Tema 6:

Diseño multiciclo del procesador

Fundamentos de computadores II

José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

Contenidos

- ✓ Introducción.
- ✓ Diseño de la ruta de datos.
- ✓ Diseño del controlador.
- ✓ Comparativa: monociclo vs. multiciclo.
- ✓ Métricas de rendimiento.

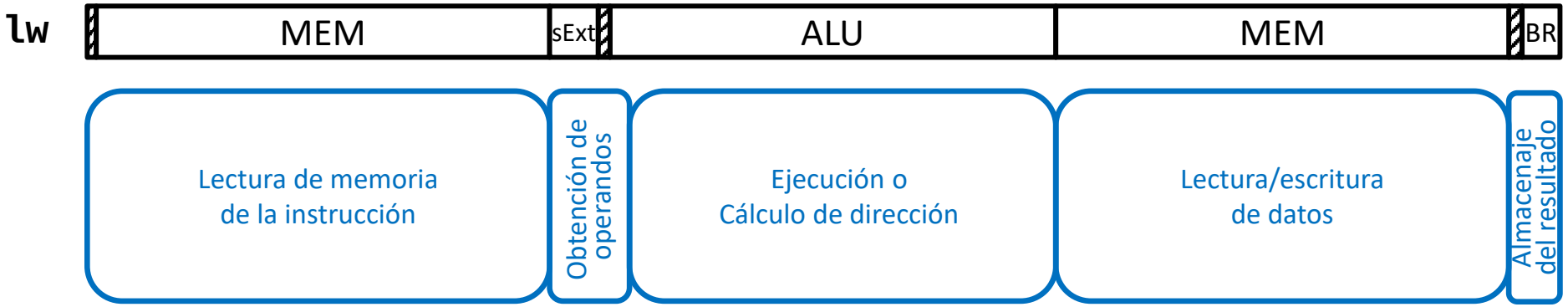
- ✓ Apéndice tecnológico.

Transparencias basadas en los libros:

- S.L. Harris and D. Harris. *Digital Design and Computer Architecture. RISC-V Edition.*
- D.A. Patterson and J.L. Hennessy. *Computer Organization and Design. RISC-V Edition.*

Introducción

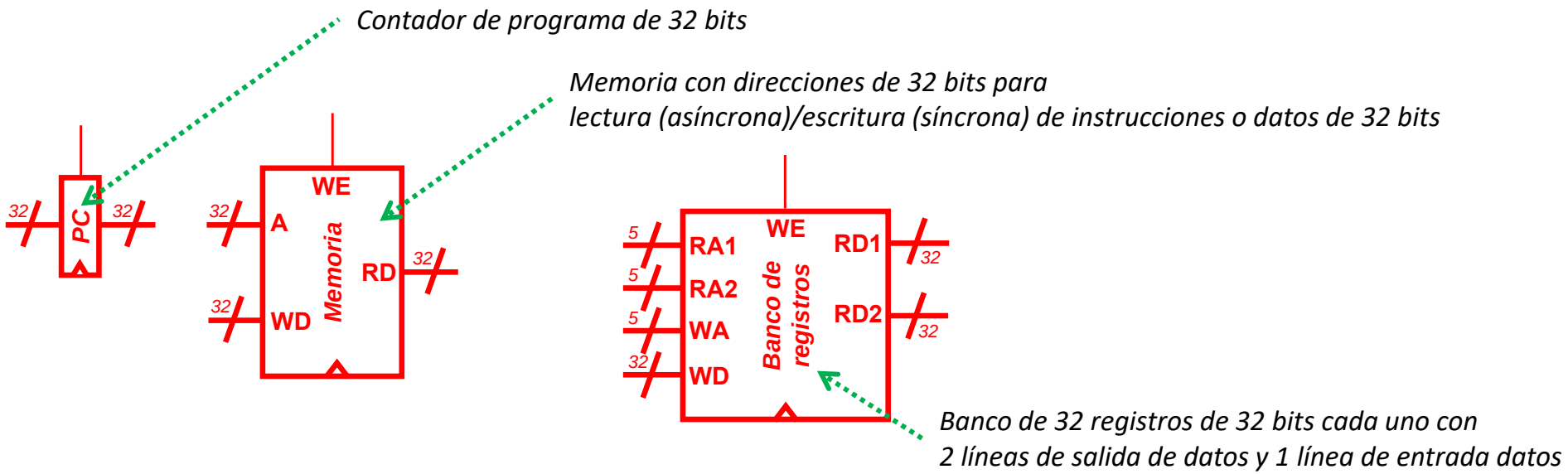
- El procesador multiciclo trata de solventar los problemas del monociclo dividiendo en varias etapas la ejecución de una instrucción:
  - El tiempo de ciclo viene dado por la etapa más lenta.
    - Será muy inferior al tiempo de ciclo del procesador monociclo.
    - Cada instrucción tardará un tiempo distinto en ejecutarse por requerir un número diferente de ciclos de reloj para hacerlo.
    - El tiempo de ejecución de una instrucción será proporcional a su complejidad.
  - Se puede reusar hardware siempre que se utilice en ciclos distintos.
    - Requiere de una única ALU y una única memoria para instrucciones y datos.



Diseño de la ruta de datos

Elementos de almacenamiento "arquitectónicos"

- Los elementos de almacenamiento visibles al programador, son los mismos que en procesador monociclo.

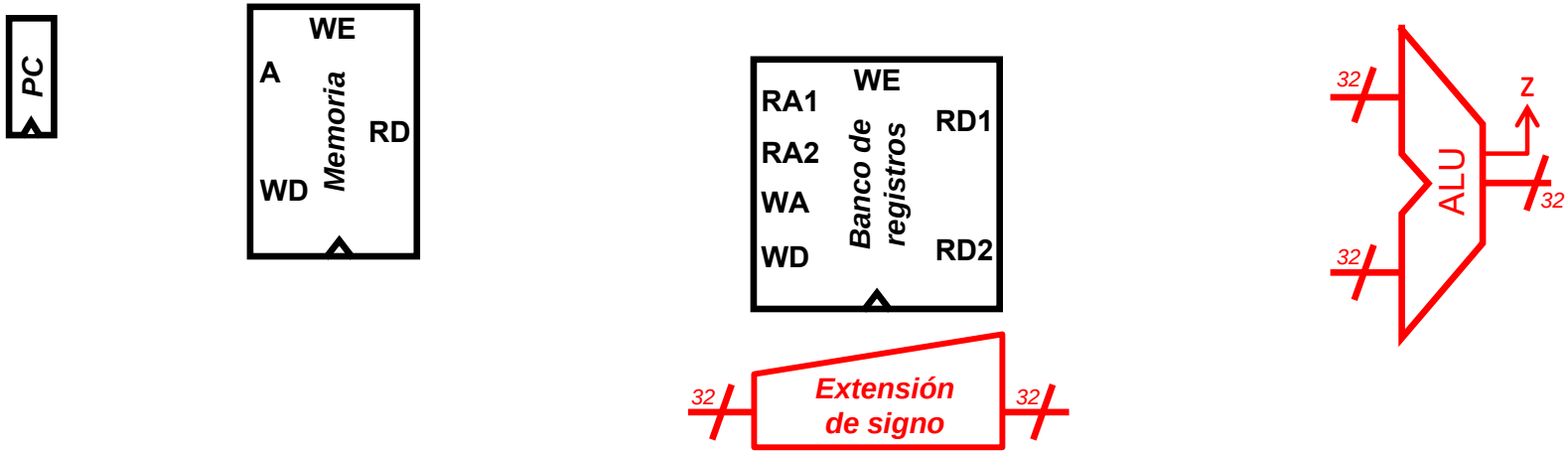


- Sin embargo, el procesador multiciclo tiene una única memoria unificada.
  - Las instrucciones se leen de memoria en un ciclo de reloj distinto al ciclo en que se leen/escriben los datos por lo que no es necesario tener 2.
- Además, el PC será un registro convencional con señal de carga.
  - En el procesador multiciclo el PC no se actualiza en todos los ciclos.

Diseño de la ruta de datos

Elementos funcionales

- Dispone también de una ALU y un Extensor de Signo combinacionales idénticos a los existentes en la implementación monociclo.

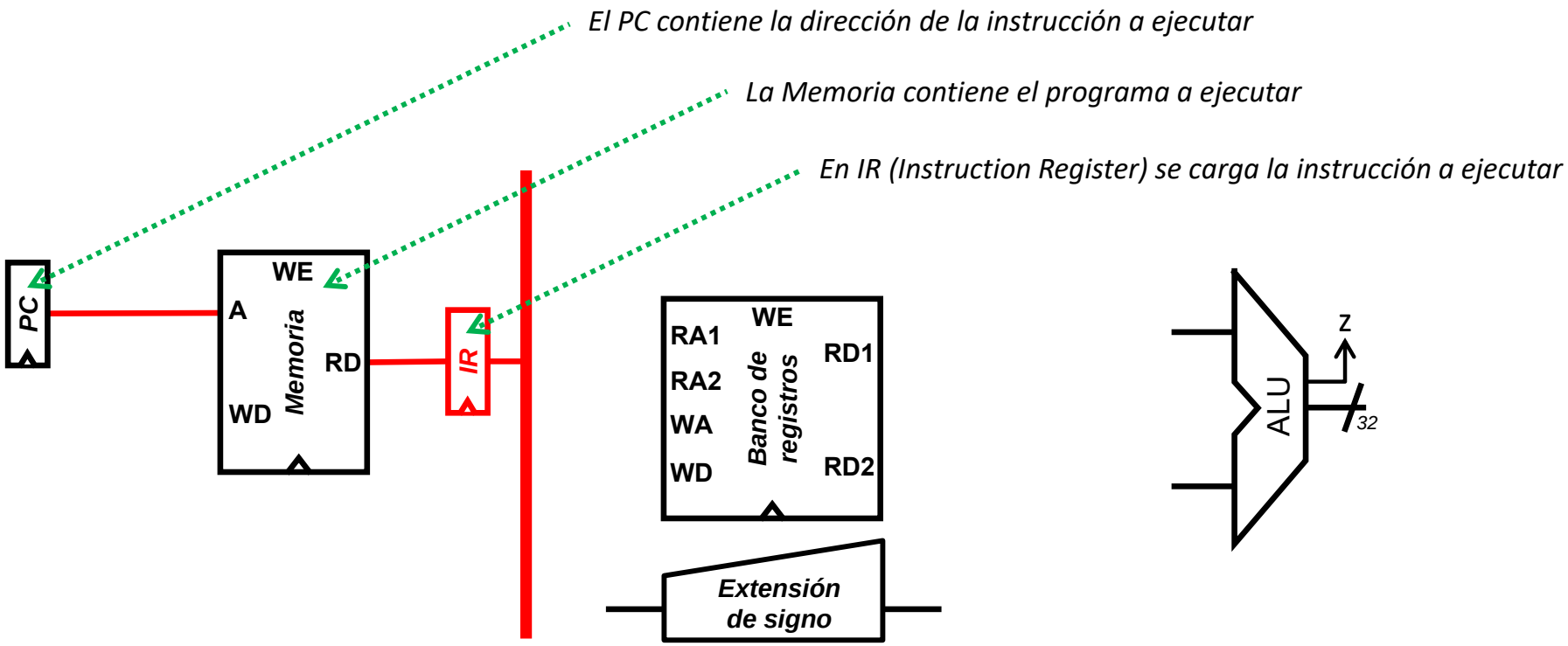


- En el procesador multiciclo, la ALU se reutiliza para hacer todas las operaciones aritméticas que sean necesarias.
  - No existen sumadores adicionales para operar con direcciones.

Diseño de la ruta de datos

Lectura de la instrucción

- La ejecución de toda instrucción comienza con su lectura de Memoria.

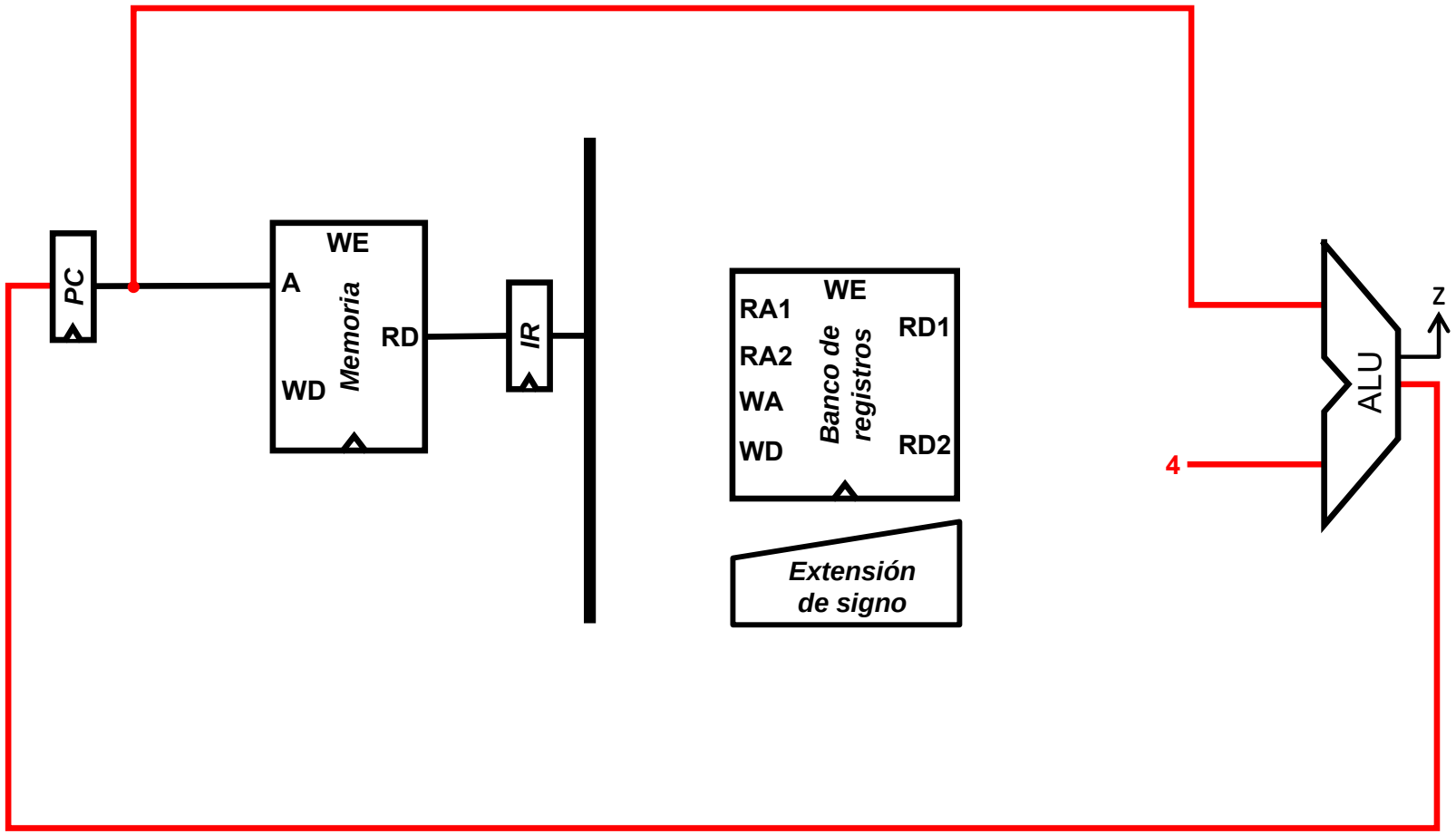


- La instrucción leída de Memoria se carga en el registro auxiliar IR que la almacenará durante todos los ciclos que dure su ejecución.

Diseño de la ruta de datos

Incremento del PC

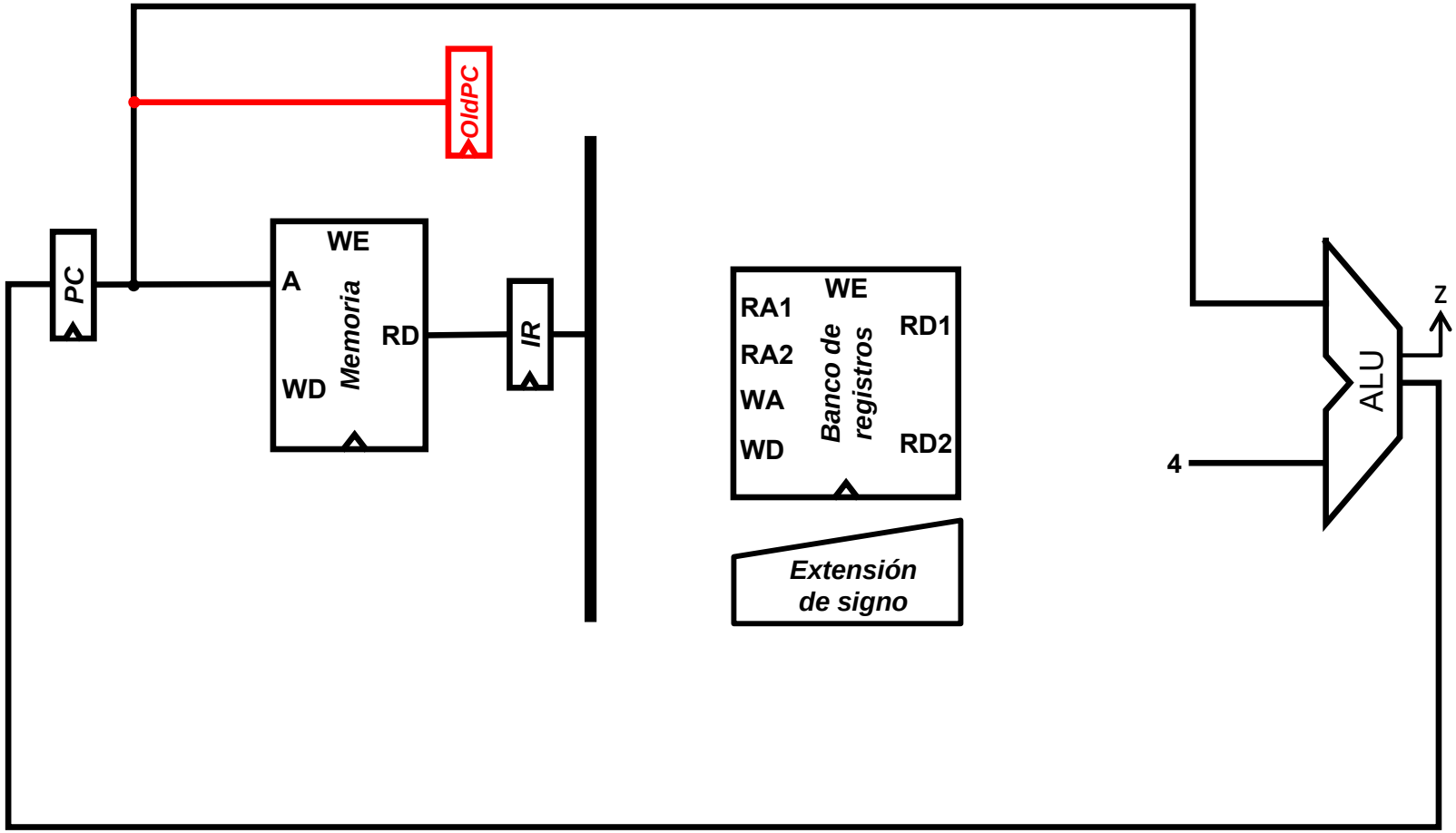
- A la vez que se carga la instrucción en IR, puede actualizarse el PC con la dirección de la instrucción siguiente sumándole 4 en la ALU



Diseño de la ruta de datos

Salvado de la dirección de la instrucción actual

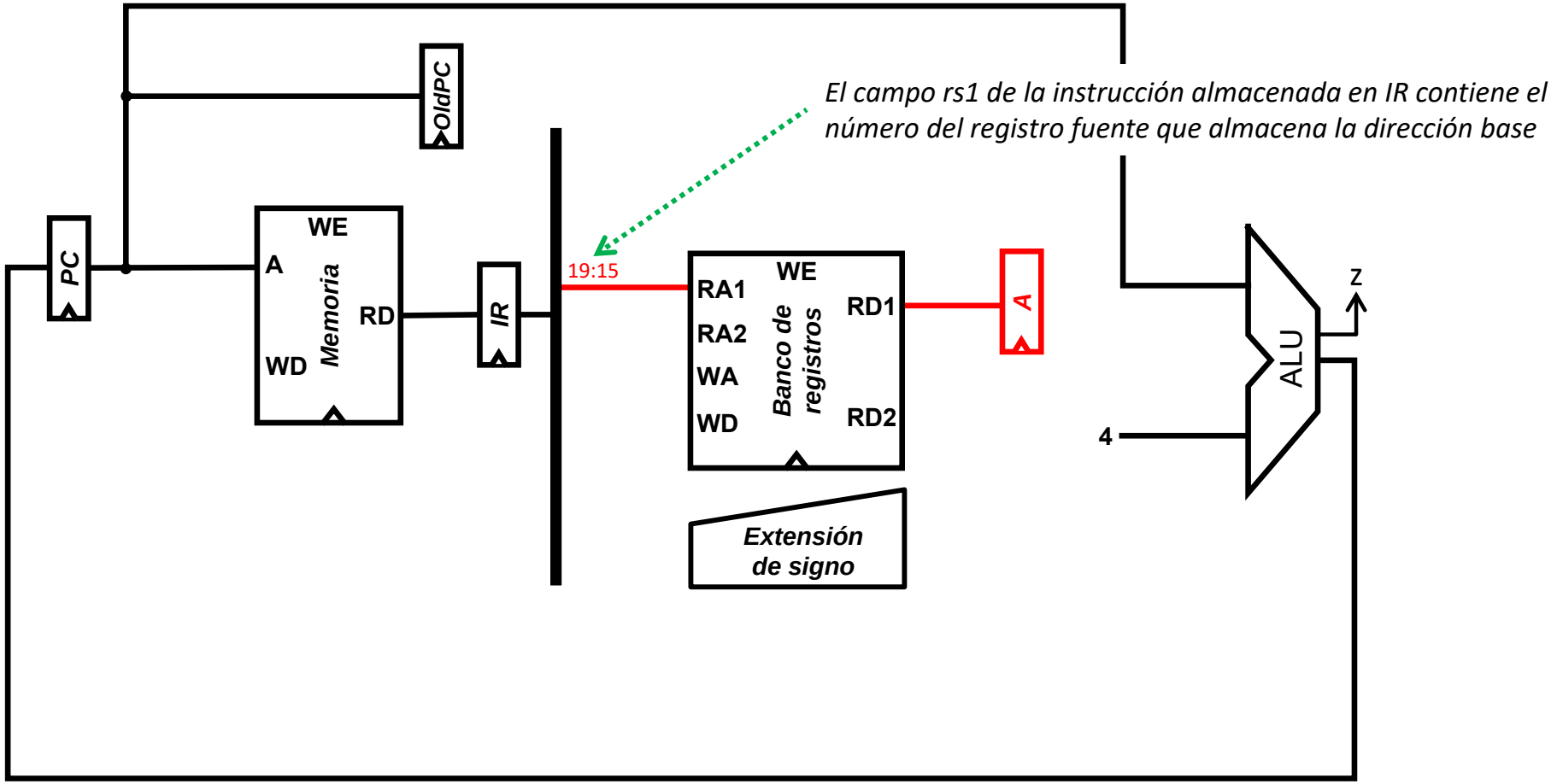
- El PC sin incrementar se salva en el registro auxiliar oldPC para el cálculo de direcciones de salto relativas a PC (instrucciones **`beq`** o **`jal`**).



Diseño de la ruta de datos

Instrucción `lw`: lectura del registro base

- Para almacenar la dirección base contenida en el registro rs1 del Banco de Registros se añade el registro auxiliar A.

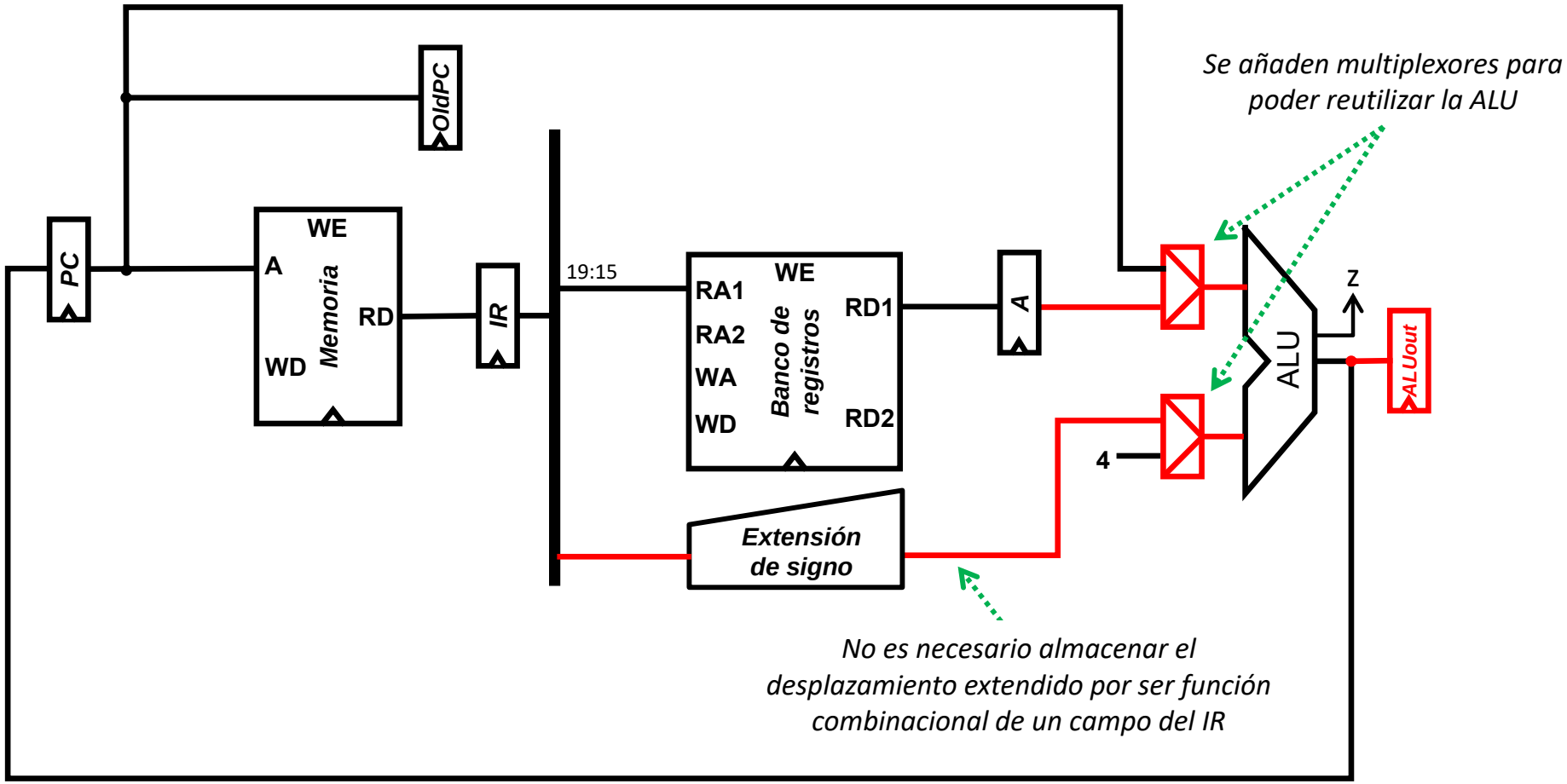


BR[rd] ← Mem[BR[rs1] + sExt(imm)], PC ← PC+4

Diseño de la ruta de datos

Instrucción `lw`: cálculo de la dirección efectiva

- El signo del desplazamiento se extiende, se suma en la ALU con la dirección base almacenada en A y se carga en el registro auxiliar ALUout.

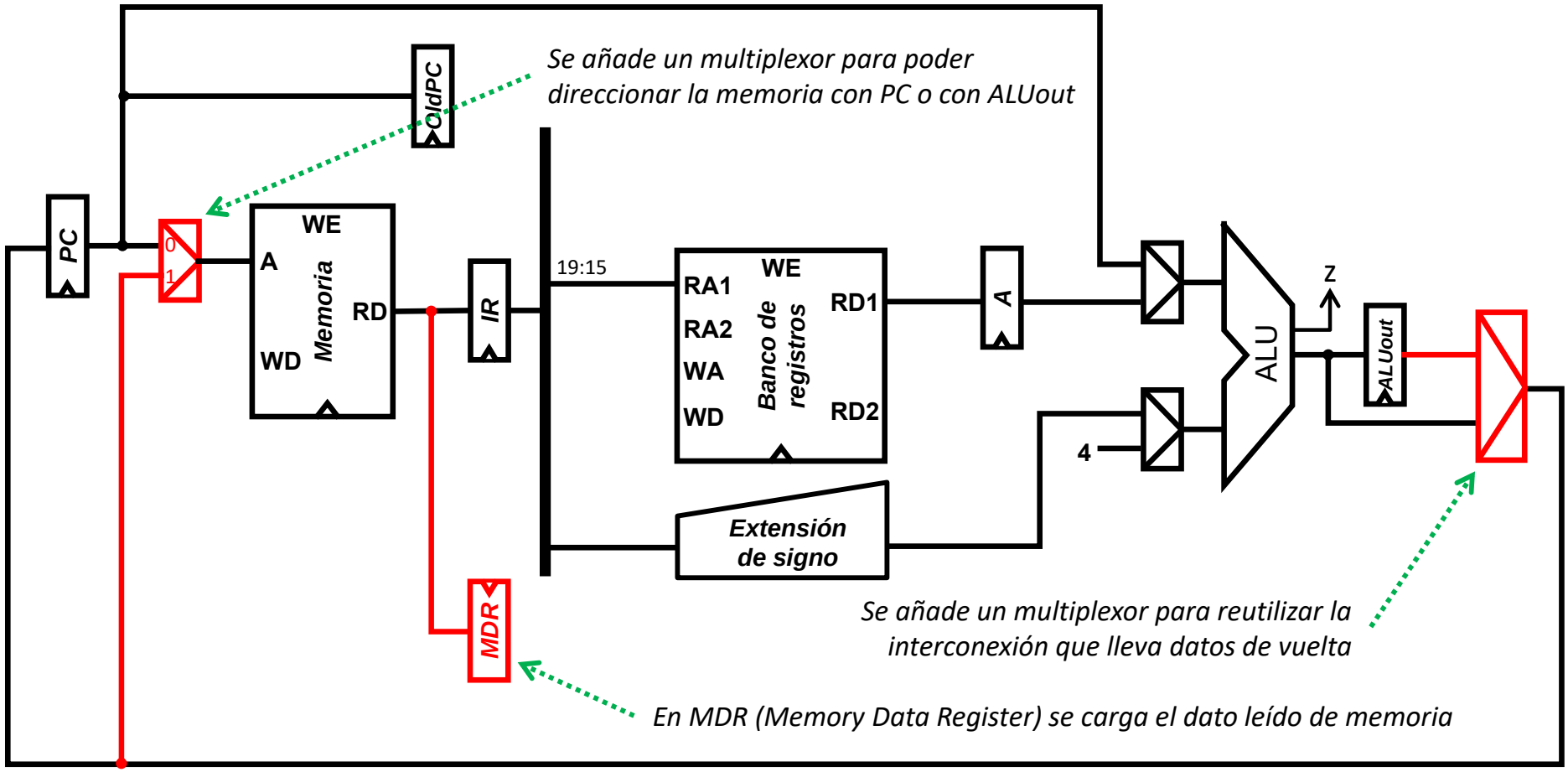


BR[rd] ← Mem[BR[rs1] + sExt(imm)], PC ← PC+4

Diseño de la ruta de datos

Instrucción `lw`: lectura del operando

- La memoria se direcciona con la dirección efectiva almacenada en ALUout y el dato leído se guarda en el registro auxiliar MDR.

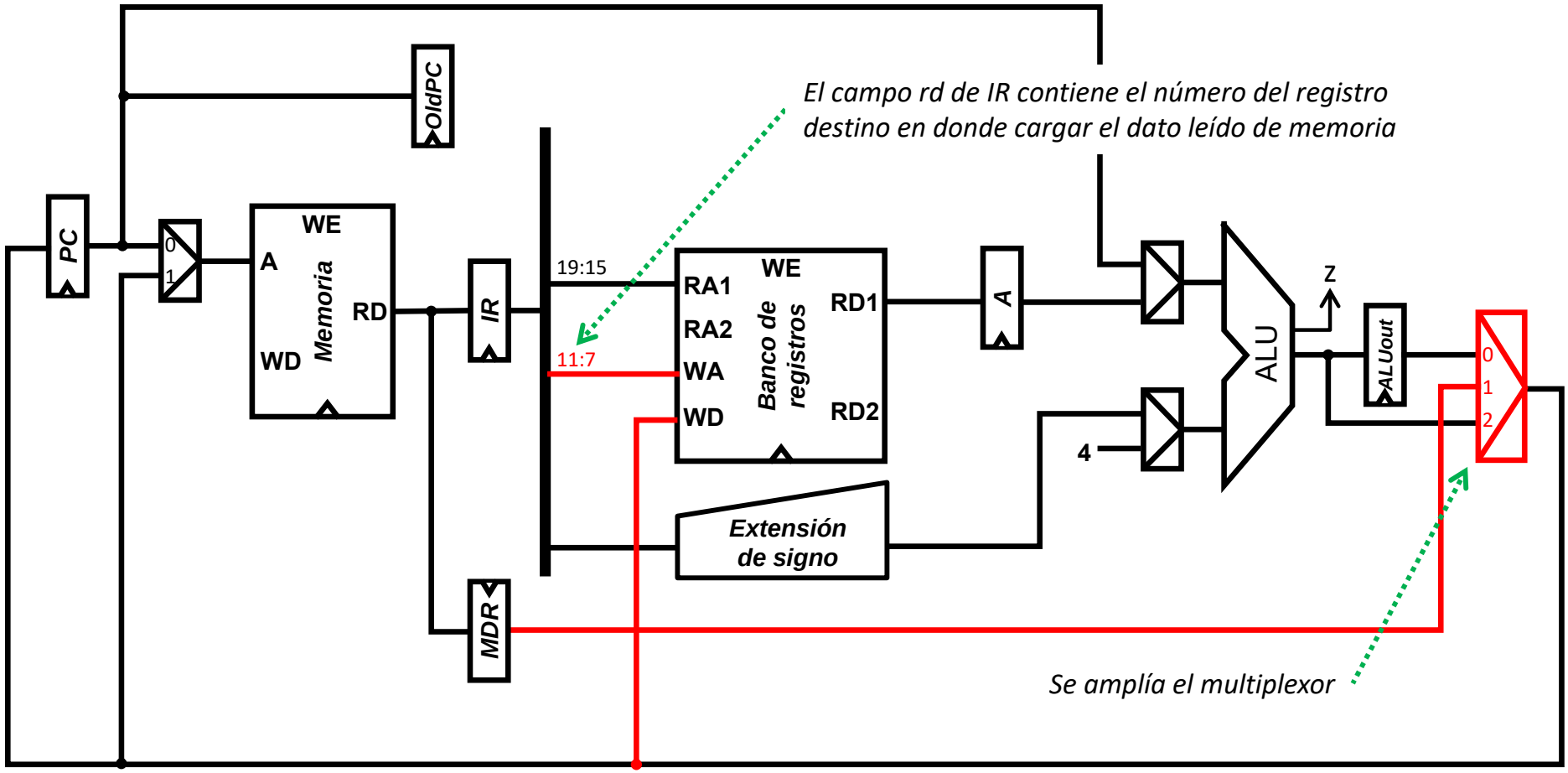


BR[rd] ← Mem[BR[rs1] + sExt(imm)], PC ← PC+4

Diseño de la ruta de datos

Instrucción `lw`: almacenaje del operando

- Se guarda en el Banco de Registros el dato almacenado en MDR (previamente leído de Memoria).

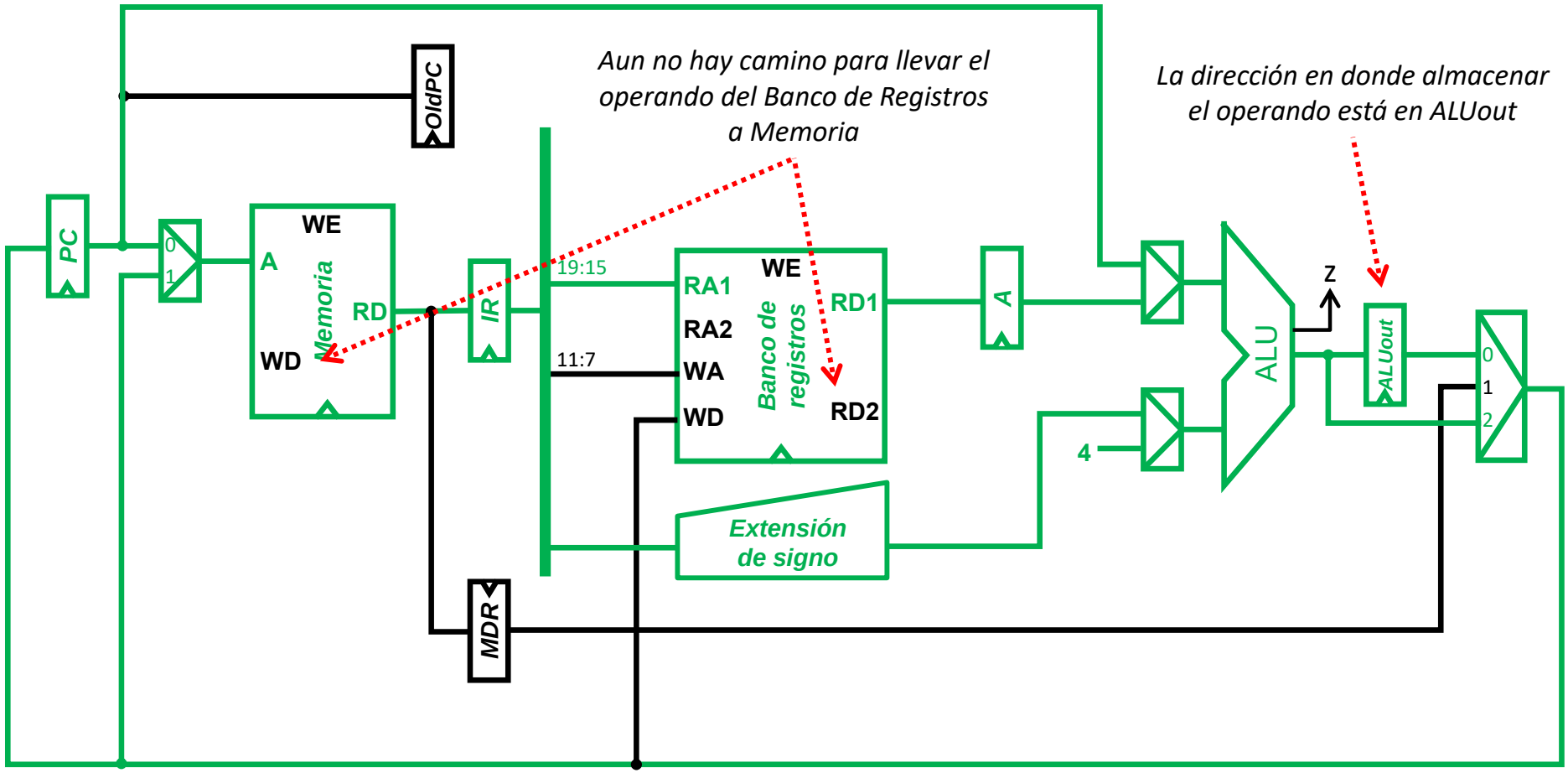


BR[rd] ← Mem[BR[rs1] + sExt(imm)], PC ← PC+4

Diseño de la ruta de datos

Ruta de datos para instrucciones **sw**

- Esta ruta de datos puede reutilizarse para efectuar todas las acciones de la instrucción **sw** menos la escritura en memoria del operando

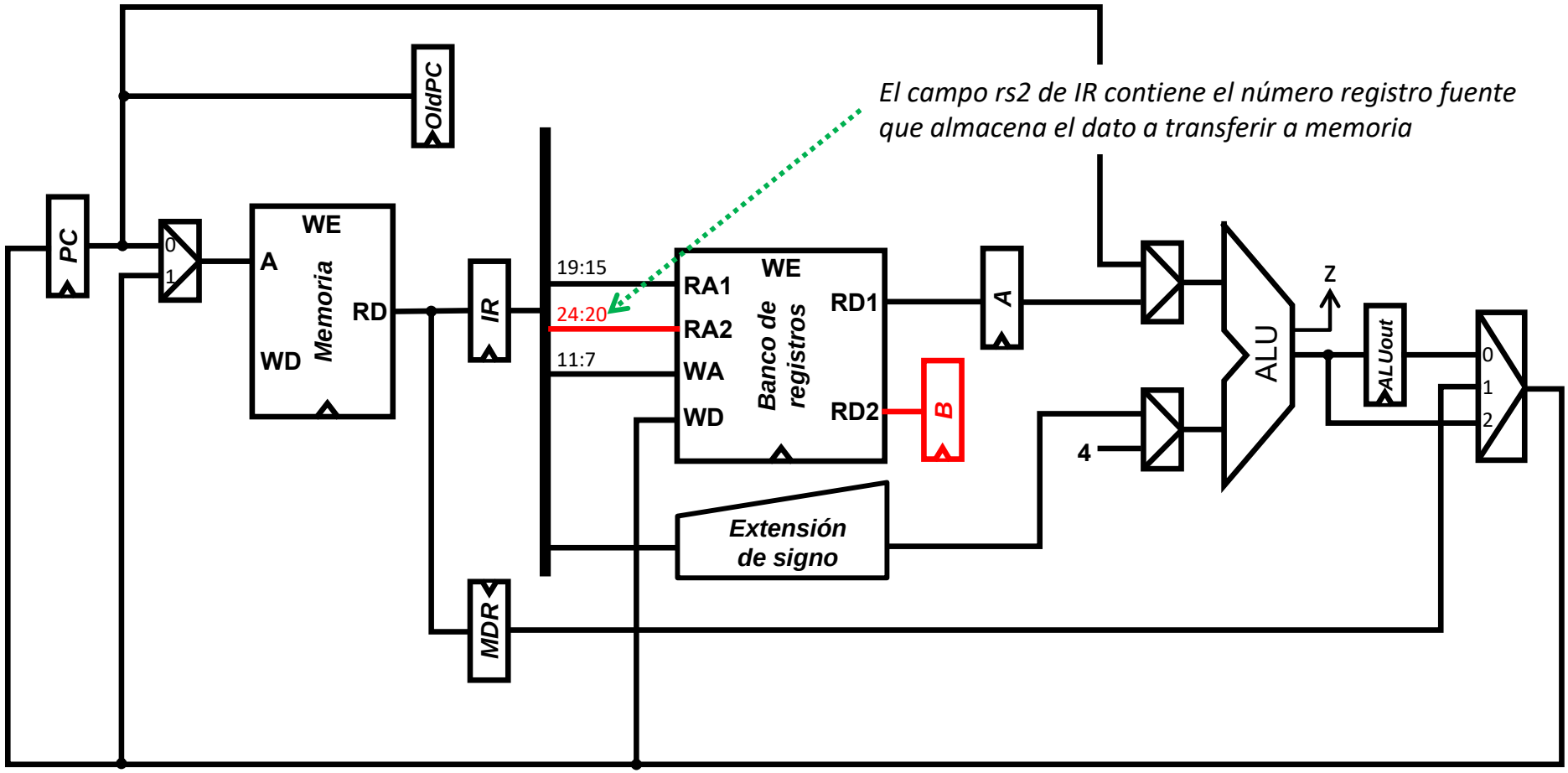


Mem[BR[rs1] + sExt(imm)] ← BR[rs2], PC ← PC+4

Diseño de la ruta de datos

Instrucción **sw**: lectura del operando

- Se añade el registro auxiliar B para almacenar el dato contenido en el registro rs2 del Banco de Registros

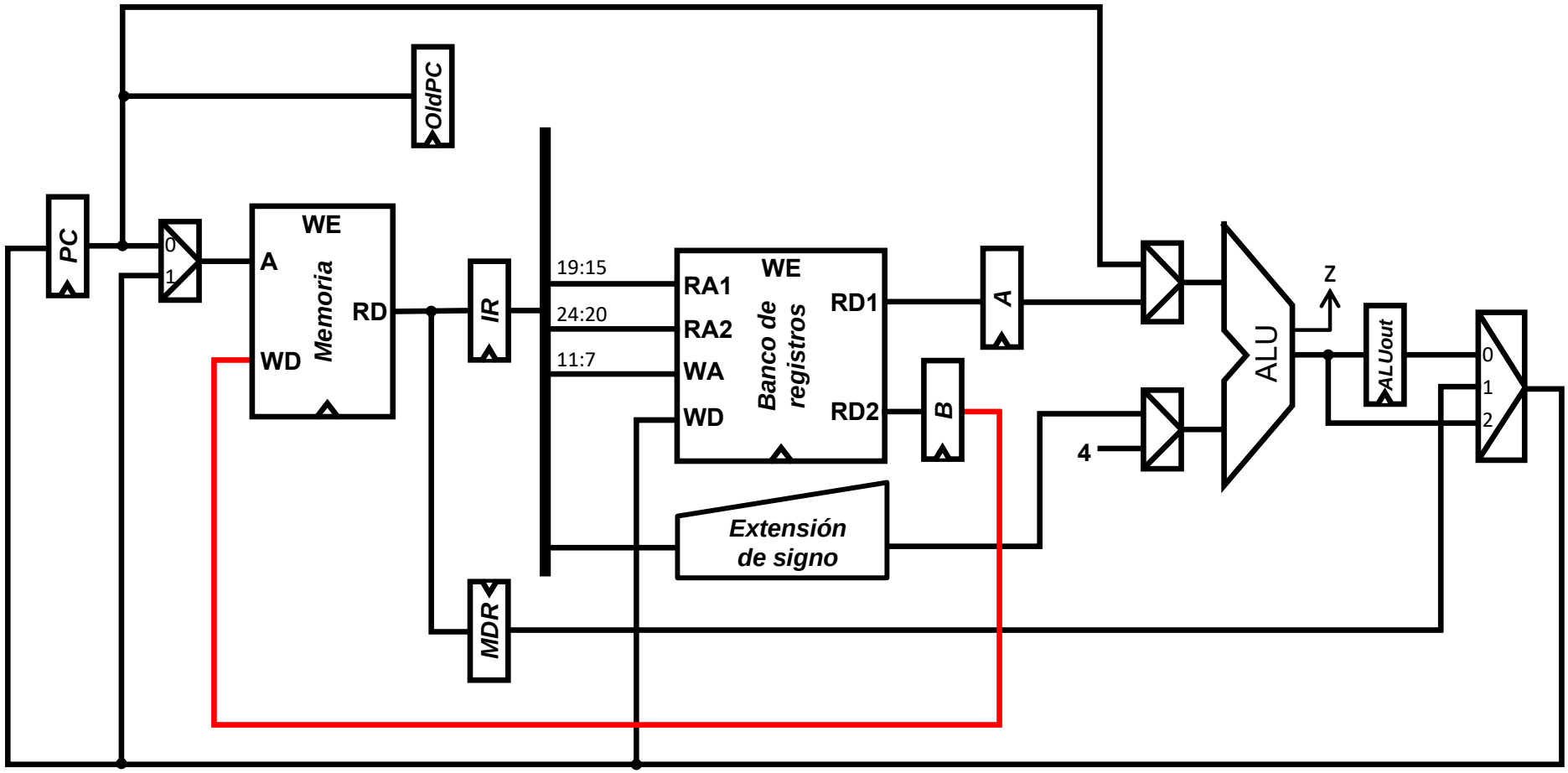


Mem[BR[rs1] + sExt(imm)] ← BR[rs2], PC ← PC+4

Diseño de la ruta de datos

Instrucción **sw**: almacenaje del operando

- Se guarda en Memoria el dato almacenado en B (leído previamente del Banco de Registros).

