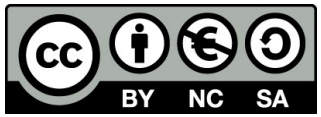
Tema 5:

Diseño monociclo del procesador

Fundamentos de computadores II

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



Contenidos

- ✓ Introducción.
- ✓ RISC-V de arquitectura reducida.
- ✓ Diseño de la ruta de datos.
- ✓ Diseño del controlador.

- ✓ 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 método más fiable para conocer el rendimiento de un computador es midiendo el tiempo que tarda en ejecutar programas.
  - El computador que los ejecute más rápido tendrá mayor rendimiento.

- Para una arquitectura dada, el tiempo de ejecución de un programa depende principalmente del:
  - Número de instrucciones que tenga el programa. ← depende del programador
  - Número de ciclos que tarde cada instrucción. ← dependen del diseño HW
  - Tiempo de ciclo (frecuencia de reloj). ← dependen del diseño HW

- Comúnmente número de ciclos y tiempo de ciclo tienen son inversos:
  - Procesador monociclo: 1 ciclo por instrucción, tiempo de ciclo largo.
  - Procesador multiciclo: Varios ciclos por instrucción, tiempo de ciclo corto.

Introducción

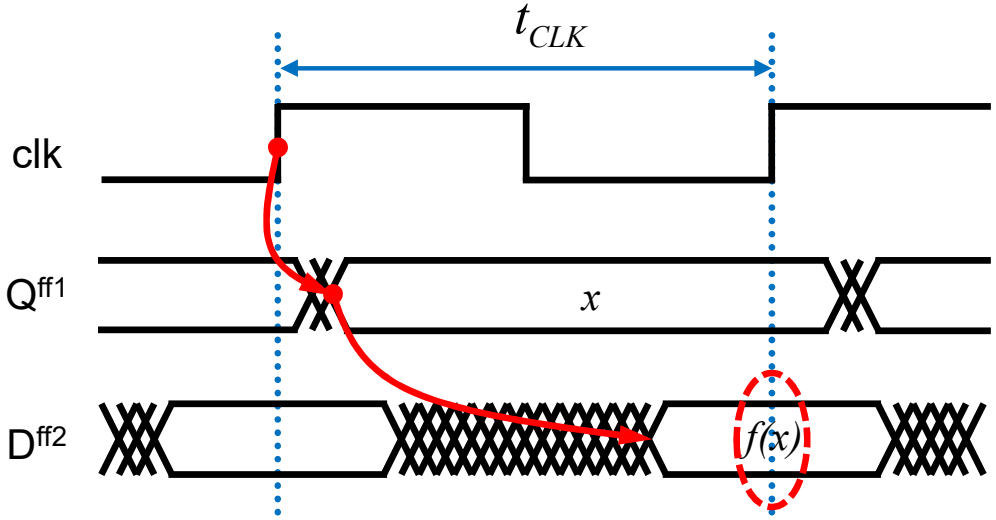
- Para diseñar un procesador se usan técnicas de diseño algorítmico en donde la especificación del circuito es su arquitectura.

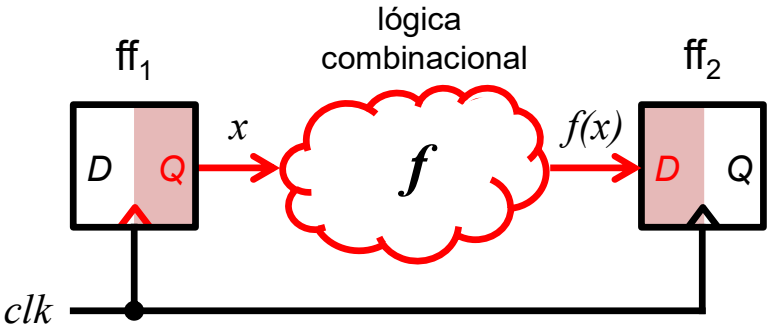
- El procesador estará formado por 2 elementos:
  - Ruta de datos: realiza operaciones y almacena resultados.
    - Al menos, deberá incluir tantos elementos de almacenamiento como estén definidos en la arquitectura (sean visibles por el programador).
    - Deberá incluir los elementos funcionales que sean necesarios para realizar todas las operaciones del repertorio de instrucciones.
  - Controlador: secuencia la realización de las transferencias entre registros definidas para cada instrucción del repertorio.

- Según la estrategia de diseño elegida tendremos:
  - Procesadores monociclo: todas las transferencias de entre registros implicadas en una instrucción se realizan en un único ciclo de reloj.
  - Procesadores multiciclo: las transferencias entre registros implicadas en una instrucción se reparten entre varios ciclos de reloj consecutivos.

Introducción

- Los procesadores se diseñarán según el modelo de temporización síncrona por flanco (de subida) de reloj global:
  - El reloj llega a todos los biestables del sistema y todos ellos cambian de estado simultáneamente en el flanco (de subida) del reloj.
  - Los nuevos valores se propagan a través de las redes combinacionales hasta estabilizarse en las entradas de los biestables.
  - Se repite el proceso indefinidamente en cada ciclo de reloj.
  - El tiempo de ciclo del reloj debe ser lo suficientemente largo para que todos los sistemas combinacionales alcancen su régimen permanente.





RISC-V de arquitectura reducida

Repertorio de instrucciones (i)

- Se diseñará una microarquitectura capaz de ejecutar un subconjunto del repertorio de instrucciones del RISCV32 que opere únicamente con datos de 32 bits.

- Instrucciones con acceso a memoria

lw ***rd, imm$_{12b}$(rs1)***	rd ← Mem[rs1 + sExt(imm)]	tipo-I
sw ***rs2, imm$_{12b}$(rs1)***	Mem[rs1 + sExt(imm$_{12b}$)] ← rs2	tipo-S

- Aritmético-lógicas con ambos operandos en registros

add ***rd, rs1, rs2***	rd ← rs1 + rs2	tipo-R
sub ***rd, rs1, rs2***	rd ← rs1 – rs2	tipo-R
and ***rd, rs1, rs2***	rd ← rs1 & rs2	tipo-R
or ***rd, rs1, rs2***	rd ← rs1 | rs2	tipo-R
slt ***rd, rs1, rs2***	rd ← *if* (rs1 <$_S$ rs2) *then* (1) *else* (0)	tipo-R

RISC-V de arquitectura reducida

Repertorio de instrucciones (ii)

- Aritmético-lógicas con un operando inmediato

`addi` ***rd, rs1,*** imm_{12b}	rd ← rs1 + sExt(imm),	tipo-I
`andi` ***rd, rs1,*** imm_{12b}	rd ← rs1 & sExt(imm)	tipo-I
`ori` ***rd, rs1,*** imm_{12b}	rd ← rs1 | sExt(imm)	tipo-I
`slti` ***rd, rs1,*** imm_{12b}	rd ← *if* (rs1 $<_S$ sExt(imm)) *then* (1) *else* (0)	tipo-I

- Instrucciones de salto condicional

`beq` ***rs1, rs2,*** imm_{13b}	PC ← *if* (rs1 = rs2) *then* (PC + sExt($imm_{12:1}$ << 1)) *else* (PC+4)	tipo-B

- Instrucciones de salto a función

`jal` **rd,** imm_{21b}	PC ← PC + sExt($imm_{20:1}$ << 1), rd ← PC+4	tipo-J

RISC-V de arquitectura reducida

Formato de instrucción

- Los formatos de instrucción y codificación de campos serán los mismos que en la arquitectura completa.

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	op	*tipo-R*
$imm_{11:0}$		rs1	funct3	rd	op	*tipo-I*
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	*tipo-S*
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	*tipo-B*
$imm_{20,10:1,11,19:12}$				rd	op	*tipo-J*

RISC-V de arquitectura reducida

Formato de instrucción: codificación de campos

Instrucción	Tipo	funct7 bits 31:25	funct3 bits 14:12	op bits 6:0
`lw`	I	–	010	0000011
`sw`	S	–	010	0100011
`add`	R	0000000	000	0110011
`sub`	R	0100000	000	
`slt`	R	0000000	010	
`or`	R	0000000	110	
`and`	R	0000000	111	
`addi`	I	–	000	0010011
`slti`	I	–	010	
`ori`	I	–	110	
`andi`	I	–	111	
`beq`	B	–	000	1100011
`jal`	J	–	–	1101111

Diseño de la ruta de datos

Elementos de almacenamiento (i)

- Los elementos de almacenamiento serán los mismos que son visibles al programador: Memoria, PC y 32 registros de propósito general.

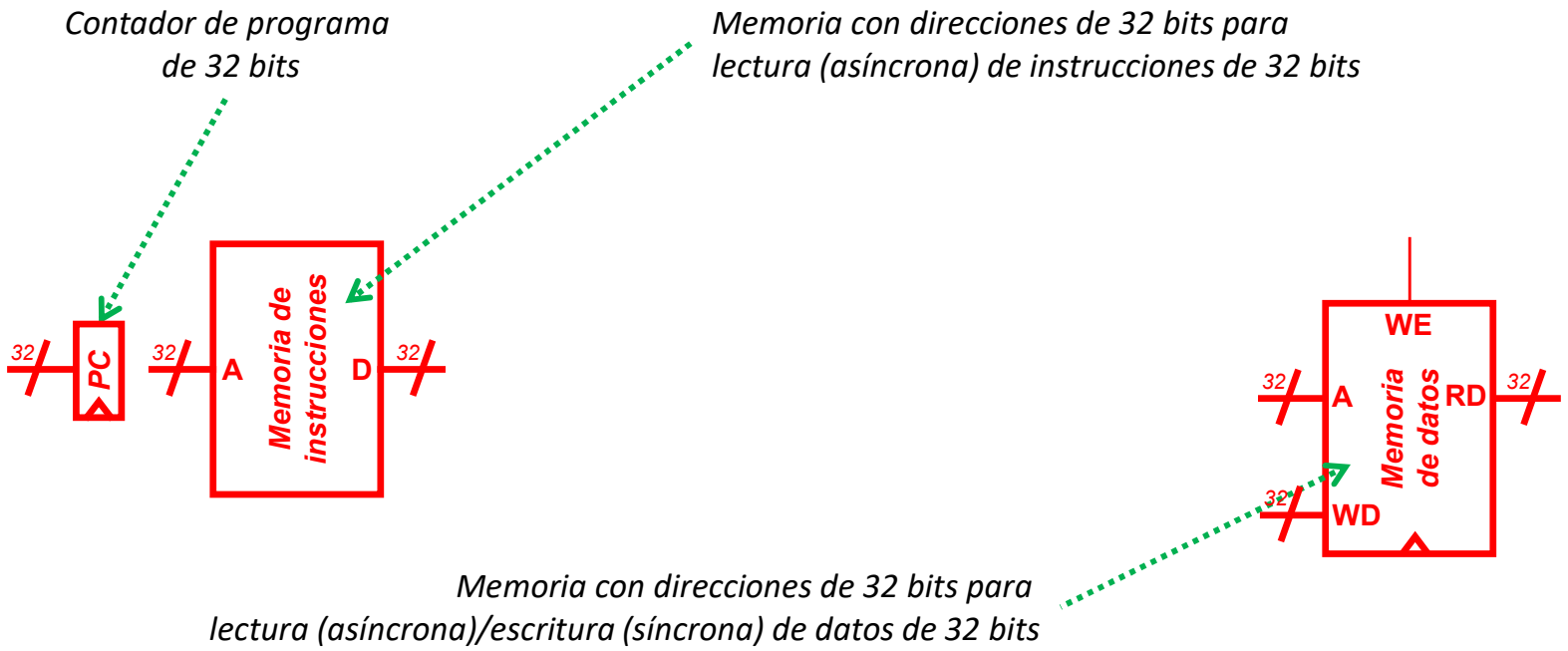
- La Memoria tendrá un comportamiento idealizado:
  - Integrada en el procesador.
  - Direccionable por bytes, pero capaz de aceptar/ofrecer 4 bytes por acceso.
  - Tiempo de acceso inferior al tiempo de ciclo del procesador.
  - Dividida en dos porque las instrucciones deben leerse de memoria en el mismo ciclo de reloj en que se leen/escriben los datos.
    - Memoria de instrucciones: se comportará como una ROM combinacional.
    - Memoria de datos: se comportará como un gran Banco de Registros con doble puerto de datos para entrada y salida separada de los mismos.

