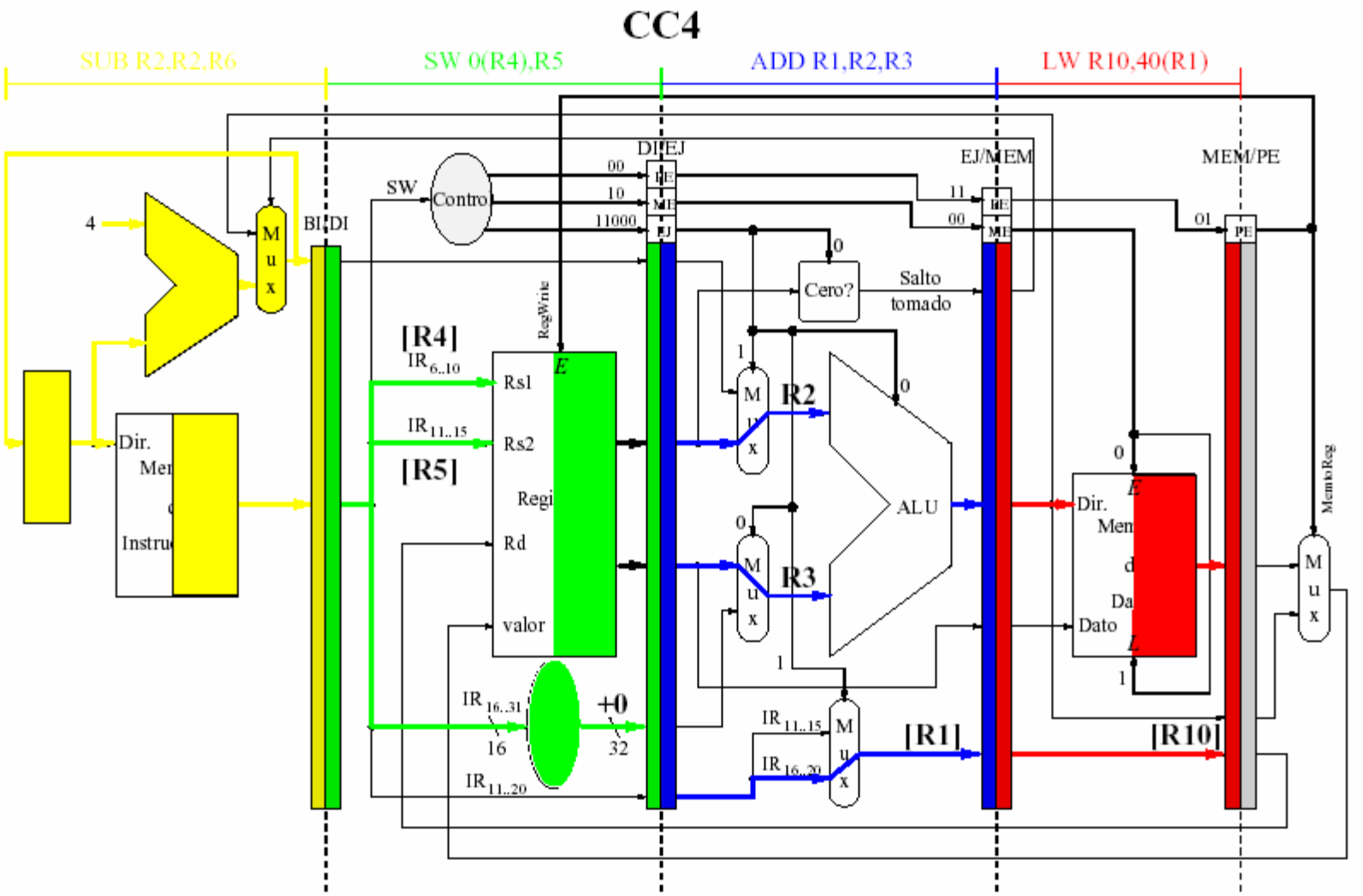
Segmentació



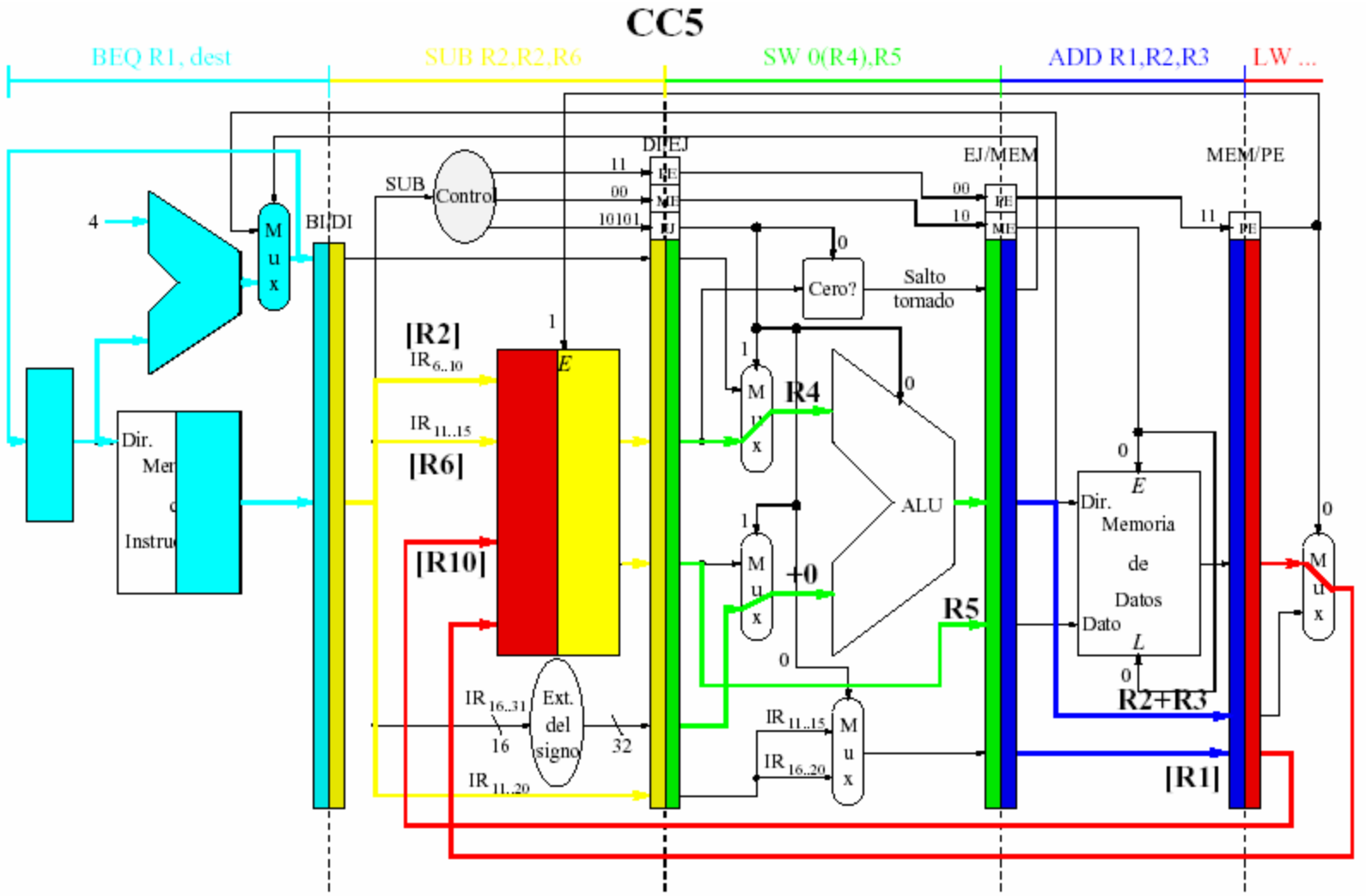
Segmentació



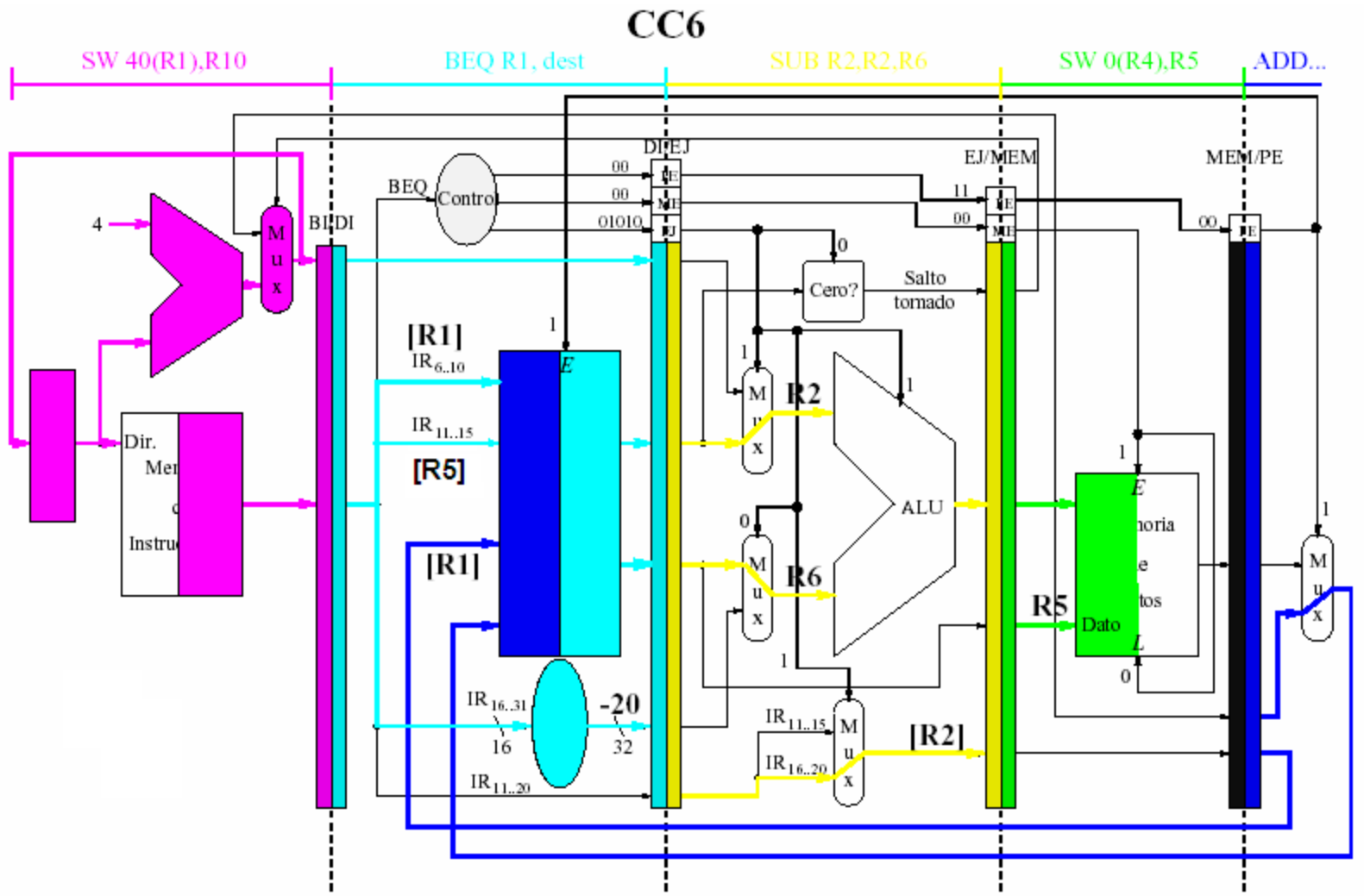
Segmentació



Segmentació



Segmentació



Segmentació (Dependències)

- **Dependència de recursos**: Dues instruccions necessiten utilitzar el mateix recurs dels processador al mateix temps.

