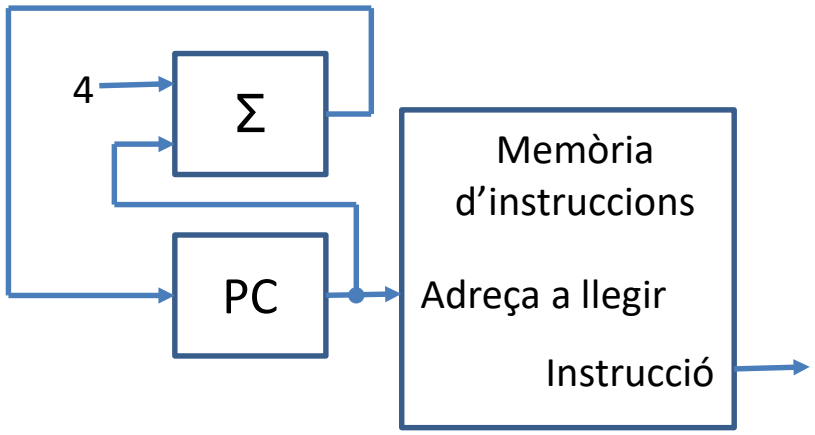
Processadors MIPS

Disseny del camí de dades
(Data path)

MIPS: Disseny del camí de dades

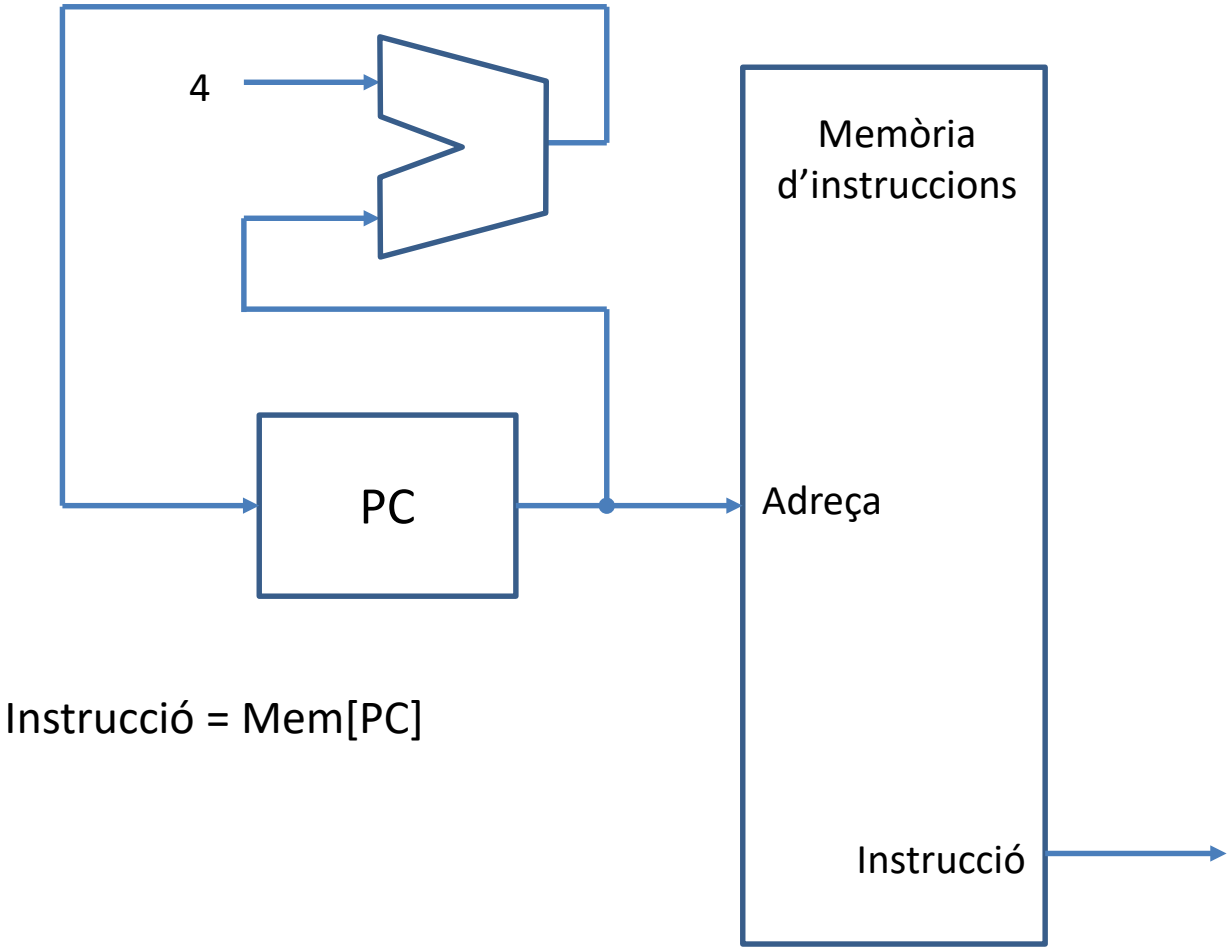
Una manera raonable de començar el disseny d'un camí de dades és examinar els components principals requerits per a executar les instruccions del MIPS. Primer es consideren els elements del camí de dades que necessita cadascuna de les instruccions i a partir d'ells es construeixen les diferents parts. Una vegada que s'ha determinat els elements necessaris, s'estudiaran els senyals de control.

1. Cerca de la instrucció (fetch)
   - Es necessita:
     - Una unitat de memòria per memoritzar les instruccions i que les pugem llegir a partir d'una adreça.
     - Un registre per guardar l'adreça de la instrucció. Rep el nom de punter de programa (PC).
     - Un sumador, que serà l'encarregat d'incrementar el PC per que passi a apuntar a la següent instrucció



Resum: per a executar qualsevol instrucció, s'ha de començar per carregar la instrucció des de la memòria.
Per poder executar la propera instrucció, s'ha d'incrementar en 4 el comptador de programa.

1. Cerca de la instrucció (fetch)



Instrucció = Mem[PC]

S'ha de donar una ordre lectura a la memòria.
Al cap d'un temps, tindrem la instrucció a bus de sortida

- Retorn

MIPS: Disseny del camí de dades

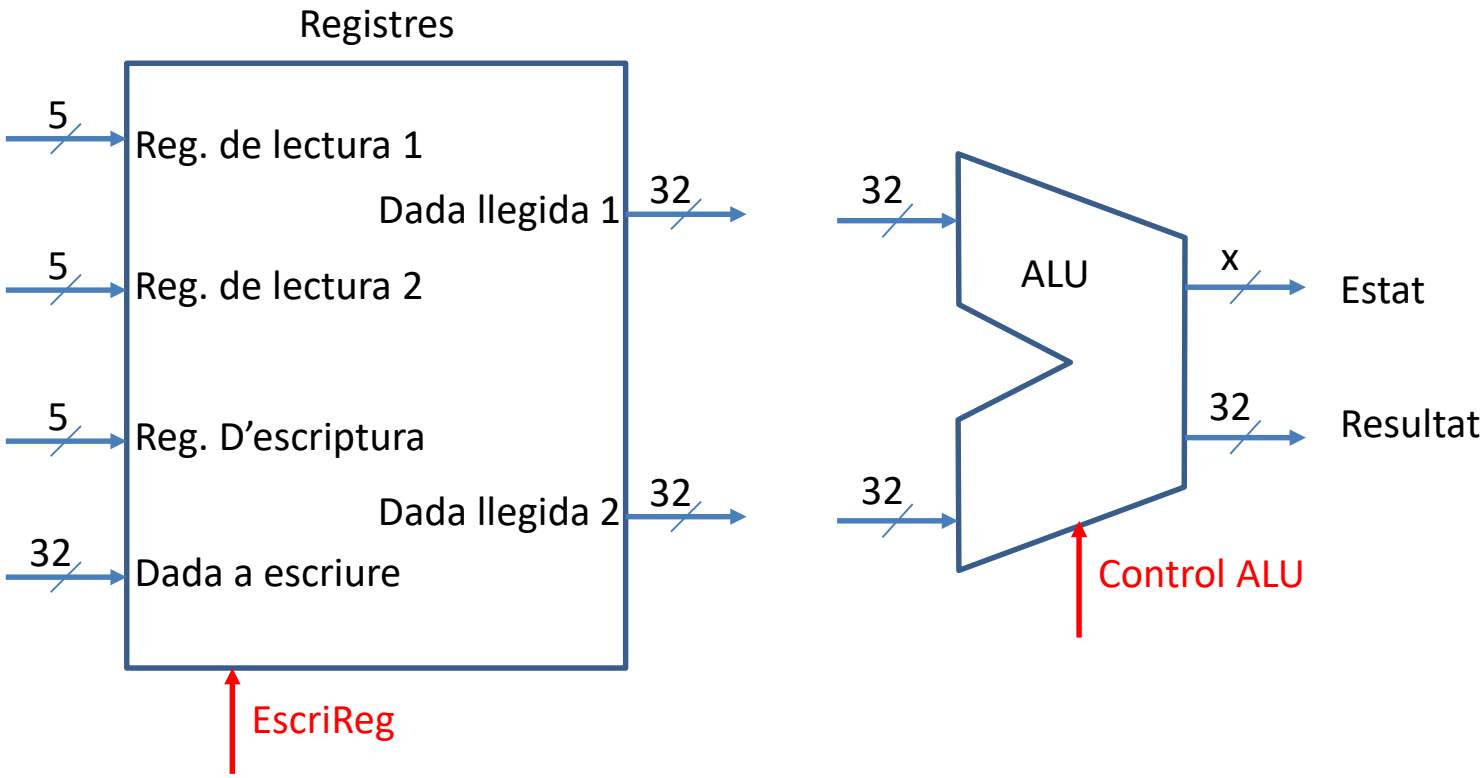
2. Instruccions del tipus R

- Aquest tipus d'instruccions agafen els operants de dos registres, els operen amb l'ALU i escriuen el resultat en un altre registre. Inclouen les instruccions aritmètiques: add, sub, slt .. I les lògiques: and, or, ..