- El Contador de Programa será un array de biestables D:
  - Al ejecutarse las instrucciones en un único ciclo, el PC debe actualizarse en todos los ciclos y no requiere tener una señal que controle su carga.

Diseño de la ruta de datos

Elementos de almacenamiento (ii)

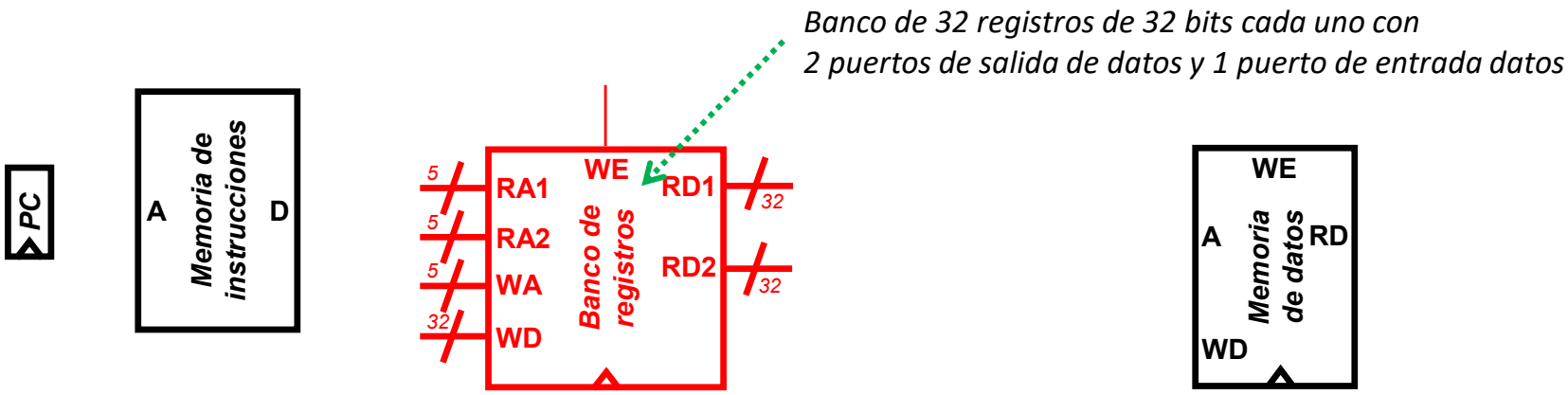
- Los elementos de almacenamiento serán los mismos que son visibles al programador: Memoria, PC y 32 registros de propósito general.



Diseño de la ruta de datos

Elementos de almacenamiento (iii)

- Los elementos de almacenamiento serán los mismos que son visibles al programador: Memoria, PC y 32 registros de propósito general.



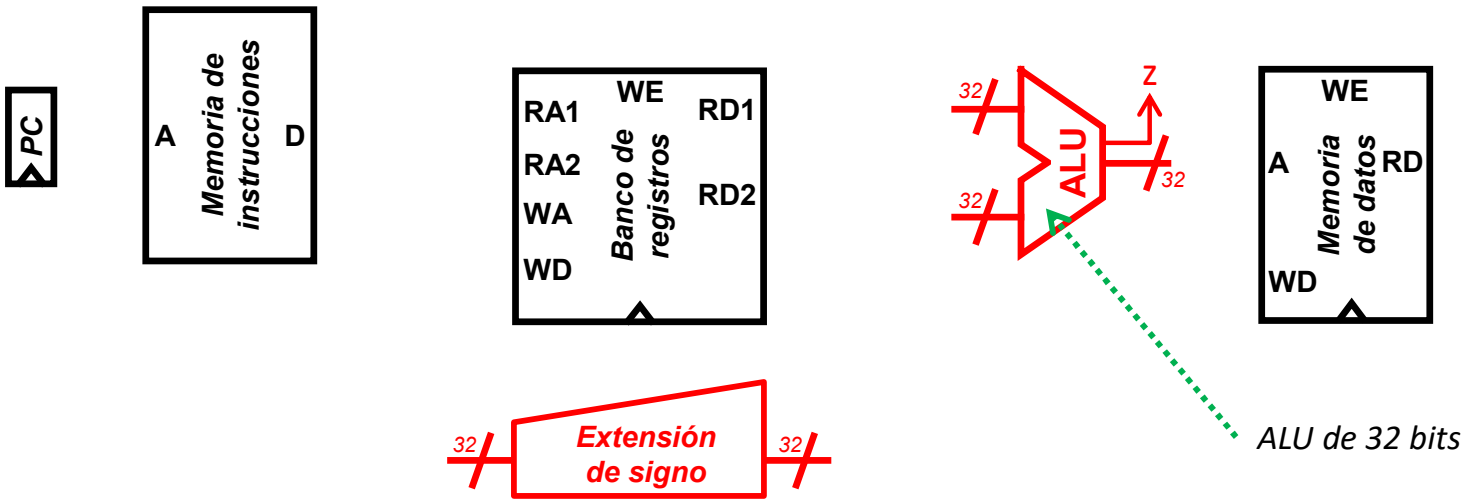
- Los 32 registros se organizan en un Banco con 3 puertos:
  - En un procesador monociclo, las instrucciones tipo-R en el mismo ciclo de reloj leen del Banco 2 registros y escriben 1 registro.

```
add rd, rs1, rs2
```
BR[rd] ← BR[rs1] + BR[rs2], PC ← PC+4

Diseño de la ruta de datos

Elementos funcionales (i)

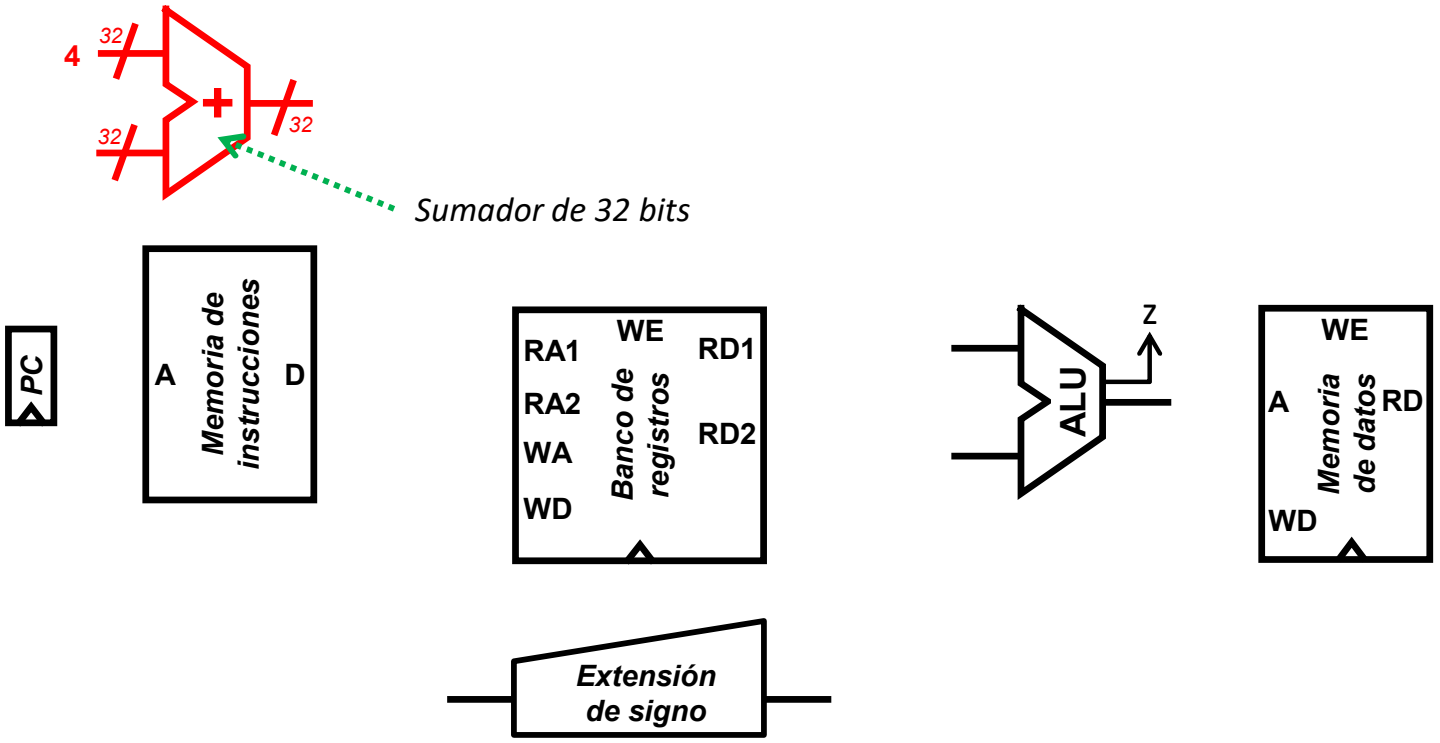
- Dispondrá de una ALU y un Extensor de Signo combinacionales:
  - La ALU realizará todas las operaciones aritmético-lógicas del repertorio.
    - Con un flag Z para realizar mediante resta la comparación de igualdad en instrucciones **`beq`**
  - El Extensor de Signo construirá el operando inmediato de 32 bits a partir de los campos imm (de menor anchura) de las instrucciones.



Diseño de la ruta de datos

Elementos funcionales (ii)

- Dispondrá de un sumador adicional para incrementar el PC
  - En un procesador monociclo, en el mismo ciclo de reloj en que opera la ALU se actualiza el PC con la dirección de la instrucción siguiente.

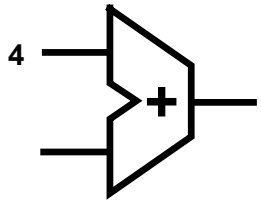


- Siempre suma 4 porque la Memoria es direccionable por bytes y las instrucciones son de 32 bits (4 bytes).

