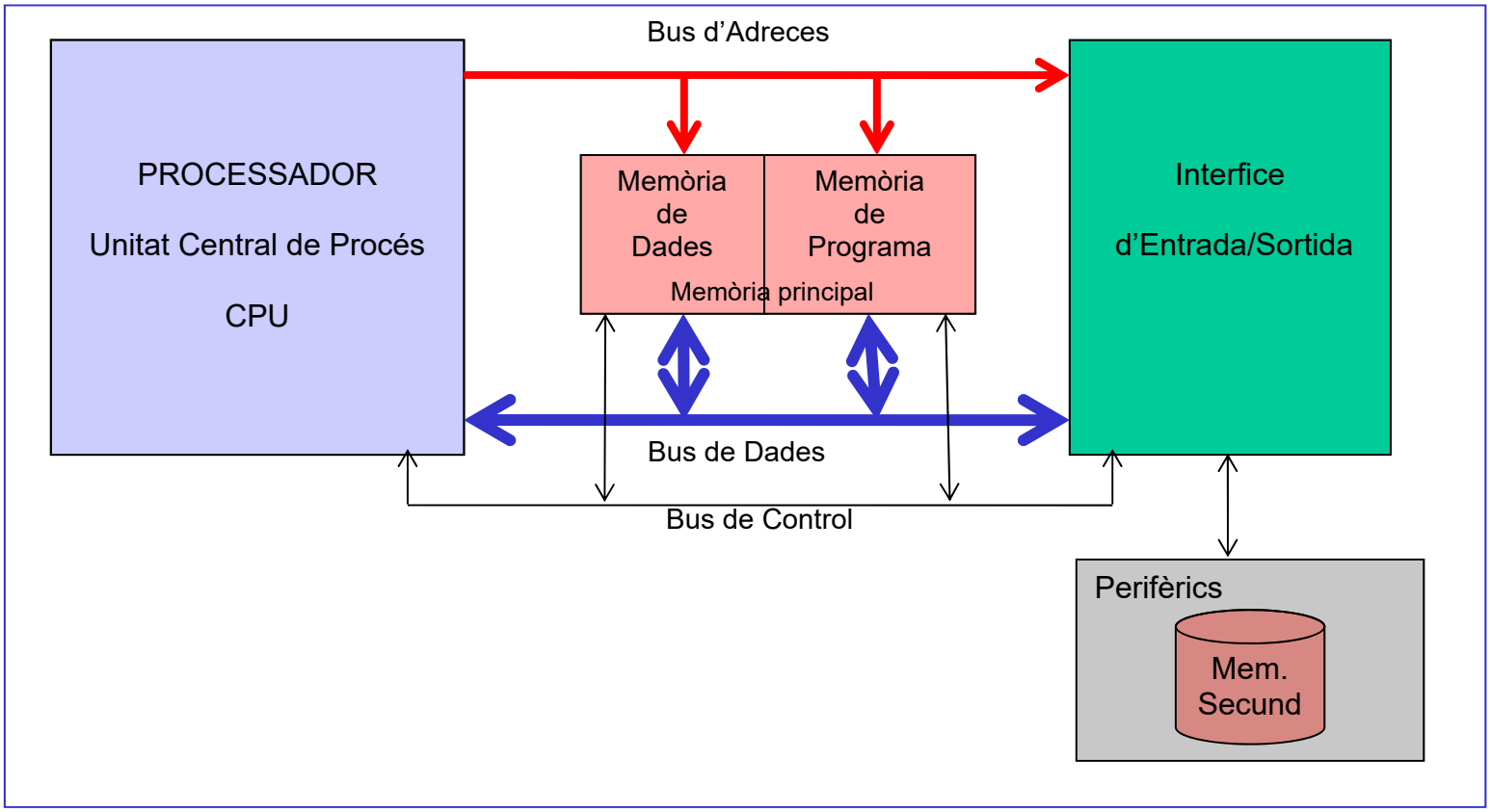
Organització
I
Estructura
dels Computadors

Funcionament d'un computador tipus Von Neumann

- Programes guardats a memòria secundària: (disc dur; CD,....)
- Unitat E/S porta el programa a executar fins a la memòria principal
- Execució d'instruccions una-a-una



Funcionament- Organització i estructura

CPU:

•Responsable d'executar les instruccions
•Controlar el funcionament de la resta d'elements del computador.
•Llegir les instruccions de memòria,
•Interpretar-les
•Realitzar les accions necessàries per executar-les.

•UNITAT DE CONTROL
•UNITAT DE PROCÉS O EXECUCIÓ
•REGISTRES
•Memòria Caché (Nivells 1 i 2)

La CPU Unitat de Procés o Unitat d'Execució (Data path)

On s'executen les instruccions.
Multitud de Tasques:

1. Càlcul, operacions
2. Moviment de dades
3. Càlcul d'adreces...

Elements típics:

- Una o més ALUs:
  - Operacions aritmètiques: suma, resta, desplaçament, complement
  - Operacions lògiques: AND, OR, XOR, NOT
  - Altre recurs de càlcul: multiplicador, desplaçadors (opcional)
- Un conjunt de registres:
  - operands i resultats
  - Nombre de registres = potència de la CPU
- Generador d'adreces.
- Cerca d'instruccions,
- Llegir, escriure dades de/a memòria

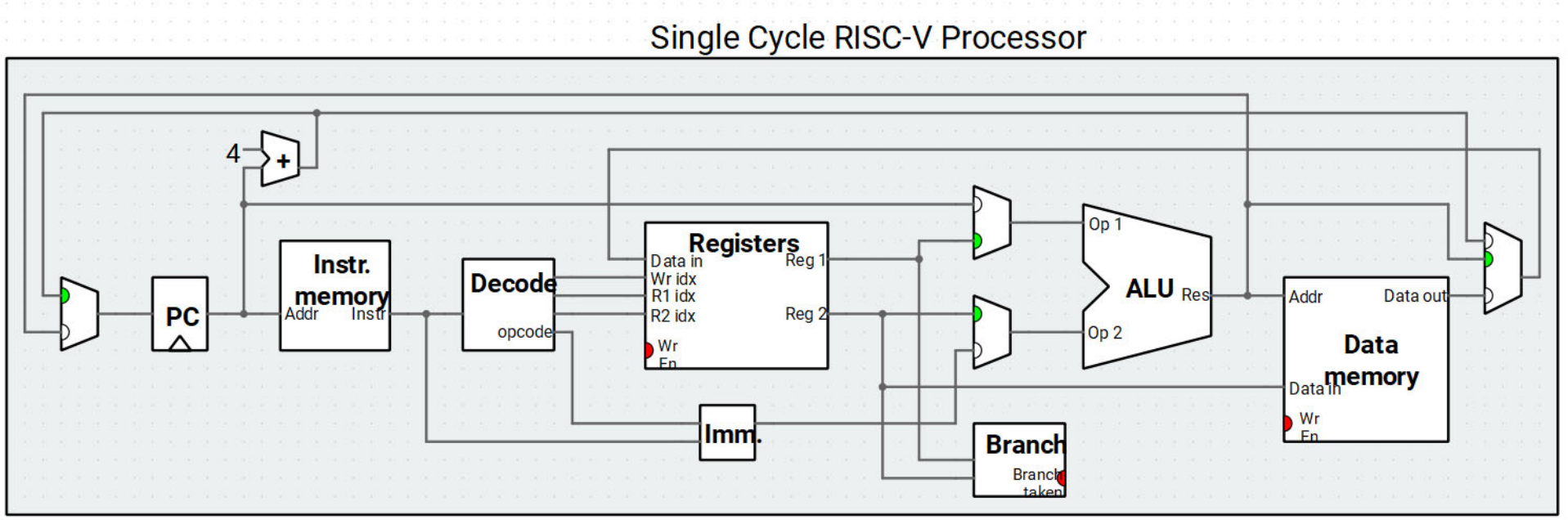
La CPU: Arquitectura vs Disseny

- L'arquitectura de la CPU vindrà donada pels requeriments dels arquitectes, que defineixen el format d'instruccions necessari i les necessitats requerides (capítol anterior)

- El disseny de la CPU intenta donar servei a aquesta arquitectura, desenvolupant un dispositiu (xip) capaç de complir els requeriments de l'arquitectura

Estructura bàsica de la CPU

Explicació / Recordatori estructura harvard



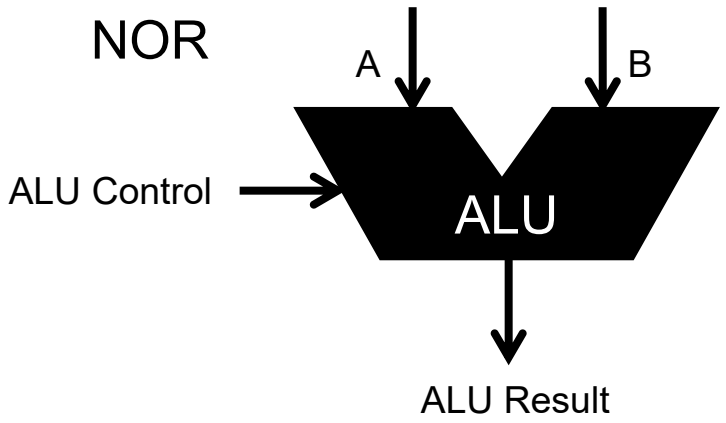
Unitat Aritmetico Lògica (I)

La Unitat Aritmètic Lògica s'encarrega de tractar les dades, executant les operacions definides d'acord amb el programa en curs que s'està executant.

Tal i com s'ha comentat (i com defineix el seu nom) la ALU fa bàsicament

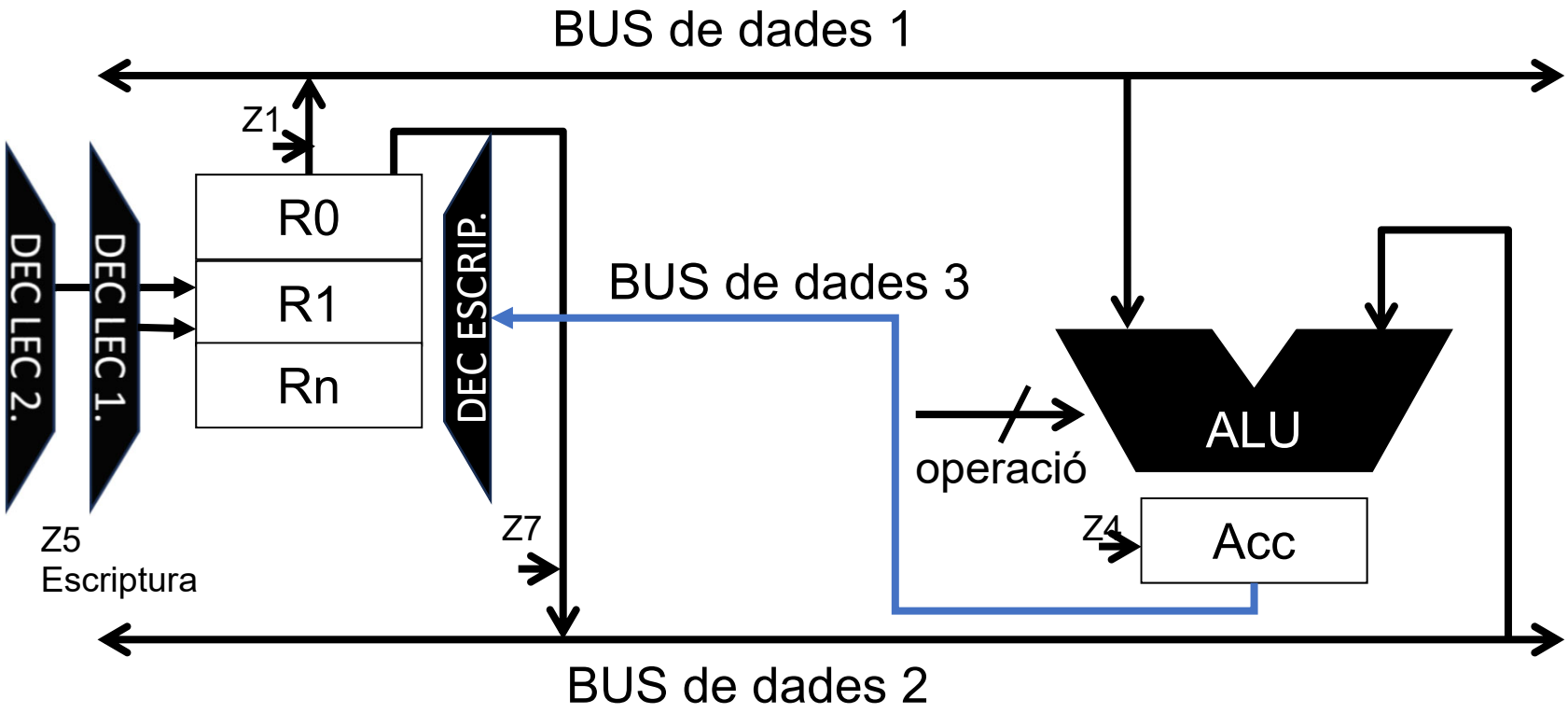
Operacions Aritmètiques
- +
- -
- desplaçaments
- multiplicacions
- divisions

Operacions Lògiques
- AND
- OR
- NOT
- NAND
- NOR



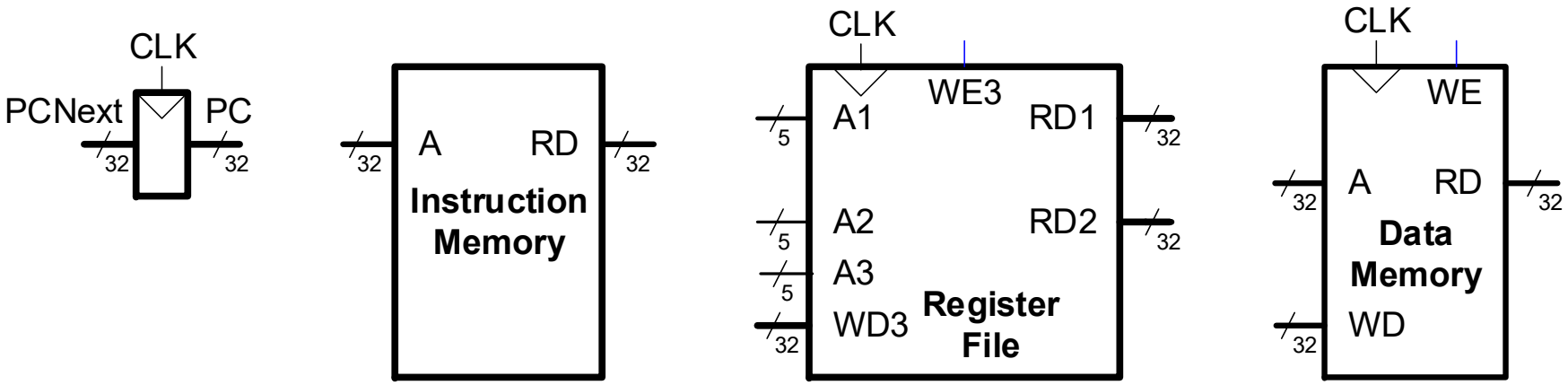
Unitat Aritmetico Lògica (XIX)

Detall de la connexió entre el banc de registres i la ALU



Elements bàsics de l'arquitectura RISC-V

Independentment del tipus de CPU aquests elements sempre els trobem en l'arquitectura RISC-V



Single-Cycle RISC-V Processor

- Camí de dades
- Control

Single-Cycle RISC-V Processor

- Disseny del camí de dades
- Exemple d'un programa executant-se

Example Program:

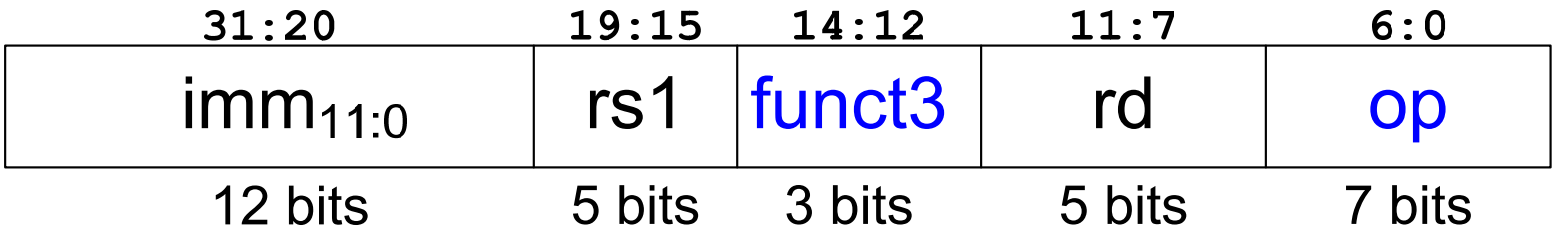
Address	Instruction	Type	Fields						Machine Language
			$imm_{11:0}$		rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100		01001	010	00110	0000011	FFC4A303
			$imm_{11:5}$	rs2	rs1	f3	$imm_{4:0}$	op	
0x1004	sw x6, 8(x9)	S	0000000	00110	01001	010	01000	0100011	0064A423
			funct7	rs2	rs1	f3	rd	op	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233
			$imm_{12,10:5}$	rs2	rs1	f3	$imm_{4:1,11}$	op	
0x100C	beq x4, x4, L7	B	1111111	00100	00100	000	10101	1100011	FE420AE3

Single-Cycle RISC-V Processor

- **Datapath:** Comencem amb la instrucció `lw`
- **Exemple:** `lw x6, -4(x9)`

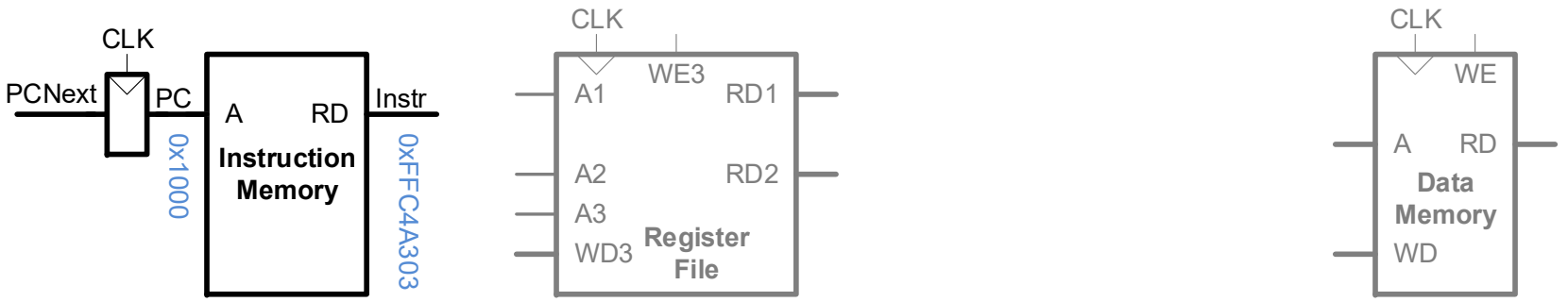
  `lw rd, imm(rs1)`

I-Type



Single-Cycle RISC-V Processor: Fetch `lw`

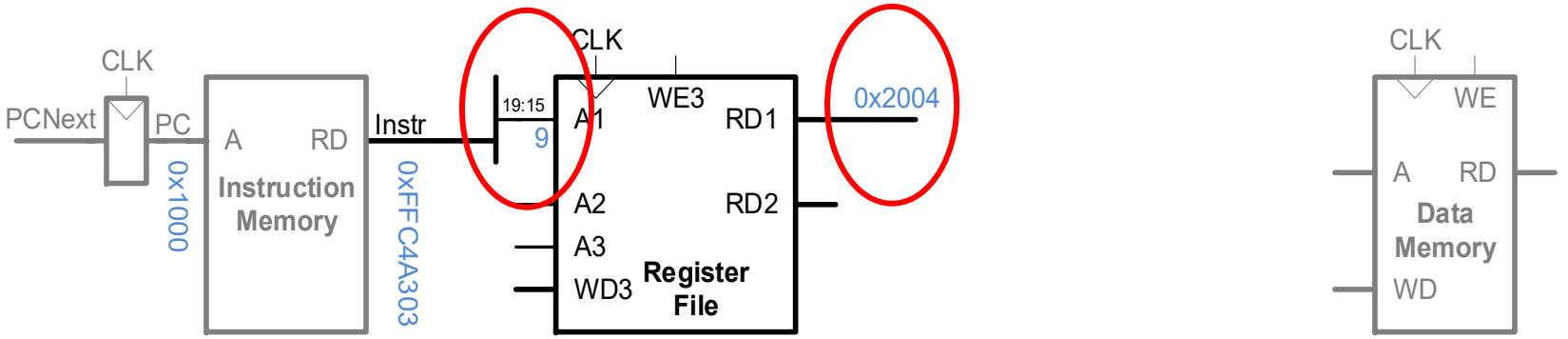
Pas 1: Fetch



Address	Instruction	Type	Fields				Machine Language
			$\mathbf{imm_{11:0}}$	**rs1**	**f3**	**rd**	**op**	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle RISC-V Processor: lectura Reg. `lw`

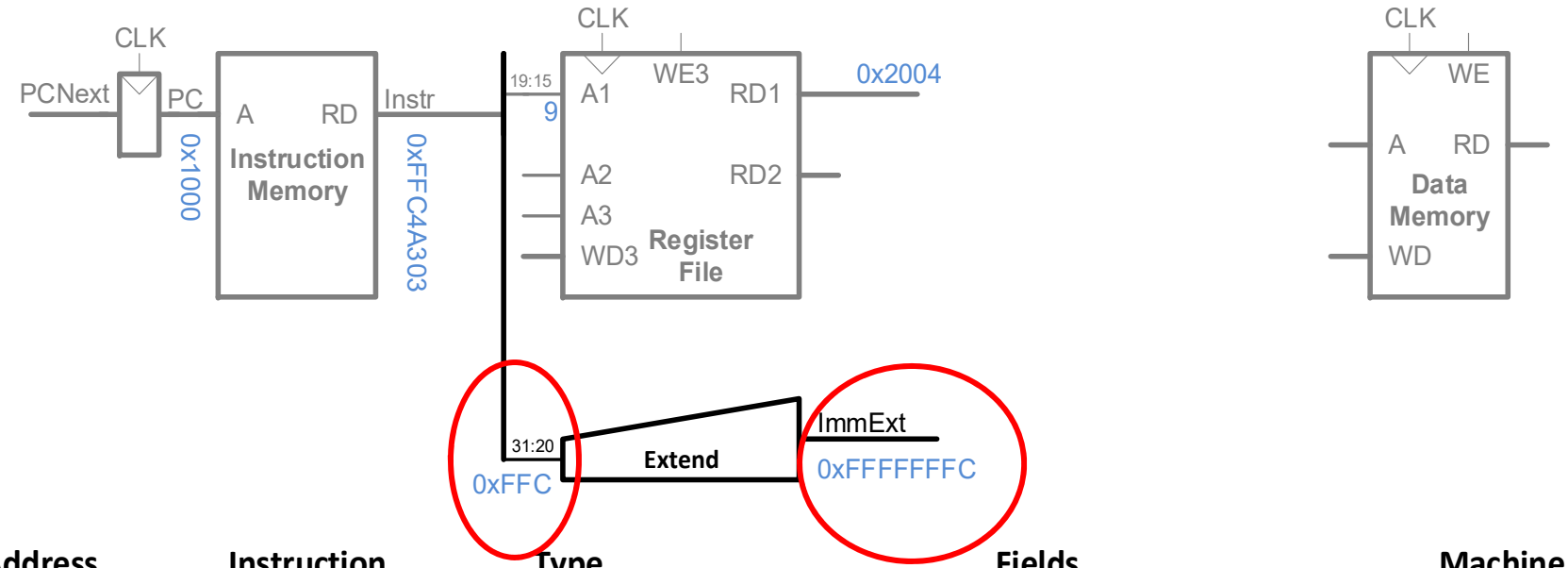
Pas 2: Lectura operand font (**rs1**)



Address	Instruction	Type	Fields				Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle RISC-V Processor: l'immediat a `lw`

Pas 3: Extenem l'immediat a 32 bits

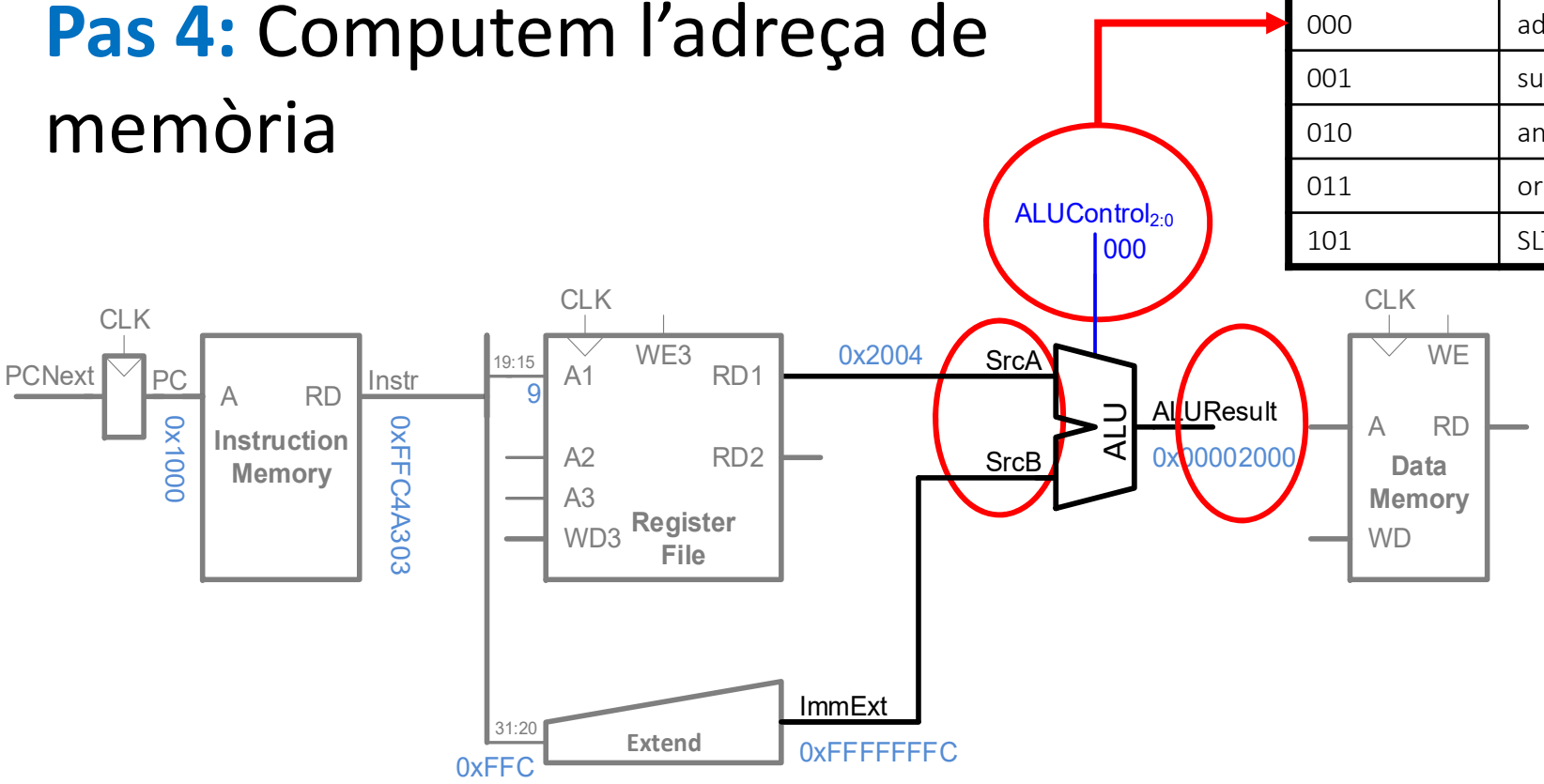


Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle RISC-V Processor: adreça de `lw`

Pas 4: Computem l'adreça de memòria

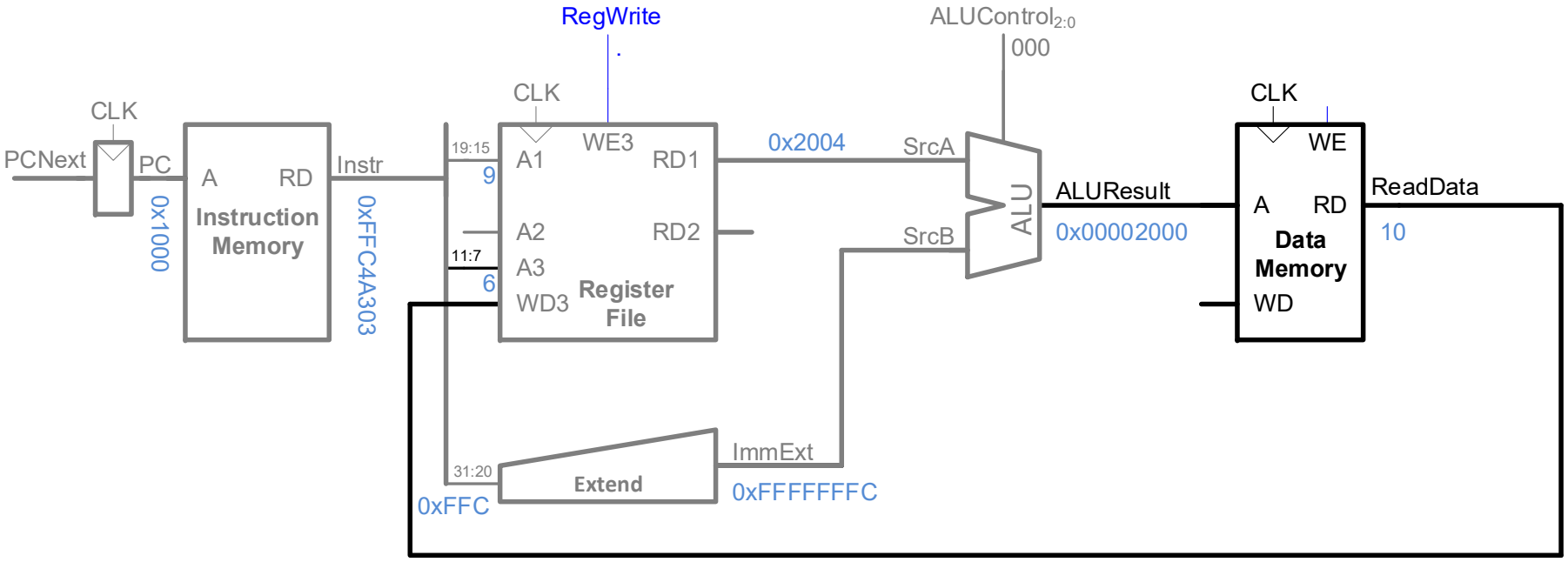
$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT