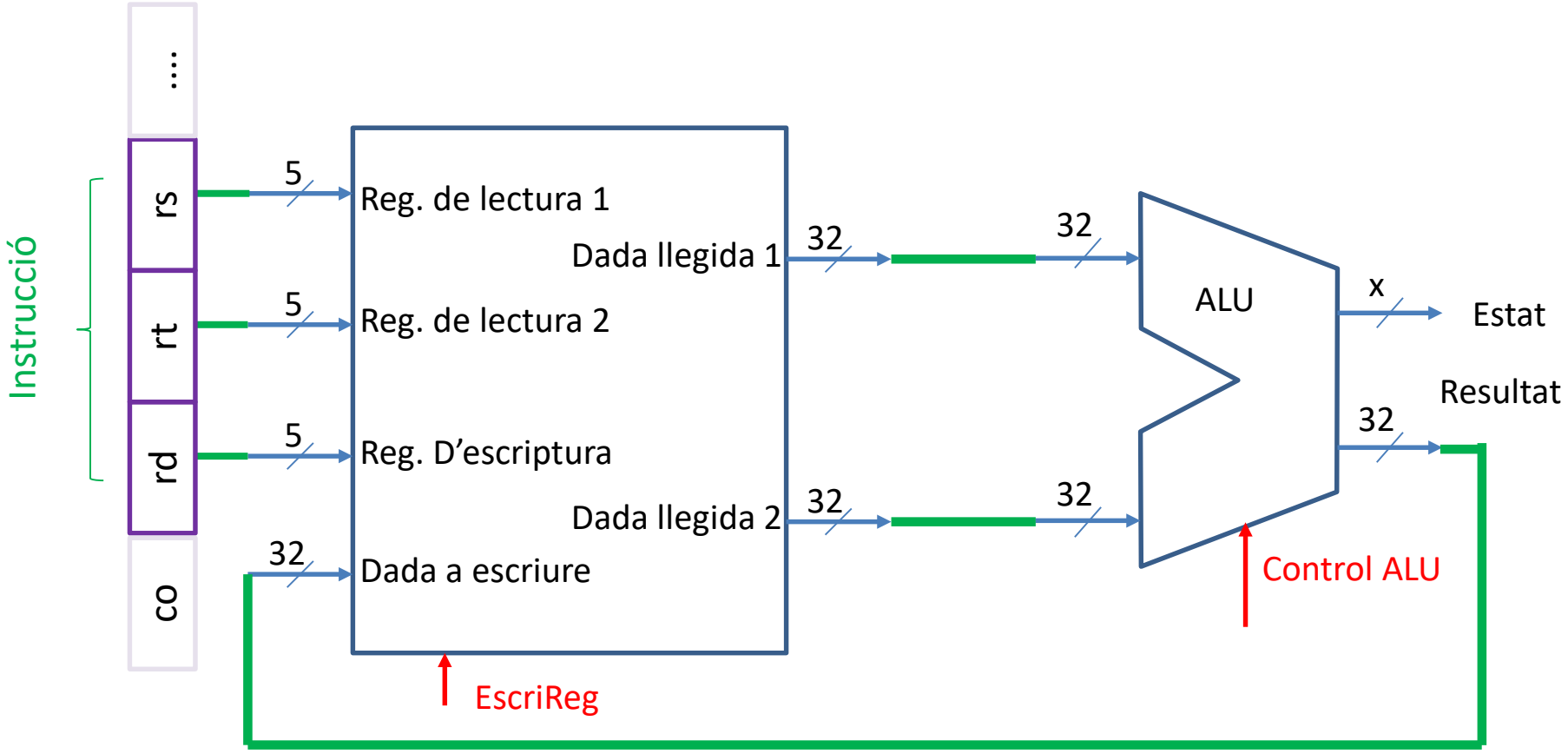
add $t1,$t2,$t3

- Es necessita:
  - Un banc de 32 registres de 32 bits. Es pot llegir o escriure en qualsevol, especificant el seu nom.
  - Una ALU per poder fer les operacions amb els valors dels registres.
- Les instruccions de tipus R tenen 3 operants:
  - Dos per apuntar els dos registres on hi ha operants fonts
  - Un per apuntar el registre que s'ha de guardar el resultat
- Els registres necessiten:
  - Accés a un operant
    - Una entrada de 5 bits per direccionar els registre (32 registres)
    - Una sortida de 32 bits per proporcionar l'operant
  - Guardar un resultat
    - Una entrada de 5 bits per direccionar el registre (32 registres)
    - Una entrada de 32 bits amb la dada a enregistrar

MIPS: Disseny del camí de dades



MIPS: Disseny del camí de dades



- Retorn

MIPS: Disseny del camí de dades

Instruccions del tipus I

Instruccions del MIPS de càrrega i emmagatzematge de paraules:

lw $t1,despl($t2)
sw $t1,despl($t2)

Calculen l'adreça de memòria afegint al registre base ($t2) el camp de desplaçament (positiu o negatiu) de 16 bits contingut en la instrucció.
Si la instrucció es un store, el valor a emmagatzemar, s'ha de llegir del banc de registres ($t1).
Si es un load, el valor llegit de la memòria, s'ha d'escriure al registre $t1.

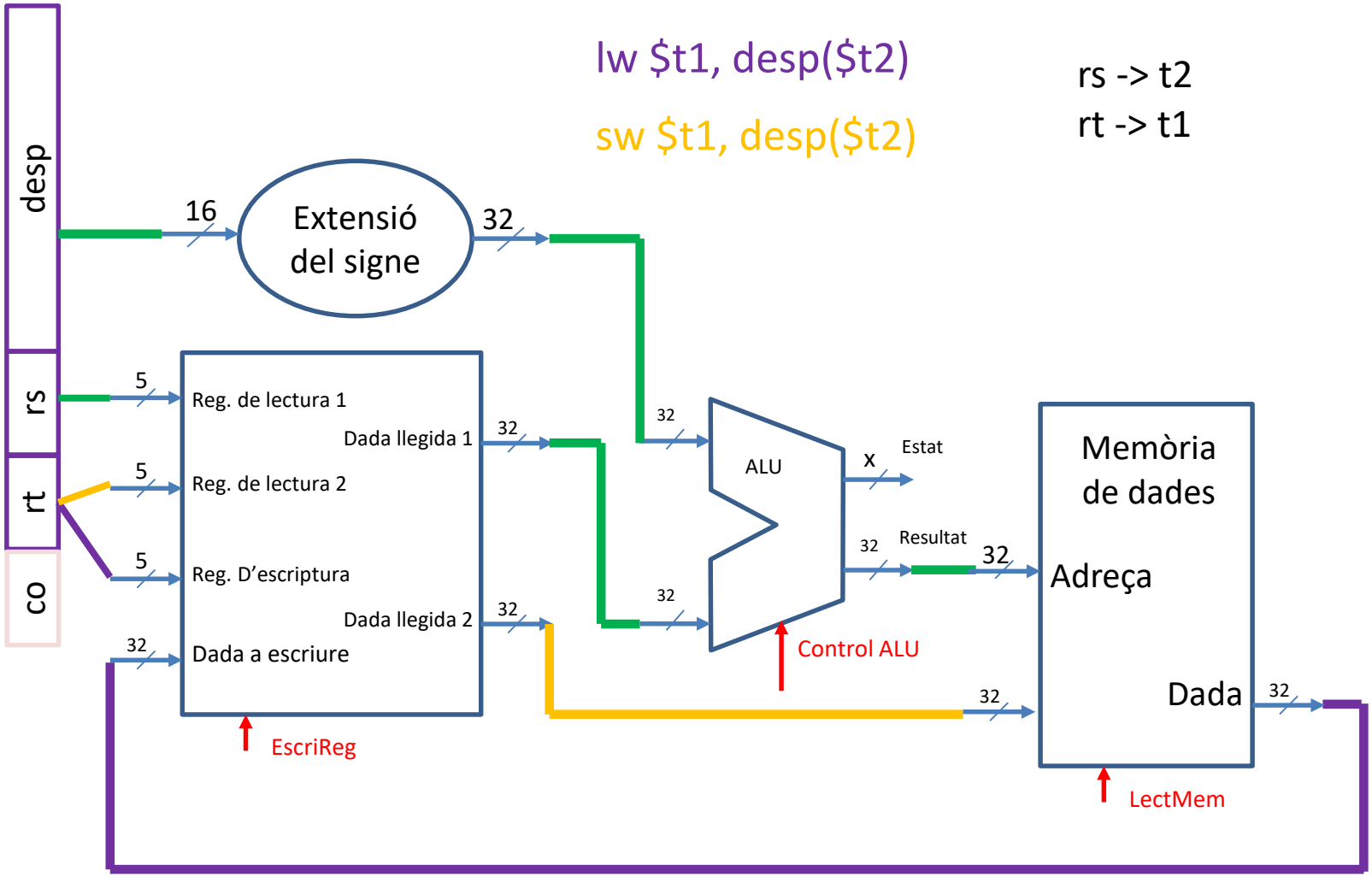
lw $t1,desp($t2) -> $t1 = Mem($t2+desp)

sw $t1,desp($t2) -> Mem($t2+desp) = $t2

Es necessita:

- Un sumador (ALU) per calcular l'adreça de la memòria
- Al haver de fer una suma del contingut d'un registre de 32 bits amb un desplaçament en C'2 de 16 bits, es necessita una unitat per estendre el signe.
- Una unitat de memòria per llegir o guardar la dada de 32 bits

MIPS: Disseny del camí de dades. Tipus I

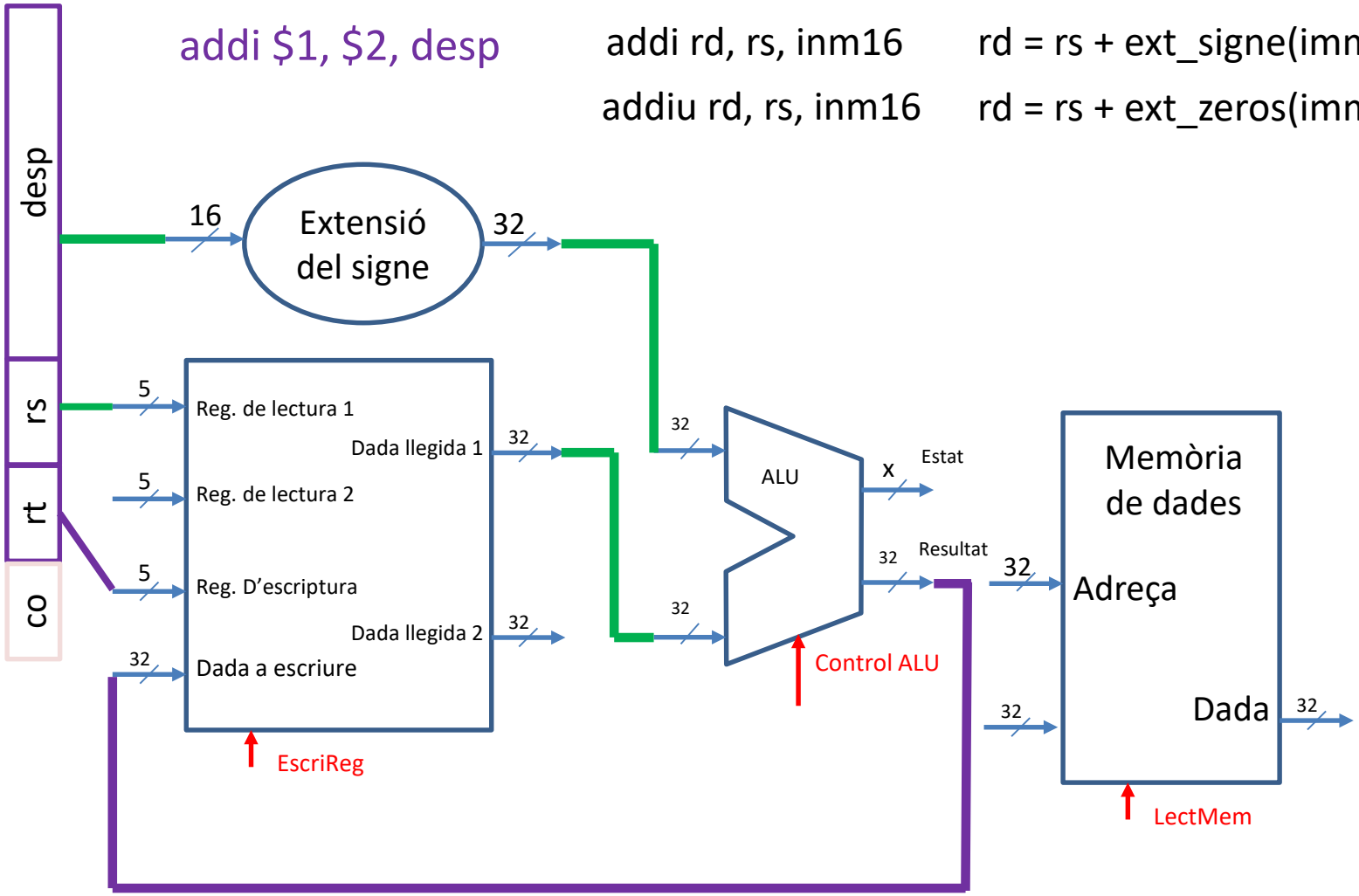


MIPS: Disseny del camí de dades

addi $1, $2, desp

addi rd, rs, inm16 rd = rs + ext_signe(imm16, 32)

addiu rd, rs, inm16 rd = rs + ext_zeros(imm16, 32)