Diseño de la ruta de datos

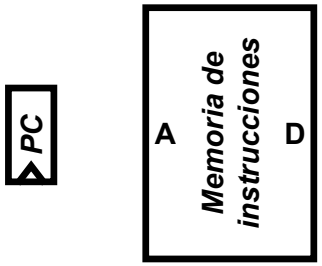
Elementos funcionales (iii)

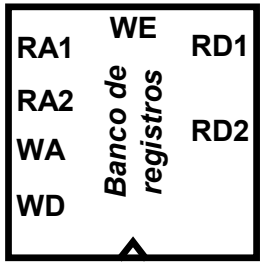
- Dispondrá de otro sumador adicional para calcular direcciones de salto
  - En un procesador monociclo, las instrucciones tipo **`beq`** en mismo ciclo de reloj operan en la ALU, efectúan PC+4 y calculan la dirección de salto.

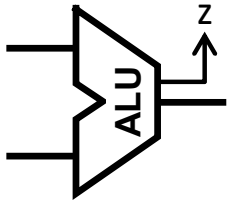


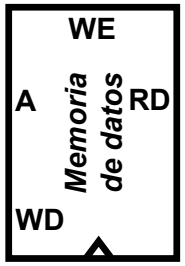
`beq rs1,rs2,imm`

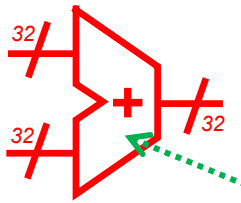
PC ← *if* (BR[rs1] = BR[rs2])
then (PC + sExt(imm)) *else* (PC+4)









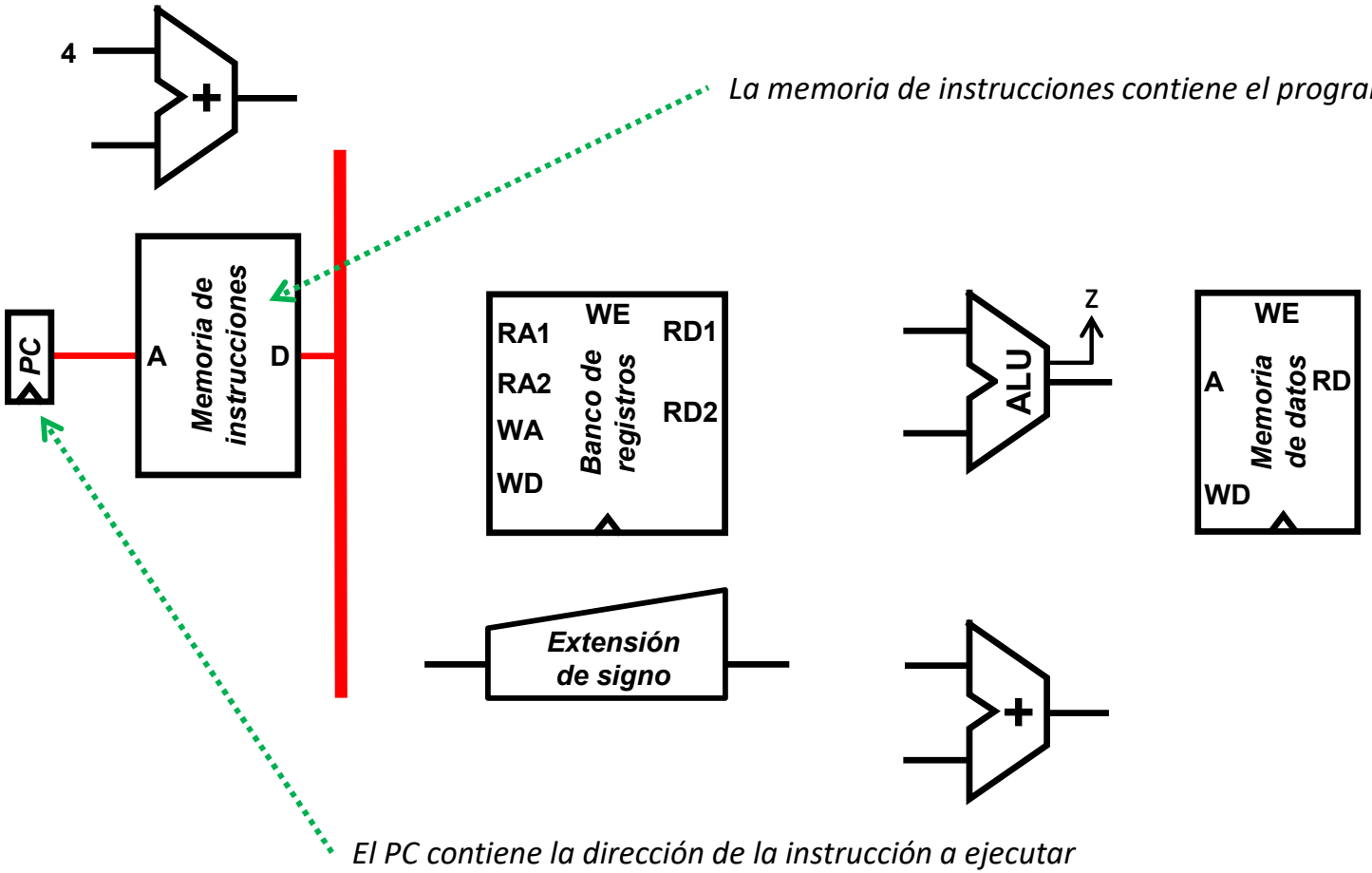


Sumador de 32 bits

Diseño de la ruta de datos

Lectura de la instrucción

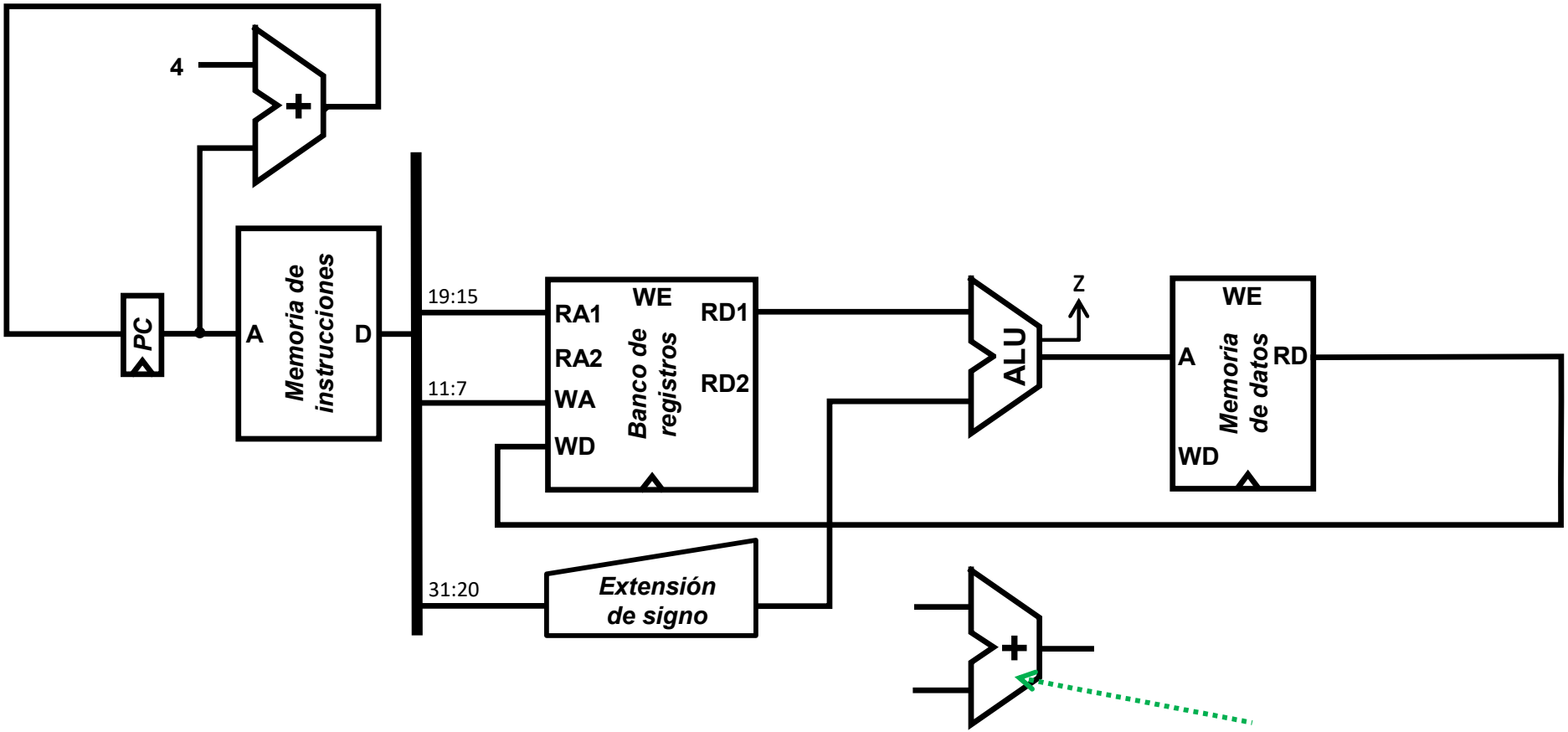
- La ejecución de toda instrucción comienza con su lectura de Memoria.
  - El PC y la Memoria de Instrucciones se conectan para leer la instrucción a ejecutar (aquella cuya dirección está almacenada en el PC).



Diseño de la ruta de datos

Ruta de datos para instrucciones `lw`

- Esta ruta de datos puede ejecutar cualquier secuencia de instrucciones `lw`, se continuará ampliando para poder ejecutar otras.