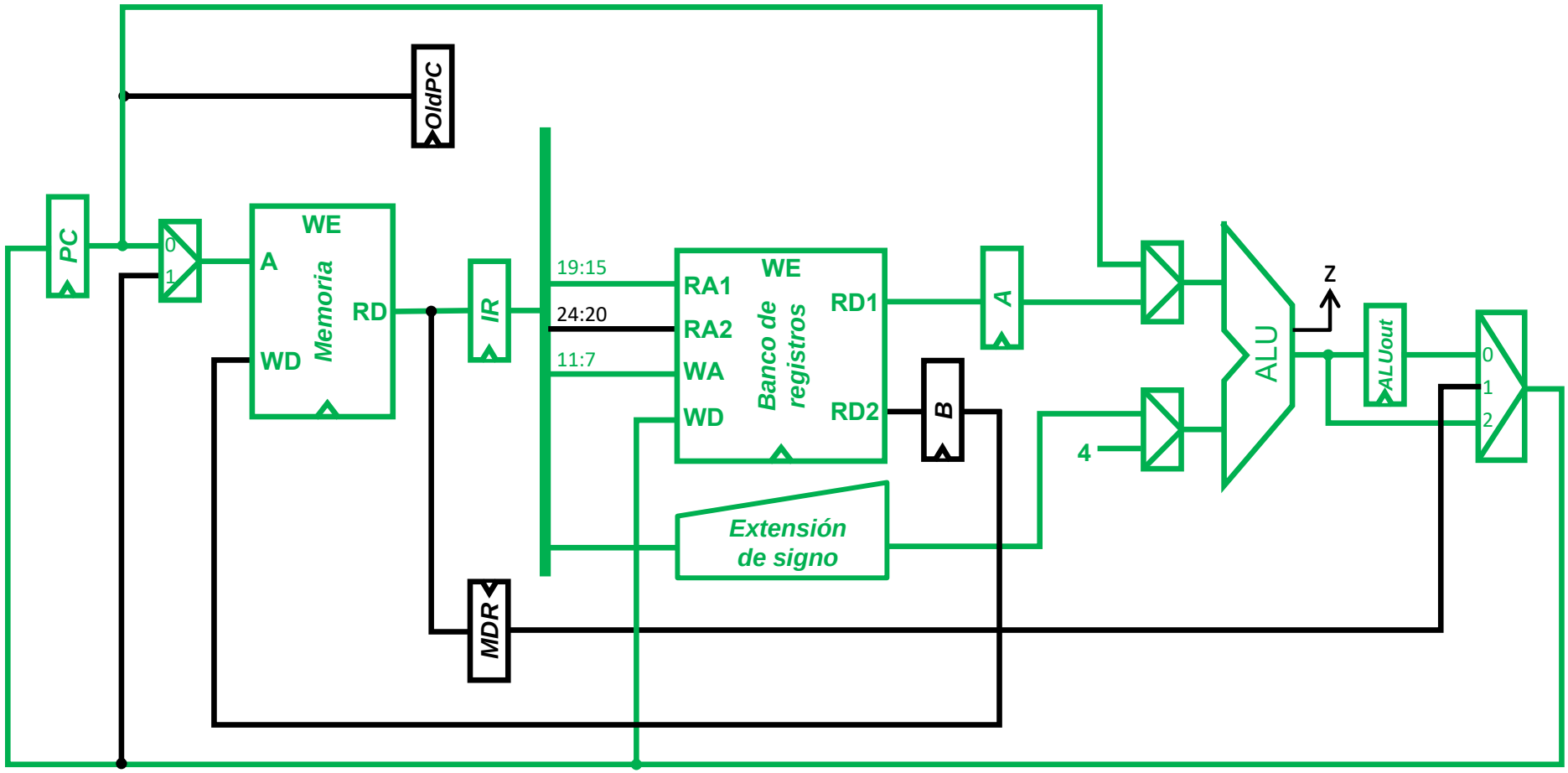


Mem[BR[rs1] + sExt(imm)] ← BR[rs2], PC ← PC+4

Diseño de la ruta de datos

Ruta de datos para instrucciones tipo **addi**

- Esta ruta de datos puede reutilizarse para efectuar todas las acciones de instrucciones aritmético-lógicas (tipo-I)

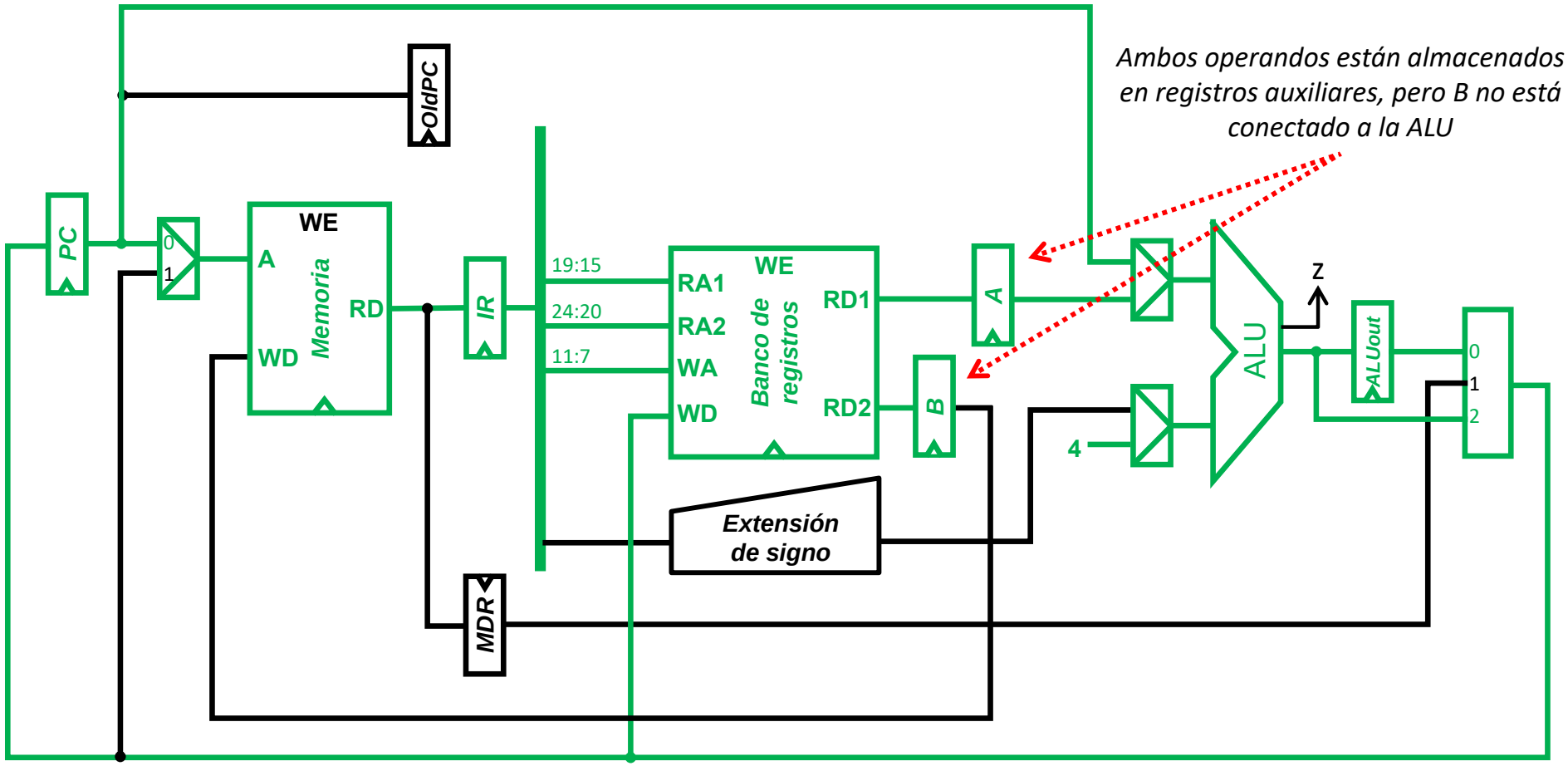


BR[rd] ← BR[rs1] *op* sExt(imm), PC ← PC+4

Diseño de la ruta de datos

Ruta de datos para instrucciones tipo **add**

- Esta ruta de datos puede reutilizarse para efectuar todas las acciones de las de instrucciones aritmético-lógicas (tipo-R) menos la operación en si.

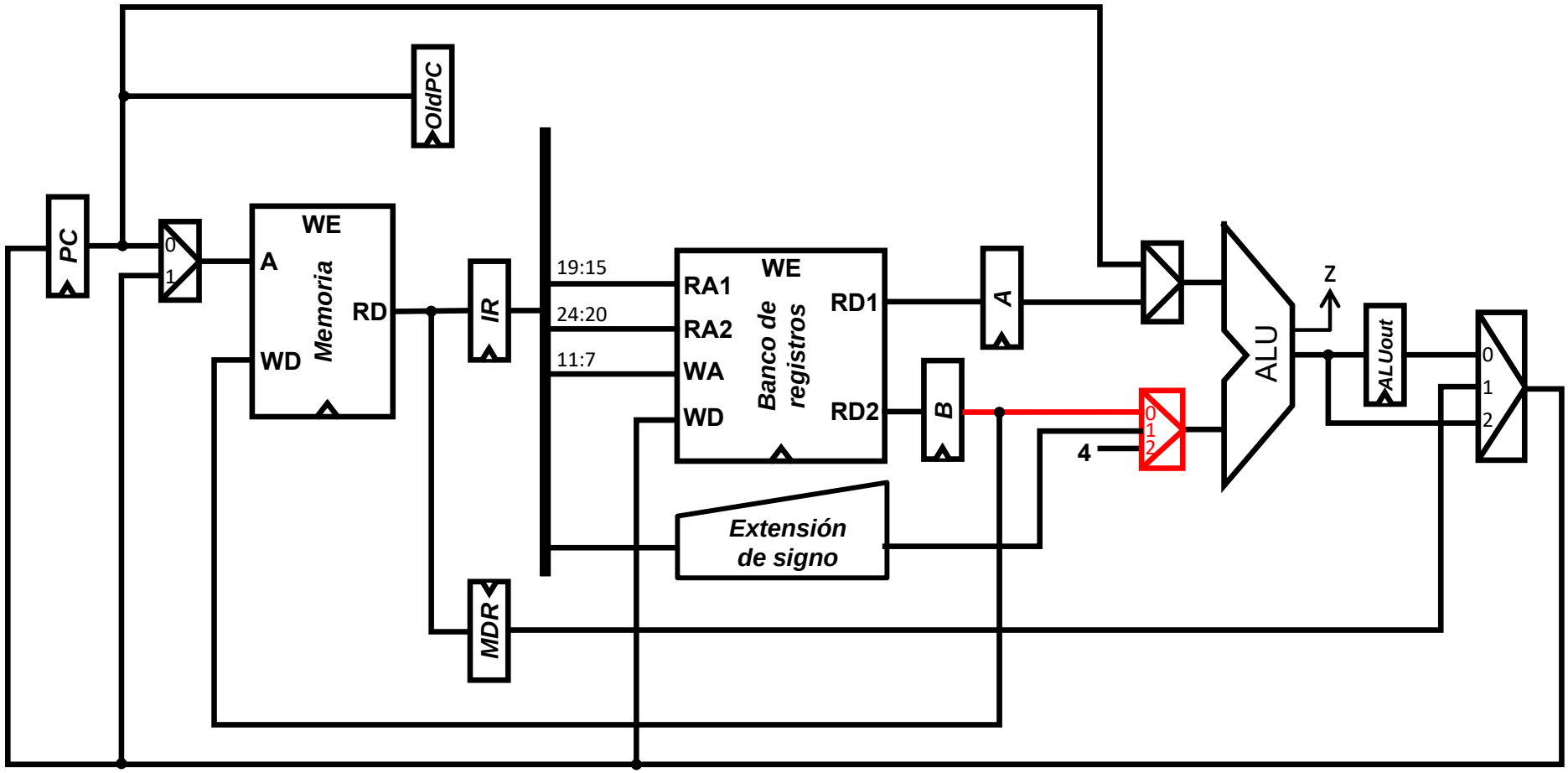


BR[rd] ← BR[rs1] *op* BR[rs2], PC ← PC+4

Diseño de la ruta de datos

Instrucciones tipo **add**: cálculo de operación

- Se amplia el multiplexor para reutilizar la ALU para que realice las operaciones aritmético lógicas con ambos operandos en el BR

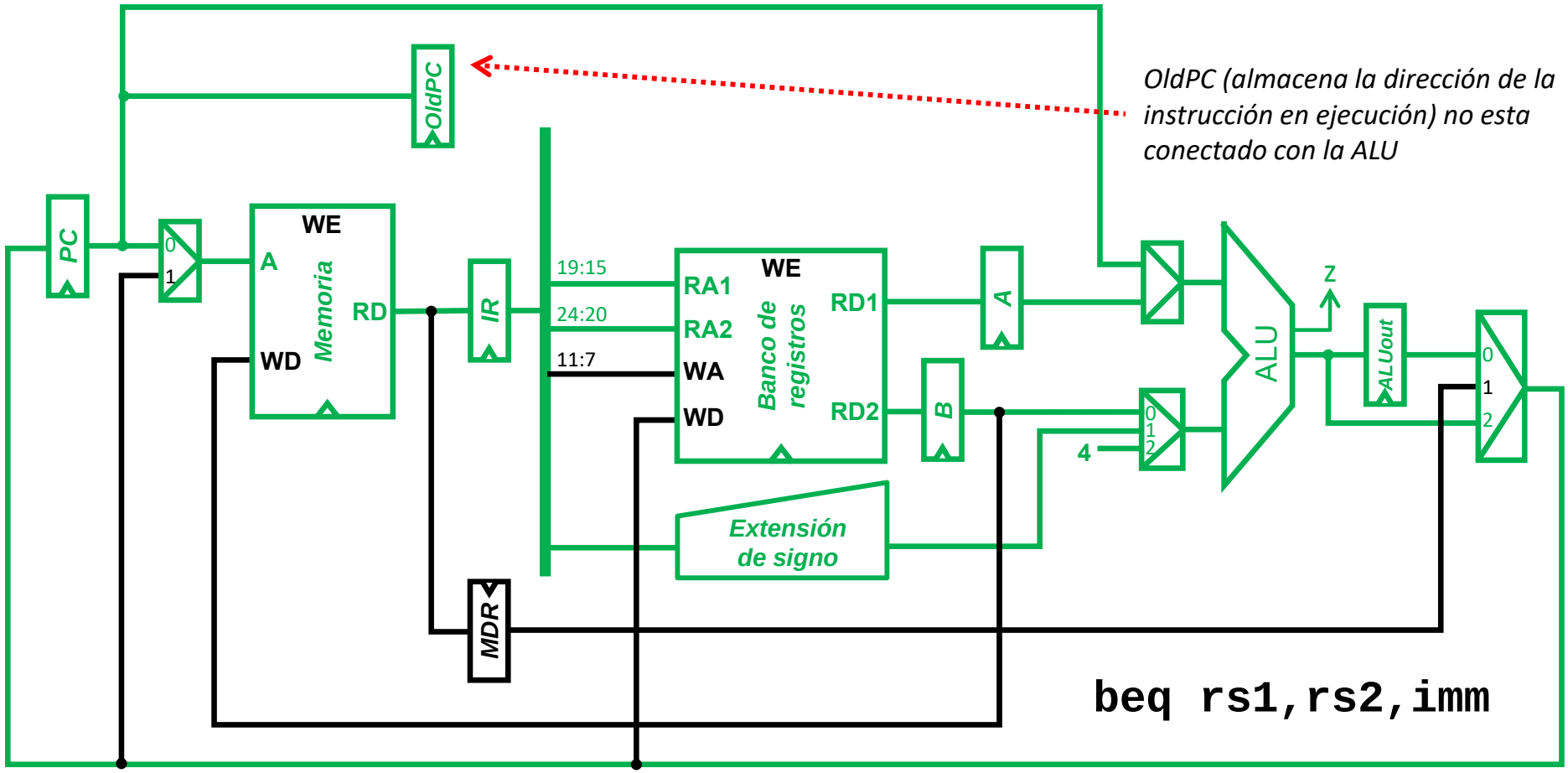


BR[rd] ← BR[rs1] *op* BR[rs2], PC ← PC+4

Diseño de la ruta de datos

Ruta de datos para instrucciones **beq**

- Esta ruta de datos puede reutilizarse para efectuar todas las acciones de la instrucción **beq** menos el cálculo de la dirección de salto

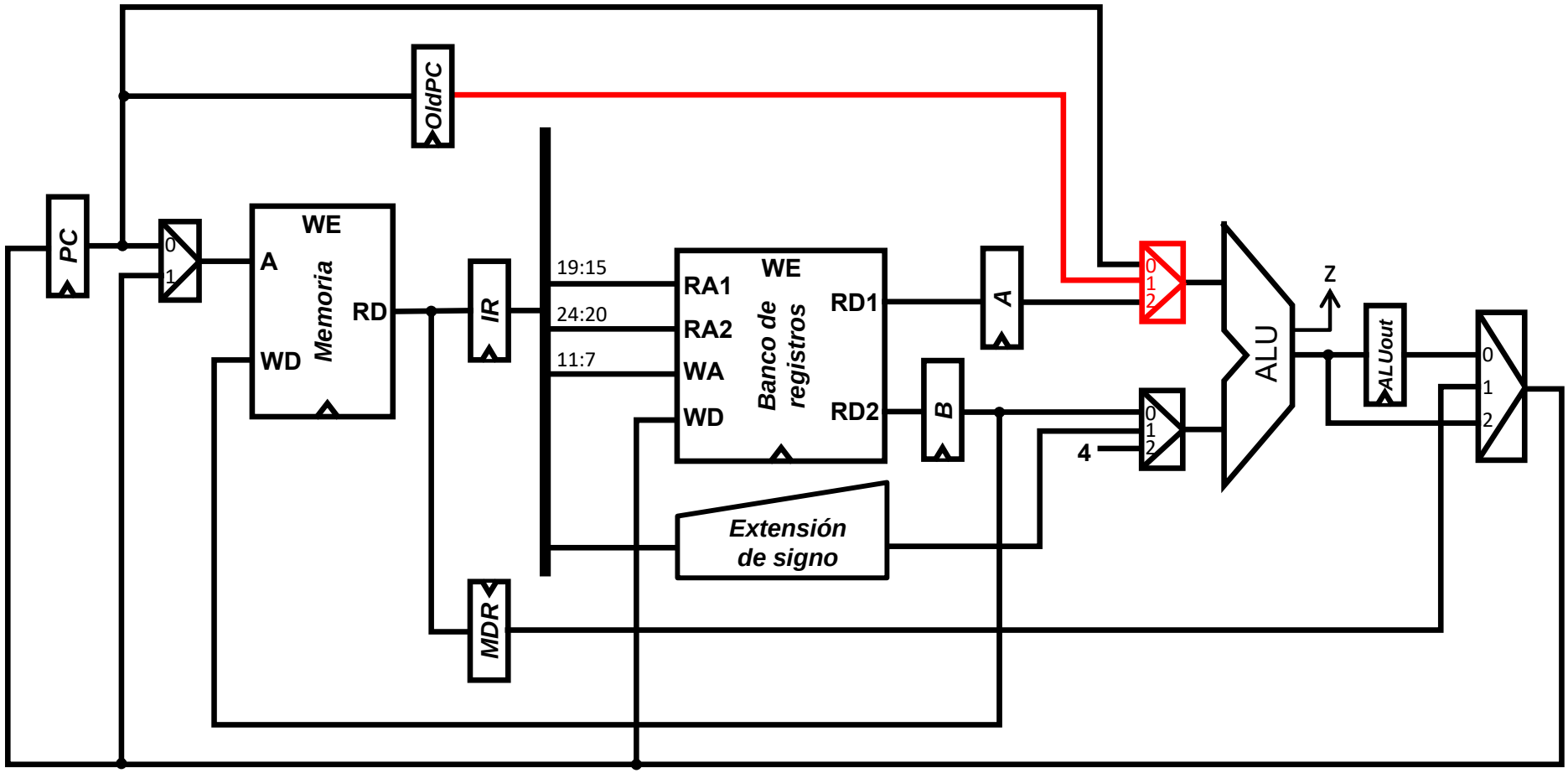


PC ← *if* (BR[rs1] = BR[rs2]) *then* (PC + sExt(imm)) *else* (PC+4)

Diseño de la ruta de datos

Instrucción **beq**: cálculo de la dirección de salto

- Se amplia el multiplexor para poder reutilizar la ALU para realizar el cálculo de la dirección de salto.

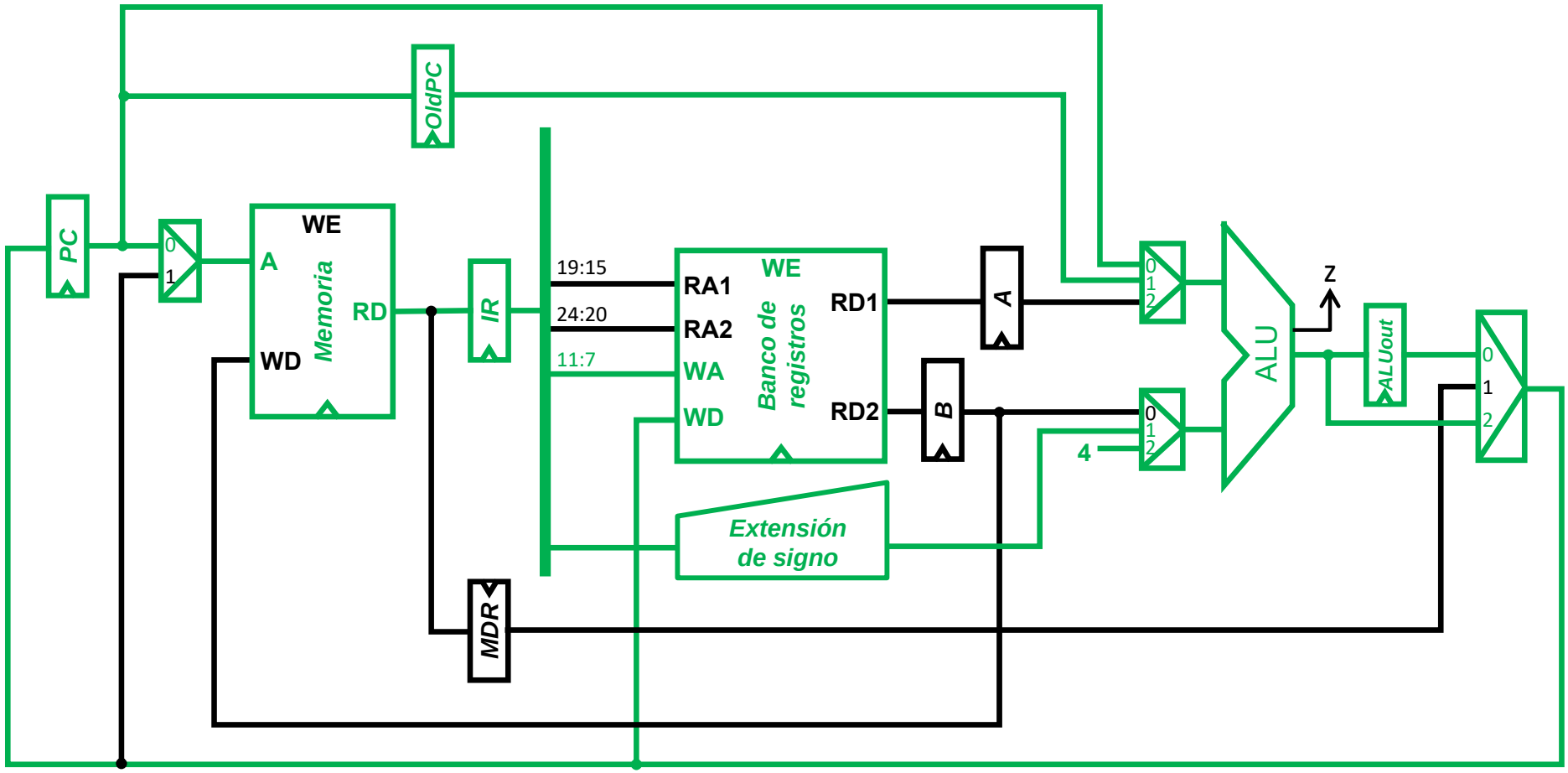


PC ← *if* (BR[rs1] = BR[rs2]) *then* (PC + sExt(imm)) *else* (PC+4)

Diseño de la ruta de datos

Ruta de datos para instrucciones `jal`

- Esta la ruta de datos puede reutilizarse para efectuar todas las acciones de la instrucción **`jal`**

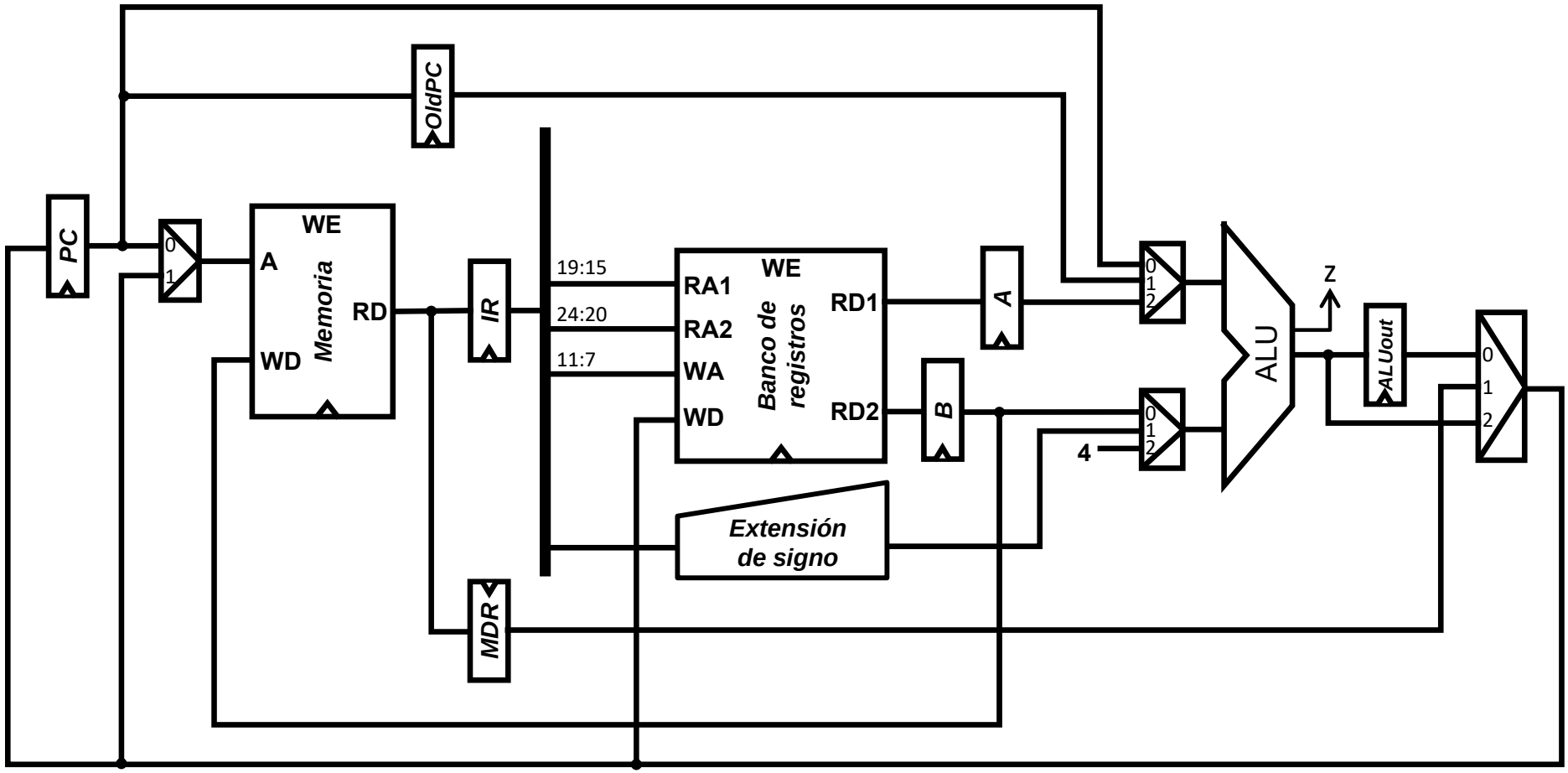


PC ← PC + sExt(imm), BR[rd] ← PC+4

Diseño de la ruta de datos

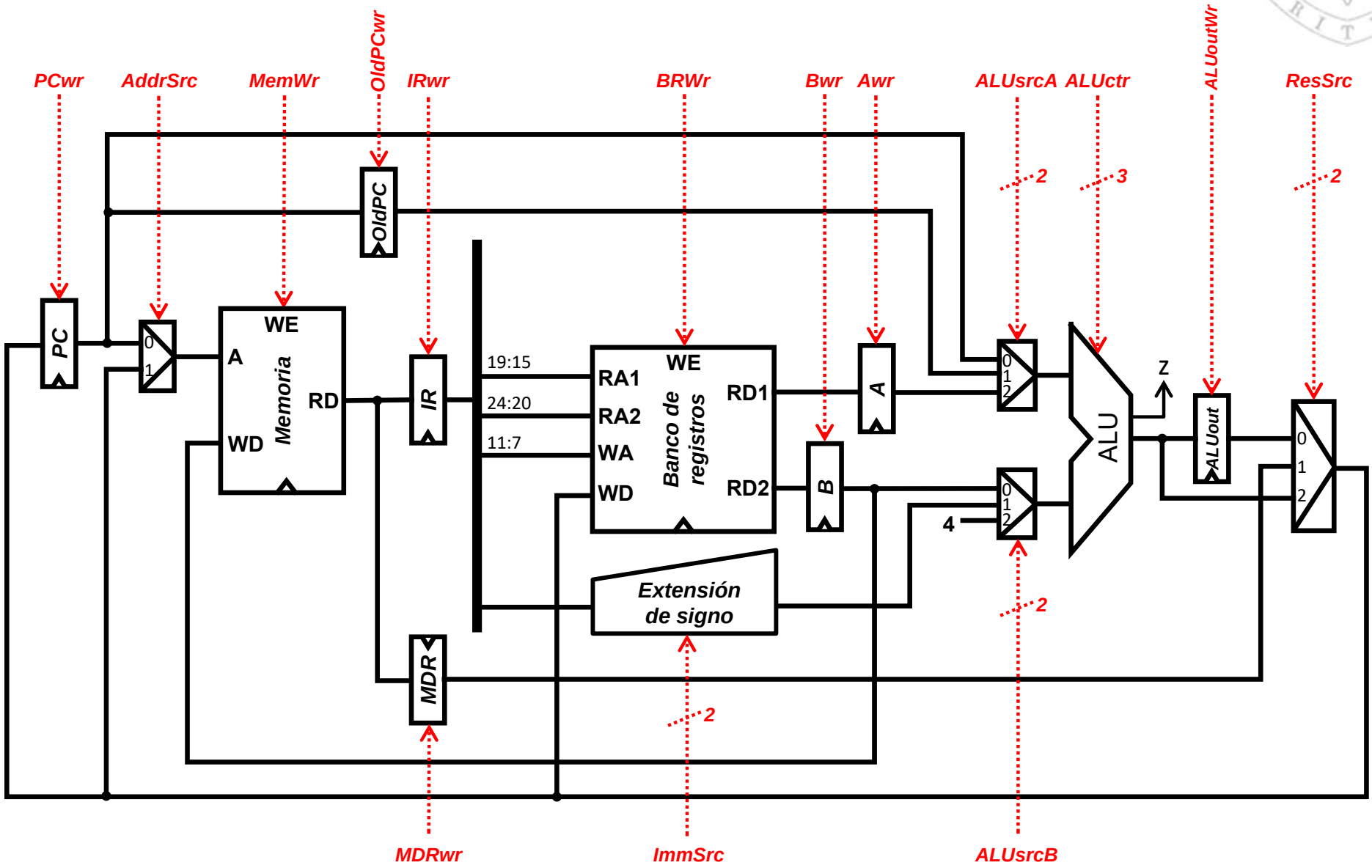
Ruta de datos RISC-V reducido

- Esta ruta de datos puede ejecutar cualquier secuencia de instrucciones del repertorio del RISC-V reducido.



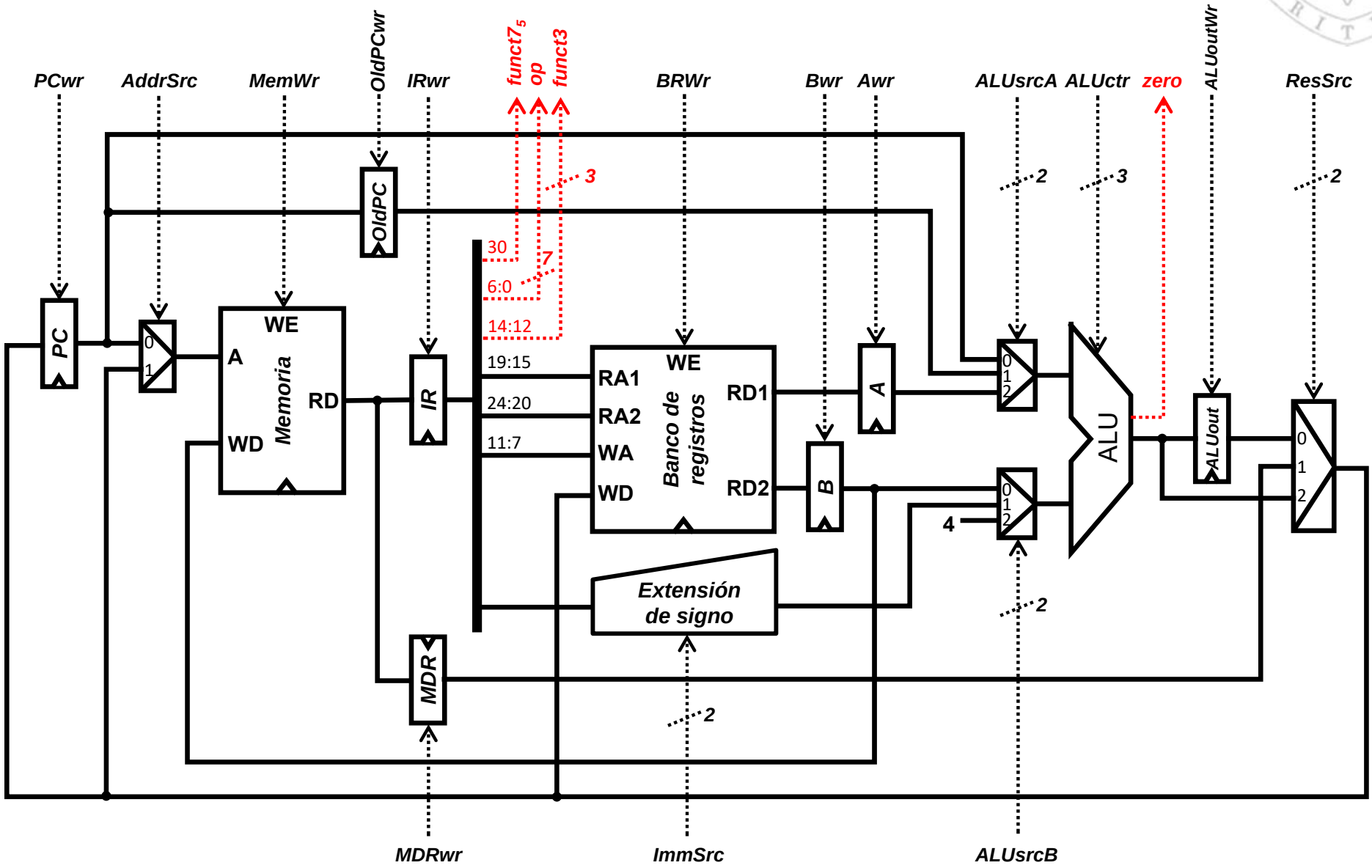
Diseño de la ruta de datos

Señales de control



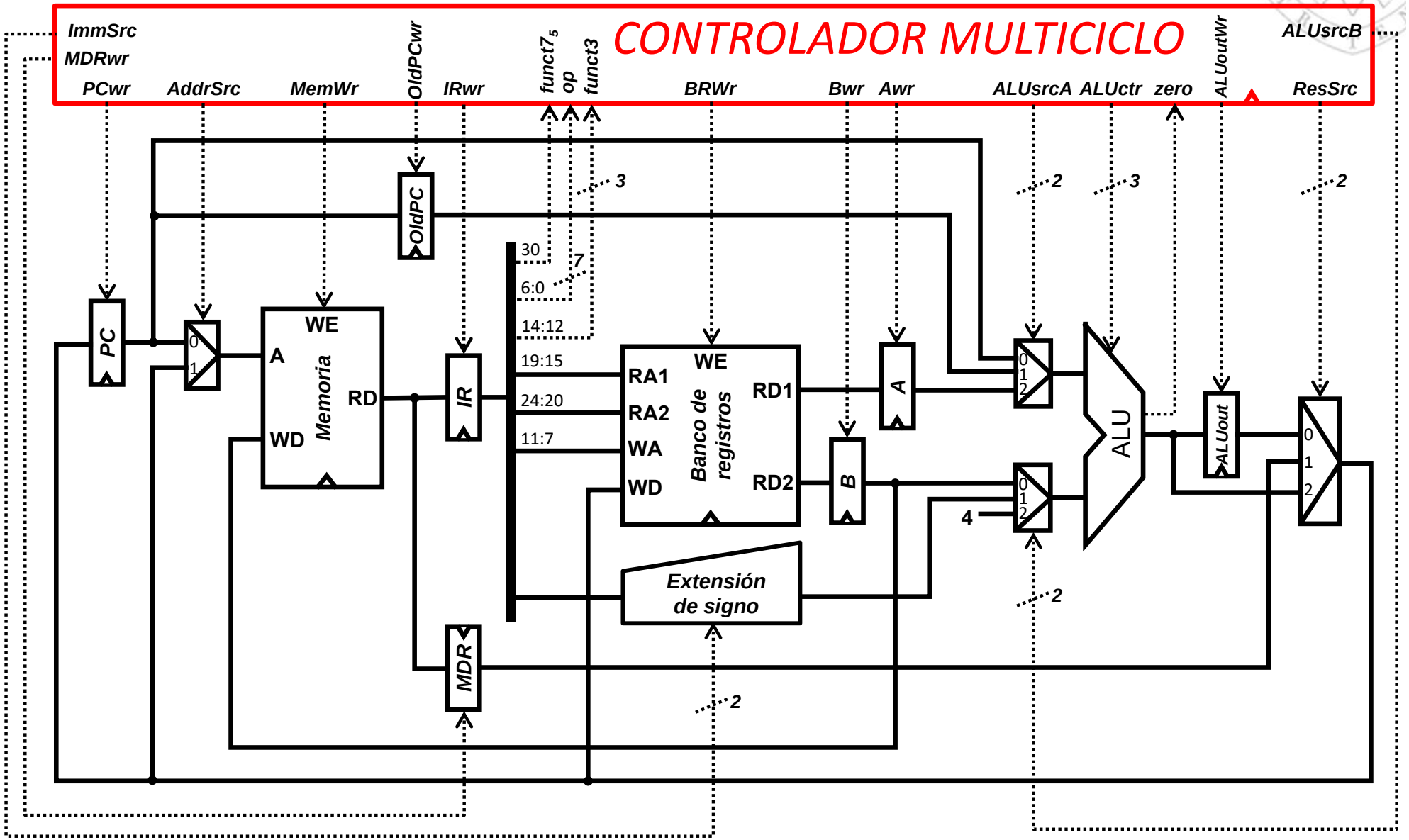
Diseño de la ruta de datos

Señales de estado



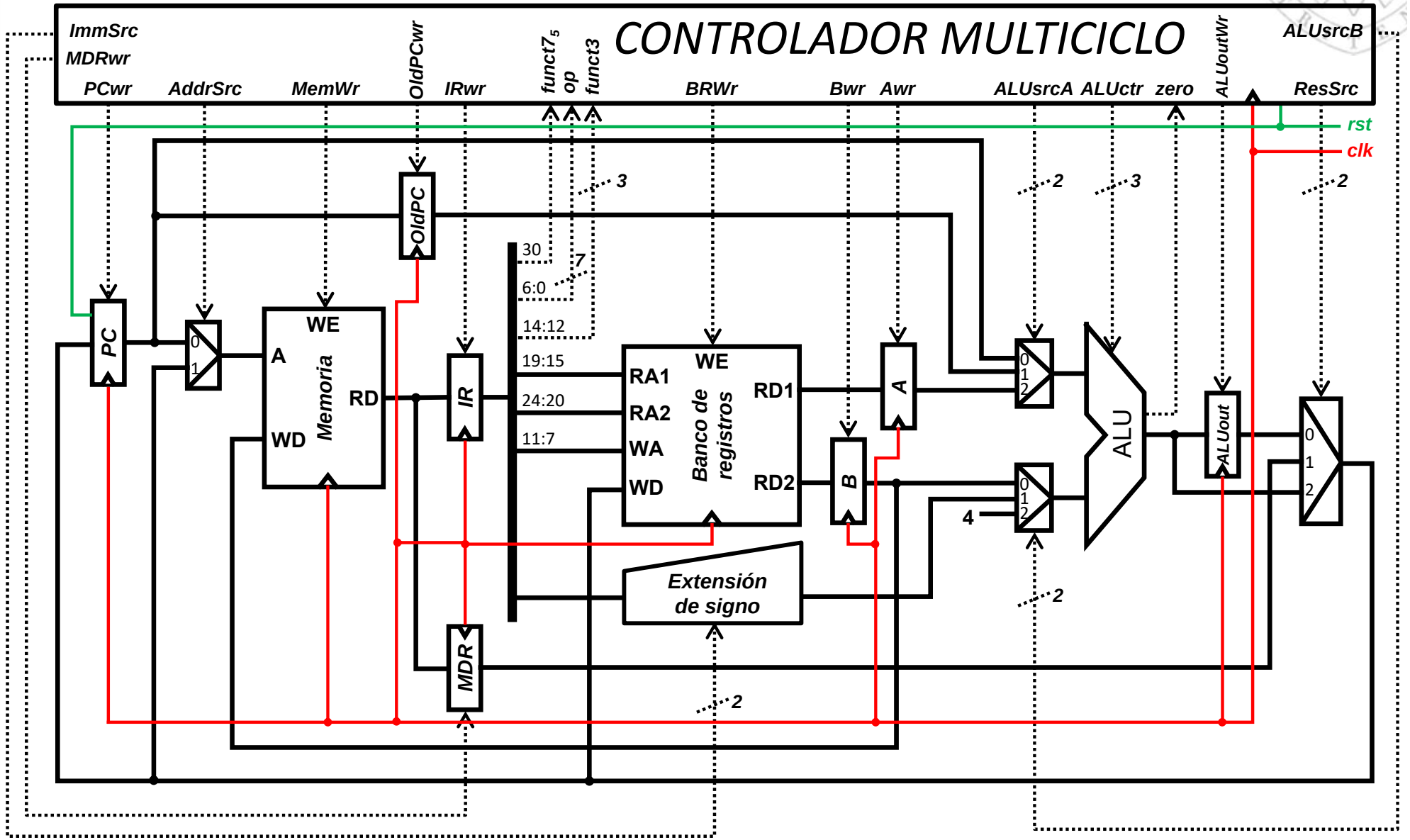
Diseño de la ruta de datos

Conexión con el controlador



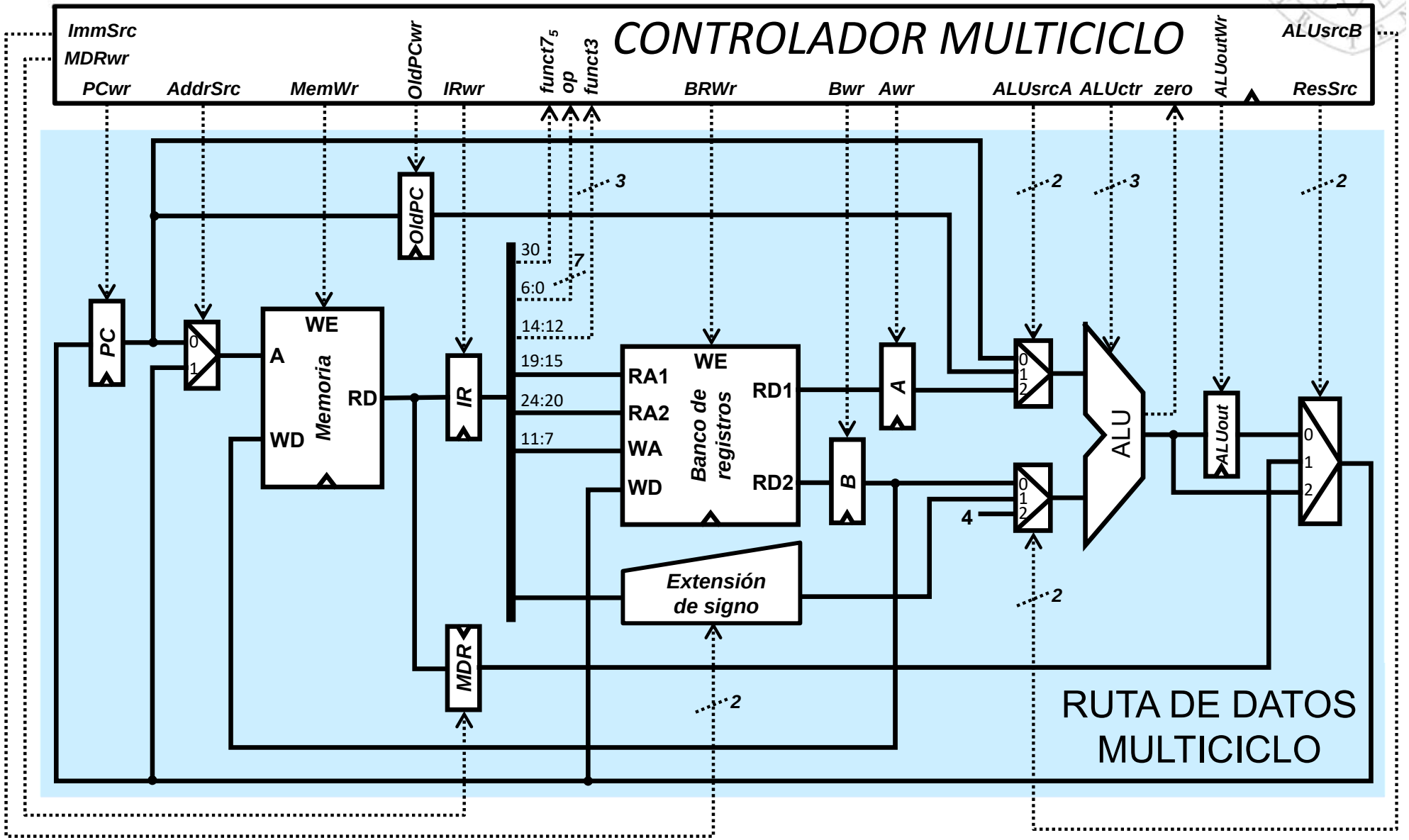
Diseño de la ruta de datos

Conexión de reloj y reset



Diseño de la ruta de datos

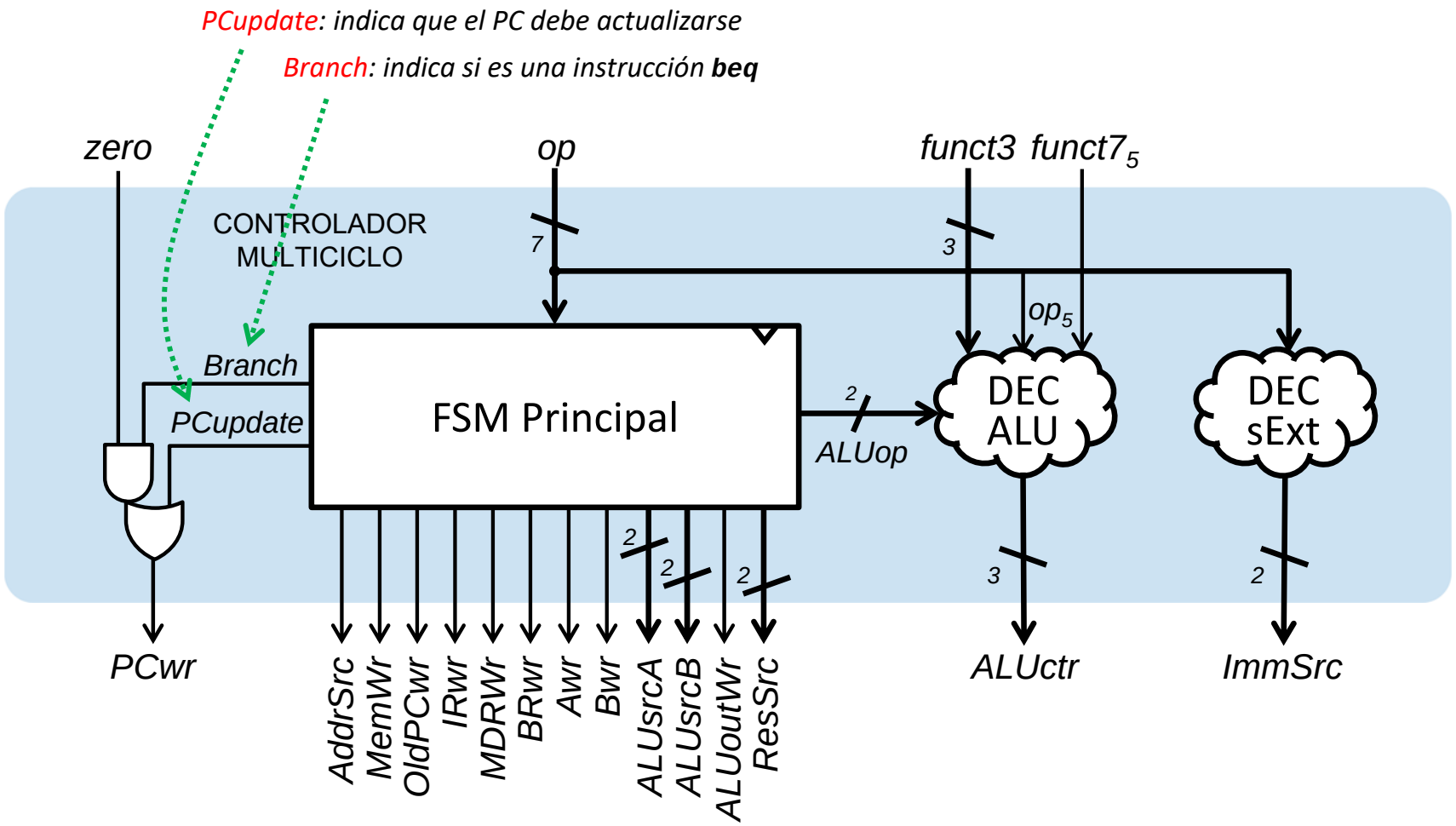
Estructura del sistema completo



Diseño del controlador

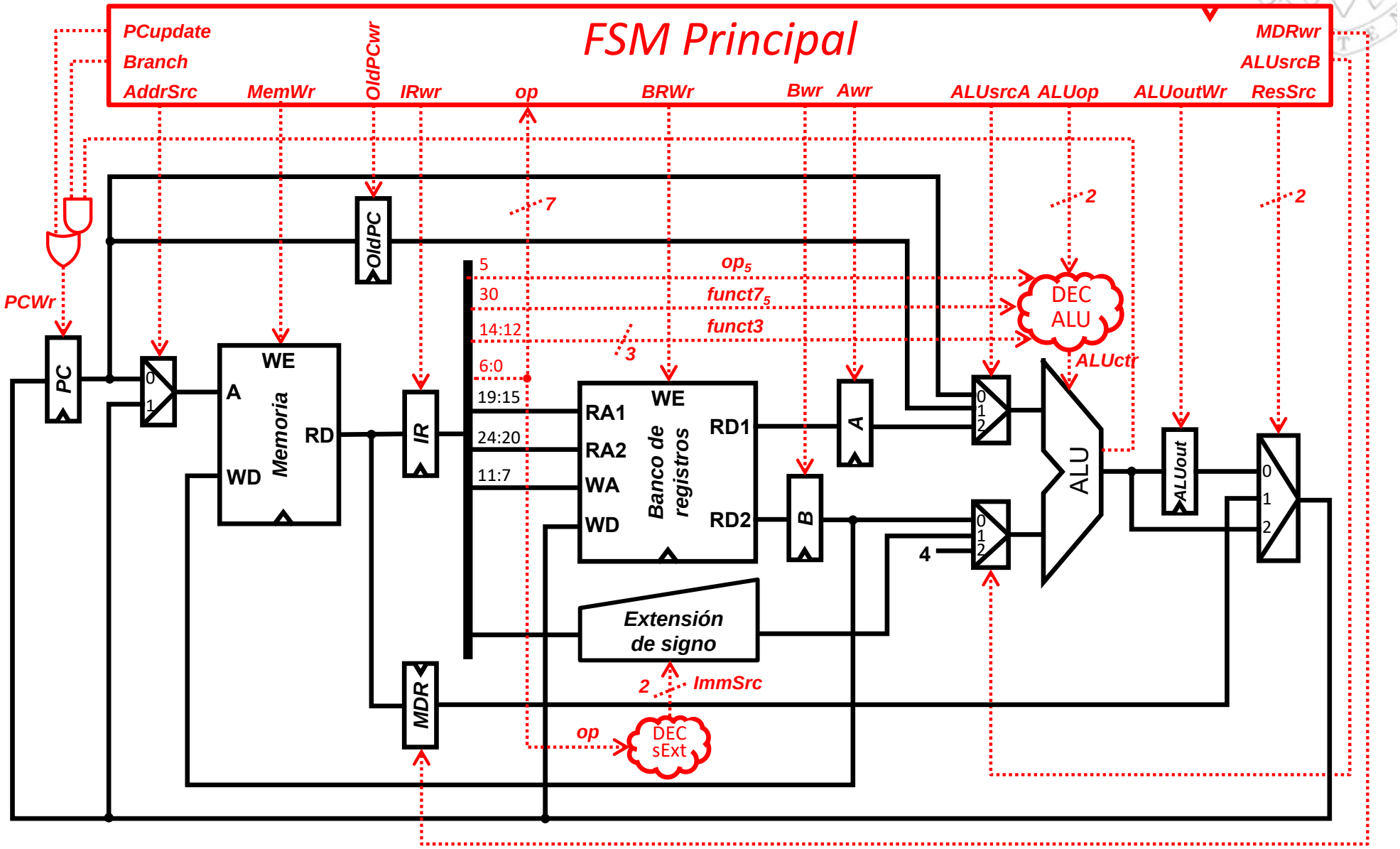
Estructura del controlador (i)