  Per evitar aquest problema s'han de realitzar modificacions a nivell estructural. Per exemple:
  - Introduir dos camins per accedir a memòria: un per obtenir la instrucció i l'altre per accedir a les dades.
  - Permetre dos accessos de lectura als registres i un d'escriptura (al mateix cicle)
  - Establir un sumador propi pel PC, per evitar que s'utilitzi l'ALU per calcular l'adreça de la següent instrucció.
- **Dependència de dades entre instruccions**: L'execució d'una instrucció depèn dels resultats d'altres instruccions que poden estar sent executades simultàniament amb la primera.
  - Dependències vertaderes,
  - Antidependències
  - Dependències de sortida.

- **Dependències de control**: Degudes a les instruccions de salt

Poden limitar el rendiment del sistema segmentat

Segmentació (Dependències)

- **Dependència de dades entre instruccions**
  - Dependències vertaderes (RAW): Una instrucció *j* es depenent d'una altra *i* si es verifica una de les dues condicions:
    - La instrucció *i* genera un resultat utilitzat per *j*.
    - La instrucció *j* és depenent d'una instrucció *k*, i *k* depèn de *i*.

Exemple:

```
I1:  LD R2, 0(R1)
I2:  ADD R4, R2, R3
I3:  SD 0(R1), R4
```

Dependència 1
entre I1 i I2
R2

Dependència 2
entre I2 i I3
R4

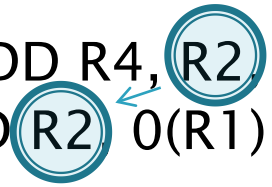
Dependència 3
entre I1 i I3
R4 <- R2

Segmentació (Dependències)

- **Dependència de dades entre instruccions**
  - Antidependència (WAR): Una instrucció *i* precedeix a una altra *j* , i *j* modifica un registre o una posició de memòria que es llegit posteriorment per *i*.

Exemple:

I_1: ADD R4, R2, R3
I_2: LD R2, 0(R1)



Dependència entre I_1 i I_2 R2

Problema: si I_2 acaba abans que I_1

  - Dependència de sortida (WAW): Si *i* i *j* escriuen al mateix registre o posició de memòria.

Exemple:

I_1: ADD R4, R2, R3
I_2: LD R4, 0(R1)

Dependència entre I_1 i I_2 R4

Problema: S'ha d'assegurar que l'escriptura al registre R4 de I_2 es faci més tard que la de I_1

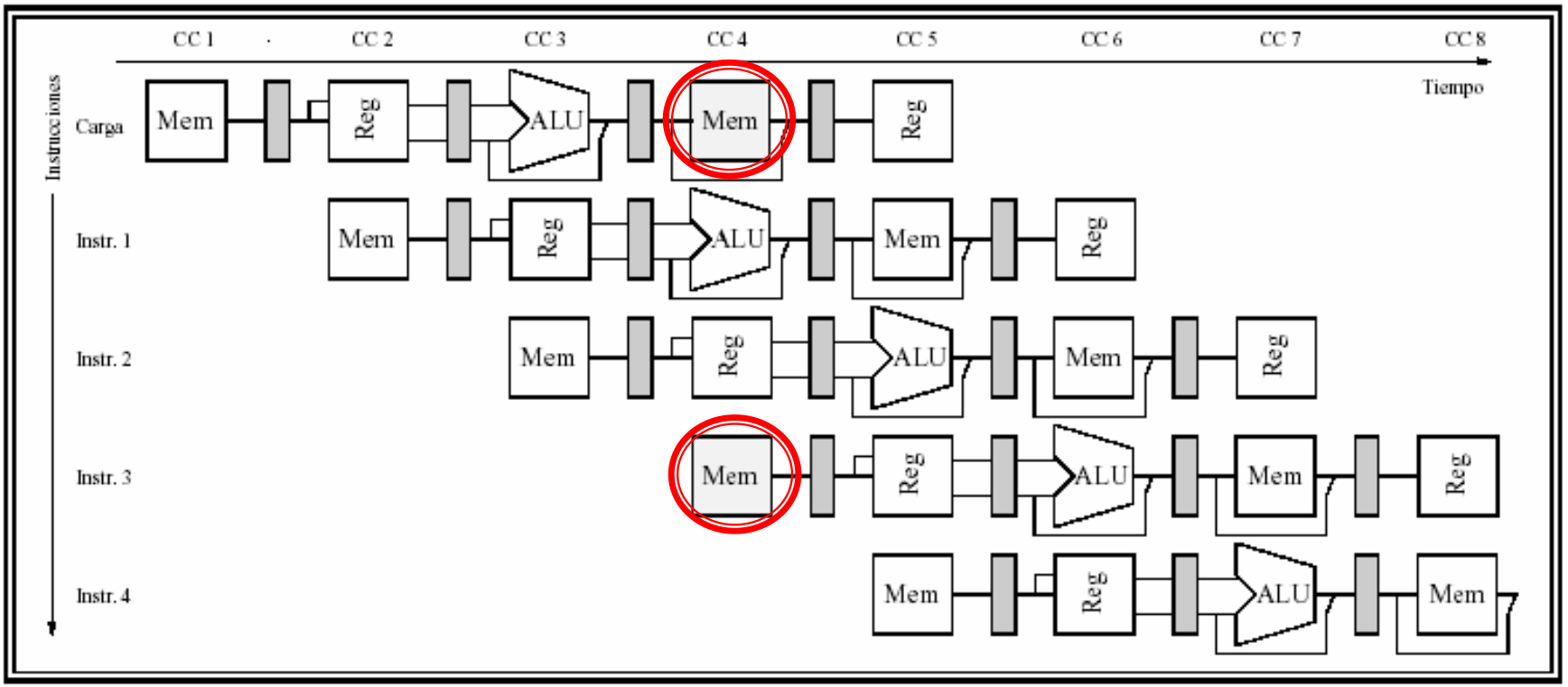
Es poden donar amb processadors segmentats amb camis de dades diferents segons els tipus de funció

Segmentació (Dependències)

- Quan l'existència d'una dependència impedeix que una instrucció sigui executada en el cicle de rellotge que li correspondria, es produeix un **Risc**
- Hi ha tres tipus de riscos:
  - **Risc estructural**: Deriven d'una dependència de recursos, quan el maquinari no pot suportar totes les combinacions possibles d'instruccions en les execucions.
  - **Risc per Dependència de dades**: Sorgeixen quan una instrucció depèn dels resultats d'una instrucció anterior, de tal manera que podrien executar-se simultàniament. Deriven de dependències de dades.
  - **Risc de control**: Sorgeixen de la segmentació dels salts i altres instruccions que modifiquen el PC. Deriven de dependències de control.