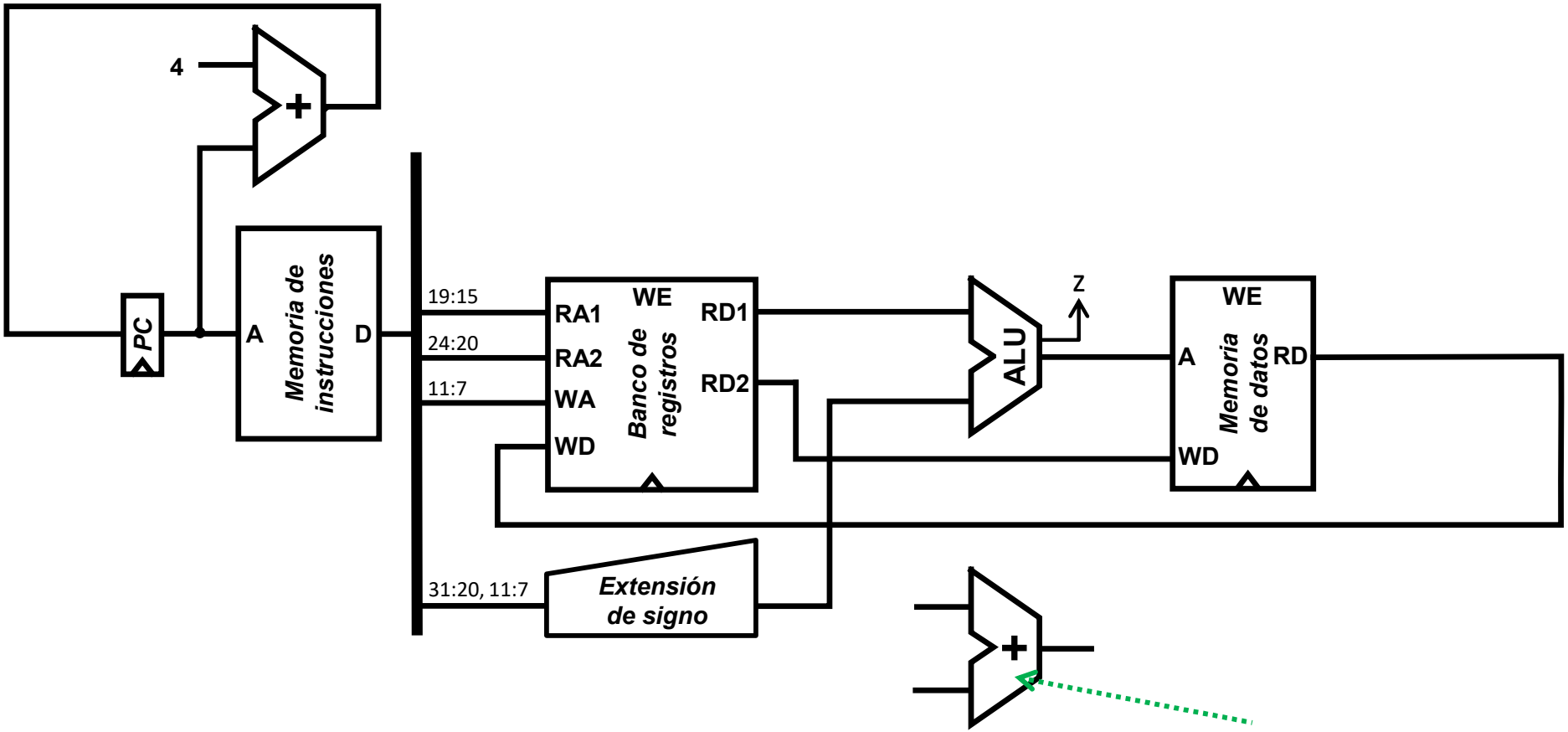


Este sumador solo se usa para calcular direcciones de salta en instrucciones `jal/beq`

Diseño de la ruta de datos

Ruta de datos para instrucciones `lw/sw`

- Esta ruta de datos puede ejecutar cualquier secuencia de instrucciones `lw` y/o `sw`, la seguiremos ampliando.

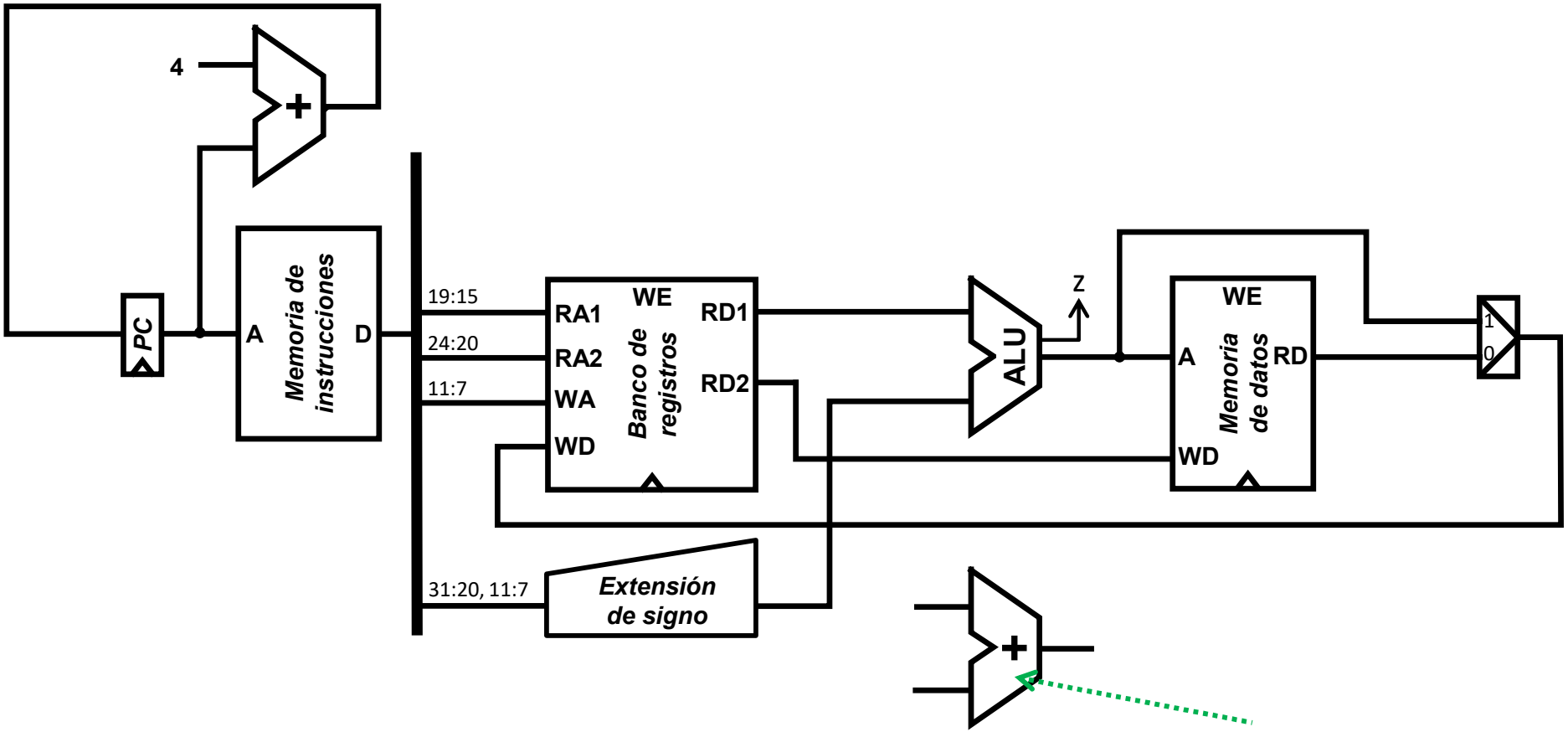


Este sumador solo se usa para calcular direcciones de salta en instrucciones `jal/beq`

Diseño de la ruta de datos

Ruta de datos para instrucciones `lw/sw/addi`

- Esta ruta de datos puede ejecutar cualquier secuencia de instrucciones **`lw`**, **`sw`**, y/o aritmético-lógica con operando inmediato.

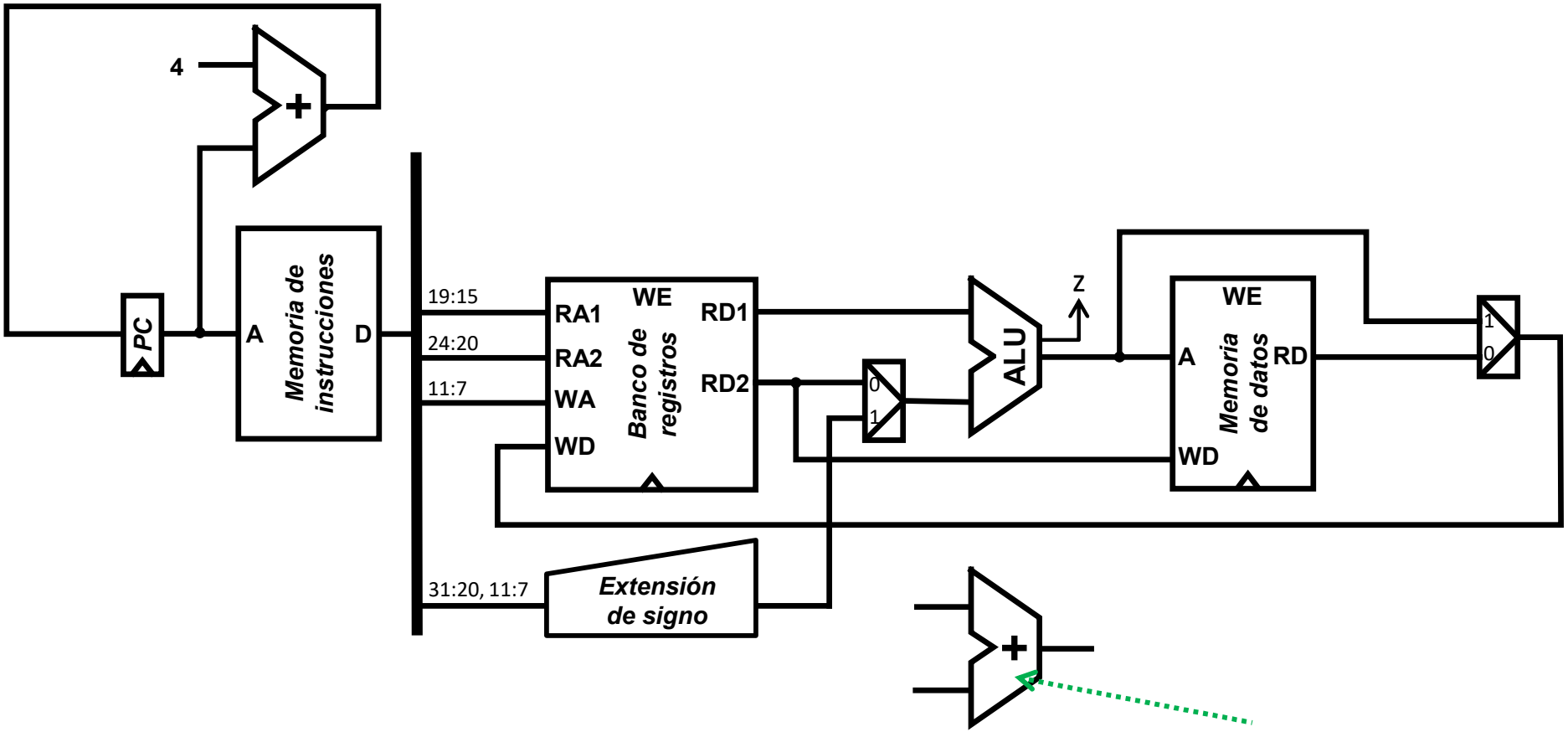


Este sumador solo se usa para calcular direcciones de salta en instrucciones **`jal/beq`**

Diseño de la ruta de datos

Ruta de datos para instrucciones `lw/sw/addi/add`

- Esta ruta de datos puede ejecutar cualquier secuencia de instrucciones **`lw`**, **`sw`**, y/o aritmético-lógica.