Address	Instruction	Type	Fields				Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle RISC-V Processor: `lw`

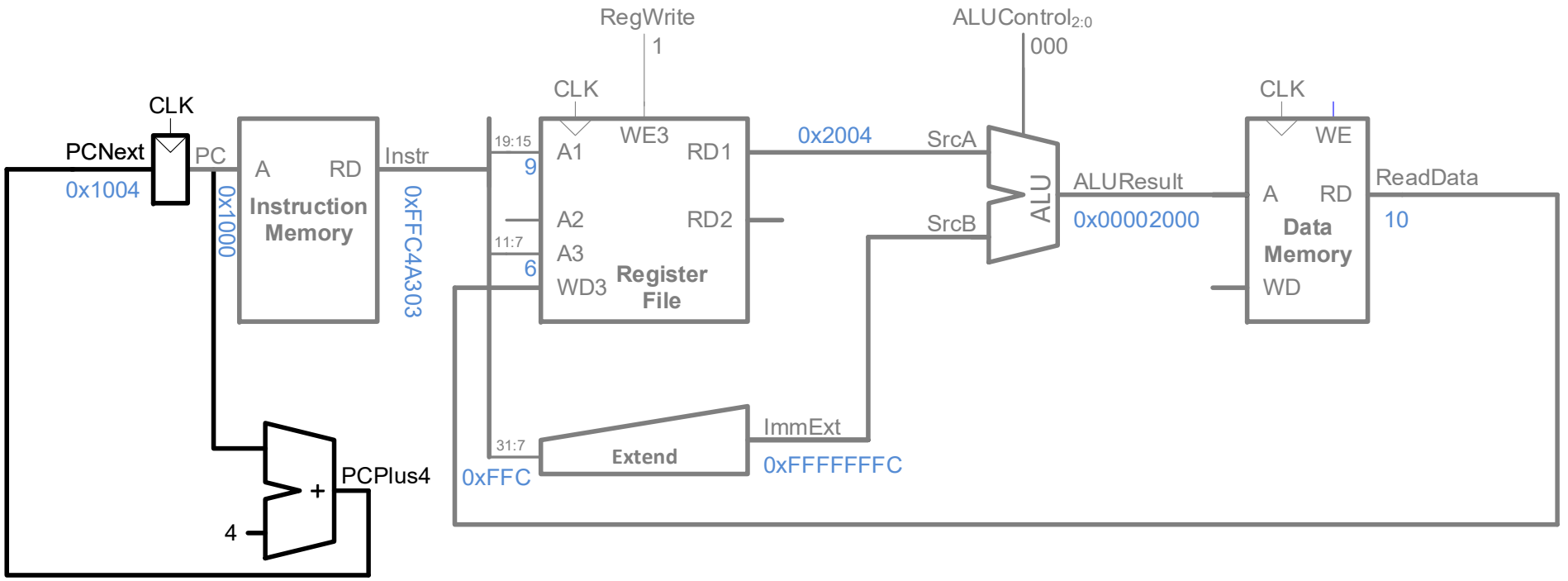
Pas 5: Lectura de dades de la memòria I escriptura en el register especificat



Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle RISC-V Processor: `lw`

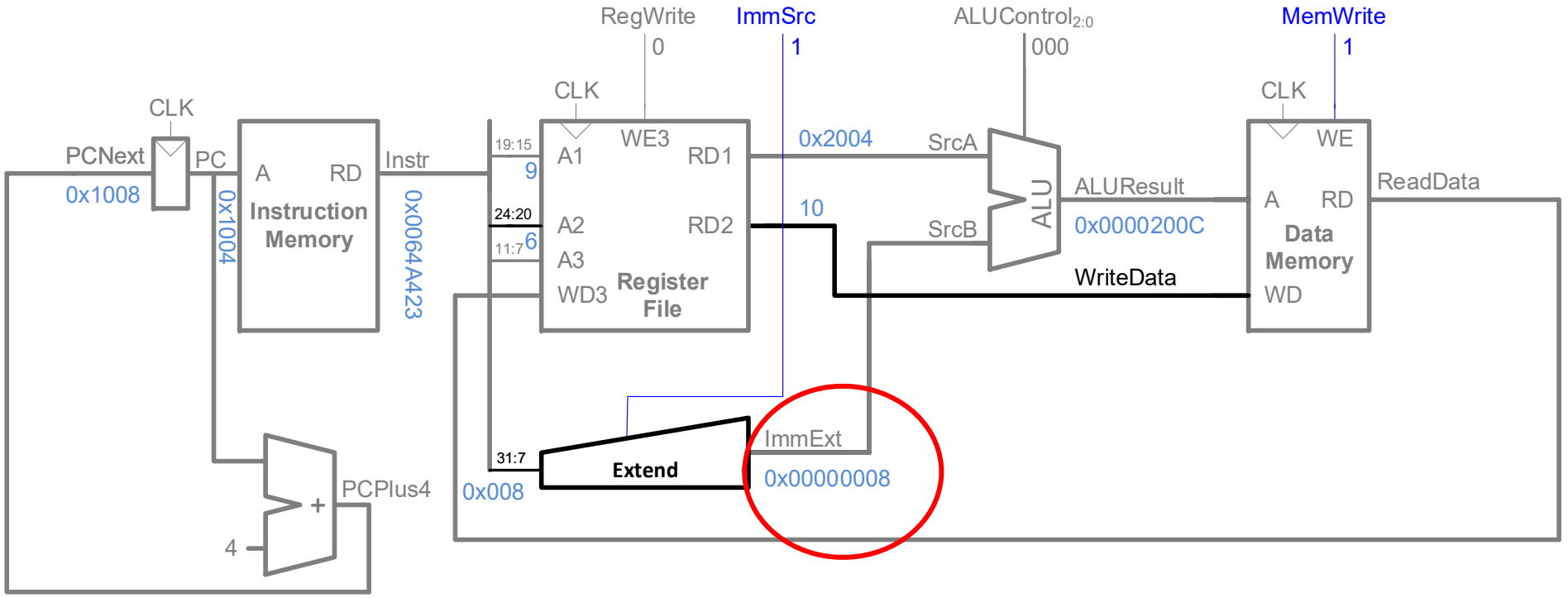
Pas 6: Calcular l'adeça de la següent instrucció



Address	Instruction	Type	Fields					Machine Language
			$imm_{11:0}$	rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100	01001	010	00110	0000011	FFC4A303

Single-Cycle RISC-V Processor: `Sw`

- **Immediat:** Ara a {instr[31:25], instr[11:7]}
- **Afegim senyals de control:** ImmSrc, MemWrite



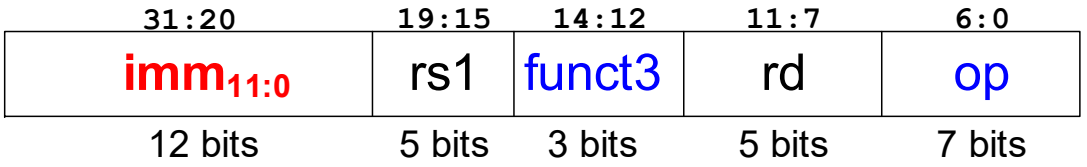
Address	Instruction	Type	Fields						Machine Language
			$imm_{11:5}$	rs2	rs1	f3	$imm_{4:0}$	op	
0x1004	`sw x6, 8(x9)`	S	0000000	00110	01001	010	01000	0100011	0064A423

Comte amb la distribució de bits de l'immediat!!

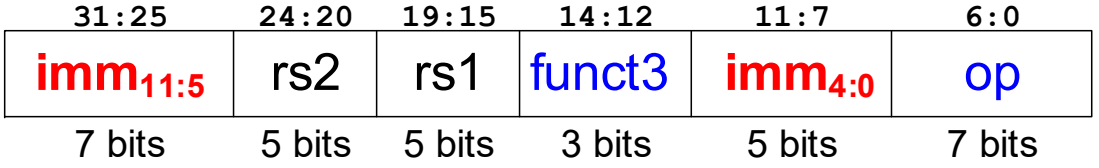
Single-Cycle RISC-V Processor: Immediat

ImmSrc	ImmExt	Instruction Type
0	{{20{instr[31]}}, **instr[31:20]**}	I-Type
1	{{20{instr[31]}}, **instr[31:25], instr[11:7]**}	S-Type

I-Type



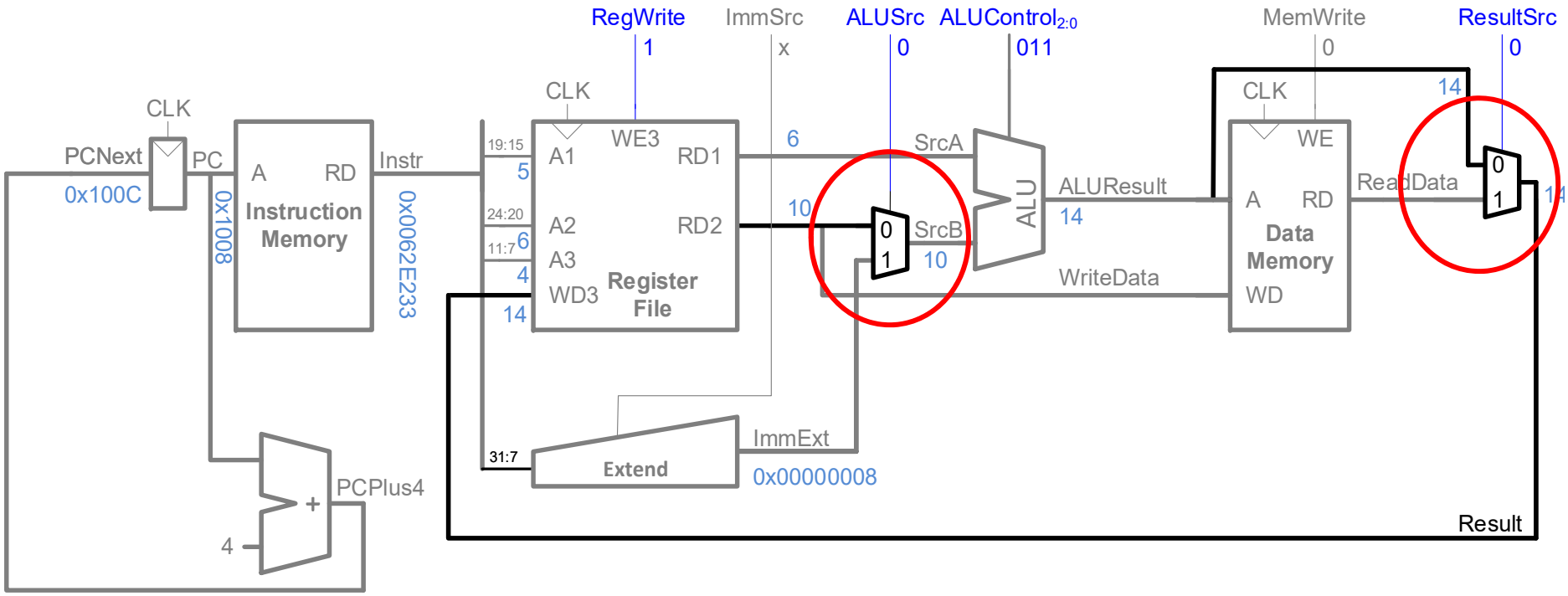
S-Type



Això explia la necessitat del bit de control ImmSrc

Single-Cycle RISC-V Processor: R-type

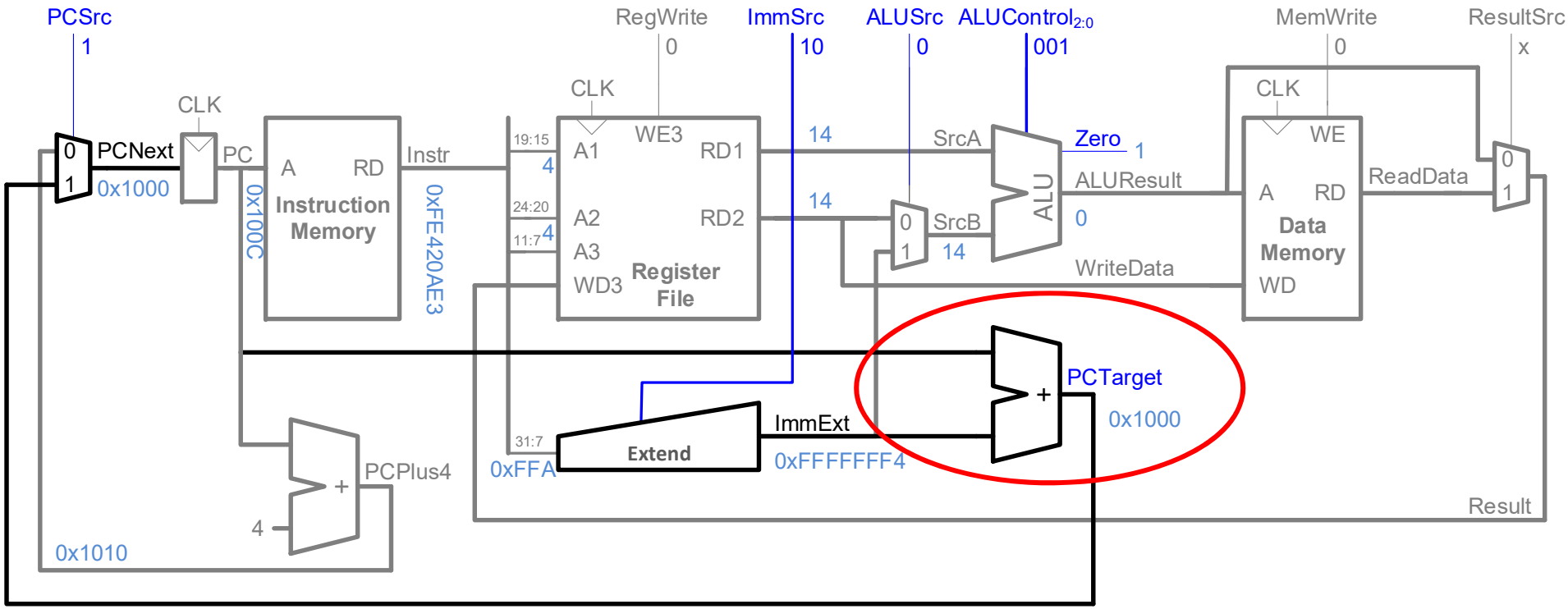
- Lectura de **rs1** i **rs2** (en lloc de **imm**)
- Escriptura de *ALUResult* en **rd**



Address	Instruction	Type	Fields					Machine Language
			funct7	rs2	rs1	f3	rd	op	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233

Single-Cycle RISC-V Processor: beq

Càlcul addreça PC: $PC_{Target} = PC + imm$

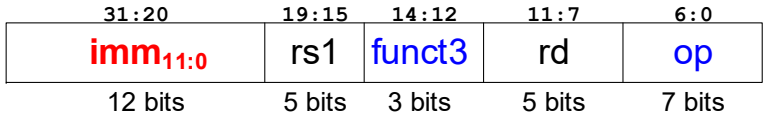


Address	Instruction	Type	Fields					Machine Language
			$imm_{12,10:5}$	rs2	rs1	f3	$imm_{4:1,11}$	op	
0x100C	beq x4, x4, L7	B	1111111	00100	00100	000	10101	1100011	FE420AE3

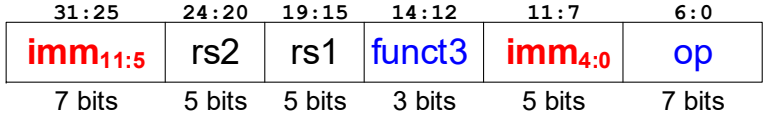
Single-Cycle Datapath: ImmExt

$ImmSrc_{1:0}$	ImmExt	Instruction Type
00	{{20{instr[31]}}, **instr[31:20]**}	I-Type
01	{{20{instr[31]}}, **instr[31:25], instr[11:7]**}	S-Type
10	{{19{instr[31]}}, **instr[31], instr[7], instr[30:25], instr[11:8], 1'b0**}	B-Type

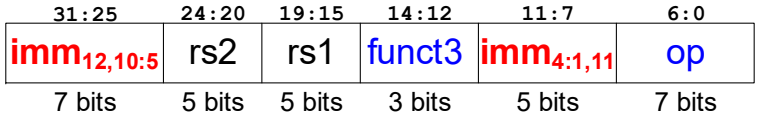
I-Type



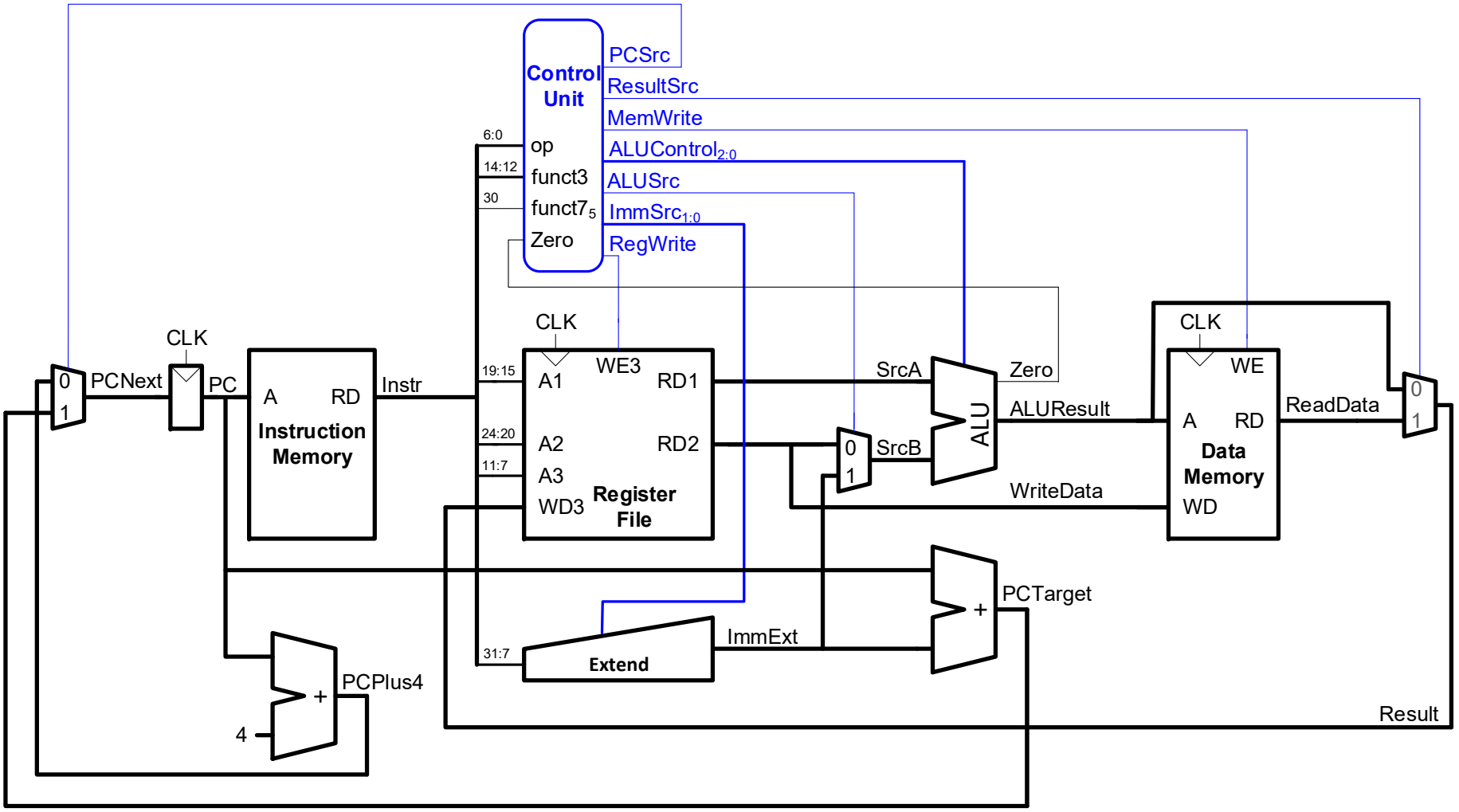
S-Type



B-Type

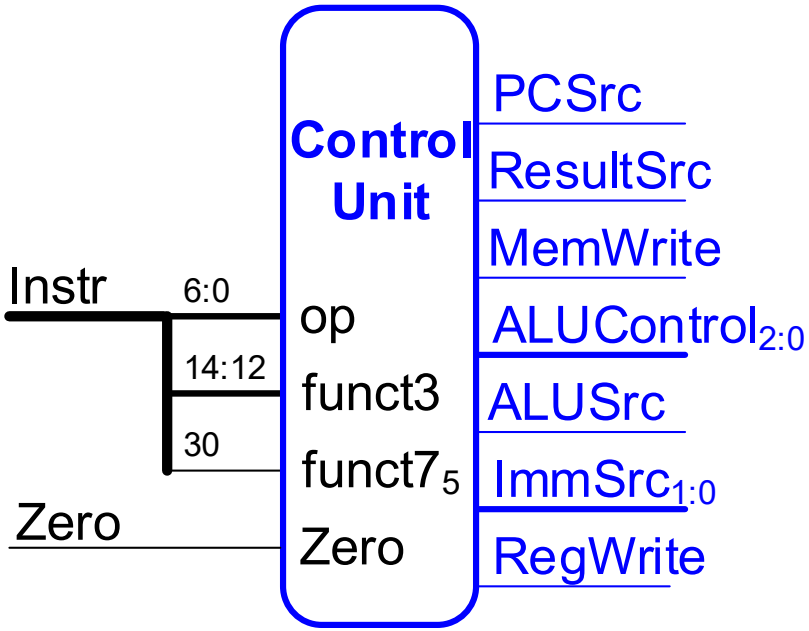


Single-Cycle RISC-V Processor

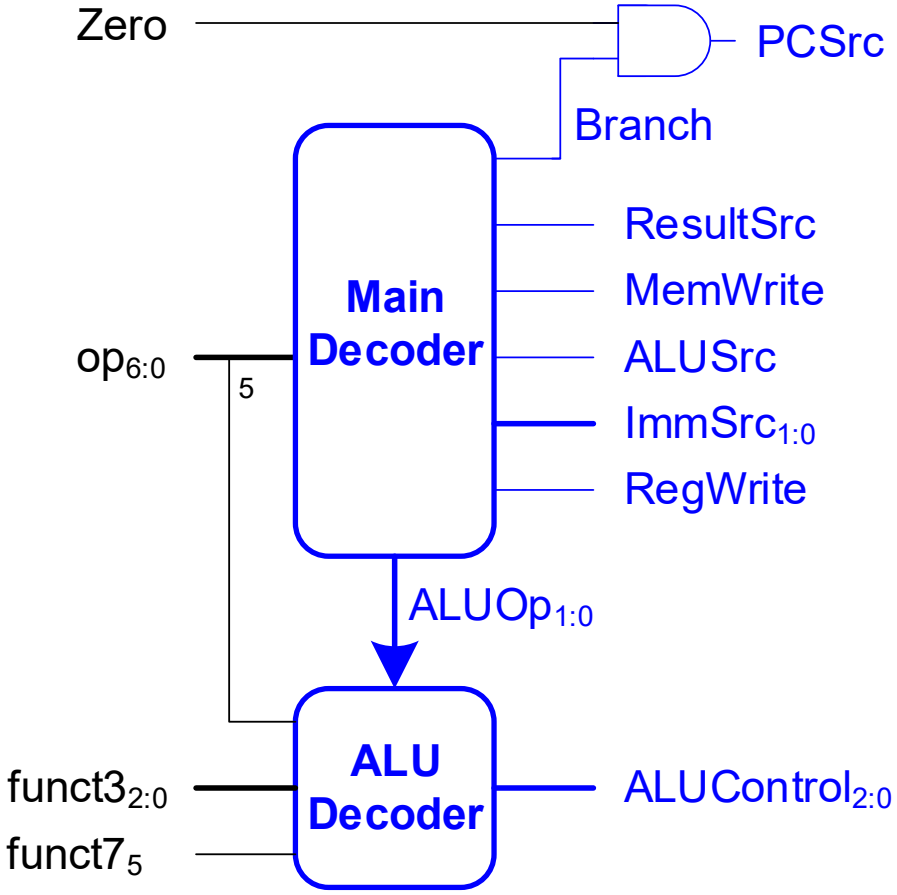


Single-Cycle Control

High-Level View

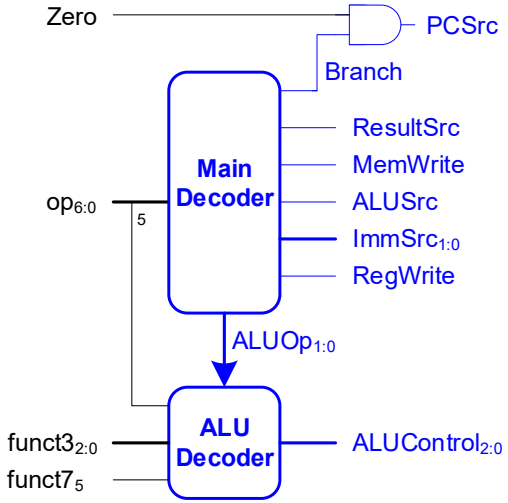


Low-Level View



Single-Cycle Control: Main Decoder

op	Instr.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	**`lw`**							
35	**`sw`**							
51	R-type							
99	**`beq`**							

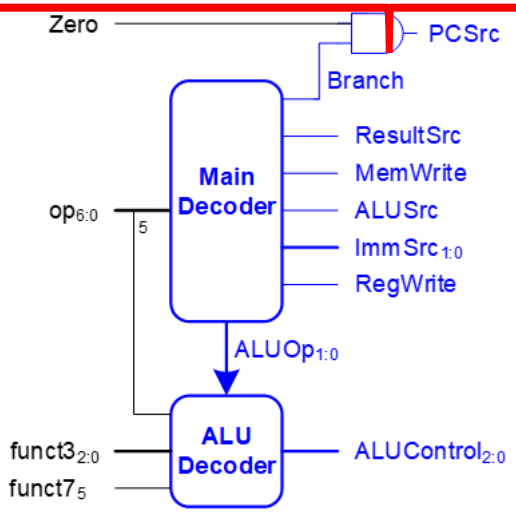


Single-Cycle Control: Main Decoder

op	Instr.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	**lw**	1	00	1	0	1	0	00
35	**sw**	0	01	1	1	X	0	00
51	R-type	1	XX	0	0	0	0	10
99	**beq**	0	10	0	0	X	1	01