- En el procesador multiciclo, el controlador es un circuito secuencial:
  - Con una estructura en 4 subcircuitos análoga al controlador monociclo



Diseño del controlador

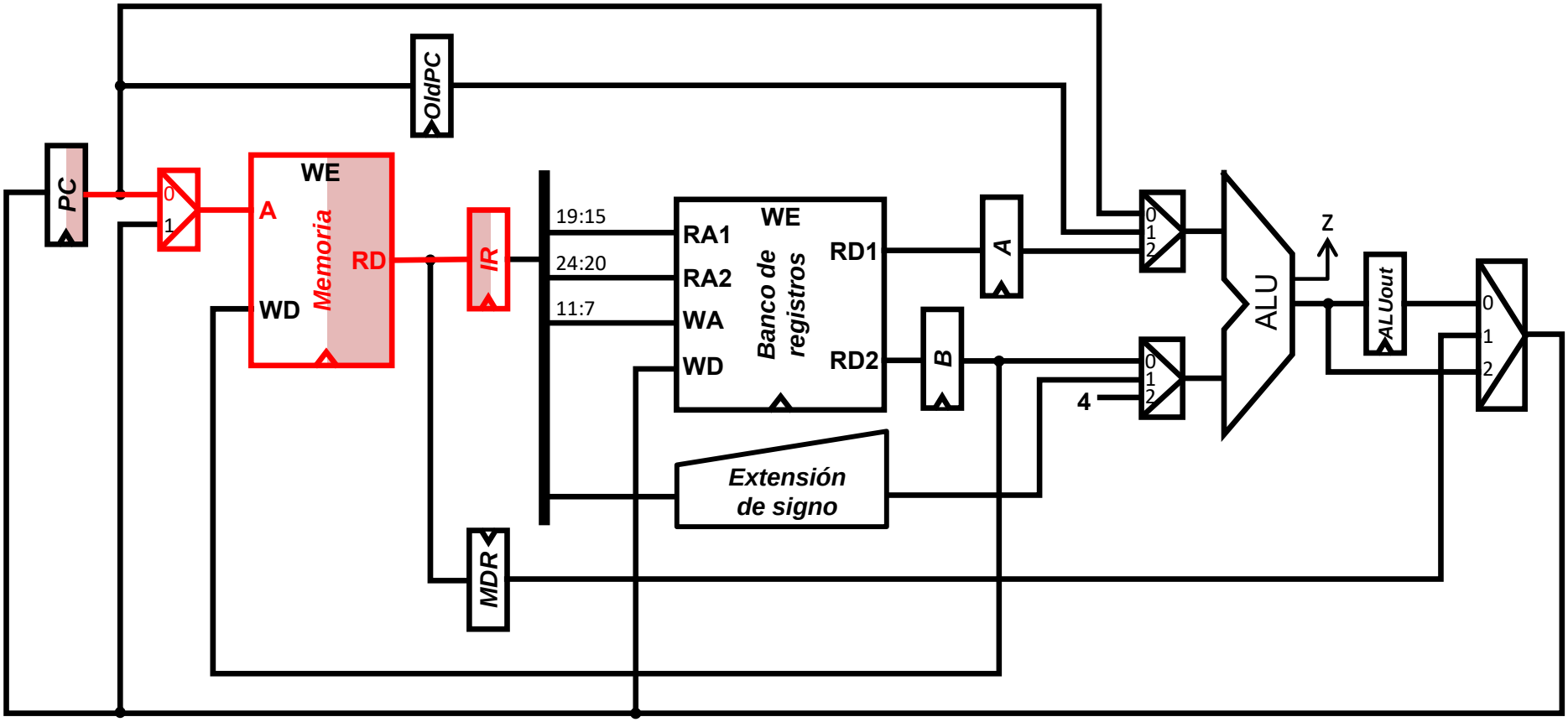
Estructura del controlador (ii)



Diseño del controlador

Instrucción `lw`: transferencias entre registros

1. IR ← Mem[PC]

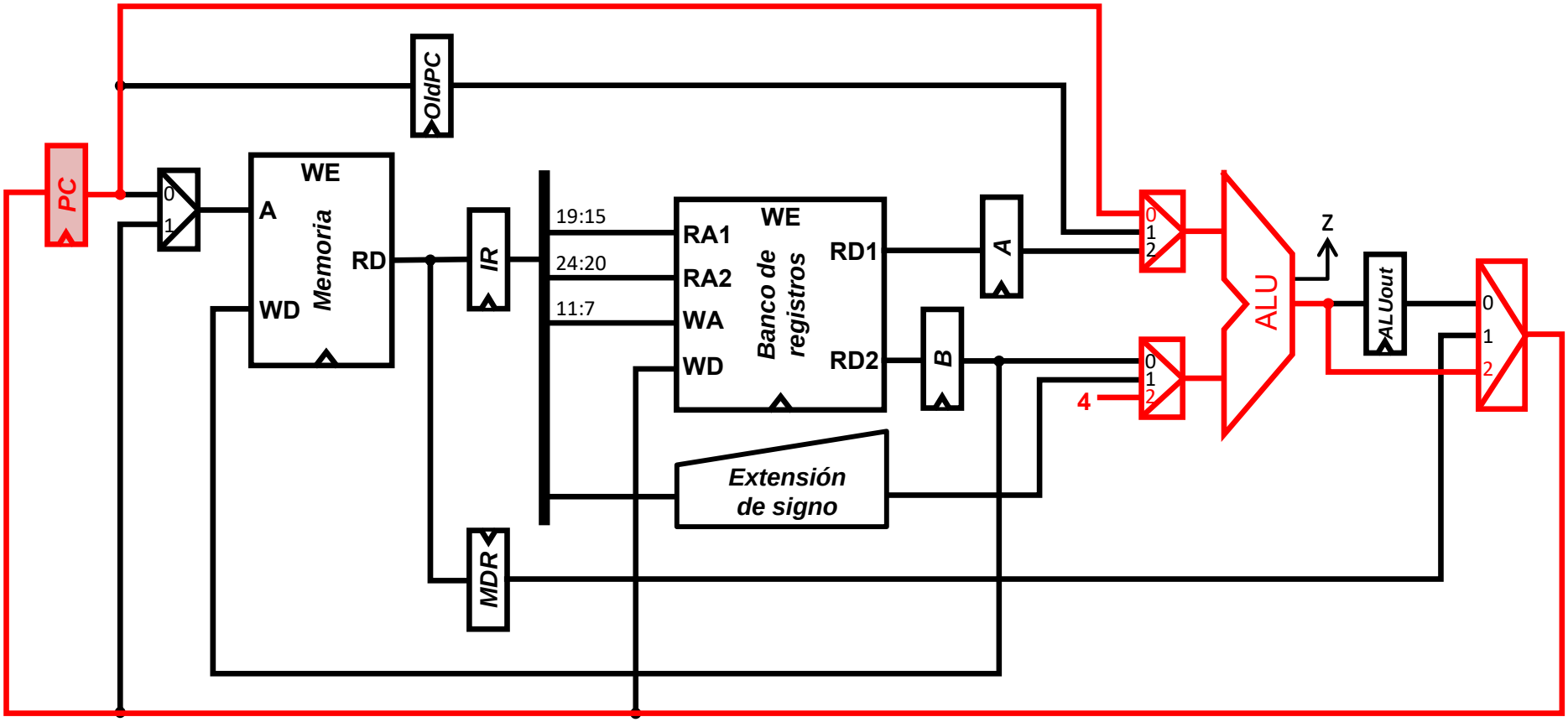


BR[rd] ← Mem[BR[rs1] + sExt(imm)], PC ← PC+4

Diseño del controlador

Instrucción `lw`: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4



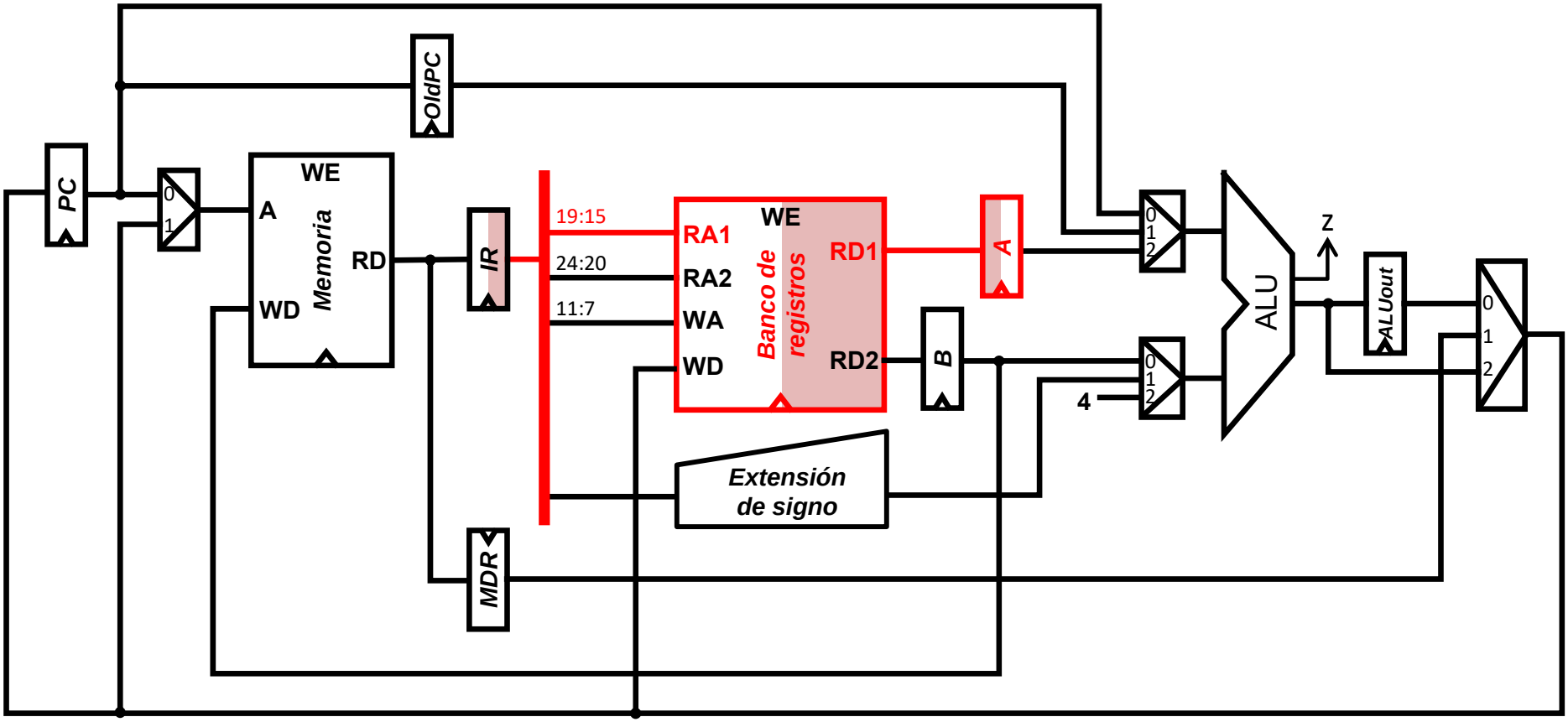
BR[rd] ← Mem[BR[rs1] + sExt(imm)], PC ← PC+4

Diseño del controlador

Instrucción `lw`: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4

2. A ← BR[rs1]

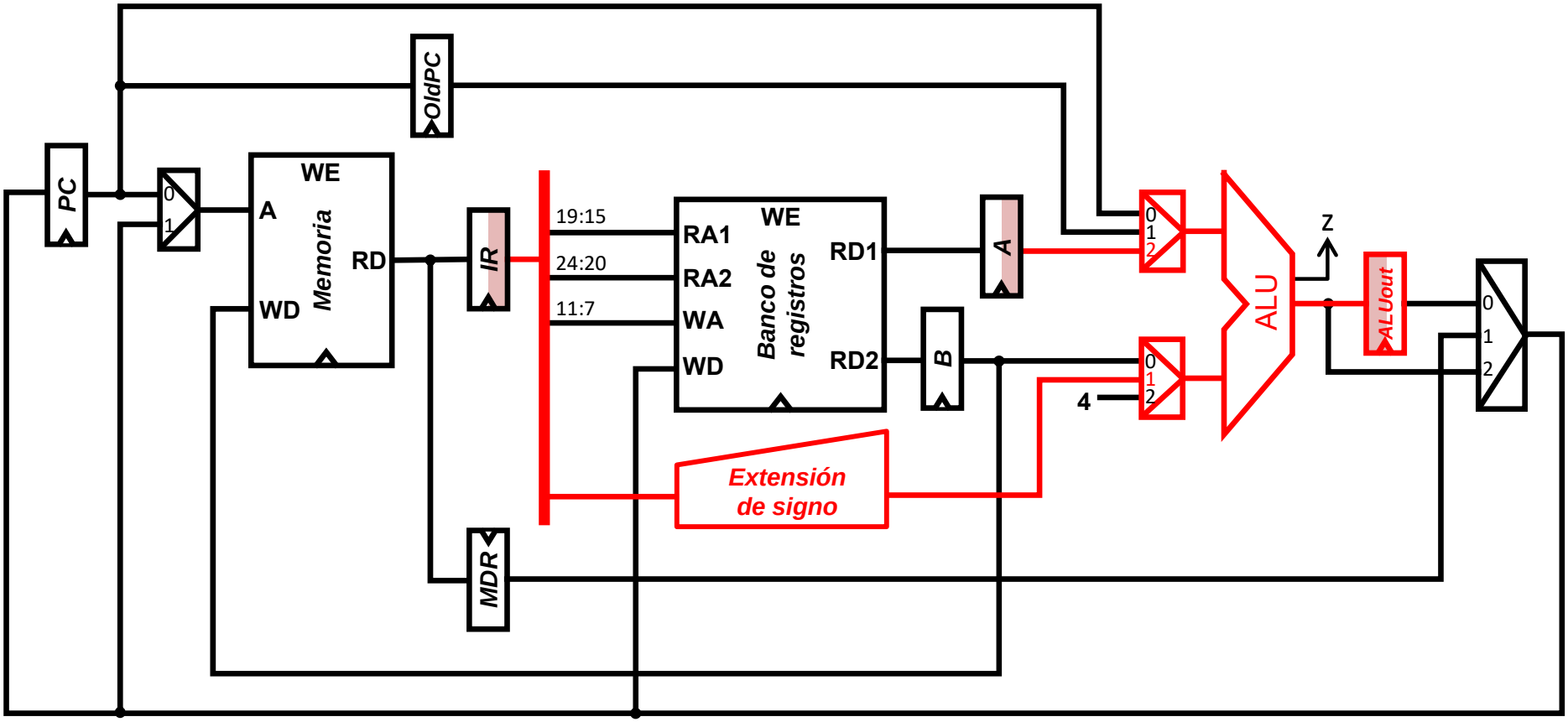


BR[rd] ← Mem[BR[rs1] + sExt(imm)], PC ← PC+4

Diseño del controlador

Instrucción `lw`: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4
2. A ← BR[rs1]
3. ALUout ← A + sExt(imm)



BR[rd] ← Mem[BR[rs1] + sExt(imm)], PC ← PC+4

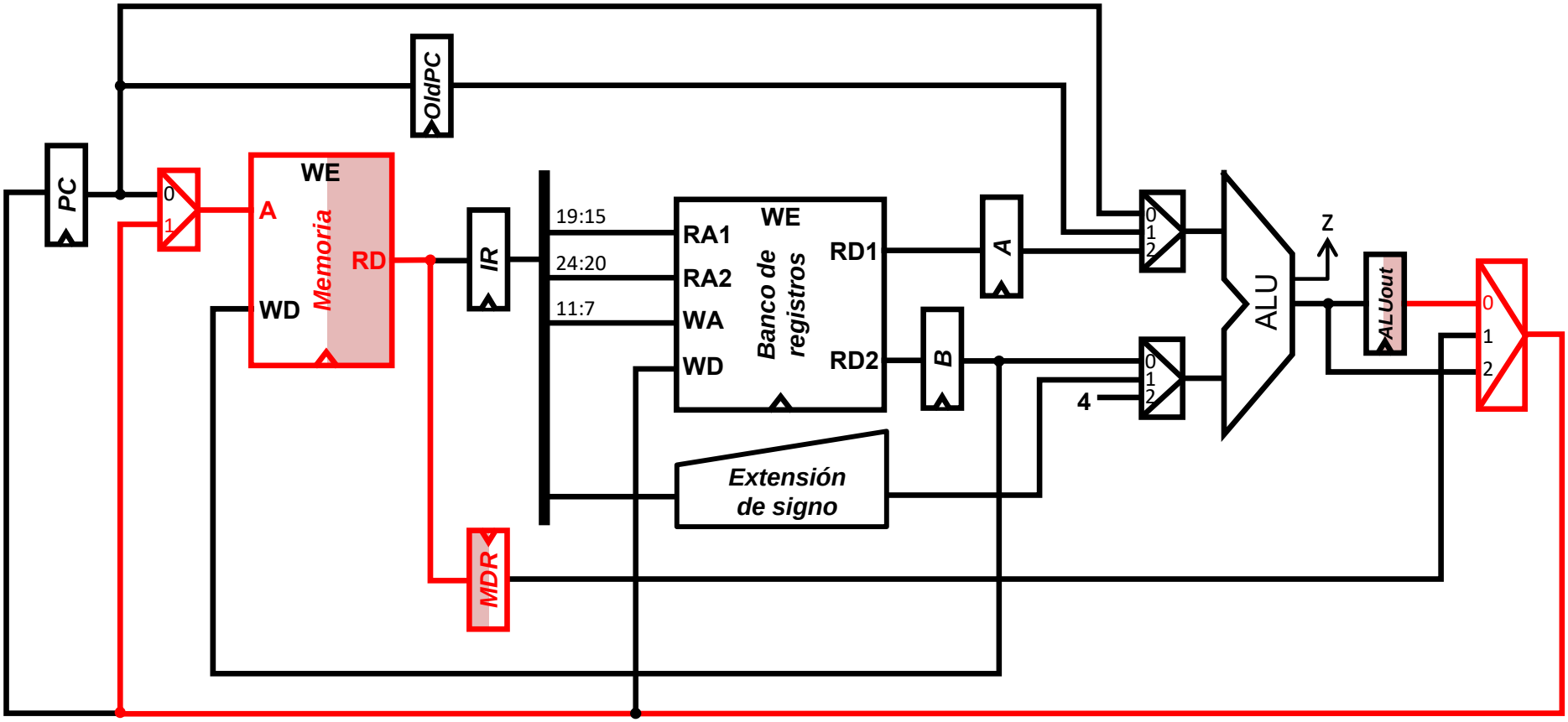
Diseño del controlador

Instrucción `lw`: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4 **4.** MDR ← Mem[ALUout]

2. A ← BR[rs1]

3. ALUout ← A + sExt(imm)



BR[rd] ← Mem[BR[rs1] + sExt(imm)], PC ← PC+4

Diseño del controlador

Instrucción `lw`: transferencias entre registros

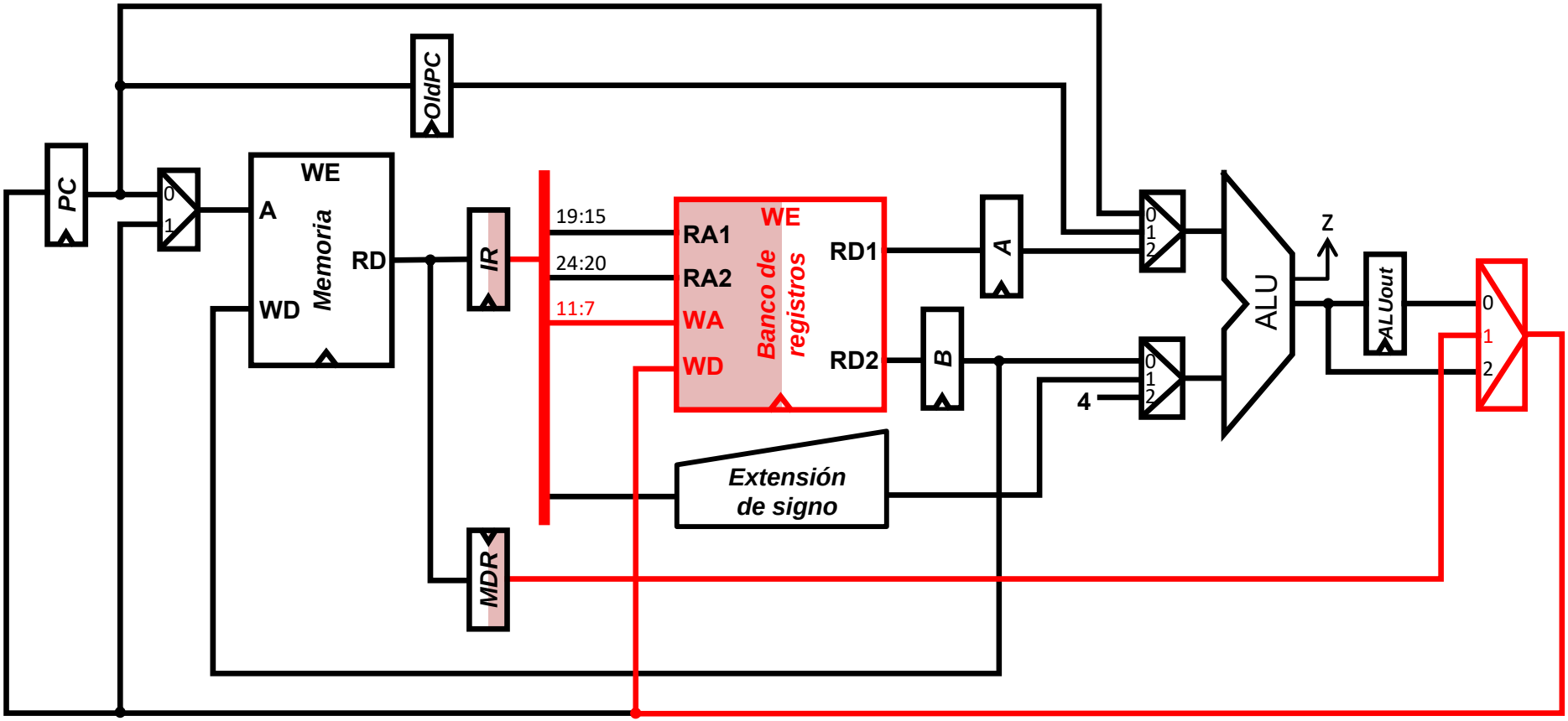
1. IR ← Mem[PC], PC ← PC + 4

2. A ← BR[rs1]

3. ALUout ← A + sExt(imm)

4. MDR ← Mem[ALUout]

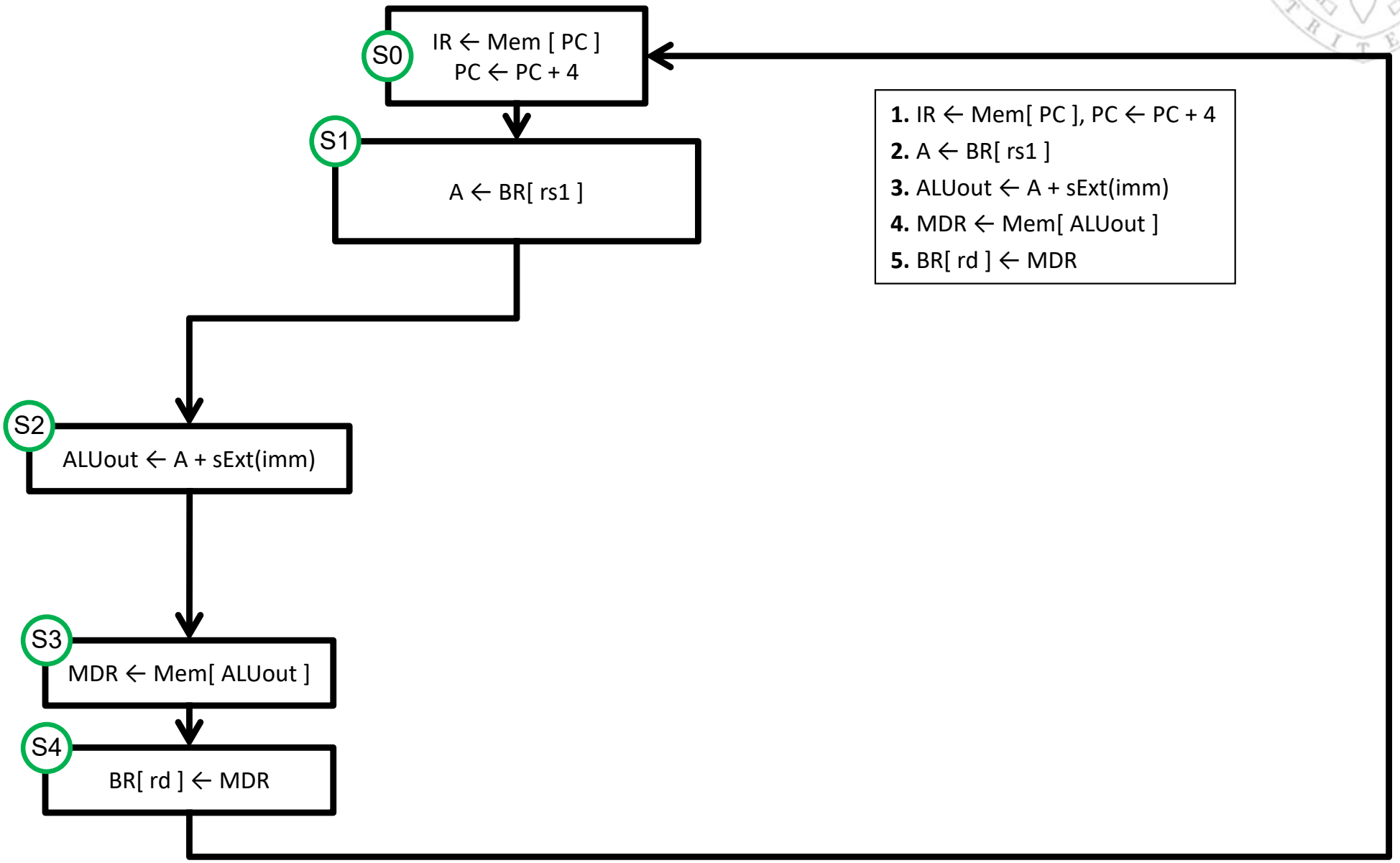
5. BR[rd] ← MDR



BR[rd] ← Mem[BR[rs1] + sExt(imm)], PC ← PC+4

Diseño del controlador

Diagrama ASM de la FSM principal: instrucción `lw`

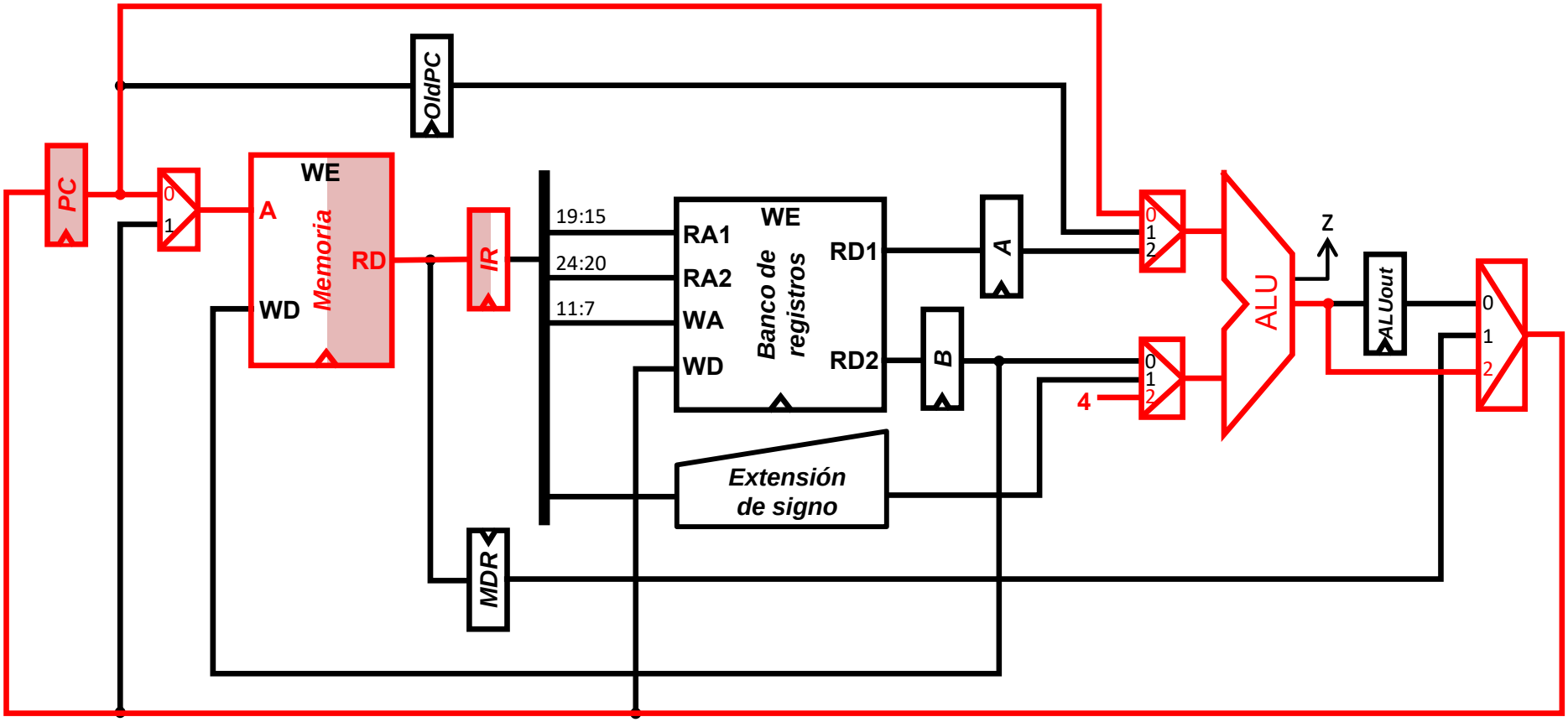


1. IR ← Mem[PC], PC ← PC + 4
2. A ← BR[rs1]
3. ALUout ← A + sExt(imm)
4. MDR ← Mem[ALUout]
5. BR[rd] ← MDR

Diseño del controlador

Instrucción **sw**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4



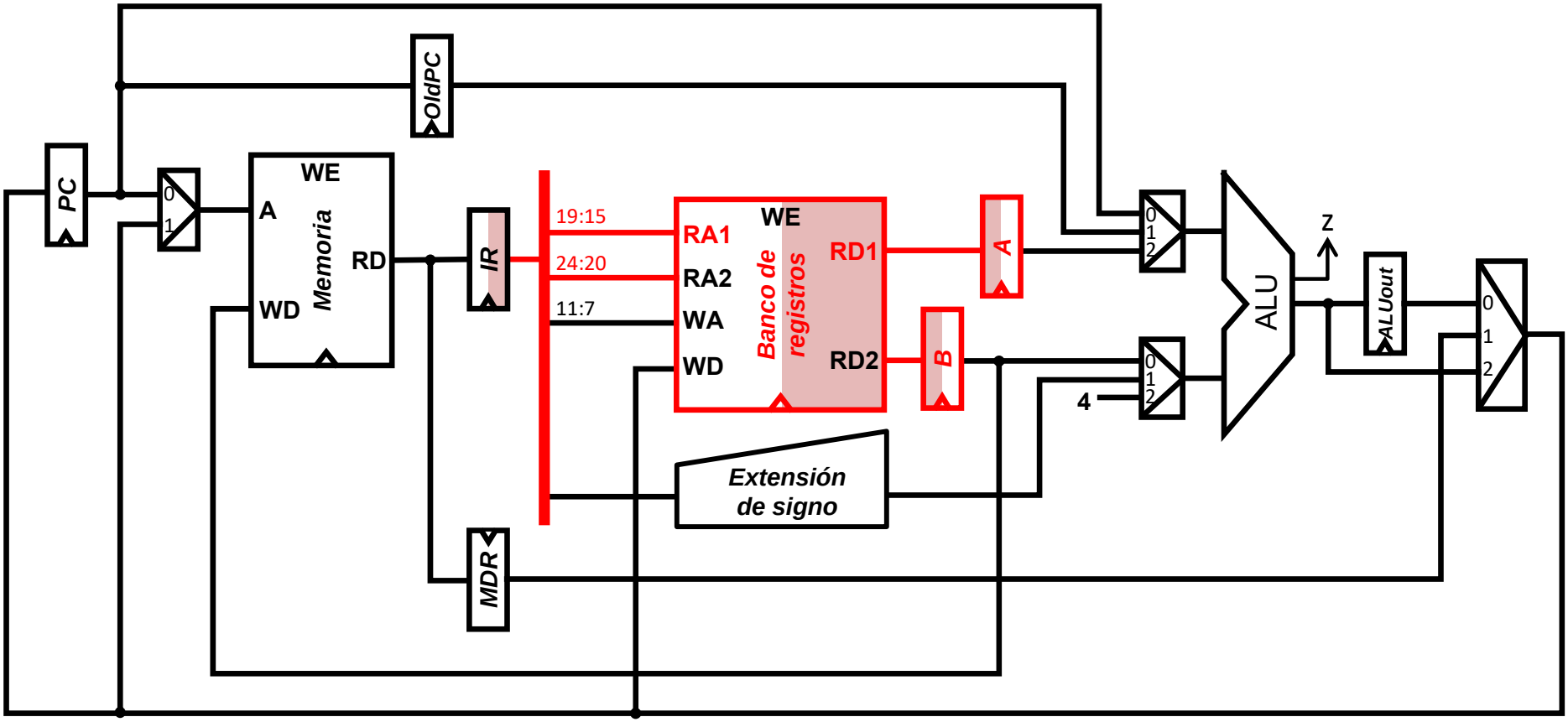
Mem[BR[rs1] + sExt(imm)] ← BR[rs2], PC ← PC+4

Diseño del controlador

Instrucción **sw**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4

2. A ← BR[rs1], B ← BR[rs2]



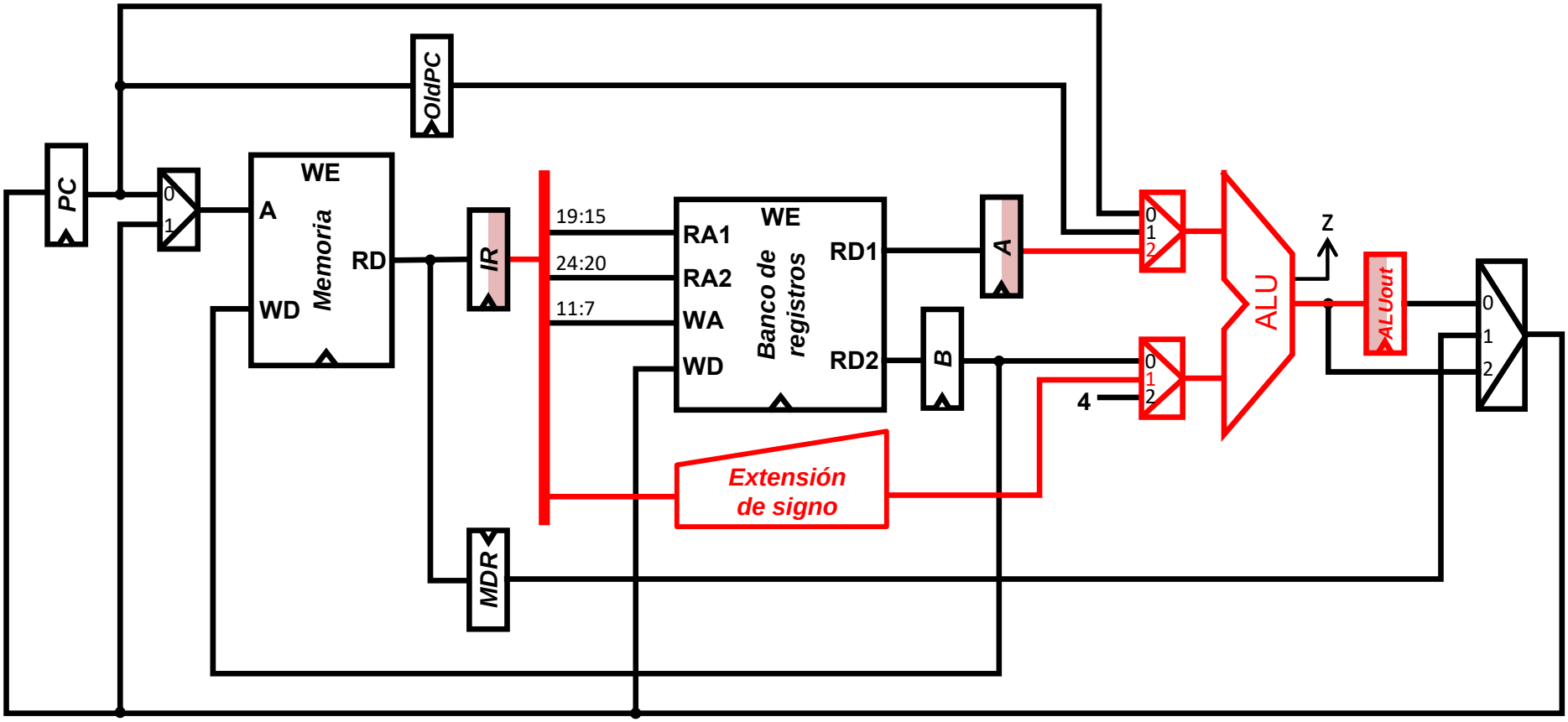
Mem[BR[rs1] + sExt(imm)] ← BR[rs2], PC ← PC+4

Diseño del controlador

Instrucción **sw**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4 **3.** ALUout ← A + sExt(imm)

2. A ← BR[rs1], B ← BR[rs2]



Mem[BR[rs1] + sExt(imm)] ← BR[rs2], PC ← PC+4

Diseño del controlador

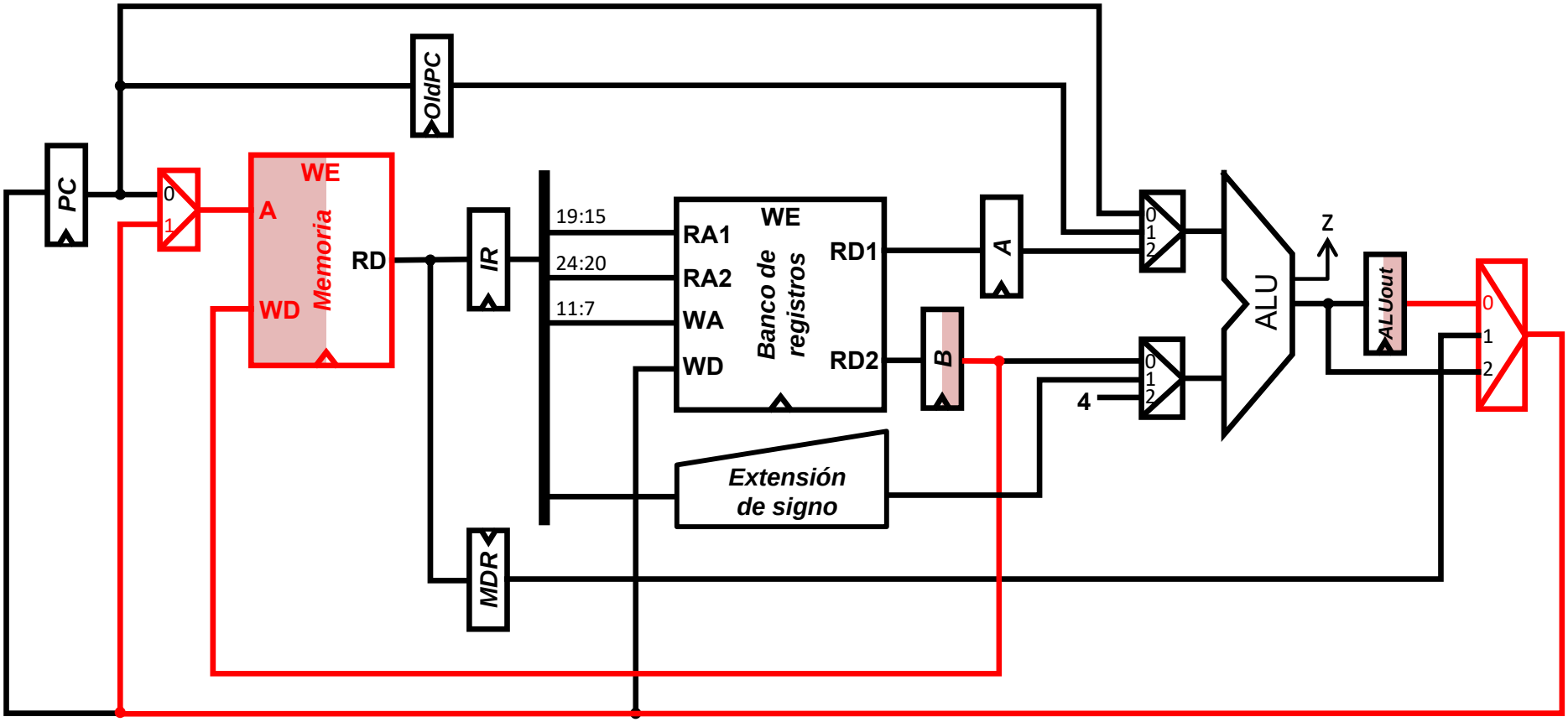
Instrucción **sw**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4

2. A ← BR[rs1], B ← BR[rs2]

3. ALUout ← A + sExt(imm)

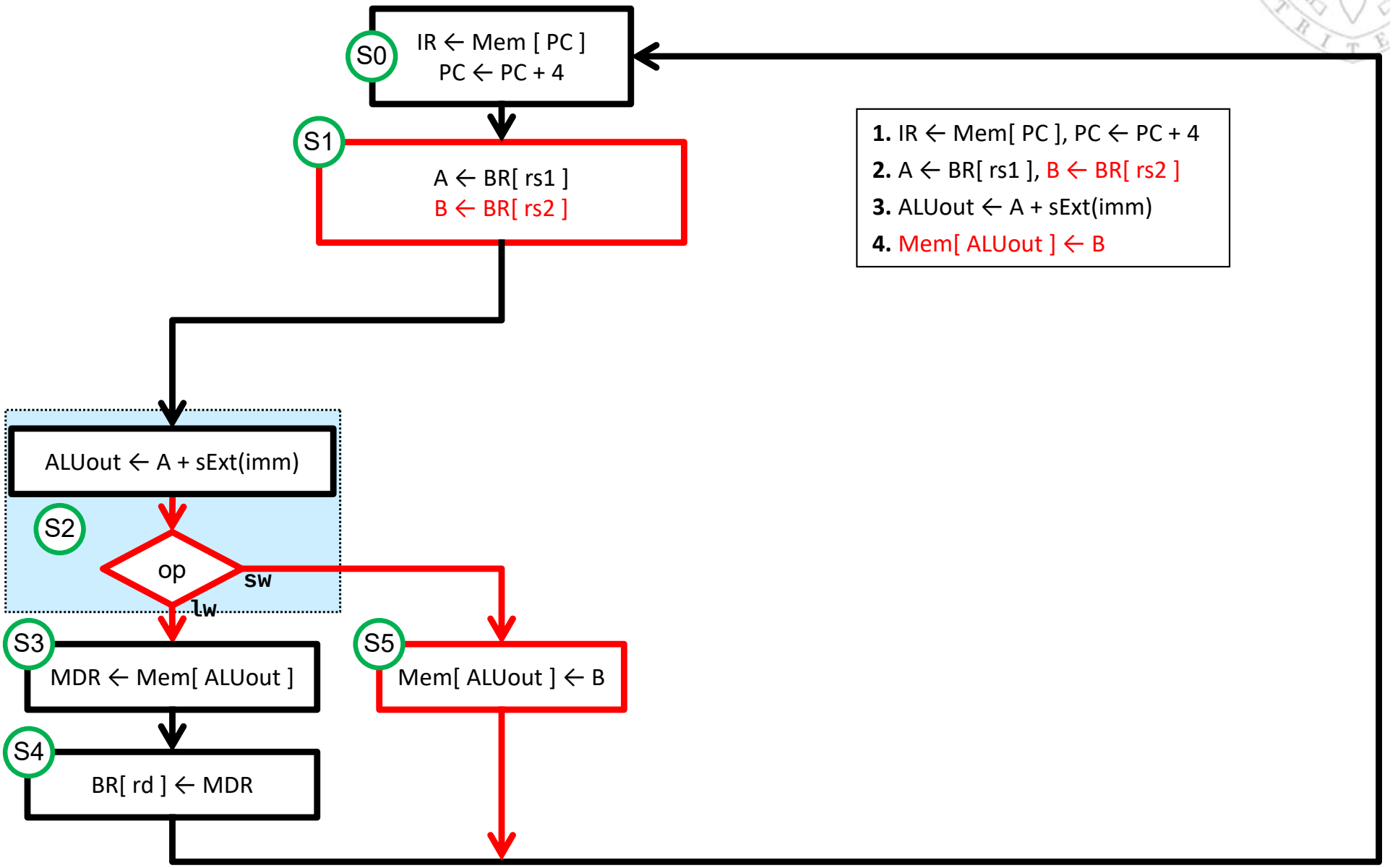
4. Mem[ALUout] ← B



Mem[BR[rs1] + sExt(imm)] ← BR[rs2], PC ← PC+4

Diseño del controlador

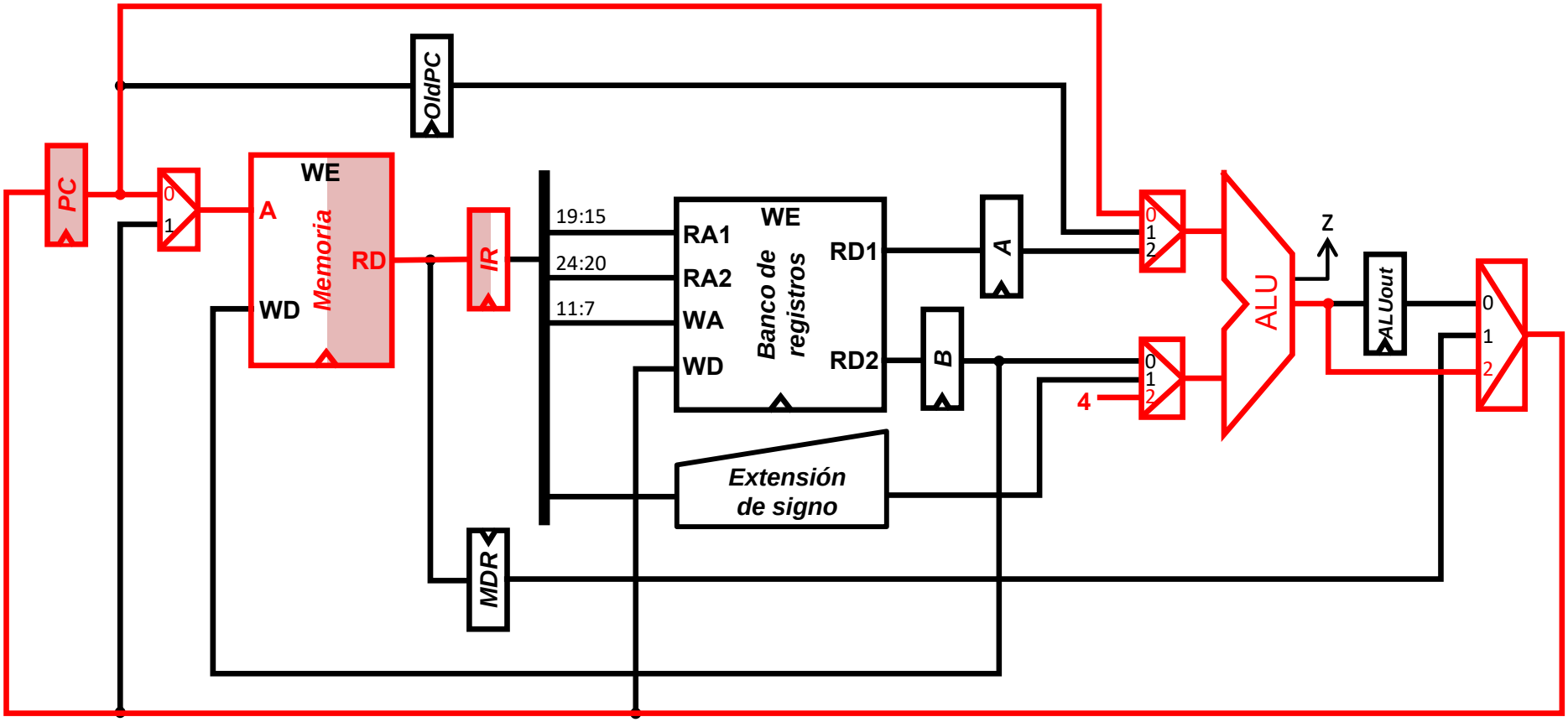
Diagrama ASM de la FSM principal: instrucción **sw**



Diseño del controlador

Instrucciones tipo **`addi`**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4