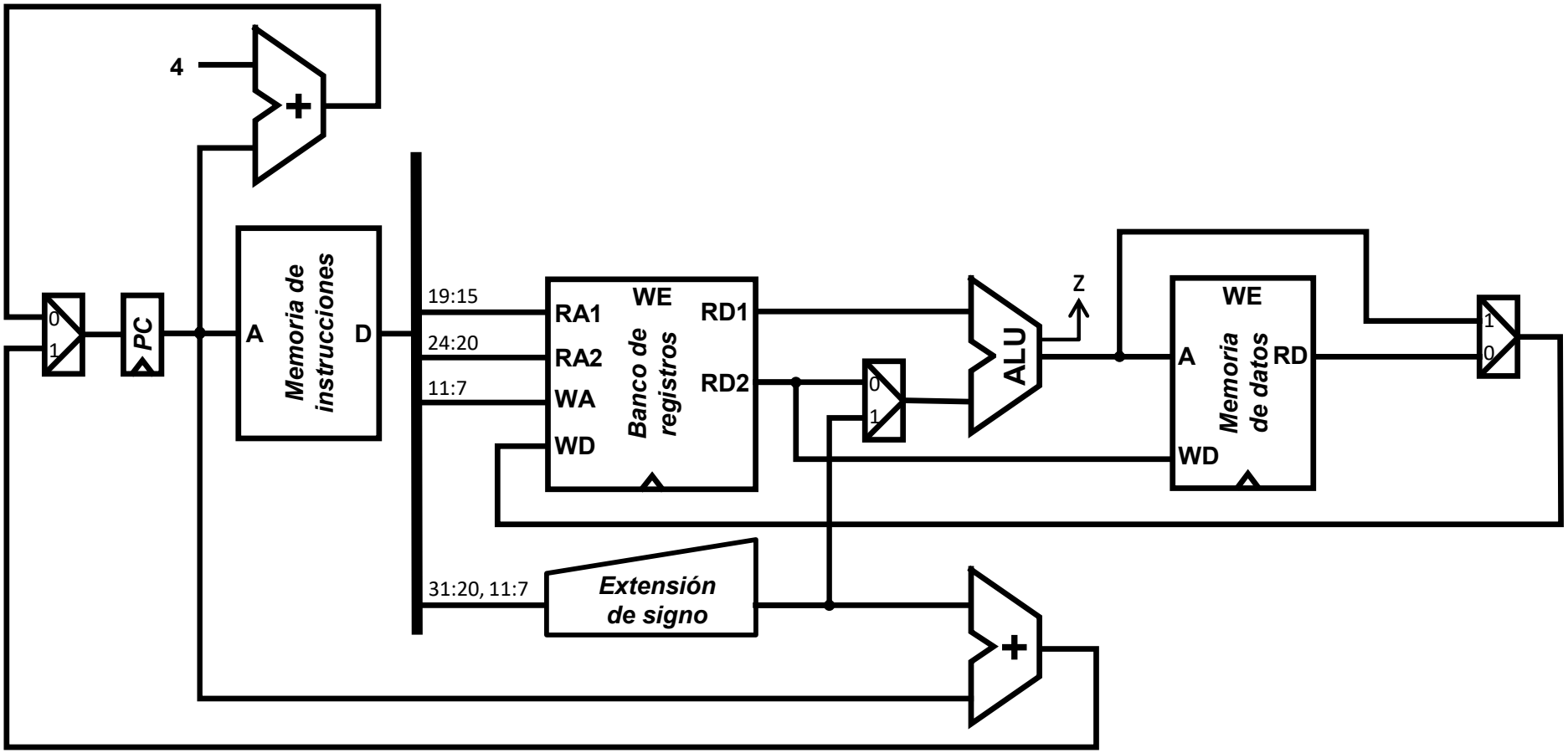


Este sumador solo se usa para calcular direcciones de salta en instrucciones **`jal/beq`**

Diseño de la ruta de datos

Ruta de datos para instrucciones `lw/sw/addi/add/beq`

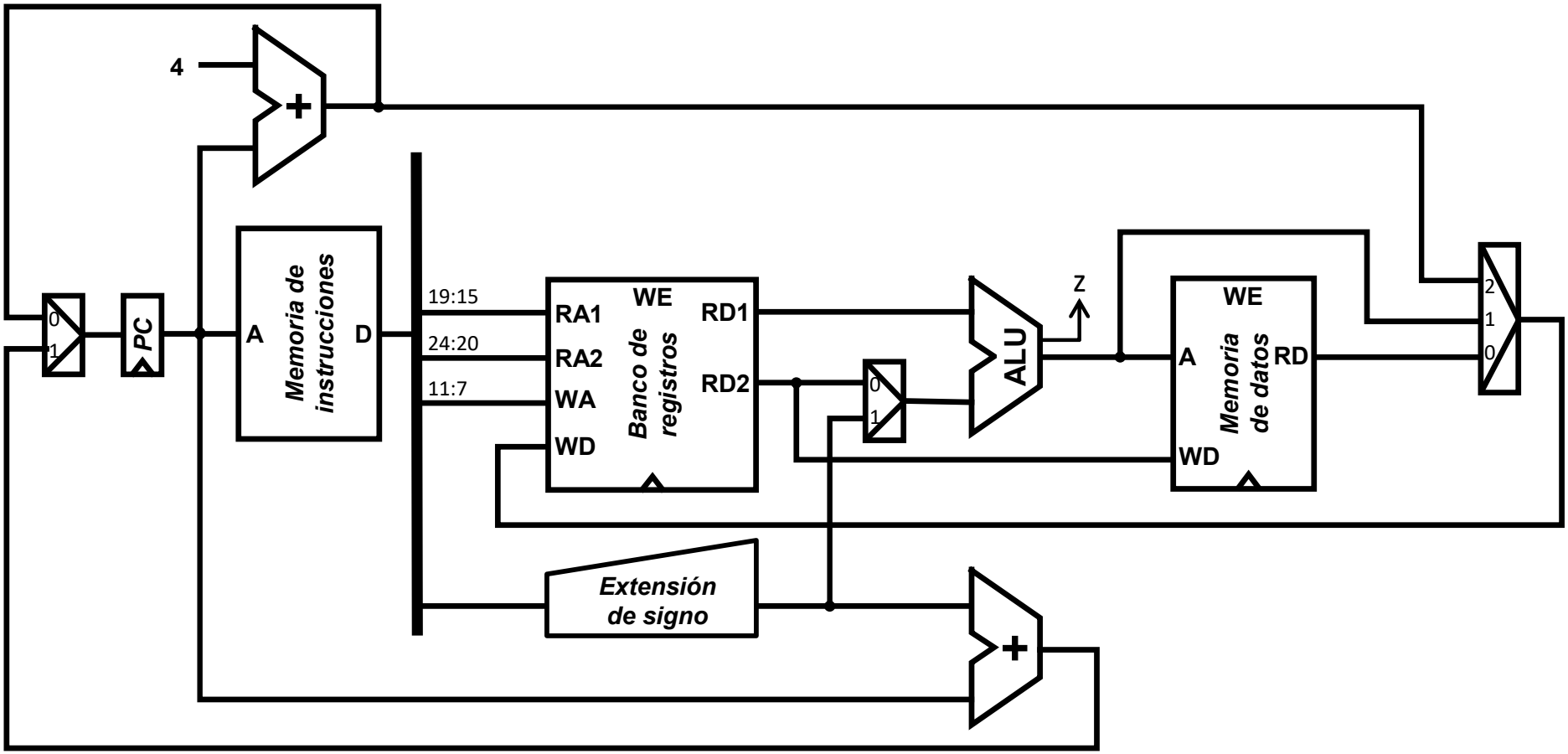
- Esta ruta de datos puede ejecutar cualquier secuencia de instrucciones **`lw`** , **`sw`**, aritmético-lógica. y/o de salto condicional.