Instrucció

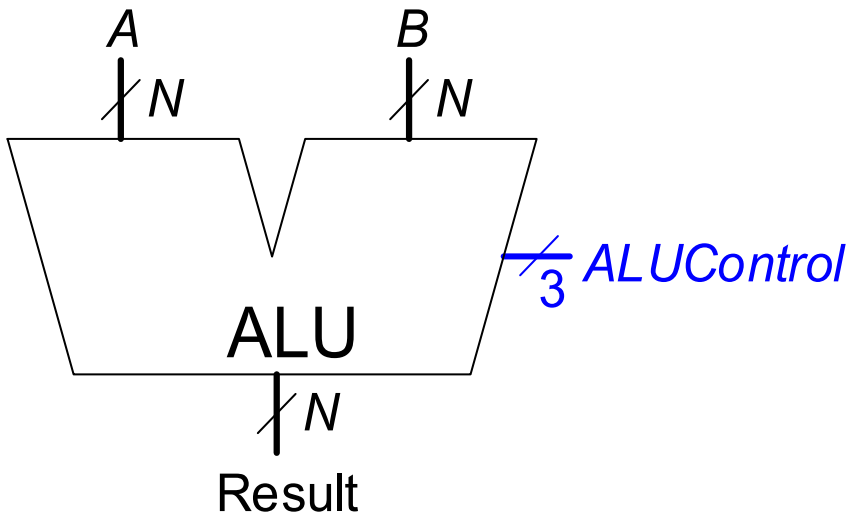
Entrada al dec. principal



Actuació sobre el data path

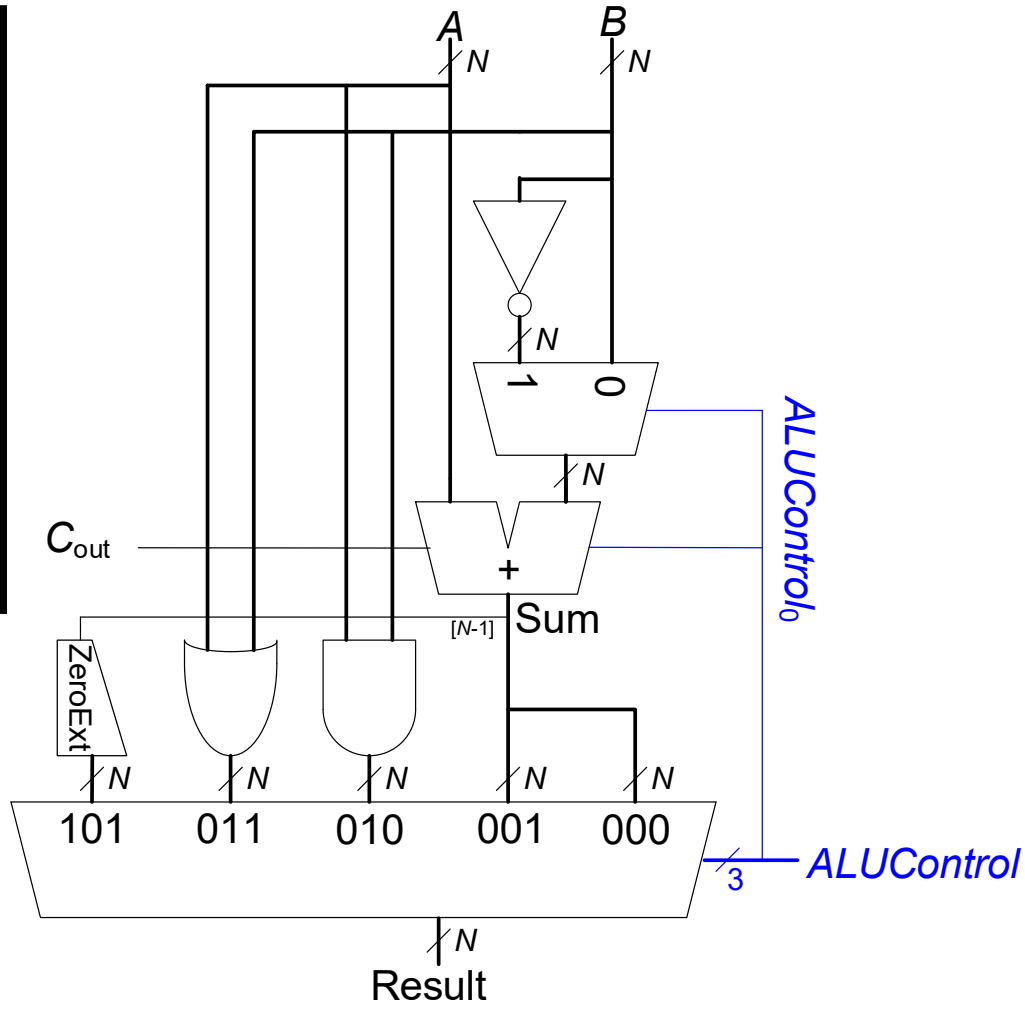
Review: ALU

$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT

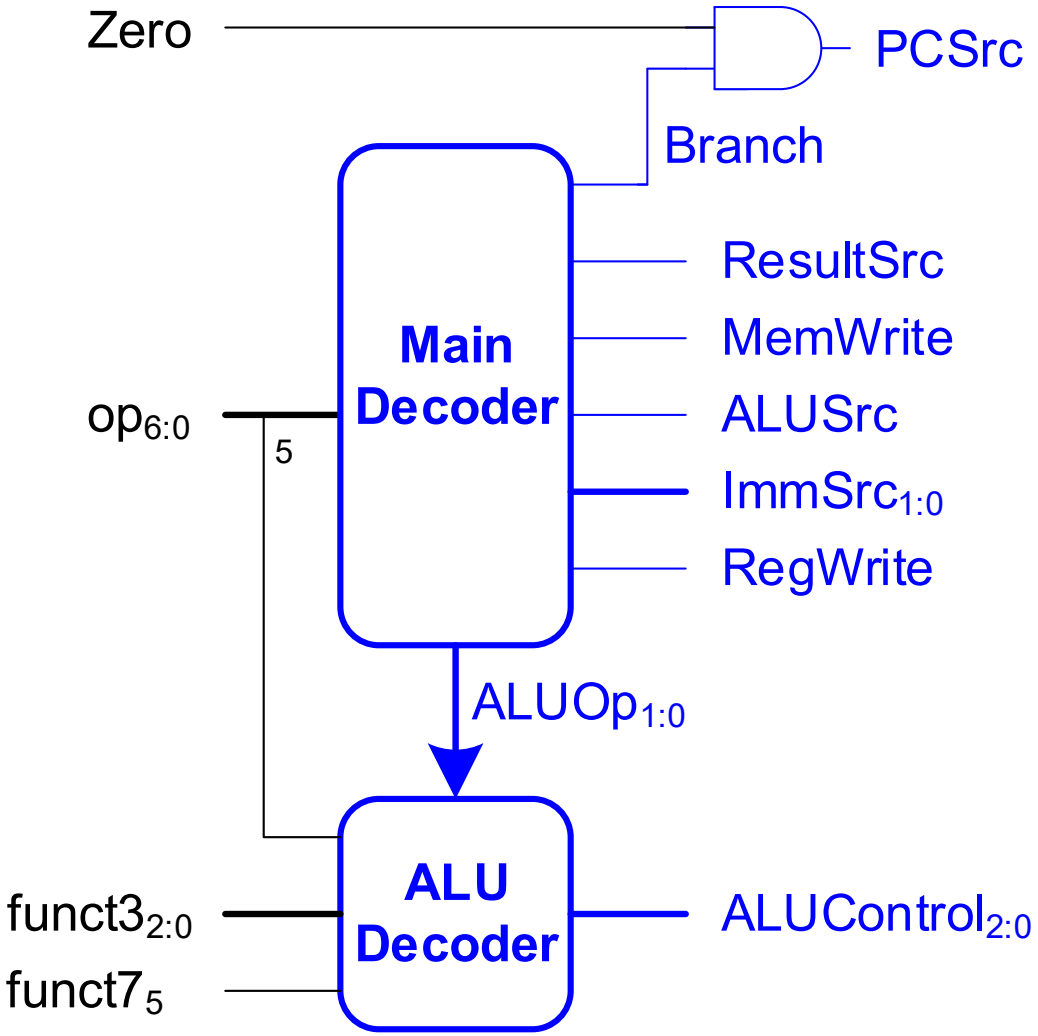


Review: ALU

$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT

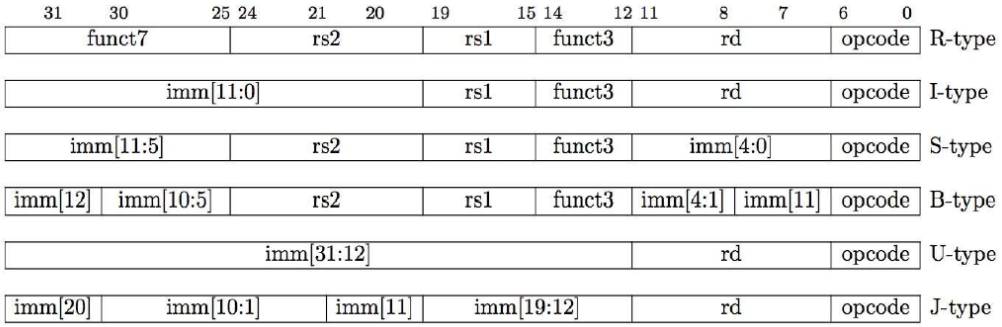


Single-Cycle Control: ALU Decoder

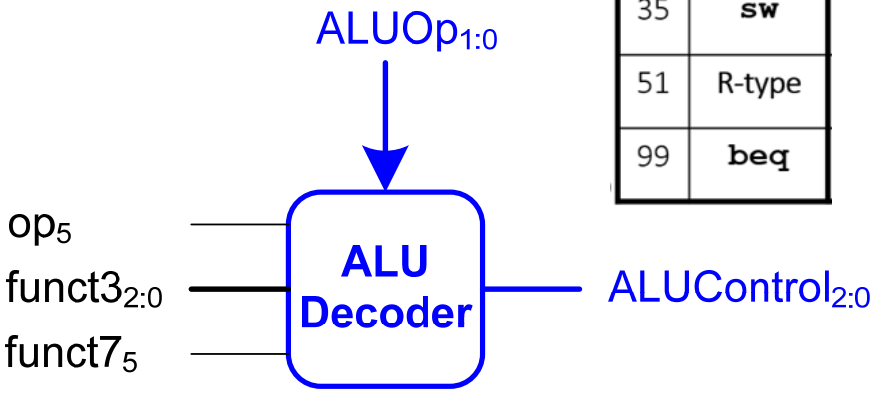


Single-Cycle Control: ALU Decoder

ALUOp	funct3	op_5, $funct7_5$	Instruction	$ALUControl_{2:0}$
00	x	x	**lw, sw**	000 (add)
01	x	x	**beq**	001 (subtract)
10	000	00, 01, 10	**add**	000 (add)
	000	11	**sub**	001 (subtract)
	010	x	**slt**	101 (set less than)
	110	x	**or**	011 (or)
	111	x	**and**	010 (and)

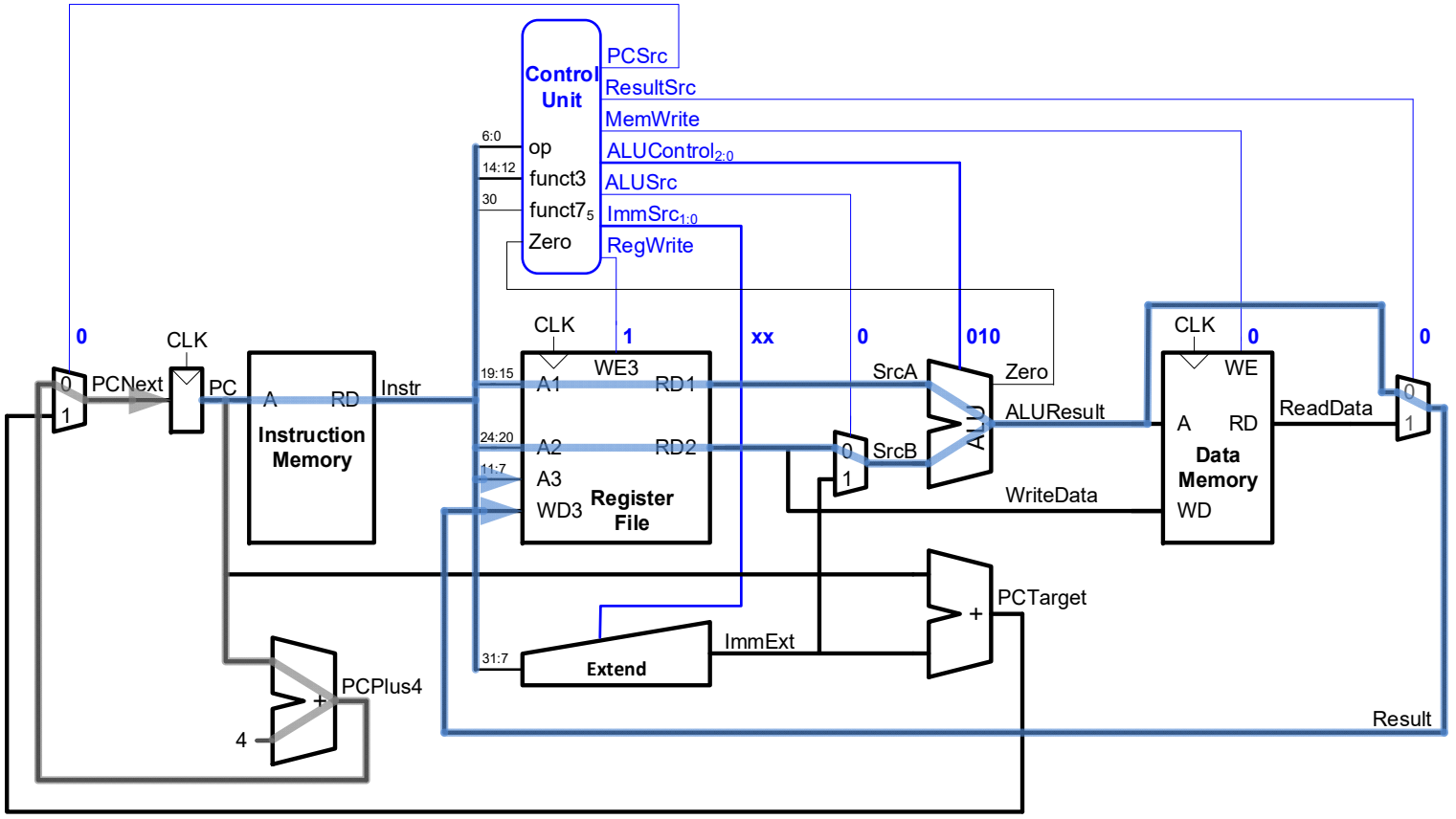


op	Instr.
3	**lw**
35	**sw**
51	R-type
99	**beq**



Exemple: `and`

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
51	R-type	1	XX	0	0	0	0	10



`and x5, x6, x7`

Extended Functionality: I-Type ALU

Millorem la funcionalitat del single-cycle processor afegint les instruccions **I-Type ALU**:

`addi`, `andi`, `ori`, and `slti`

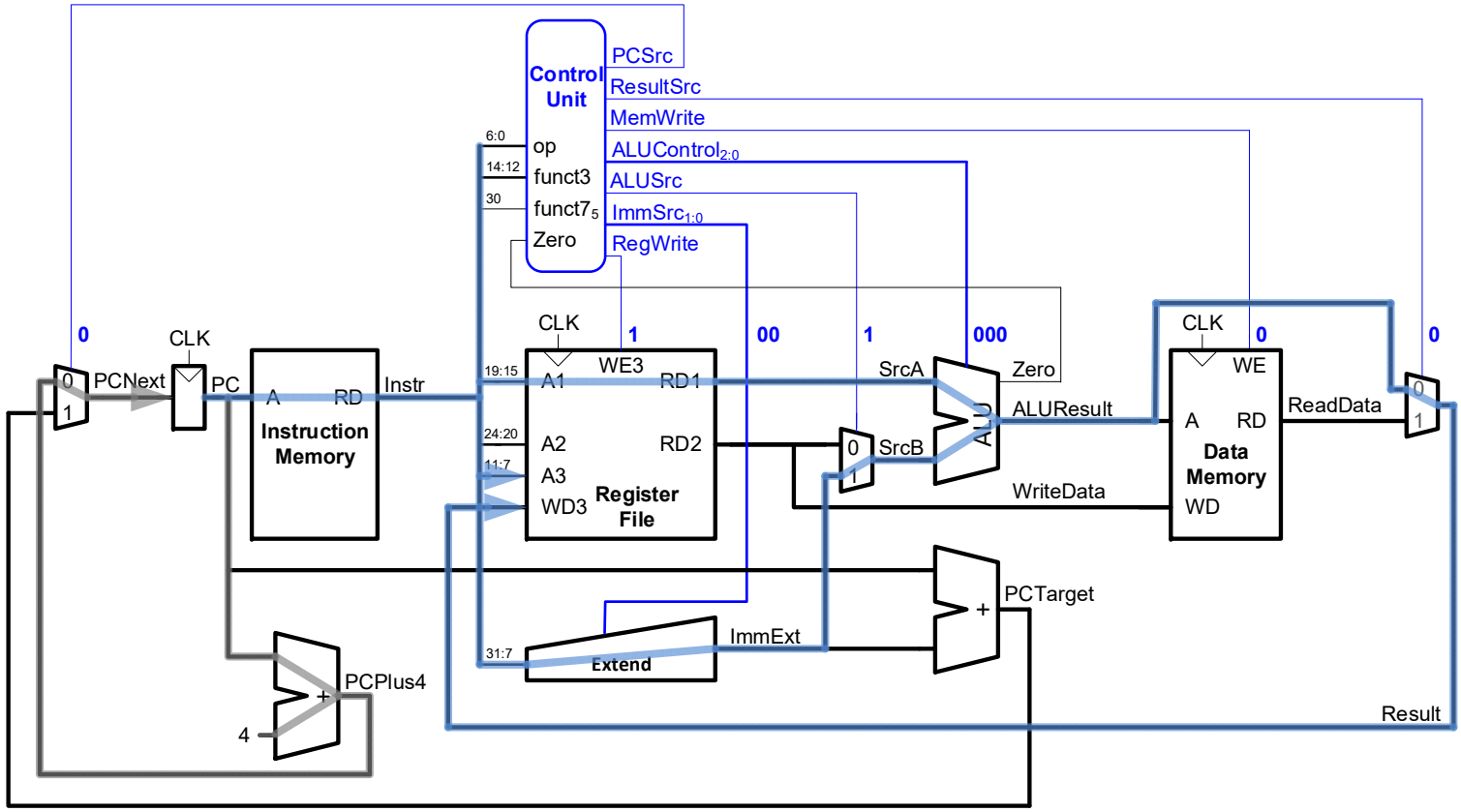
- **Similar a** les instruccions **R-type**
- Pero la **segona font** és un **immediat**
- Cal canviar la ***ALUSrc*** per escollir l'immediat
- I ***ImmSrc*** per seleccionar l'opció correcta de l'immediat

Extended Functionality: I-Type ALU

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	**lw**	1	00	1	0	1	0	00
35	**sw**	0	01	1	1	X	0	00
51	R-type	1	XX	0	0	0	0	10
99	**beq**	0	10	0	0	X	1	01
19	**I-type**	1	00	1	0	0	0	10

Extended Functionality: addi

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
19	**I-type**	1	00	1	0	0	0	10



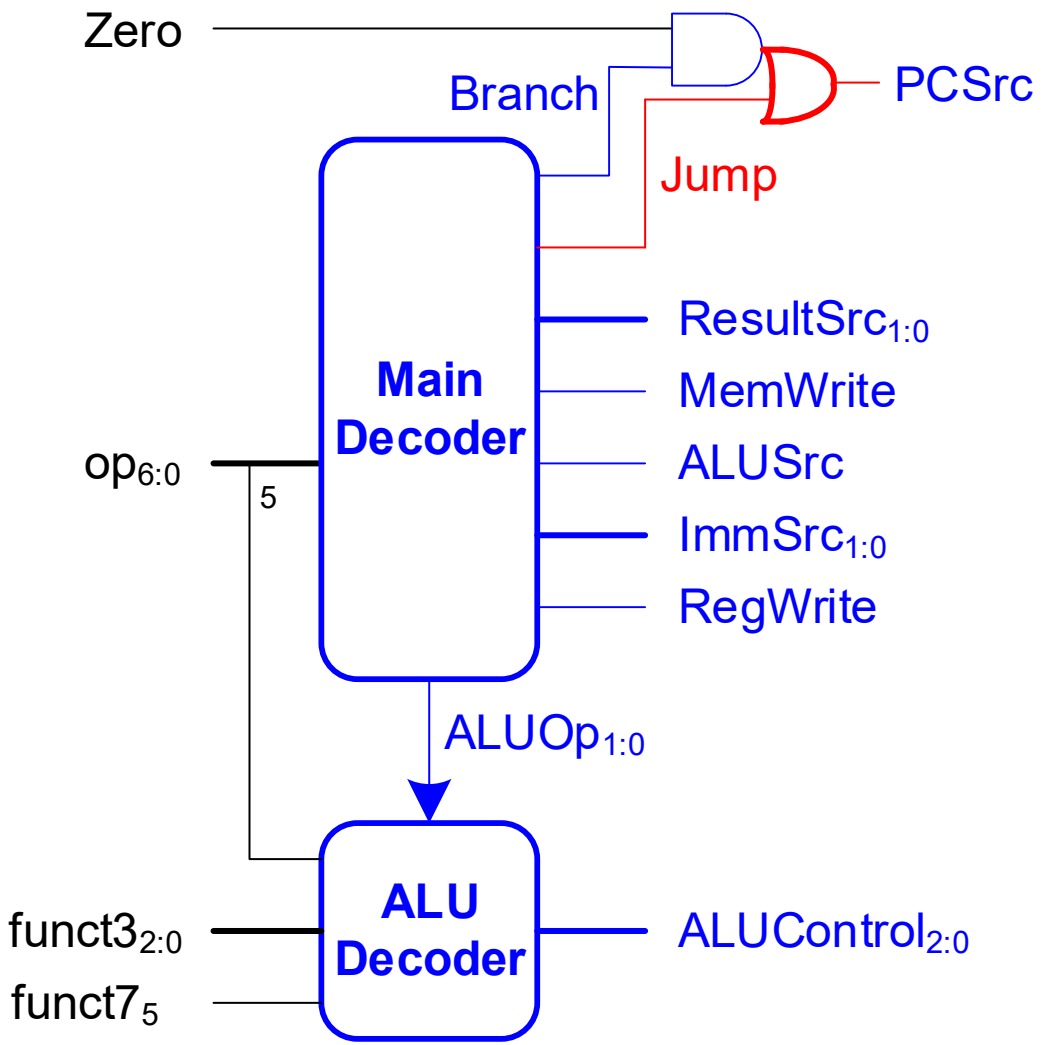
addi x5, x6, -33

Extended Functionality: `jal`

Afegim ara al single-cycle processor la possibilitat de fer servir instruccions de salt tipus `jal`

- **Similar a `beq`**
- Però ara el salt **sempre es fa**
  - *PCSrc* hauria de ser 1
- **El format de l'Immediat** és diferent
  - Necessitem un nou valor de *ImmSrc* Igual a 11
- A més `jal` ha de fer el càlcul **PC+4** i guardar-lo en **`rd`**
  - Agafar el valor de PC+4 del sumador i guardar-ho al banc de registres a través del ResultMux

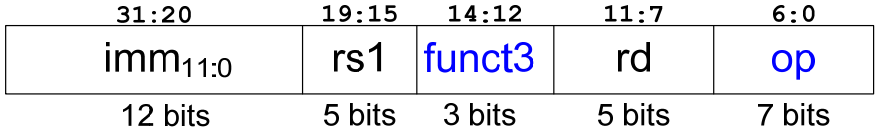
Extended Functionality: `jal`



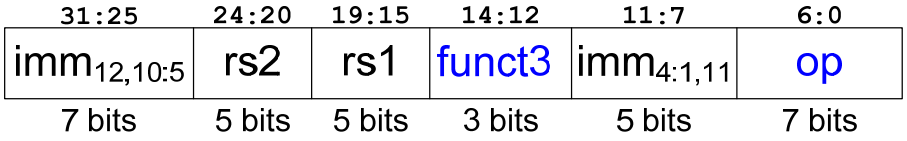
Extended Functionality: *ImmExt*

$ImmSrc_{1:0}$	ImmExt	Instruction Type
00	{{20{instr[31]}}, instr[31:20]}	I-Type
01	{{20{instr[31]}}, instr[31:25], instr[11:7]}	S-Type
10	{{19{instr[31]}}, instr[31], instr[7], instr[30:25], instr[11:8], 1'b0}	B-Type
11	{{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0}	J-Type

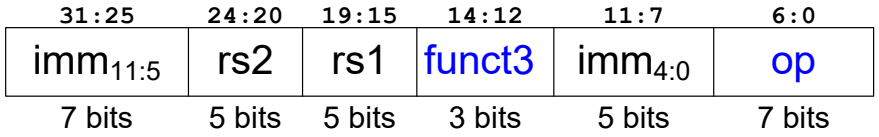
I-Type



B-Type



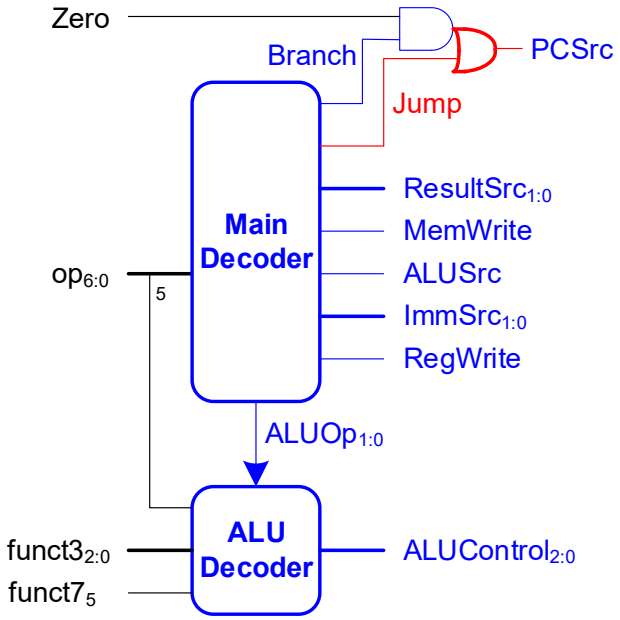
S-Type



J-Type

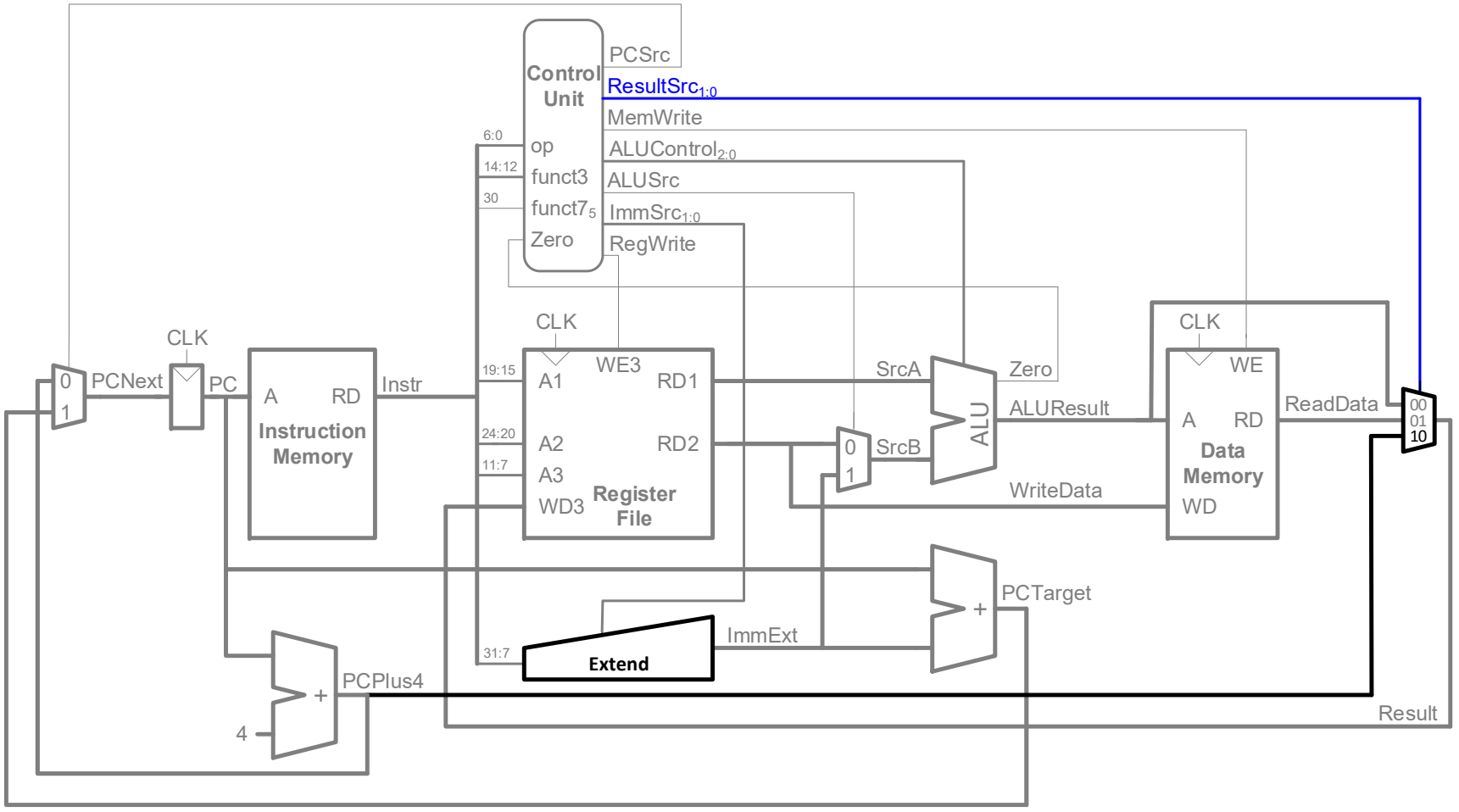
Extended Functionality: `jal`

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
3	**`lw`**	1	00	1	0	10	0	00	0
35	**`sw`**	0	01	1	1	XX	0	00	0
51	R-type	1	XX	0	0	01	0	10	0
99	**`beq`**	0	10	0	0	XX	1	01	0
19	**I-type**	1	00	1	0	01	0	10	0
111	**`jal`**	1	11	X	0	10	0	XX	1

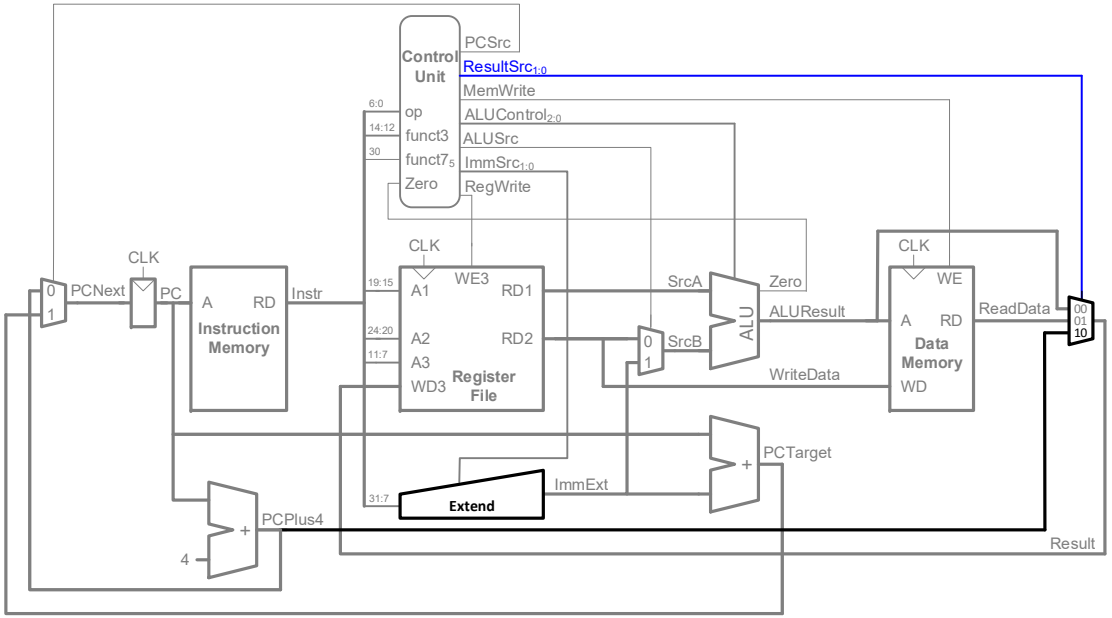


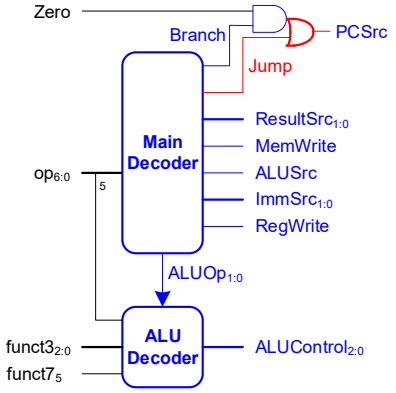
Extended Functionality: `jal`

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
111	**jal**	1	11	X	0	10	0	XX	1