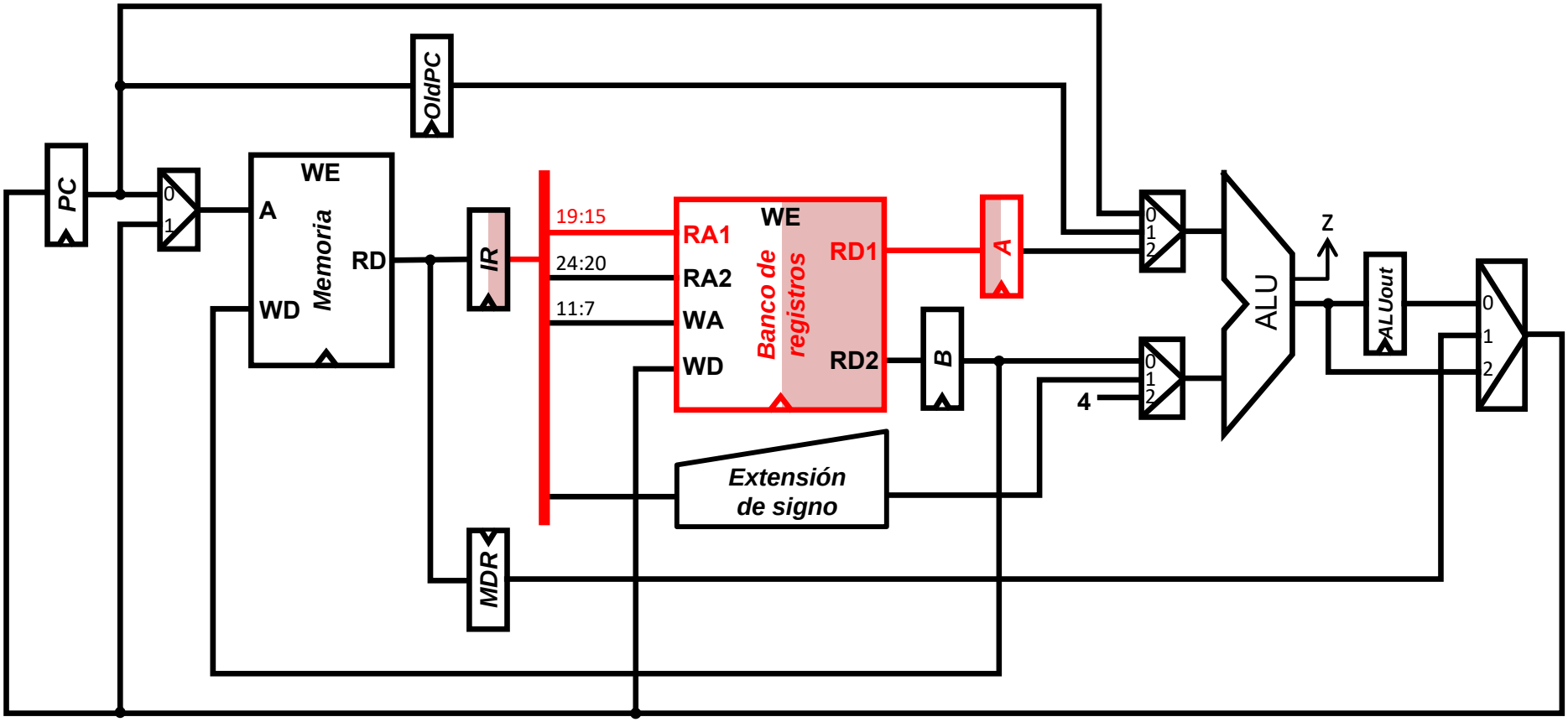
BR[rd] ← BR[rs1] *op* sExt(imm), PC ← PC+4

Diseño del controlador

Instrucciones tipo **`addi`**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4

2. A ← BR[rs1]



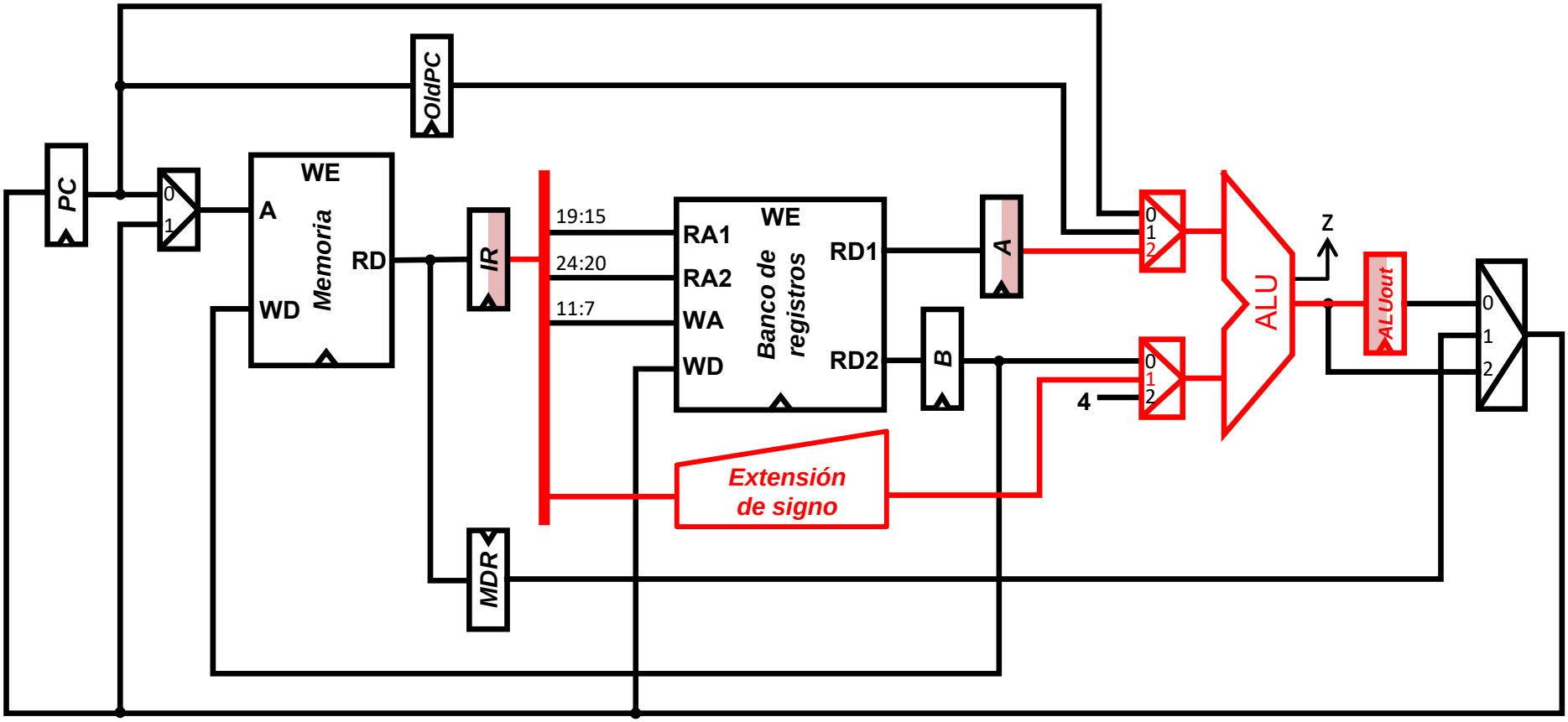
BR[rd] ← BR[rs1] *op* sExt(imm), PC ← PC+4

Diseño del controlador

Instrucciones tipo **`addi`**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4 **3.** ALUout ← A *op* sExt(imm)

2. A ← BR[rs1]



BR[rd] ← BR[rs1] *op* sExt(imm), PC ← PC+4

Diseño del controlador

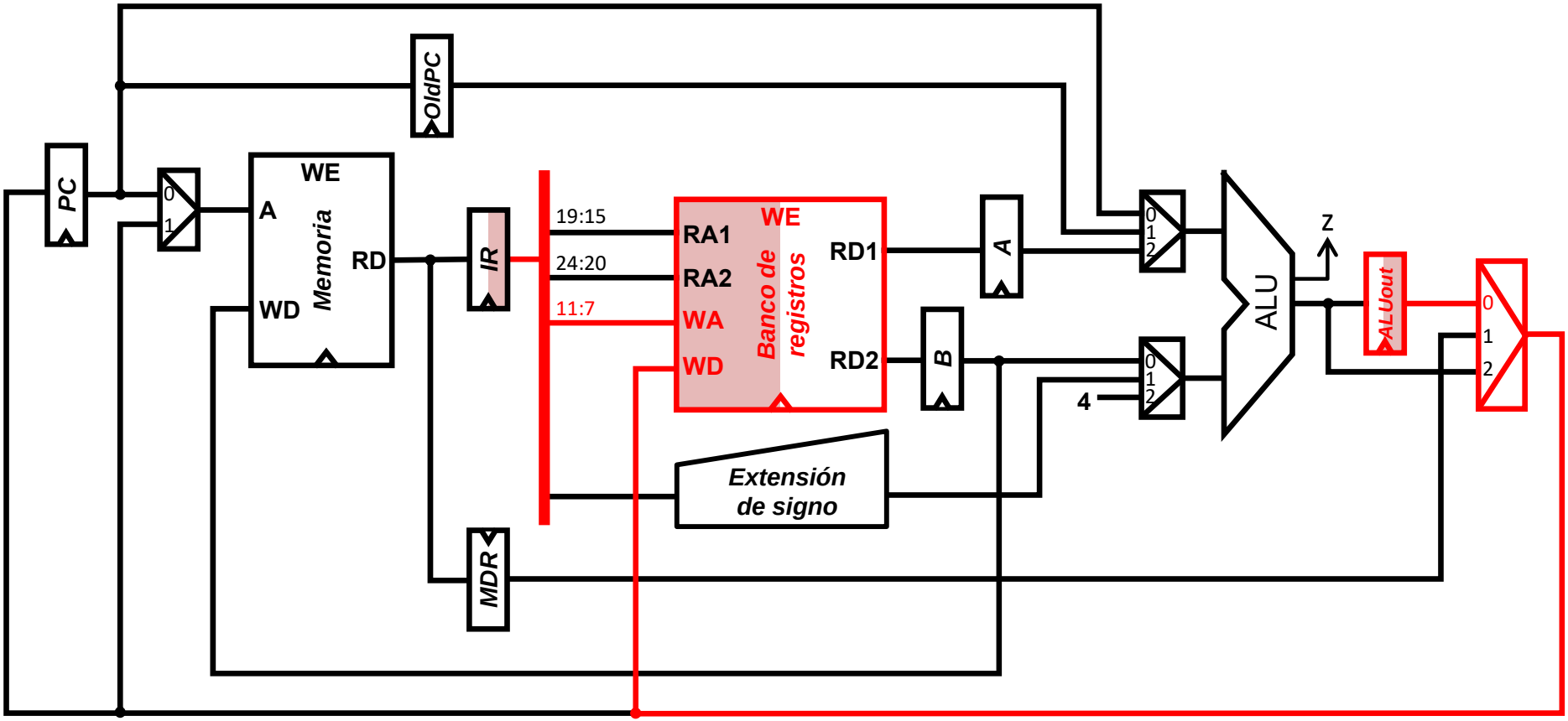
Instrucciones tipo **`addi`**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4

2. A ← BR[rs1]

3. ALUout ← A *op* sExt(imm)

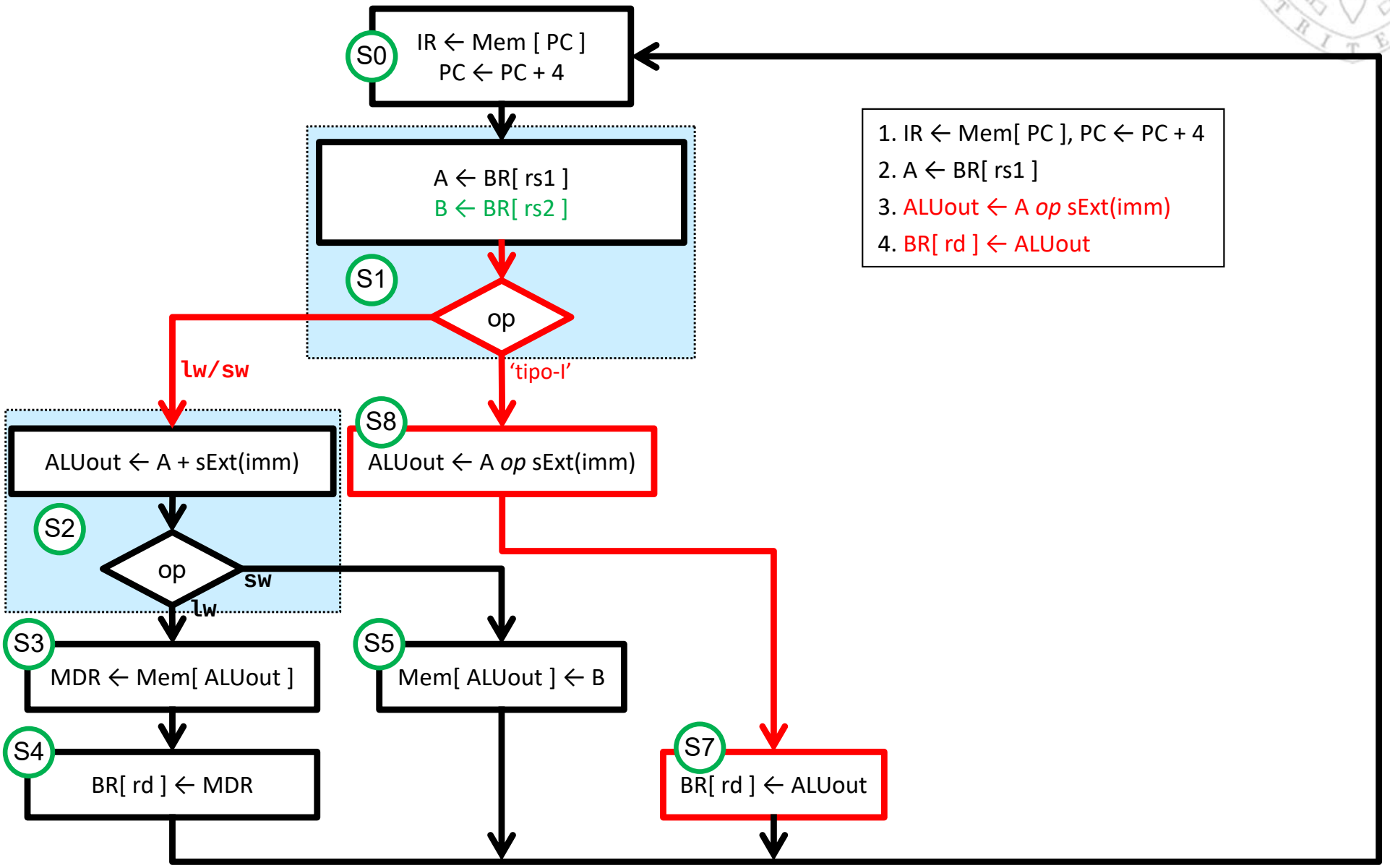
4. BR[rd] ← ALUout



BR[rd] ← BR[rs1] *op* sExt(imm), PC ← PC+4

Diseño del controlador

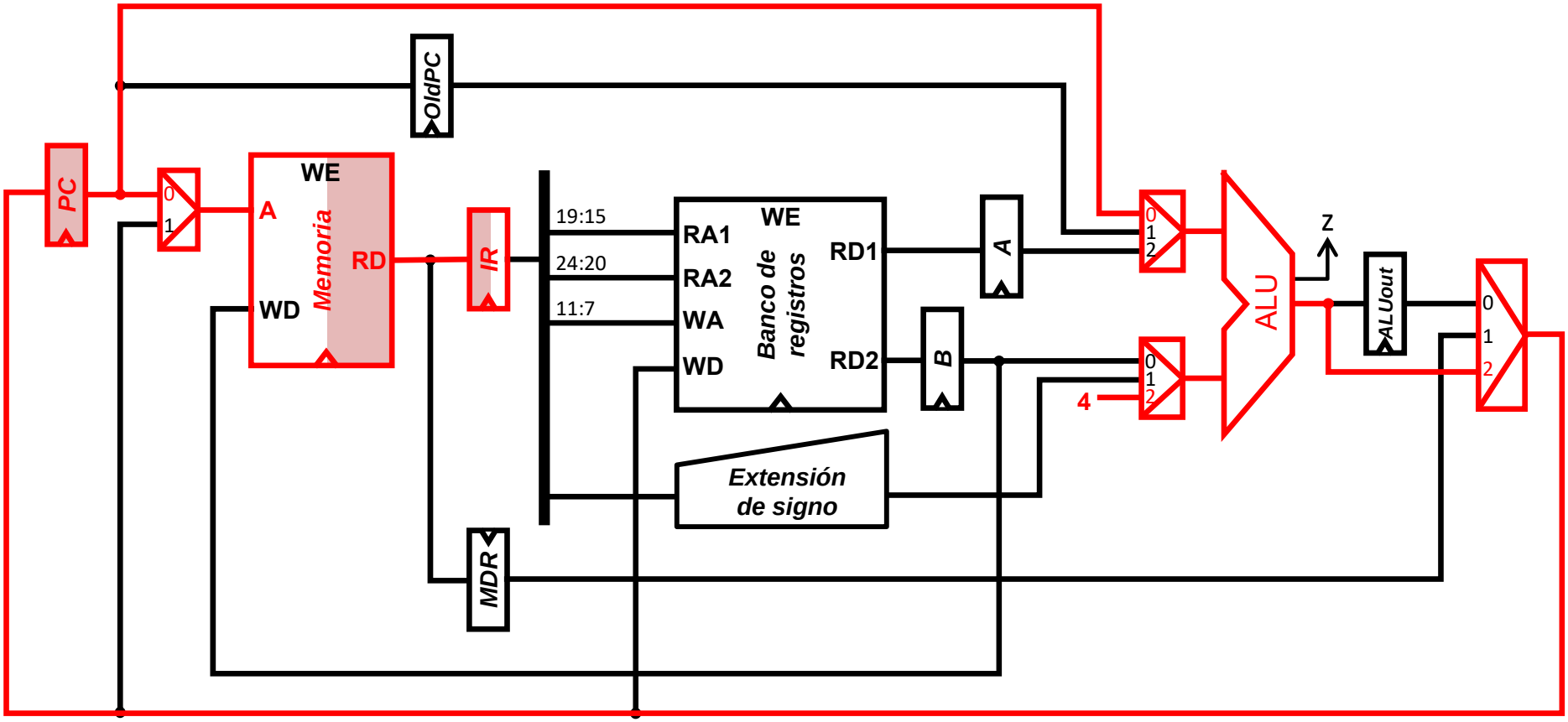
Diagrama ASM de la FSM principal: intruc. tipo **addi**



Diseño del controlador

Instrucciones tipo **add**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4



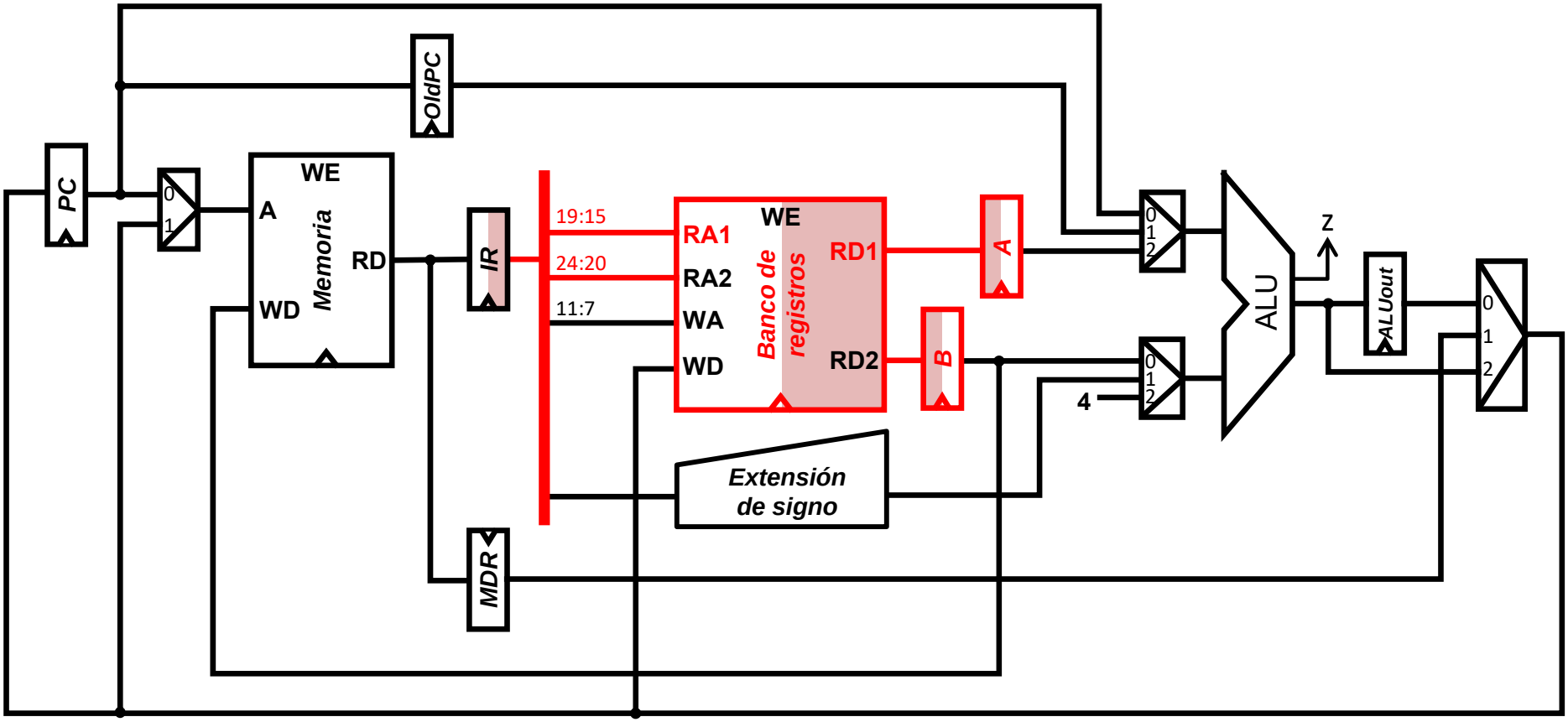
BR[rd] ← BR[rs1] *op* BR[rs2], PC ← PC+4

Diseño del controlador

Instrucciones tipo **add**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4

2. A ← BR[rs1], B ← BR[rs2]



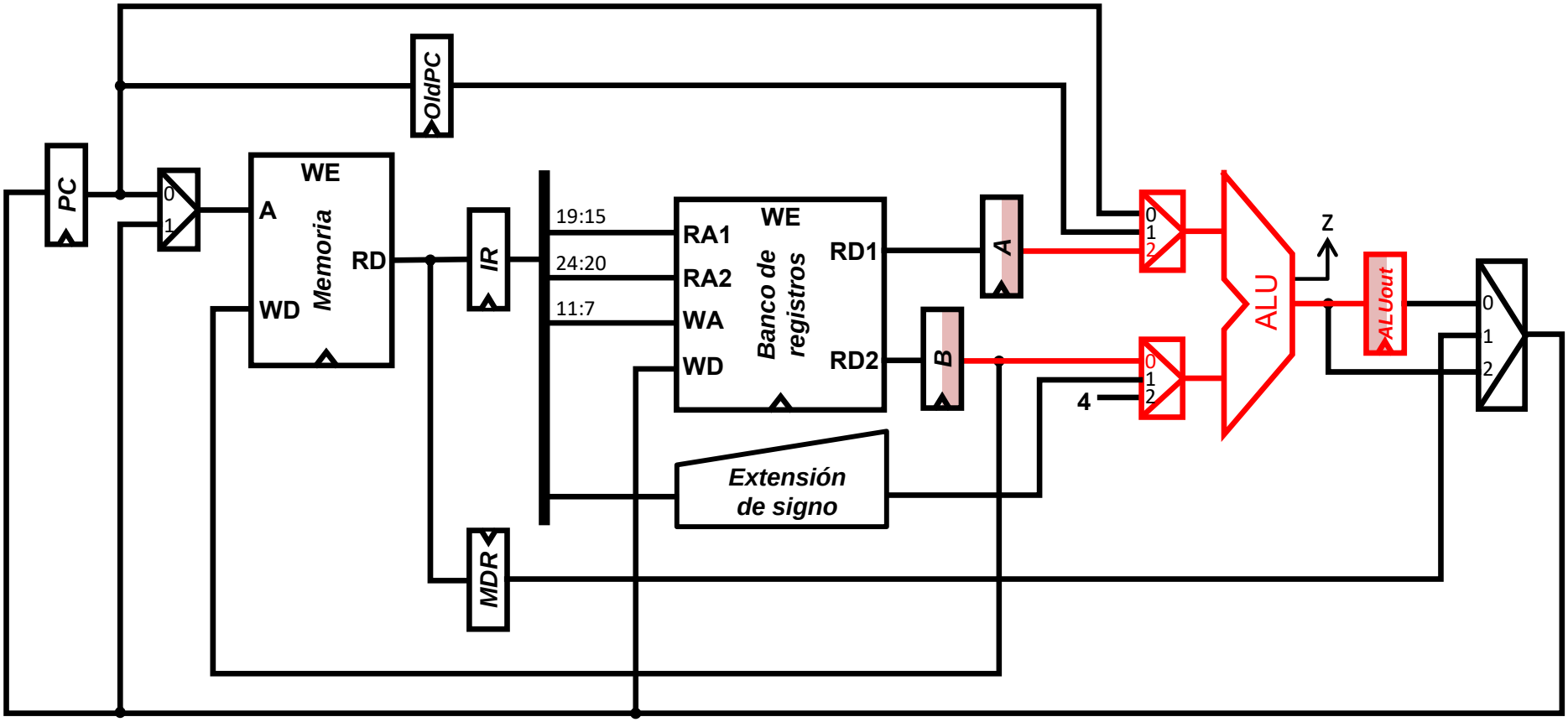
BR[rd] ← BR[rs1] *op* BR[rs2], PC ← PC+4

Diseño del controlador

Instrucciones tipo **add**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4 **3.** ALUout ← A *op* B

2. A ← BR[rs1], B ← BR[rs2]



BR[rd] ← BR[rs1] *op* BR[rs2], PC ← PC+4

Diseño del controlador

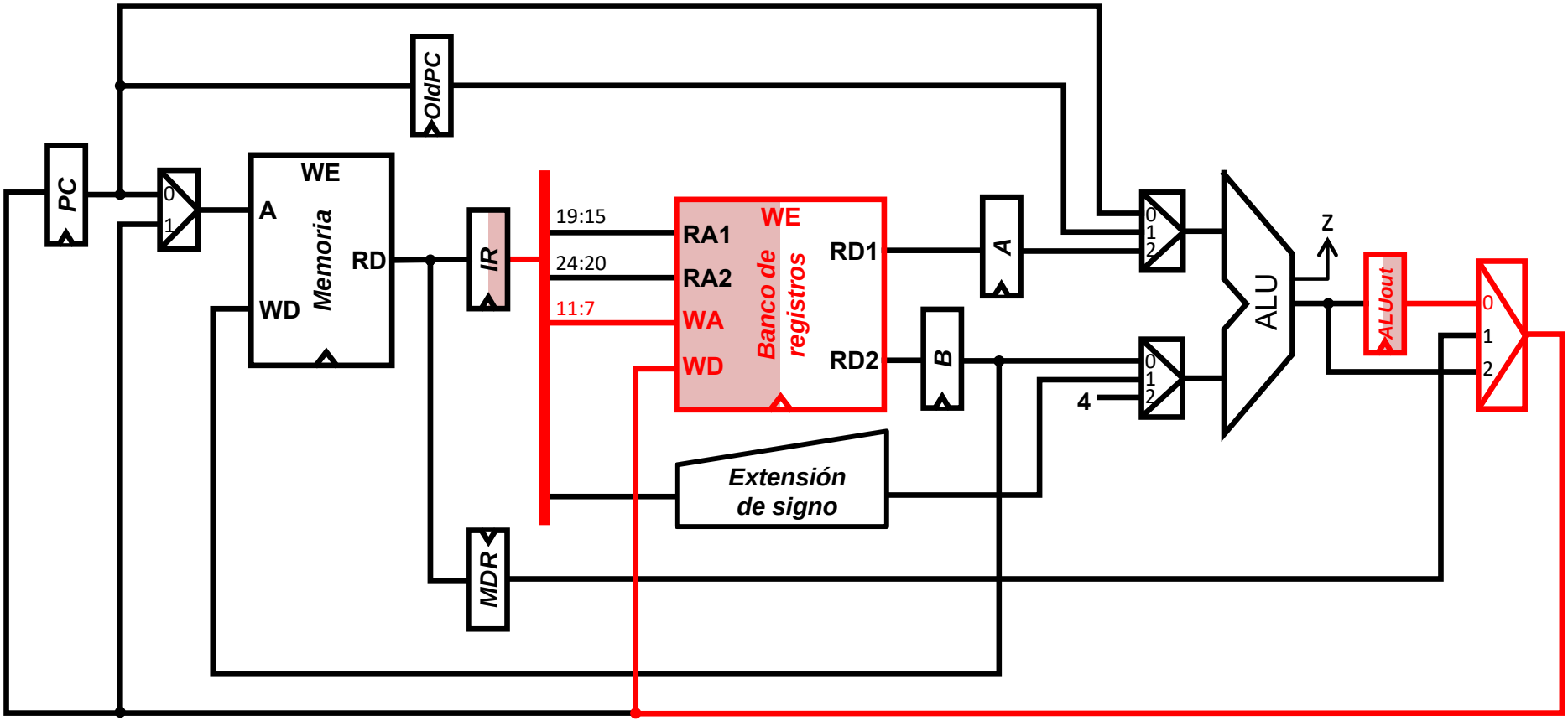
Instrucciones tipo **add**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4

2. A ← BR[rs1], B ← BR[rs2]

3. ALUout ← A *op* B

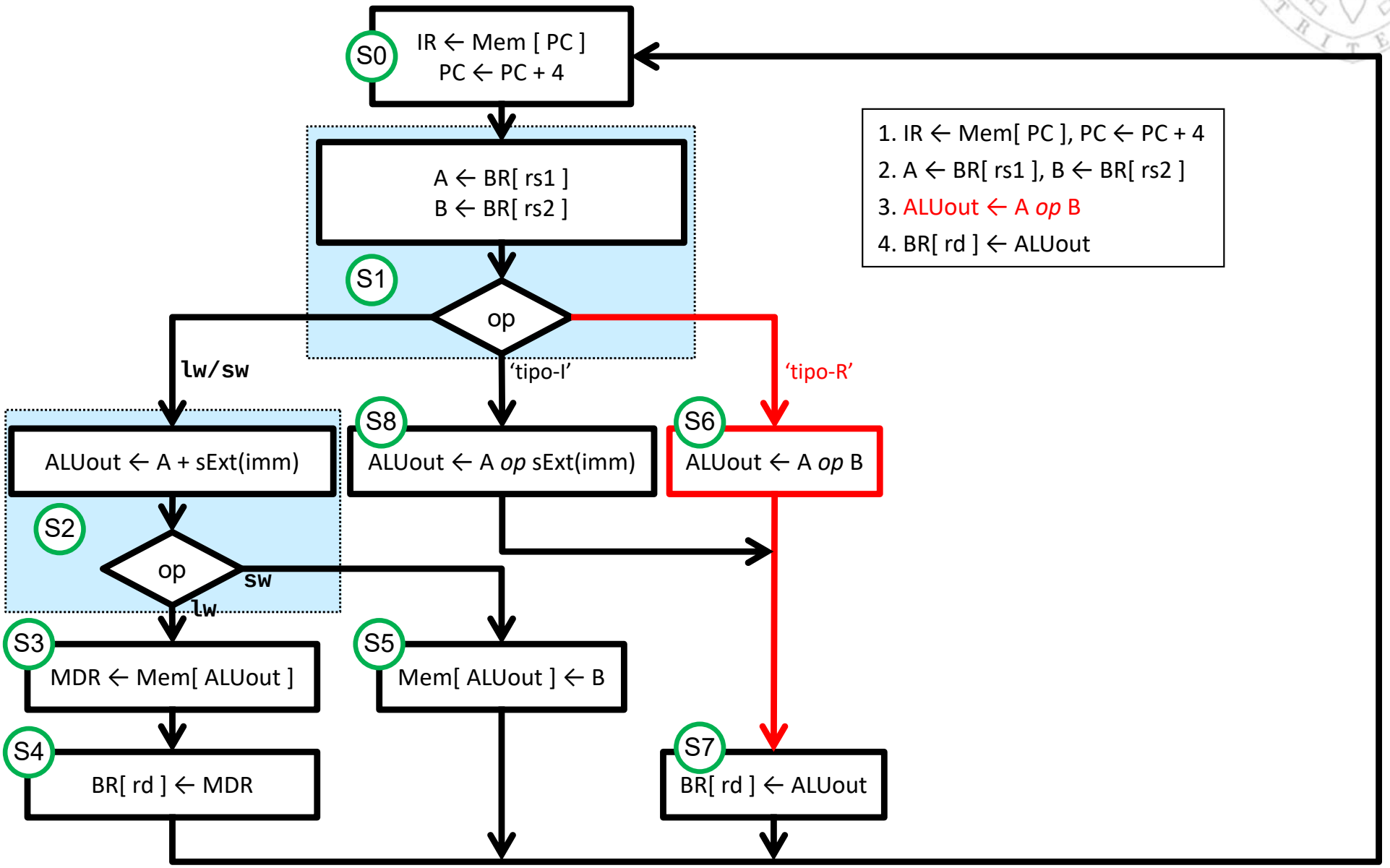
4. BR[rd] ← ALUout



BR[rd] ← BR[rs1] *op* BR[rs2], PC ← PC+4

Diseño del controlador

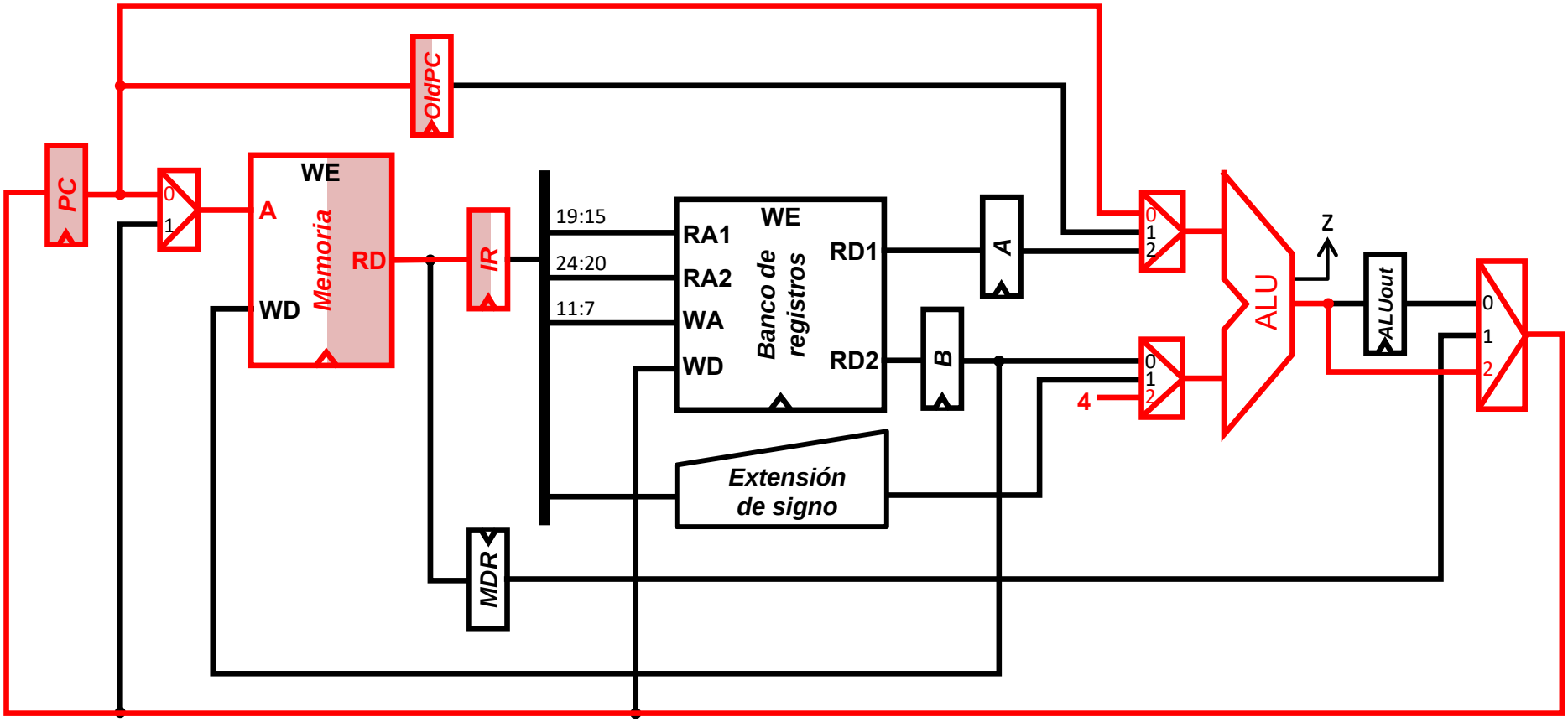
Diagrama ASM de la FSM principal: intruc. tipo **add**



Diseño del controlador

Instrucción **beq**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4, OldPC ← PC



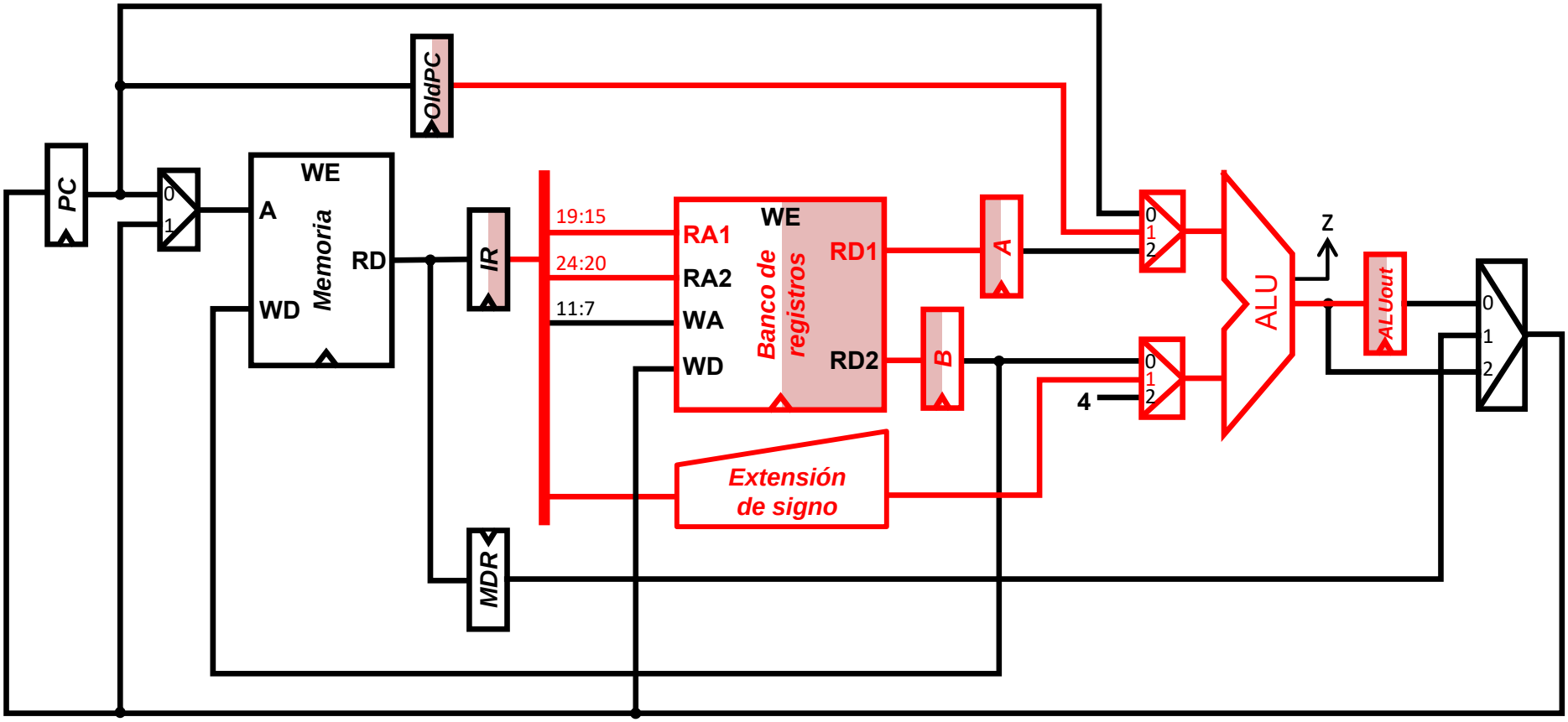
PC ← *if* (BR[rs1] = BR[rs2]) *then* (PC + sExt(imm)) *else* (PC+4)

Diseño del controlador

Instrucción **beq**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4, OldPC ← PC

2. A ← BR(rs1), B ← BR(rs2), ALUout ← oldPC + sExt(imm)



PC ← *if* (BR[rs1] = BR[rs2]) *then* (PC + sExt(imm)) *else* (PC+4)

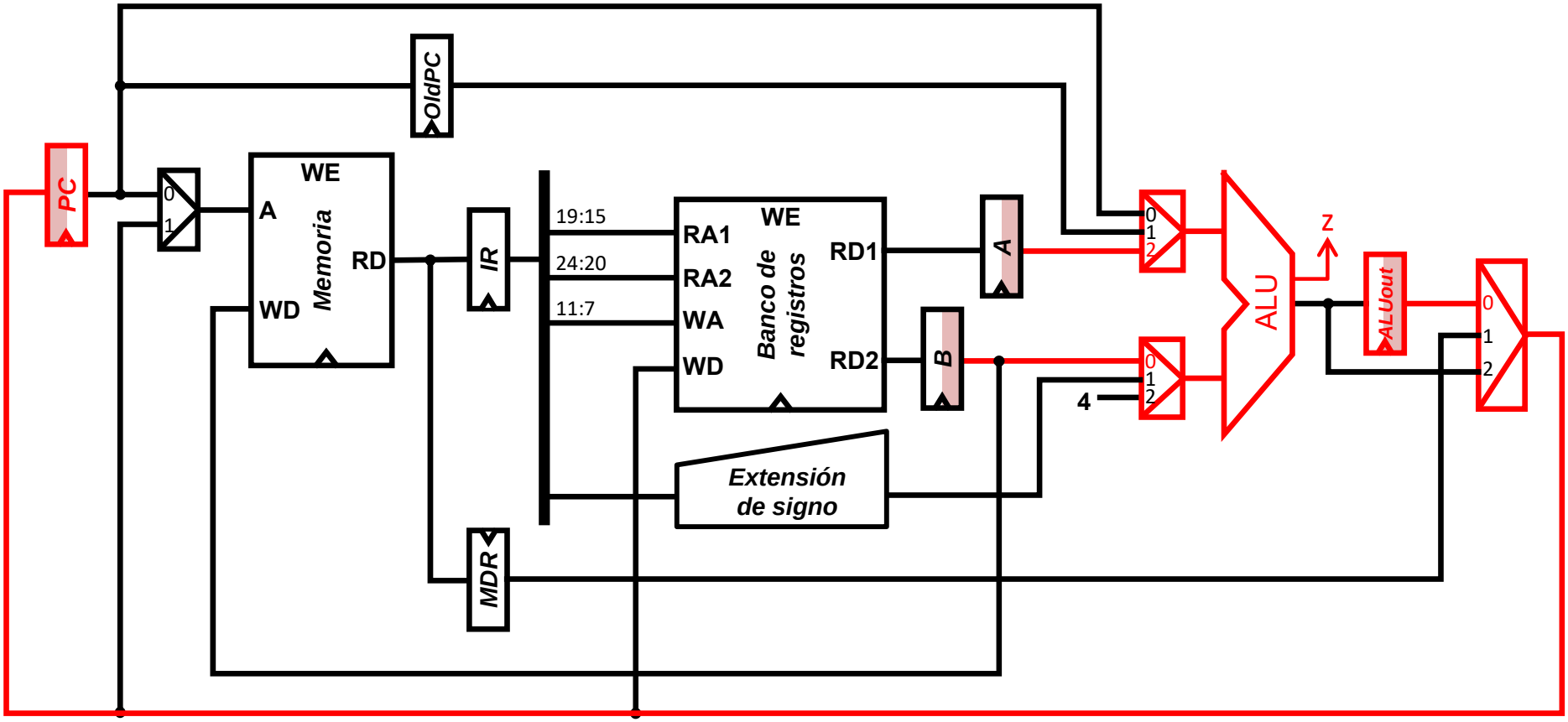
Diseño del controlador

Instrucción **beq**: transferencias entre registros

1. IR ← Mem[PC], PC ← PC + 4, OldPC ← PC

2. A ← BR(rs1), B ← BR(rs2), ALUout ← oldPC + sExt(imm)

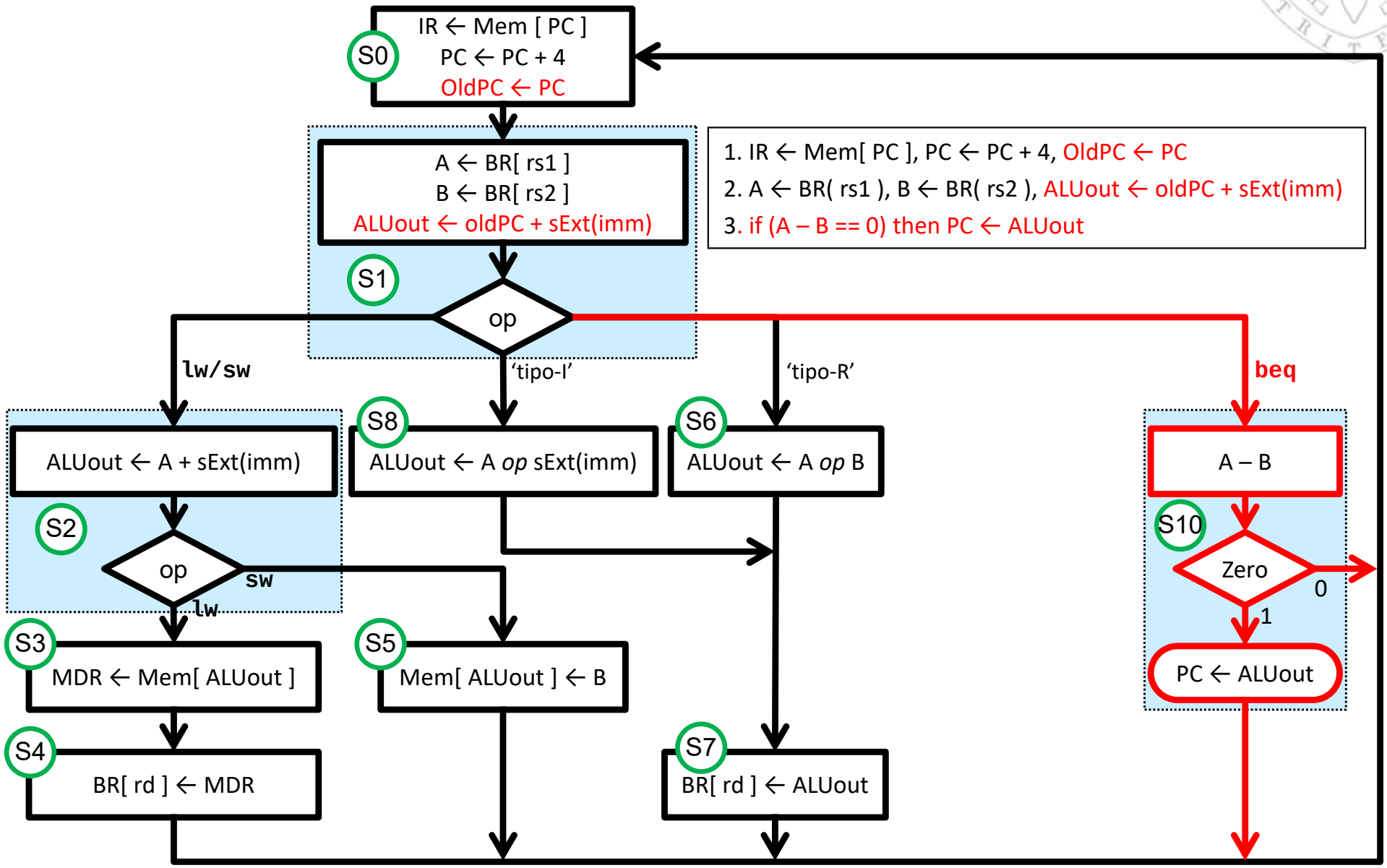
3. if (A – B == 0) then PC ← ALUout



PC ← *if* (BR[rs1] = BR[rs2]) *then* (PC + sExt(imm)) *else* (PC+4)

Diseño del controlador

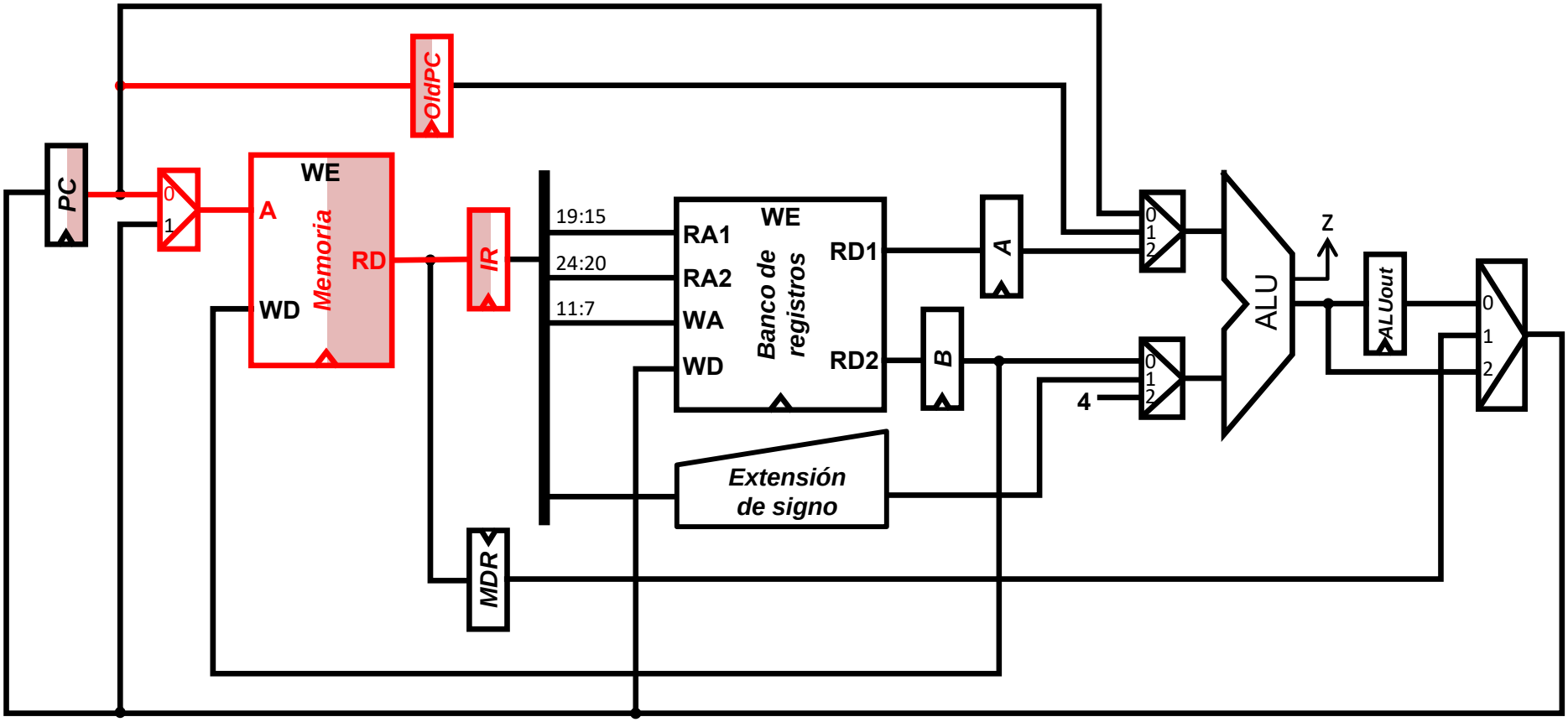
Diagrama ASM de la FSM principal: instrucción **beq**