Diseño de la ruta de datos

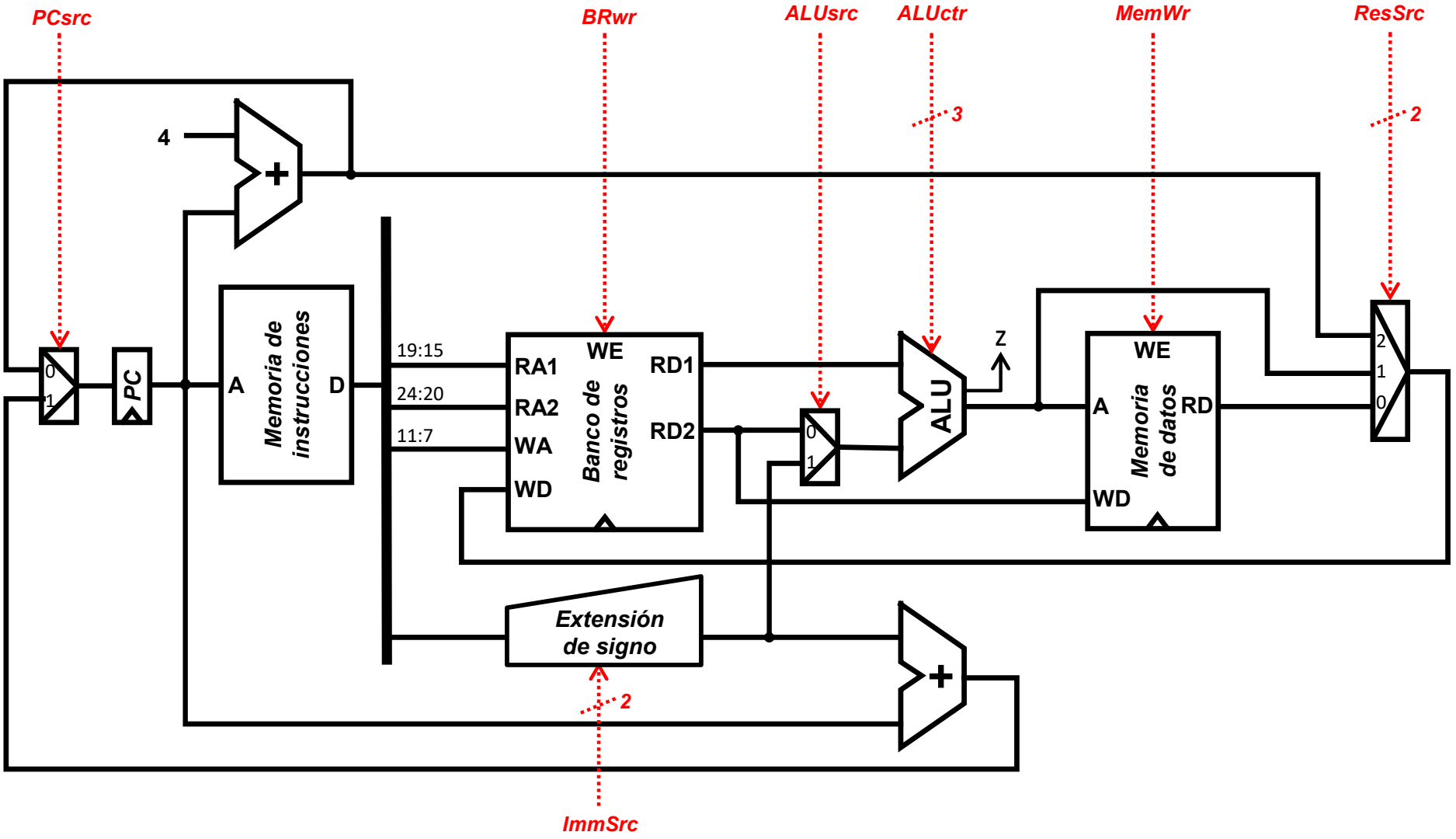
Ruta de datos RISC-V reducido

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



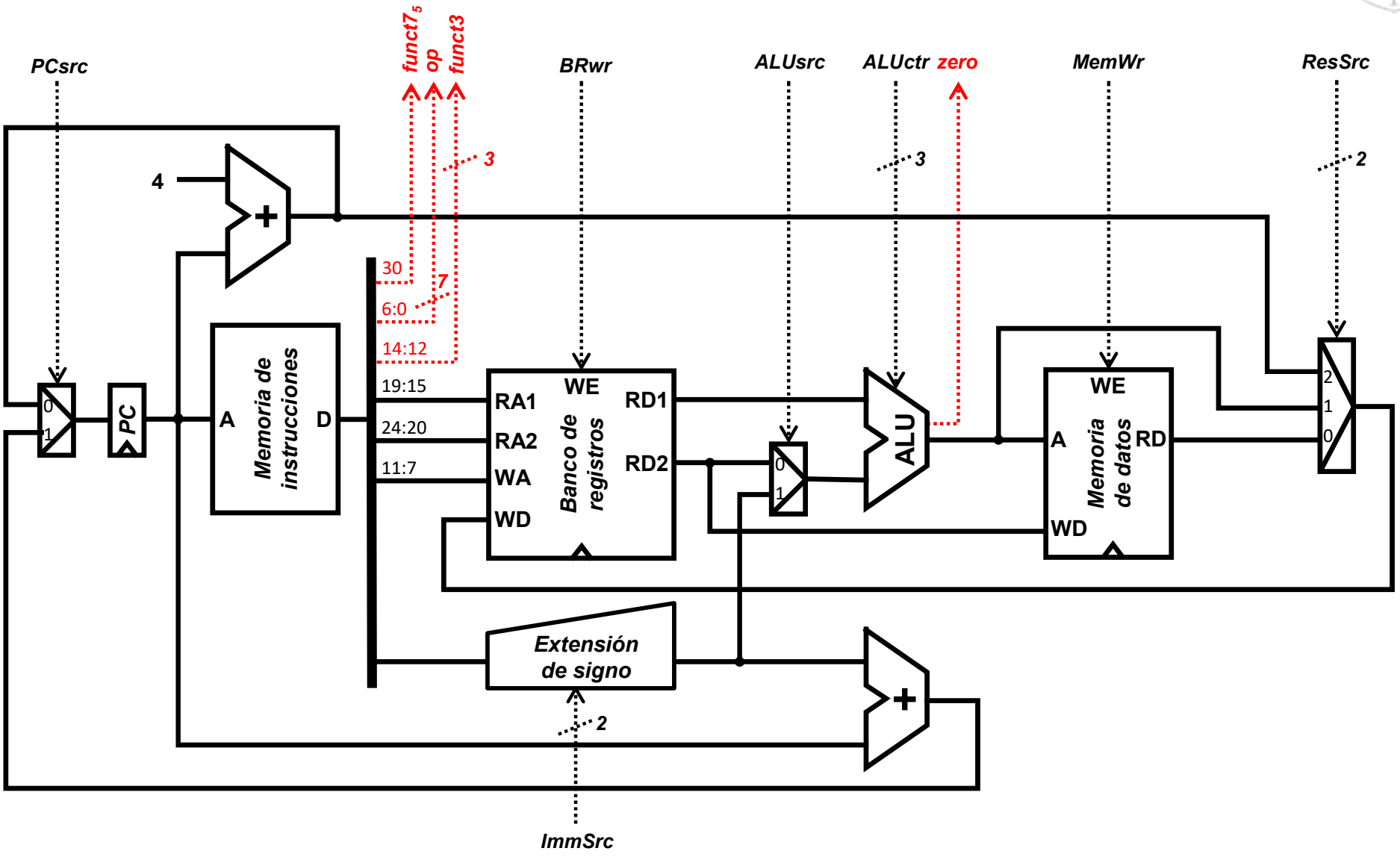
Diseño de la ruta de datos

Señales de control



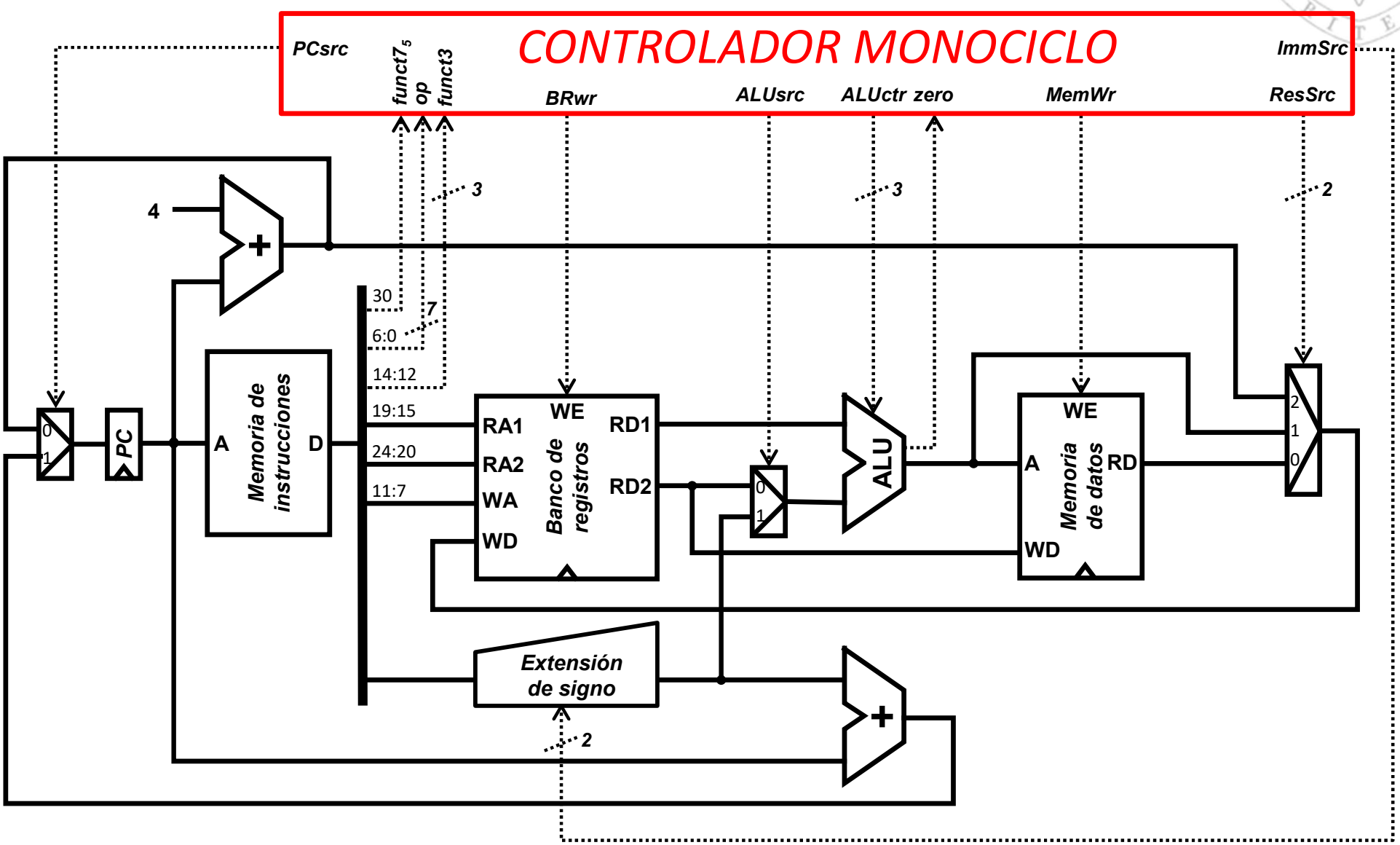
Diseño de la ruta de datos

Señales de estado



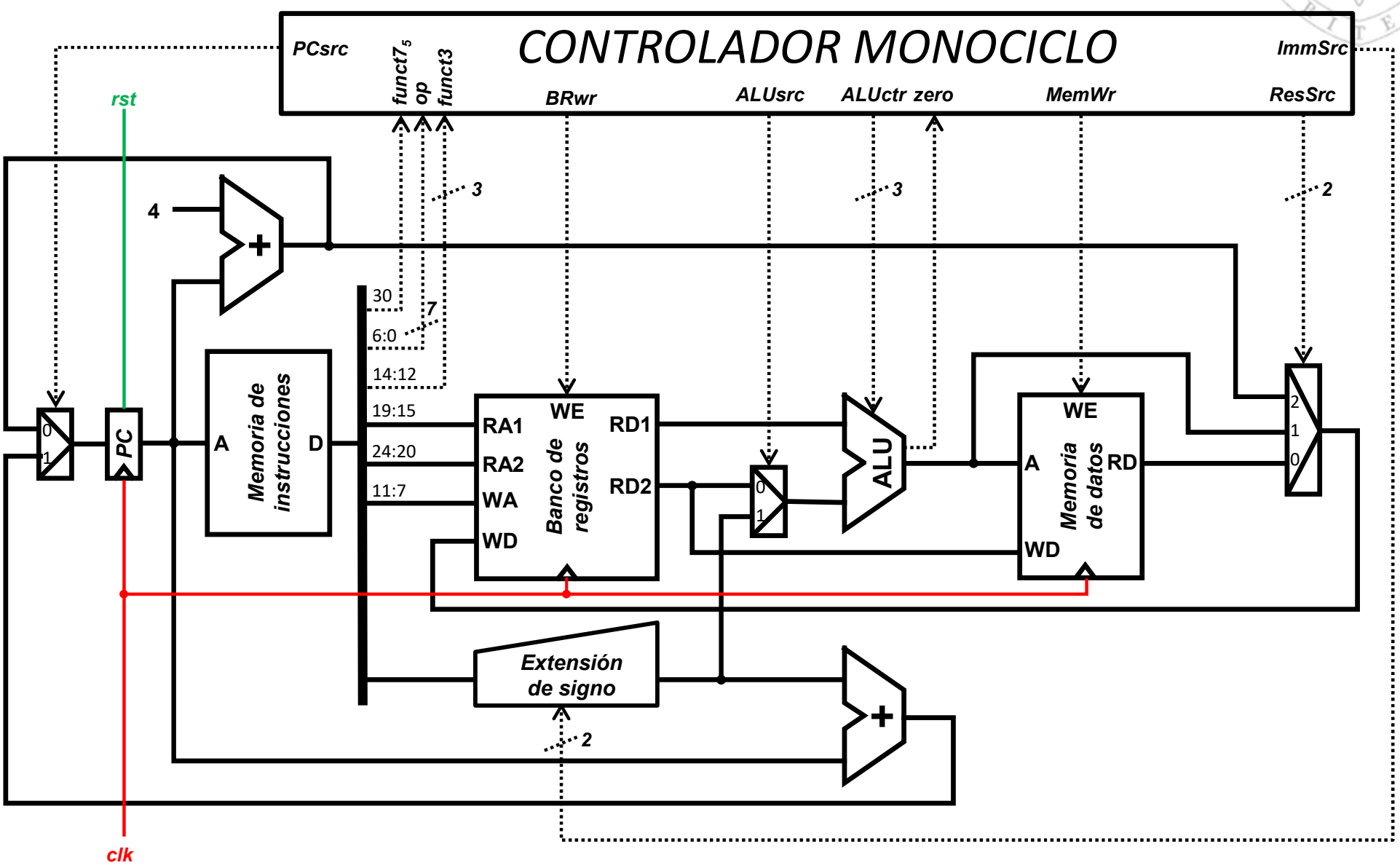
Diseño de la ruta de datos

Conexión con el controlador