MIPS: Disseny del camí de dades

Instruccions del MIPS de salt condicional:

beq $t1,$t2,desp si (t1 = t2) → PC = PC + desp * 4;
sinó PC = PC + 4

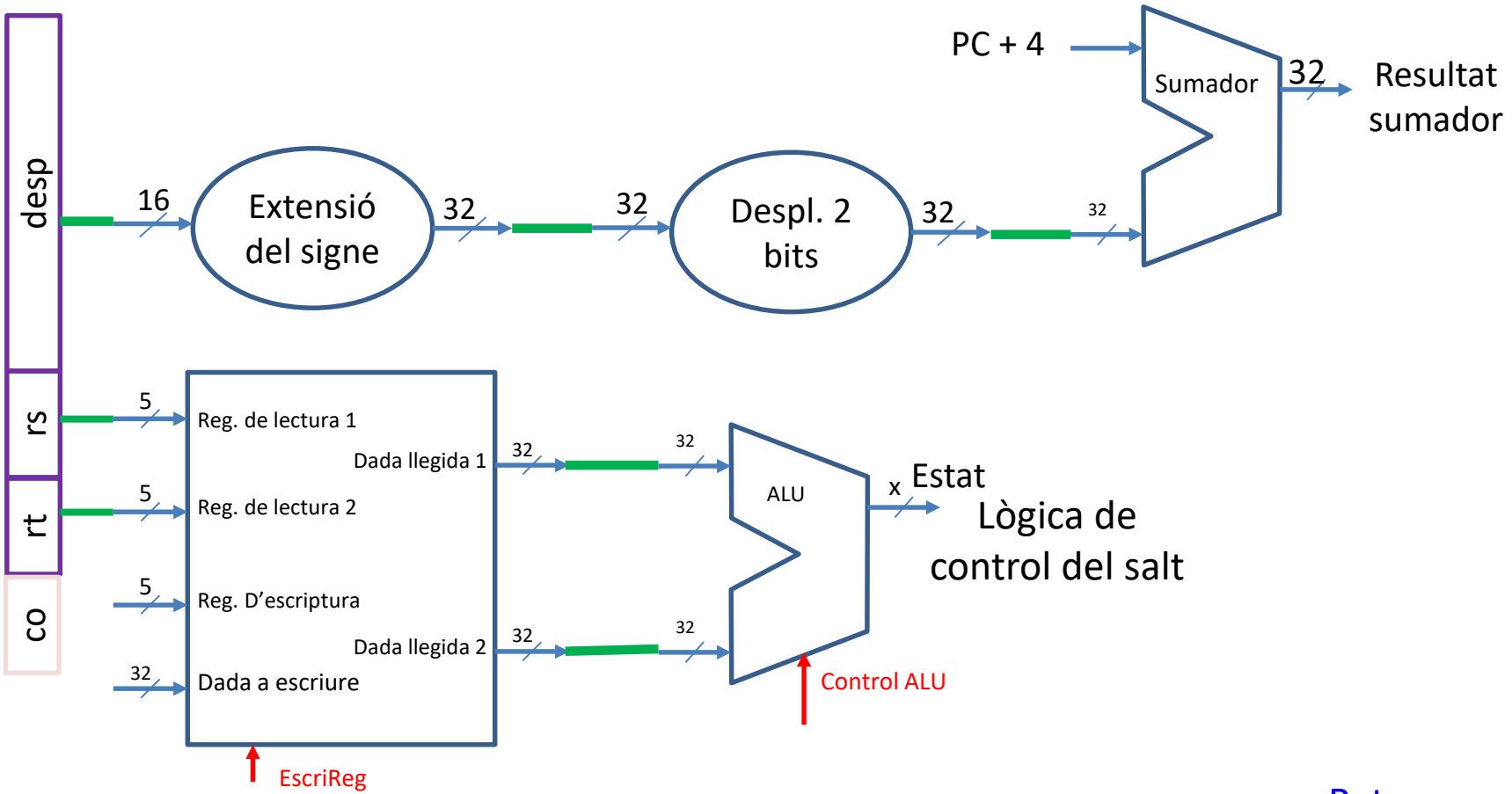
Per realitzar aquesta instrucció s'ha de calcular l'adreça destí del salt, sumant el PC amb el camp del desplaçament amb el signe estès.
Detalls del salt condicional:

- L'arquitectura del repertori d'instruccions especifica que és l'adreça de la següent instrucció en ordre seqüencial la que s'utilitza com a base per al càlcul de l'adreça de destí. Ja què es calcula el PC + 4 (l'adreça de la pròxima instrucció) en el camí de dades de la càrrega d'instruccions.
- L'arquitectura també imposa que el camp de desplaçament es desplaci cap a l'esquerra 2 bits per a què aquest es correspongui a una paraula (32 bits); Queda multiplicat per un factor de 4.

A més a més de calcular l'adreça destí del salt, també s'ha de determinar si la següent instrucció a executar és la que segueix seqüencialment o la situada a l'adreça destí del salt. Si es compleix la condició (els operands són iguals), l'adreça calculada passa a ser el nou valor del PC, i es diu que el alt condicional s'ha complert. Si els operands són diferents, el PC incrementat ha de substituir al PC actual; en aquest cas es diu que el salt no s'ha complert.

MIPS: Disseny del camí de dades

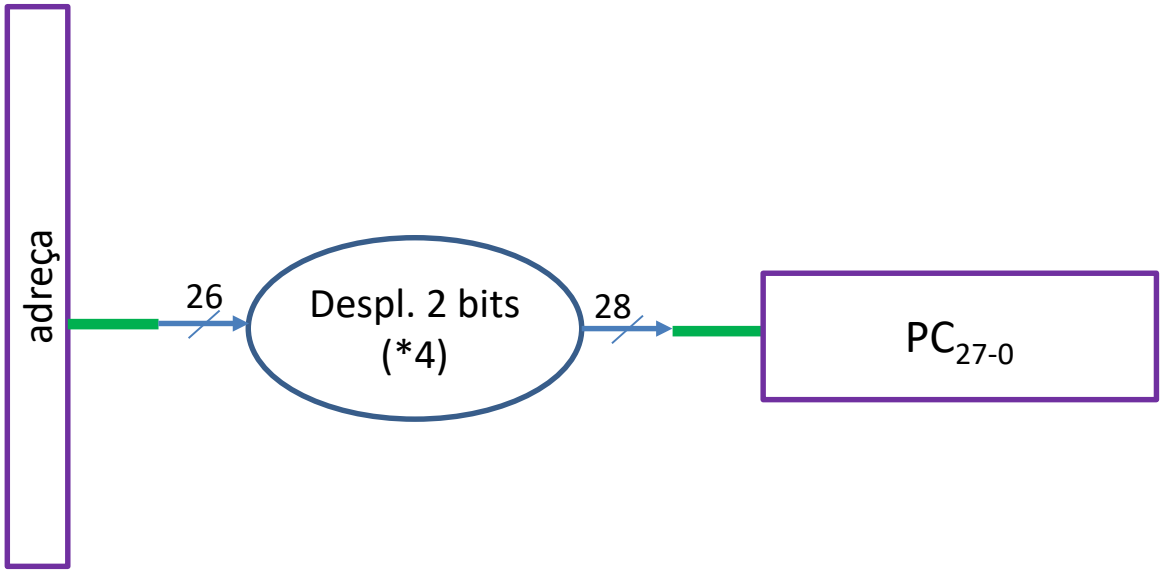
beq rt,rs,desp si (rt = rs) → PC = PC + desp * 4;
si (rt ≠ rs) → PC = PC + 4



- Retorn

MIPS: Disseny del camí de dades

Salt incondicional: j adreça



Els 4 bits més significatius del PC no es modifiquen

- Retorn

MIPS: Disseny del camí de dades (maquina monocicle)

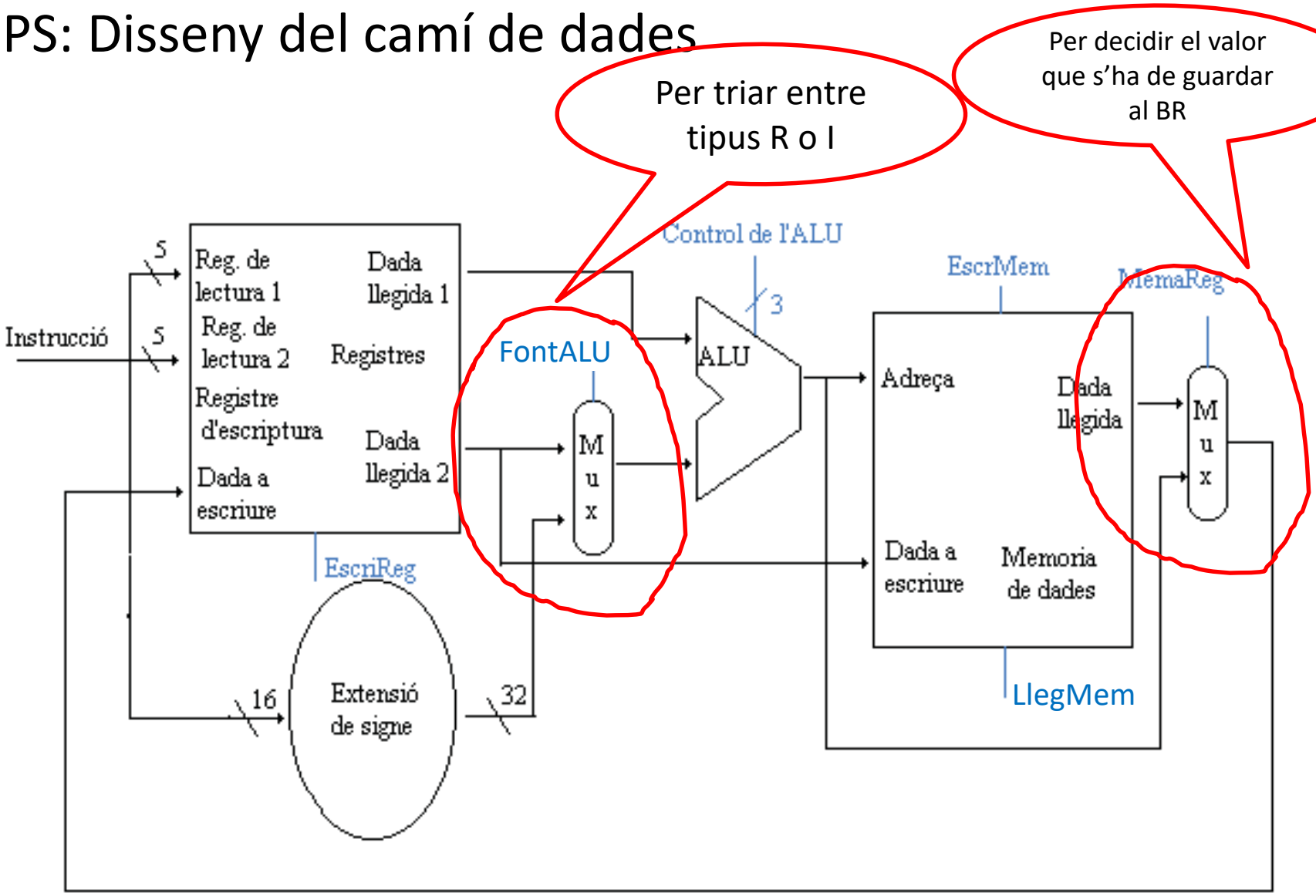
Cap element del camí de dades pot utilitzar-se més d'una vegada per instrucció, de manera que qualsevol recurs que es necessiti més d'una vegada haurà de repetir-se. Per tant, la memòria d'instruccions ha d'estar separada de la memòria de dades. Encara que es necessiti duplicar algunes de les unitats funcionals, molts d'aquests elements poden compartir-se en els diferents fluxos d'instruccions quan els camins de dades individuals de les figures anteriors es combinen.

Per compartir un element del camí de dades entre dues classes d'instruccions diferents, es requereix que l'element disposi de múltiples entrades, i un senyal de control que seleccioni l'adequada en cada instant (un **multiplexor**)

El camí de dades de les instruccions aritmètic - lògiques (o tipus R) i el de les instruccions d'accés a la memòria, són molt similars. Les principals diferències son:

- La segona entrada de l'ALU és, o bé un registre en el cas d'una instrucció aritmètica o lògica, o bé els bits de menor pes d'una instrucció de memòria amb el signe estès.
- El valor emmagatzemat al registre destí, prové de l'ALU (per a instruccions de tipus R) o de la memòria (en cas de load).