Diseño del controlador

Instrucción `jal`: transferencias entre registros

1. IR ← Mem[PC], OldPC ← PC



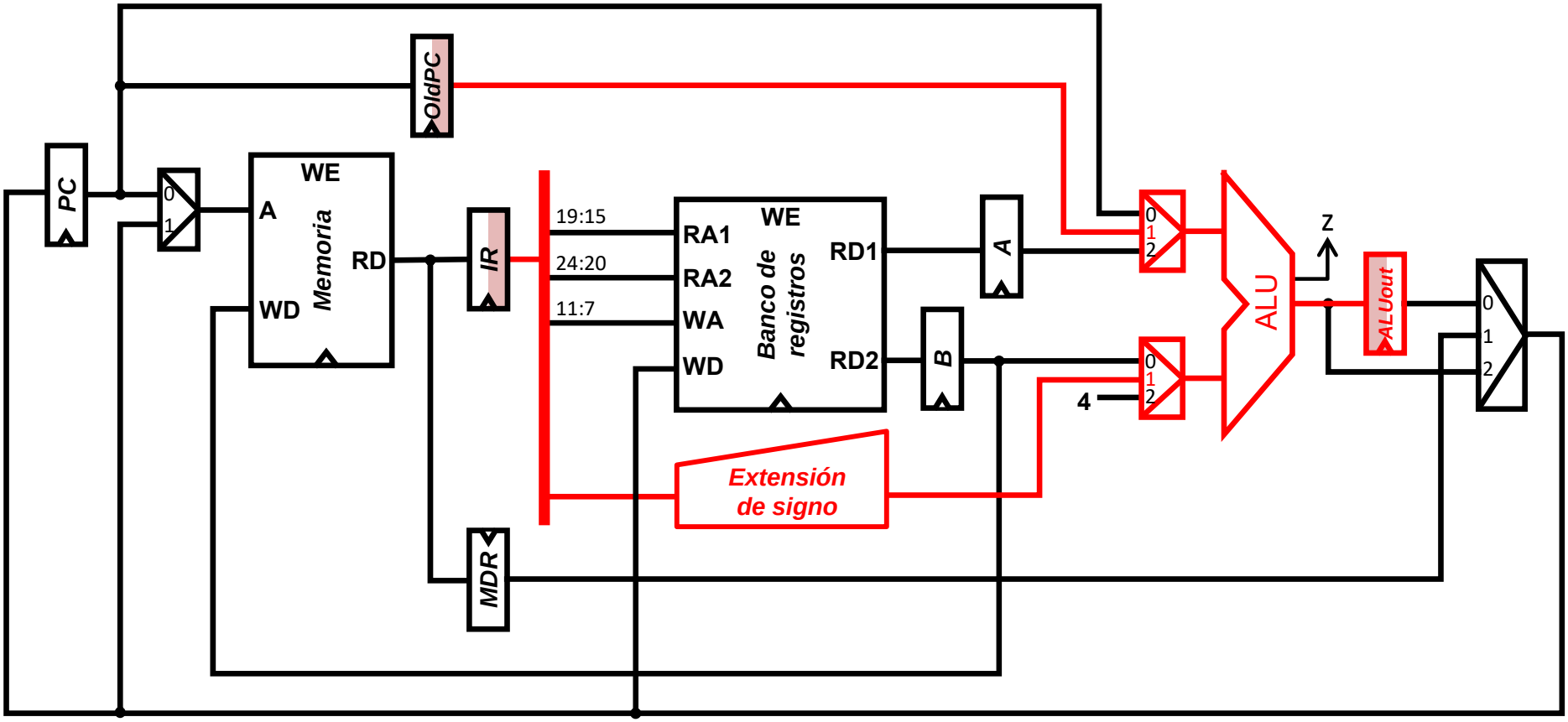
PC ← PC + sExt(imm), BR[rd] ← PC+4

Diseño del controlador

Instrucción `jal`: transferencias entre registros

1. IR ← Mem[PC], OldPC ← PC

2. ALUout ← oldPC + sExt(imm)



PC ← PC + sExt(imm), BR[rd] ← PC+4

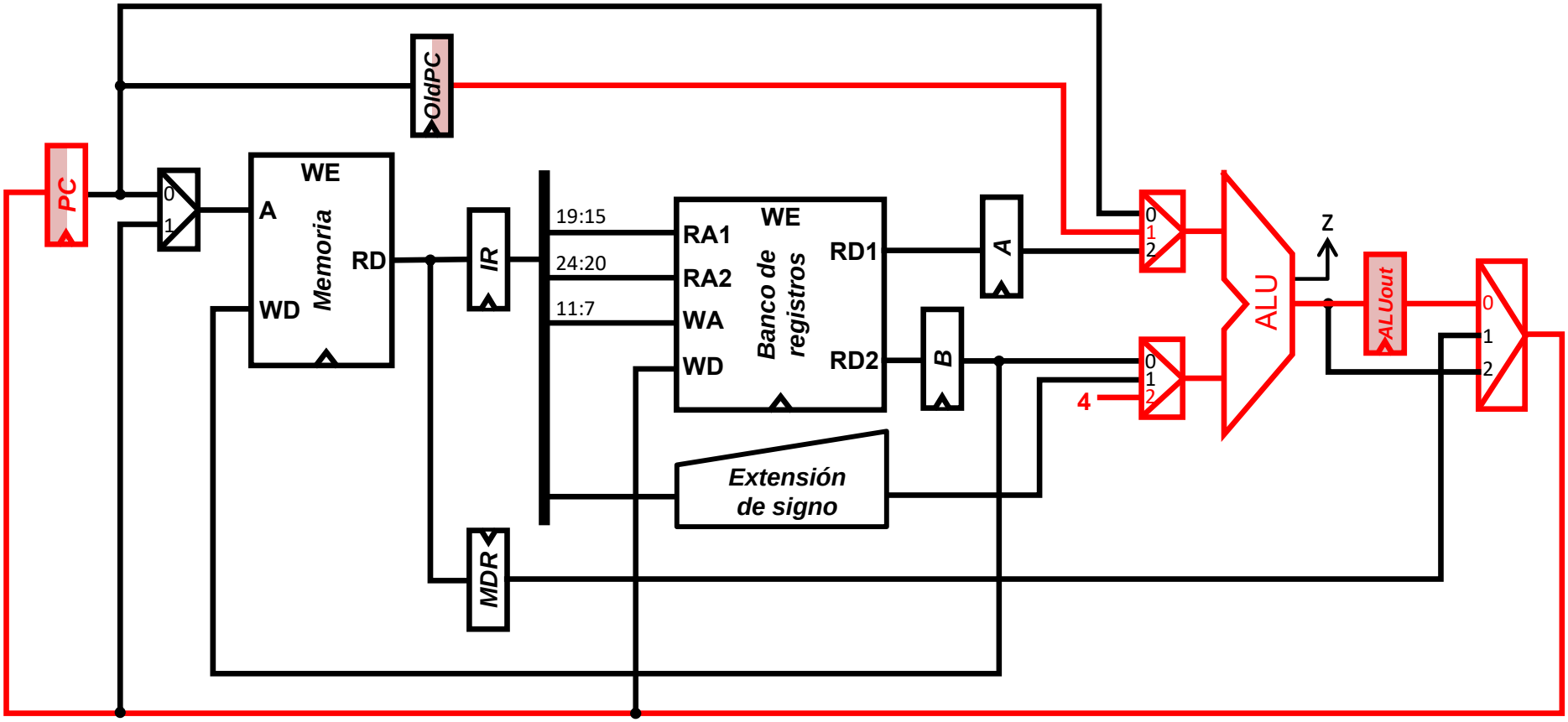
Diseño del controlador

Instrucción `jal`: transferencias entre registros

1. IR ← Mem[PC], OldPC ← PC

2. ALUout ← oldPC + sExt(imm)

3. PC ← ALUout, ALUout ← OldPC + 4



PC ← PC + sExt(imm), BR[rd] ← PC+4

Diseño del controlador

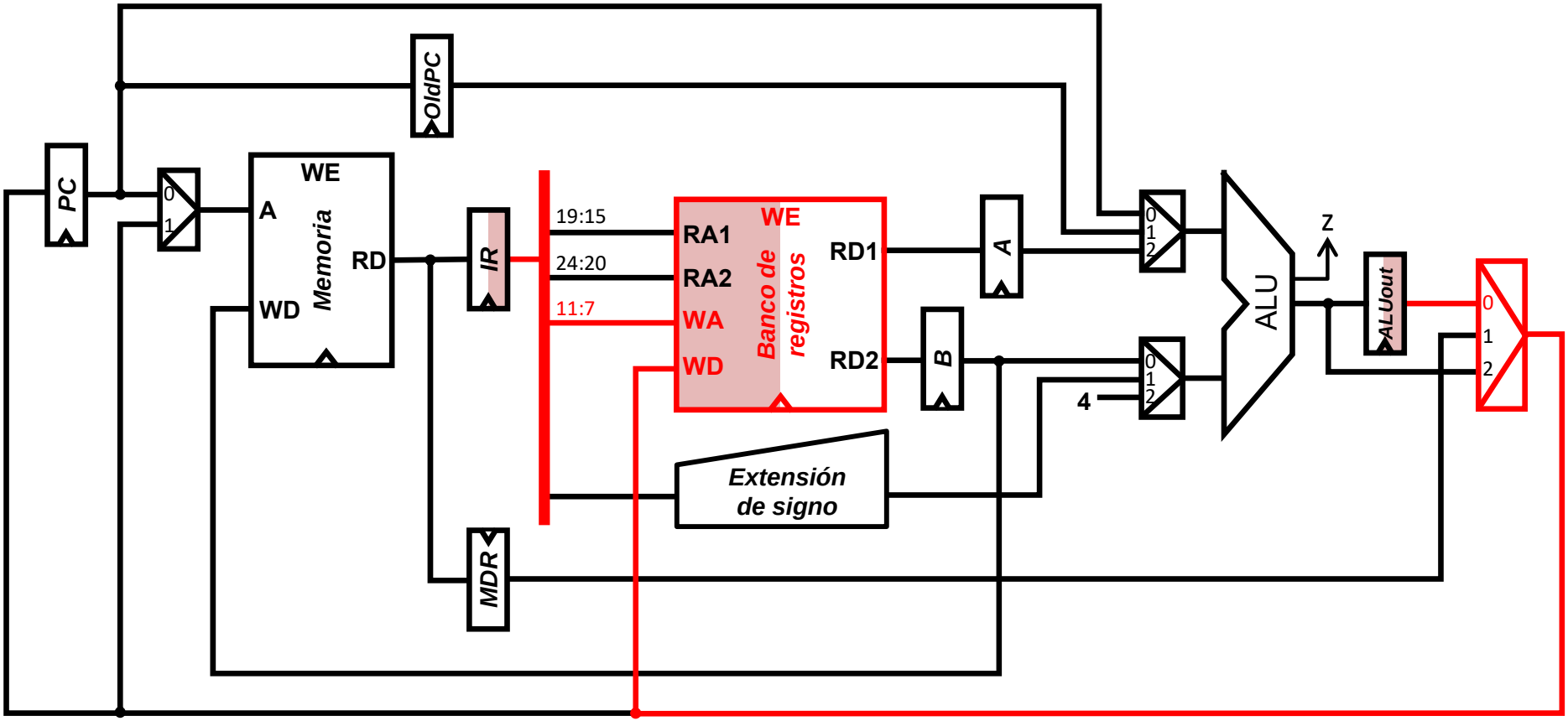
Instrucción `jal`: transferencias entre registros

1. IR ← Mem[PC], OldPC ← PC

2. ALUout ← oldPC + sExt(imm)

3. PC ← ALUout, ALUout ← OldPC + 4

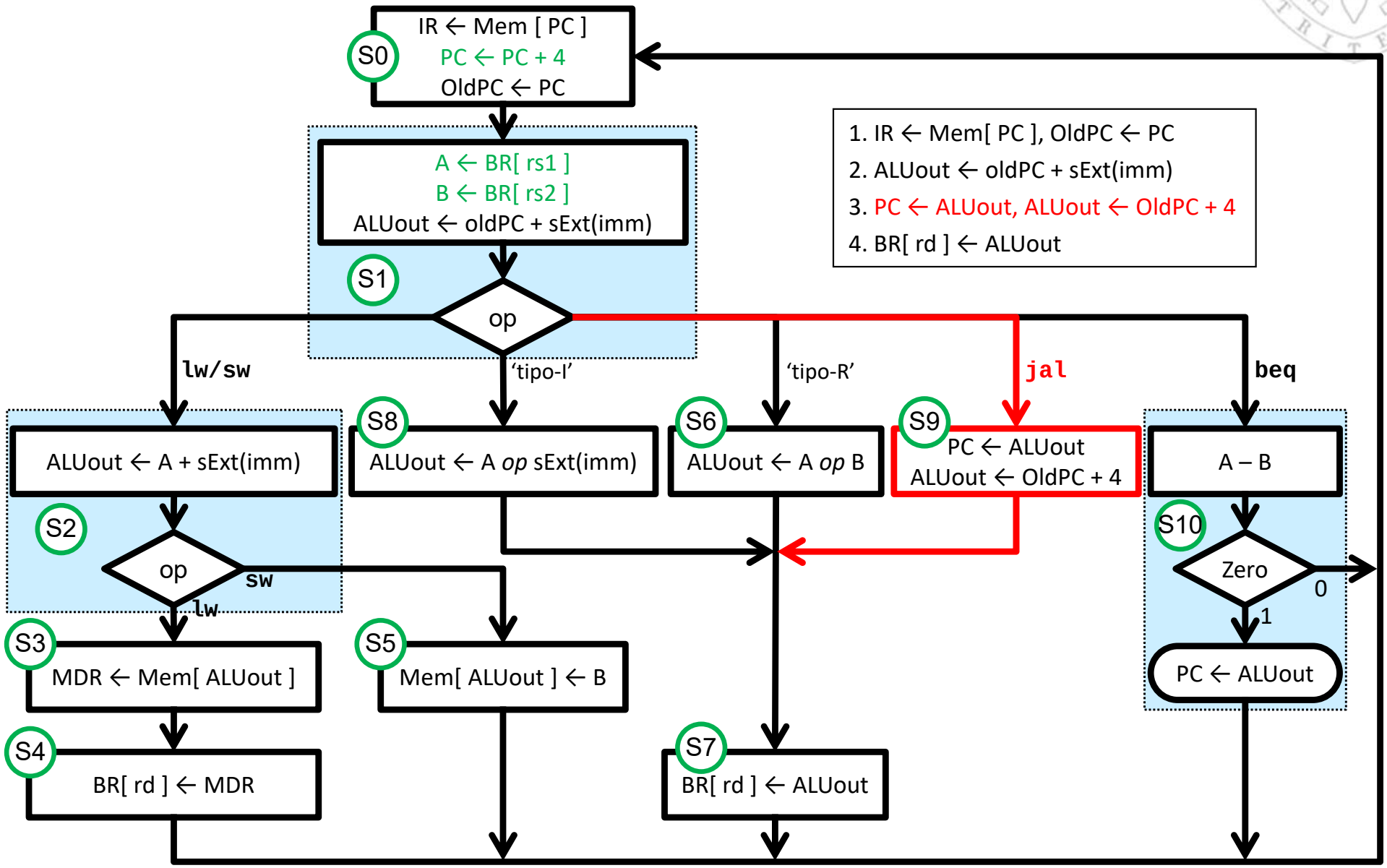
4. BR[rd] ← ALUout



PC ← PC + sExt(imm), BR[rd] ← PC+4

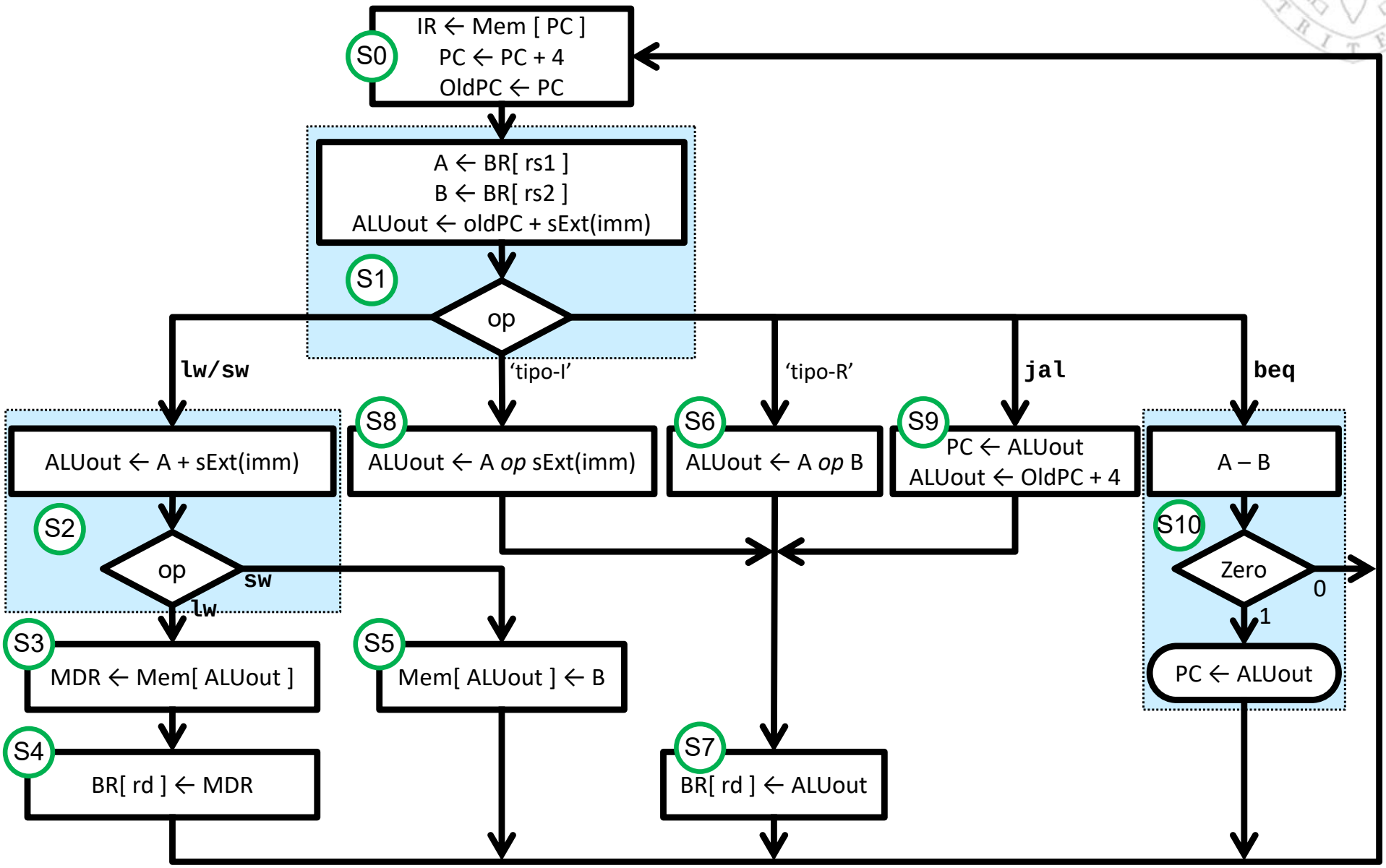
Diseño del controlador

Diagrama ASM de la FSM principal: instrucción `jal`



Diseño del controlador

diagrama ASM completo de la FSM principal



Diseño del controlador

ciclo de instrucción (i)

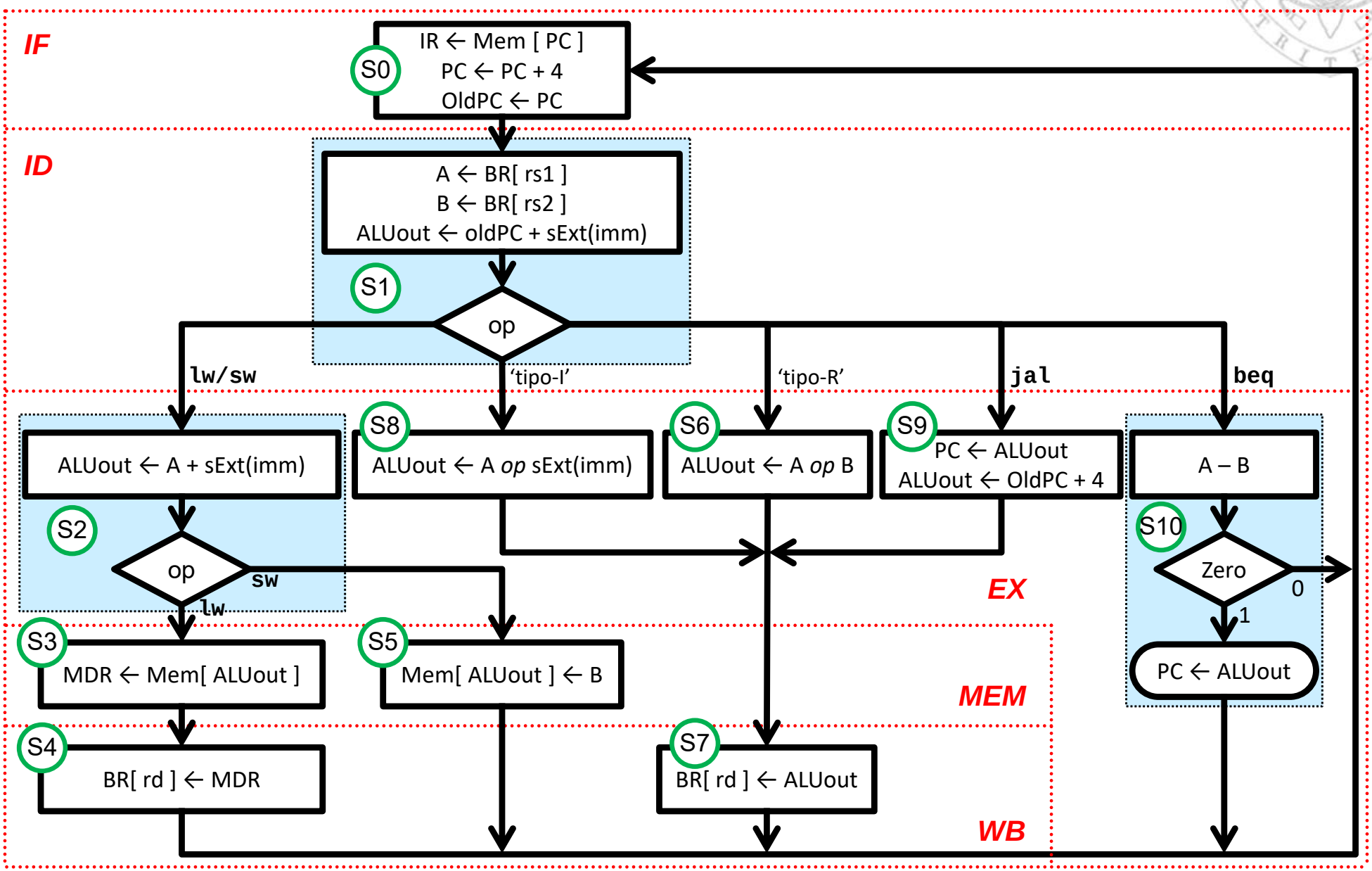
- Conceptualmente la ejecución de instrucciones se efectúa pasando cíclicamente por 5 etapas:
  - Búsqueda en memoria de la instrucción
  - Decodificación y obtención de operandos
  - Ejecución o cálculo de dirección
  - Acceso a memoria de datos
  - Escritura del resultado

  IF: Instruction Fetch
  ID: Instruction Decode
  EX: Execution
  MEM: Memory R/W
  WB: Write-Back
- Es lo que se conoce como ciclo de instrucción
  - En el procesador multiciclo cada fase se realiza en 1 ciclo de reloj.
  - Cada instrucción tarda un número distinto de ciclos, según las fases del ciclo de instrucción que deba realizar:

Instrucción	# de ciclos
`lw`	5
`sw` / `jal` / aritmético-logicas	4
`beq`	3

Diseño del controlador

ciclo de instrucción (ii)



Diseño del controlador

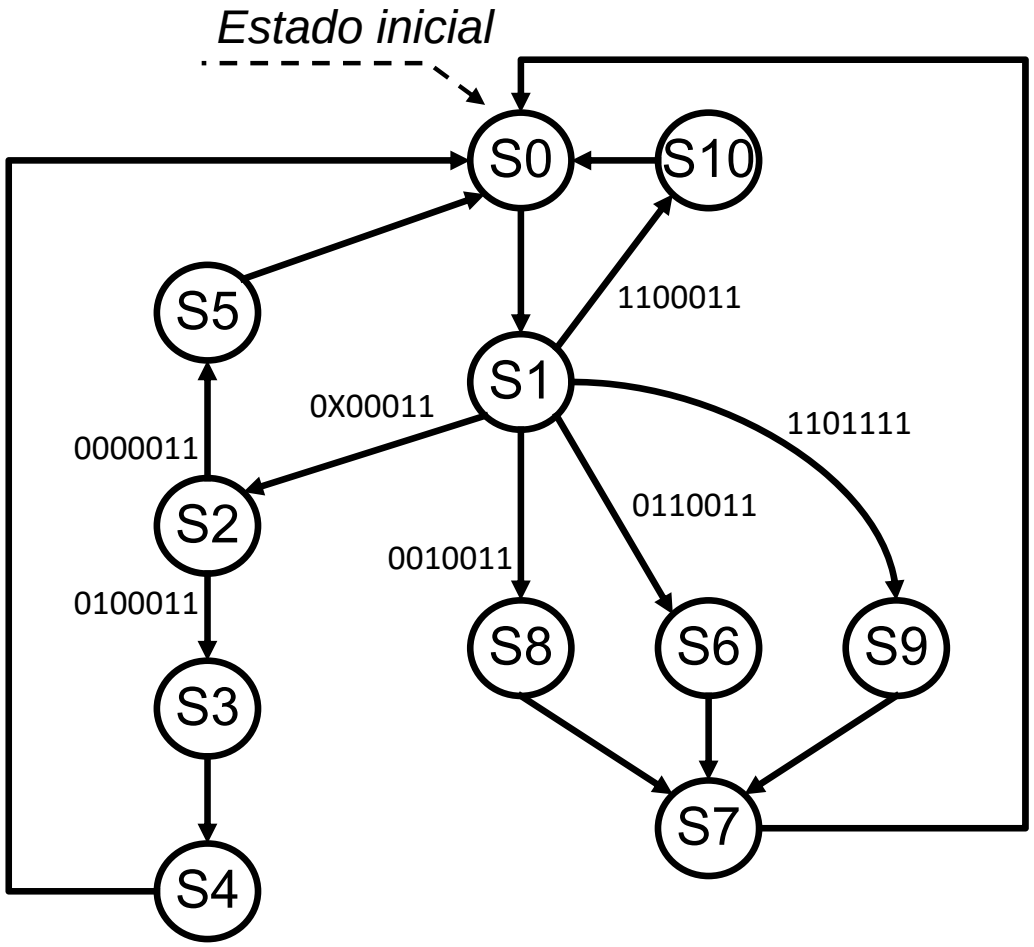
- Diagrama ASM = Diagrama de estados de una FSM.
  - Cada bloque ASM equivale a un estado de la FSM.
  - Las transiciones entre bloques equivalen a transiciones entre estados.
  - Cada transferencia entre registros de un bloque ASM se traduce a valores de señales de control en el estado correspondiente.

- Las transferencias entre registros realizadas en cada estado se traducen:
  - Poniendo a 1 las señales de carga de los registros destino.
  - Poniendo a 0 las señales de carga del resto de registros.
  - Poniendo al valor que corresponda las señales de selección de los MUX implicados en las transferencias.
  - Poniendo a *don't care* las señales de selección del resto de MUX.

- Una vez obtenido el diagrama de estados del controlador, se optimiza para reducir el coste de la lógica.

Diseño del controlador

FSM Principal: función de transición de estados

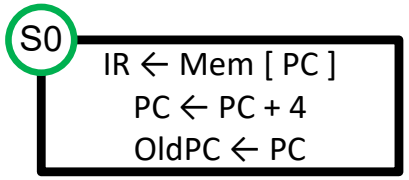


Función de transición de estados

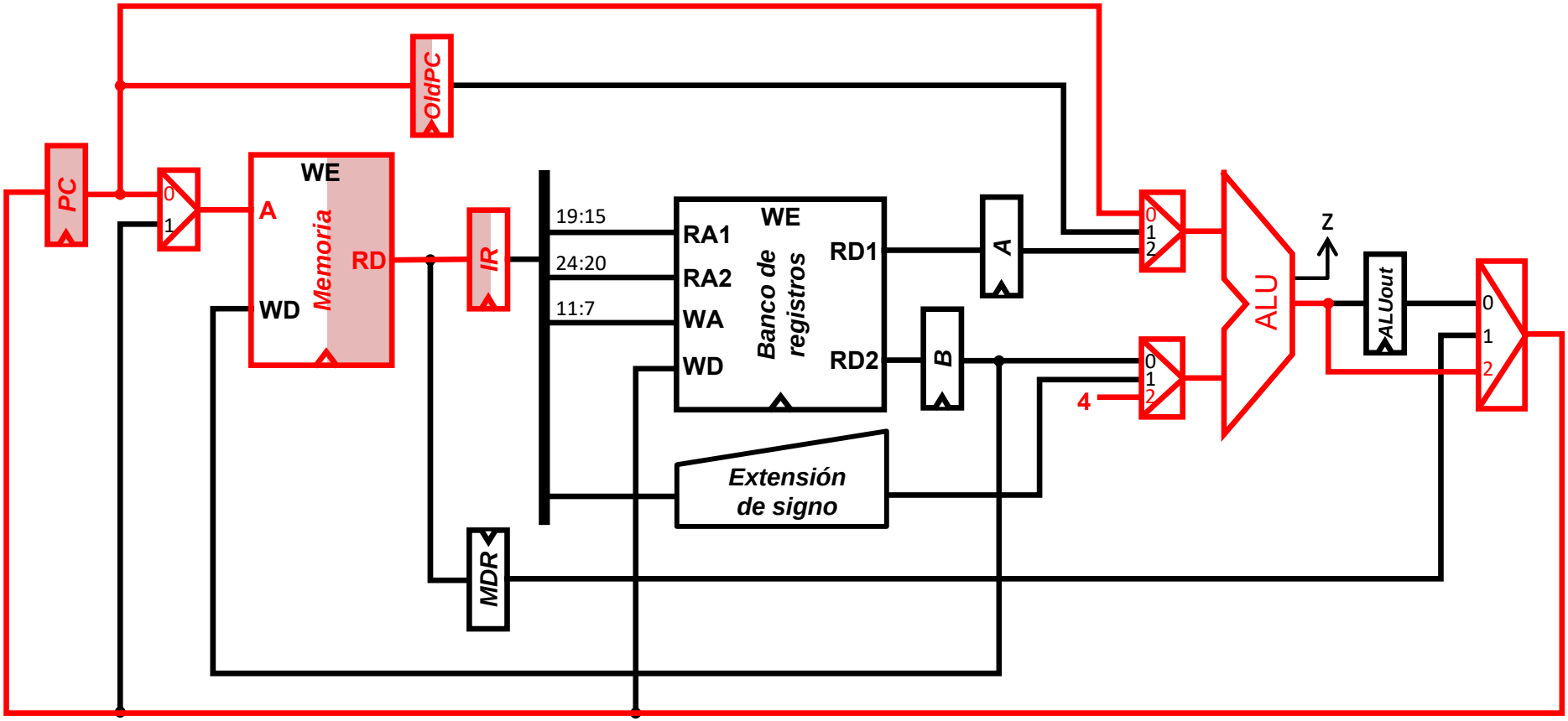
estado	op	estado'
S0	XXXXXXX	S1
S1	0X00011 (`lw/sw`)	S2
S1	0010011 (tipo-I)	S8
S1	0110011 (tipo-R)	S6
S1	1101111 (`jal`)	S9
S1	1100011 (`beq`)	S10
S2	0000011 (`lw`)	S3
S2	0100011 (`sw`)	S5
S3	XXXXXXX	S4
S4	XXXXXXX	S0
S5	XXXXXXX	S0
S6	XXXXXXX	S7
S7	XXXXXXX	S0
S8	XXXXXXX	S7
S9	XXXXXXX	S7
S10	XXXXXXX	S0

Diseño del controlador

FSM Principal: función de salida

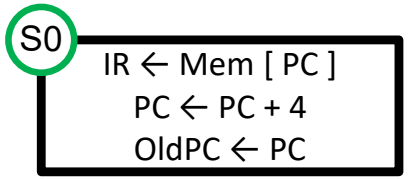


estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0															

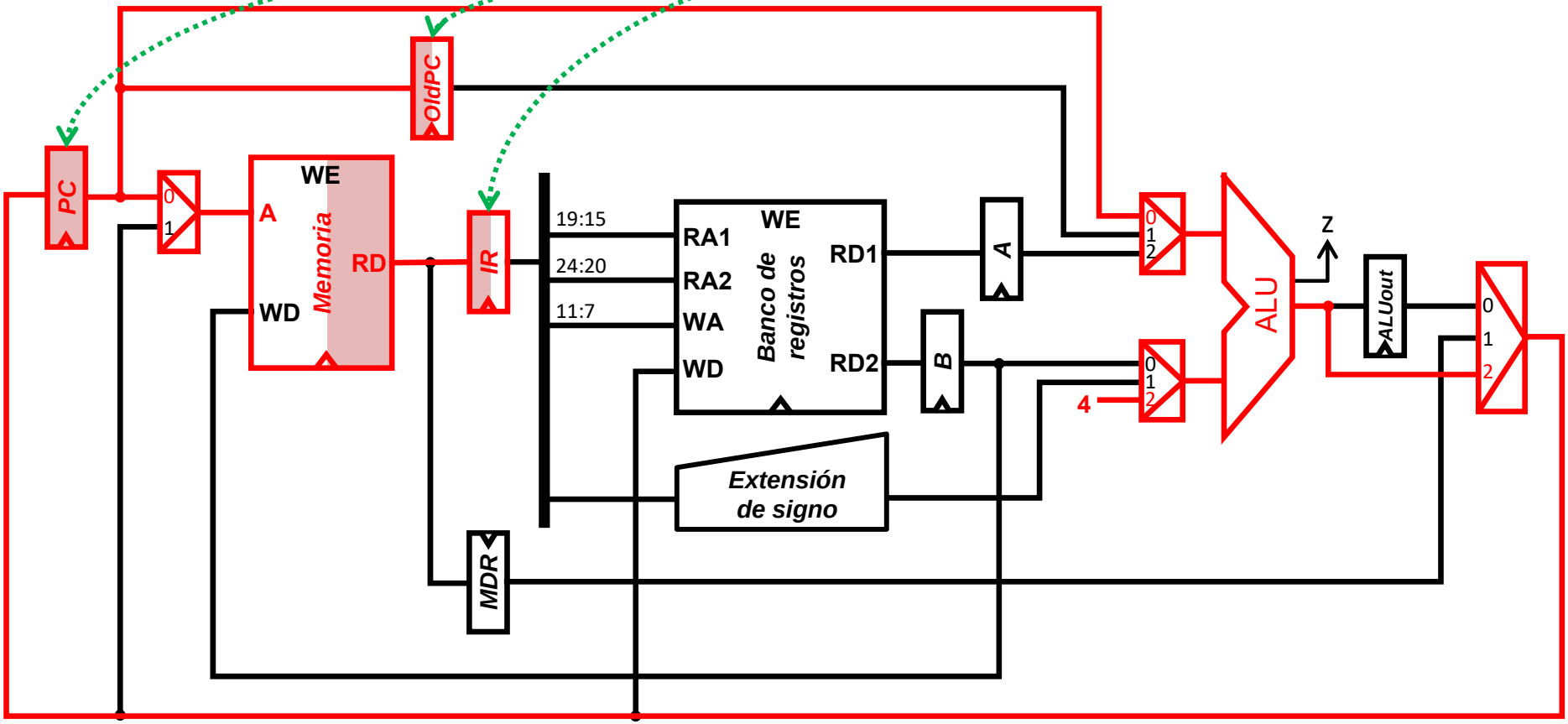


Diseño del controlador

FSM Principal: función de salida

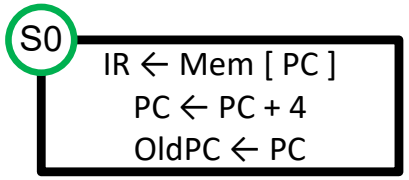


estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1			1	1									

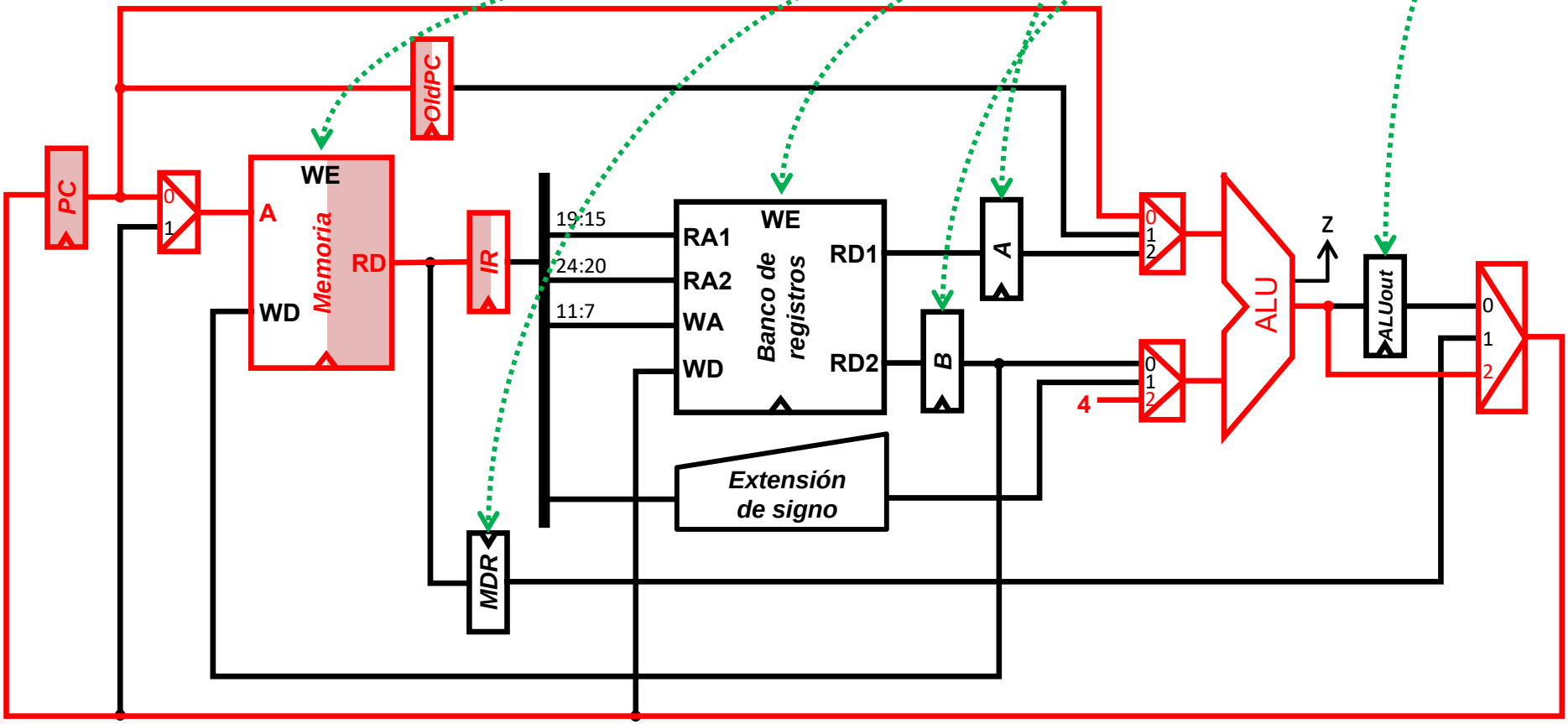


Diseño del controlador

FSM Principal: función de salida

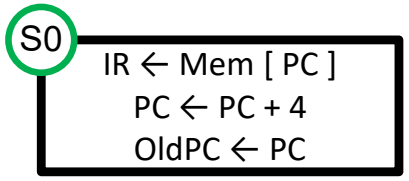


estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1		0	1	1	0	0	0	0				0	

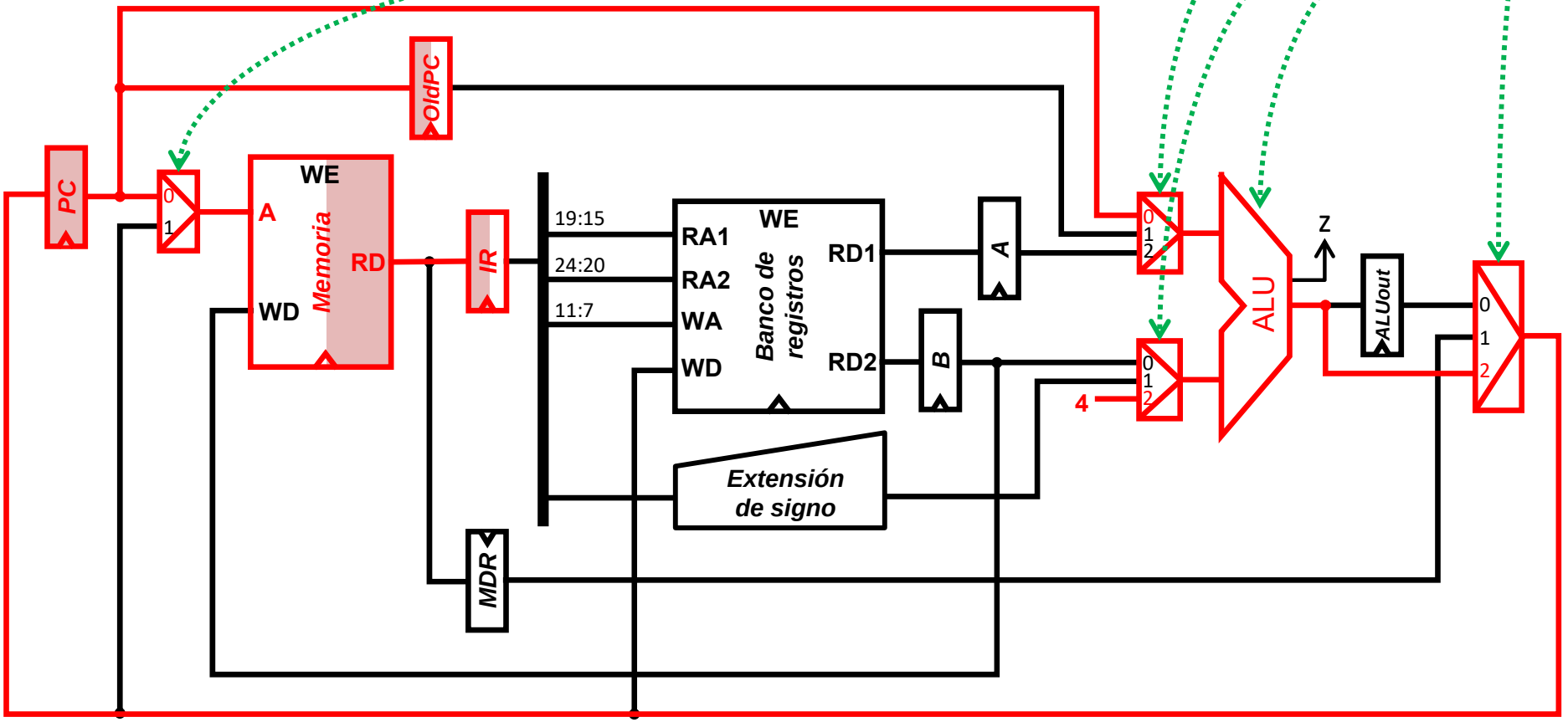


Diseño del controlador

FSM Principal: función de salida



estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10

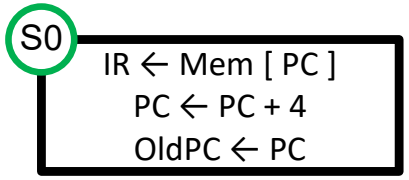


Diseño del controlador

FSM Principal: función de salida

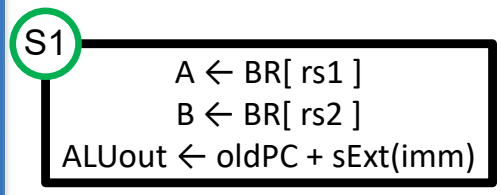
Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1															
S2															
S3															
S4															
S5															
S6															
S7															
S8															
S9															
S10															

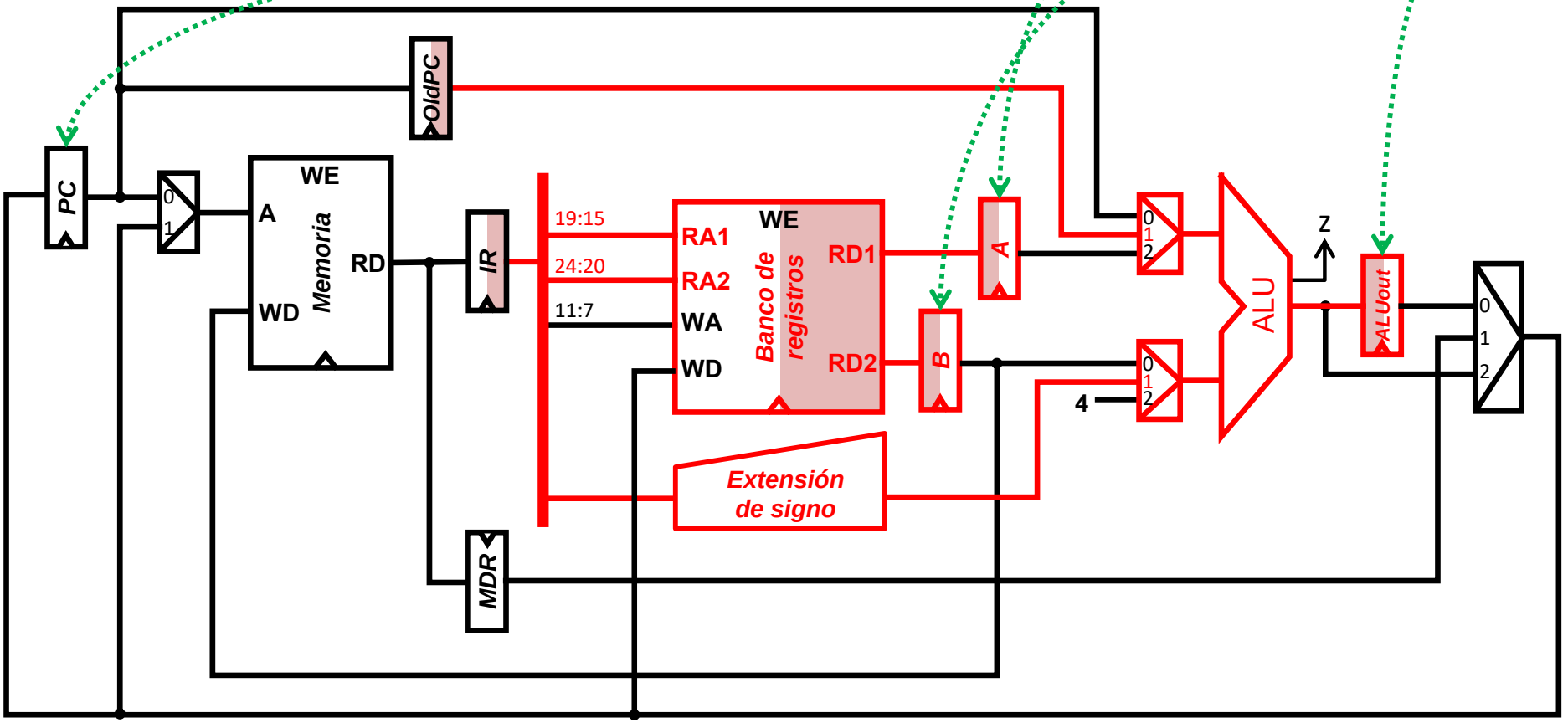


Diseño del controlador

FSM Principal: función de salida



estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S1	0	0							1	1				1	



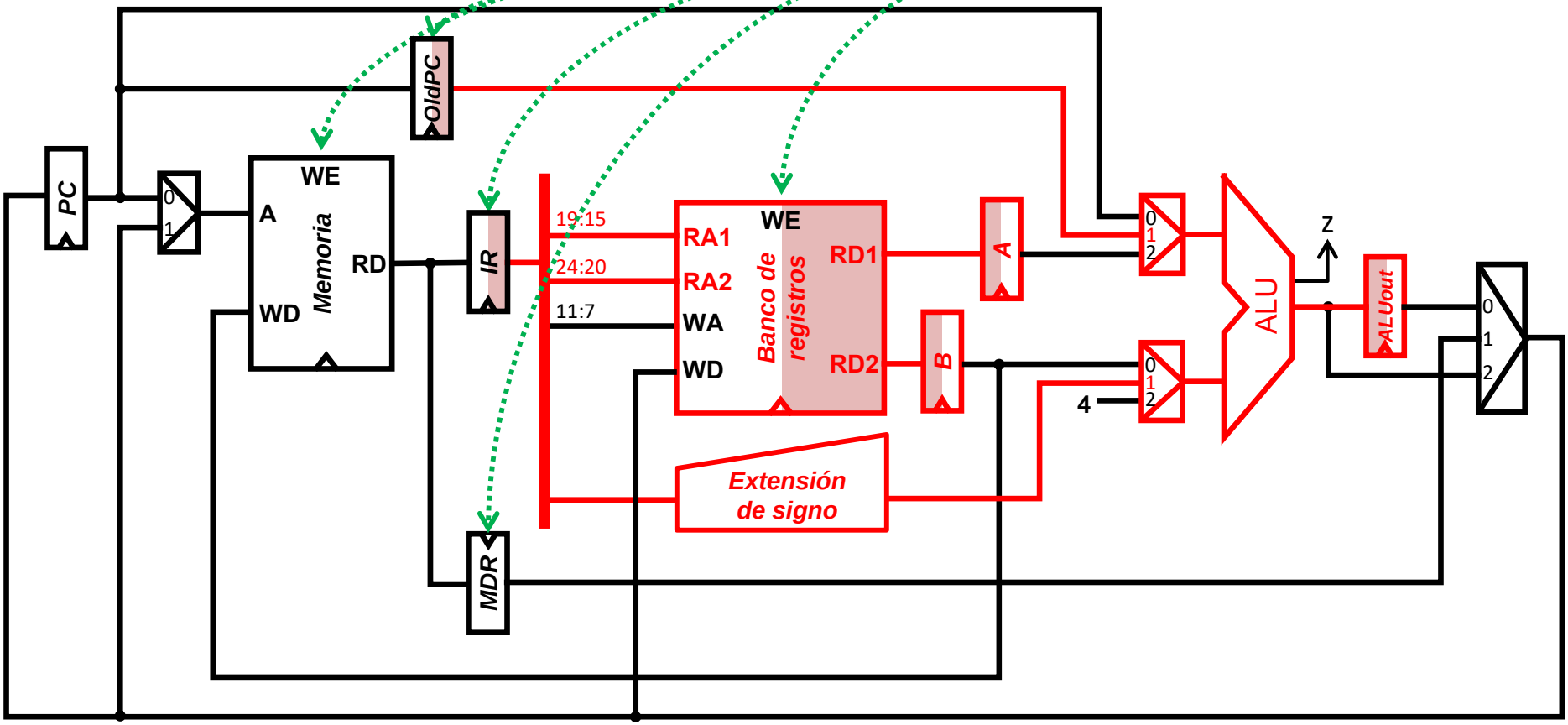
Diseño del controlador

FSM Principal: función de salida

S1

A ← BR[rs1]
B ← BR[rs2]
ALUout ← oldPC + sExt(imm)