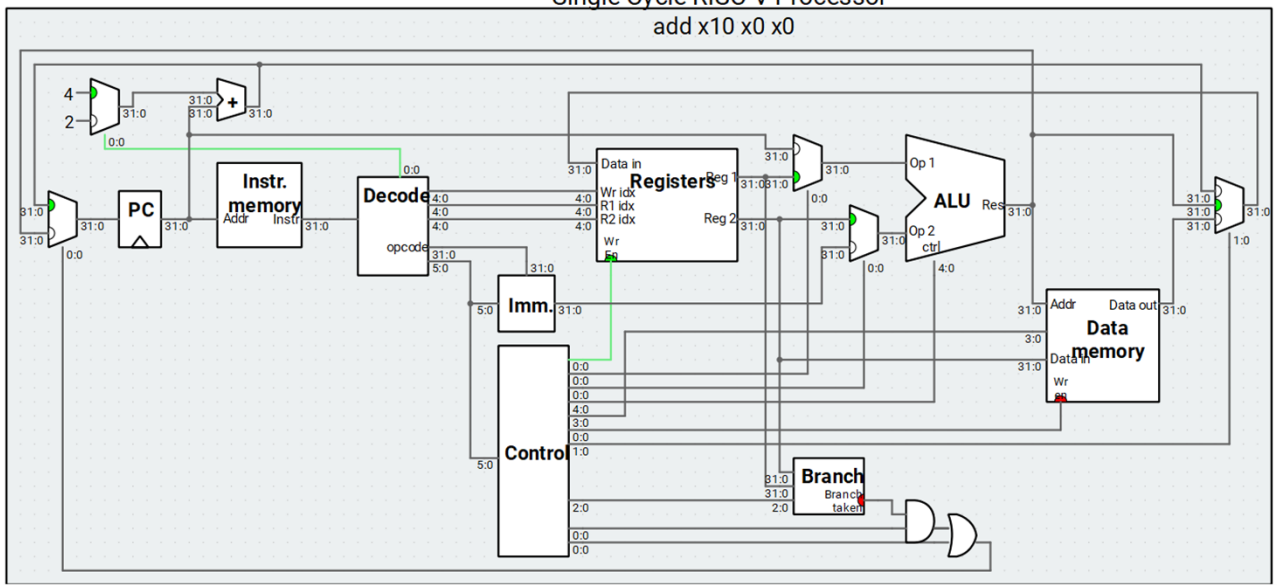


Comparativa disseny risc-v





Single Cycle RISC-V Processor
add x10 x0 x0

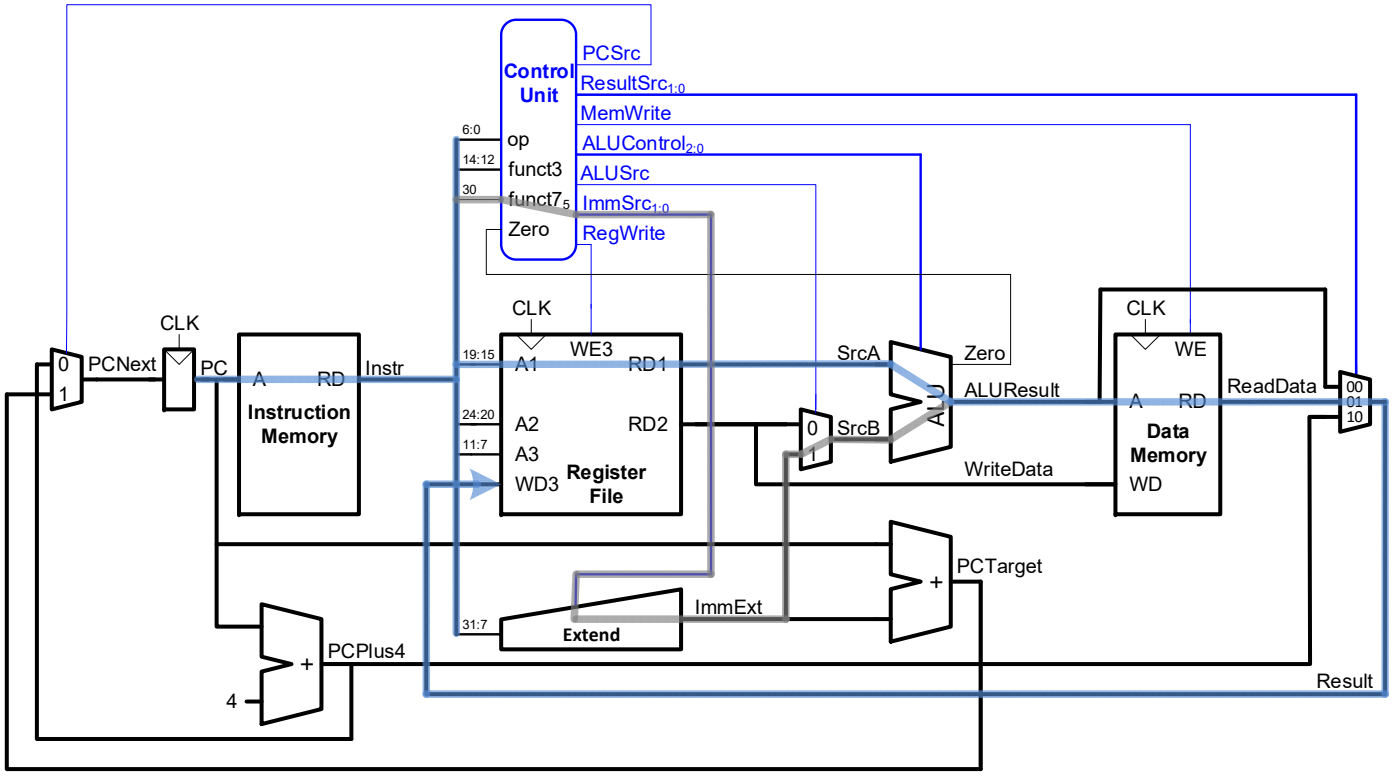


Processor Performance

Program Execution Time

= (#instructions)(cycles/instruction)(seconds/cycle)

= # instructions x CPI x T_C



T_C limitat pel camí més critic (`lw`)

Single-Cycle Processor Performance

- Single-cycle critical path:

$$T_{c\_single} = t_{pcq\_PC} + t_{\text{mem}} + \max[t_{RF\text{read}}, t_{dec} + t_{ext} + t_{\text{mux}}] + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$

- Typically, limiting paths are:
  - memory, ALU, register file
  - *So,*

$$\begin{aligned} T_{c\_single} &= t_{pcq\_PC} + t_{\text{mem}} + t_{RF\text{read}} + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}} \\ &= t_{pcq\_PC} + 2t_{\text{mem}} + t_{RF\text{read}} + t_{\text{ALU}} + t_{\text{mux}} + t_{RF\text{setup}} \end{aligned}$$

Single- vs. Multicycle Processor

- **Single-cicle:**
  - + simple
  - - Temps de cicle limitat per la instrucció més llarga (`lw`)
  - - Memòries separades per instruccions i dades
  - - 3 adders/ALUs
- **Multi-cicle:**
  - + Velocitat de clk més elevada!
  - + Les instruccions més senzilles corren més ràpides en execució
  - + reutilitzem hardware fent servir múltiples cicles
  - - El seqüenciament de les capçaleres implica una pèrdua de temps en l'execució multi-cicle

Mateixos passos que en single-cycle:

- **Primer datapath**
- **Després control**

Multicycle Risc-v processor

Multiycle RISC-V Processor

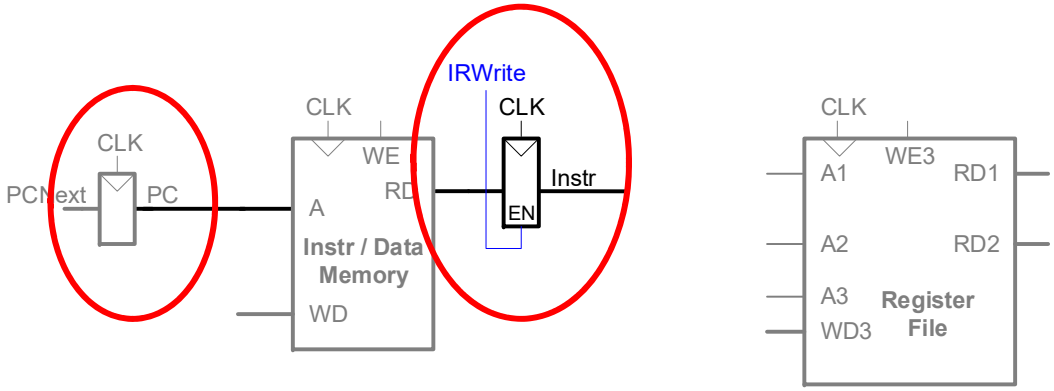
- Disseny del camí de dades (igual que abans)
- Exemple d'un programa executant-se

Example Program:

Address	Instruction	Type	Fields						Machine Language
			$imm_{11:0}$		rs1	f3	rd	op	
0x1000	L7: lw x6, -4(x9)	I	111111111100		01001	010	00110	0000011	FFC4A303
			$imm_{11:5}$	rs2	rs1	f3	$imm_{4:0}$	op	
0x1004	sw x6, 8(x9)	S	0000000	00110	01001	010	01000	0100011	0064A423
			funct7	rs2	rs1	f3	rd	op	
0x1008	or x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233
			$imm_{12,10:5}$	rs2	rs1	f3	$imm_{4:1,11}$	op	
0x100C	beq x4, x4, L7	B	1111111	00100	00100	000	10101	1100011	FE420AE3

Multicycle Datapath: Instruction Fetch

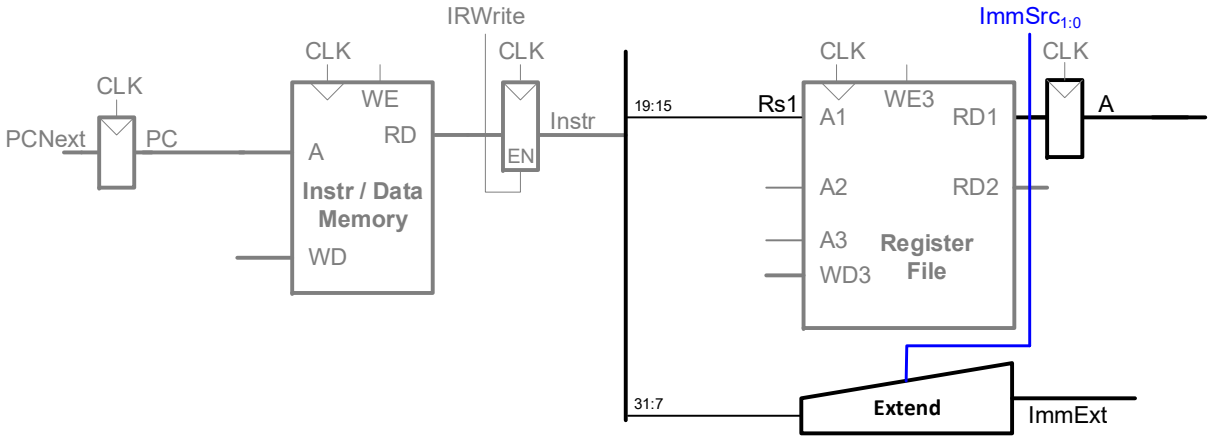
Pas 1: Fetch (un cicle)



lw x6, -4(x9)

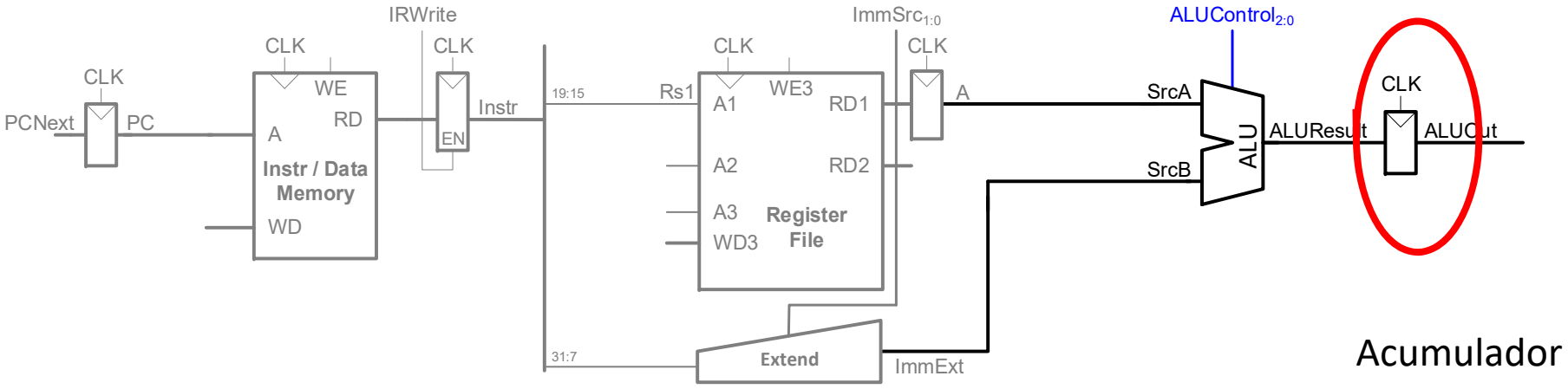
Multicycle Datapath: `lw` Get Sources

Pas 2: Lectura dels operands font del conjunt de registres i extenent l'immediat (1 cicle)



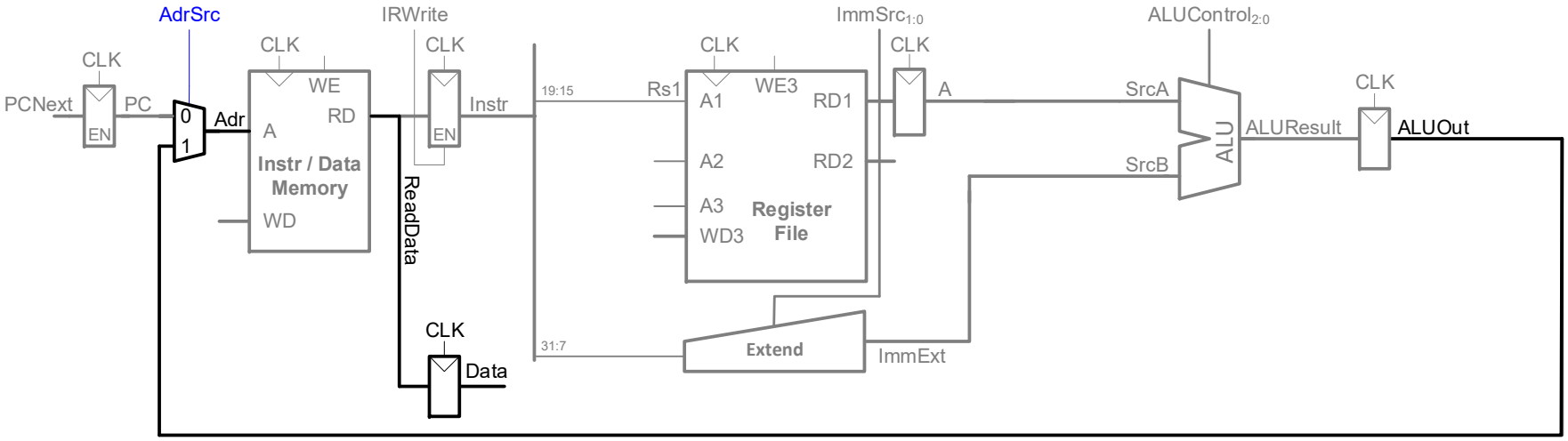
Multicycle Datapath: `lw` Address

Pas 3: Calculem l'adreça de memòria a la que volem accedir (1 cicle)



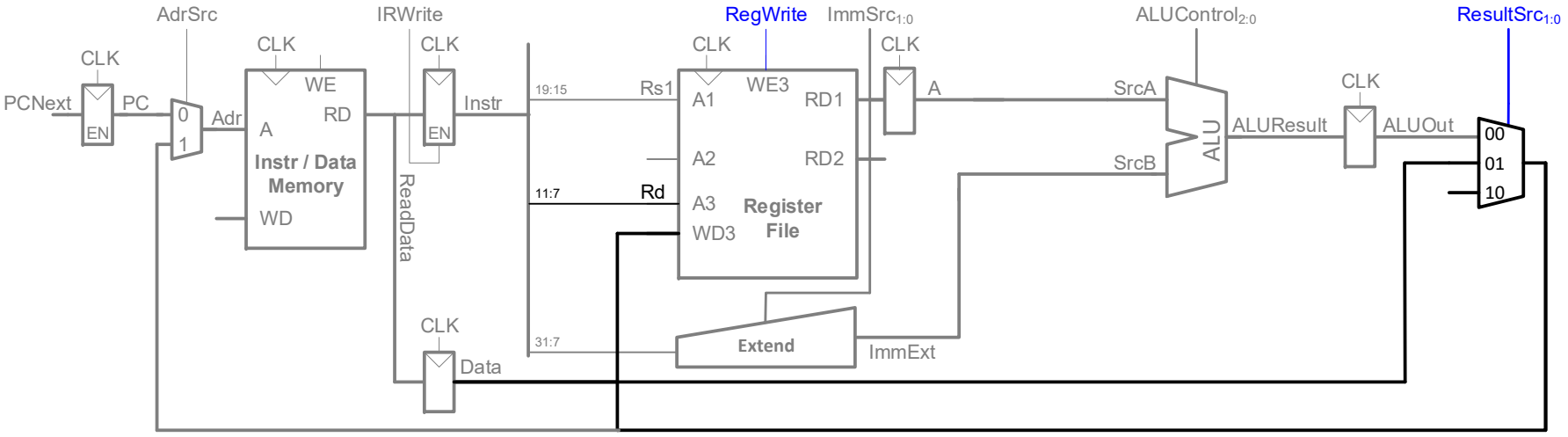
Multicycle Datapath: `lw` Memory Read

Pas 4: Lectura de dades de memòria (1 cicle)



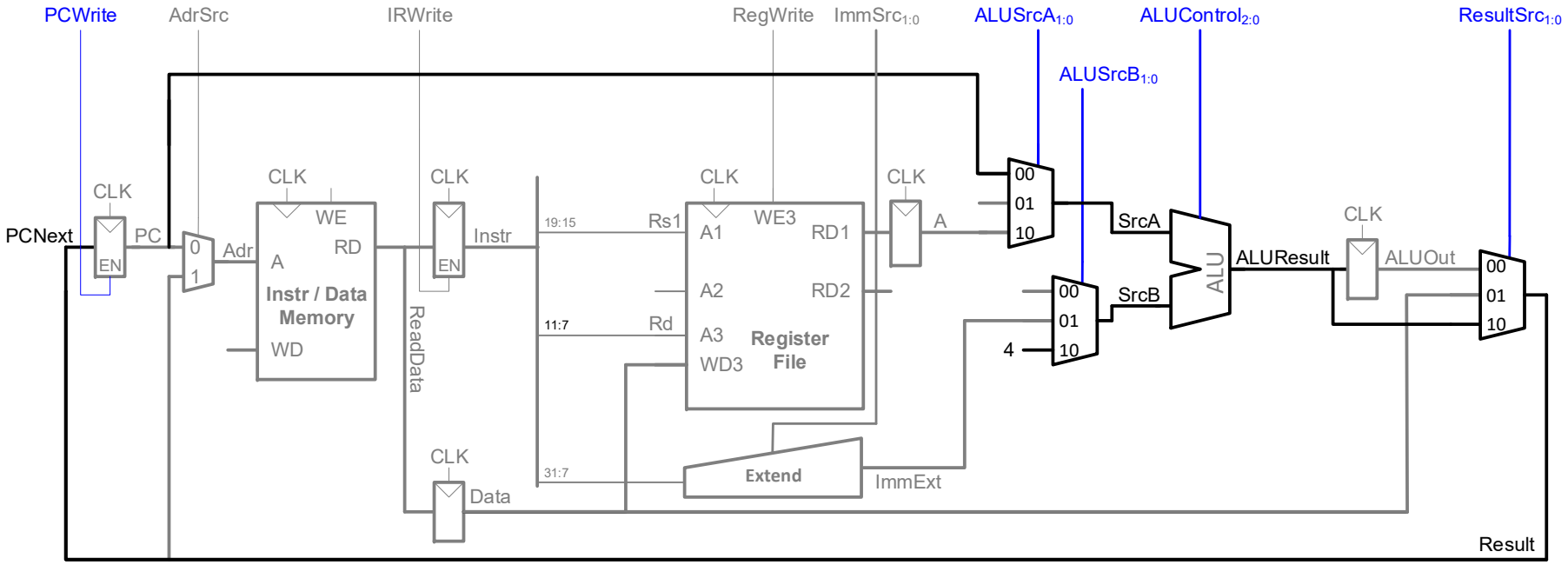
Multicycle Datapath: `lw` Write Register

Pas 5: Escriptura de la dada al registre destí (1 cicle)



Multicycle Datapath: Increment PC

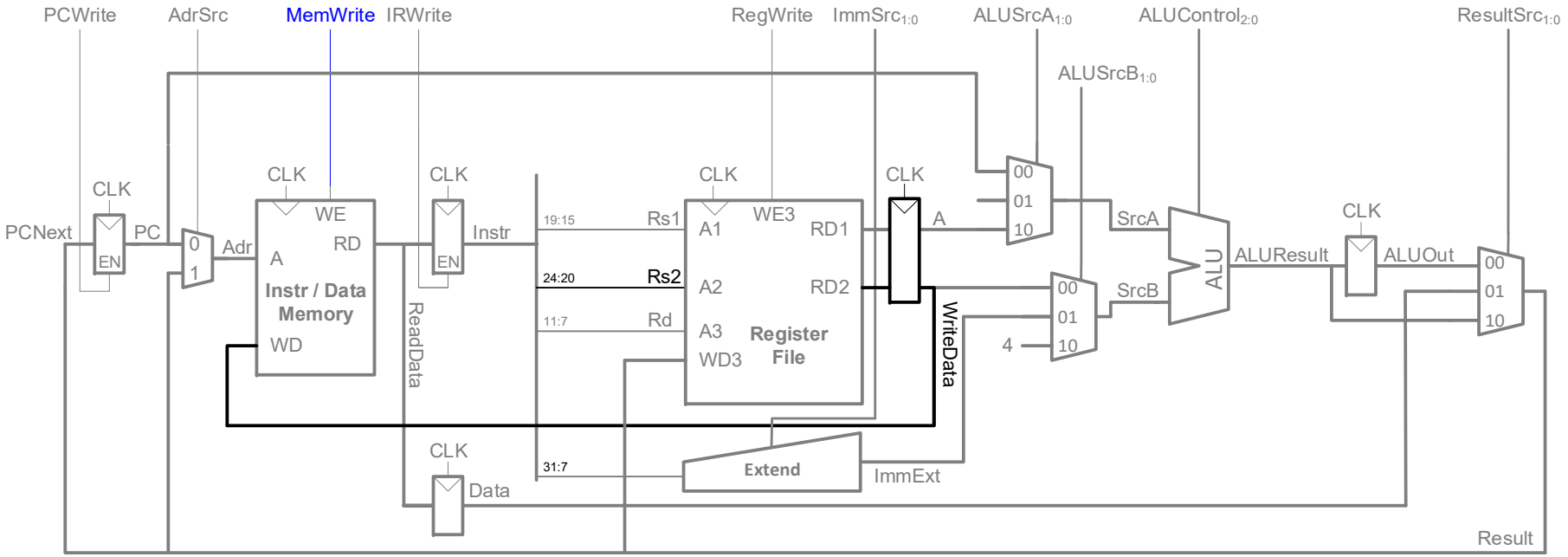
Pas 6: Incrementem PC: PC = PC+4 (es pot incloure en l'anterior cicle si posem un sumador extra)



Multicycle Datapath: `sw`

Escriptura de dades de **rs2** a memòria

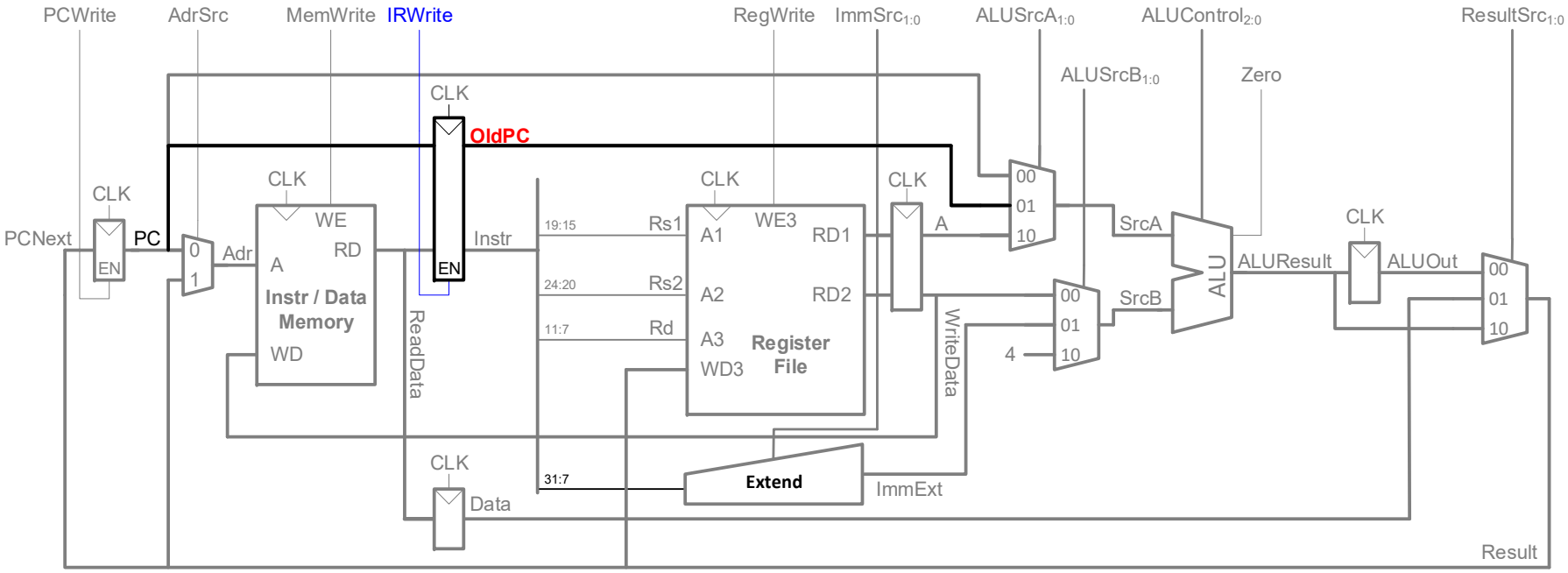
(aquesta és l'etapa que faltaria. Com seria l'execució complerta?)



Multicycle Datapath: beq

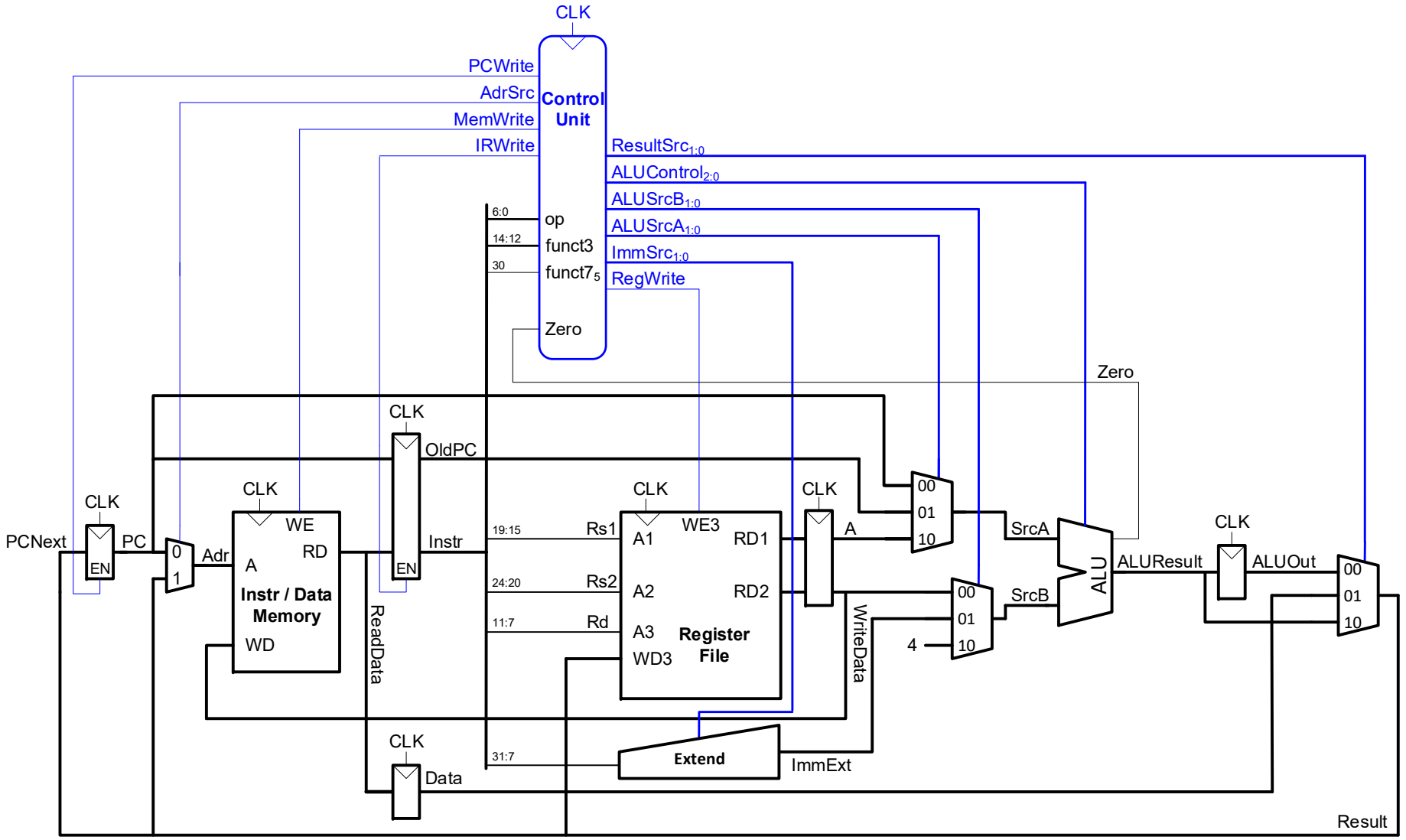
Càlcul Adreça de Salt associada al Brach:

ASB = PC + imm



PC l'actualitzem a la fase de Fetch stage, cal guardar l'antic **(actual) PC**

Multicycle RISC-V Processor

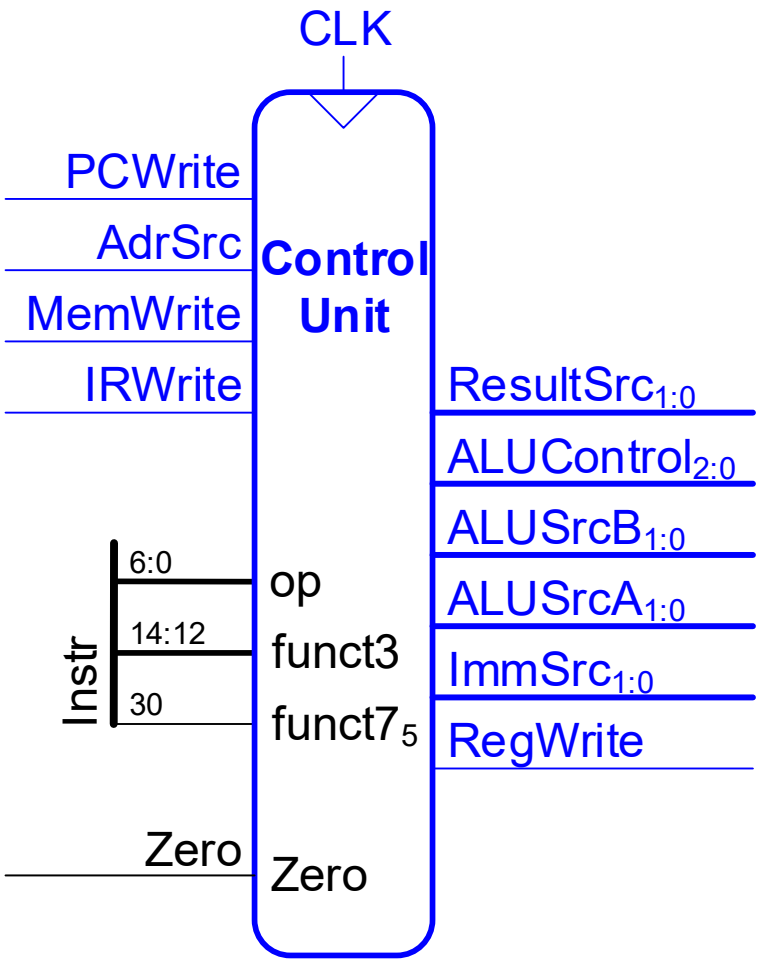


Multicycle Risc-V Processor

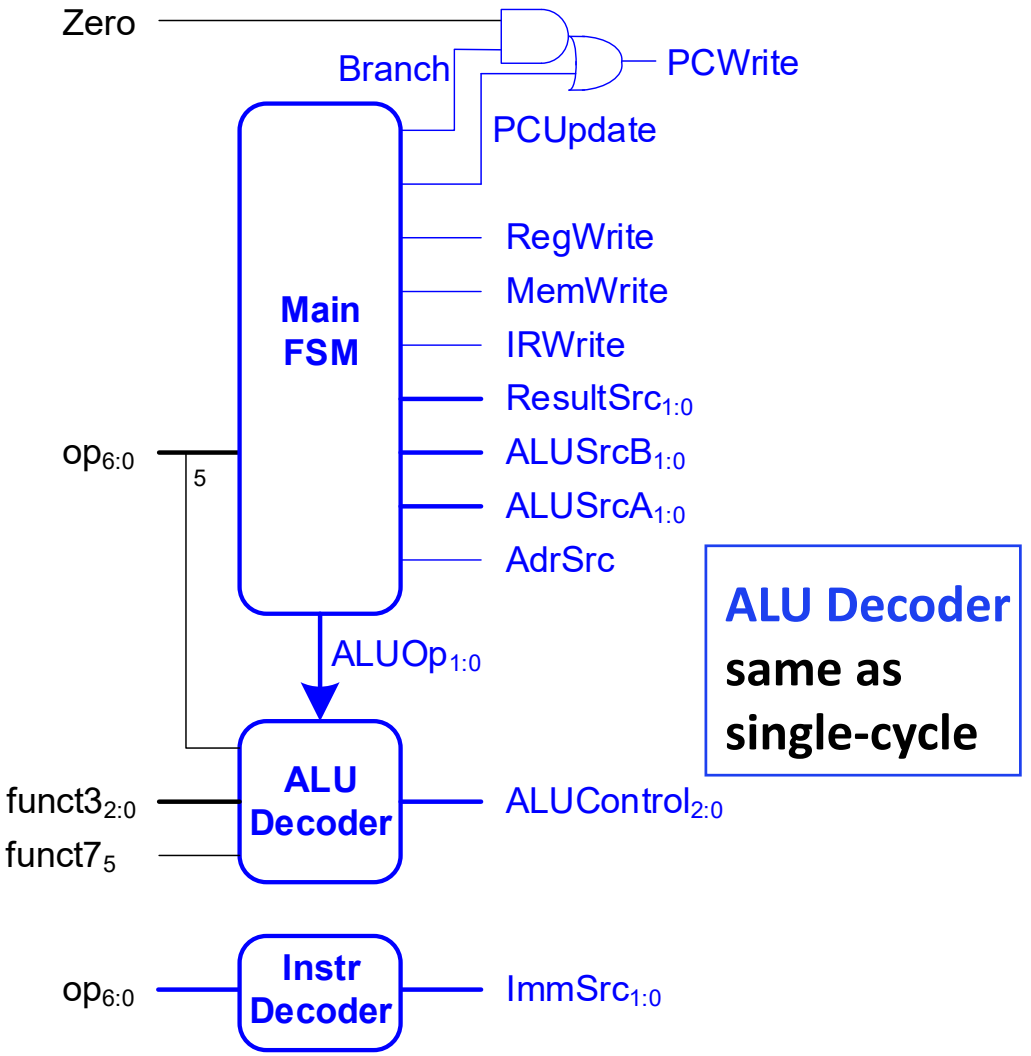
Unitat de Control del procesador multi-cicle

Multicycle RISC-V Processor

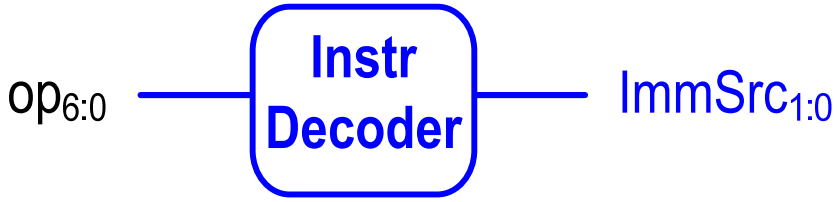
High-Level View



Low-Level View

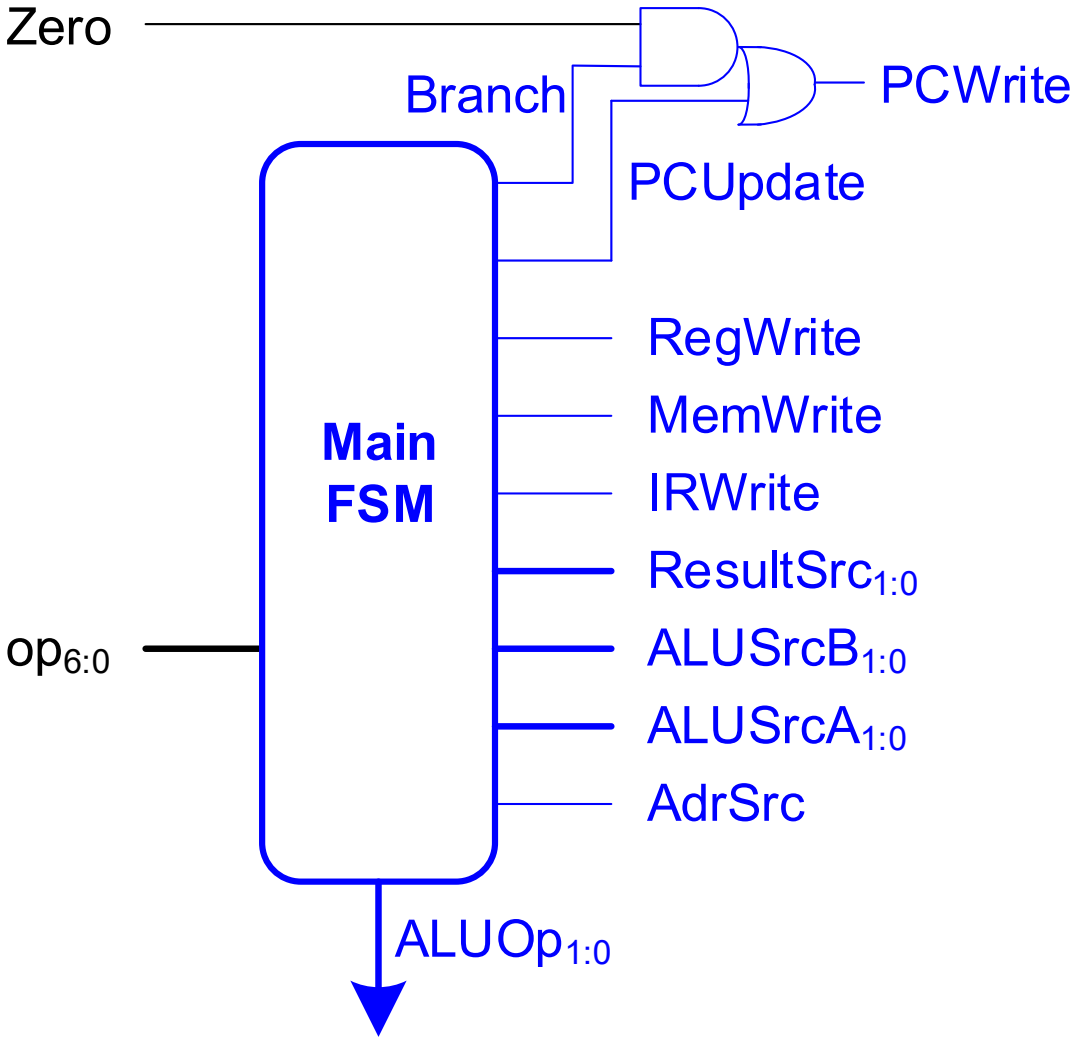


Multicycle RISC-V Processor



op	Instruction	ImmSrc
3	**`lw`**	00
35	**`sw`**	01
51	R-type	XX
99	**`beq`**	10

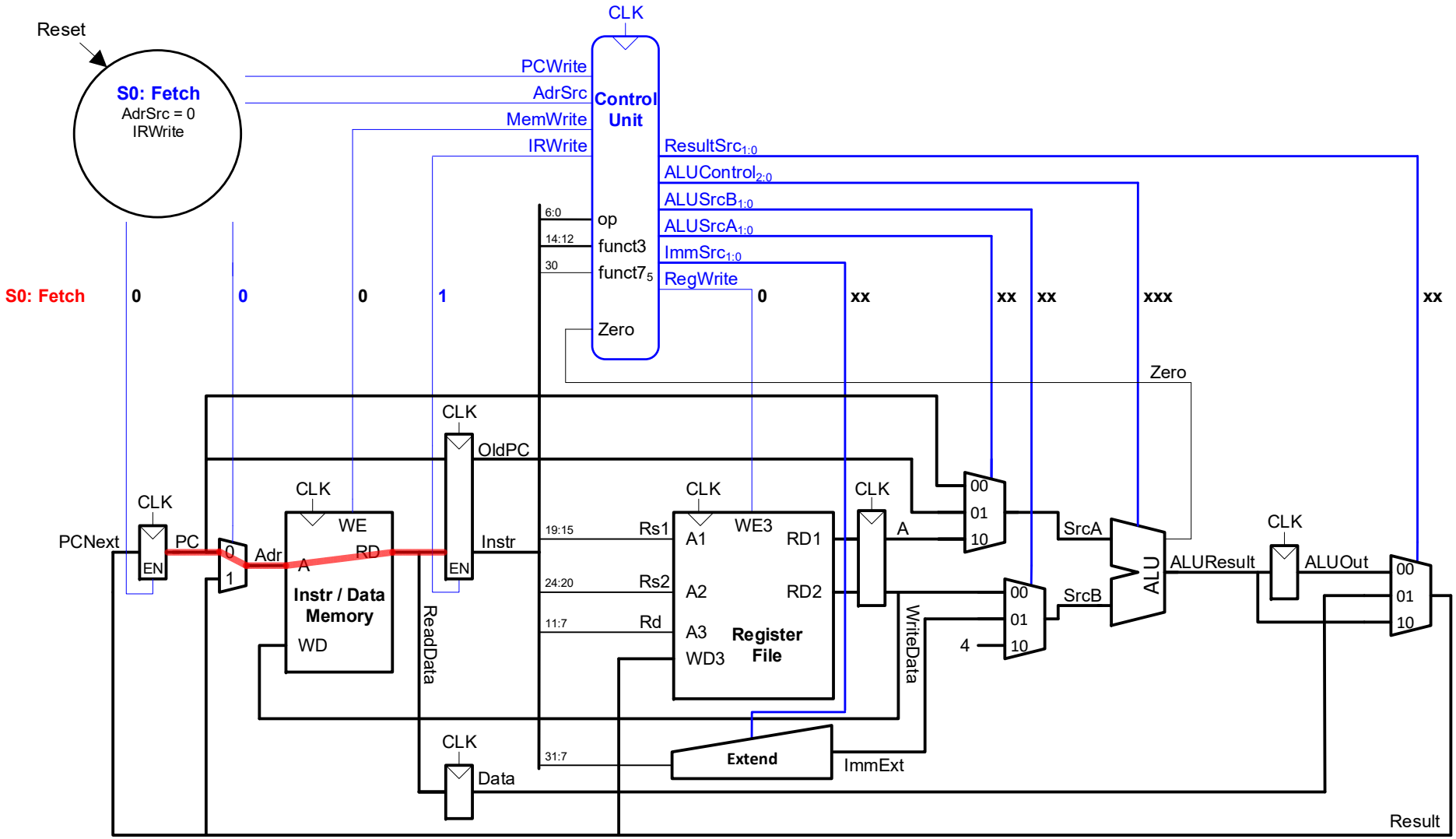
Multicycle RISC-V Processor: FSM



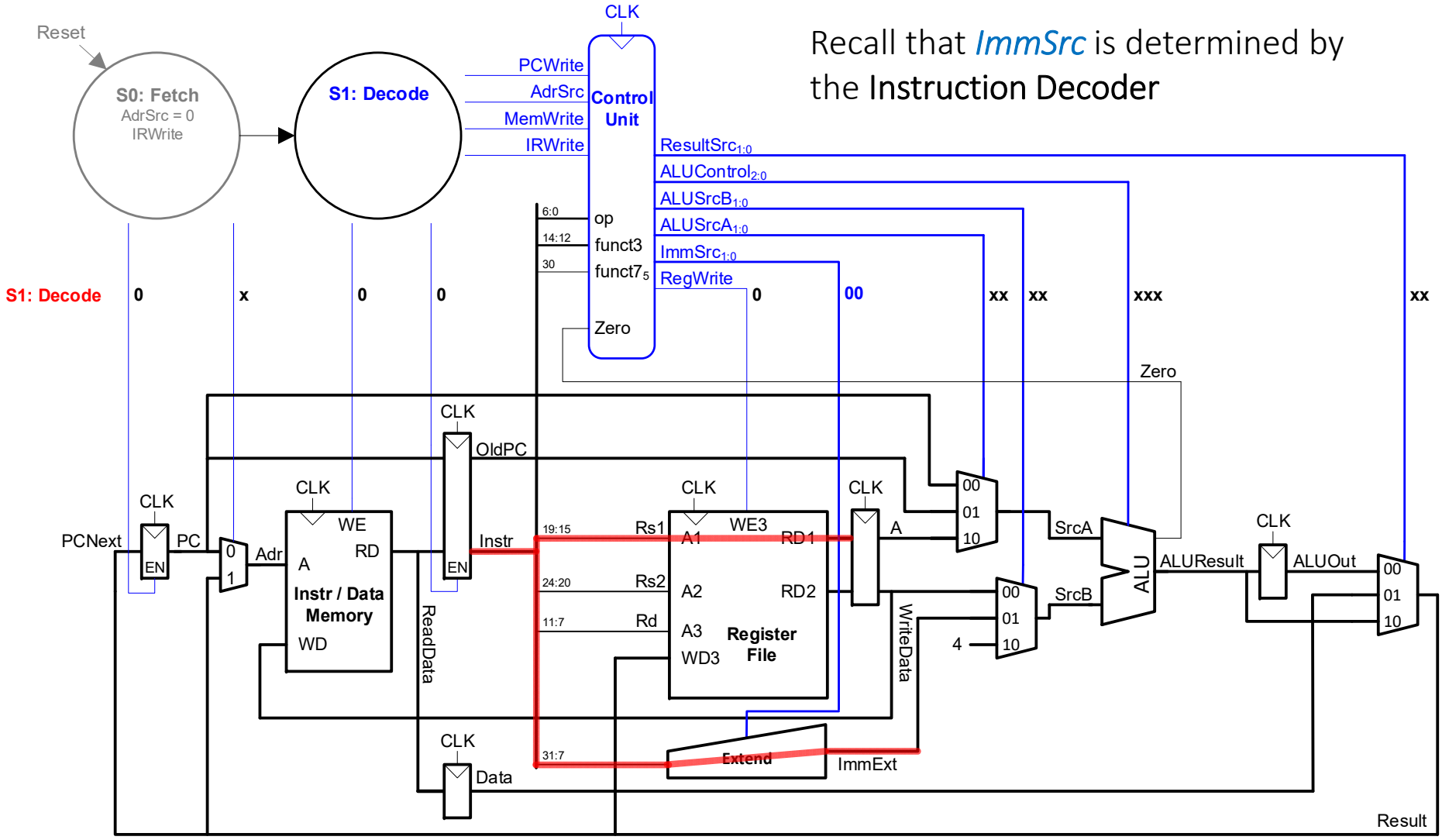
Posant ordre a la FSM:

- Senyals de Write enable (RegWrite, MemWrite, IRWrite, PCUpdate, i Branch) són 0 si no apareixen a l'estat.
- Altres senyals no importa el seu valor si no apareixen a l'estat