MIPS: Disseny del camí de dades

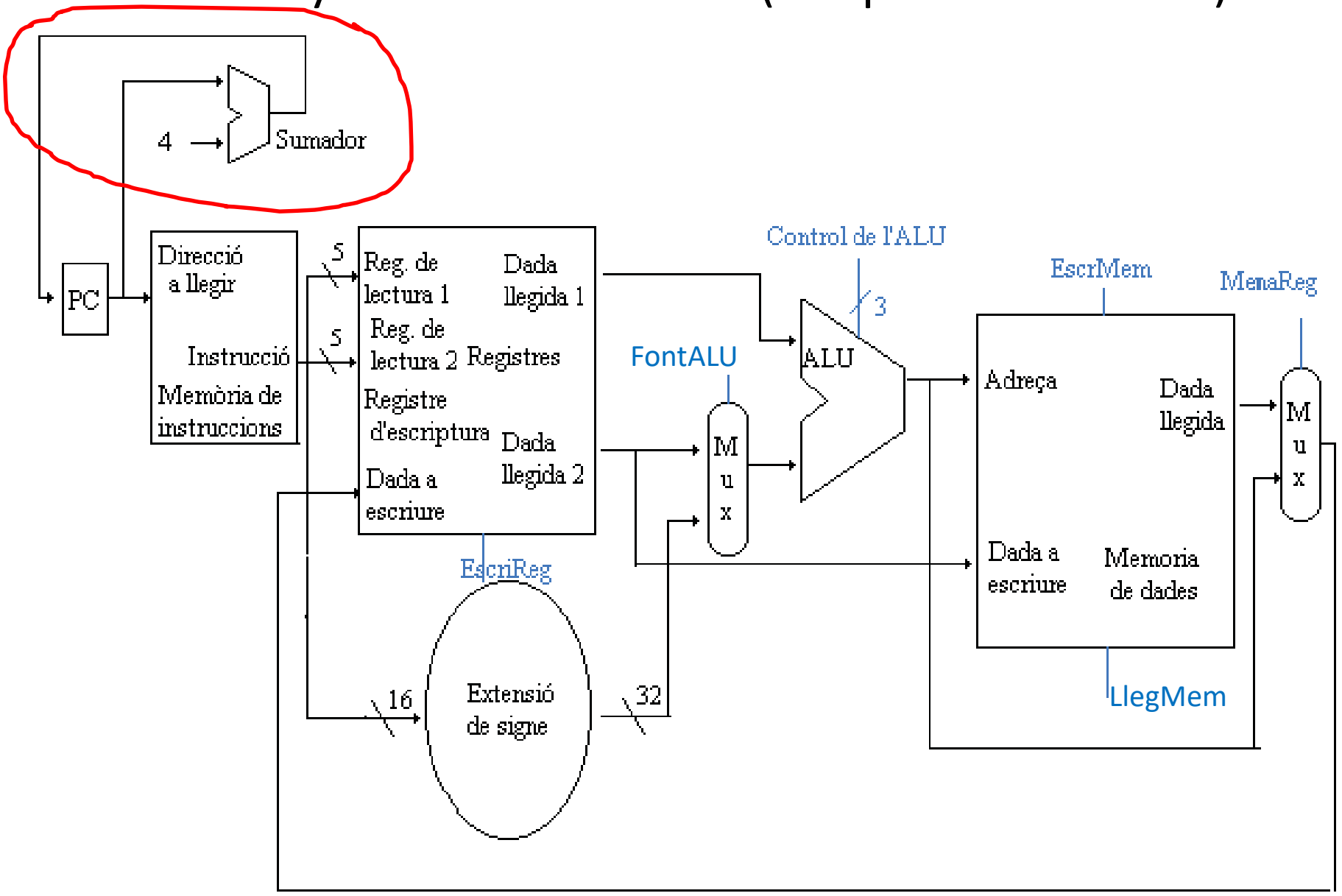


Instruccions del tipus I i R

MIPS: Disseny del camí de dades (maquina monocicle)

La part del camí de dades encarregada de la cerca d'instruccions, pot afegir-se fàcilment a aquest nou camí de dades. Aquest nou camí de dades té memòries separades per a dades i per a instruccions, així com un sumador independent de l'ALU, utilitzat per a incrementar el PC mentre que l'ALU executa la instrucció en el mateix cicle del rellotge.

MIPS: Disseny del camí de dades (maquina monocicle)

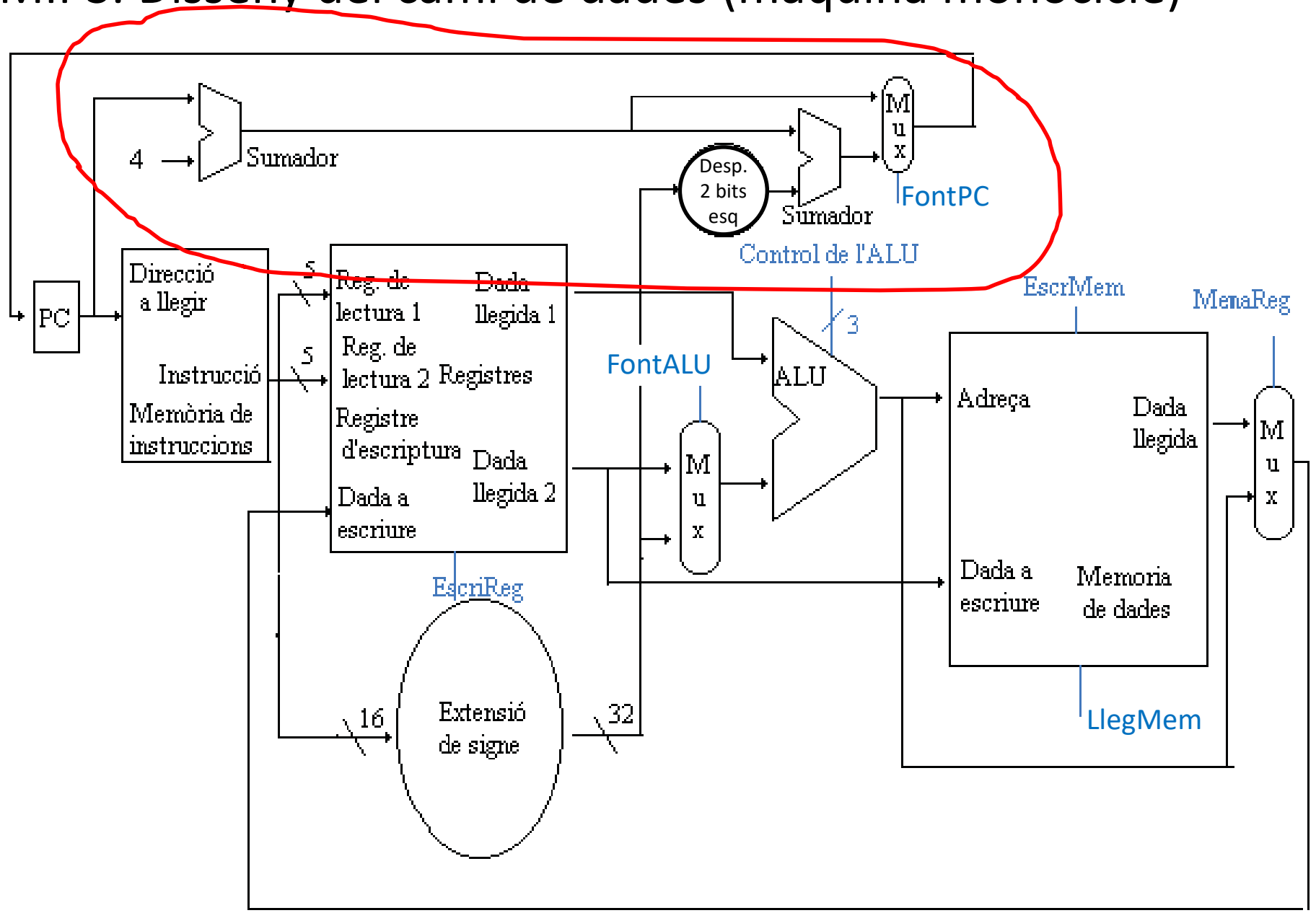


MIPS: Disseny del camí de dades (maquina monocicle)

També poden combinar-se la resta de les peces per a obtenir un camí de dades únic per a l'arquitectura MIPS, afegint la part encarregada de l'execució de les instruccions de salt.

Les instruccions de salt utilitzen l'ALU per a la comparació dels registres font, s'ha d'afegir un sumador per a calcular l'adreça de destí del salt. També és necessari un multiplexor per escollir entre seguir en seqüència (PC + 4) i l'adreça destí del salt per a escriure aquesta nova adreça al PC.

MIPS: Disseny del camí de dades (maquina monocicle)

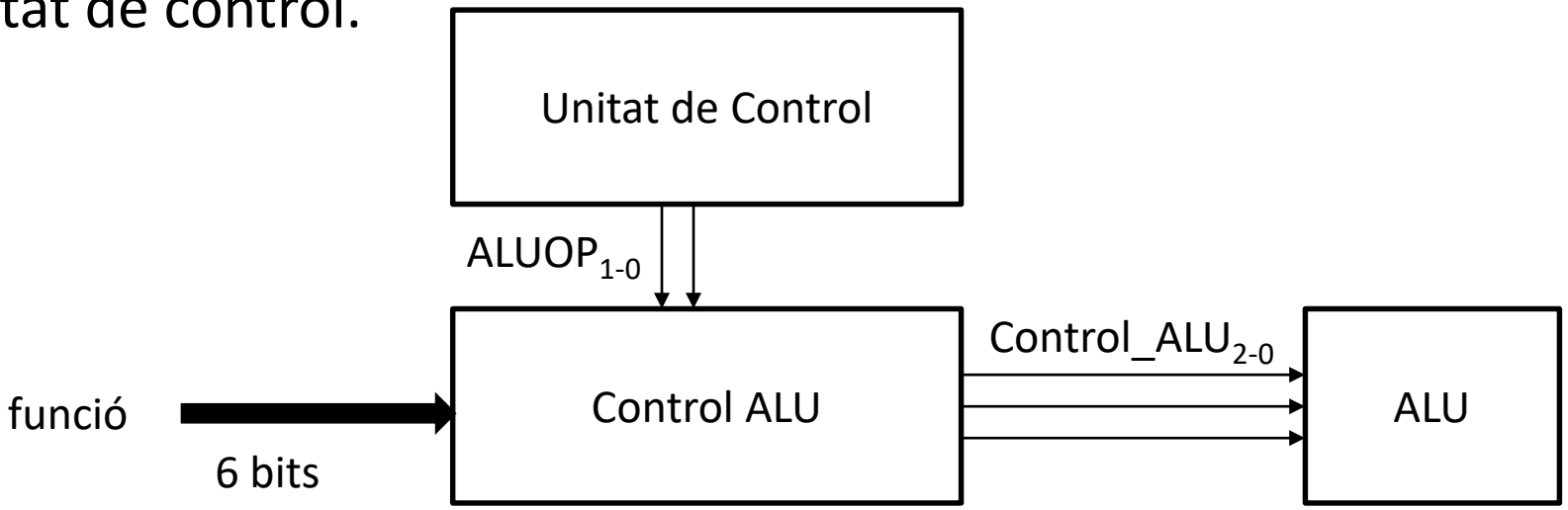


MIPS: Disseny del camí de dades (maquina monocicle)

Només falta afegir la unitat de control.

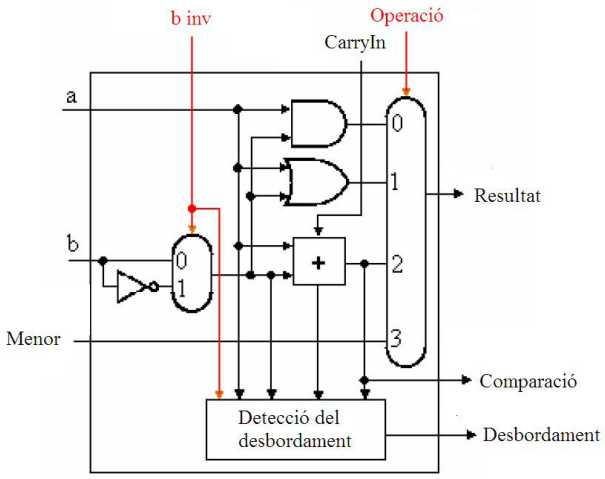
Aquesta ha de generar els senyals d'escriptura per a cadascun dels elements d'estat i les de control per a l'ALU i per a cadascun dels multiplexors a partir de les entrades.

Ja què el control de l'ALU és diferent de la resta en major o menor mesura, va molt bé dissenyar-lo com a pas previ al disseny de la unitat de control.