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S1	0	0		0	0	0	0	0	1	1				1	



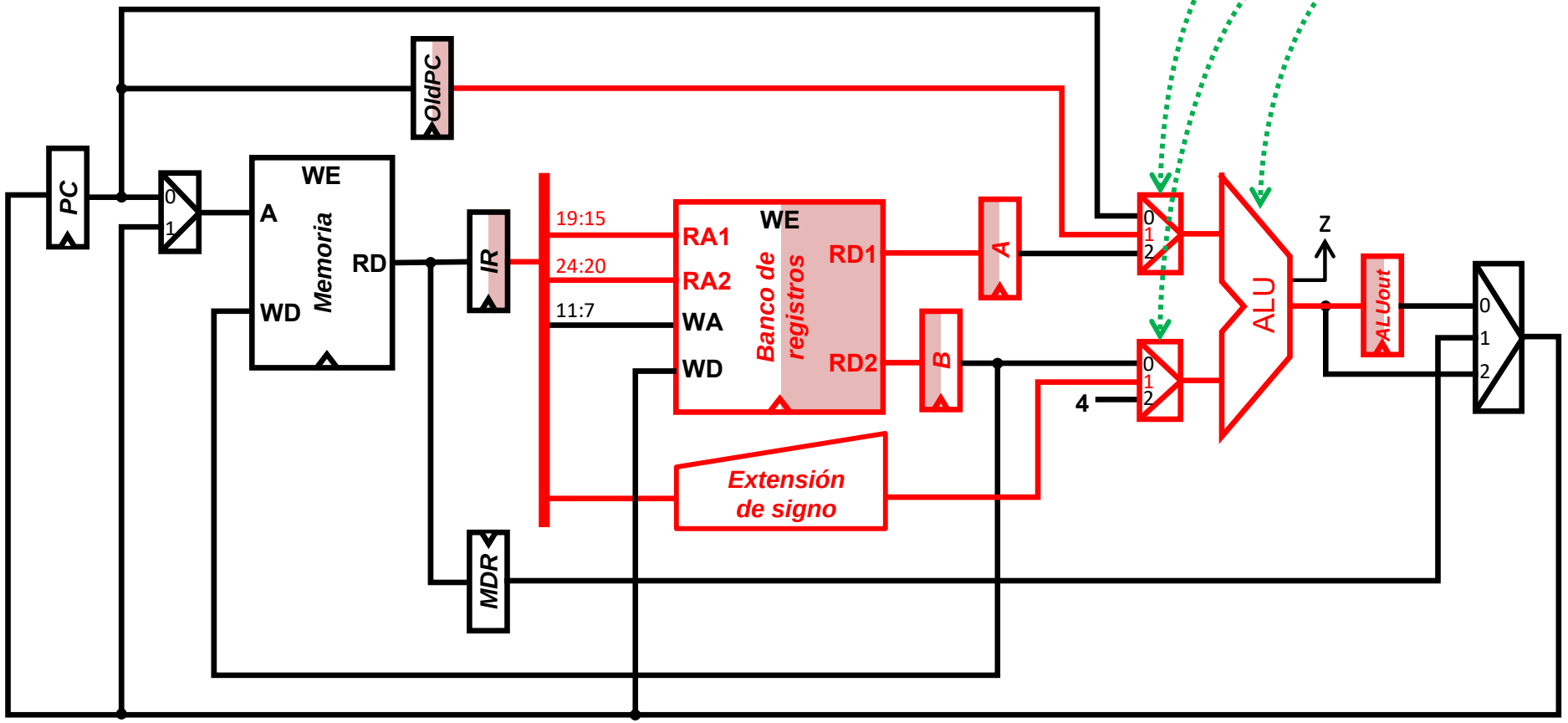
Diseño del controlador

FSM Principal: función de salida

S1

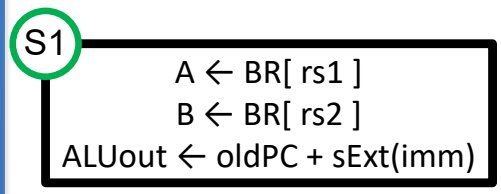
A ← BR[rs1]
B ← BR[rs2]
ALUout ← oldPC + sExt(imm)

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S1	0	0		0	0	0	0	0	1	1	01	01	00	1	

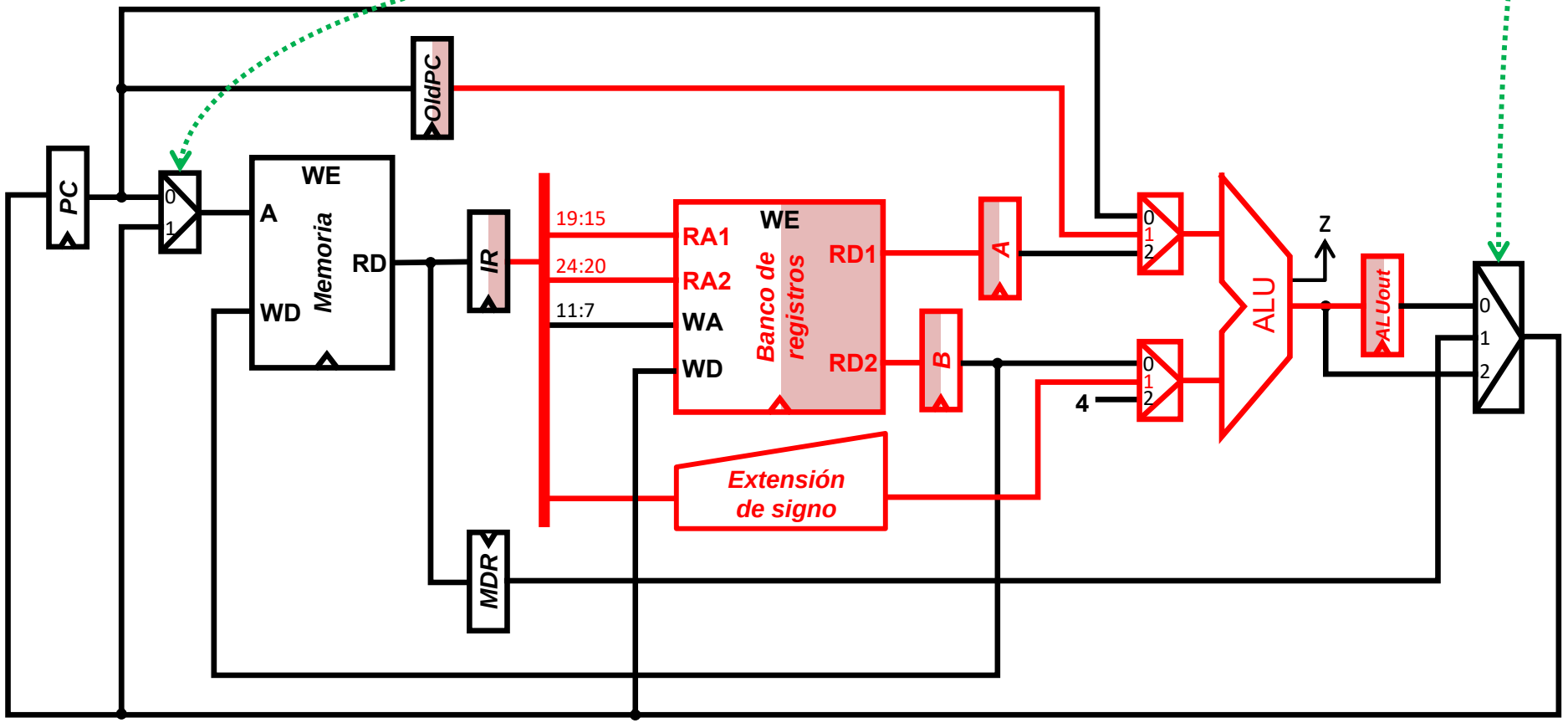


Diseño del controlador

FSM Principal: función de salida



estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–

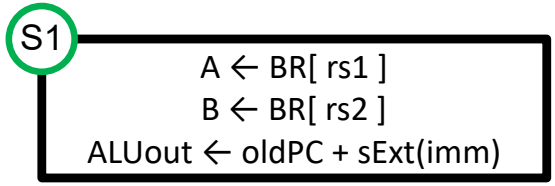


Diseño del controlador

FSM Principal: función de salida

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–
S2															
S3															
S4															
S5															
S6															
S7															
S8															
S9															
S10															

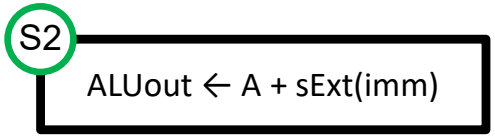


Diseño del controlador

FSM Principal: función de salida

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–
S2	0	0	–	0	0	0	0	0	0	0	10	01	00	1	–
S3															
S4															
S5															
S6															
S7															
S8															
S9															
S10															

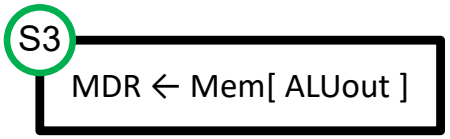


Diseño del controlador

FSM Principal: función de salida

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–
S2	0	0	–	0	0	0	0	0	0	0	10	01	00	1	–
S3	0	0	1	0	0	0	1	0	0	0	–	–	–	0	00
S4															
S5															
S6															
S7															
S8															
S9															
S10															

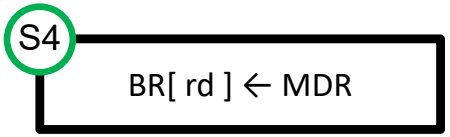


Diseño del controlador

FSM Principal: función de salida

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–
S2	0	0	–	0	0	0	0	0	0	0	10	01	00	1	–
S3	0	0	1	0	0	0	1	0	0	0	–	–	–	0	00
S4	0	0	–	0	0	0	0	1	0	0	–	–	–	0	01
S5															
S6															
S7															
S8															
S9															
S10															

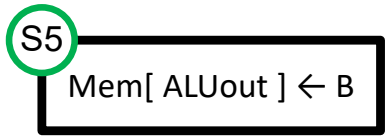


Diseño del controlador

FSM Principal: función de salida

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–
S2	0	0	–	0	0	0	0	0	0	0	10	01	00	1	–
S3	0	0	1	0	0	0	1	0	0	0	–	–	–	0	00
S4	0	0	–	0	0	0	0	1	0	0	–	–	–	0	01
S5	0	0	1	1	0	0	0	0	0	0	–	–	–	0	00
S6															
S7															
S8															
S9															
S10															

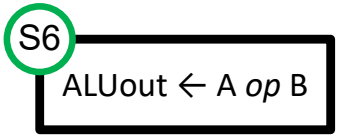


Diseño del controlador

FSM Principal: función de salida

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–
S2	0	0	–	0	0	0	0	0	0	0	10	01	00	1	–
S3	0	0	1	0	0	0	1	0	0	0	–	–	–	0	00
S4	0	0	–	0	0	0	0	1	0	0	–	–	–	0	01
S5	0	0	1	1	0	0	0	0	0	0	–	–	–	0	00
S6	0	0	–	0	0	0	0	0	0	0	10	00	10	1	–
S7															
S8															
S9															
S10															

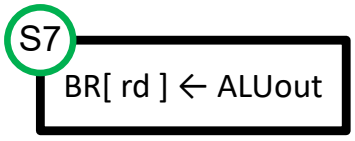


Diseño del controlador

FSM Principal: función de salida

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–
S2	0	0	–	0	0	0	0	0	0	0	10	01	00	1	–
S3	0	0	1	0	0	0	1	0	0	0	–	–	–	0	00
S4	0	0	–	0	0	0	0	1	0	0	–	–	–	0	01
S5	0	0	1	1	0	0	0	0	0	0	–	–	–	0	00
S6	0	0	–	0	0	0	0	0	0	0	10	00	10	1	–
S7	0	0	–	0	0	0	0	1	0	0	–	–	–	0	00
S8															
S9															
S10															

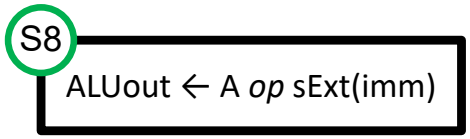


Diseño del controlador

FSM Principal: función de salida

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–
S2	0	0	–	0	0	0	0	0	0	0	10	01	00	1	–
S3	0	0	1	0	0	0	1	0	0	0	–	–	–	0	00
S4	0	0	–	0	0	0	0	1	0	0	–	–	–	0	01
S5	0	0	1	1	0	0	0	0	0	0	–	–	–	0	00
S6	0	0	–	0	0	0	0	0	0	0	10	00	10	1	–
S7	0	0	–	0	0	0	0	1	0	0	–	–	–	0	00
S8	0	0	–	0	0	0	0	0	0	0	10	01	10	1	–
S9															
S10															

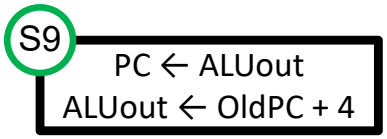


Diseño del controlador

FSM Principal: función de salida

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–
S2	0	0	–	0	0	0	0	0	0	0	10	01	00	1	–
S3	0	0	1	0	0	0	1	0	0	0	–	–	–	0	00
S4	0	0	–	0	0	0	0	1	0	0	–	–	–	0	01
S5	0	0	1	1	0	0	0	0	0	0	–	–	–	0	00
S6	0	0	–	0	0	0	0	0	0	0	10	00	10	1	–
S7	0	0	–	0	0	0	0	1	0	0	–	–	–	0	00
S8	0	0	–	0	0	0	0	0	0	0	10	01	10	1	–
S9	0	1	–	0	0	0	0	0	0	0	01	10	00	1	00
S10															

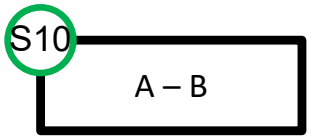


Diseño del controlador

FSM Principal: función de salida

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–
S2	0	0	–	0	0	0	0	0	0	0	10	01	00	1	–
S3	0	0	1	0	0	0	1	0	0	0	–	–	–	0	00
S4	0	0	–	0	0	0	0	1	0	0	–	–	–	0	01
S5	0	0	1	1	0	0	0	0	0	0	–	–	–	0	00
S6	0	0	–	0	0	0	0	0	0	0	10	00	10	1	–
S7	0	0	–	0	0	0	0	1	0	0	–	–	–	0	00
S8	0	0	–	0	0	0	0	0	0	0	10	01	10	1	–
S9	0	1	–	0	0	0	0	0	0	0	01	10	00	1	00
S10	1	0	–	0	0	0	0	0	0	0	10	00	01	0	00



Diseño del controlador

FSM Principal: función de salida

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–
S2	0	0	–	0	0	0	0	0	0	0	10	01	00	1	–
S3	0	0	1	0	0	0	1	0	0	0	–	–	–	0	00
S4	0	0	–	0	0	0	0	1	0	0	–	–	–	0	01
S5	0	0	1	1	0	0	0	0	0	0	–	–	–	0	00
S6	0	0	–	0	0	0	0	0	0	0	10	00	10	1	–
S7	0	0	–	0	0	0	0	1	0	0	–	–	–	0	00
S8	0	0	–	0	0	0	0	0	0	0	10	01	10	1	–
S9	0	1	–	0	0	0	0	0	0	0	01	10	00	1	00
S10	1	0	–	0	0	0	0	0	0	0	10	00	01	0	00

Diseño del controlador

1ª optimización

idénticas

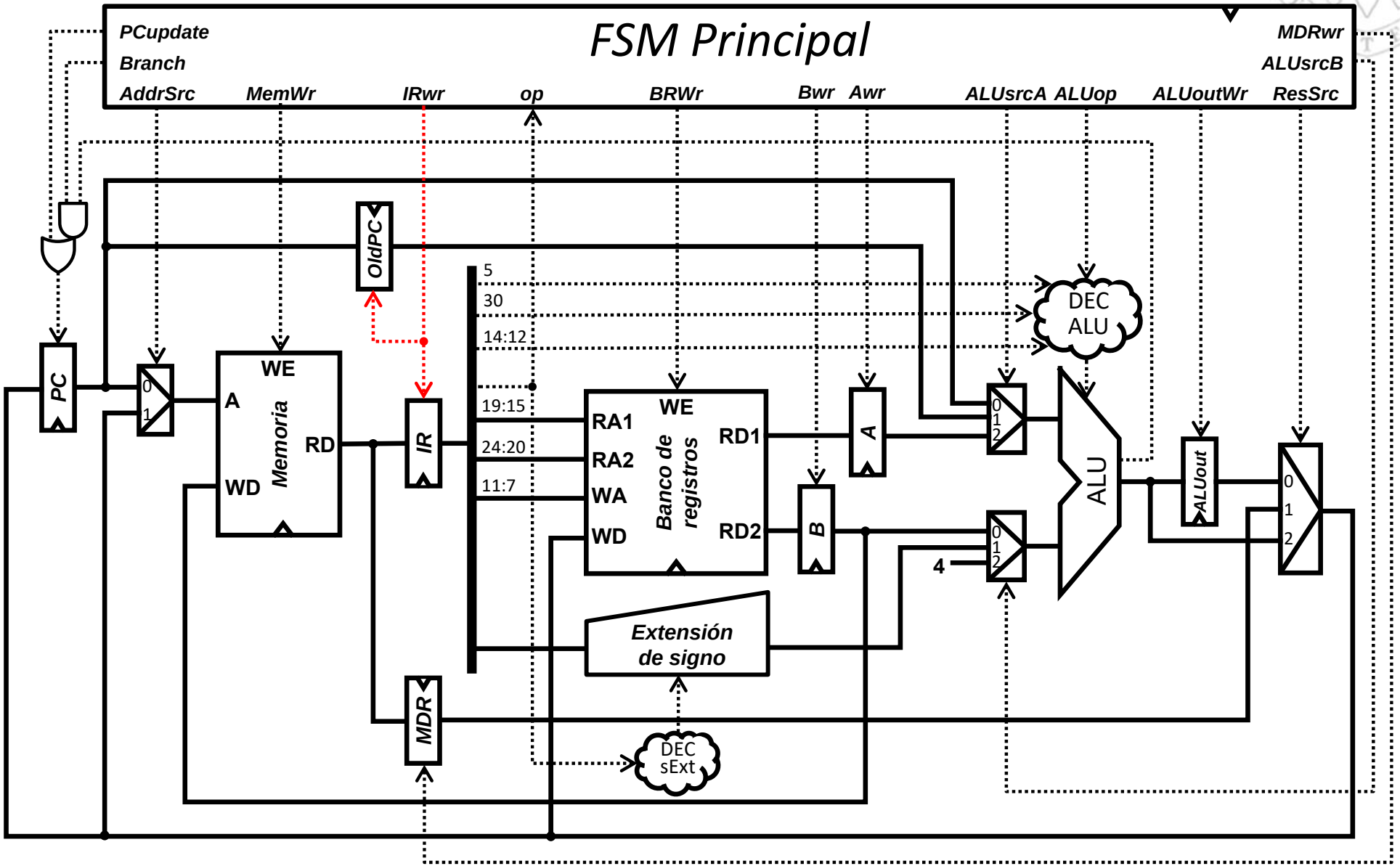
Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	OldPCwr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	1	0	0	0	0	00	10	00	0	10
S1	0	0	–	0	0	0	0	0	1	1	01	01	00	1	–
S2	0	0	–	0	0	0	0	0	0	0	10	01	00	1	–
S3	0	0	1	0	0	0	1	0	0	0	–	–	–	0	00
S4	0	0	–	0	0	0	0	1	0	0	–	–	–	0	01
S5	0	0	1	1	0	0	0	0	0	0	–	–	–	0	00
S6	0	0	–	0	0	0	0	0	0	0	10	00	10	1	–
S7	0	0	–	0	0	0	0	1	0	0	–	–	–	0	00
S8	0	0	–	0	0	0	0	0	0	0	10	01	10	1	–
S9	0	1	–	0	0	0	0	0	0	0	01	10	00	1	00
S10	1	0	–	0	0	0	0	0	0	0	10	00	01	0	00

- Las señales OldPCwr e IRwr tienen la misma tabla de verdad
  - Una puede usarse para controlar ambos puntos de control
  - La otra puede eliminarse del controlador

Diseño del controlador

1ª optimización

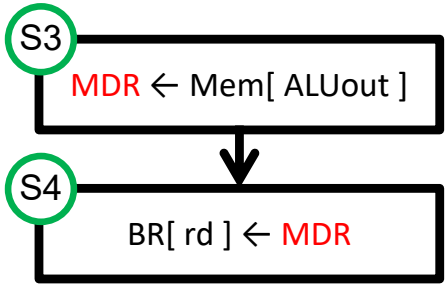


Diseño del controlador

2ª optimización

- Se denomina vida de un dato, el intervalo de ciclos comprendido entre su carga en un registro y su último uso.

- Los datos almacenados en algunos de los registros auxiliares del procesador multiciclo tienen una vida muy corta:
  - El registro auxiliar MDR solo se usa en la ejecución de instrucciones **`lw`**
  - En el estado S3 se carga en MDR un dato leído de memoria, y dicho dato se consume en S4 (el estado siguiente al S3) para almacenarlo en el Banco de Registros.
  - Una vez almacenado el dato, no vuelve a ser necesario en esa instrucción.
  - Por ello, lo que se haga con MDR en el resto de estados es irrelevante.



Diseño del controlador

2ª optimización

Función de salida

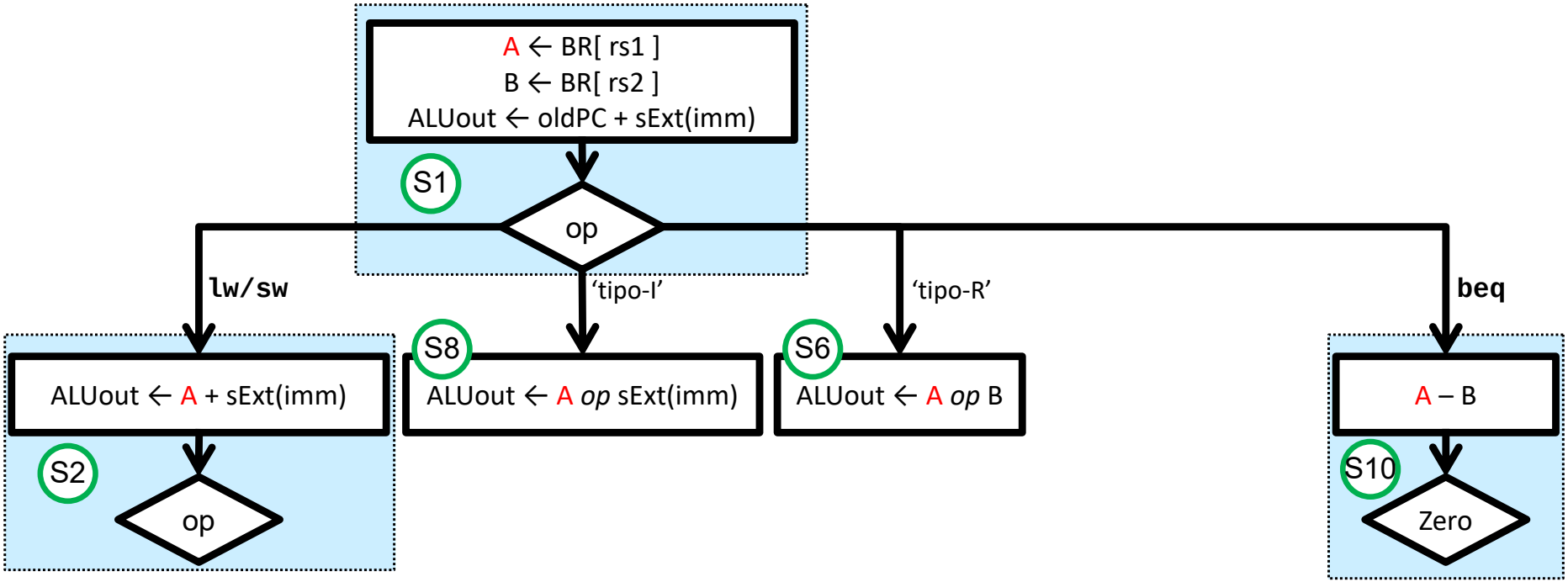
MDRwr	estado	Branch	PCupdate	AddrSrc	MemWr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
0	S0	0	1	0	0	1	–	0	0	0	00	10	00	0	10
0	S1	0	0	–	0	0	–	0	1	1	01	01	00	1	–
0	S2	0	0	–	0	0	–	0	0	0	10	01	00	1	–
1	S3	0	0	1	0	0	1	0	0	0	–	–	–	0	00
0	S4	0	0	–	0	0	–	1	0	0	–	–	–	0	01
0	S5	0	0	1	1	0	–	0	0	0	–	–	–	0	00
0	S6	0	0	–	0	0	–	0	0	0	10	00	10	1	–
0	S7	0	0	–	0	0	–	1	0	0	–	–	–	0	00
0	S8	0	0	–	0	0	–	0	0	0	10	01	10	1	–
0	S9	0	1	–	0	0	–	0	0	0	01	10	00	1	00
0	S10	1	0	–	0	0	–	0	0	0	10	00	01	0	00

- Dado que lo único relevante es que MDR se cargue en el estado S3, la señal MDRwr debe valer 1 en S3
  - Lo que valga en el resto de estados no importa

Diseño del controlador

2ª optimización

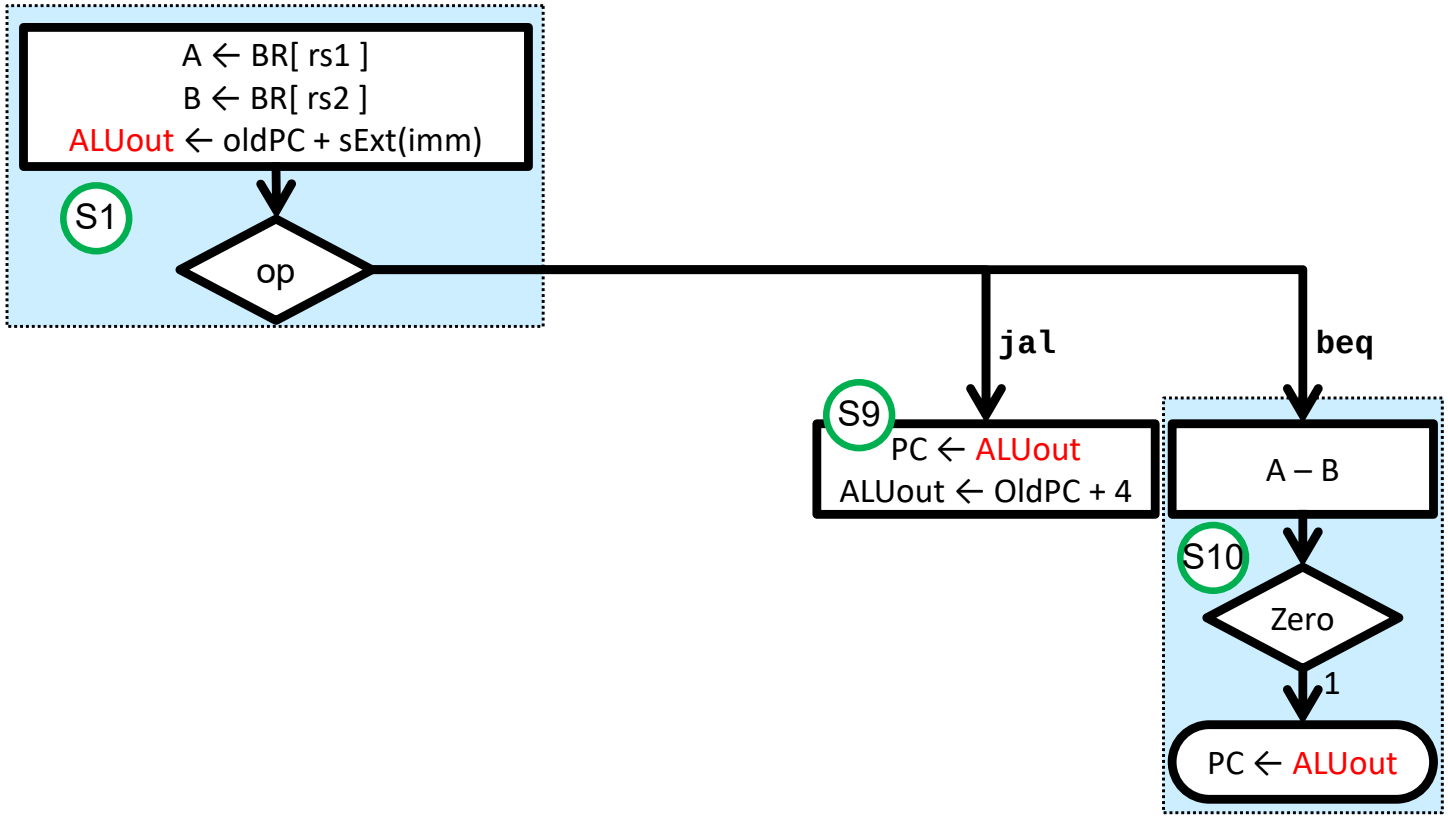
- Caso similar ocurre con los datos almacenados en el registro auxiliar A:
  - En el estado S1 se carga en A un operando fuente leído del Banco de Registros que se consume siempre en el estado siguiente: S2, S8, S6 o S10 según el tipo de instrucción que sea.



Diseño del controlador

2ª optimización

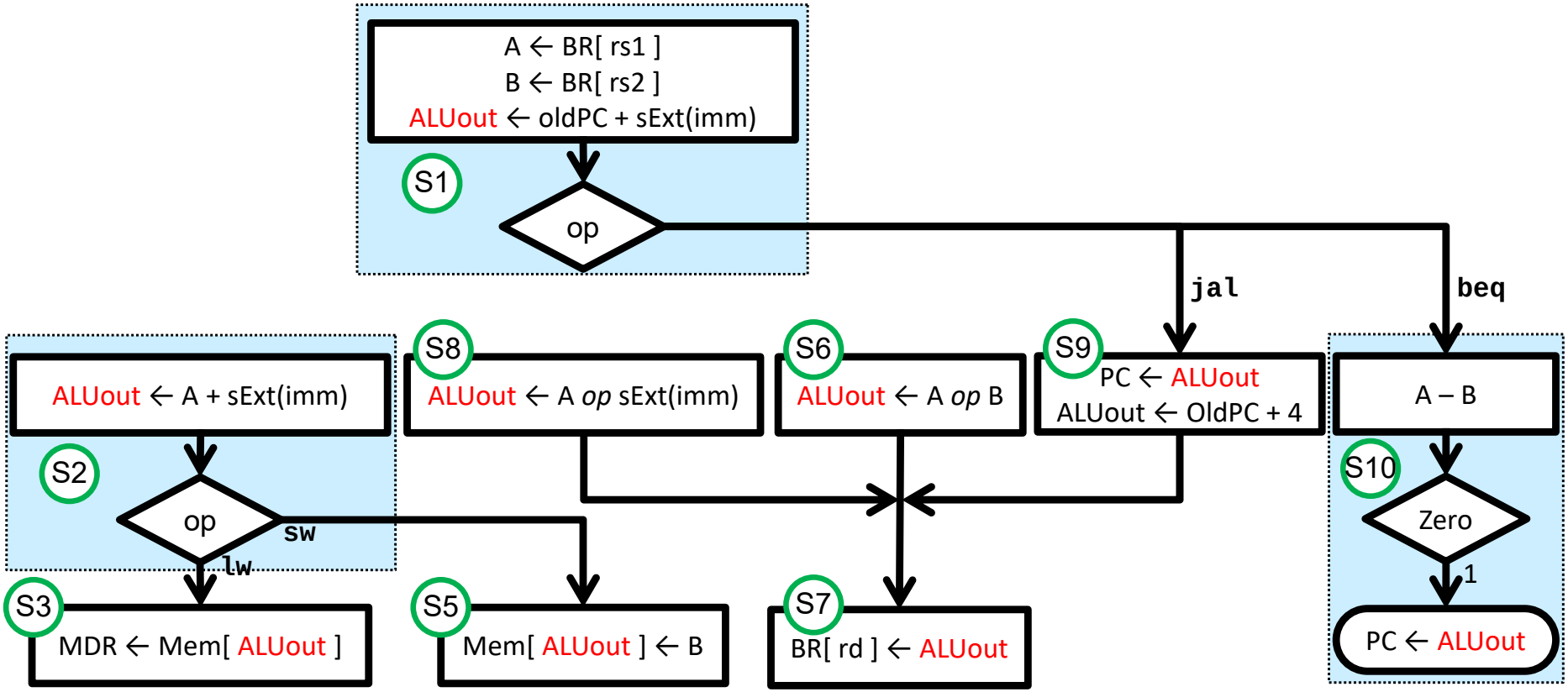
- Ídem con los datos almacenados en el registro auxiliar ALUout:
  - En S1 se carga la dirección de salto que se consume en S9 o S10.



Diseño del controlador

2ª optimización

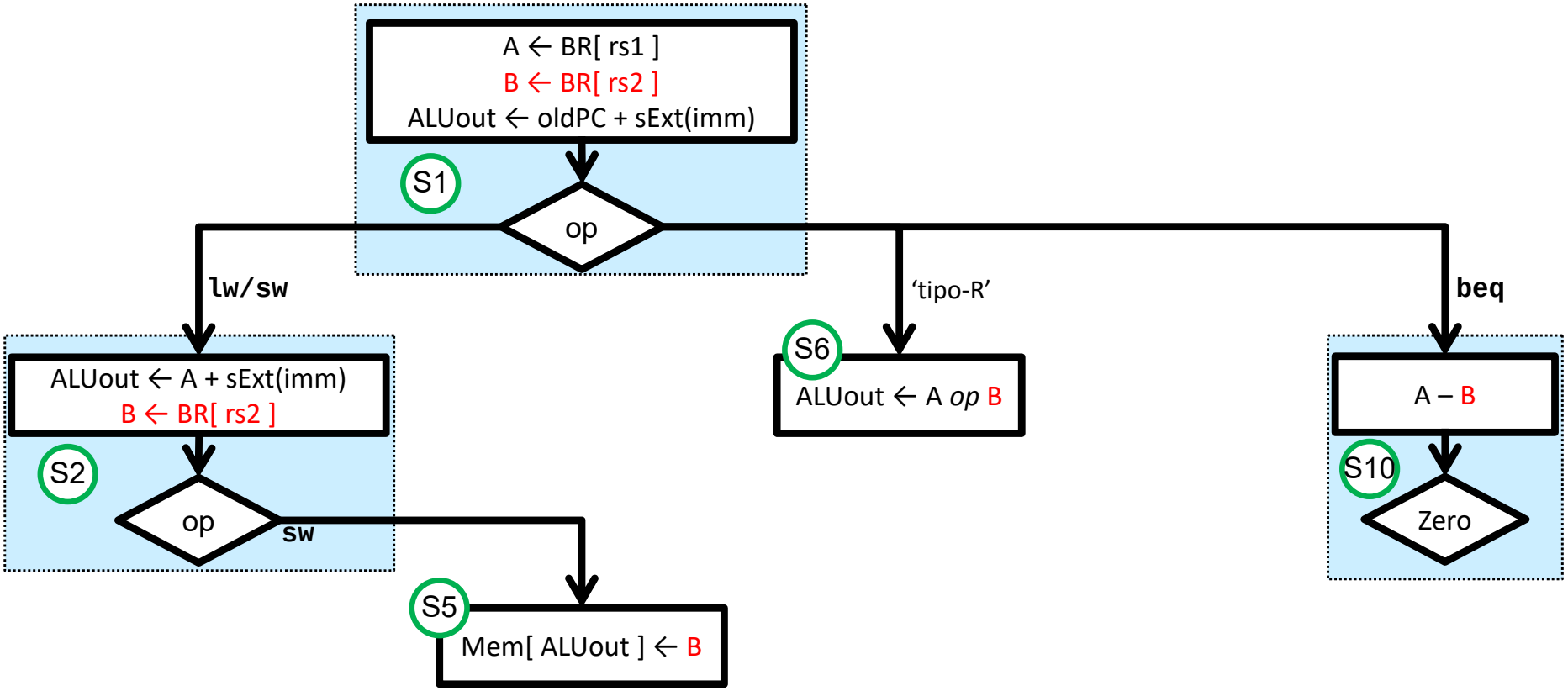
- Ídem con los datos almacenados en el registro auxiliar ALUout:
  - En S1 se carga la dirección de salto que se consume en S9 o S10.
  - En S2 se carga el resultado de una operación de la ALU que se consume en S3 o S5 y los que se cargan en S8, S6 y S9 se consumen en S7.



Diseño del controlador

2ª optimización

- El caso del registro auxiliar B es ligeramente diferente:
  - En el estado S1 se carga en B un operando fuente leído del Banco de Registros que se consume en el ciclo siguiente (S6 o S10) o 2 ciclos después (S5).
  - Pero si B volviera a cargarse en S2 el comportamiento final no se alteraría y el caso sería análogo a los anteriores.



Diseño del controlador

2ª optimización

Función de salida

MDRwr	Awr	Bwr	ALUoutWr
0	0	0	0
0	1	1	1
0	0	0	1
1	0	0	0
0	0	0	0
0	0	0	0
0	0	0	1
0	0	0	0
0	0	0	1
0	0	0	1
0	0	0	0

estado	Branch	PCupdate	AddrSrc	MemWr	IRwr	MDRwr	BRwr	Awr	Bwr	ALUsrcA	ALUsrcB	ALUop	ALUoutWr	ResSrc
S0	0	1	0	0	1	–	0	–	–	00	10	00	–	10
S1	0	0	–	0	0	–	0	1	1	01	01	00	1	–
S2	0	0	–	0	0	–	0	–	1	10	01	00	1	–
S3	0	0	1	0	0	1	0	–	–	–	–	–	–	00
S4	0	0	–	0	0	–	1	–	–	–	–	–	–	01
S5	0	0	1	1	0	–	0	–	–	–	–	–	–	00
S6	0	0	–	0	0	–	0	–	–	10	00	10	1	–
S7	0	0	–	0	0	–	1	–	–	–	–	–	–	00
S8	0	0	–	0	0	–	0	–	–	10	01	10	1	–
S9	0	1	–	0	0	–	0	–	–	01	10	00	1	00
S10	1	0	–	0	0	–	0	–	–	10	00	01	–	00

- Todas estas señales de control pueden valer permanentemente 1 y ser eliminadas del controlador
  - Estos registros auxiliares almacenarán basura en los ciclos no relevantes.

Diseño del controlador

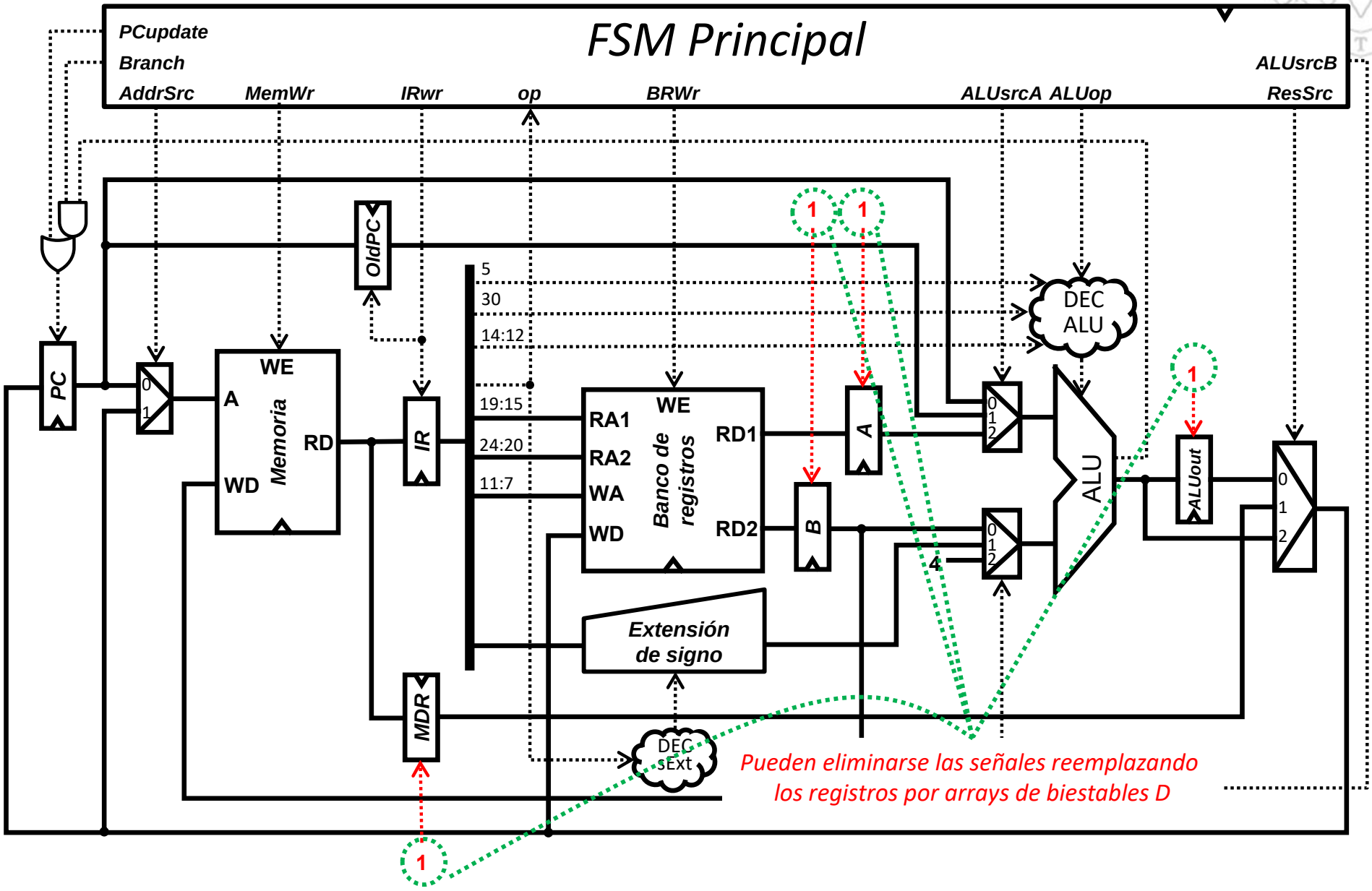
FSM Principal: función de salida optimizada

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	IRwr	BRwr	ALUsrcA	ALUsrcB	ALUop	ResSrc
S0	0	1	0	0	1	0	00	10	00	10
S1	0	0	–	0	0	0	01	01	00	–
S2	0	0	–	0	0	0	10	01	00	–
S3	0	0	1	0	0	0	–	–	–	00
S4	0	0	–	0	0	1	–	–	–	01
S5	0	0	1	1	0	0	–	–	–	00
S6	0	0	–	0	0	0	10	00	10	–
S7	0	0	–	0	0	1	–	–	–	00
S8	0	0	–	0	0	0	10	01	10	–
S9	0	1	–	0	0	0	01	10	00	00
S10	1	0	–	0	0	0	10	00	01	00