Multicycle RISC-V Processor. FETCH

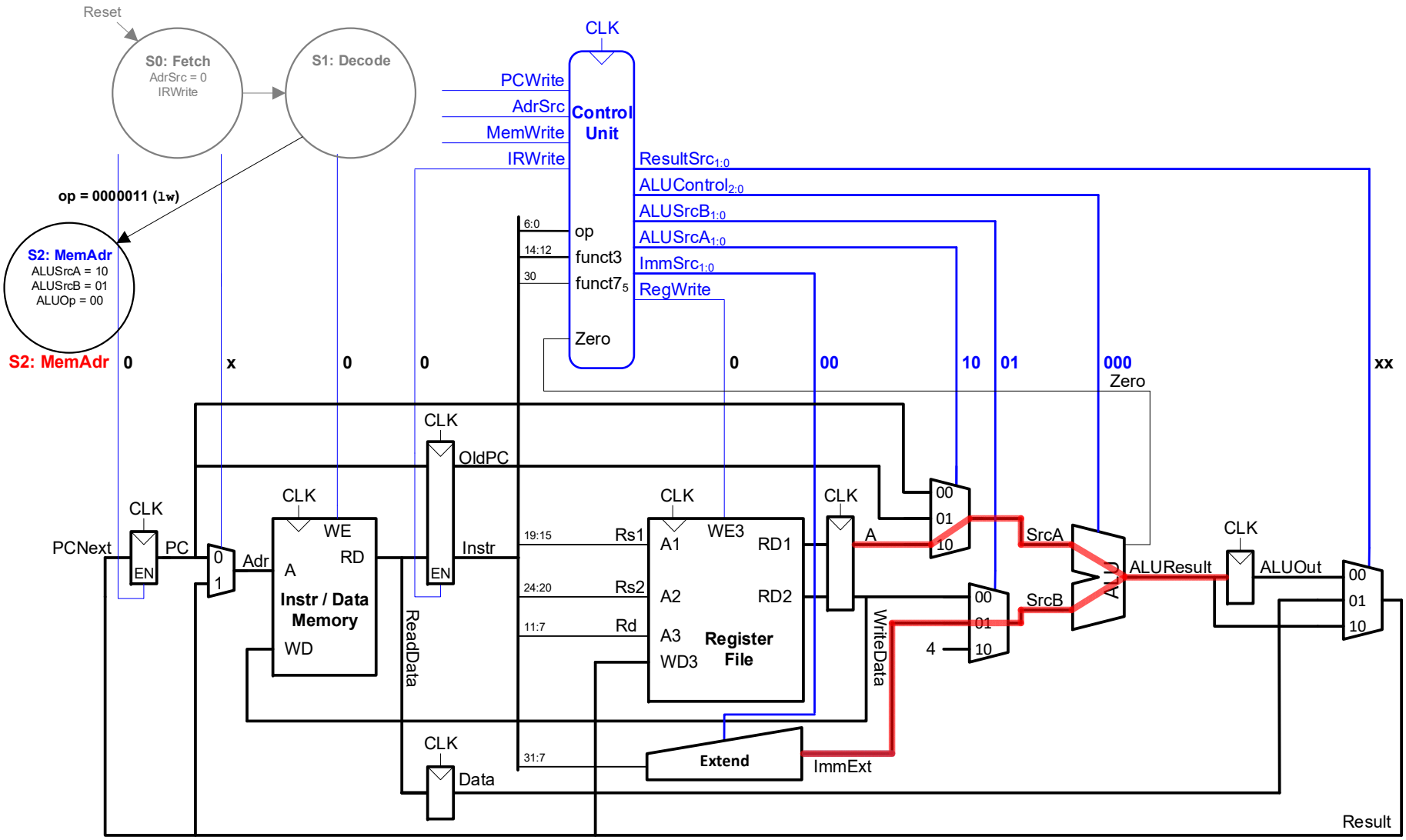


Multicycle RISC-V Processor. DEC

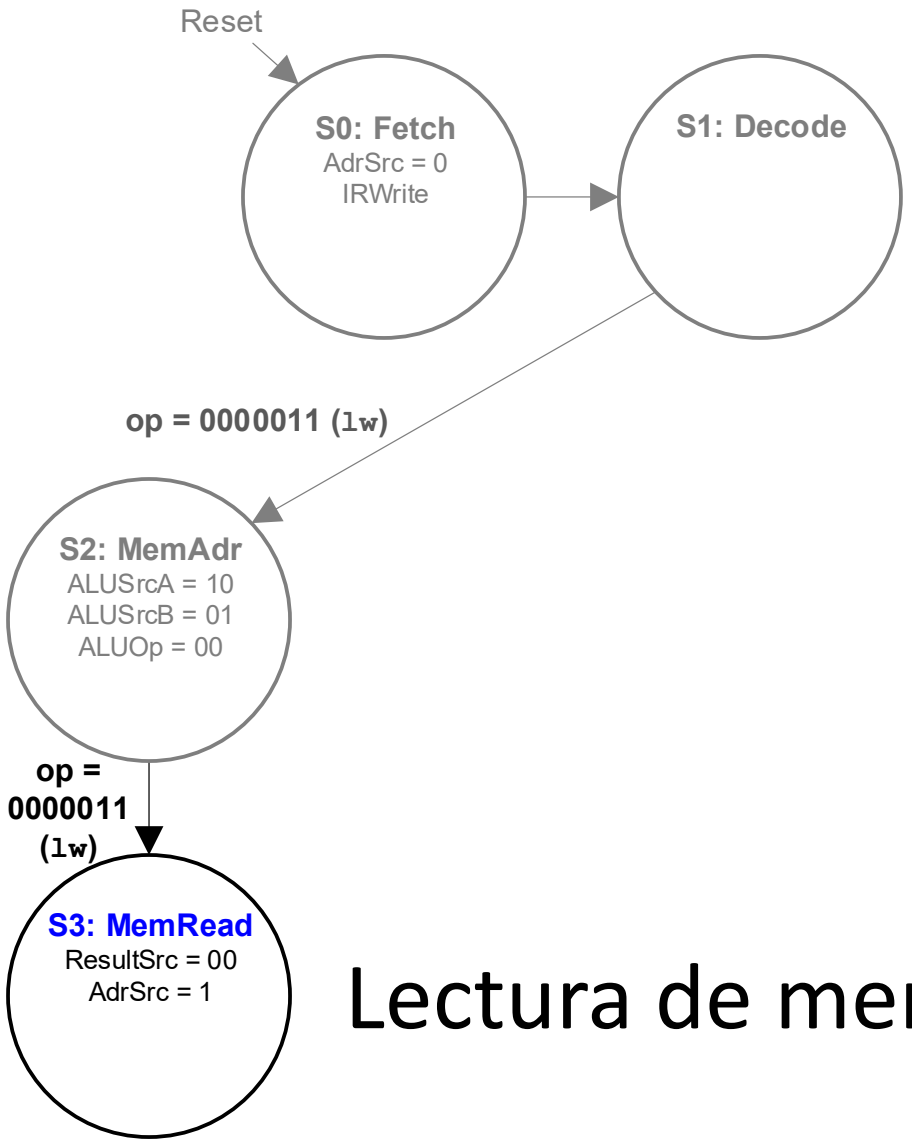


Recall that *ImmSrc* is determined by the **Instruction Decoder**

Multicycle RISC-V Processor. Adress

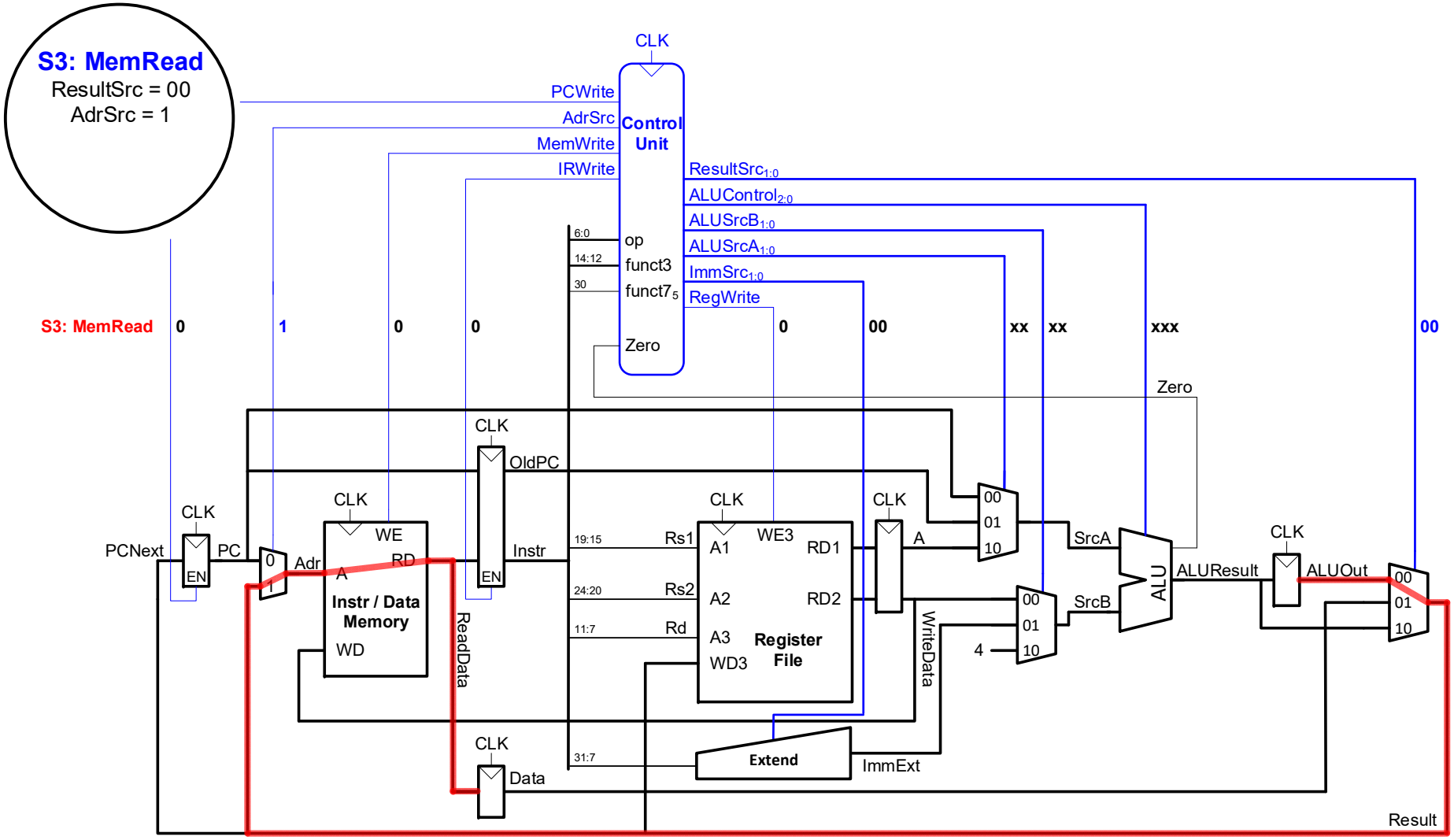


Multicycle RISC-V Processor

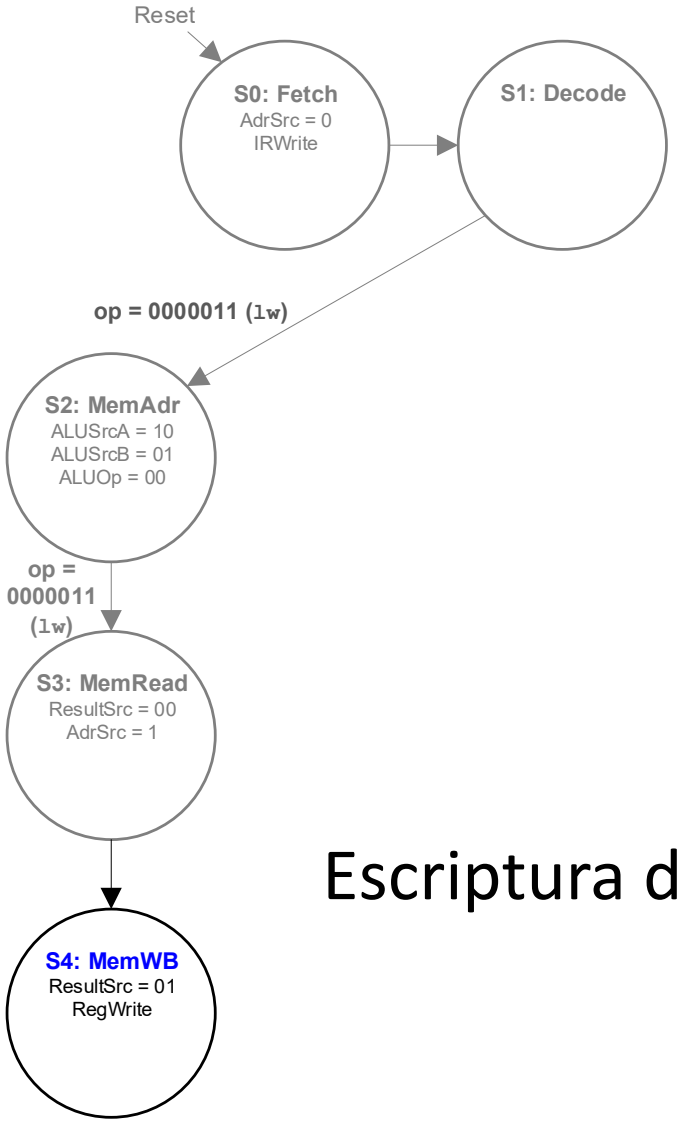


Lectura de memòria

Multicycle RISC-V Processor

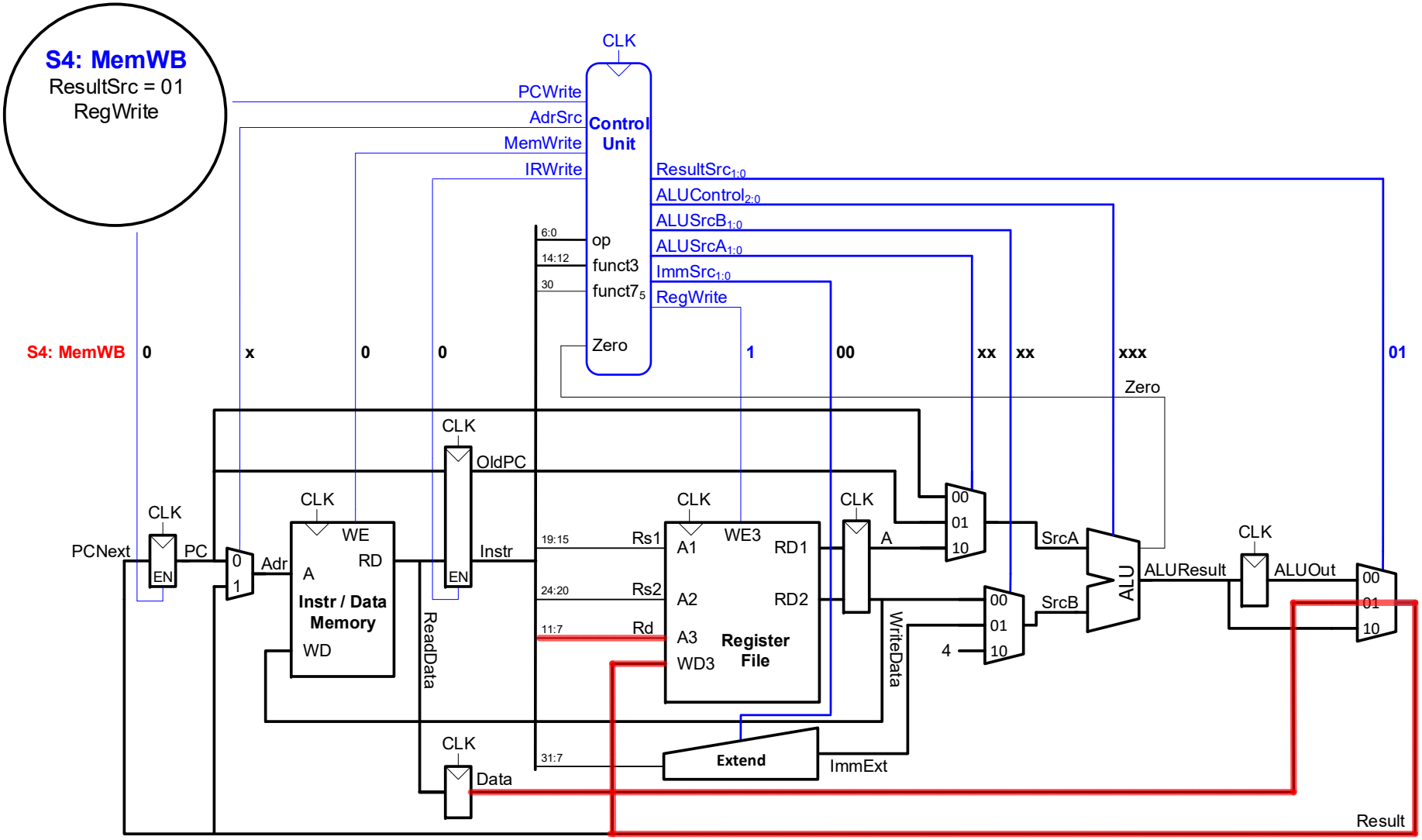


Multicycle RISC-V Processor



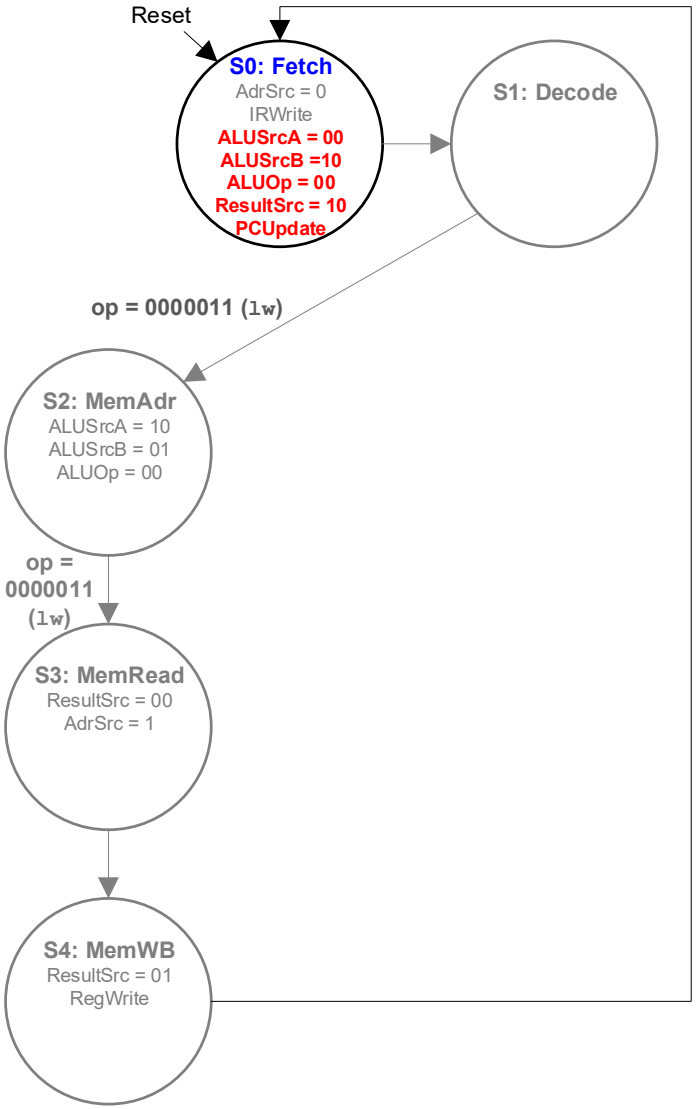
Escriptura de registre

Multicycle RISC-V Processor



Multicycle RISC-V Processor

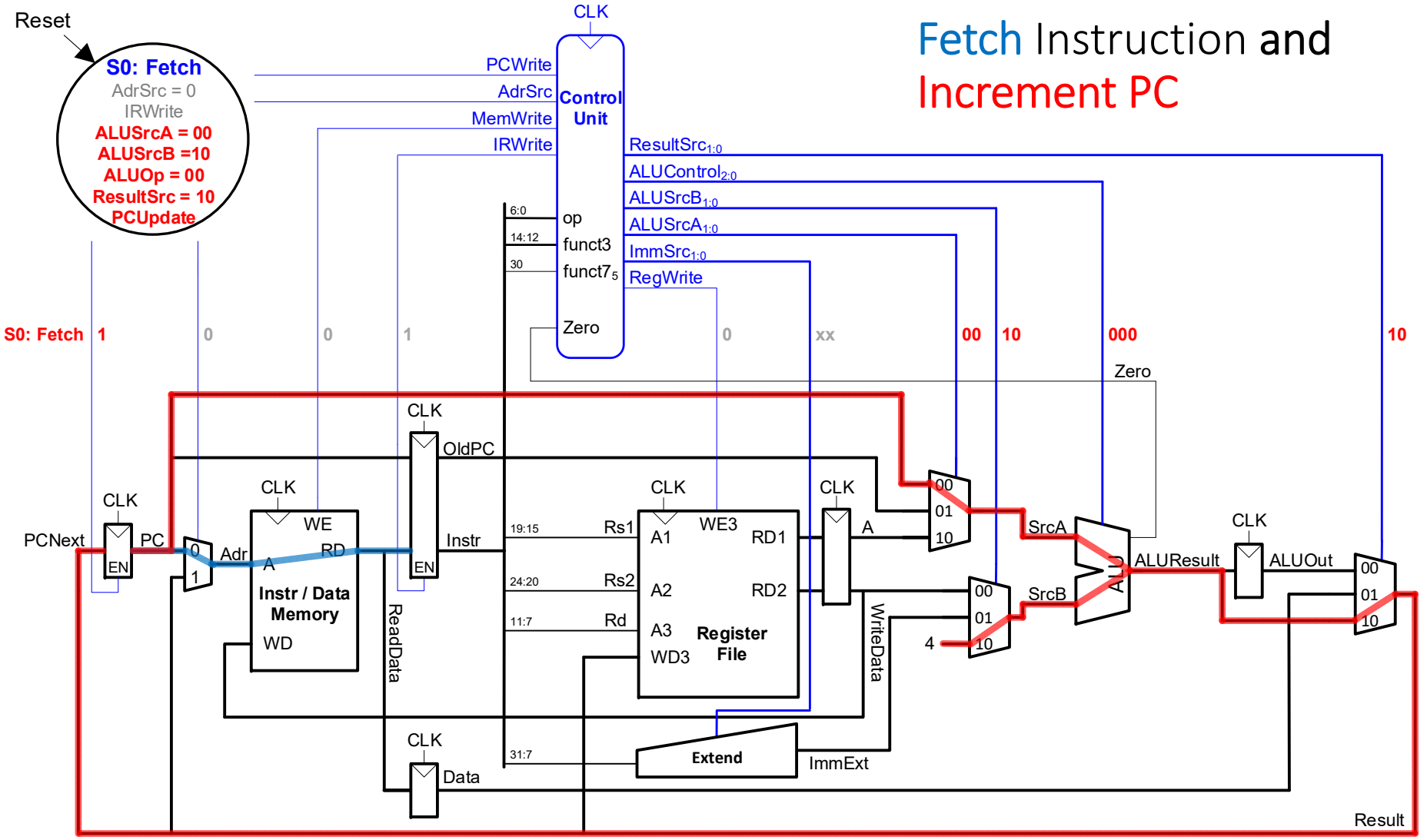
Calculate PC+4 during Fetch stage (ALU isn't being used)



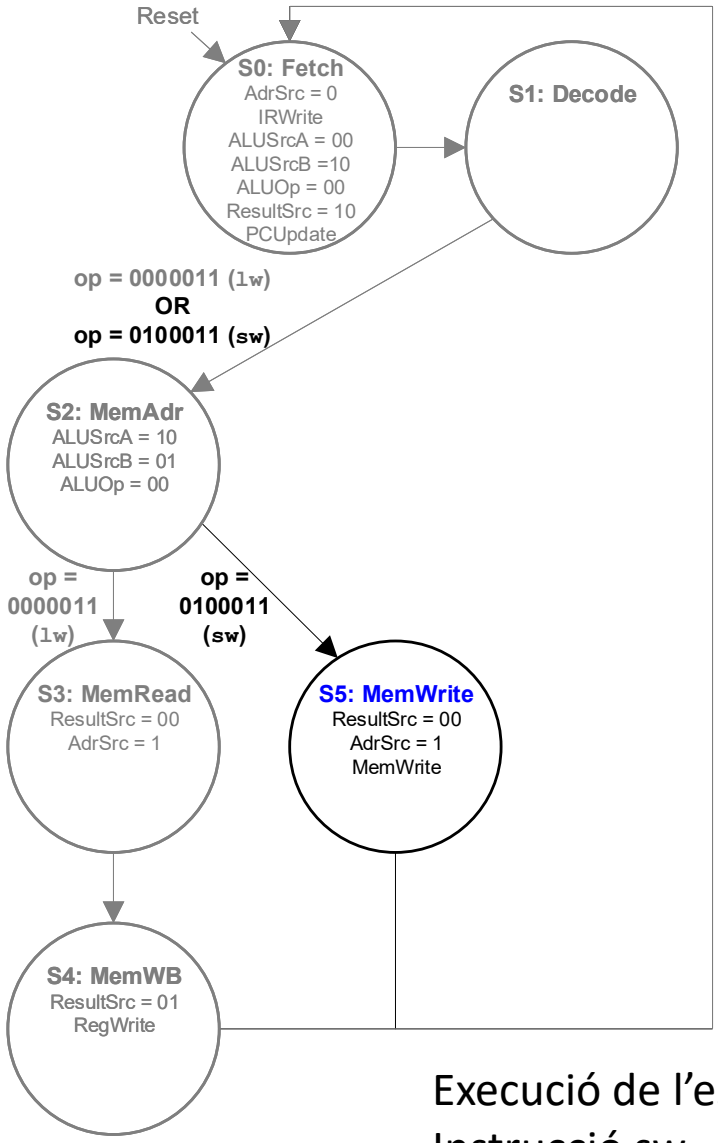
Recàlcul del FETCH

Multicycle RISC-V Processor

Fetch Instruction and Increment PC

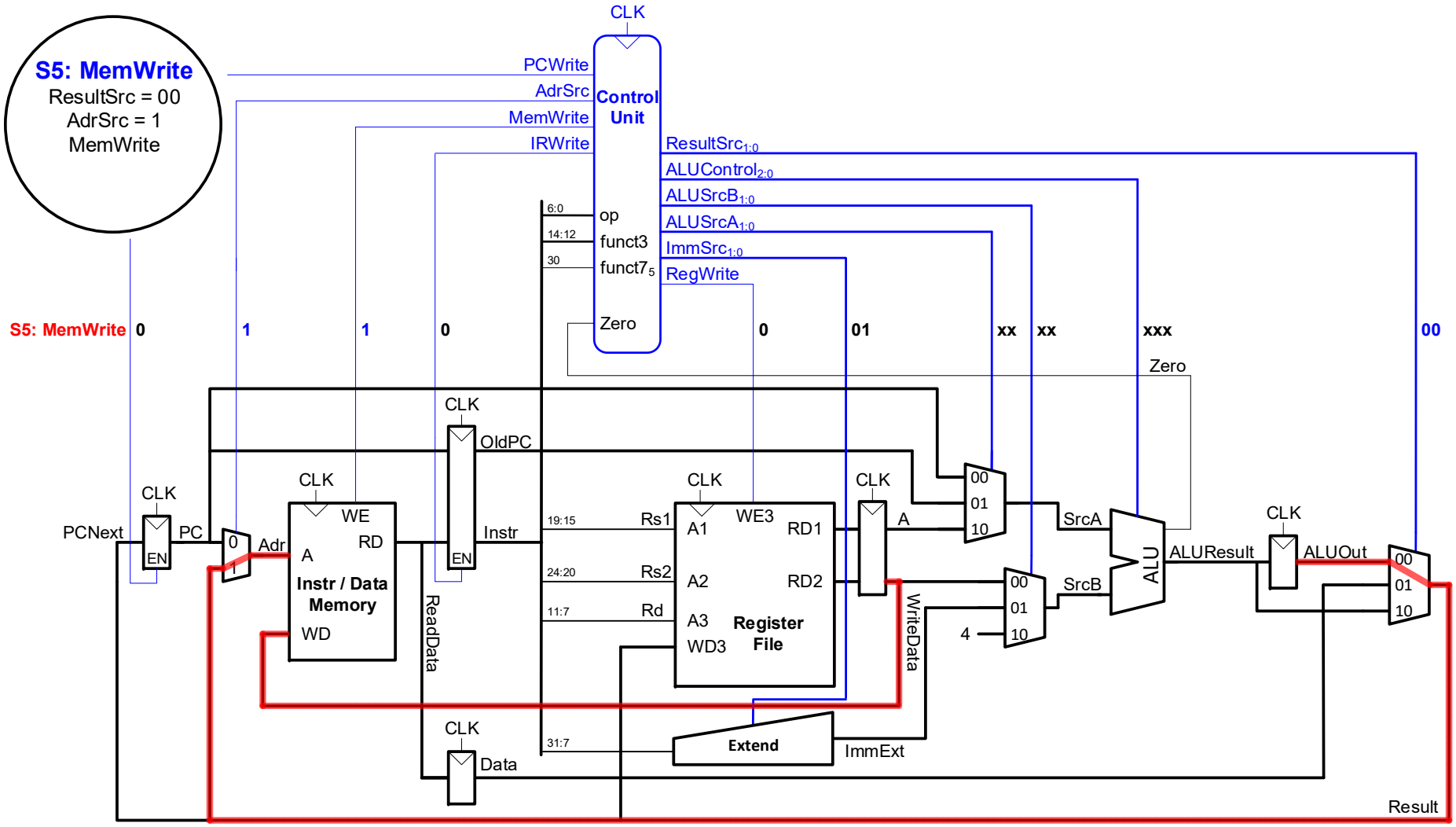


Multicycle RISC-V Processor. sw

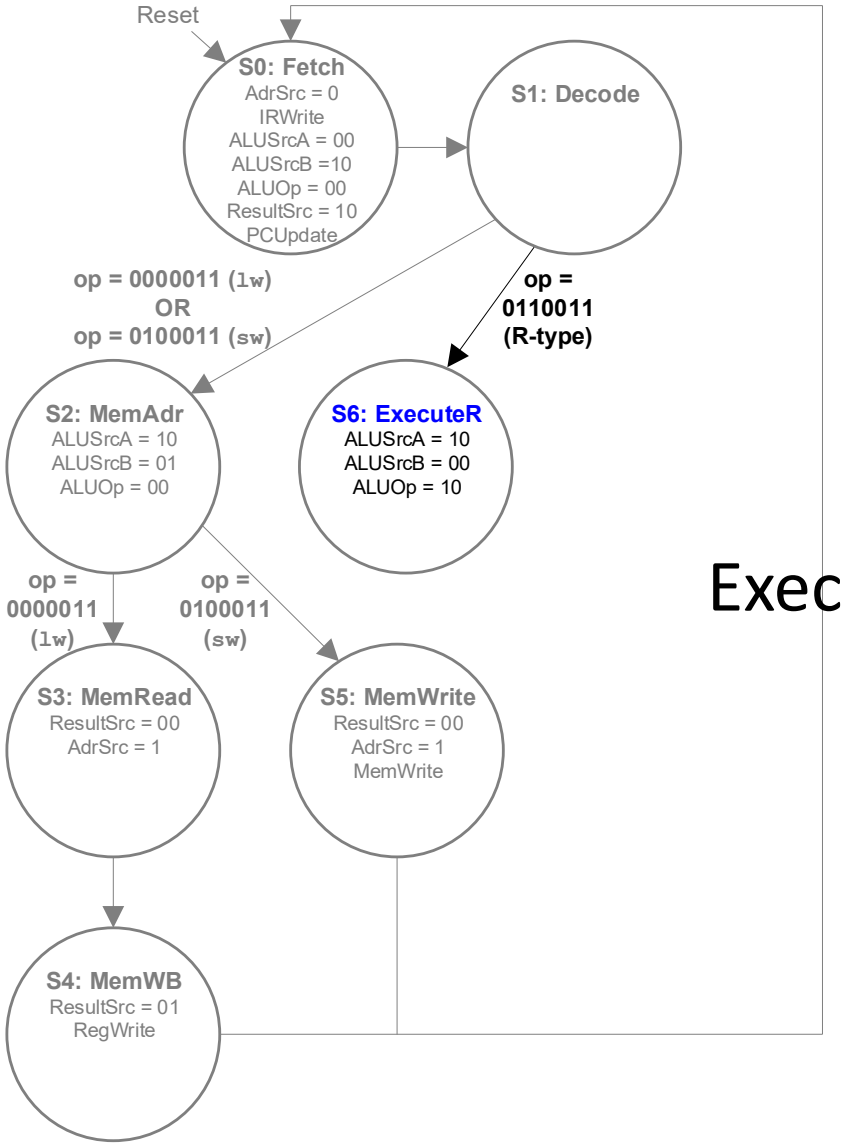


Execució de l'escriptura en Memòria
Instrucció sw

Multicycle RISC-V Processor

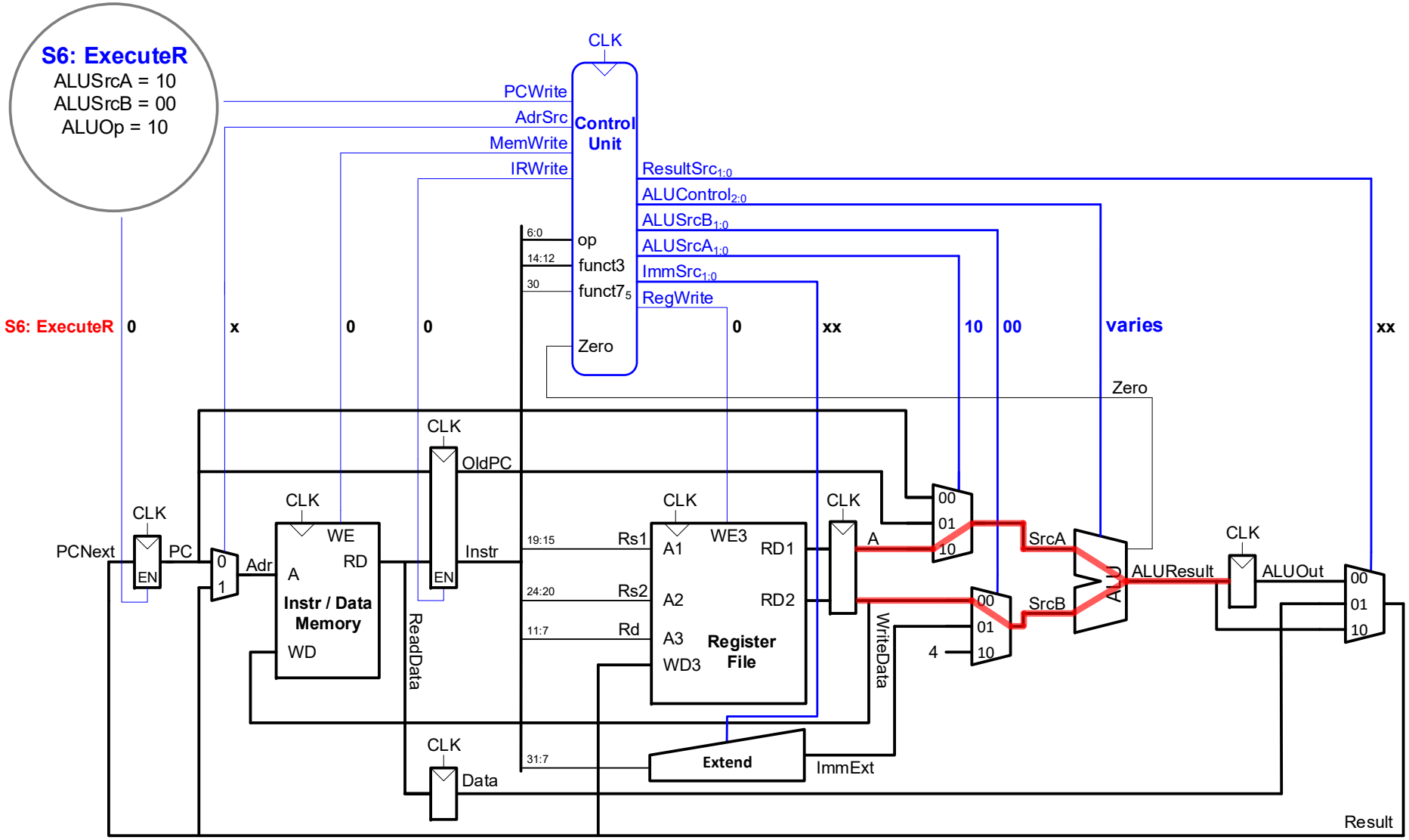


Multicycle RISC-V Processor. R-type

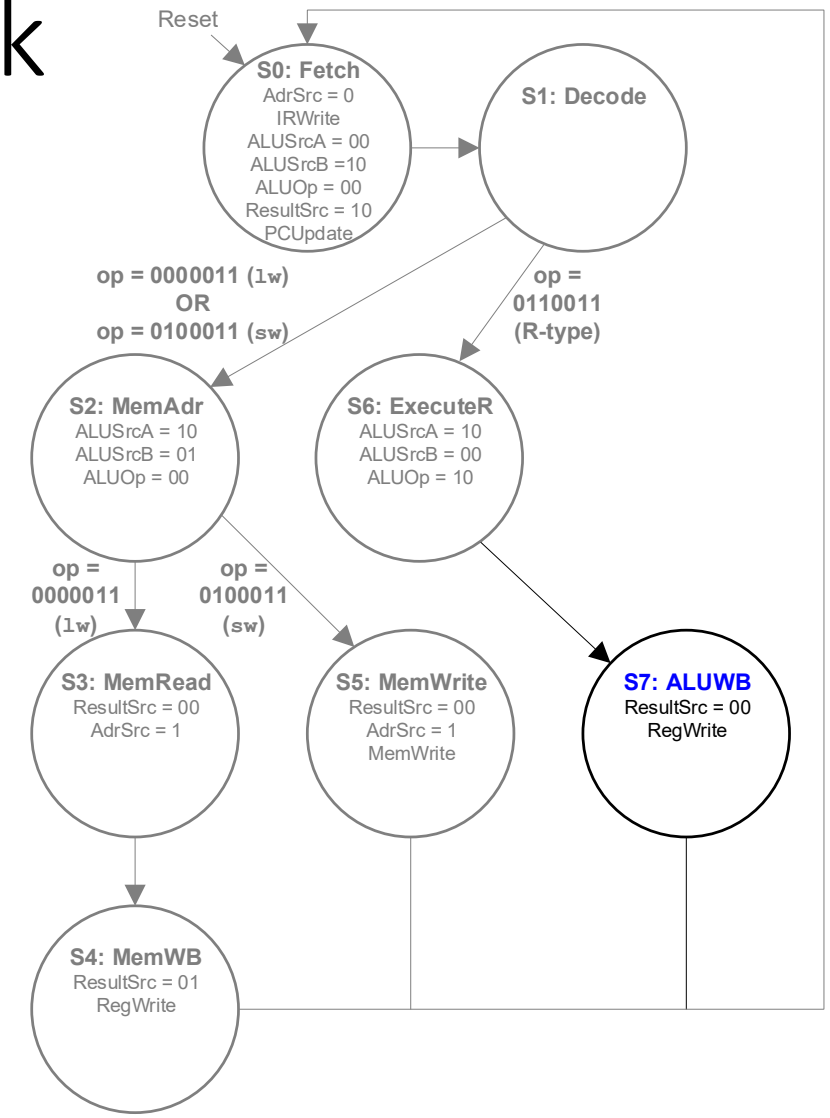


Execució

Multicycle RISC-V Processor

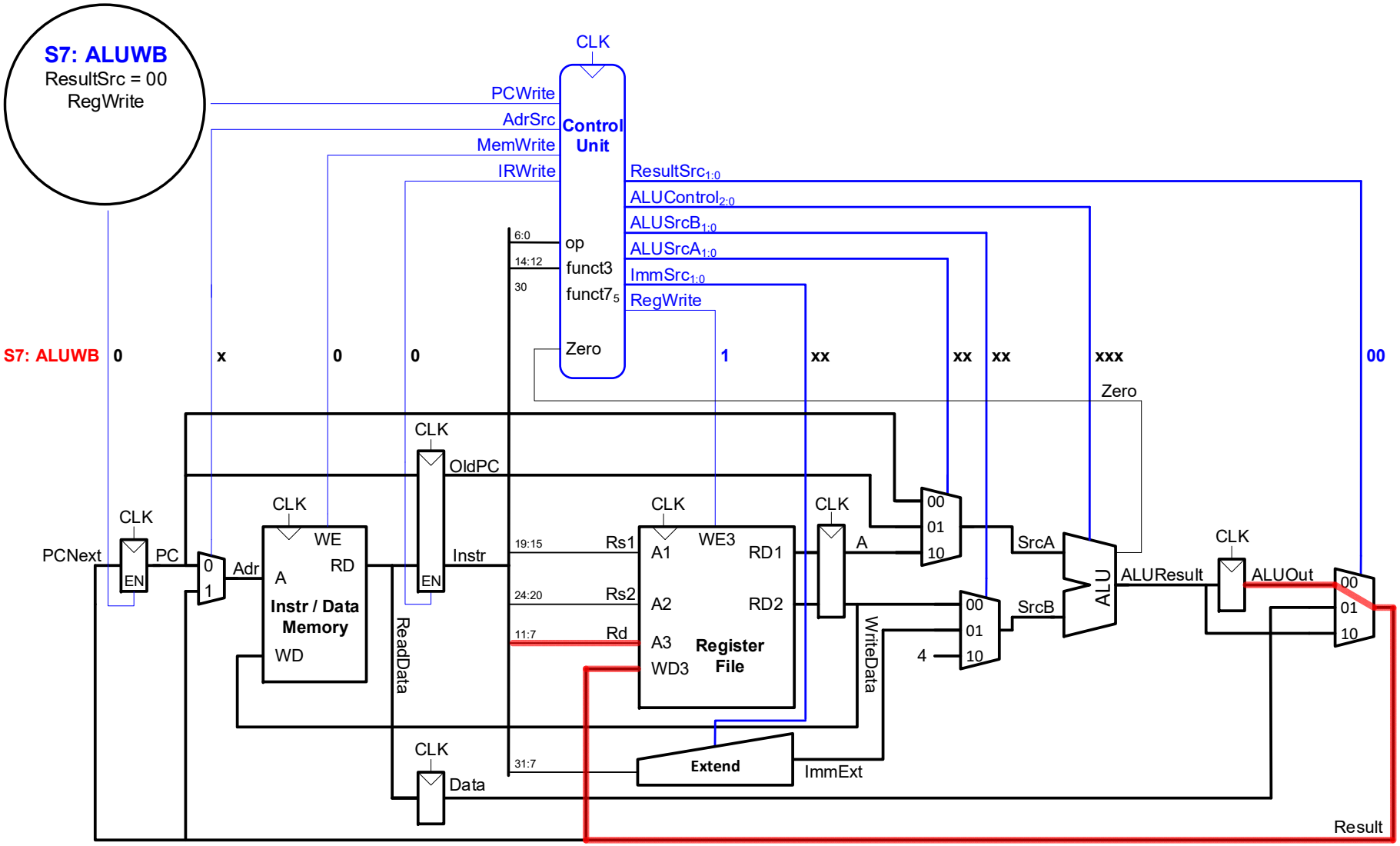


Multicycle RISC-V Processor. R-type write back



Escriptura en el registre destí

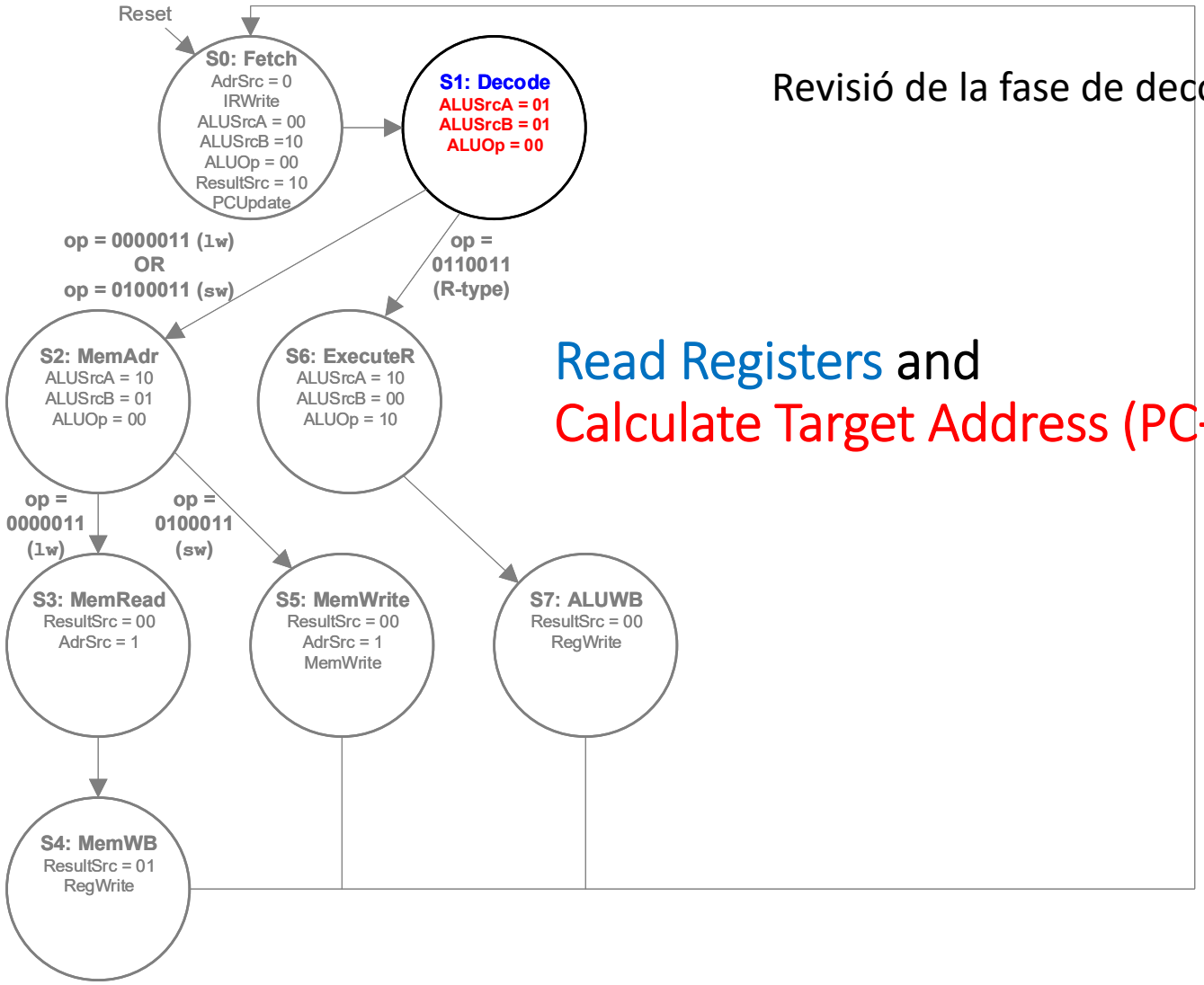
Multicycle RISC-V Processor



Multicycle RISC-V Processor. beq

- **Need to calculate:**
  - Branch Target Address
  - **rs1** - **rs2** (to see if equal)
- **ALU** isn't being used in Decode stage
  - Use it to calculate Target Address (PC + imm)