Segmentació (Risc estructural)

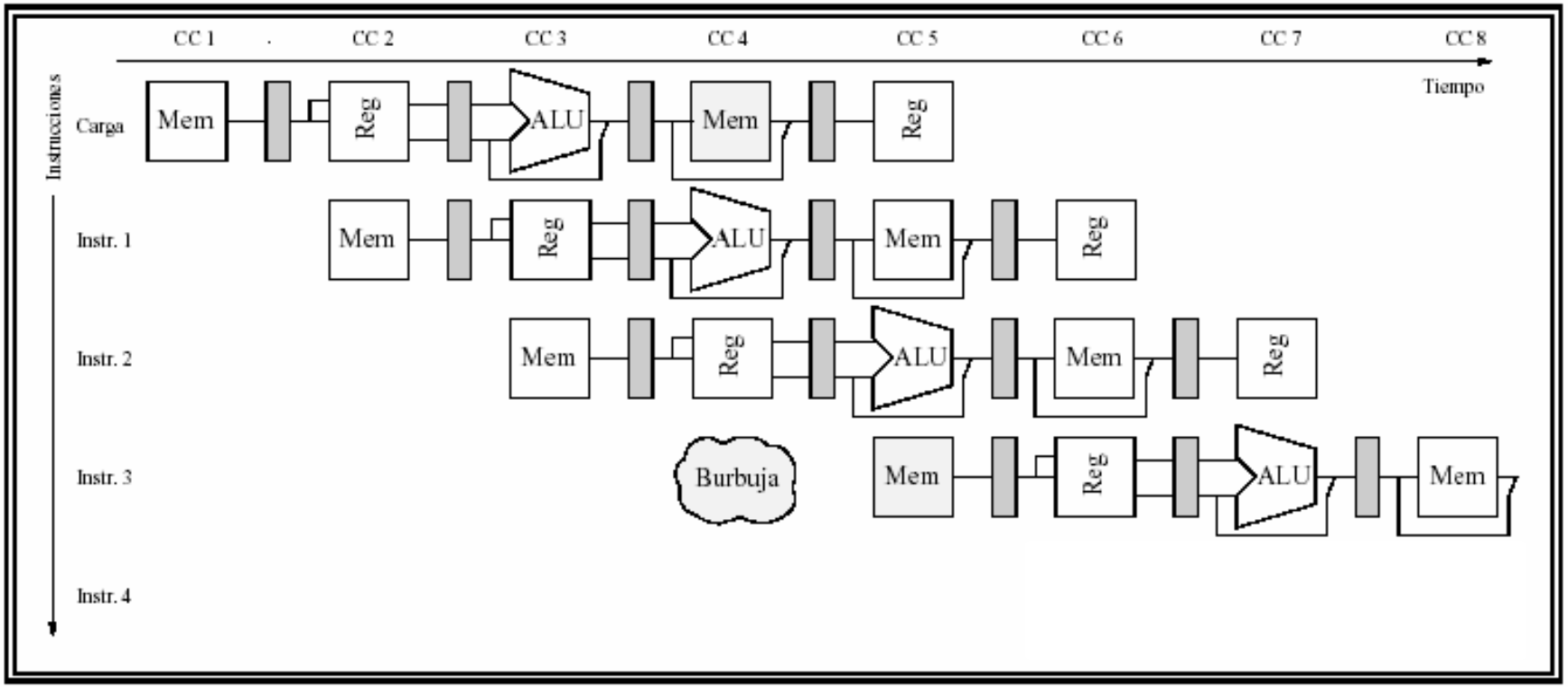
Exemple: Sistema amb una memòria única de dades i instruccions



El Risc estructural impedeix que la instrucció tres comenci a executar-se en el cicle 4

Segmentació (Risc estructural)

Solució: Parar l'avanç de les instruccions durant un cicle



Hi ha una pèrdua en l'ample de banda degut a la parada

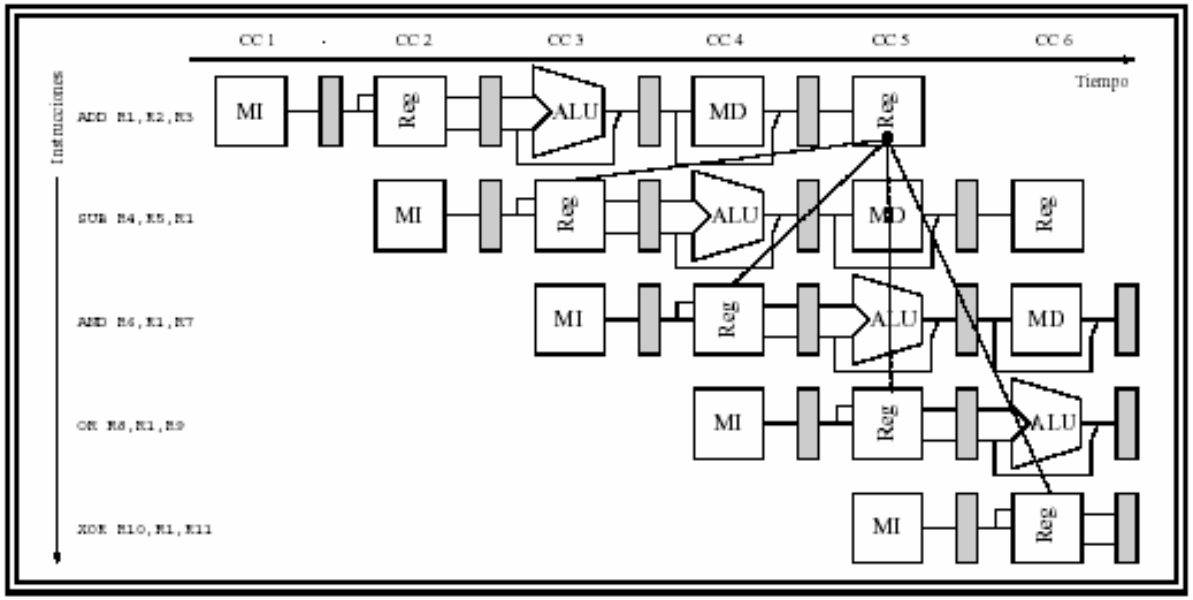
Segmentació (Risc de dades)