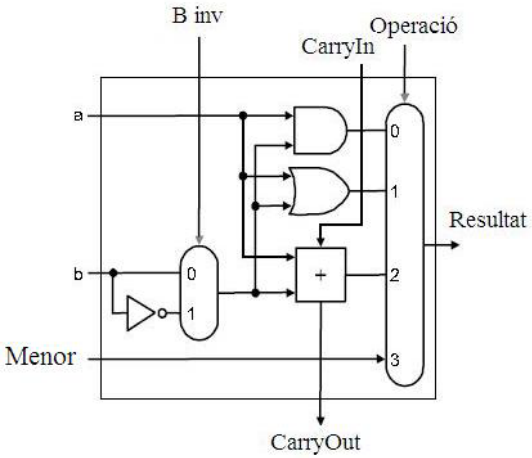


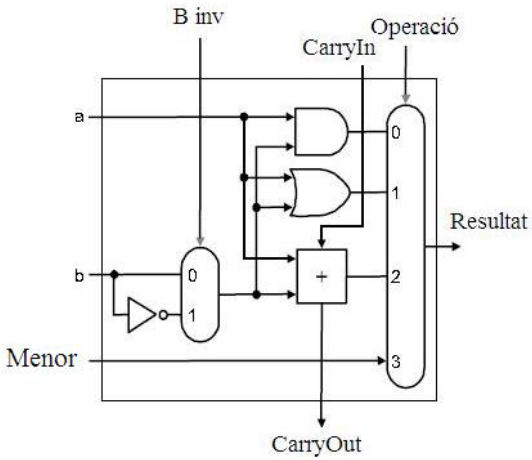
MIPS: Disseny del camí de dades (maquina monocicle)

Pel bit 31

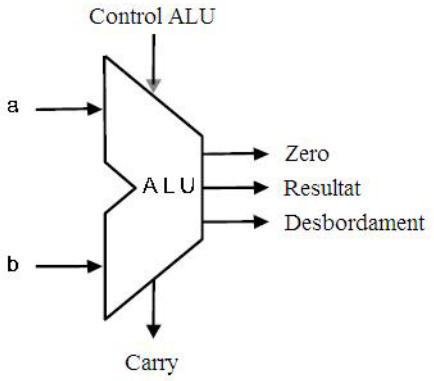


Pel bit 30 ..... Pel bit 0





Entrades de control de l'ALU (B inv, Op1, Op0)	Funció
0 0 0	AND
0 0 1	OR
0 1 0	Suma
1 1 0	Resta
1 1 1	Activar si menor que

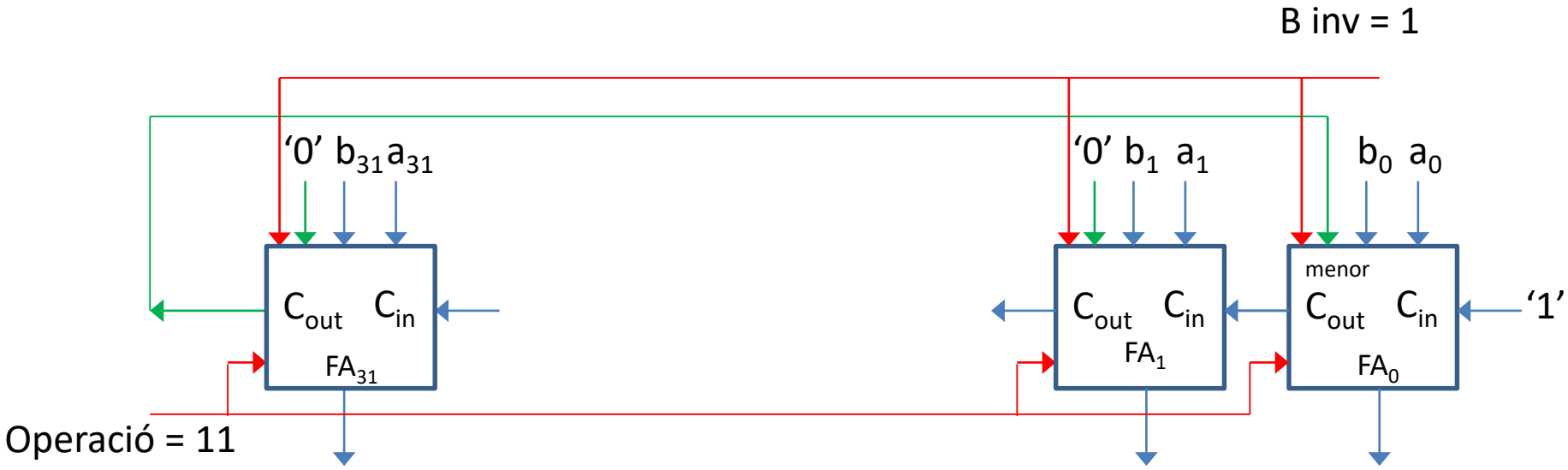


MIPS: Disseny del camí de dades (maquina monocicle)

Comparació: slt rd, rs, rt

Activa si és menor:
- Si rs < rt: rd = 1
- En cas contrari: rd = 0

Implementació: Recollir el signe de a-b (C_{out} del FA_{31})
Resultat: [0000....000C_{out}]



MIPS: Disseny del camí de dades (maquina monocicle)

L'ALU té tres línies de control com a entrades. Només l'utilitzarem cinc de les vuit combinacions possibles.
Les cinc possibles combinacions, son:

Entrades de control de l'ALU	Funció
0 0 0	AND
0 0 1	OR
0 1 0	Suma
1 1 0	Resta
1 1 1	Activar si menor que

Depenent del tipus d'instrucció a executar, l'ALU ha de realitzar una d'aquestes cinc operacions.

- Les instruccions d'accés a la memòria utilitzen l'ALU per a calcular l'adreça de la memòria mitjançant una suma.
- Per a les instruccions de tipus R, l'ALU ha d'executar una de les cinc operacions (AND, OR, add, sub i slt) en funció del valor dels 6 bits de menor pes de la instrucció, els quals composen el codi de funció,
- Les instruccions de saltar si igual, l'ALU ha de realitzar una resta.

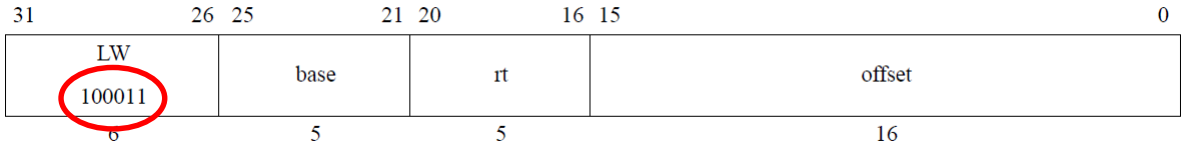
MIPS: Disseny del camí de dades (maquina monocicle)

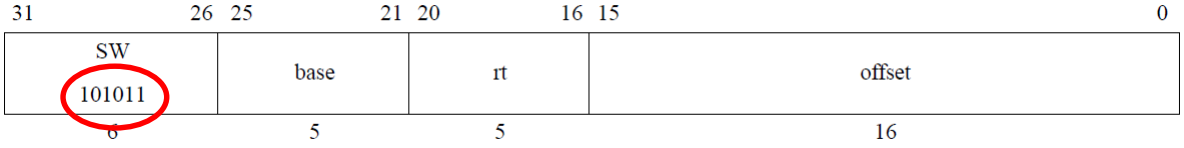
El control de l'ALU

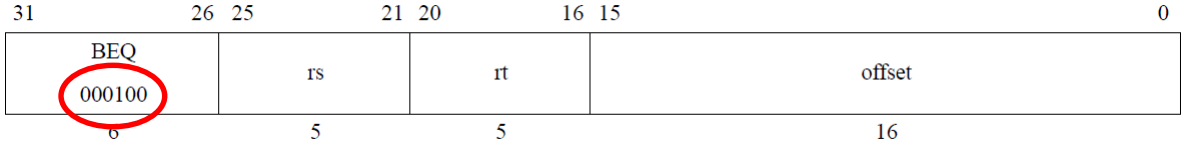
Depenent del tipus d'instrucció a executar, l'ALU ha de realitzar una d'aquestes cinc operacions. Les instruccions d'accés a la memòria utilitzen l'ALU per a calcular l'adreça de la memòria mitjançant una suma. Per a les instruccions de tipus R, l'ALU ha d'executar una de les cinc operacions (AND, OR, add, sub i slt) en funció del valor dels 6 bits de menor pes de la instrucció, els quals composen el codi de funció, mentre que per a les instruccions de saltar si igual, l'ALU ha de realitzar una resta.

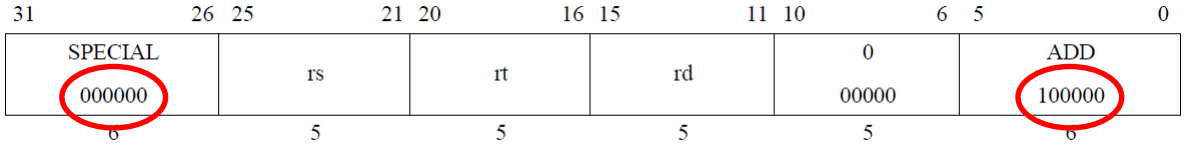
C.O instrucció	ALUOP	Operació	Camp de funció	Acció ALU	Control de l'ALU
lw	00	Llegir paraula	xxxxxx	Sumar	010
sw	00	Escriure paraula	xxxxxx	Sumar	010
Saltar si igual	01	Saltar si igual	xxxxxx	Restar	110
Tipus R	10	Suma	100000	Sumar	010
Tipus R	10	Resta	100010	Restar	110
Tipus R	10	AND	100100	AND	000
Tipus R	10	OR	100101	OR	001
Tipus R	10	Activar si menor que	101010	Activa si <	111

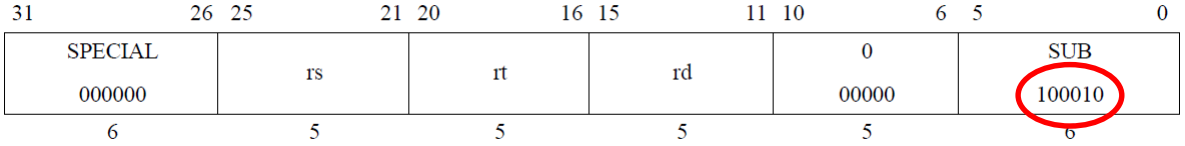
Instruccions

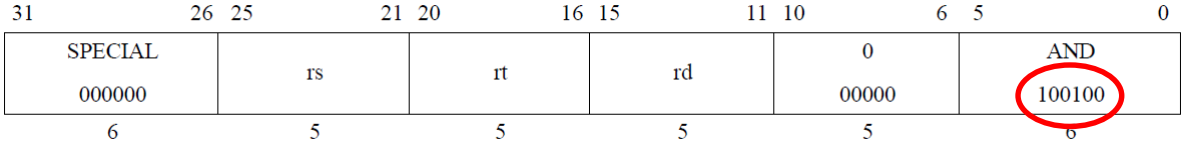


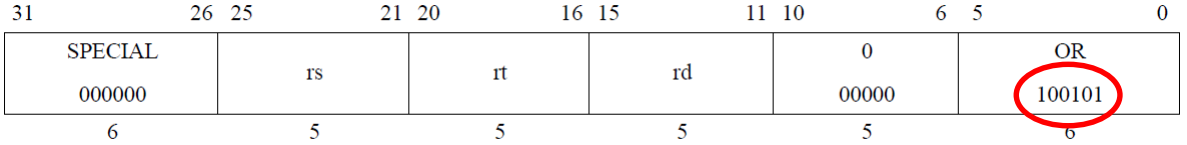


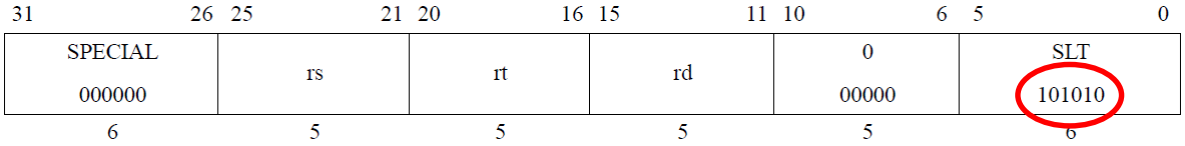












MIPS: Disseny del camí de dades (maquina monocicle)

Mitjançant una petita unitat de control que té com a entrades el codi de funció de la instrucció (Camp de funció) i dos bits de control addicionals (ALUOp), es poden generar els tres bits que configuren els senyals de control de l'ALU.
Aquests bits indiquen si l'operació a realitzar, ha de ser:

- Una suma (00) pels accessos a la memòria,
- Una resta (01) pels salts condicionals,
- Si l'operació a realitzar està codificada al codi de funció (10).