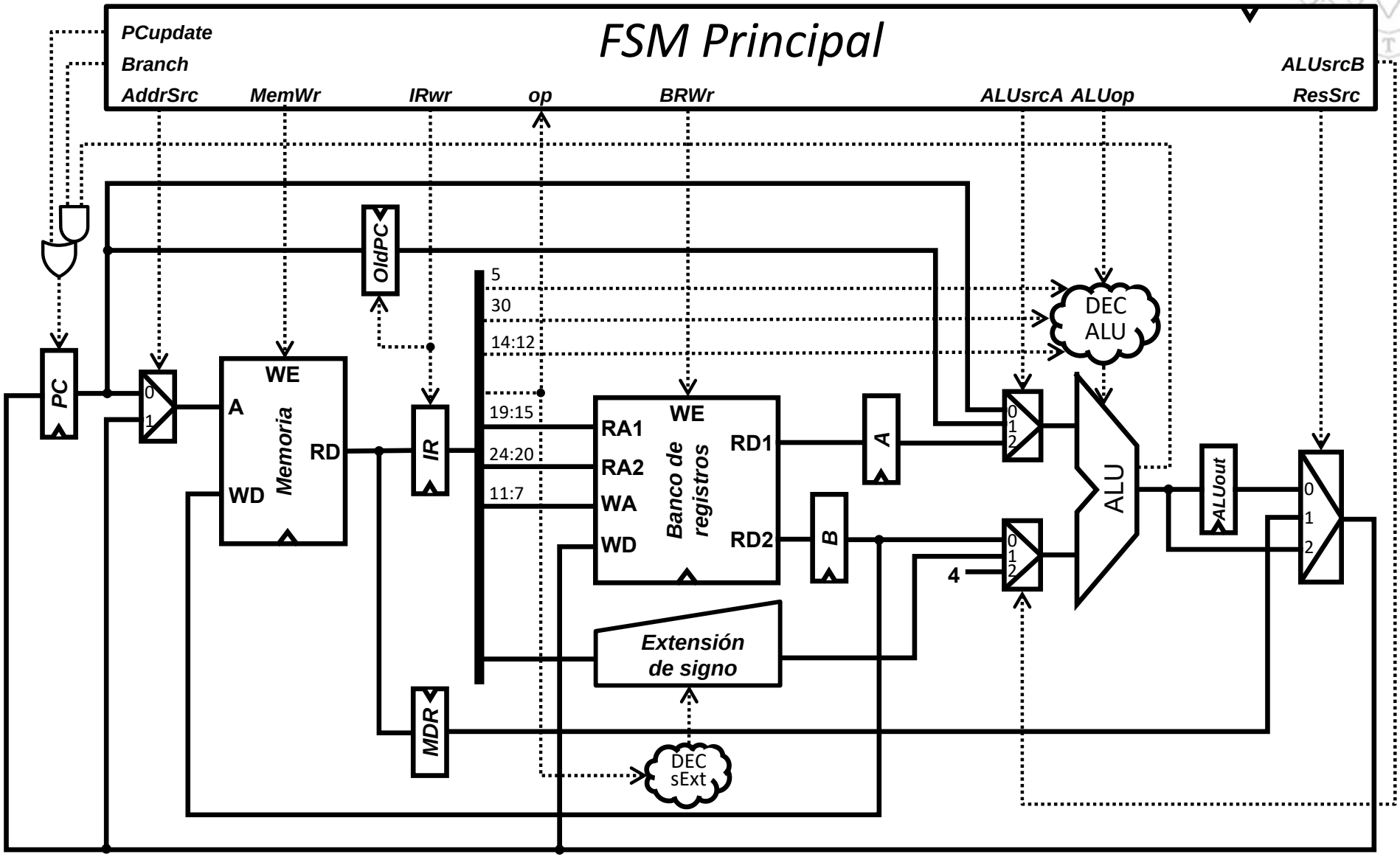
Diseño del controlador

Estructura del sistema completo optimizado



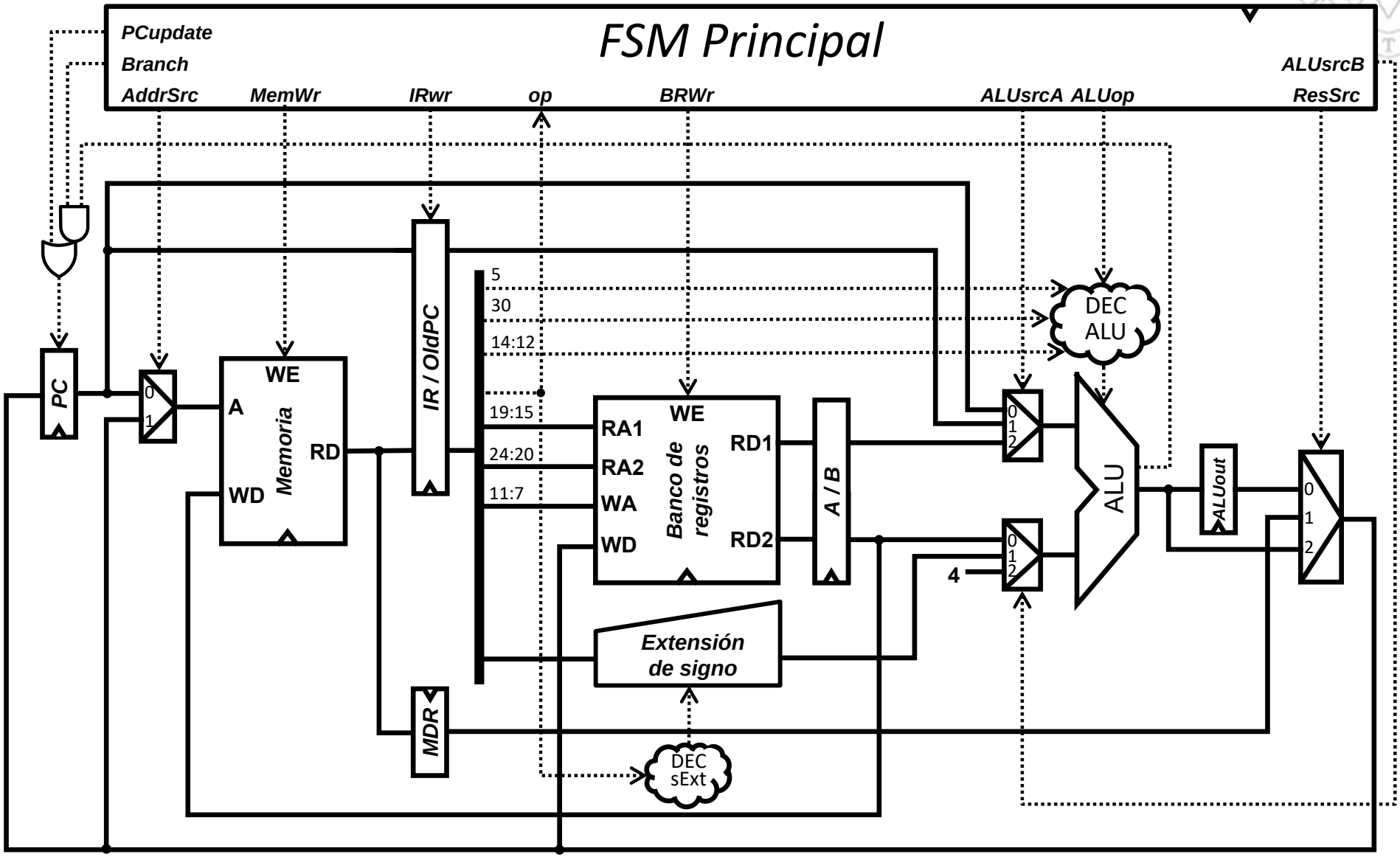
Diseño del controlador

Estructura del sistema completo optimizado



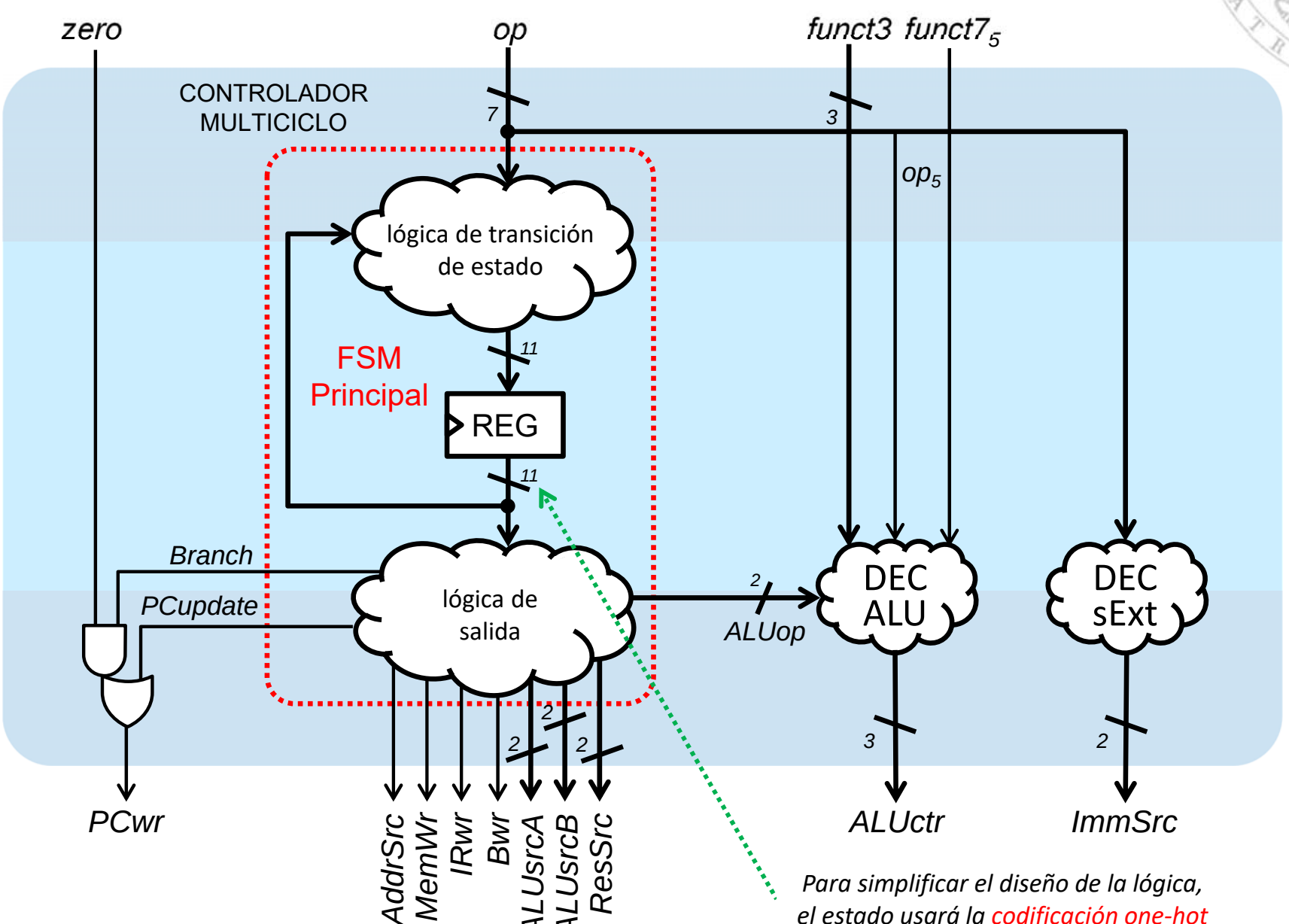
Diseño del controlador

Estructura según nomenclatura Harris & Harris



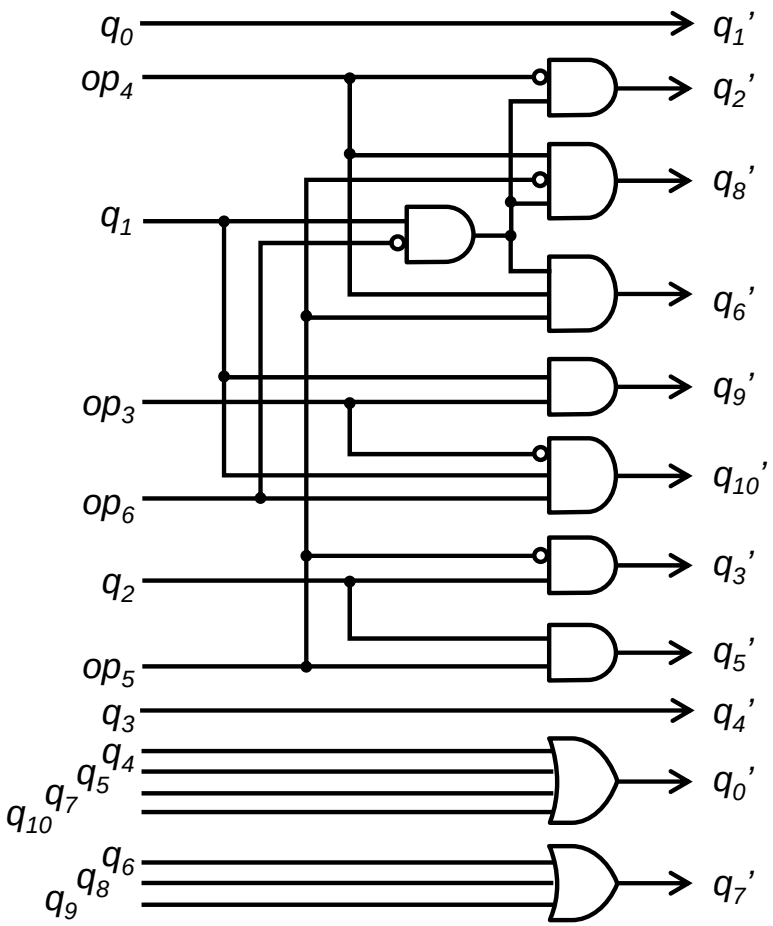
Diseño del controlador

Diseño de la FSM Principal



Diseño del controlador

Diseño de la FSM Principal: función de transición

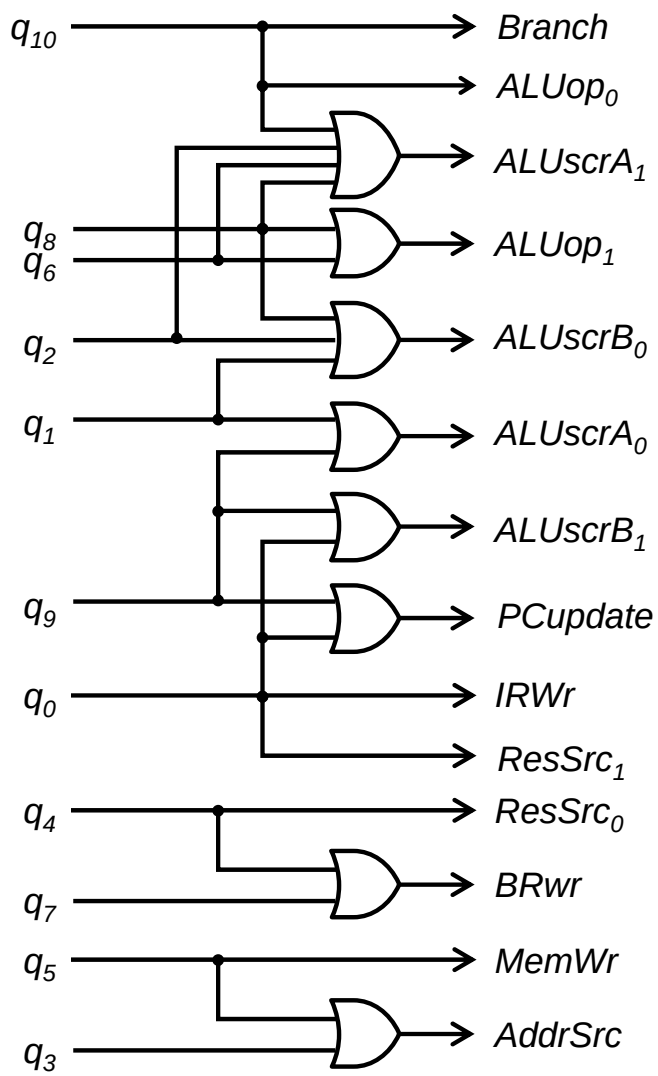


Función de transición de estados

$(q_{10}...q_0)$	op	$(q_{10}...q_0)'$
00000000001 (S0)	XXXXXXX	00000000010 (S1)
00000000010 (S1)	0X00011	00000000100 (S2)
00000000010 (S1)	0010011	00100000000 (S8)
00000000010 (S1)	0110011	00001000000 (S6)
00000000010 (S1)	1101111	01000000000 (S9)
00000000010 (S1)	1100011	10000000000 (S10)
00000000100 (S2)	0000011	00000001000 (S3)
00000000100 (S2)	0100011	00000100000 (S5)
00000001000 (S3)	XXXXXXX	00000010000 (S4)
00000010000 (S4)	XXXXXXX	00000000001 (S0)
00000100000 (S5)	XXXXXXX	00000000001 (S0)
00001000000 (S6)	XXXXXXX	00010000000 (S7)
00010000000 (S7)	XXXXXXX	00000000001 (S0)
00100000000 (S8)	XXXXXXX	00010000000 (S7)
01000000000 (S9)	XXXXXXX	00010000000 (S7)
10000000000 (S10)	XXXXXXX	00000000001 (S0)

Diseño del controlador

Diseño de la FSM Principal: función de salida



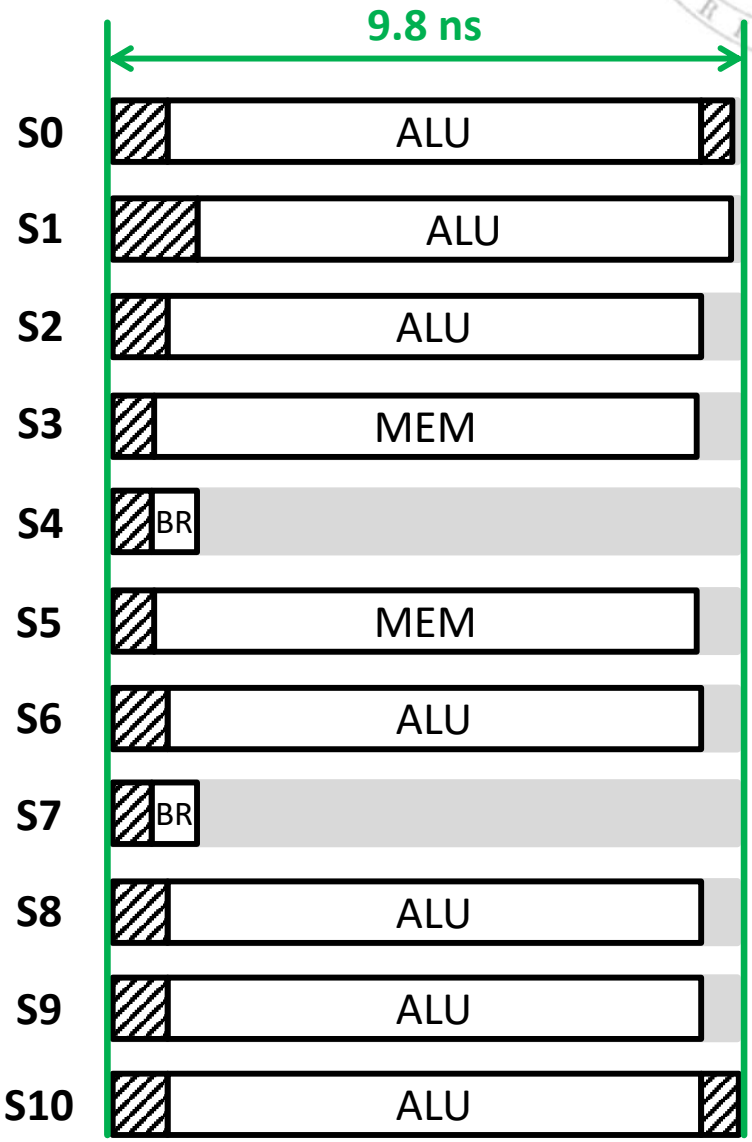
Función de salida

$(q_{10}...q_0)$	Branch	PCupdate	AddrSrc	MemWr	IRwr	BRwr	ALUsrcA	ALUsrcB	ALUop	ResSrc
00000000001 (S0)	0	1	0	0	1	0	00	10	00	10
00000000010 (S1)	0	0	–	0	0	0	01	01	00	–
00000000100 (S2)	0	0	–	0	0	0	10	01	00	–
00000001000 (S3)	0	0	1	0	0	0	–	–	–	00
00000010000 (S4)	0	0	–	0	0	1	–	–	–	01
00000100000 (S5)	0	0	1	1	0	0	–	–	–	00
00001000000 (S6)	0	0	–	0	0	0	10	00	10	–
00010000000 (S7)	0	0	–	0	0	1	–	–	–	00
00100000000 (S8)	0	0	–	0	0	0	10	01	10	–
01000000000 (S9)	0	1	–	0	0	0	01	10	00	00
10000000000 (S10)	1	0	–	0	0	0	10	00	01	00

Procesador multiciclo

Coste y tiempo de ciclo (CMOS 90 nm)

transferencia	estado	camino crítico
IR ← Mem [PC]	S0	9312 ps
PC ← PC+4		9689 ps
OldPC ← PC		589 ps
A ← BR[rs1]	S1	890 ps
B ← BR[rs2]		890 ps
ALUout ← oldPC + sExt(imm)		9688 ps
ALUout ← A + sExt(imm)	S2	9216 ps
MDR ← Mem[ALUout]	S3	9140 ps
BR[rd] ← MDR	S4	1321 ps
Mem[ALUout] ← B	S5	9140 ps
ALUout ← A *op* B	S6	9216 ps
BR[rd] ← ALUout	S7	1321 ps
ALUout ← A *op* sExt(imm)	S8	9216 ps
PC ← ALUout	S9	940 ps
ALUout ← OldPC + 4		9216 ps
if (A – B = 0) *then* PC ← ALUout	S10	**9790 ps**
	máx.	**9790 ps**



Procesador multiciclo

Coste y tiempo de ciclo (CMOS 90 nm)

transferencia	estado	camino crítico
IR ← Mem [PC]	S0	9312 ps
PC ← PC+4		9689 ps
OldPC ← PC		589 ps
A ← BR[rs1]	S1	890 ps
B ← BR[rs2]		890 ps
ALUout ← oldPC + sExt(imm)		9688 ps
ALUout ← A + sExt(imm)	S2	9216 ps
MDR ← Mem[ALUout]	S3	9140 ps
BR[rd] ← MDR	S4	1321 ps
Mem[ALUout] ← B	S5	9140 ps
ALUout ← A *op* B	S6	9216 ps
BR[rd] ← ALUout	S7	1321 ps
ALUout ← A *op* sExt(imm)	S8	9216 ps
PC ← ALUout	S9	940 ps
ALUout ← OldPC + 4		9216 ps
if (A – B = 0) *then* PC ← ALUout	S10	**9790 ps**
	máx.	**9790 ps**

$$area = 65626\ \mu m^2$$

$$t_{clk} = 9.8\ \text{ns}$$

$$f_{clk} = \frac{1}{t_{clk}} = \frac{1}{9.8 \cdot 10^{-9}\text{s}} = 102\ \text{MHz}$$

```
   ...
L7:
   lw   x6, -4(x9)   ·····→  5 ciclos
   sw   x6, 8(x9)    ·····→  4 ciclos
   or   x4, x5, x6   ·····→  4 ciclos
   beq  x4, x4, L7   ·····→  + 3 ciclos
   ...                       16 ciclos
```

$$t_{ejec} = 16 \times 9.8\ ns = 156.8\ ns$$

Comparativa

RISC-V reducido: monociclo vs. multiciclo

- Procesador monociclo:
  - Todas las instrucciones tardan un ciclo en ejecutarse.
  - Tiene un tiempo de ciclo largo limitado por la instrucción más lenta.
  - Todos los recursos están dedicados a hacer una única operación.
  - Requiere memoria de datos e instrucciones separada.

- Procesador multiciclo:
  - Todas las instrucciones tardan más de un ciclo en ejecutarse.
    - Las instrucciones simples tardan menos ciclos en ejecutarse que las complejas.
  - Tiene un tiempo de ciclo corto limitado por micro-operación más lenta.
  - Los recursos pueden reutilizarse para hacer diferentes operaciones en distintos ciclos de reloj.
  - Solo existe una única memoria común para datos e instrucciones.
  - Requiere usar registros auxiliares no arquitectónicos (es decir, invisibles al programador) para almacenar resultados parciales.

Comparativa

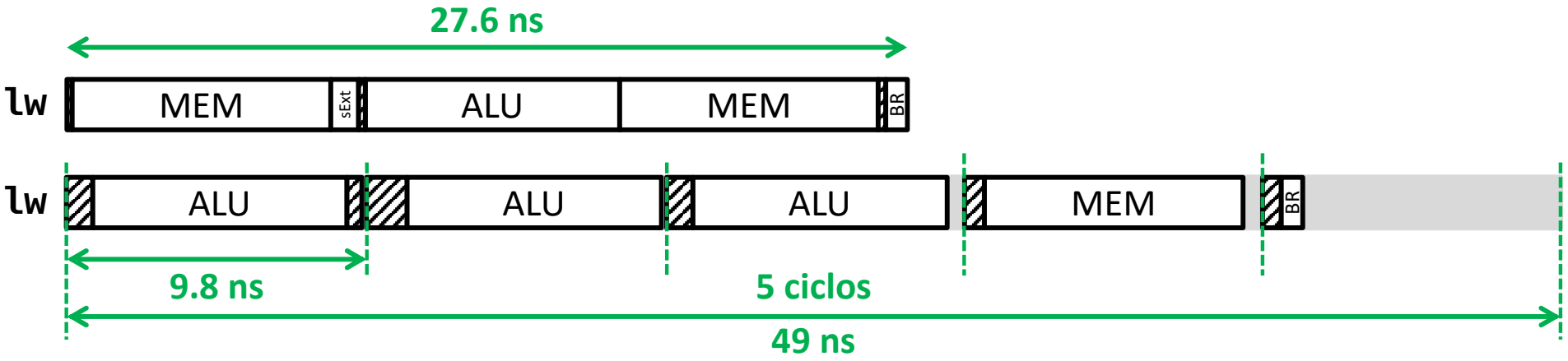
RISC-V reducido: monociclo vs. multiciclo

- El procesador multiciclo es más caro que el monociclo

$$65626\ \mu m^2 = coste_{multiciclo} > coste_{monociclo} = 59181\ \mu m^2$$

- El procesador multiciclo tiene peor rendimiento que el monociclo

$$156.8\ ns = t_{ejec-multiciclo} > t_{ejec-monociclo} = 110.4\ ns$$



- Aparentemente el procesador multiciclo es peor diseño, sin embargo, no existen en el mercado procesadores monociclo.
  - La razón de esta paradoja se debe a las simplificaciones que hemos asumido durante el proceso de diseño.

Comparativa

RISC-V reducido: monociclo vs. multiciclo

- El procesador monociclo ha resultado más barato porque:
  - No se ha tenido el cuenta el coste de la memoria.
    - Este procesador requiere 2 memorias y la versión multiciclo solo 1.
  - Reutilizar la ALU requiere añadir multiplexores y registros que tienen un coste mayor que los sumadores eliminados.
    - Con un grado de reuso mayor, el procesador multiciclo se abarata.
- El procesador multiciclo ha resultado tener peor rendimiento porque:
  - Se ha asumido que el tiempo de acceso de la memoria y el tiempo de cómputo de la ALU son menores que el tiempo de ciclo.
    - Las memorias RAM externas y las ALU complejas son más lentas.
  - La carga computacional de los estados no está perfectamente equilibrada.
    - Existen ciclos en donde la mayor parte del HW está inactivo.
- Un procesador monociclo solo es mejor en arquitecturas simplificadas.
  - Repertorios complejos, requieren añadir mayor número de elementos funcionales que no pueden reutilizarse y que alargan el camino crítico.

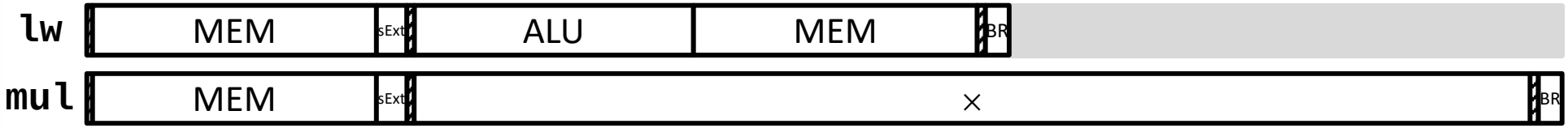
Comparativa

RISC-V reducido: monociclo vs. multiciclo

- Por ejemplo, supongamos que ampliamos el repertorio para que soporte la instrucción de multiplicación:

`mul rd, rs1, rs2`	rd ← rs1 × rs2	tipo-R

- Añadimos a ambas rutas de datos un multiplicador combinacional de 32 bits, con un tiempo de cómputo 4 veces superior al de la ALU.
  - En el procesador monociclo, el tiempo de ciclo estaría determinado por la ejecución de la instrucción **`mul`** que es ahora la más lenta.



$$t_{clk-monociclo} = \mathbf{44\ ns}\quad \mathbf{(18928\ ps\ +\ 3 \times 8360\ ps)}$$
$$t_{ejec-moniciclo} = 4 \times 44\ ns\ = \mathbf{176\ ns}$$

  - En el procesador multiciclo, el tiempo de ciclo podría dejarse inalterado a costa de que la instrucción **`mul`** tarde 7 ciclos en ejecutarse.
    - El número de ciclos requerido por las restantes instrucciones no cambiaría.

$$t_{clk-multiciclo} = \mathbf{9.8\ ns}$$
$$t_{ejec-multiciclo} = 16 \times 9.8\ ns\ = \mathbf{156.8\ ns}$$

Métricas de rendimiento

Tiempo de ejecución de un programa

- Las métricas de rendimiento permiten comparar objetivamente las prestaciones de distintos computadores.
  - Desde el punto de vista del usuario, el tiempo de ejecución total de un programa es el más fiable.
  - Dado que depende de múltiples factores nos centraremos en tiempo de CPU.
- El tiempo de ejecución de N instrucciones de i tipos de un programa dado en una cierta CPU será:

$$\boldsymbol{t_{ejec}} = t_{clk} \cdot \sum_i c_i \cdot n_i \equiv \sum_i c_i \cdot n_i / f_{clk}$$

- donde:
  - n_i = número de instrucciones de tipo i ejecutadas.
  - c_i = número de ciclos que tarda en ejecutarse cada instrucción de tipo i.
  - t_{clk} = tiempo de ciclo (en segundos/ciclo).
  - f_{clk} = frecuencia de reloj (en herzios = ciclos/segundo).
  - $N \equiv \sum_i n_i$ = número total de instrucciones ejecutadas.
  - $\sum_i c_i \cdot n_i$ = número de ciclos total que tarda en ejecutarse el programa.

Métricas de rendimiento

Tiempo de ejecución de un programa

- Para poder dar una expresión más compacta del tiempo de ejecución se introduce el concepto de Ciclos Promedio por Instrucción (CPI)
  - CPI es la suma ponderada del número de ciclos que tarda por separado cada tipo de instrucción.
  - El CPI es específico para cada programa y cada CPU.

$$\boldsymbol{CPI} = \frac{\sum_i c_i \cdot n_i}{\sum_i n_i}$$

- El tiempo de ejecución de N instrucciones de un programa dado en una cierta CPU será:

$$\boldsymbol{t_{ejec}} = N \cdot CPI \cdot t_{clk} \equiv \frac{N \cdot CPI}{f_{clk}}$$

- donde:
  - $N \equiv \sum_i n_i$ = número total de instrucciones ejecutadas.
  - $N \cdot CPI$ = número de ciclos total que tarda en ejecutarse el programa.
  - t_{clk} = tiempo de ciclo (en segundos/ciclo).
  - f_{clk} = frecuencia de reloj (en hertzios = ciclos/segundo).

Métricas de rendimiento

Tiempo de ejecución de un programa

- Esta expresión es interesante porque permite localizar la contribución que tiene cada elemento del diseño en el rendimiento del computador.

Habilidad del programador
Compilador
Arquitectura del computador

Organización del computador
Tecnología microelectrónica

CISC

R1 ← [addr(A) + 4 * R2] + 1

RISC

R3 ← 4 * R2
R3 ← addr (A) + R3
R5 ← [R4]
R1 ← R5 + 1

$$t_{ejec} = N \cdot CPI \cdot t_{clk}$$

Arquitectura del computador
Organización del computador

Métricas de rendimiento

MIPS y MFLOPS

- Comúnmente se usan otras métricas menos fiables que indican el número de instrucciones que ejecuta un computador por segundo.
  - Las instrucciones de una familia arquitectónica pueden tener funcionalidades y duraciones muy diferentes a otra (RISC vs CISC) haciendo inútil la comparación.

- Millones de Instrucciones por Segundo (MIPS)

$$MIPS = \frac{N}{10^6 \cdot t_{ejec}} \equiv \frac{f_{clk}}{10^6 \cdot CPI}$$

- Millones de instrucciones en punto flotante por segundo (MFLOPS)
  - Se escogen estas por ser las instrucciones que más tardan en ejecutarse.
  - No sirve para comparar procesadores/programas con aritmética entera.

Métricas de rendimiento

RISC-V reducido: monociclo vs. multiciclo

- Sea un programa que ejecuta 10^8 instrucciones (100 millones) tal que:
  - 25% de las instrucciones son de tipo **`lw`**
  - 10% de las instrucciones son de tipo **`sw`**
  - 11% de las instrucciones son de tipo **`beq`**
  - 2% de las instrucciones son de tipo **`jal`**
  - 52% de las instrucciones son aritmético-lógicas

- CPI monociclo: todas las instrucciones tardan un ciclo (CPI = 1).

$$\boldsymbol{CPI} = \mathbf{1}$$
$$\boldsymbol{t_{ejec}} = 10^8 \cdot 1 \cdot 27.6\, ns = \mathbf{2.76\ s}$$
$$\boldsymbol{MIPS} = 10^8/(10^6 \cdot 2.76\ s) = \mathbf{32.6\ Minst/s}$$

- CPI multiciclo: todas las instrucciones tardan más de un ciclo (CPI > 1).
  - **`lw`**: 5 ciclos, **`sw`**: 4 ciclos, **`beq`**: 3 ciclos, **`jal`**: 4 ciclos, aritmético-lógicas: 4 ciclos

$$\boldsymbol{CPI} = 0.25 \cdot 5 + 0.10 \cdot 4 + 0.11 \cdot 3 + 0.02 \cdot 4 + 0.52 \cdot 4 = \mathbf{4.14}$$
$$\boldsymbol{t_{ejec}} = 10^8 \cdot 4.14 \cdot 9.8\, ns = \mathbf{4.06\ s}$$
$$\boldsymbol{MIPS} = 10^8/(10^6 \cdot 4.06\ s) = \mathbf{25\ Minst/s}$$

Métricas de rendimiento

Speedup

- La comparación entre los tiempos absolutos de ejecución de un mismo programa en 2 procesadores distintos es poco visual.
  - Es más efectivo indicar cuanto más rápido es un procesador respecto a otro.
- Se denomina *speedup* a la relación entre los tiempos de ejecución de un mismo programa en dos procesadores A y B
  - Mide cuánto más rápido es el procesador A respecto al procesador B ejecutando el programa.

$$Speedup = \frac{t_{ejec}^{B}}{t_{ejec}^{A}}$$

- Así, diremos que el procesador monociclo ejecuta el anterior programa 1.47 veces más rápido que el procesador multiciclo porque:

$$t_{ejec}^{monociclo} = 2.76\ s \qquad t_{ejec}^{multiciclo} = 4.06\ s$$

$$Speedup = \frac{t_{ejec}^{multiciclo}}{t_{ejec}^{monociclo}} = \frac{4.06}{2.76} = 1.47$$

Métricas de rendimiento

Benchmarking

- El resultado de cualquier métrica de rendimiento depende del programa que se ejecute
  - Para que la comparación sea justa el programa debe ser el mismo.
  - Pero un único programa puede no ser suficiente representativo.

- Un benchmark es una colección de programas que se usan para comparar computadores entre sí
  - Dhrystone, CoreMark: programas sintéticos de cálculo entero.
  - Whetstone: programas sintéticos de cálculo en punto flotante.
  - Linkpack: programas reales de cálculo científico.
  - SPEC: programas reales con versión entera y punto flotante.

Métricas de rendimiento

TOP500

- El proyecto Top500 es un ranking de las 500 supercomputadoras con mayor rendimiento del mundo.

#	Computador	País	Fabricante	# cores	Procesador	Rpico (Pflops*)	Potencia (kW)
1	Frontier	EEUU	HP	8730112	AMD Opt 3rd Gen EPYC 64C @ 2GHz	**1685.65**	21100
2	Fugaku	Japón	Fujitsu	7630848	A64FX 48C @ 2.2GHz	537.21	29899
3	LUMI	Finlandia	HP	1110144	AMD Opt 3rd Gen EPYC 64C @ 2GHz	214.35	2942
4	Summit	EEUU	IBM	2414592	IBM POWER9 22C @ 3.07GHz	200.79	10096
5	Sierra	EEUU	IBM	1572480	IBM POWER9 22C @ 3.1GHz	125.71	7438
...	...	...	...	...	...	...	...
82	MareNostrum	España	Lenovo	153216	Xeon Platinum 8160 24C @ 2.1GHz	10.30	1632

fuente: Top 500 (June 2022), https://www.top500.org/

(*) 1 Pflops = 10^9 Mflops

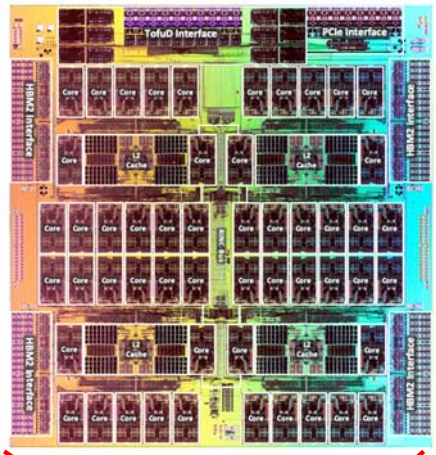
- Mi portátil.

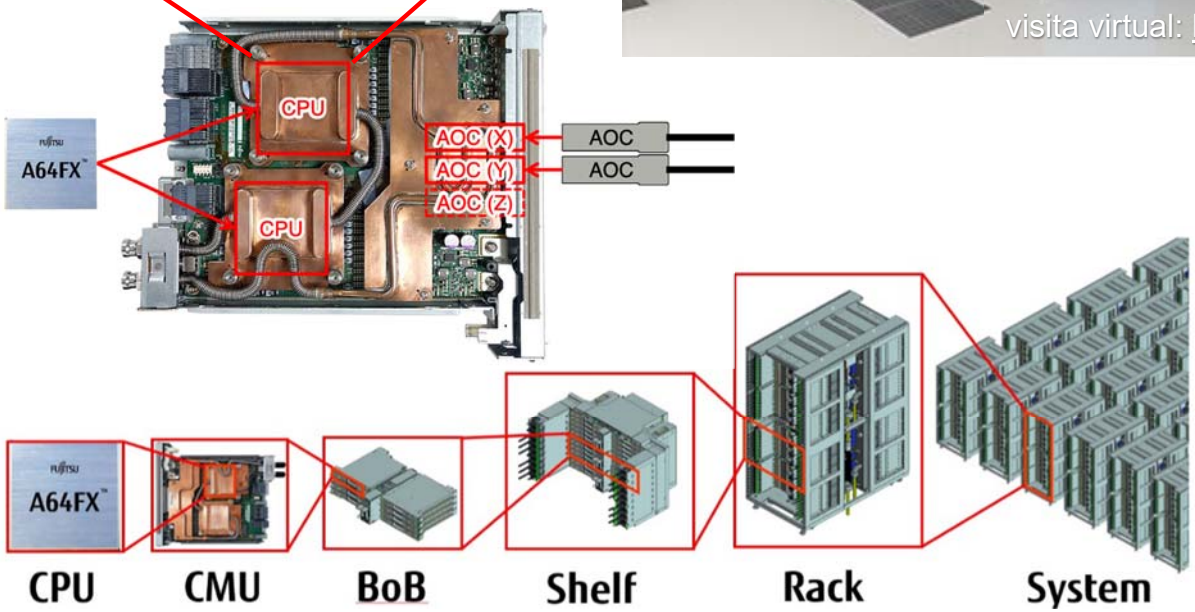
#	Computador	País	Fabricante	# cores	Procesador	Rpico (Mflops)	Potencia (kW)
–	Elite Dragonfly	EEUU	HP	4	Intel Core i7-8565U @ 1.80GHz	**13.77**	–

fuente: PassMark Software Pty Ltd , https://www.cpubenchmark.net

Métricas de rendimiento

TOP500: Fugaku



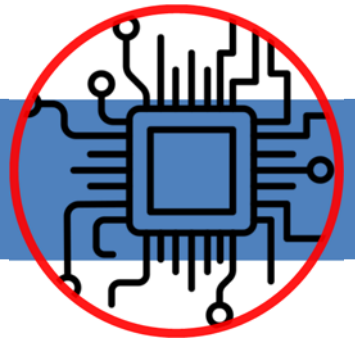


Unidad	# nodos	Descripción
CPU	1	Nodo de 48+2 procesadores
CMU	2	CPU Memory Unit: 2 CPUs
BoB	16	Bunch of Blades: 8 CMUs
Shelf	48	3 BoBs
Rack	192/384	4/8 Shelfs
Fugaku	158976	396 + 36 Racks (8/4 Shelfs)

fuentes: Jack Dongarra, Report on the Fujitsu Fugaku System, Tech Report No. ICL-UT-20-06 (2020)
Riken Center for Computational Science, https://www.r-ccs.riken.jp/en/

- Cálculo del coste.
- Cálculo del tiempo de ciclo.

Apéndice tecnológico



Cálculo del coste y tiempo de ciclo

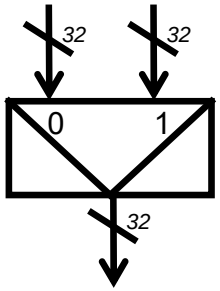
- Para calcular el coste del procesador es la suma del coste de cada uno de los módulos que lo componen.
  - El coste de cada módulo se calcula sumando el coste de sus celdas.

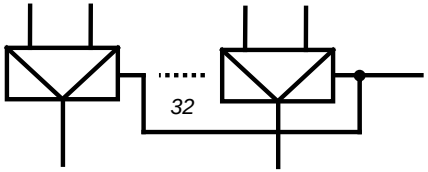
- El tiempo de ciclo del procesador es el máximo de los caminos críticos de las transferencias entre registros que realiza el procesador.
  - El camino crítico de una transferencia entre registros es el camino de datos de mayor retardo de todos los implicados en dicha transferencia.
  - En el procesador multiciclo, en cada ciclo se realizan entre 1 y 3 transferencias entre registros, la ejecución de una instrucción implica la realización en total de entre 7 y 9 transferencias entre registros.

- Para todos los cálculos se utilizará la misma biblioteca de celdas (CMOS 90nm) usada en FC-1

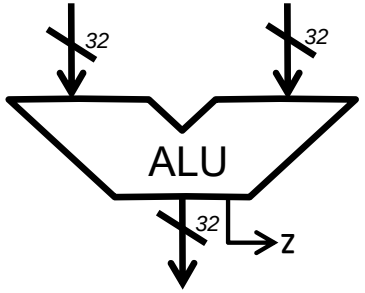
Cálculo del coste y tiempo de ciclo

CMOS 90 nm

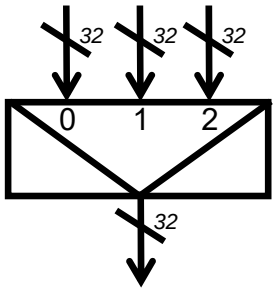


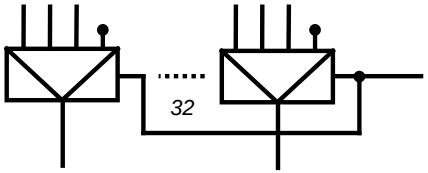


área: 32×11.05 = **354 μm²**
retardo: **223 ps**

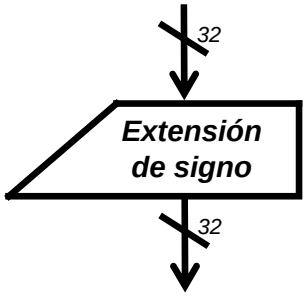


área: **3052 μm²**
retardo: **8360 ps**





área: 32×23.04 = **737 μm²**
retardo: **250 ps**



área: **202 μm²**
retardo: **460 ps**

área: **56 μm²**
retardo: **490 ps**

área: **15 μm²**
retardo: **351 ps**

área: **21 μm²**
retardo: **451 ps**

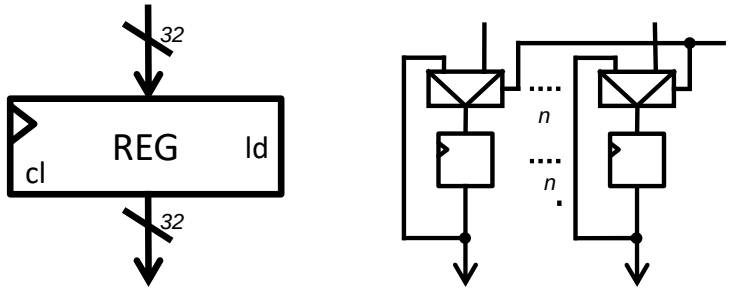
área: **107 μm²**
retardo: **486 ps**

área: **64 μm²**
retardo: **199 ps**

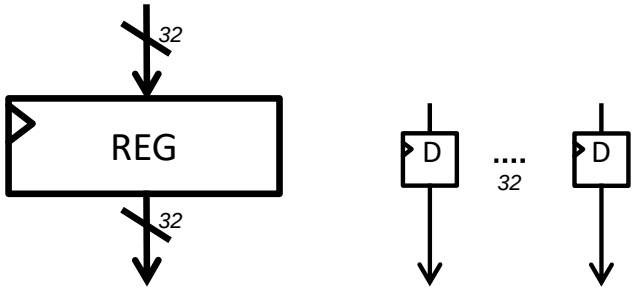
Biblioteca de celdas CMOS 90 nm, fuente: Synopsys (SAED EDK 90 nm)

Cálculo del coste y tiempo de ciclo

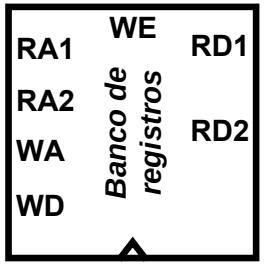
CMOS 90 nm



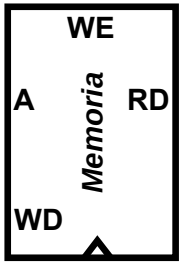
área: 32×11.05 + 32×32.26 = **1386 μm²**
retardo CLK→Q: **167 ps**
setup: 1×223 = **223 ps** (debido a MUX de carga)



área: 32×24.88 = **796 μm²**
retardo CLK→Q: **167 ps**
setup: **0 ps**

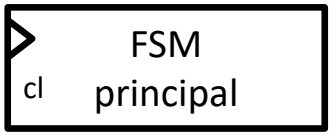


área: **51405 μm²**
retardo lectura: **723 ps**
setup escritura: **705 ps**
(debido al DEC de dirección)



área: –
tiempo acceso: **8500 ps**

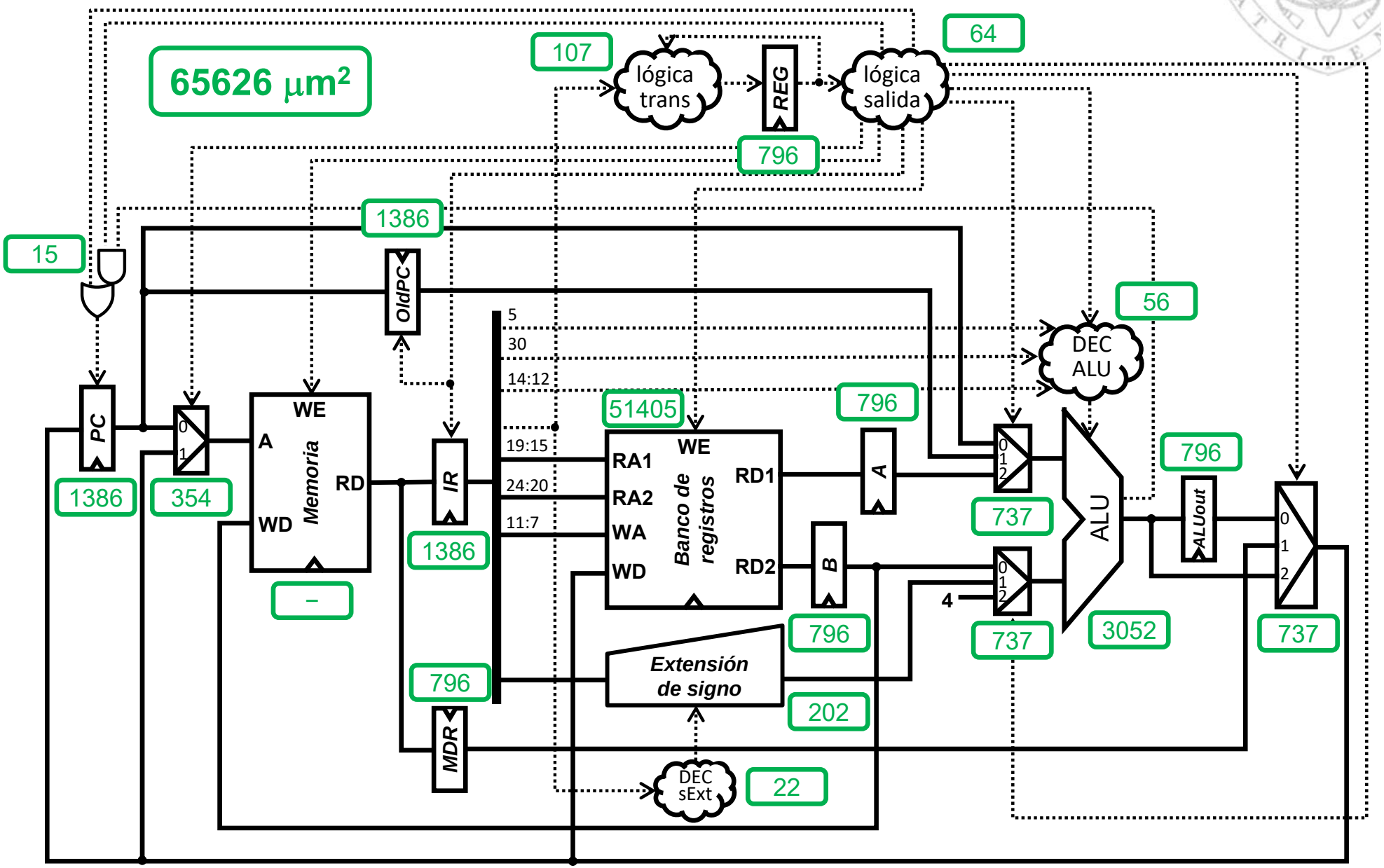
Comportamiento idealizado:
retardo comparable al de la ALU
(para que pueda leerse en un ciclo de reloj)



área: **525 μm²**
retardo CLK→Q: **366 ps**
setup: **486 ps** (debido a la lógica trans.)