- Degut al canvi que la segmentació produeix en l'ordre d'accessos als operants, respecte a l'execució seqüencial, es poden considerar tres tipus:
  - **Risc RAW (read after write)** Una instrucció *j* intenta llegir un operant que ha de ser modificat per una instrucció prèvia *i*. Sorgeix de dependències vertaderes.
  - **Risc WAR (write after read)** *j* intenta escriure un destí abans que sigui llegit per *i*. Sorgeix d'antidependències
  - **RISC WAW (write after write)** *j* intenta escriure un operant abans que sigui escrit per *i*. Sorgeix de dependències de sortida.

Riscs RAW:

I1: ADD R1, R2, R3
I2: SUB R4, R5, R1
I3: AND R6, R1, R7
I4: OR R8, R1, R9
I5: XOR R10, R1, R11

Totes les instruccions tenen una dependència vertadera amb I1. Només I2, I3 i I4 produeixen riscos RAW.



Les sortides d'una unitat funcional son realimentades com entrades a la mateixa unitat funcional o a les unitats prèvies.

•Els resultats de l'ALU s'avancen al registre EX/MEM a l'entrada de l'ALU.
•La lògica de control dels avançaments detecta si la font de la nova operació ha de ser el valor llegit dels registres o la sortida de l'ALU del cicle anterior.

Solucions als riscos de dades

- Es poden tractar de diferents formes:
  - Per programa: Si el maquinari del processador segmentat no pot detectar i resoldre els riscos, ho ha de fer el compilador.
    - Ha d'afegir instruccions del tipus "NOP" després de cada instrucció que causi o pugui causar riscos. Evita el mal funcionament, però no redueix els retards.
    - Pot reordenar el codi del programa per millorar el funcionament.
  - Maquinari:
    - Interbloqueig: (La més simple) Detectar el risc i aturar el flux de dades fins que desaparegui.
    - Anticipació (Forwarding). (Més complex i necessita maquinari addicional) Consisteix en aprofitar les dades en el moment en que estan disponibles (curtcircuits)

Riscos RAW: Solució per programa

```
sub r2, r1, r3
and r12, r2, r5
or  r13, r6, r2
and r14, r2, r2
sw  r15, 100(r2)
```

Cicle	1	2	3	4	5	6	7	8	9	10	11
sub r2, r1, r3	IF	ID	EX	M	WR						
nop		IF	ID	–	–	–					
nop			IF	ID	–	–	–				
nop				IF	ID	–	–	–			
and r12, r2, r5					IF	ID	Ex	M	WR		
or r13, r6, r2						IF	ID	Ex	M	WR	
and r14, r2, r2							IF	ID	Ex	M	WR
sw r15, 100(r2)								IF	ID	Ex	M

Riscos RAW: Solució per programa

Reordenar el codi

```
sub r2, r1, r3
and r12, r2, r5
or  r13, r6, r2
and r14, r2, r2
sw  r15, 100(r2)
addi r1, r1, 1
```

```
sub r2, r1, r3
addi r1, r1, 1
and r12, r2, r5
or  r13, r6, r2
and r14, r2, r2
sw  r15, 100(r2)
```

Cicle	1	2	3	4	5	6	7	8	9	10	11
sub r2, r1, r3	IF	ID	EX	M	WR						
addi r1, r1, 1		IF	ID	Ex	M	WR					
nop			IF	ID	–	–	–				
nop				IF	ID	–	–	–			
and r12, r2, r5					IF	ID	Ex	M	WR		
or r13, r6, r2						IF	ID	Ex	M	WR	
and r14, r2, r2							IF	ID	Ex	M	WR
sw r15, 100(r2)								IF	ID	Ex	M