La sortida de la unitat de control de l'ALU és un senyal de tres bits, que controla l'ALU codificant una de les cinc combinacions.

Es pot observar el conjunt de combinacions dels senyals d'entrada formades pels dos bits que conformen el senyal d'ALUOp i els 6 bits del codi de funció.
Finalment, també es mostra la relació existent entre els codis d'operació de les instruccions i el camp ALUOp.

MIPS: Disseny del camí de dades (maquina monocicle)

Existeixen diferents maneres d'establir la correspondència entre els dos bits del camp de ALUOp i els 6 bits del codi de funció amb els 3 bits que conformen l'operació a realitzar per l'ALU. Com a conseqüència que només una petita part dels 64 (2^6) valors possibles del codi de funció són importants i que el camp de funció només s'utilitza quan ALUOp val 10, podria utilitzar-se una petita part de lògica que reconegués aquest subconjunt de valors possibles i donés com a resultat els valors correctes dels bits de control de l'ALU.

Com a pas previ al disseny de la lògica combinatòria, és útil construir una taula de veritat per a aquelles combinacions d'interès dels codis de funció i dels bits d'ALUOp.

ALUOp		**Camp de la funció**						**Operació**
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

MIPS: Disseny del camí de dades (maquina monocicle)

ALUOp		Camp de la funció						Operació
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

Els genera el controlador

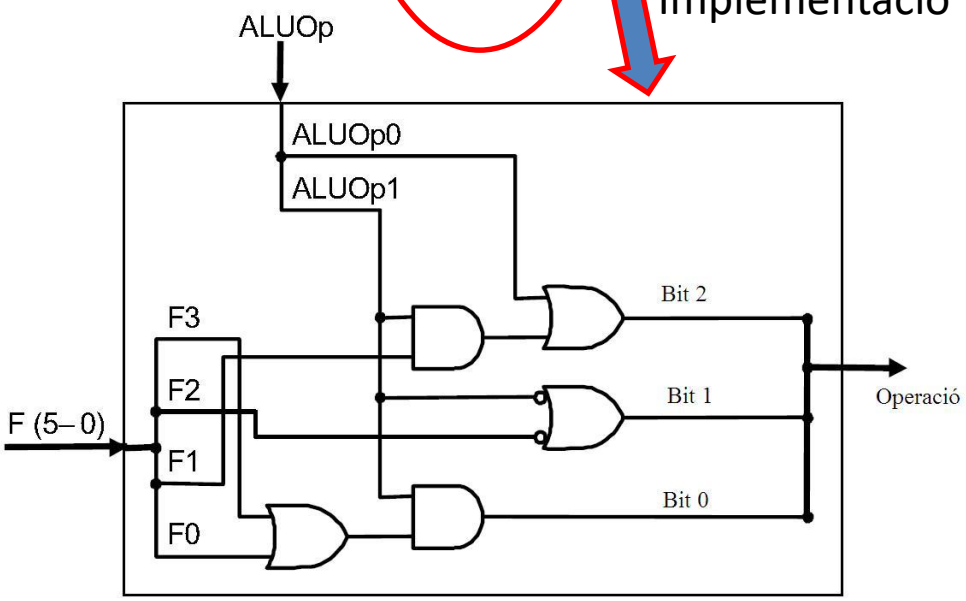
Sempre valen el mateix (10)

Implementació

$Bit_2 = ALUOP_0 + ALUOP_1 * F1$

$Bit_1 = \overline{ALUOP_1} + \overline{F2}$

$Bit_0 = ALUOP_1 * (F3 + F0)$



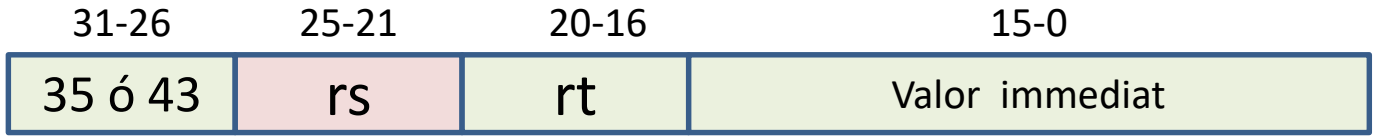
MIPS: Disseny del camí de dades (maquina monocicle)

- Disseny de la unitat de control principal

S'han d'identificar els camps de les instruccions i les línies de control necessàries per a la construcció del camí de dades.
Els formats dels tres tipus d'instruccions son:

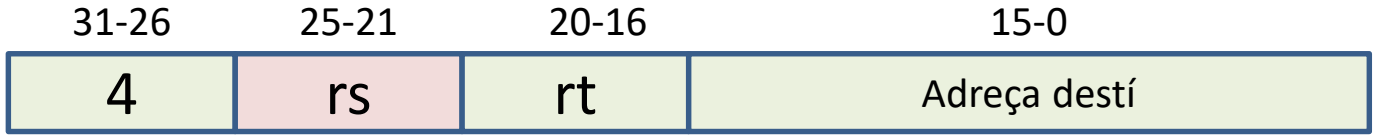
Format R:

Format I:

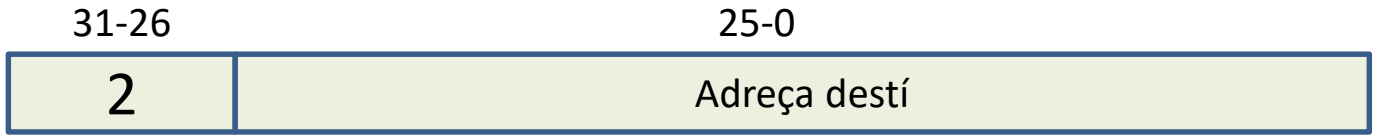


Load 100011(35) i Store 101011 (43)

Salt condicional



Salt incondicional

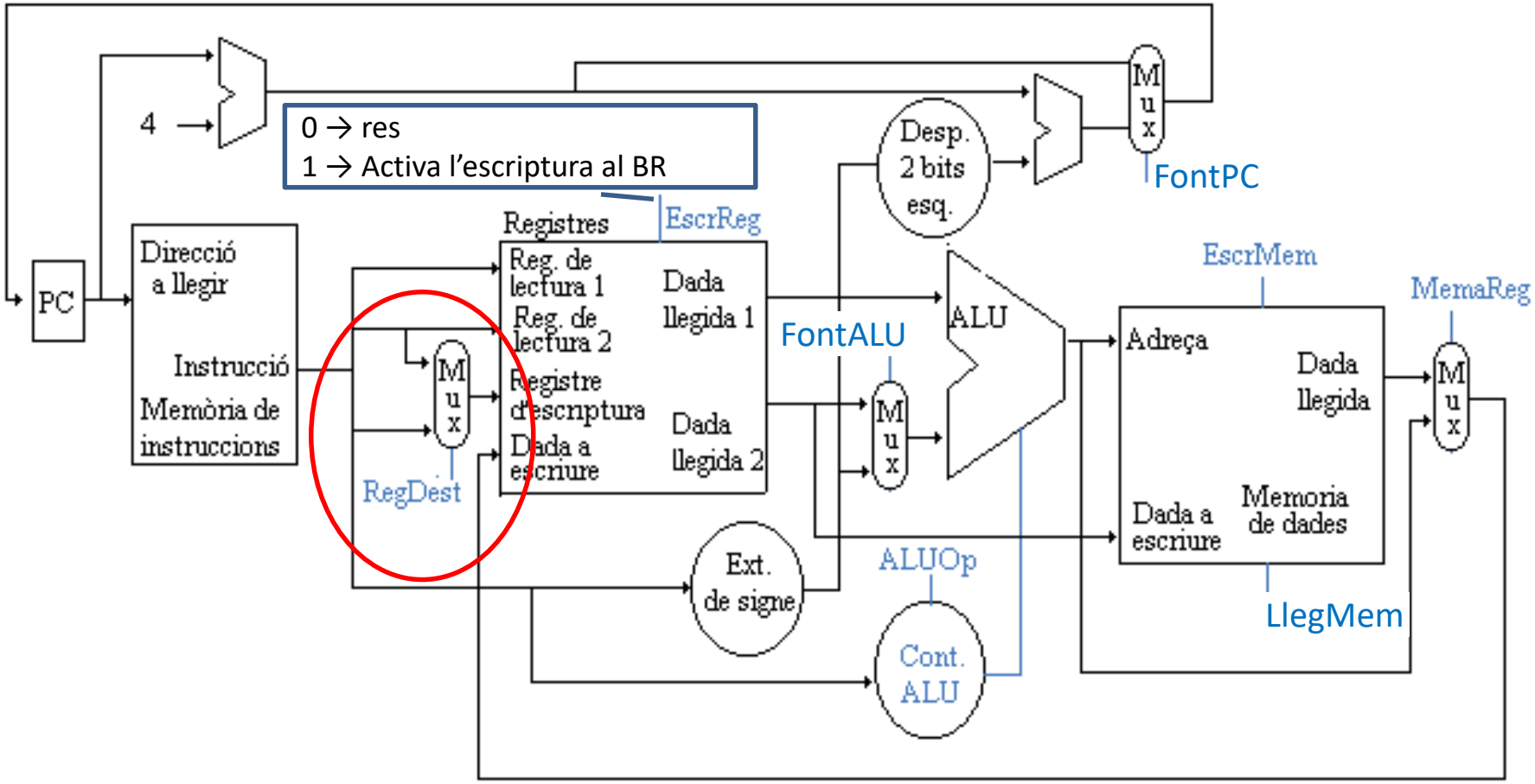


MIPS: Disseny del camí de dades (maquina monocicle)