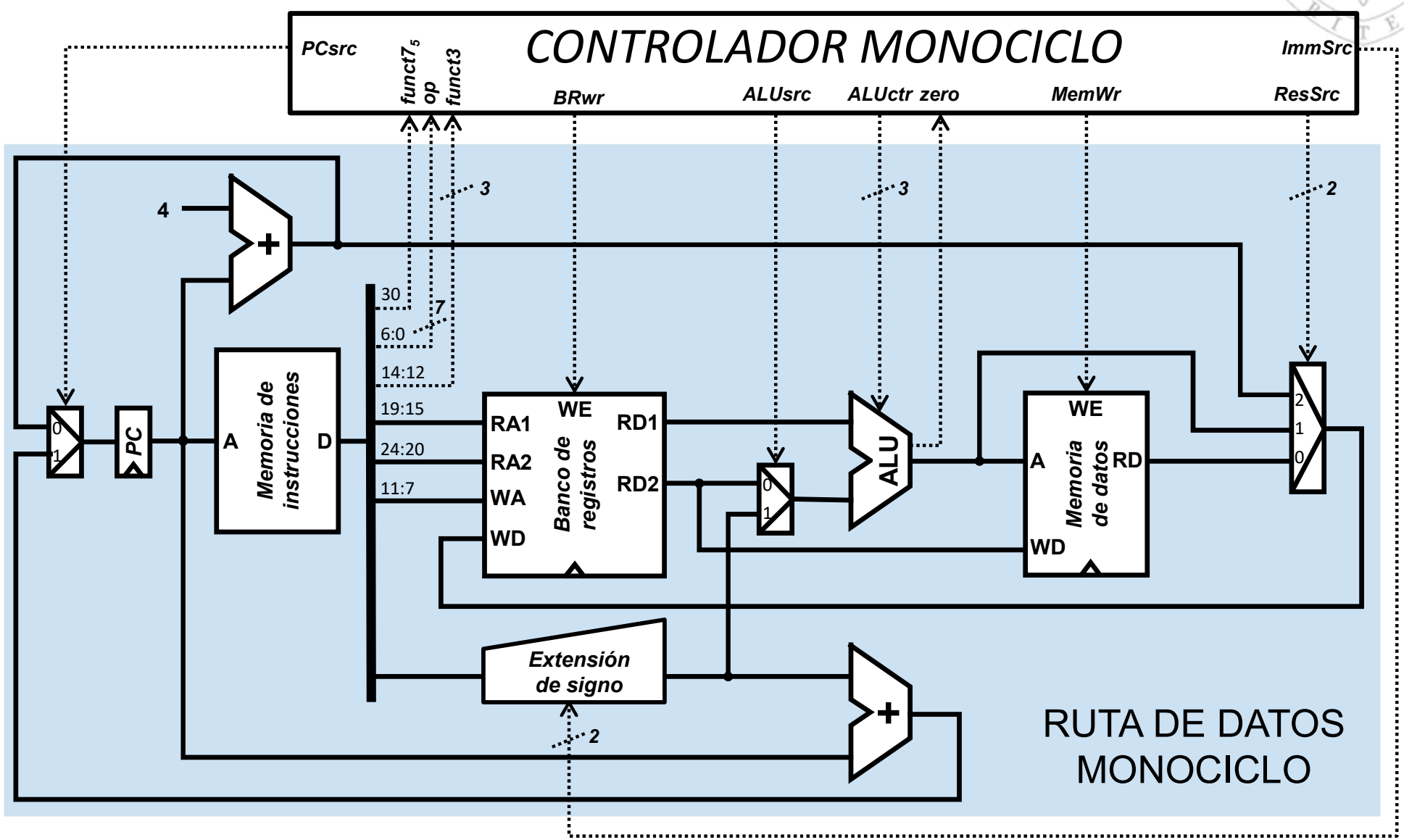
Diseño de la ruta de datos

Conexión del reloj y reset



Diseño de la ruta de datos

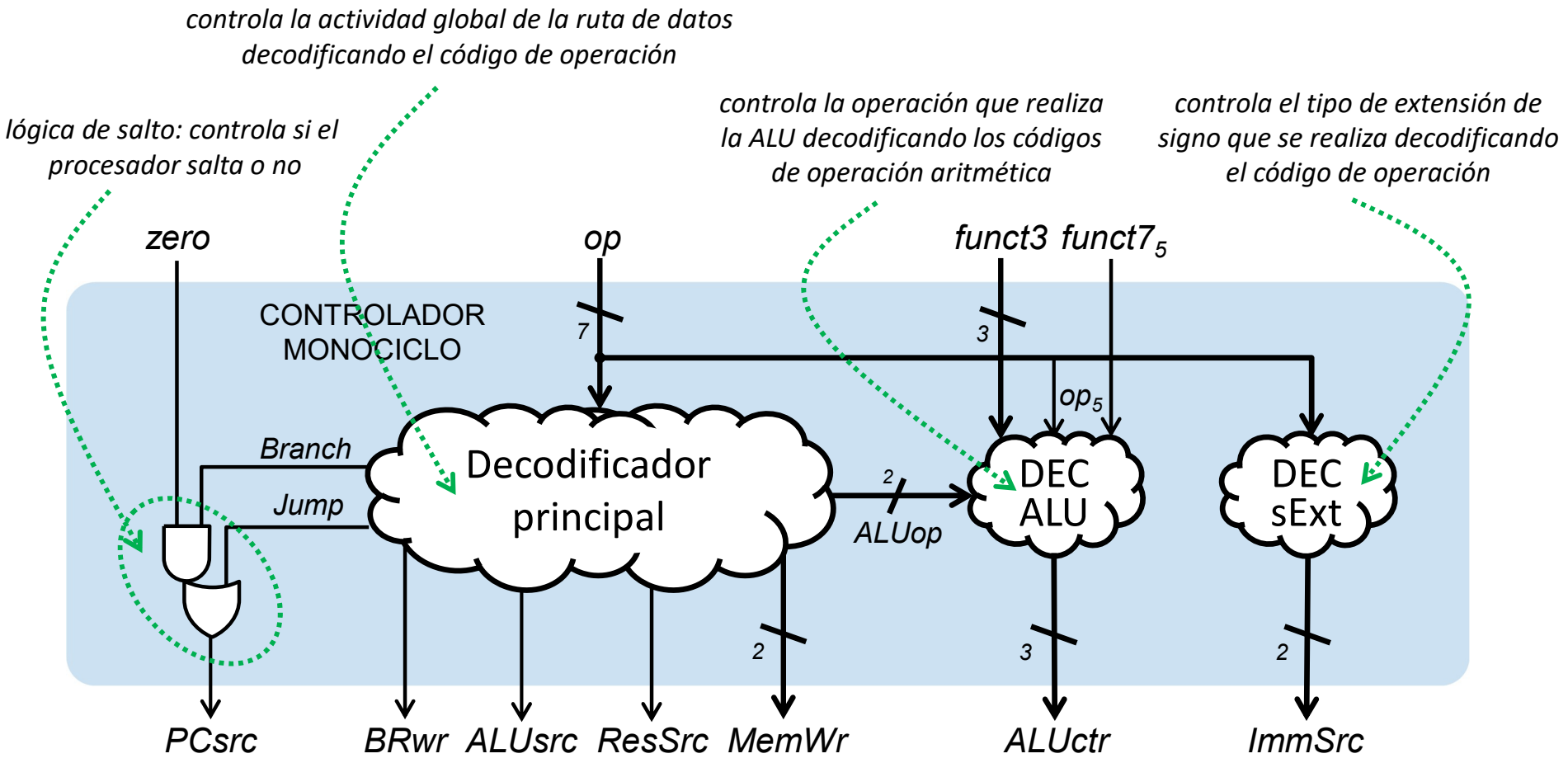
Estructura del sistema completo



Diseño del controlador

Estructura del controlador (i)

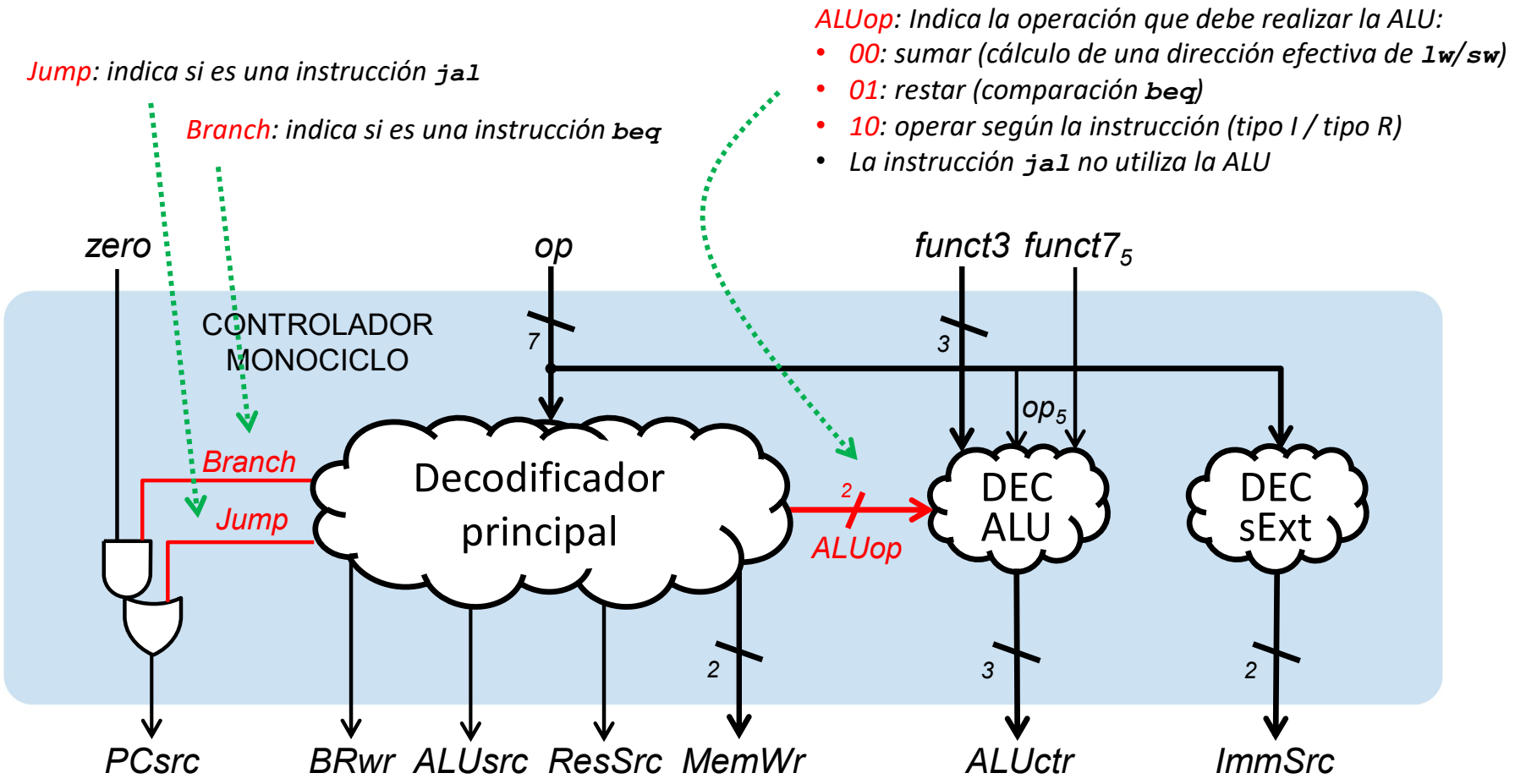
- En el procesador monociclo, el controlador es un circuito combinacional:
  - Que para facilitar el diseño estará formado por 4 subcircuitos.