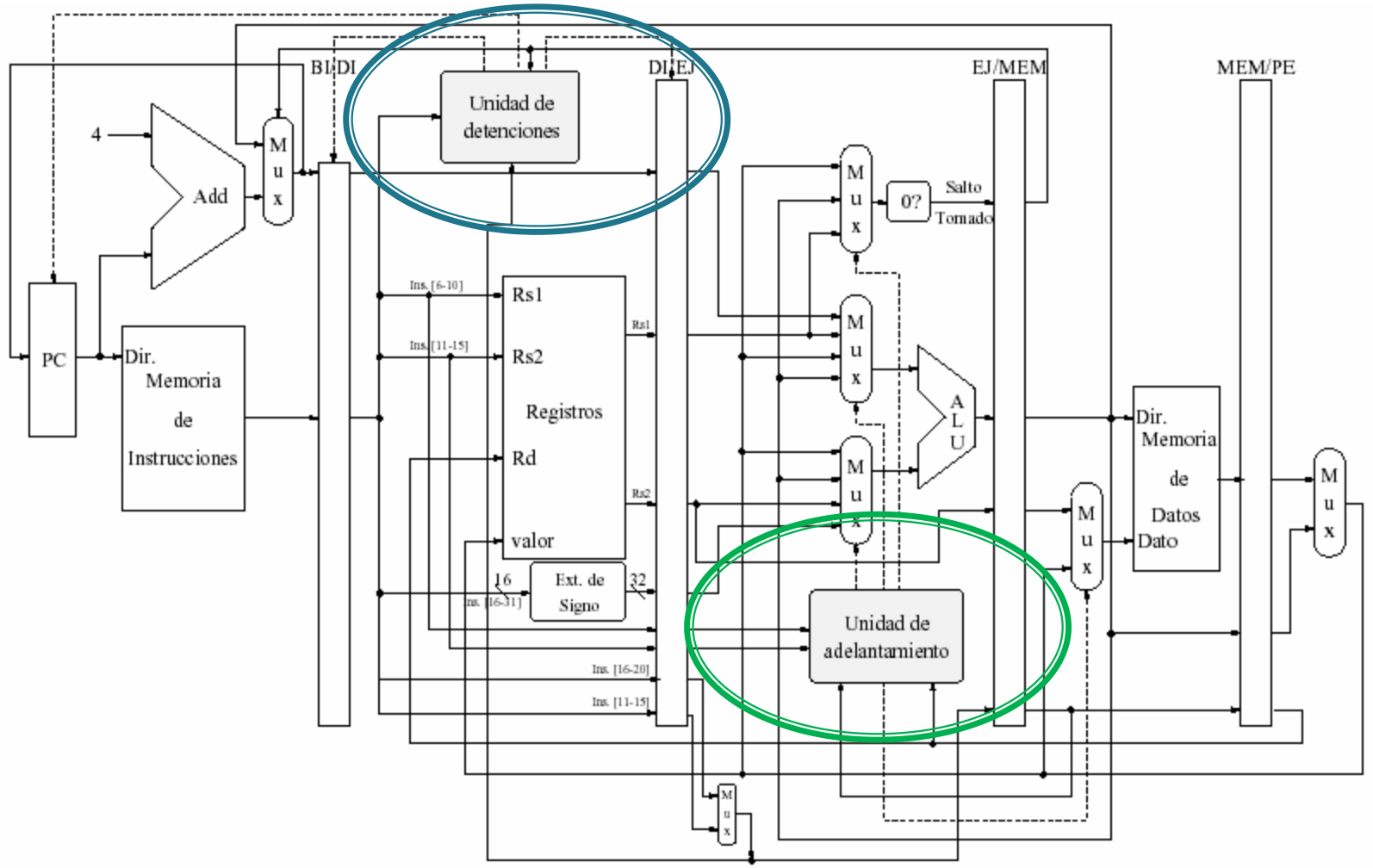
Riscos RAW: Solució interbloqueig

Cicle	1	2	3	4	5	6	7	8	9	10	11
sub r2, r1, r3	IF	ID	EX	M	WR						
and r12, r2, r5		IF	ID	ID	ID	ID	Ex	M	WR		
or r13, r6, r2						IF	ID	Ex	M	WR	
and r14, r2, r2							IF	ID	Ex	M	WR
sw r15, 100(r2)								IF	ID	Ex	M

Anticipació

- Utilitzar el resultats quan estan disponibles, abans d'escriure'ls al banc de registres.
- Anticipació al banc de registres per poder llegir i escriure al mateix registre.
- Anticipació a l'ALU, per avançar els operants a l'entrada de l'ALU, sense haver d'esperar a portar-los del banc de registres.
- Anticipació ALU-ALU: De la sortida de l'ALU, a alguna de les entrades (registre d'entrada)
- Anticipació MEM-ALU: de l'etapa d'accés a la memòria de dades s'avança la dada llegida a l'entrada de l'ALU.