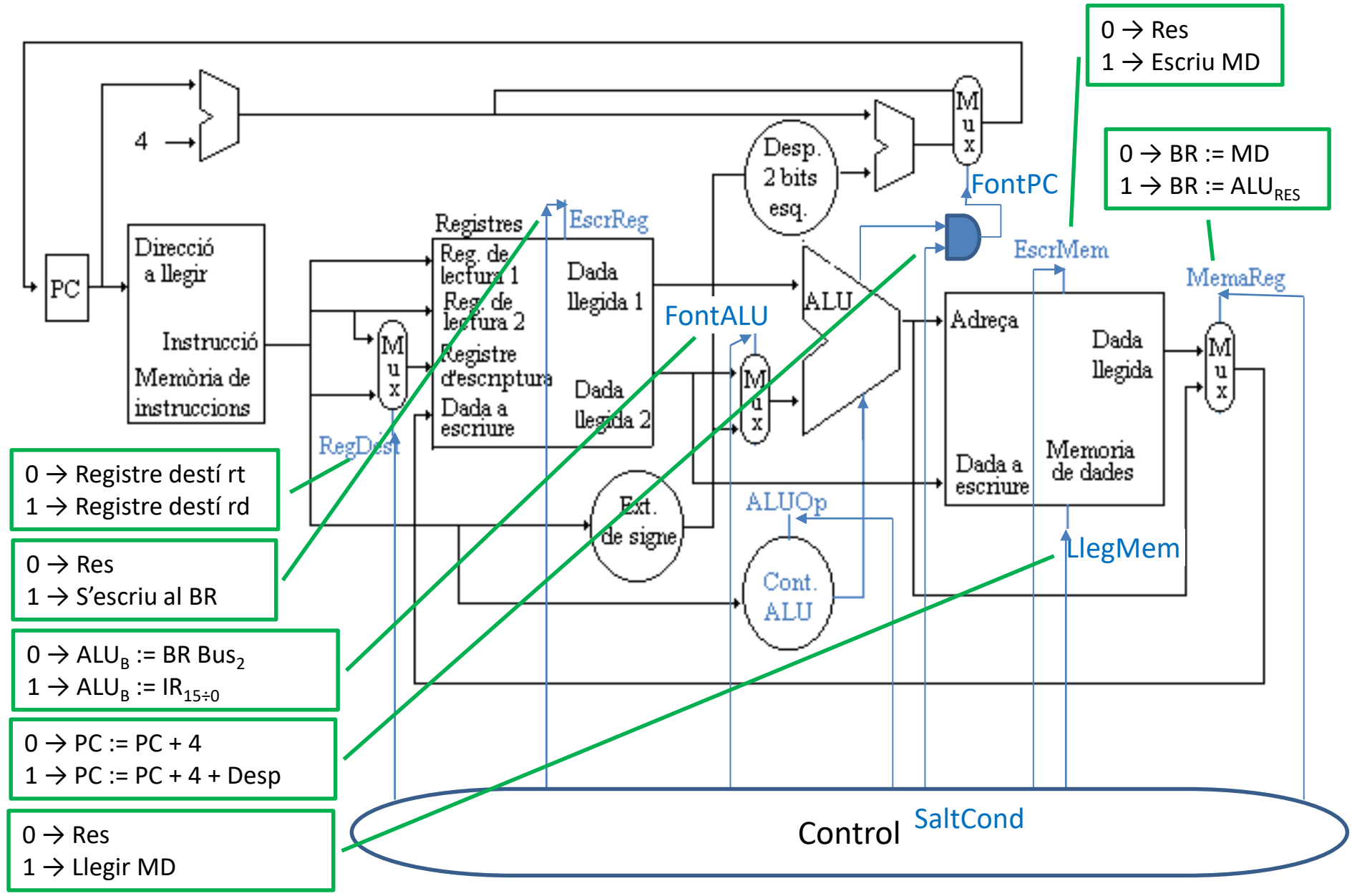
Observacions del format de les instruccions:

- El camp del tipus d'operació sempre és als bits 31-26. →Op[5-0].
- Els dos registres de lectura sempre s'especifiquen als camps rs i rt als bits 25-21 i 20-16 (instruccions de tipus R, I i de salt condicional).
- El registre base per a les instruccions d'accés a memòria es troba sempre a rs (bits 25-21).
- El desplaçament relatiu de 16 bits per a saltar si és igual (beq), load i store, és sempre als bits 15-0.
- El registre destí pot ser a dos llocs. Per les instruccions de load, als bits 20-16 (rt), mentre que en les instruccions aritmètiques i lògiques, es troba als bits 15-11 (rd). Implica haver d'afegir un multiplexor per a seleccionar quin d'aquests dos camps de la instrucció s'ha d'utilitzar per indicar el registre on escriure.

MIPS: Disseny del camí de dades (maquina monocicle)



MIPS: Disseny del camí de dades (maquina monocicle)



MIPS: Disseny del camí de dades (maquina monocicle)

Funcionalitat de les línies de control

Senyal de control	Efecte (no actiu) (0)	Efecte (actiu) (1)
RegDest	L'identificador del registre destí ve determinat pel camp rt (bits 20-16)	L'identificador del registre destí ve determinat pel camp rd (bits 15-11)
EscrReg	Cap	S'actualitza el registre destí
FontALU	El segon operant de l'ALU prové del segon registre llegit del BR	El segon operant de l'ALU són els 16 bits de menys pes de la instrucció amb el signe estès (valor immediat)
SaltCond	PC = PC + 4	PC = la sortida del sumador, que calcula l'adreça destí dels salt.
LlegMem	Cap	El contingut adreçat de la memòria es col·loca a la sortida de lectura
SaltIncod	PC = PC + 4	PC = Adreça absoluta
EscrMem	Cap	El contingut adreçat de la memòria es substitueix pel valor de l'entrada
MemaReg	El valor d'entrada del banc de registres prové de l'ALU	El valor d'entrada del banc de registres prové de la memòria

MIPS: Disseny del camí de dades (maquina monocicle)

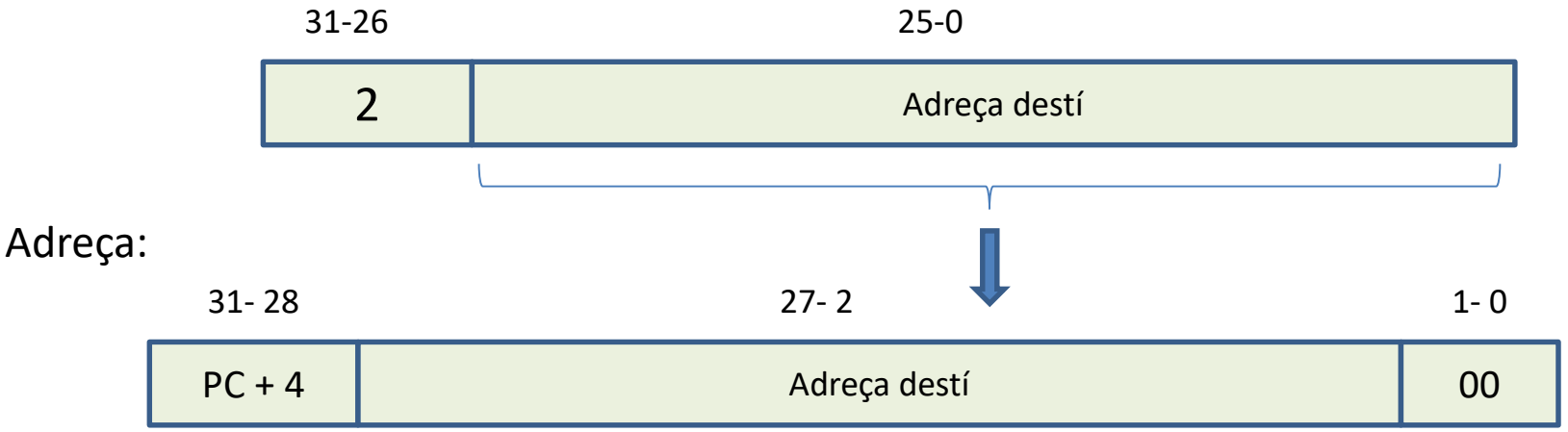
Abans d'intentar escriure el conjunt d'equacions de la taula de veritat de la unitat de control seria útil definir-la informalment.
Com que l'activació de les línies de control únicament depèn del codi d'operació, es defineix si cadascun dels senyals de control ha de valdre 0, 1 o bé indeterminat (X), per a cadascun dels codis d'operació.
L'activació dels senyals de control per a cada codi d'operació, pot ser

Instrucció	RegDest	FontALU	MemaReg	EscReg	LlegMem	EscrMem	SaltCond	ALUOp1	ALUOP0
Format-R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

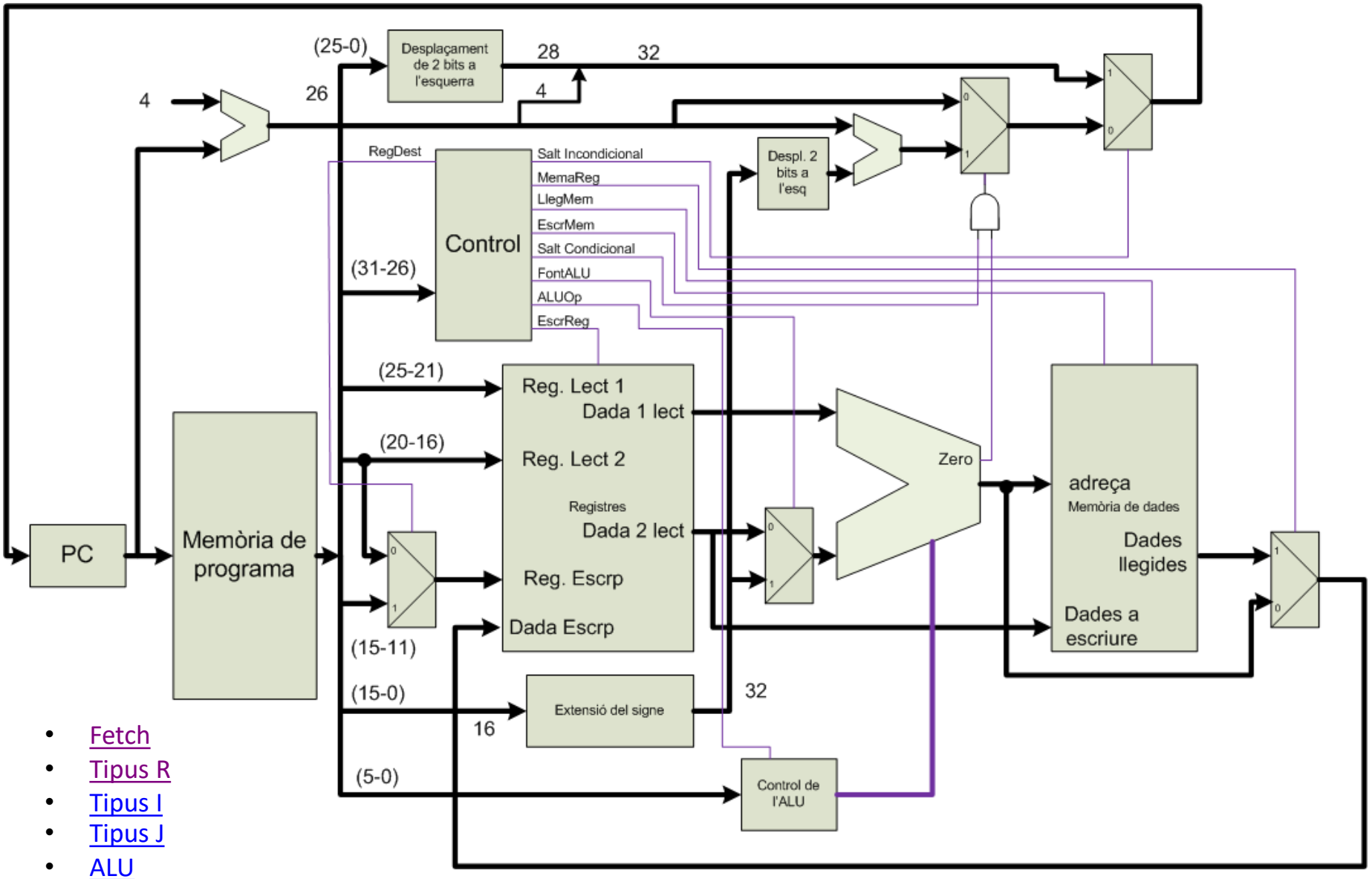
MIPS: Disseny del camí de dades (maquina monocicle)

Per donar una mica de complexitat al simulador, s'ha afegit la instrucció "j" (salt), la qual té un certa semblança amb la instrucció de salt condicional, però es calcula l'adreça destí d'una manera diferent, i a més, és incondicional. Com els salts, els dos bits de menys pes de l'adreça del salt són sempre 00. Els següents 26 bits de menor pes de l'adreça es troben en el camp immediat de la instrucció. Els 4 bits de més pes de l'adreça que haurien de substituir el PC venen del PC de la instrucció al qual s'ha sumat el valor 4. Així es podria realitzar un "*jump*" emmagatzemat al registre de PC la concatenació de:

- Els 4 bits de més pes de l'actual PC + 4 (són els bits 31-28 de l'adreça de la següent instrucció en ordre seqüencial).
- Els 26 bits corresponents al camp immediat de la instrucció *jump*.
- Els bits de menys pes: 00.