Multicycle RISC-V Processor

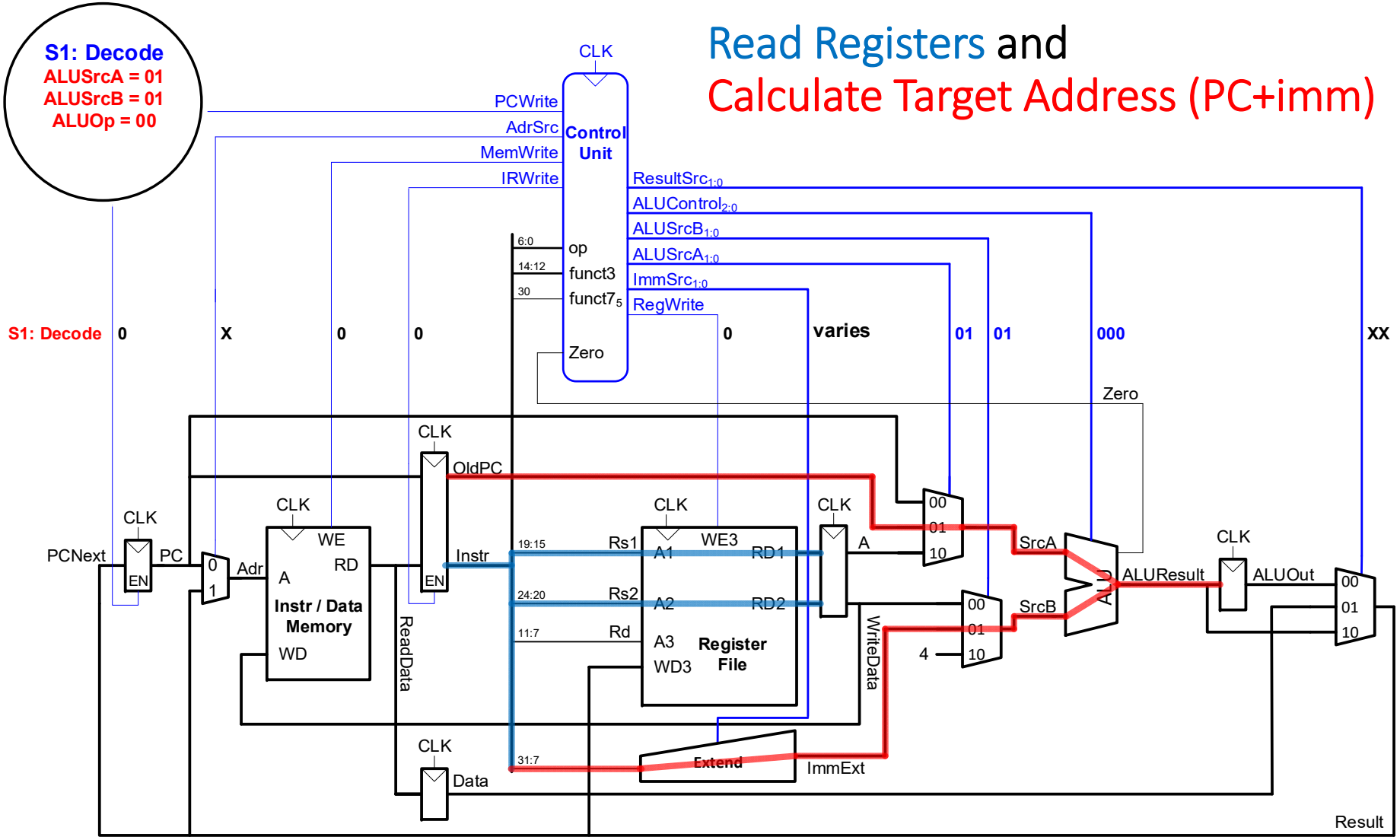


Revisió de la fase de decodificació

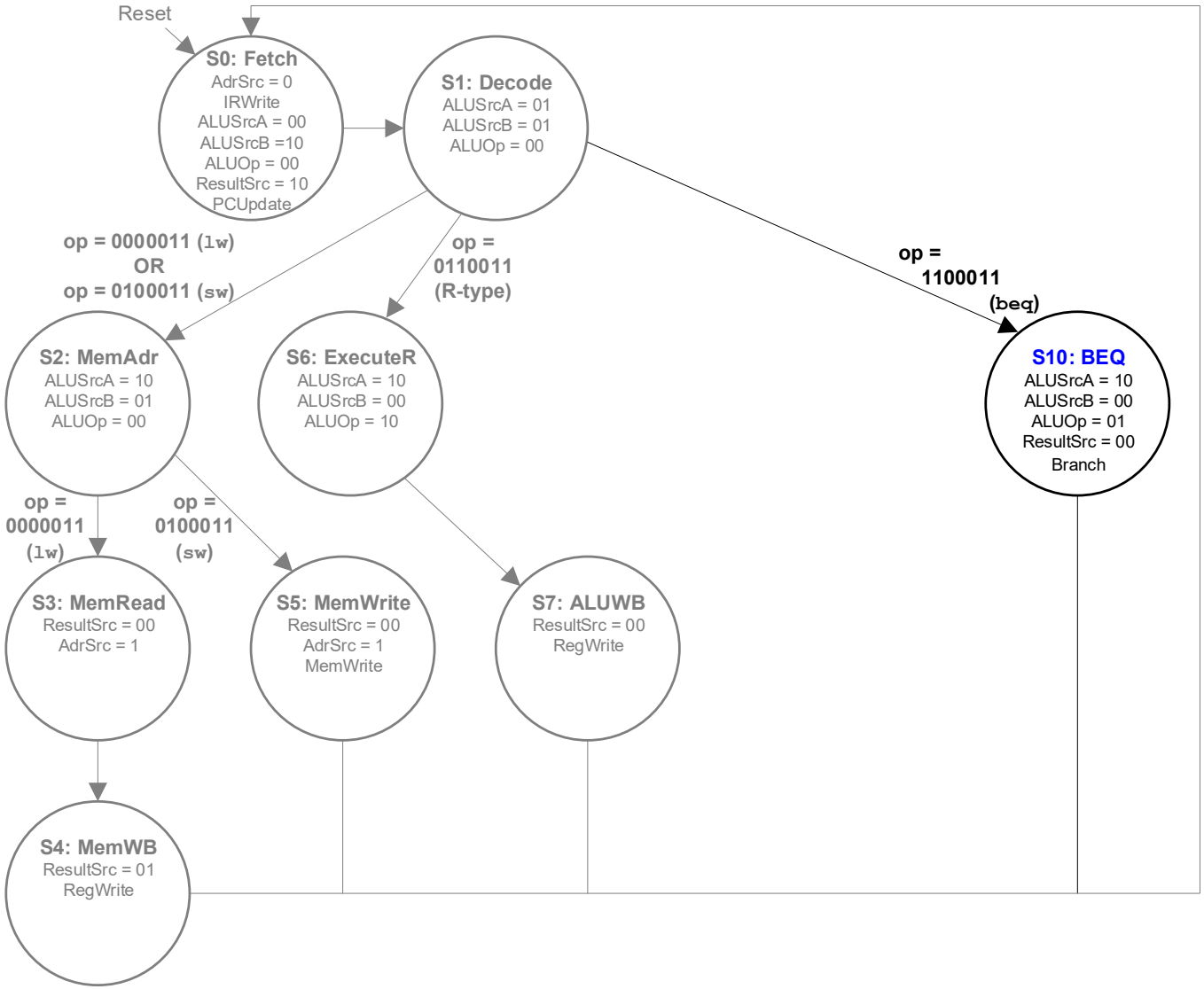
Read Registers and
Calculate Target Address (PC+imm)

Multicycle RISC-V Processor

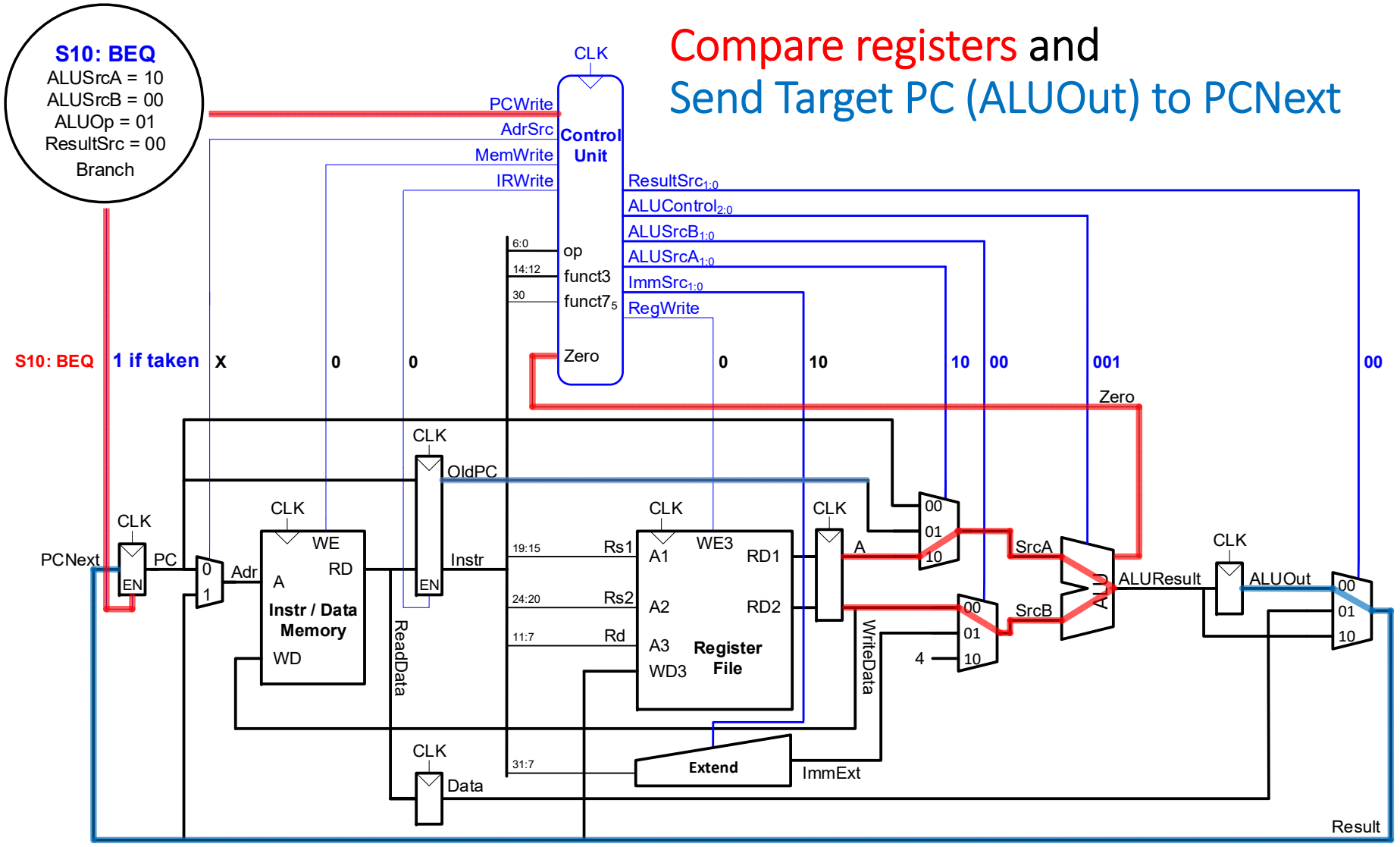
Read Registers and Calculate Target Address (PC+imm)



Multicycle RISC-V Processor

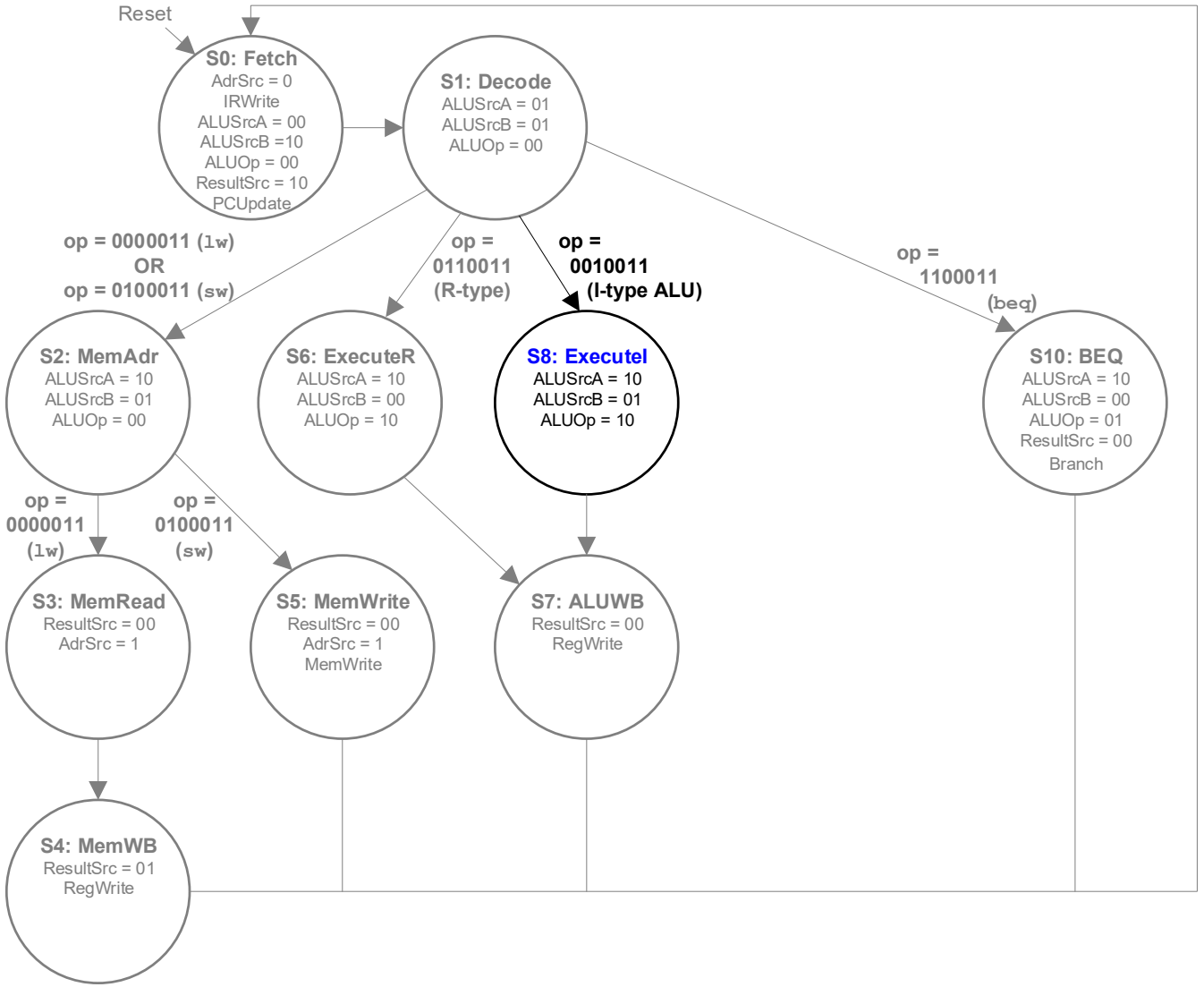


Multicycle RISC-V Processor



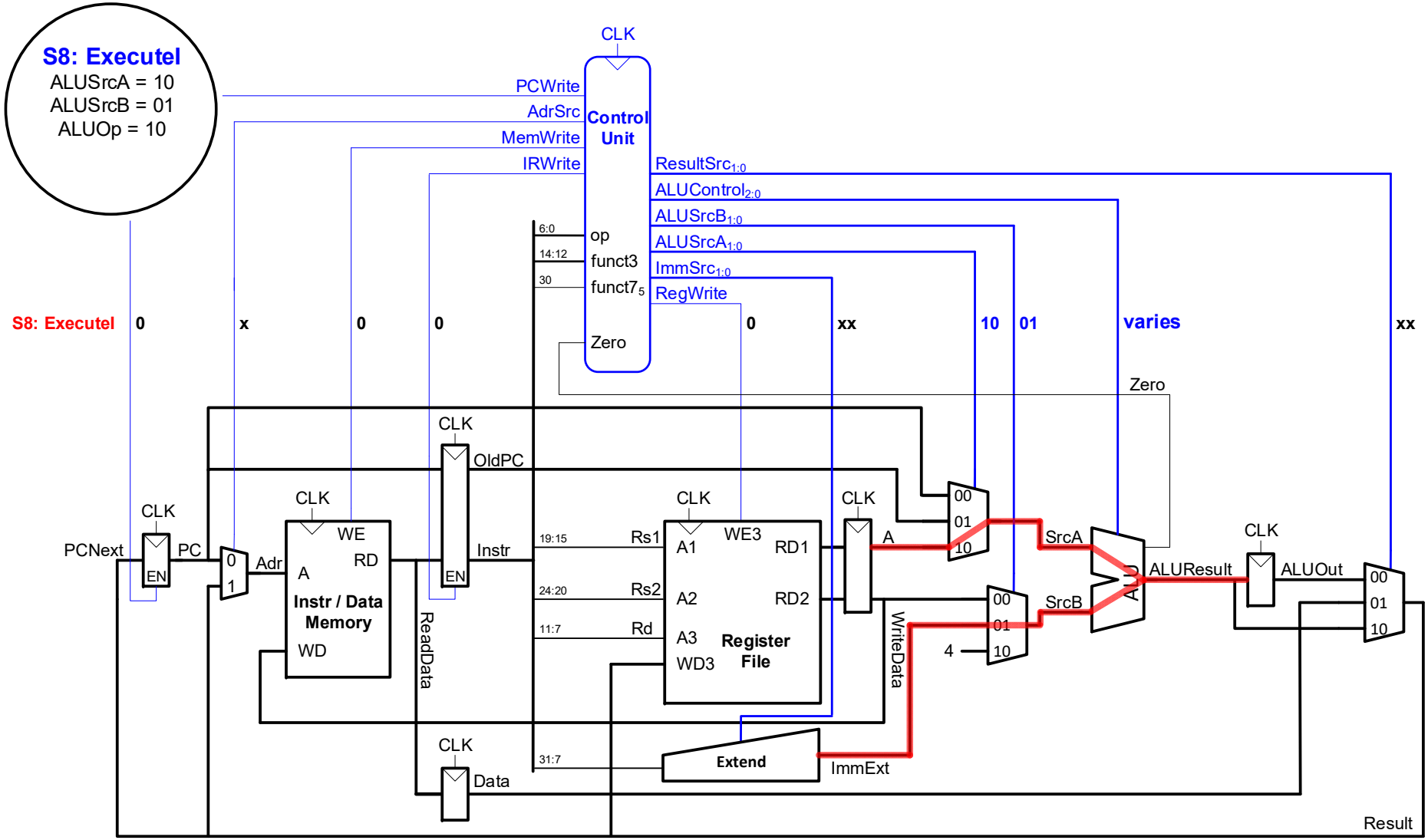
Extending the RISC-V Multicycle Processor