Segmentació

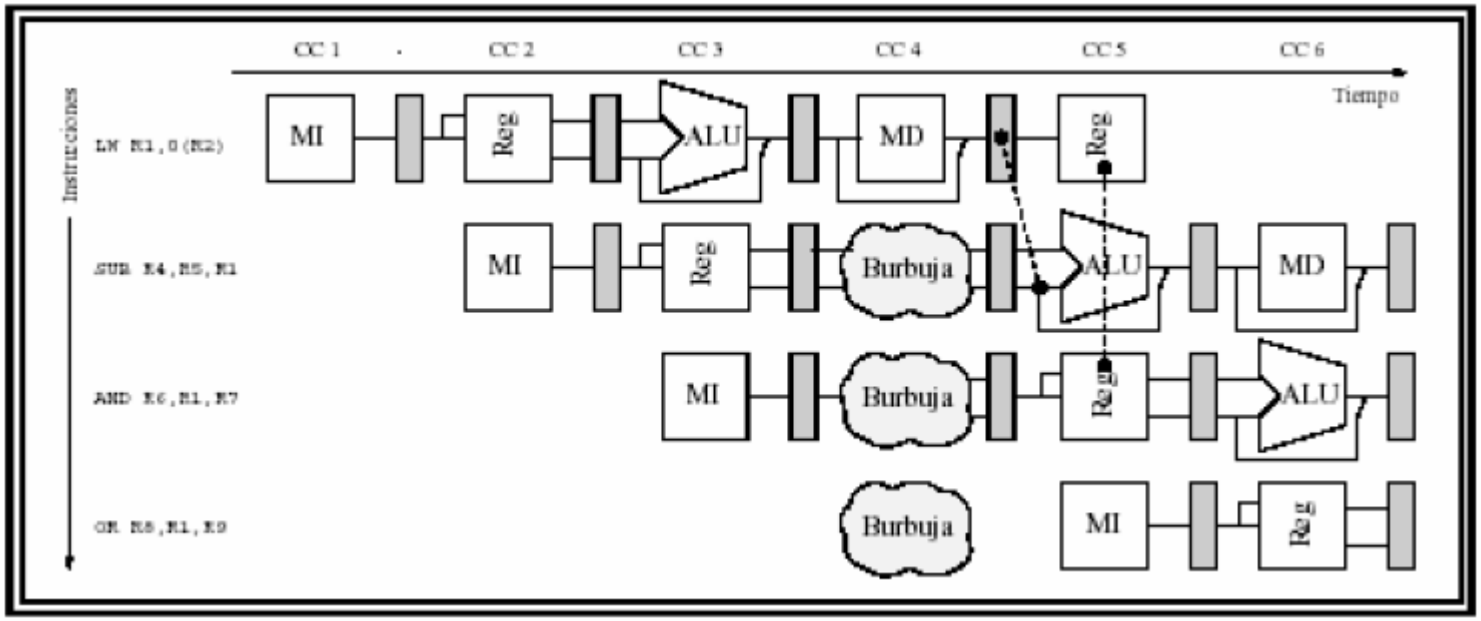


Segmentació

Reducció del risc de dependència de dades

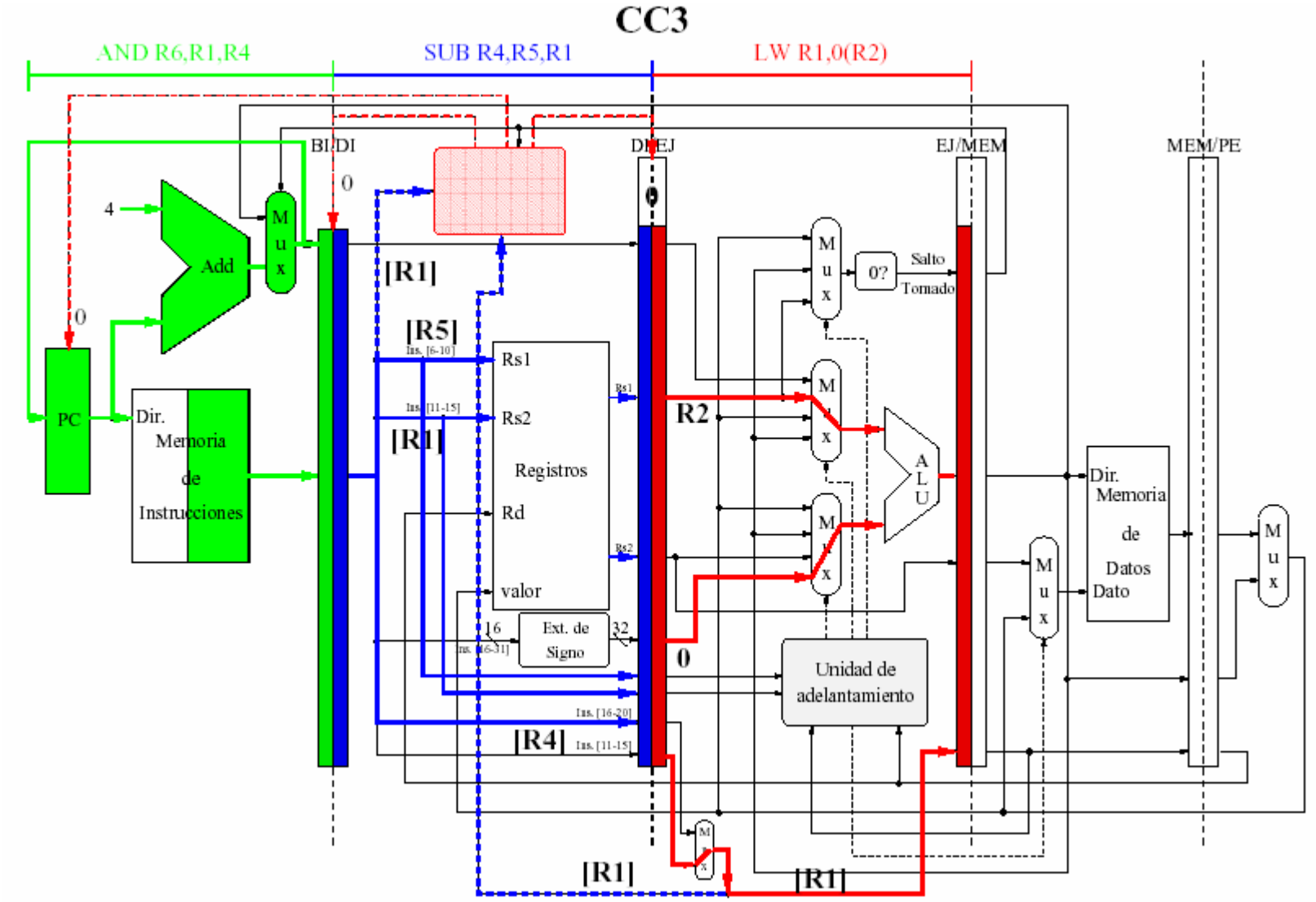
- LW R1, O(R2)
- SUB R4, R5, R1
- AND R6, R1, R4
- OR R8, R1, R9
- XOR R10, R1, R11