MIPS: maquina monocicle



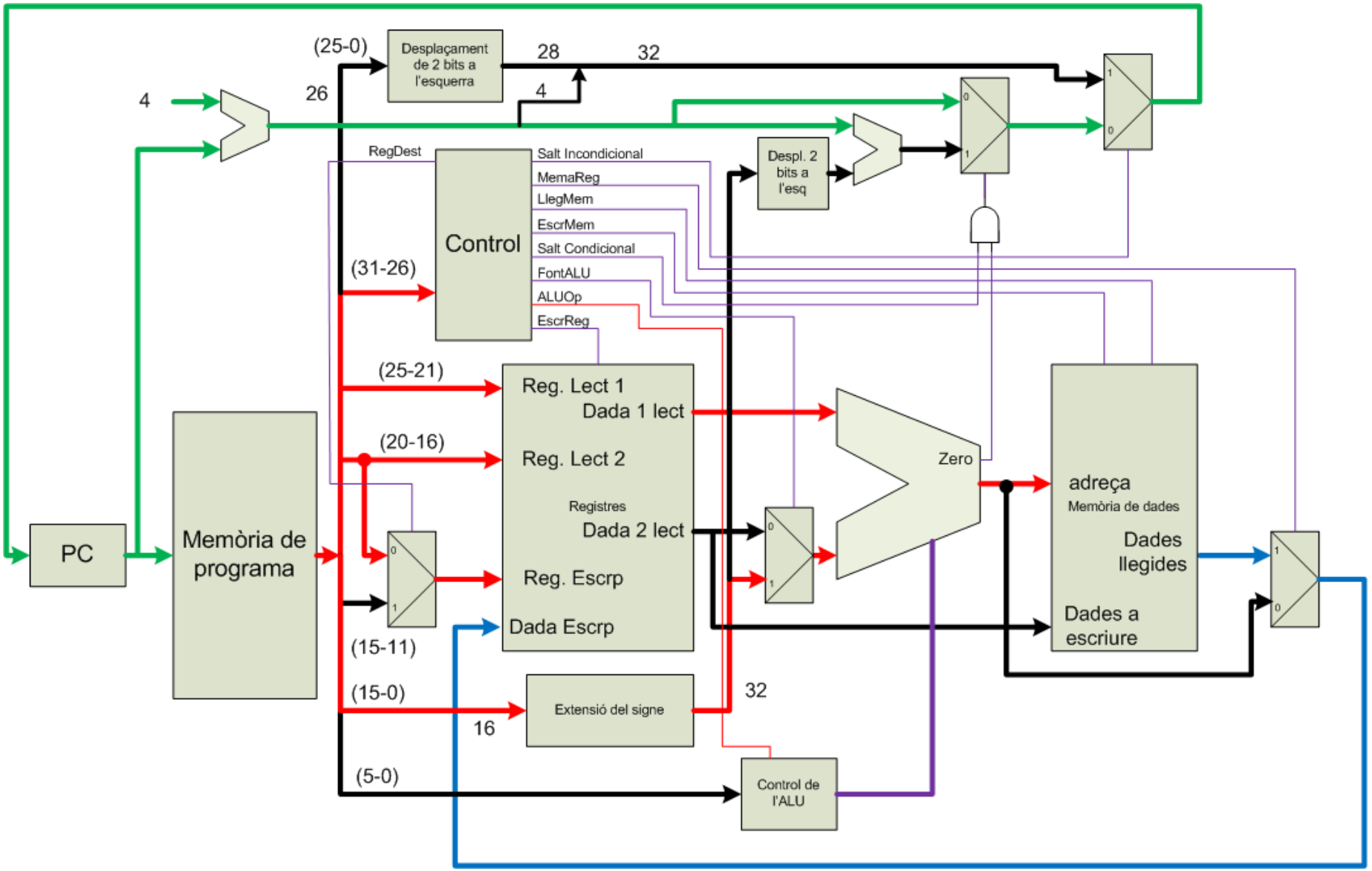
- Fetch
- Tipus R
- Tipus I
- Tipus J
- ALU

Control de la màquina

	Senyal	Format-R	lw	sw	beq	j
Entrades	Op5	0	1	1	0	0
	Op4	0	0	0	0	0
	Op3	0	0	1	0	0
	Op2	0	0	0	1	0
	Op1	0	1	1	0	1
	Op0	0	1	1	0	0
	Op_{5-0}	000000	100011	101011	000100	000010
Sortides	RegDest	1	0	X	X	0
	FontALU	0	1	1	0	0
	MemaReg	0	1	X	X	0
	EscrReg	1	1	0	0	0
	LlegMem	0	1	0	0	0
	EscrMem	0	0	1	0	0
	Salt Condicional	0	0	0	1	0
	ALUOp1	1	0	0	0	0
	ALUOp0	0	0	0	1	0
	Salt Incondicional	0	0	0	0	1

lw $t1, desplaçament($t2)

Fetch Operació Resultat

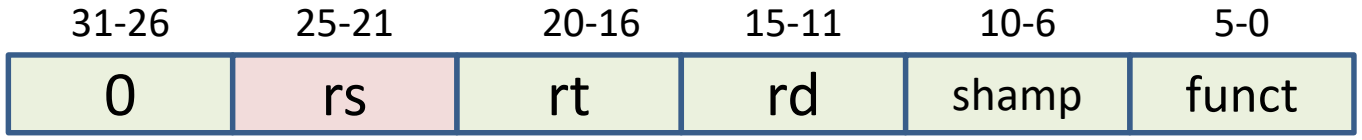


MIPS: Disseny del camí de dades (maquina monocicle)
Disseny de la unitat de control principal

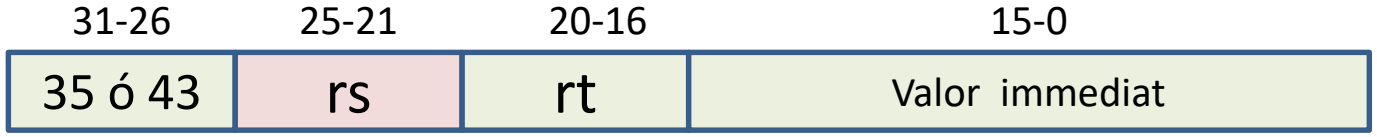
S'han d'identificar els camps de les instruccions i les línies de control necessàries per a la construcció del camí de dades.

Els formats dels tres tipus d'instruccions son:

Format R:

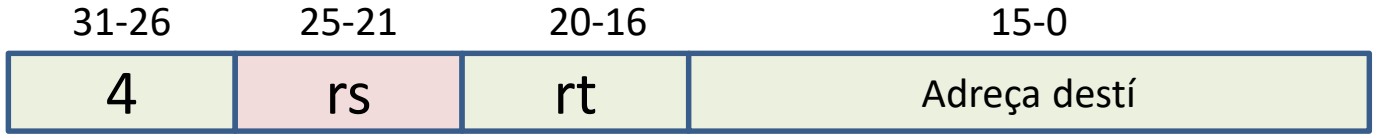


Format I:

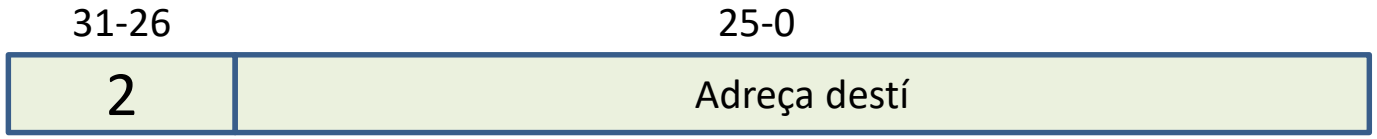


Load 100011_b 23_h (35_d) i Store 101011_b $2b_h$ (43_d)

Salt condicional



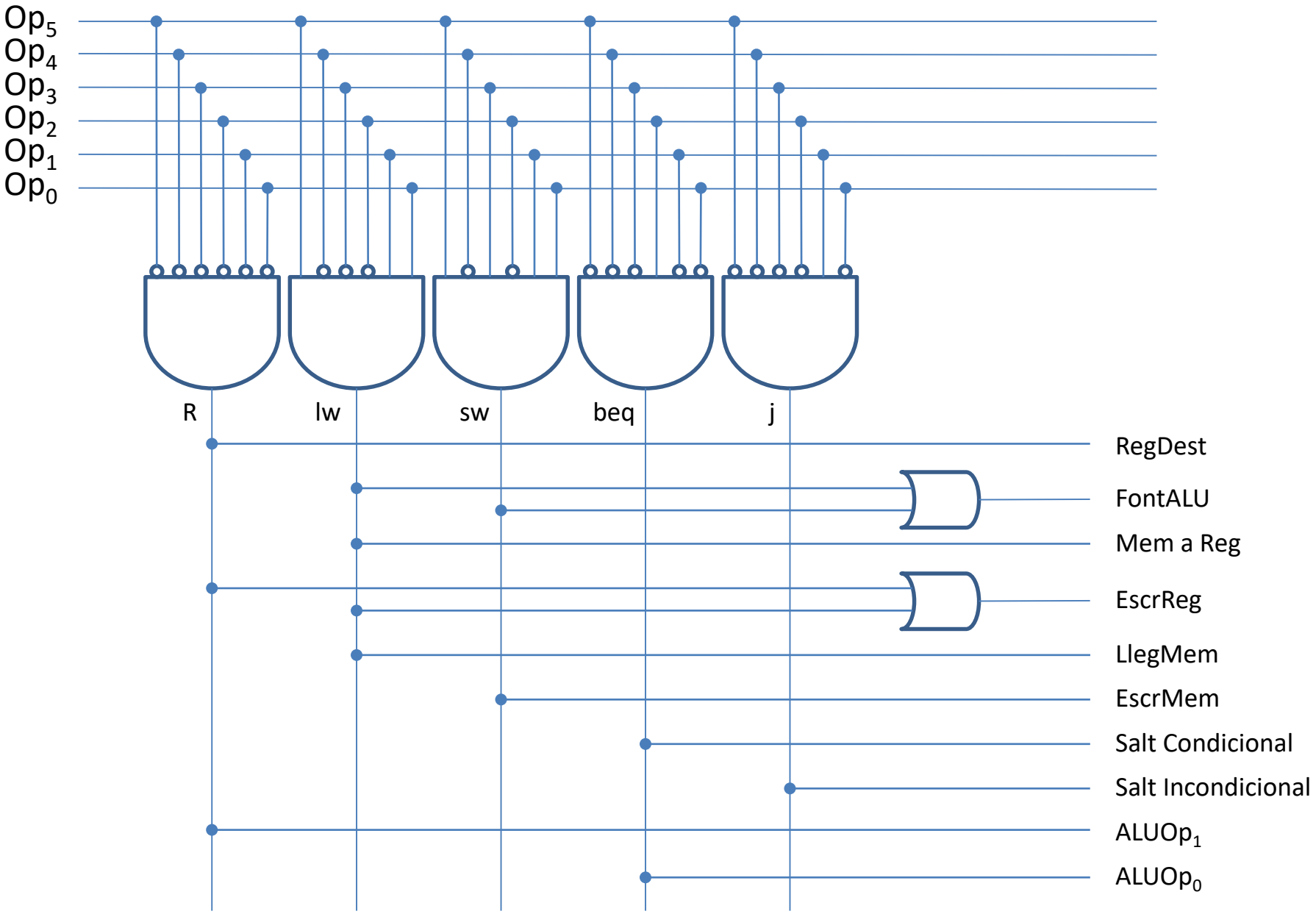
Salt incondicional



Unitat de control principal

Op_{5-0} -> (bits 31-26)	**00 0000 (0)**	**10 0011 (35)**	**10 1011 (43)**	**00 0100 (4)**	**00 0010 (2)**
	Format R	lw	sw	beq	j
RegDest	1	0	X	X	X
FontALU	0	1	1	0	X
Mem a Reg	0	1	X	X	X
EscrReg	1	1	0	0	0
LlegMem	0	1	0	0	0
EscrMem	0	0	1	0	0
Salt Condicional	0	0	0	1	0
Salt Incondicional	0	0	0	0	1
$ALUOp_1$	1	0	0	0	X
$ALUOp_0$	0	0	0	1	X

Unitat de control principal: Implementació



Problemàtica dels processadors monocicle

- Implementació funciona correctament, però son ineficients.
  - Un cicle de rellotge és igual per a totes les instruccions. I queda determinat, pel temps d'execució de la instrucció més lenta. (LOAD)
  - LOAD -> utilitza 5 unitats funcionals en sèrie: Memòria d'instruccions, Banc de registres, ALU, Memòria de dades i el Banc de registres.
  - La penalització per utilitzar un processador monocicle, pot ser acceptable si hi ha poques instruccions (primers processadors). Si hi ha una unitat de coma flotant o moltes instruccions (complexes) s'allarga molt el temps d'execució.