Multicycle RISC-V Processor

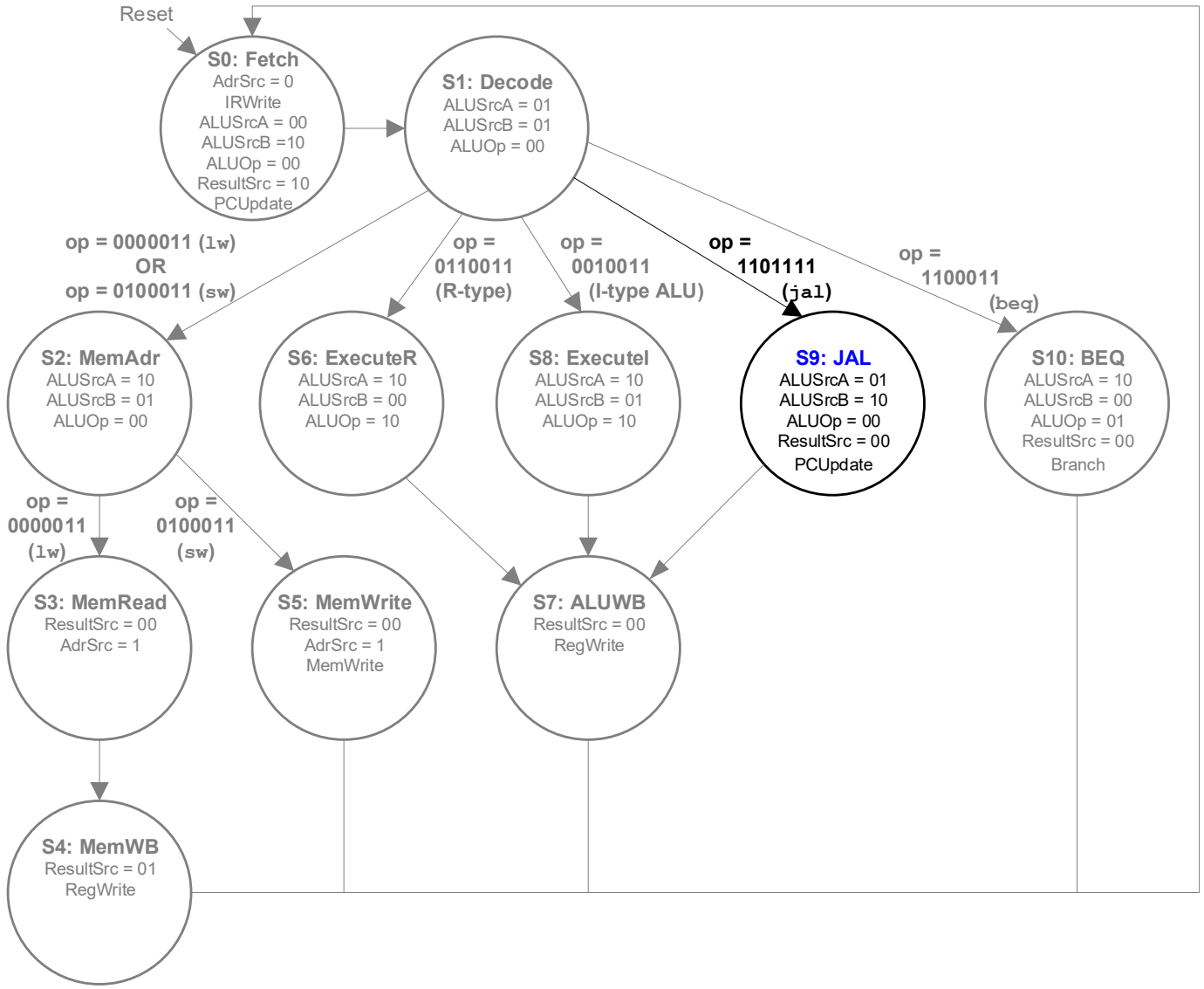


I type Execució en ALU

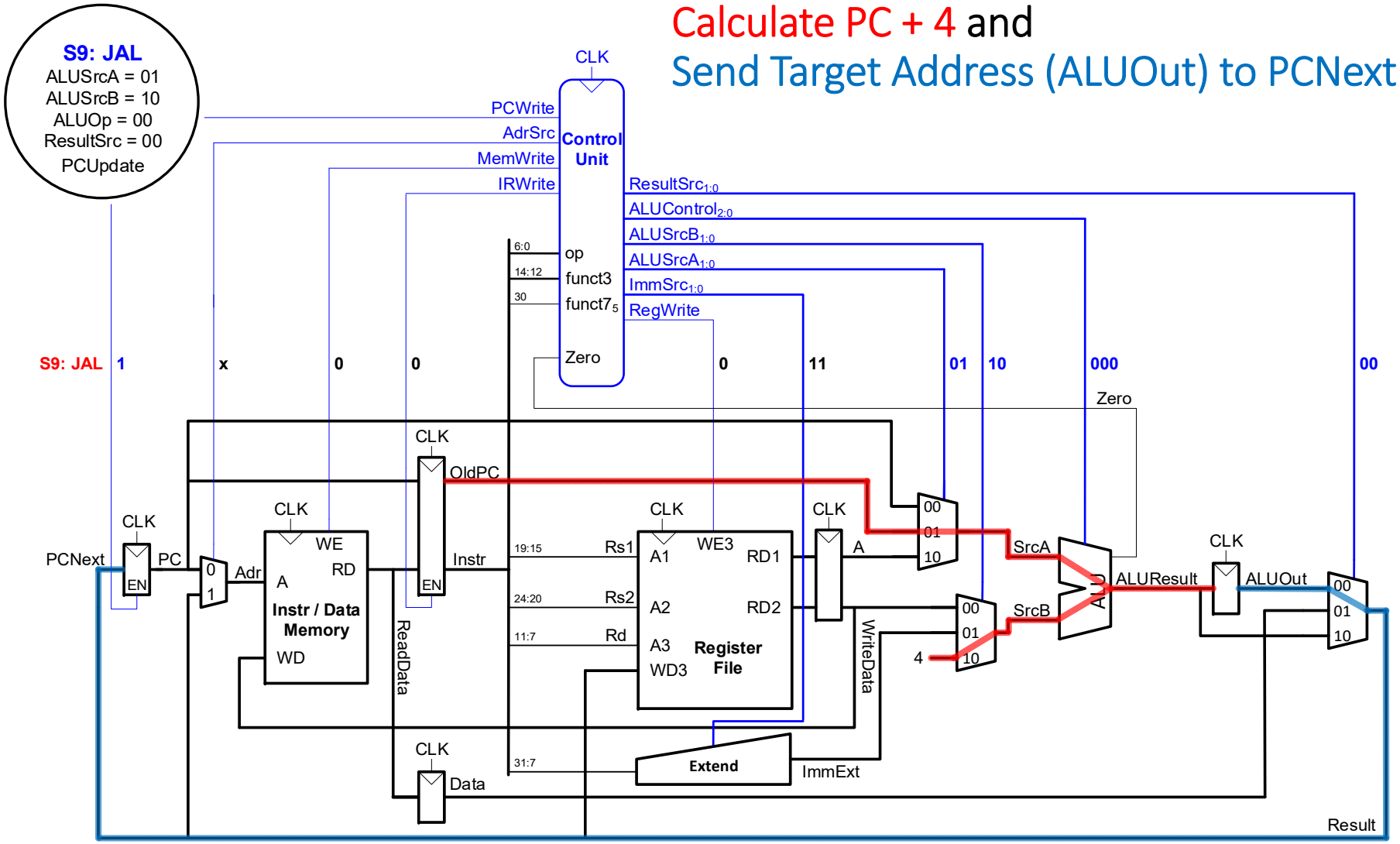
Multicycle RISC-V Processor



Multicycle RISC-V Processor. Jal

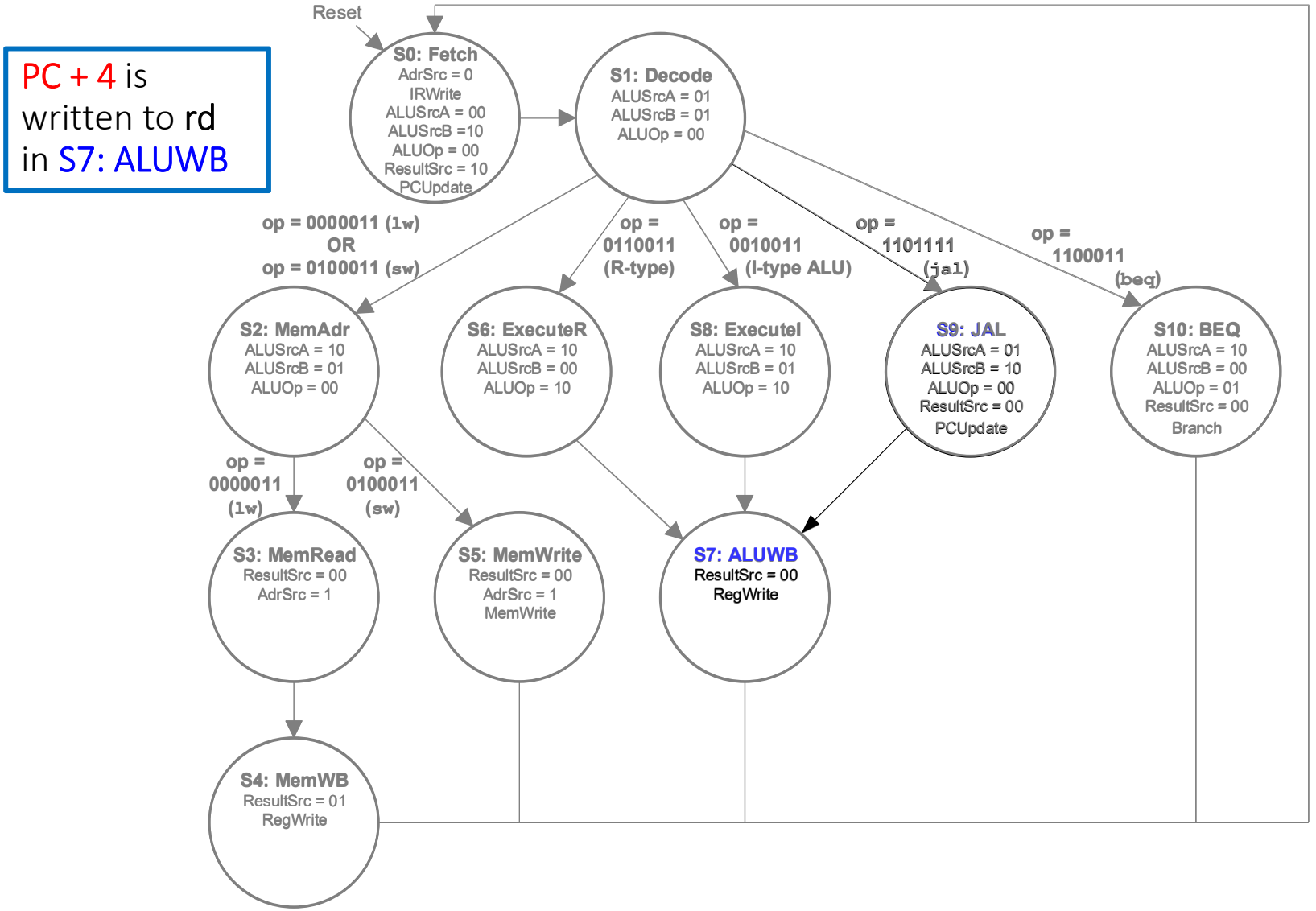


Multicycle RISC-V Processor



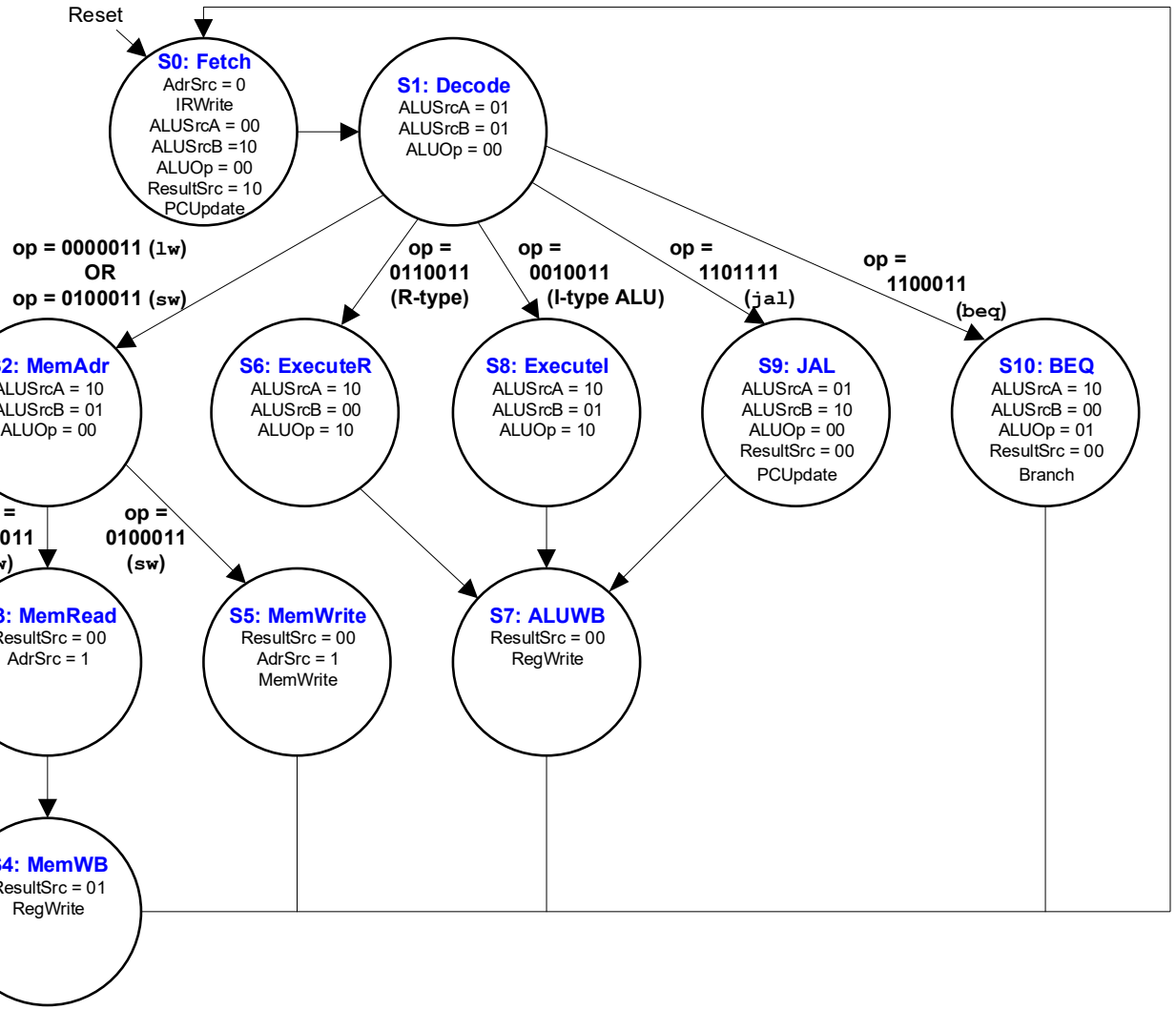
Multicycle RISC-V Processor

PC + 4 is written to rd in S7: ALUWB



Multicycle RISC-V Processor

State	Datapath μOp
Fetch	Instr ←Mem[PC]; PC ← PC+4
Decode	ALUOut ← PCTarget
MemAdr	ALUOut ← rs1 + imm
MemRead	Data ← Mem[ALUOut]
MemWB	rd ← Data
MemWrite	Mem[ALUOut] ← rd
ExecuteR	ALUOut ← rs1 op rs2
ExecuteI	ALUOut ← rs1 op imm
ALUWB	rd ← ALUOut
BEQ	ALUResult = rs1-rs2; if Zero, PC = ALUOut
JAL	PC = ALUOut; ALUOut = PC+4



Multicycle Performance

Multicycle RISC-V Processor

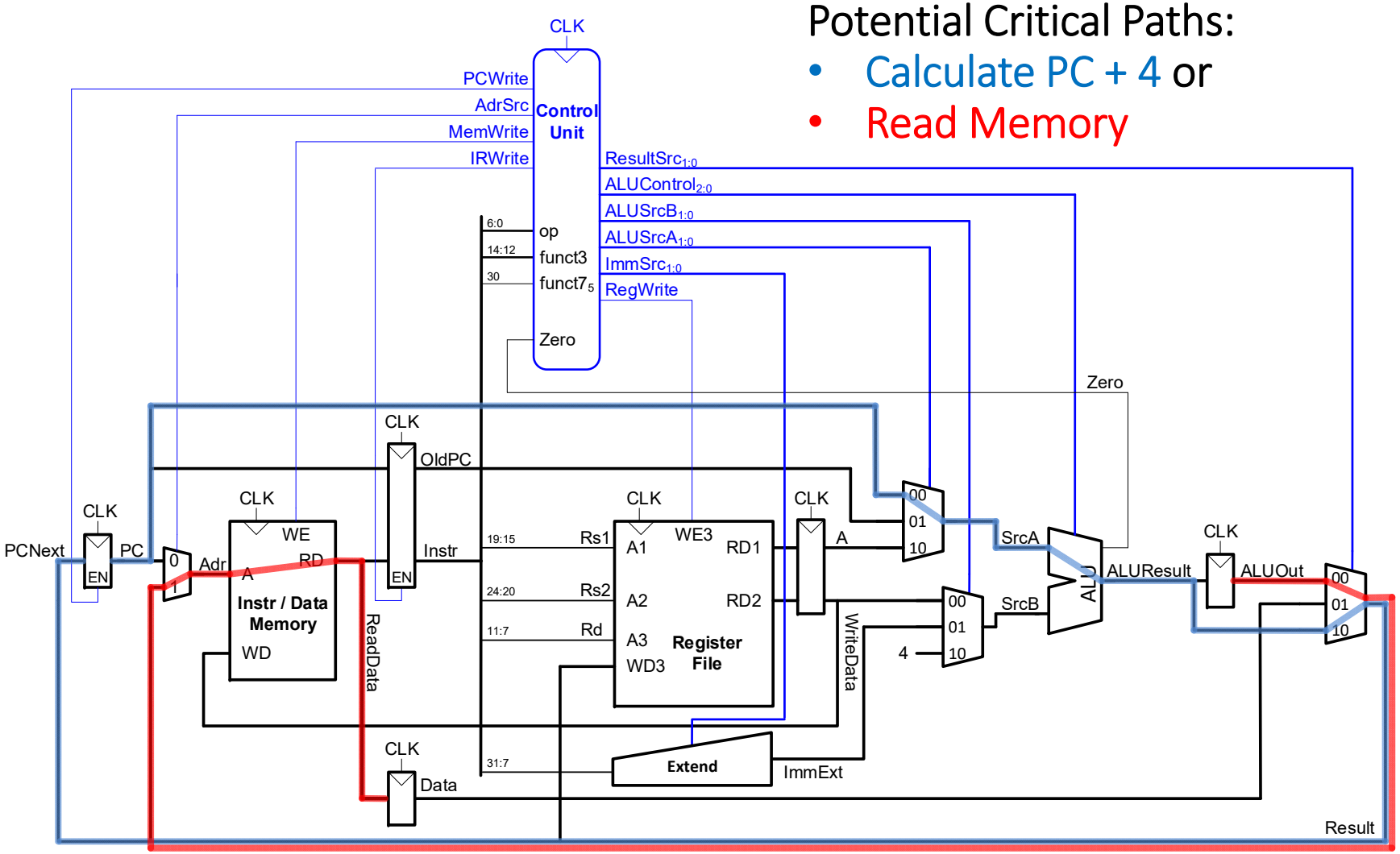
- Les instruccions prenen different nombre de cicles:
  - **3 cicles:** `beq`
  - **4 cicles: R-type**, `addi`, `sw`, `jal`
  - **5 cicles:** `lw`
- CPI es calcula promitjant
- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 13% branches
  - 52% R-type

<CPI> = (0.13)(3) + (0.52 + 0.10)(4) + (0.25)(5) = 4.12

Multicycle RISC-V Processor

Potential Critical Paths:

- Calculate PC + 4 or
- Read Memory



Multicycle RISC-V Processor

Camins critics del processador multicicle:

- **Consideracions:**
  - RF is més ràpid que la memòria
  - L'escriptura de memòria és més ràpida que la lectura

$$T_{c\_multi} = t_{pcq} + t_{\text{dec}} + 2t_{\text{mux}} + \max(t_{\text{ALU}}, t_{\text{mem}}) + t_{\text{setup}}$$

Multicycle RISC-V Processor

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	30
AND-OR gate	$t_{AND\text{-}OR}$	20
ALU	t_{ALU}	120
Decoder (Control Unit)	t_{dec}	25
Extend unit	t_{dec}	35
Memory read	t_{mem}	200
Register file read	t_{RFread}	100
Register file setup	$t_{RFsetup}$	60

$$T_{c\_multi} = t_{pcq} + t_{dec} + 2t_{mux} + \max(t_{ALU}, t_{mem}) + t_{setup}$$

$$=$$

Multicycle RISC-V Processor

For a program with **100 billion** instructions executing on a **multicycle** RISC-V processor

- **CPI** = 4.12 cycles/instruction
- **Clock cycle time:** T_{c_multi} = 375 ps

Execution Time = (# instructions) × CPI × T_c

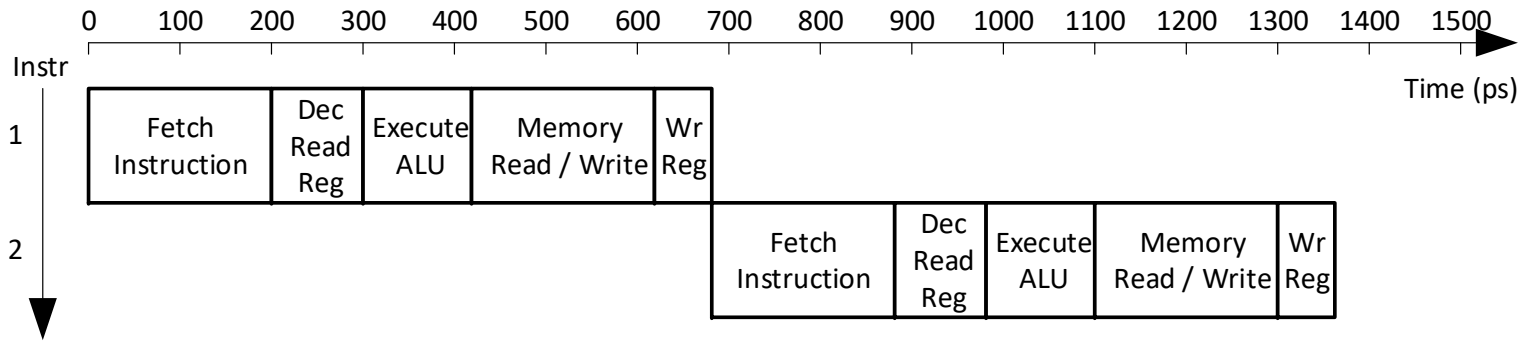
Pipelined RISC-V Processor

Multicycle RISC-V Processor

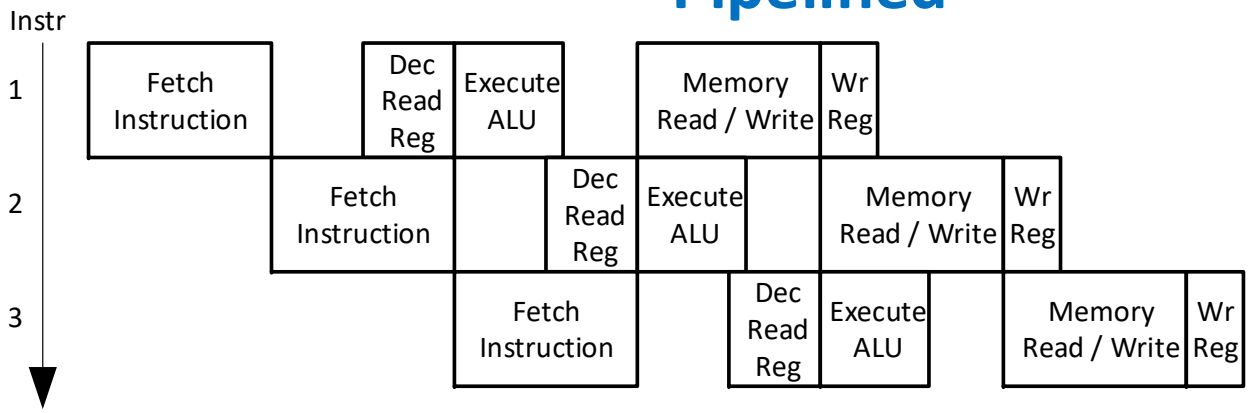
- Temporal parallelism
- Divide single-cycle processor into 5 stages:
  - Fetch
  - Decode
  - Execute
  - Memory
  - Writeback
- Add pipeline registers between stages

Multicycle RISC-V Processor

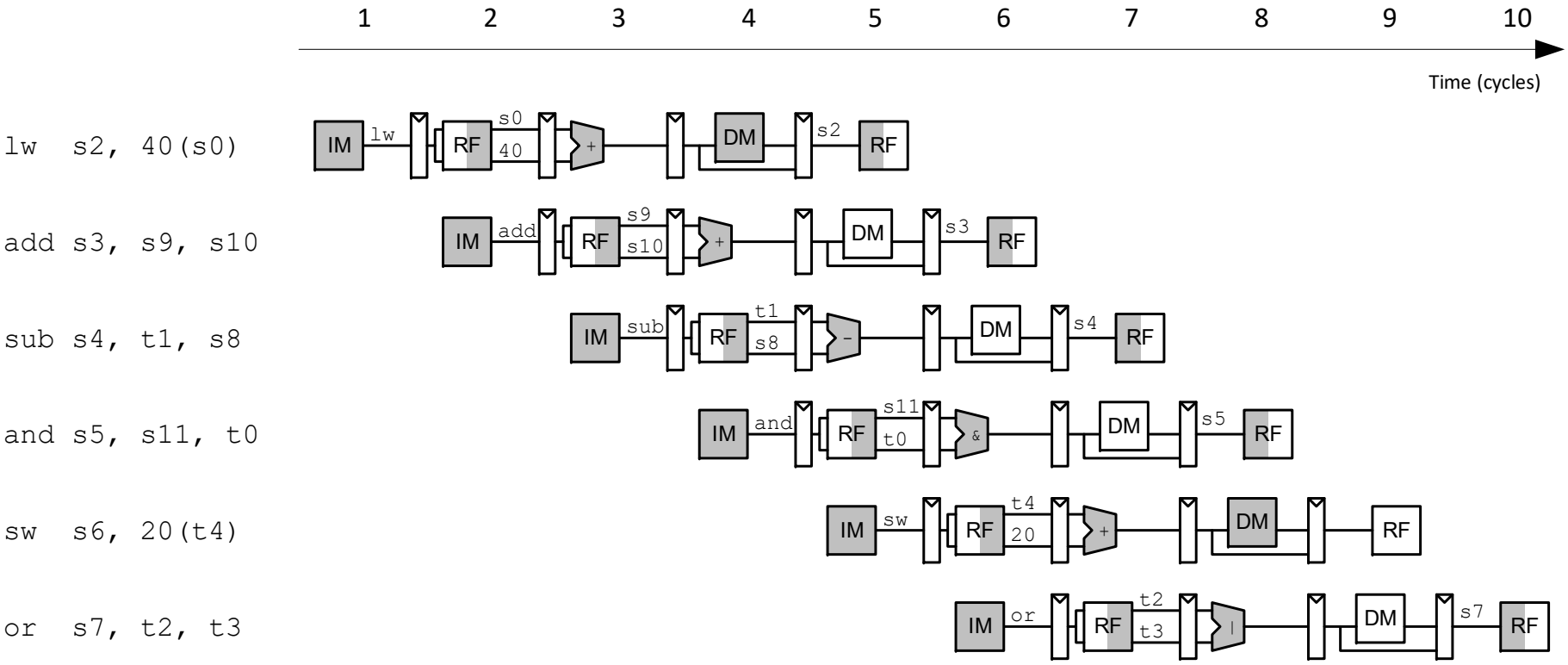
Single-Cycle



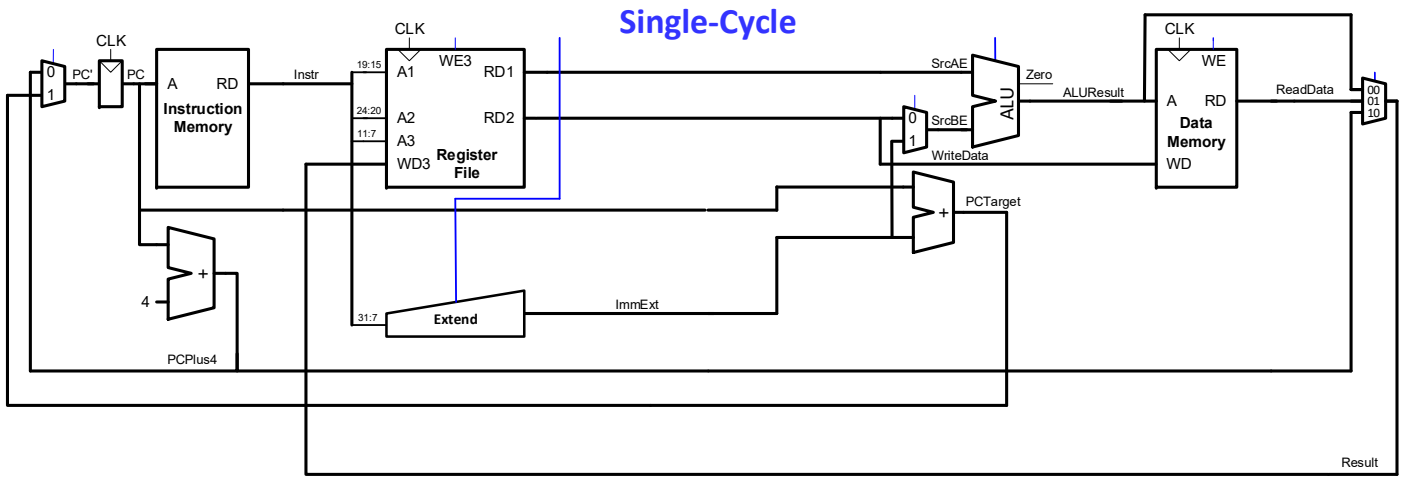
Pipelined

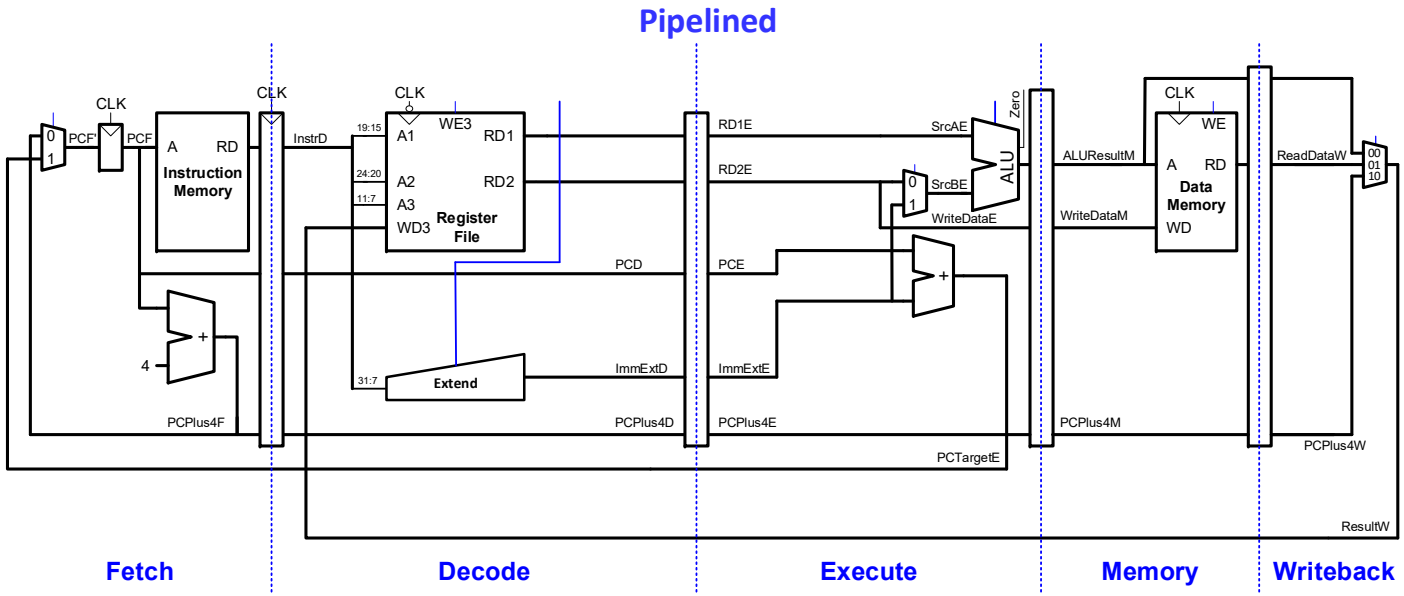


Multicycle RISC-V Processor



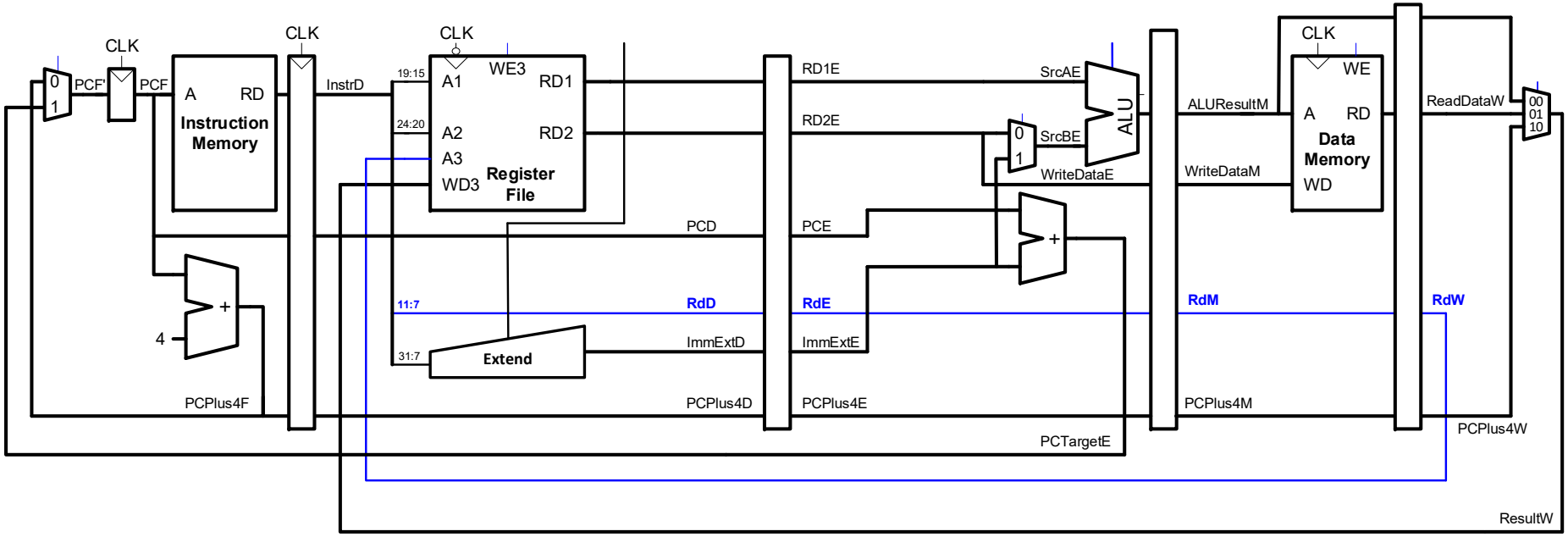
Multicycle RISC-V Processor





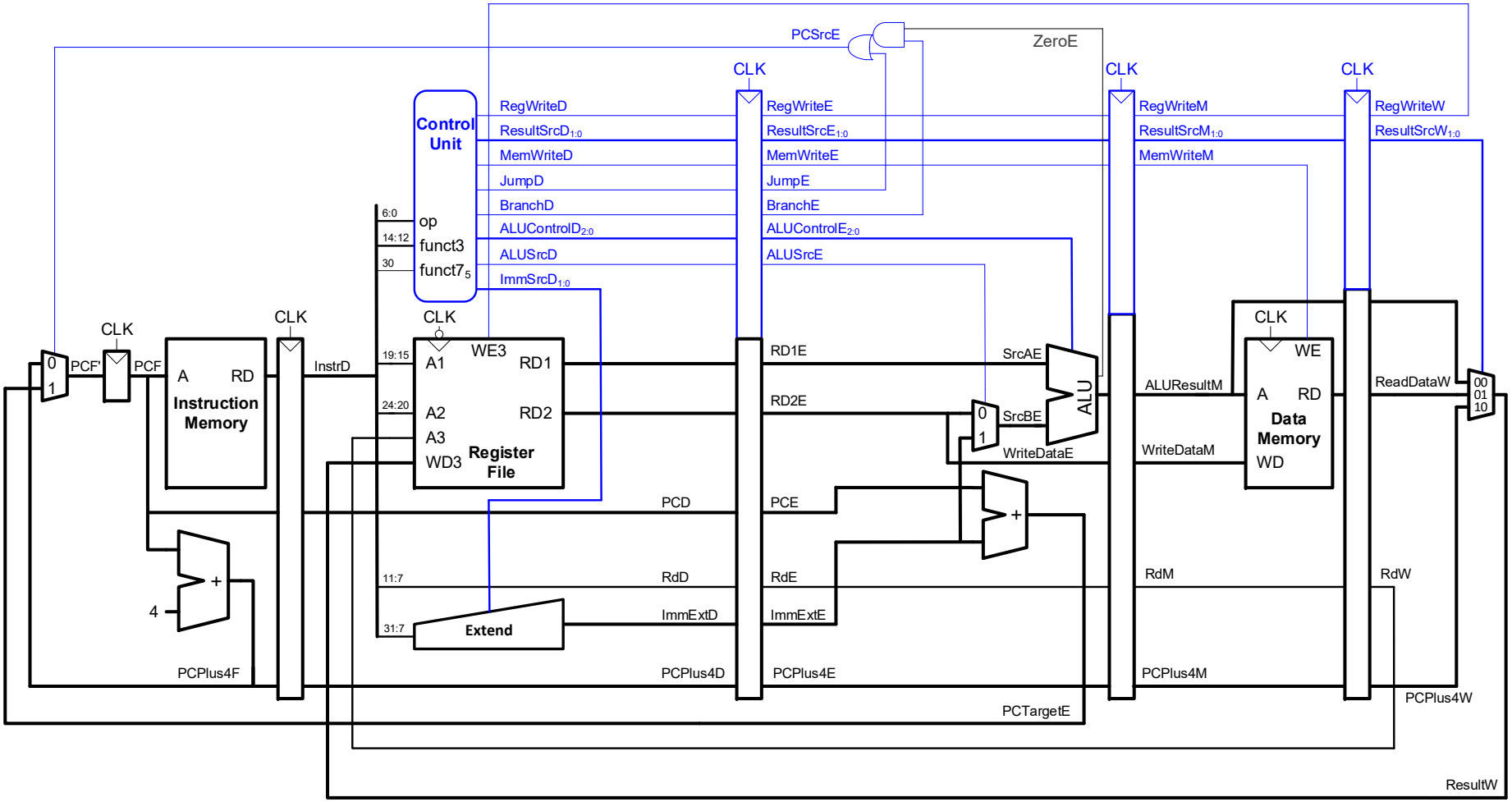
Signals in Pipelined Processor are appended with first letter of stage (i.e., PCF, PCD, PCE).

Multicycle RISC-V Processor



- *Rd* must arrive at same time as *Result*
- Register file written on falling edge of *CLK*

Multicycle RISC-V Processor



- **Same control unit** as single-cycle processor
- **Control signals travel with** the instruction (drop off when used)