- R1: Dependència RAW
- R4: Dependència RAW



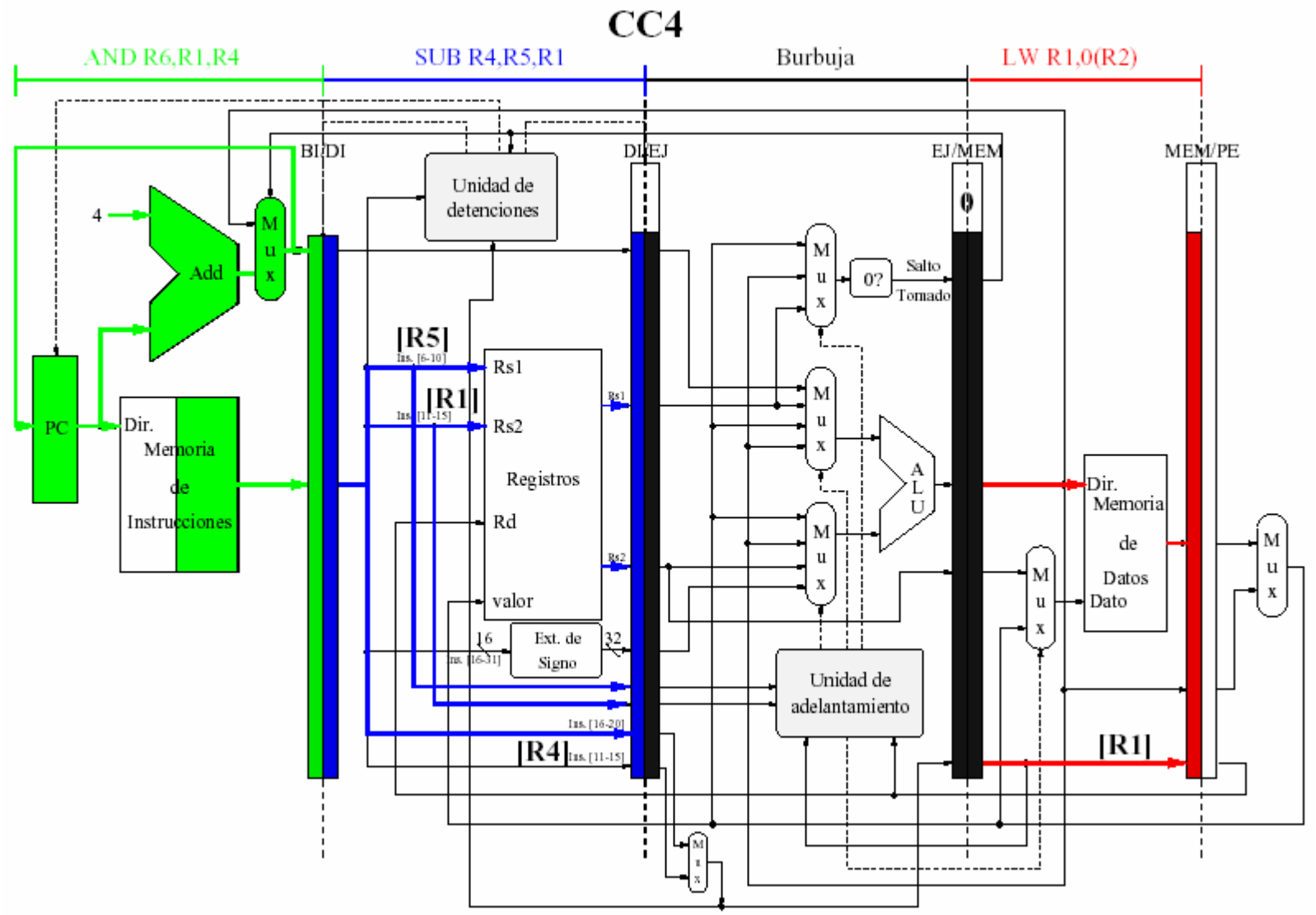
Segmentació

Reducció del risc de dependència de dades



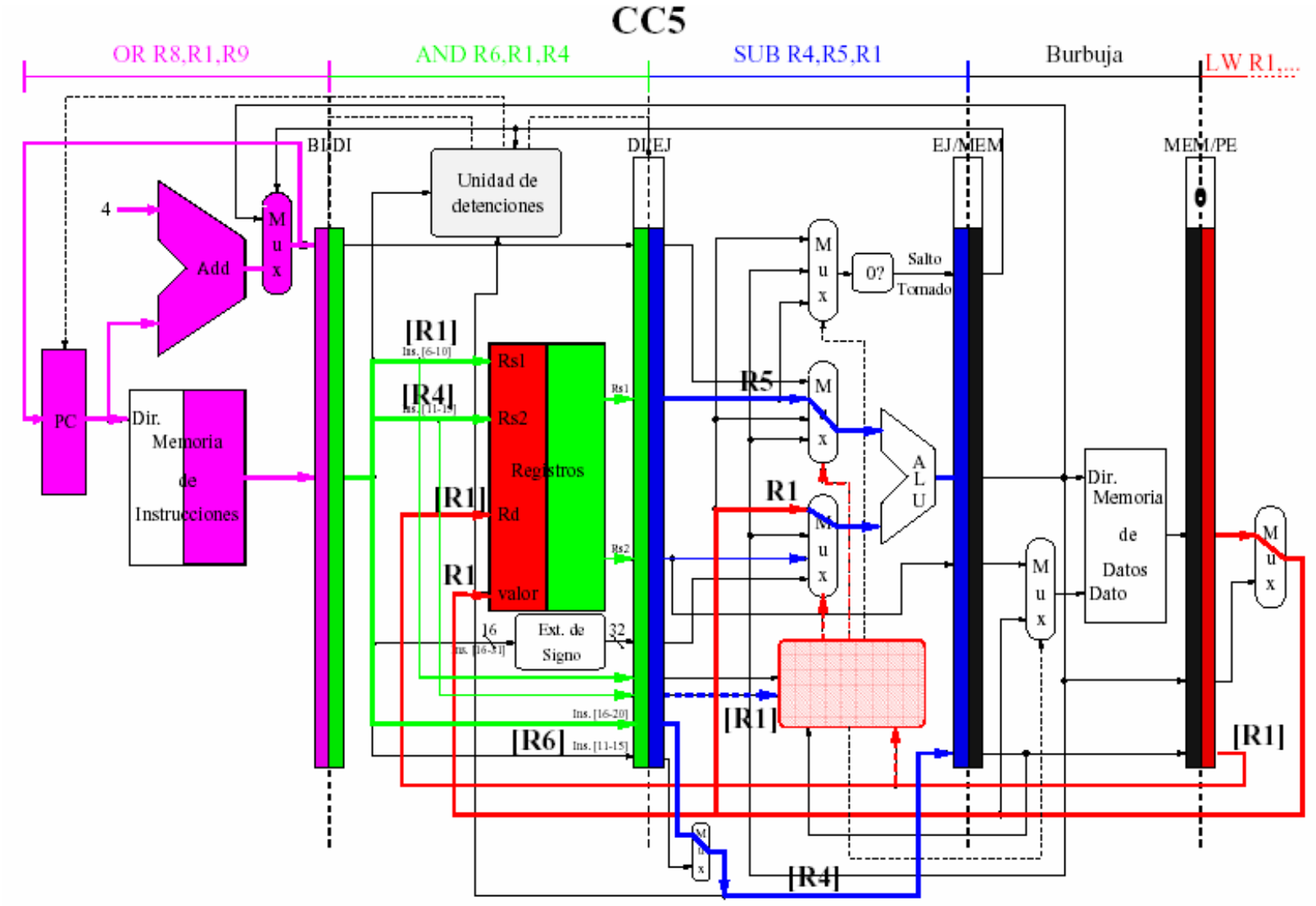
Segmentació

Reducció del risc de dependència de dades



Segmentació

Reducció del risc de dependència de dades



Segmentació

Reducció del risc de dependència de dades