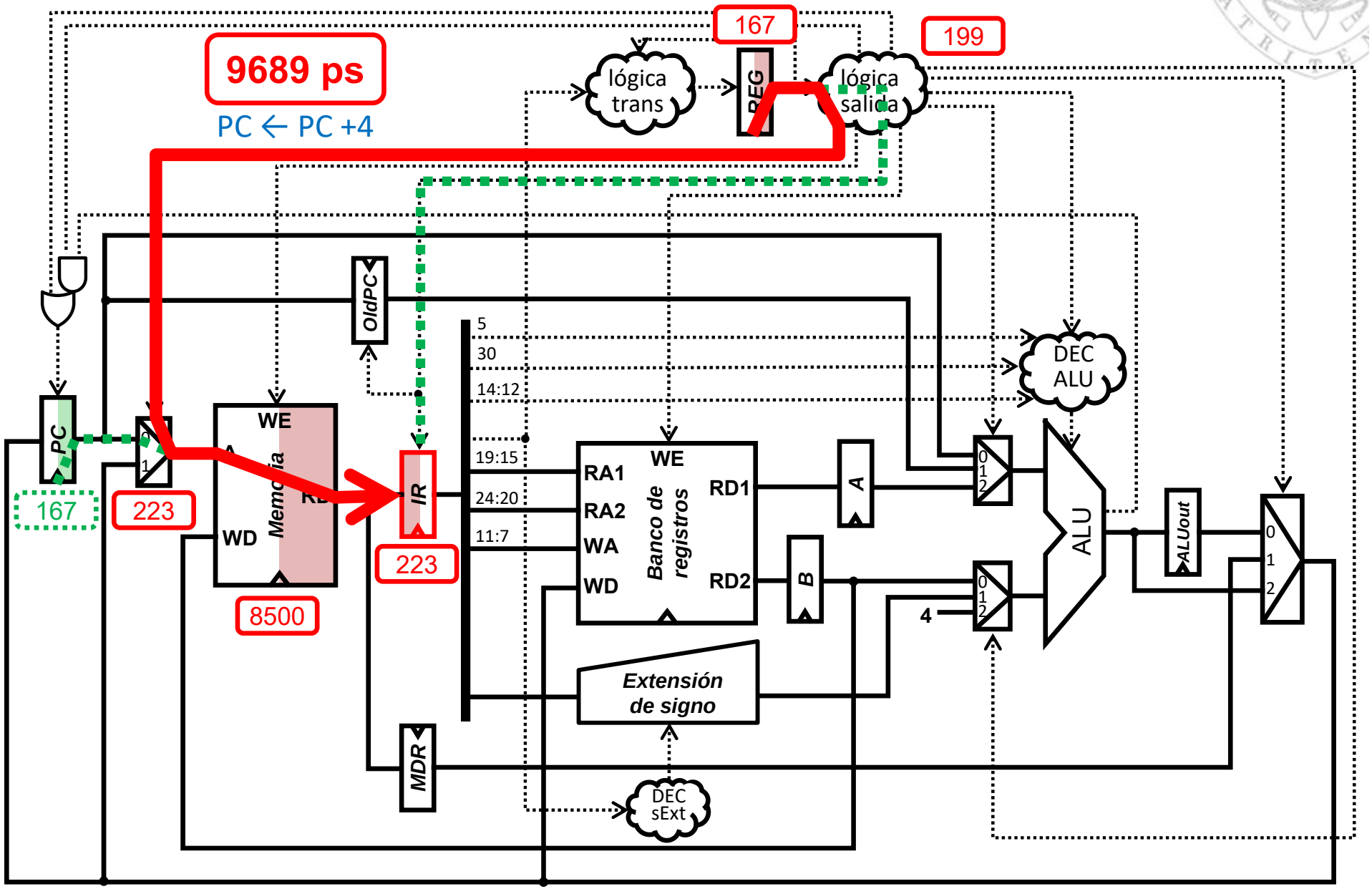
Biblioteca de celdas CMOS 90 nm, fuente: Synopsys (SAED EDK 90 nm)

Cálculo del coste



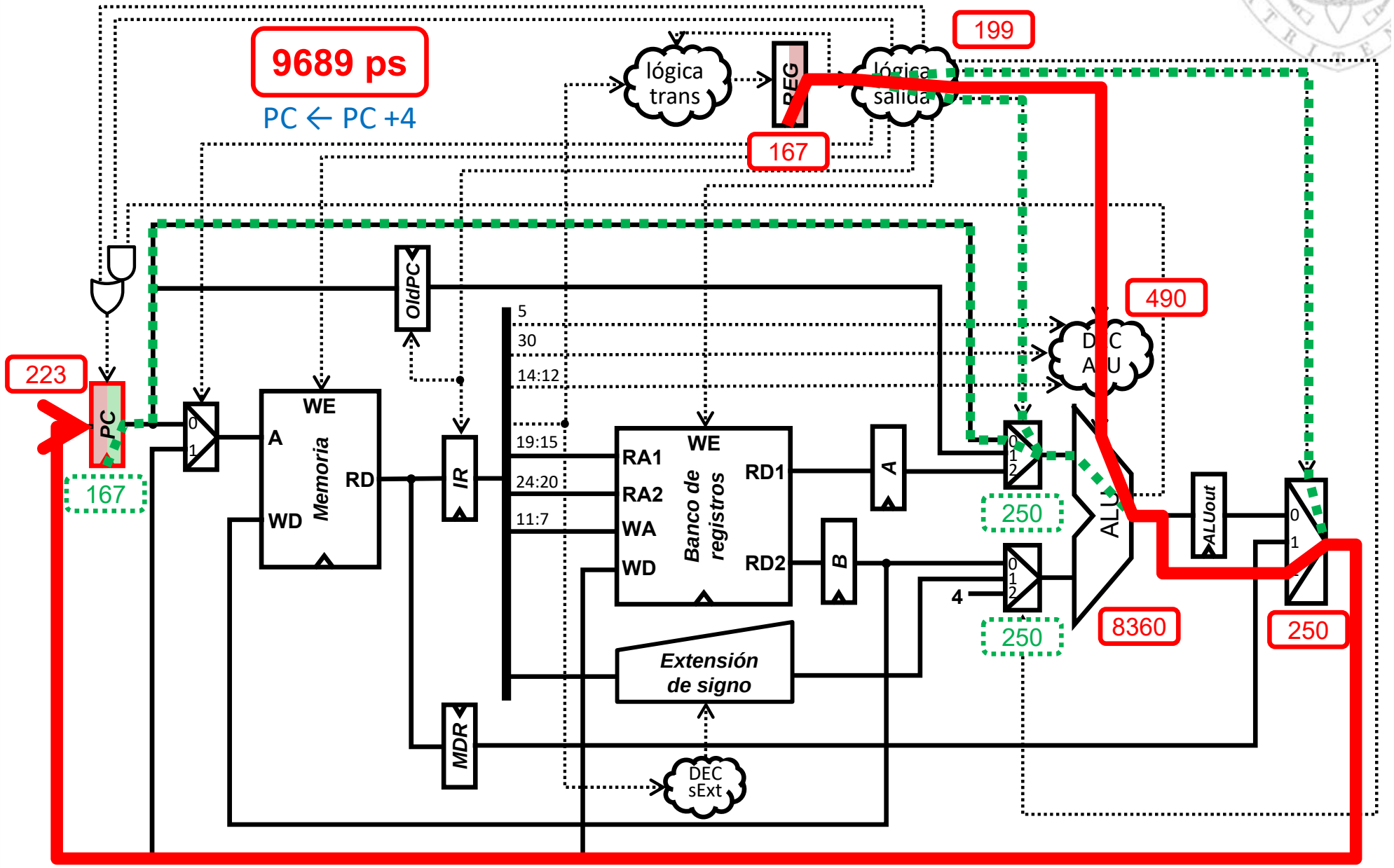
Cálculo del tiempo de ciclo

Estado S0 (búsqueda de instrucción): camino crítico



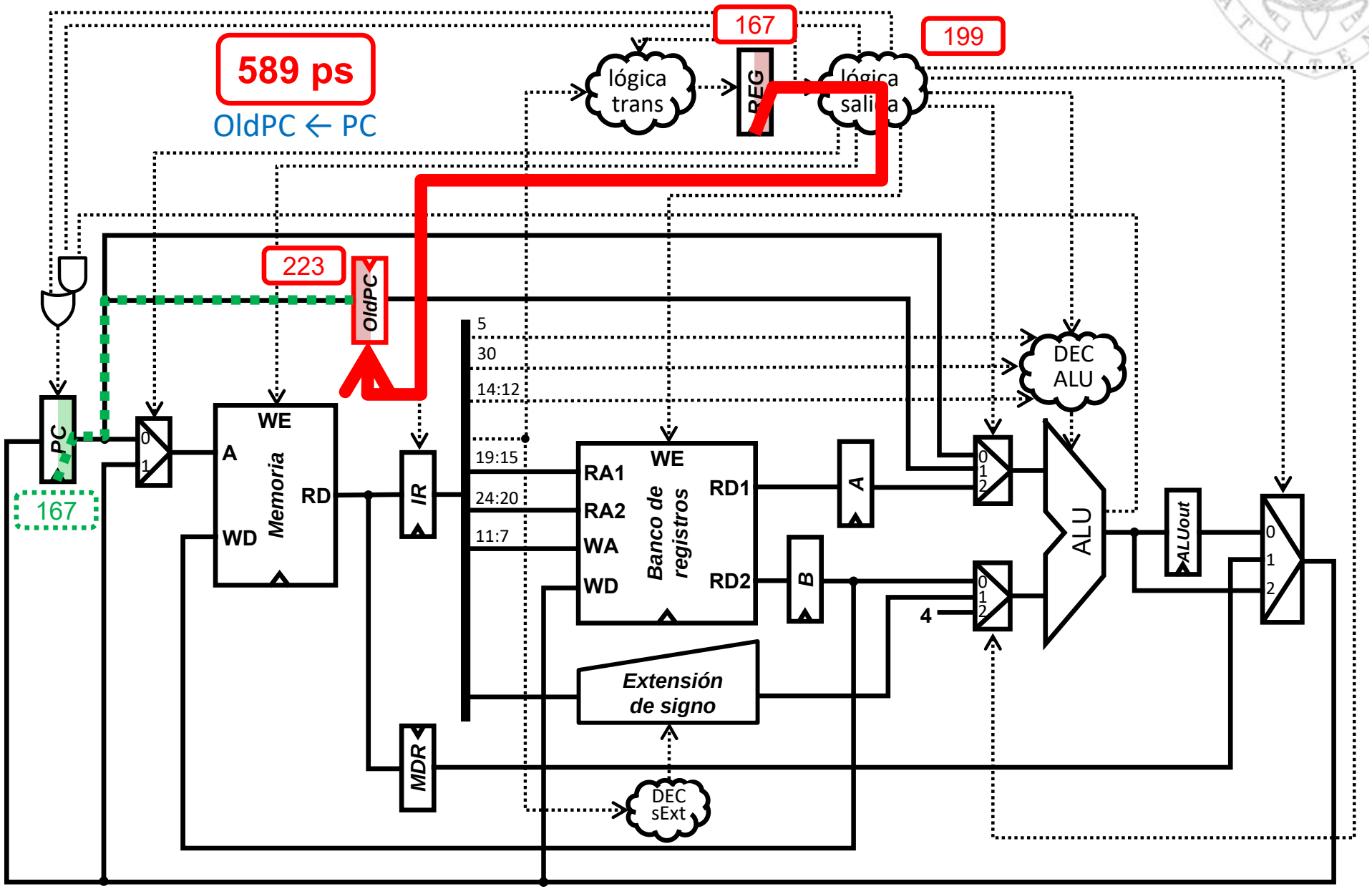
Cálculo del tiempo de ciclo

Estado S0 (incremento del PC): camino crítico



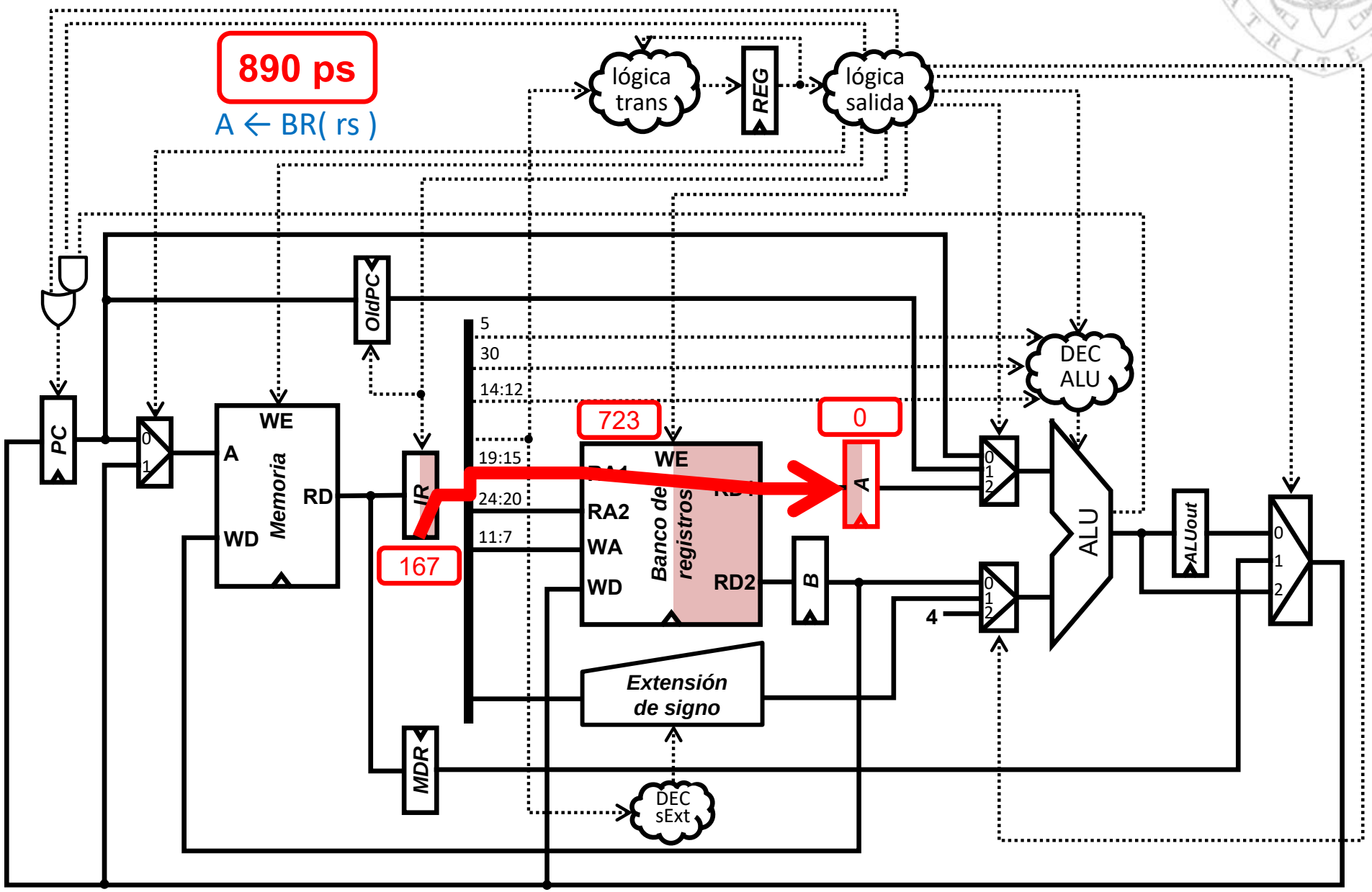
Cálculo del tiempo de ciclo

Estado S0 (salvado del PC): camino crítico



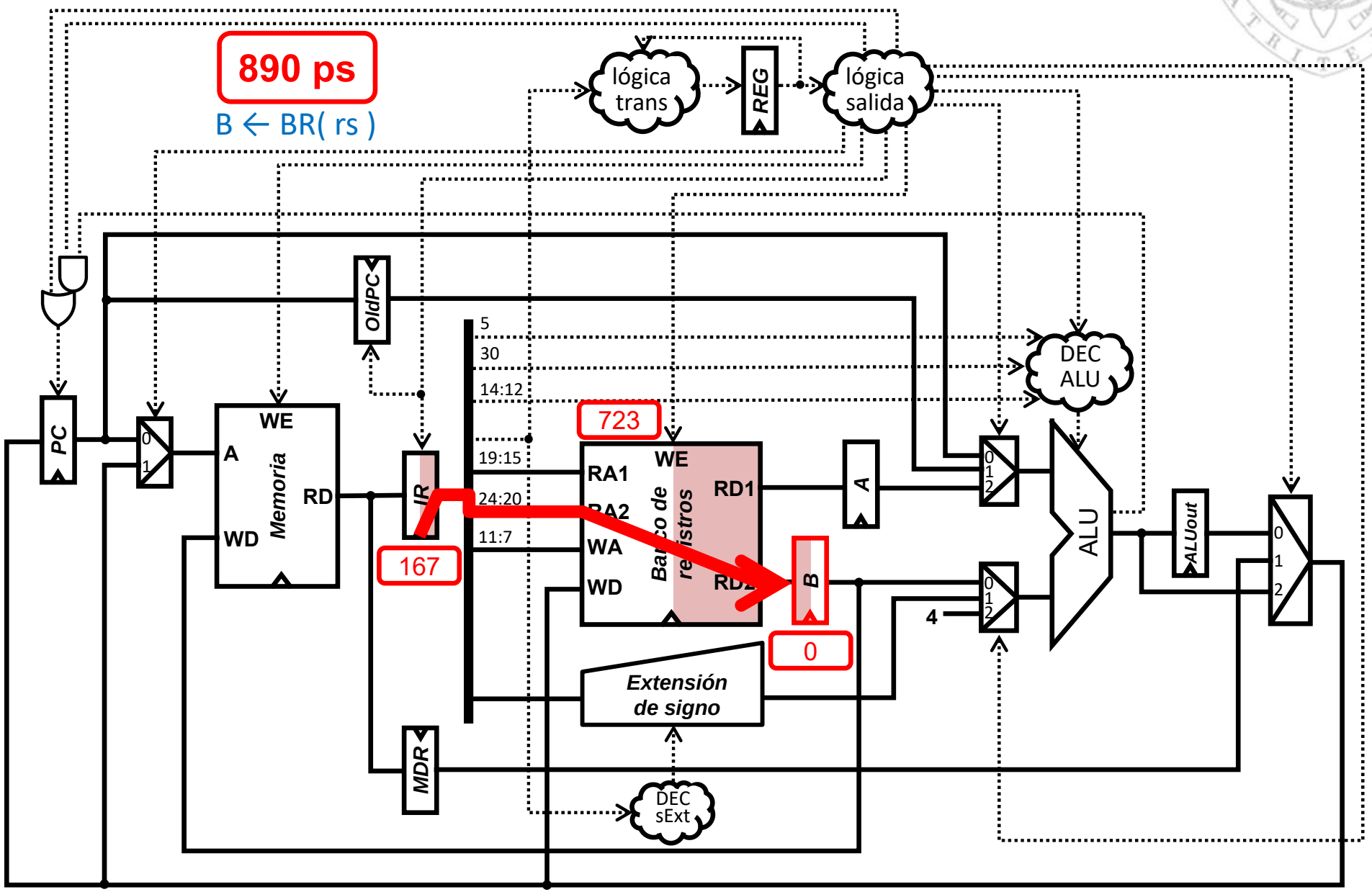
Cálculo del tiempo de ciclo

Estado S1 (búsqueda de operandos): camino crítico



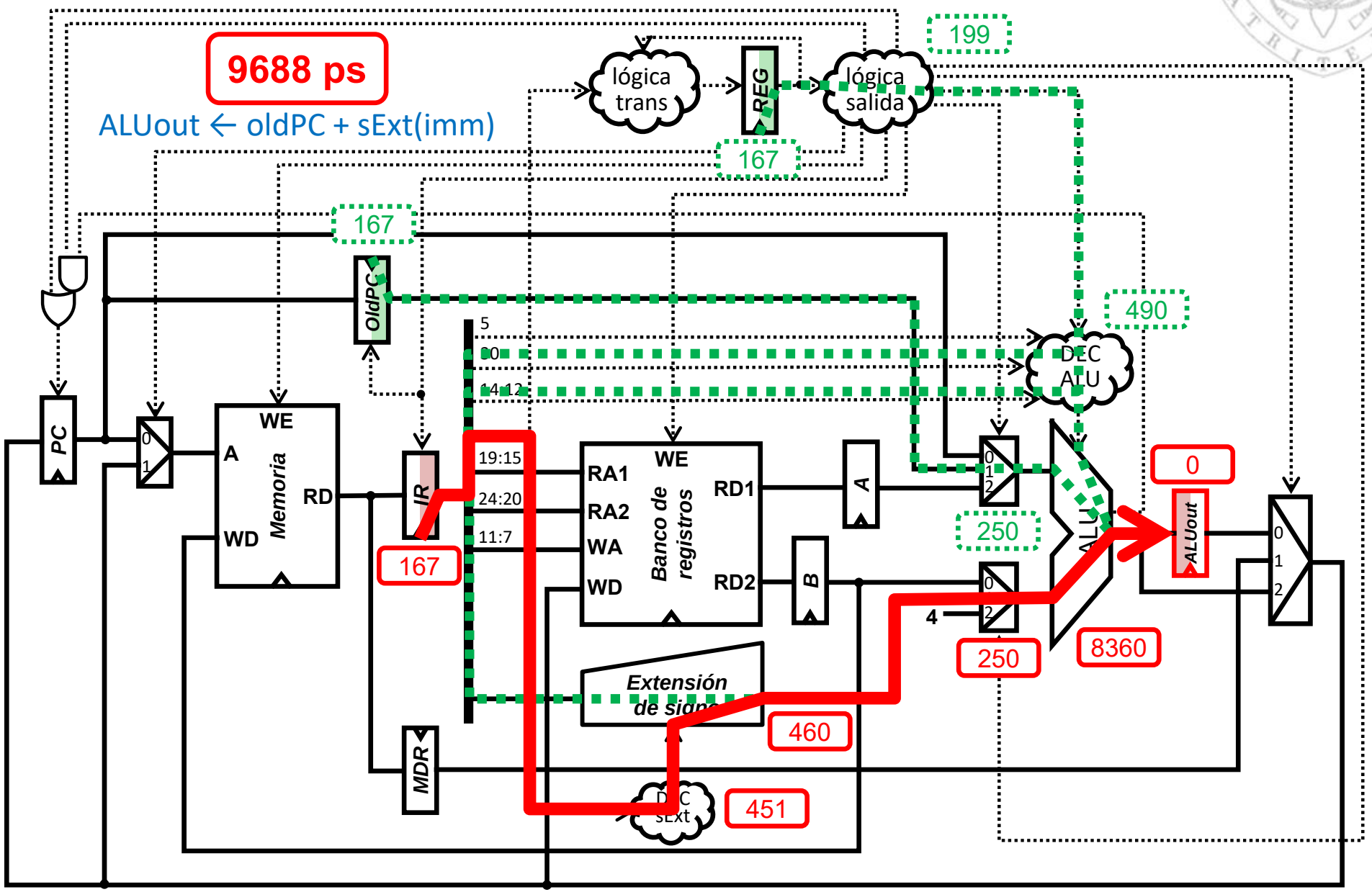
Cálculo del tiempo de ciclo

Estado S1 (búsqueda de operandos): camino crítico



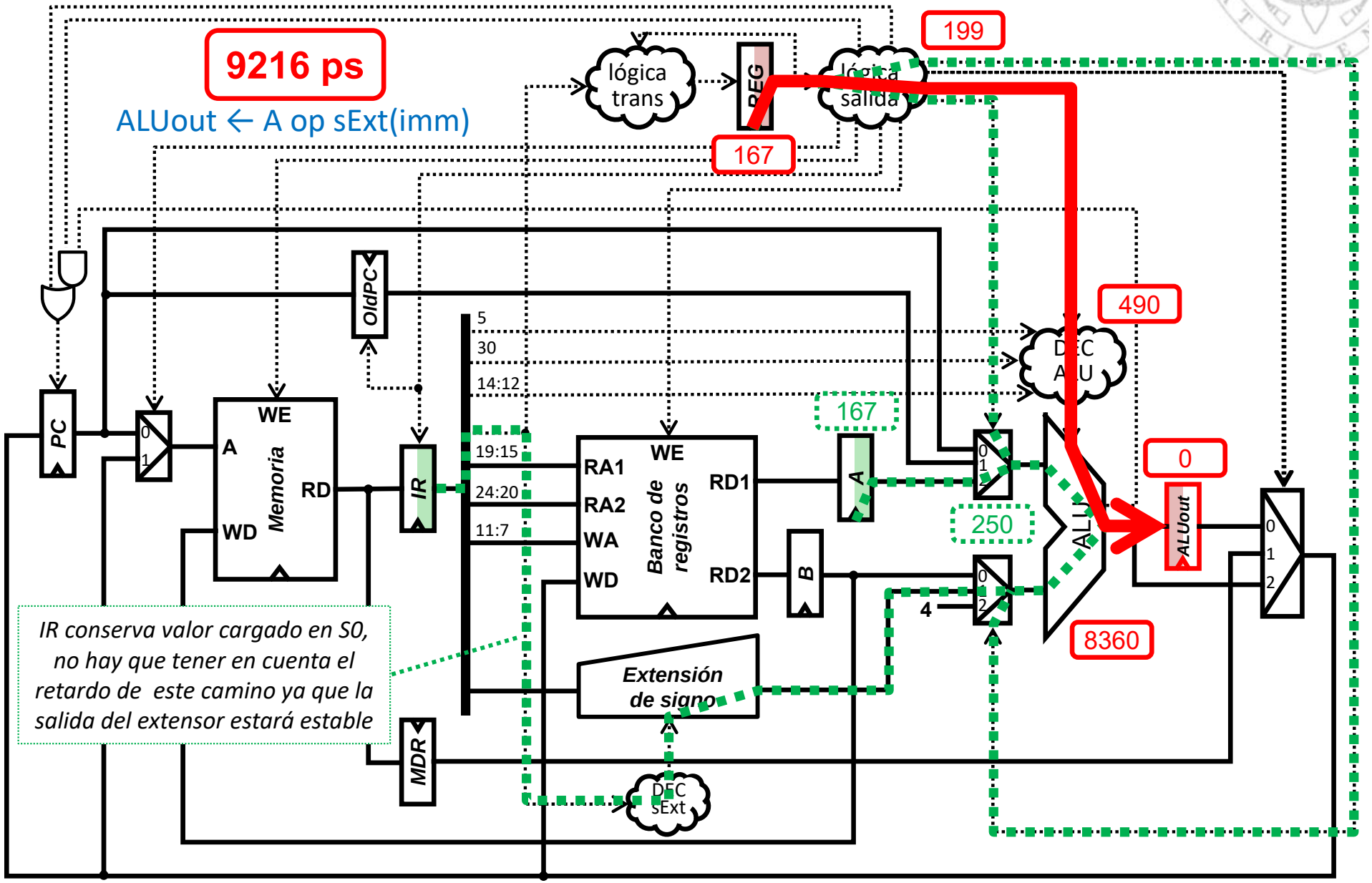
Cálculo del tiempo de ciclo

Estado S1 (cálculo de la dirección de salto): camino crítico



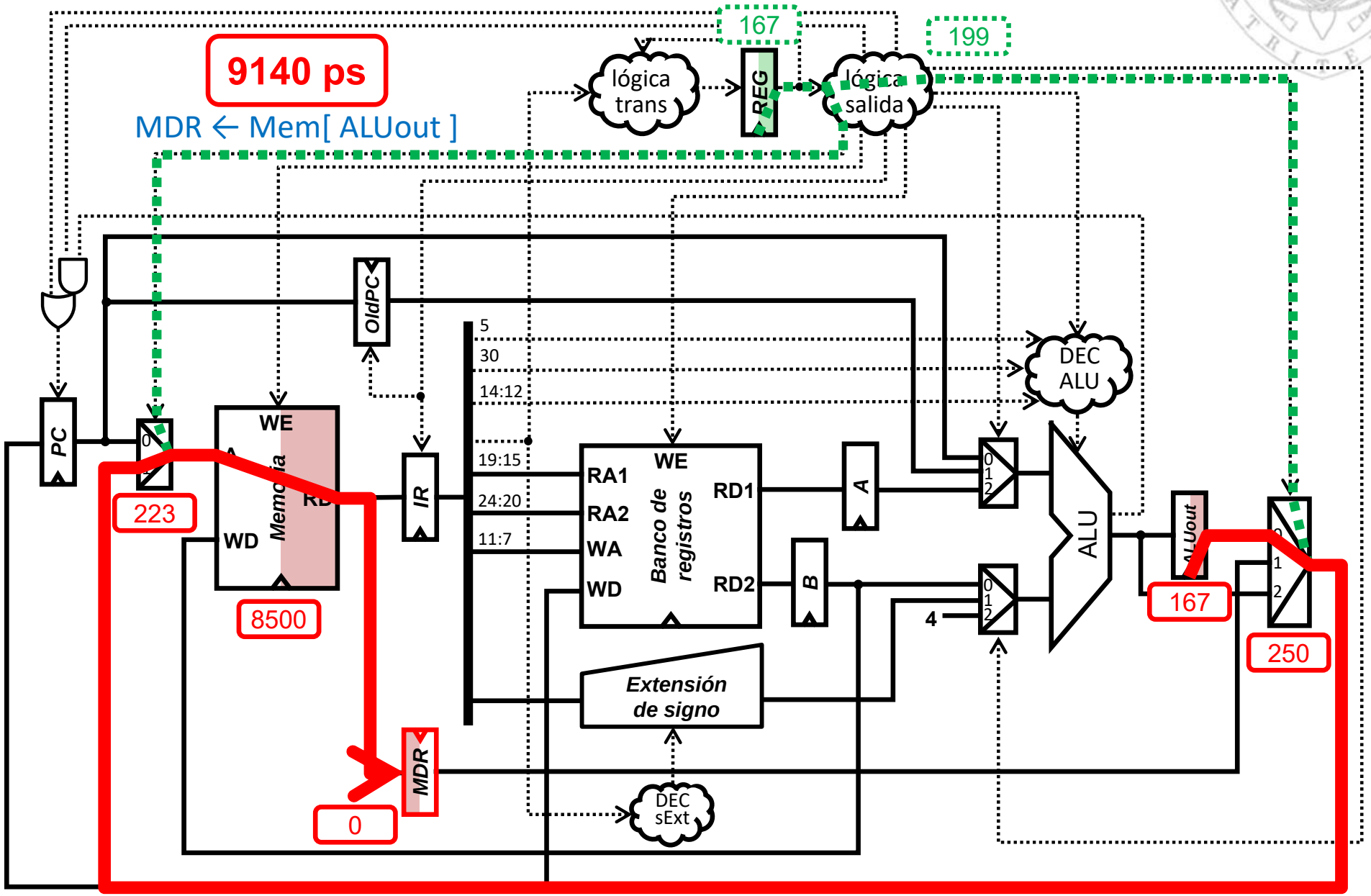
Cálculo del tiempo de ciclo

Estado S2 / S6 / S8 (cálculo en ALU): camino crítico



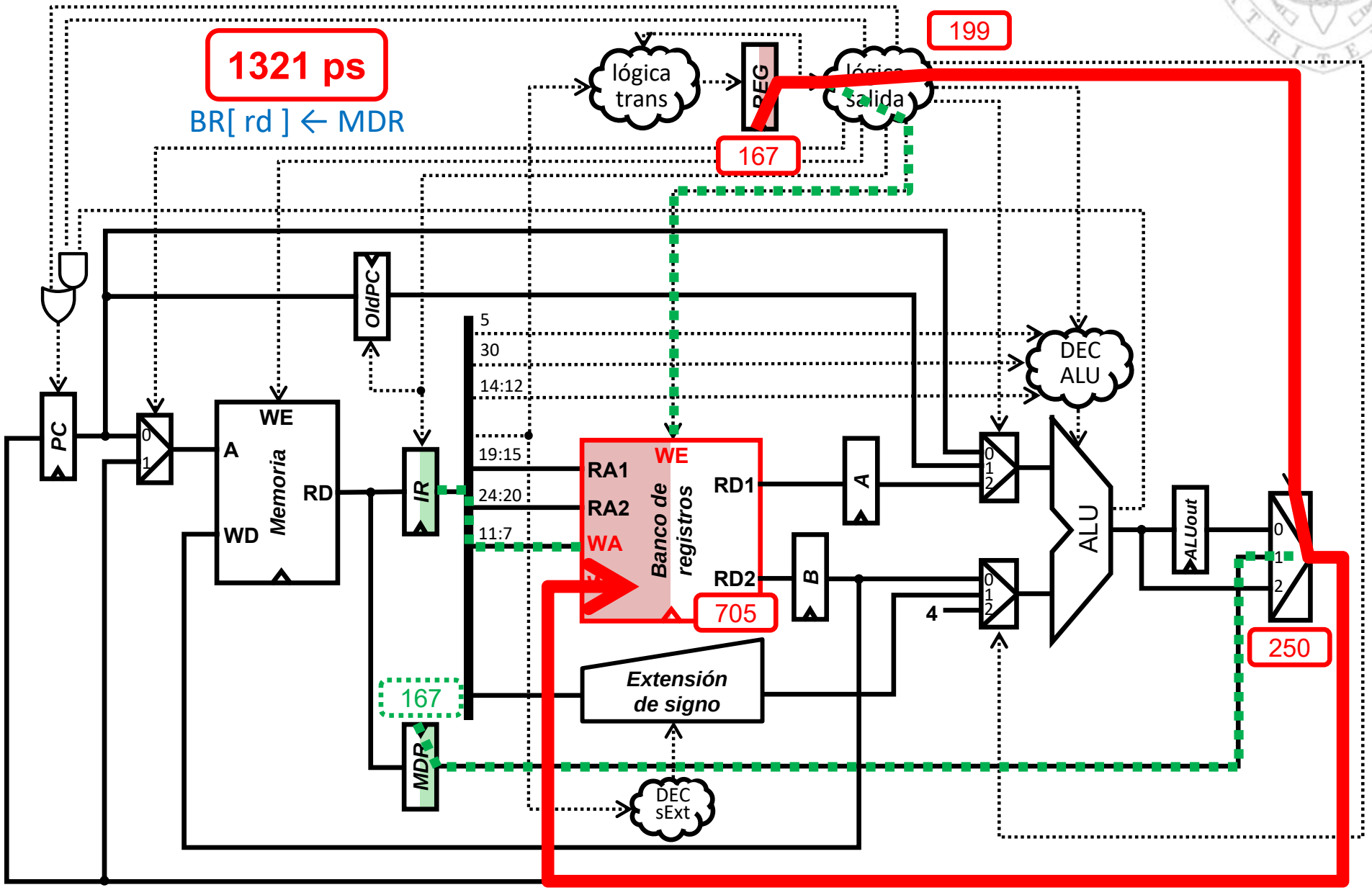
Cálculo del tiempo de ciclo

Estado S3 (lectura de memoria): camino crítico



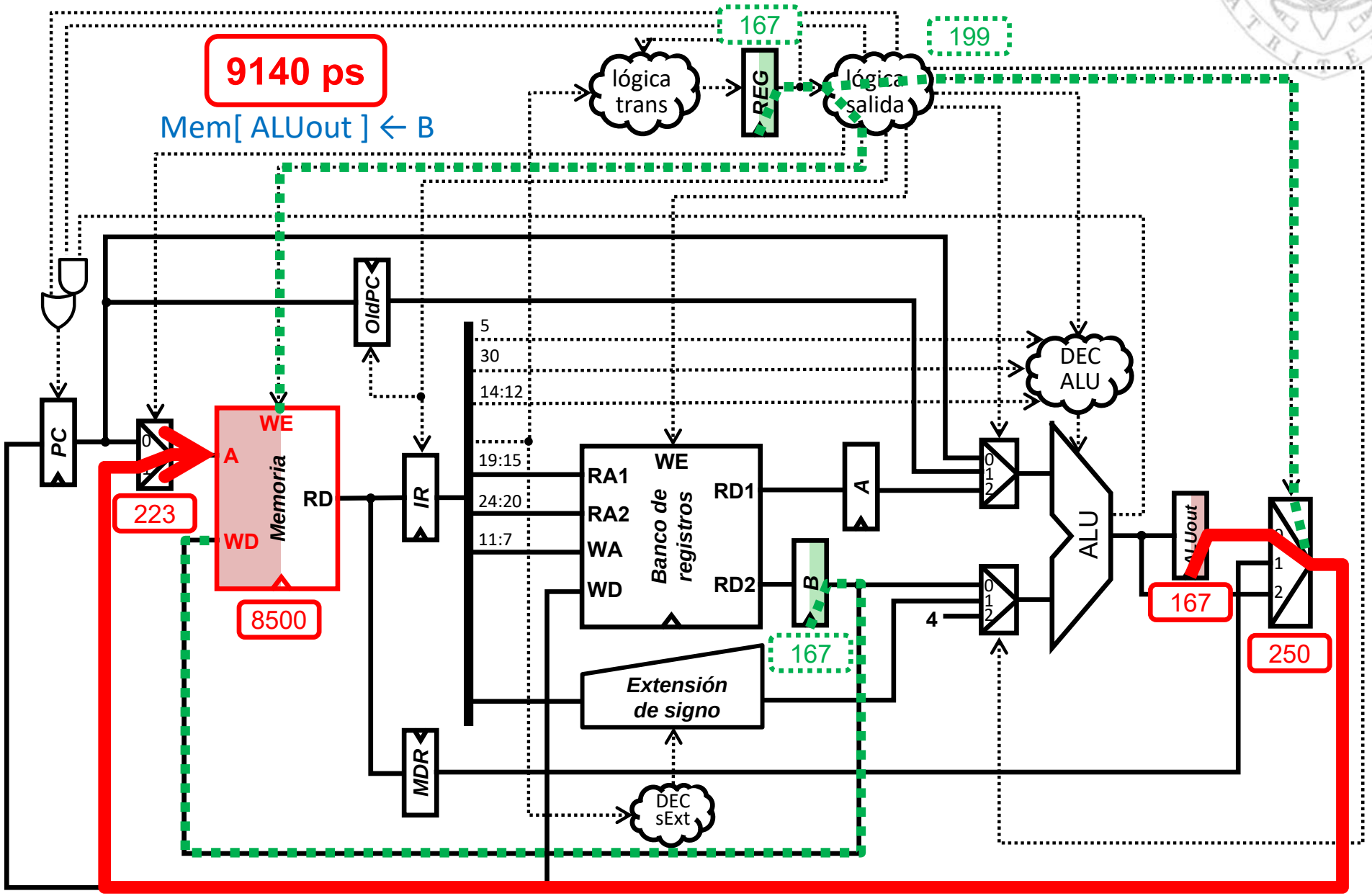
Cálculo del tiempo de ciclo

Estado S4 (escritura en el BR): camino crítico



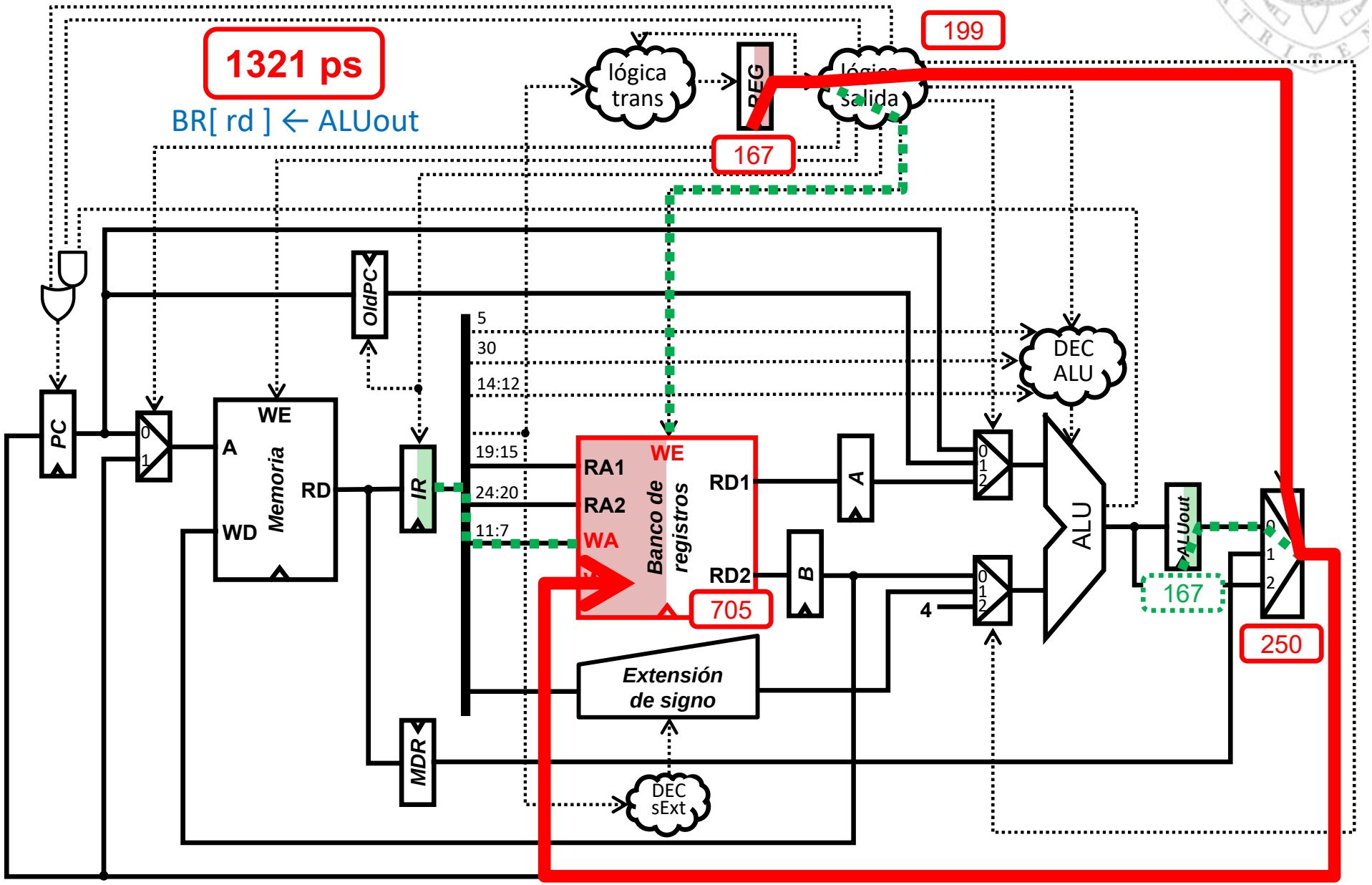
Cálculo del tiempo de ciclo

Estado S5 (escritura en memoria): camino crítico



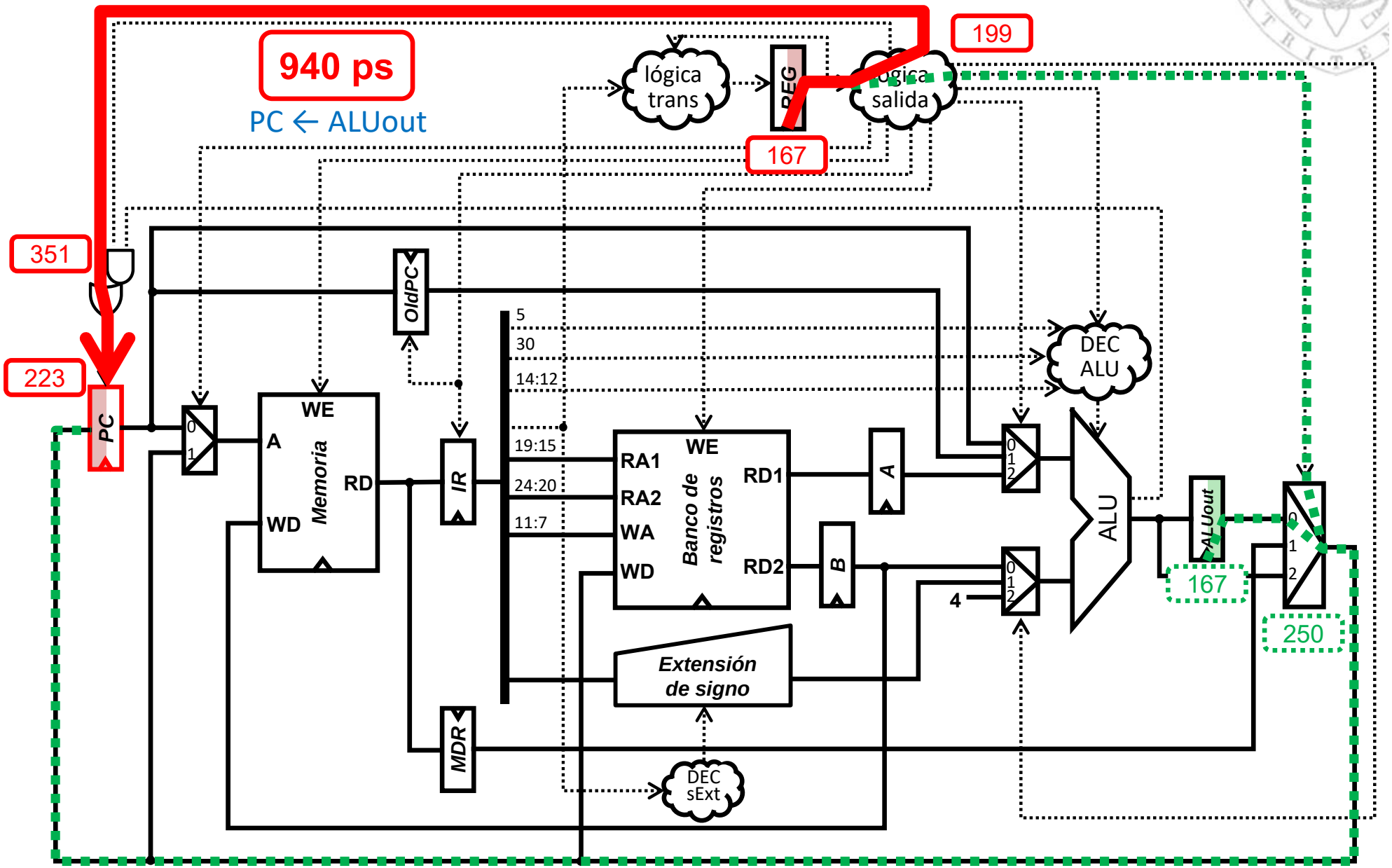
Cálculo del tiempo de ciclo

Estado S7 (escritura en el BR): camino crítico



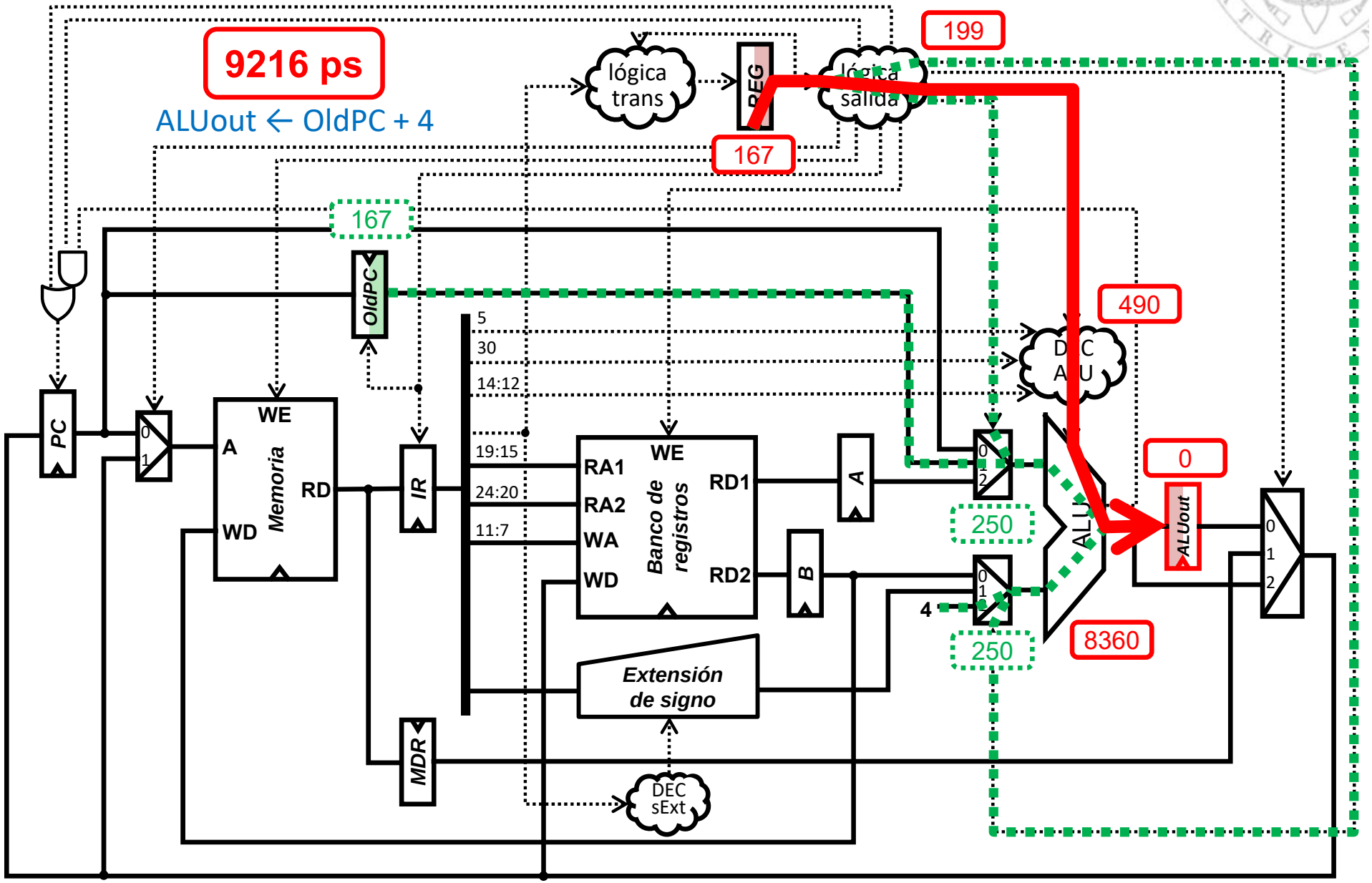
Aspectos tecnológicos

Estado S9 (actualización del PC): camino crítico



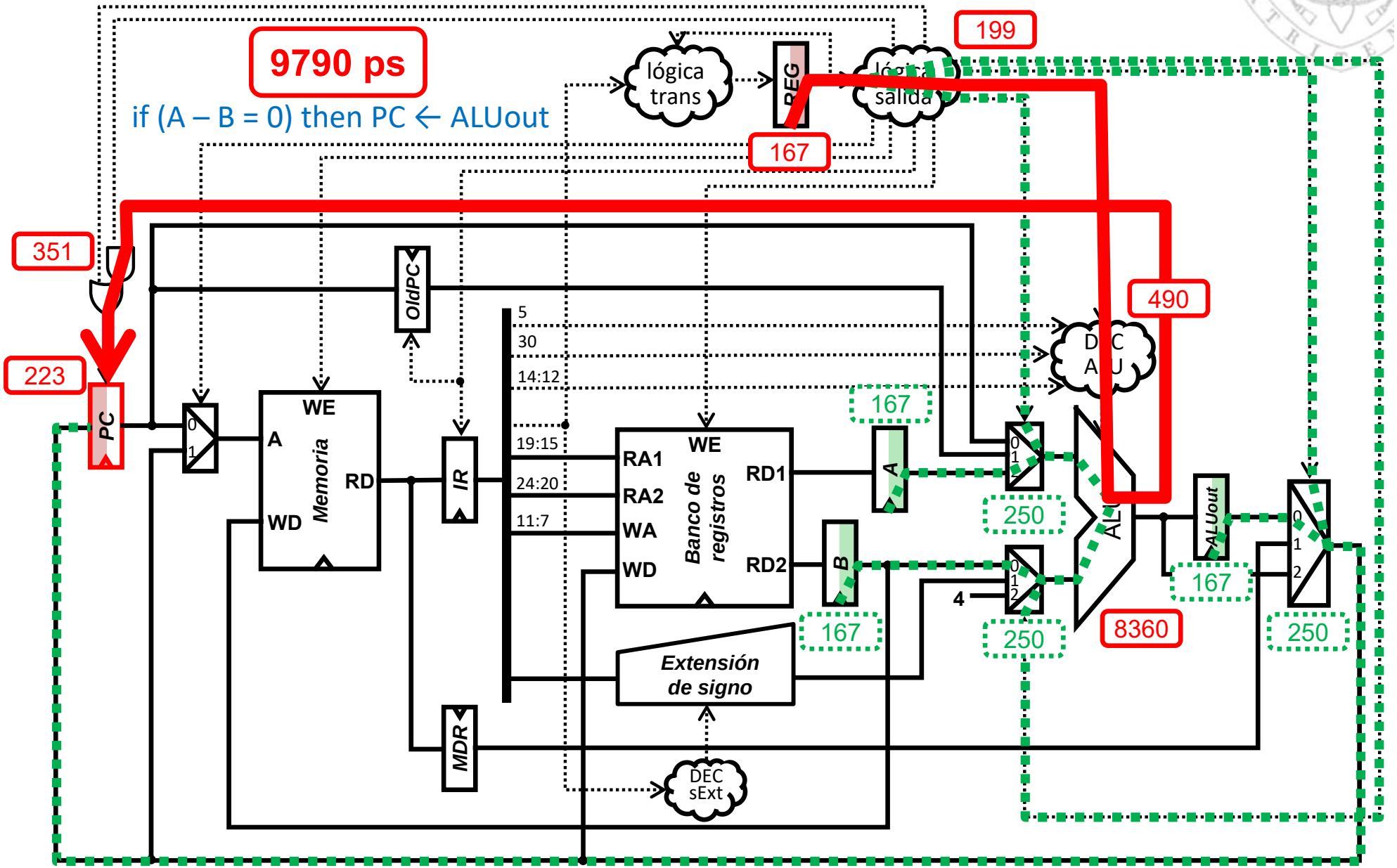
Cálculo del tiempo de ciclo

Estado S9 (cálculo dir. retorno): camino crítico



Cálculo del tiempo de ciclo

Estado S10: camino crítico



Acerca de *Creative Commons*

- Licencia CC (Creative Commons)
  - Ofrece algunos derechos a terceras personas bajo ciertas condiciones. Este documento tiene establecidas las siguientes:

Reconocimiento (*Attribution*):
En cualquier explotación de la obra autorizada por la licencia hará falta reconocer la autoría.

No comercial (*Non commercial*):
La explotación de la obra queda limitada a usos no comerciales.

Compartir igual (*Share alike*):
La explotación autorizada incluye la creación de obras derivadas siempre que mantengan la misma licencia al ser divulgadas.

Más información: https://creativecommons.org/licenses/by-nc-sa/4.0/