Diseño del controlador

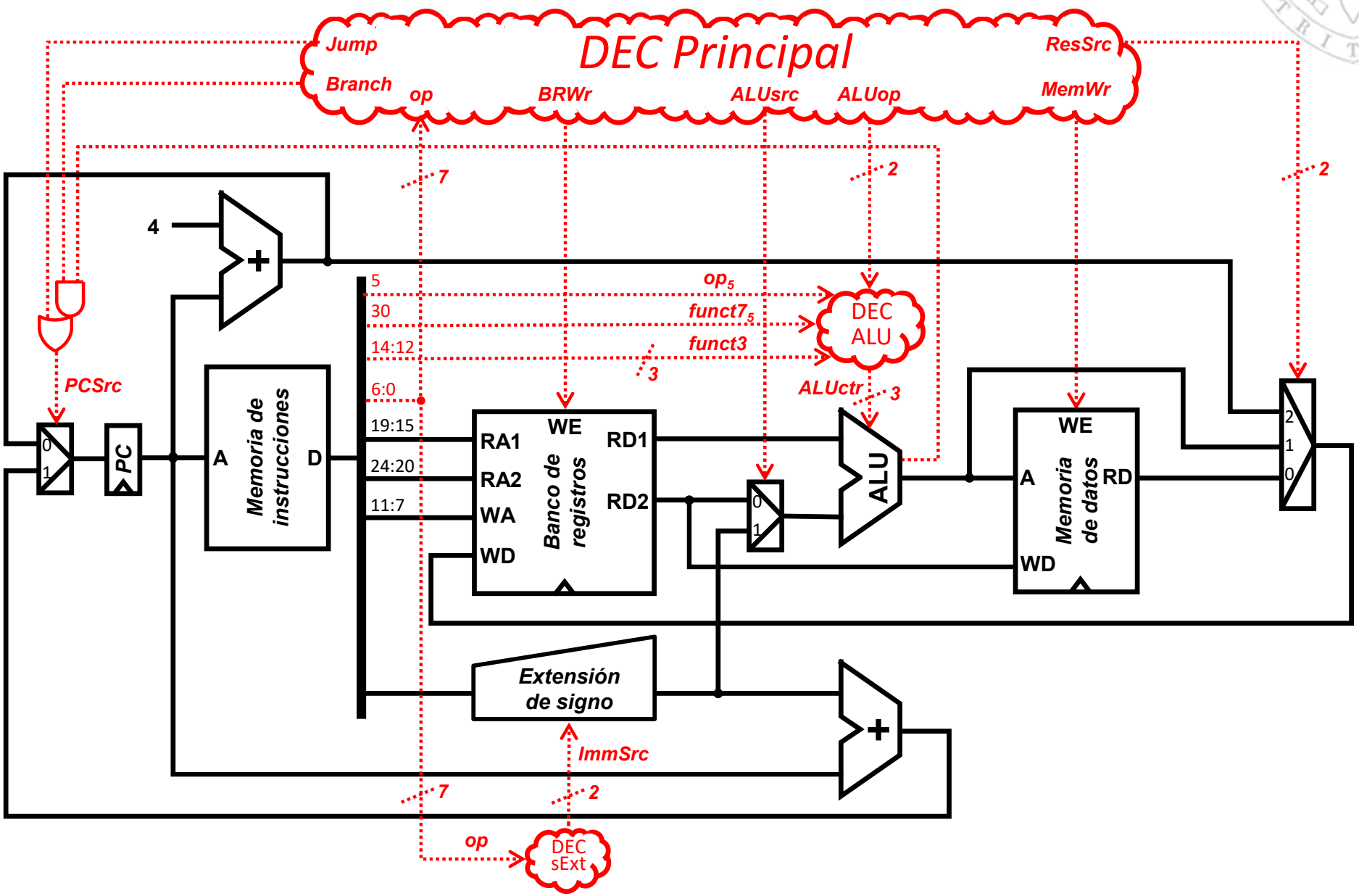
Estructura del controlador (ii)

- En el procesador monociclo, el controlador es un circuito combinacional:
  - Que para facilitar el diseño estará formado por 4 subcircuitos:



Diseño del controlador

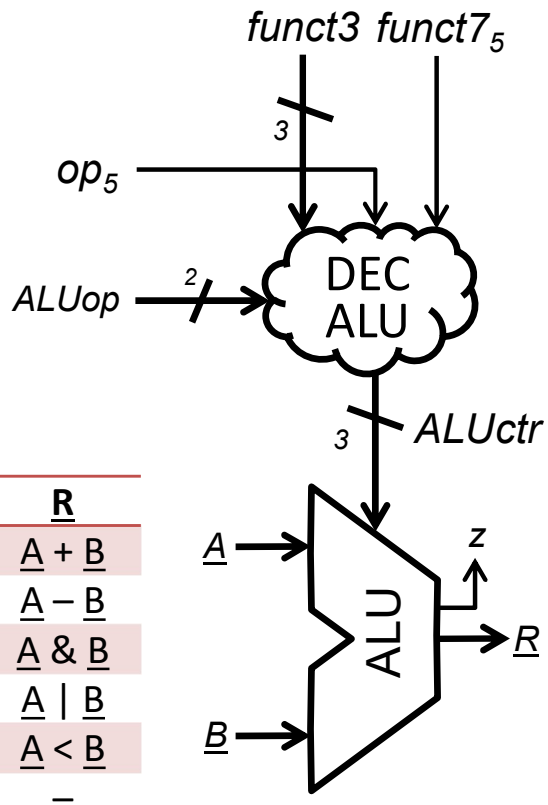
Estructura del controlador (iii)



Diseño del controlador

Diseño del controlador: DEC local a la ALU

- Este subcircuito indica a la ALU el tipo de operación que deber realizar.
  - Adapta la codificación de la operación en la instrucción a la codificación de operaciones de la ALU.



ALUctr	R
000	A + B
001	A – B
010	A & B
011	A | B
101	A < B
resto	–

Tabla de verdad

ALUop	op_5	$funct7_5$	funct3	ALUctr
00 (sumar)	X	X	XXX	000 (A + B)
01 (restar)	X	X	XXX	001 (A – B)
10 (operar)	0	X	000 (addi)	000 (A + B)
10 (operar)	1	0	000 (add)	000 (A + B)
10 (operar)	1	1	000 (sub)	001 (A – B)
10 (operar)	X	X	010 (slt/slti)	101 (A < B)
10 (operar)	X	X	110 (or/ori)	011 (A | B)
10 (operar)	X	X	111 (and/andi)	010 (A & B)

Diseño del controlador

Diseño del controlador: DEC local a la ALU

- Este subcircuito indica a la ALU el tipo de operación que deber realizar.
  - Adapta la codificación de la operación en la instrucción a la codificación de operaciones de la ALU.

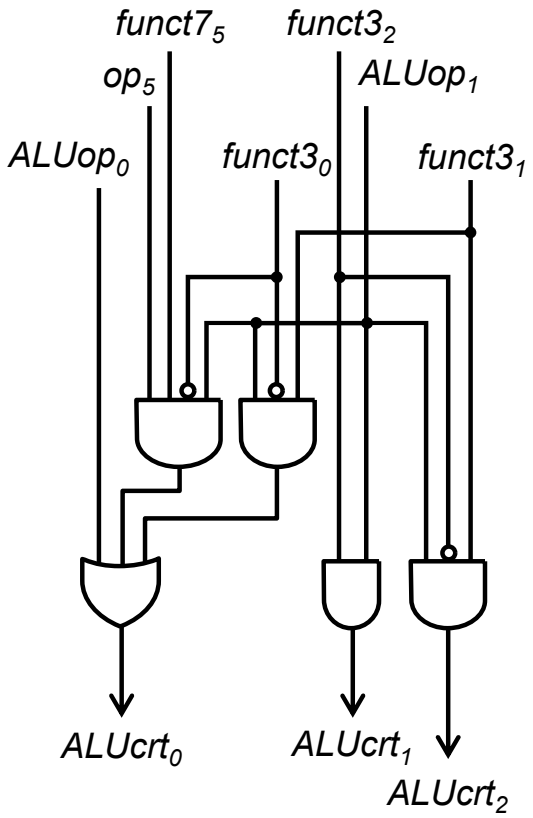


Tabla de verdad

ALUop	op_5	$funct7_5$	funct3	ALUctr
00 (sumar)	X	X	XXX	000 (A + B)
01 (restar)	X	X	XXX	001 (A – B)
10 (operar)	0	X	000 (addi)	000 (A + B)
10 (operar)	1	0	000 (add)	000 (A + B)
10 (operar)	1	1	000 (sub)	001 (A – B)
10 (operar)	X	X	010 (slt/slti)	101 (A < B)
10 (operar)	X	X	110 (or/ori)	011 (A | B)
10 (operar)	X	X	111 (and/andi)	010 (A & B)

Diseño del controlador

Diseño del controlador: DEC local al Extensor de Signo

- Esté subcircuito indica al Extensor de Signo el tipo de extensión que deber realizar.

ImmSrc	R
00	sExt(X) tipo-I
01	sExt(X) tipo-S
10	sExt(X) tipo-B
11	sExt(X) tipo-J

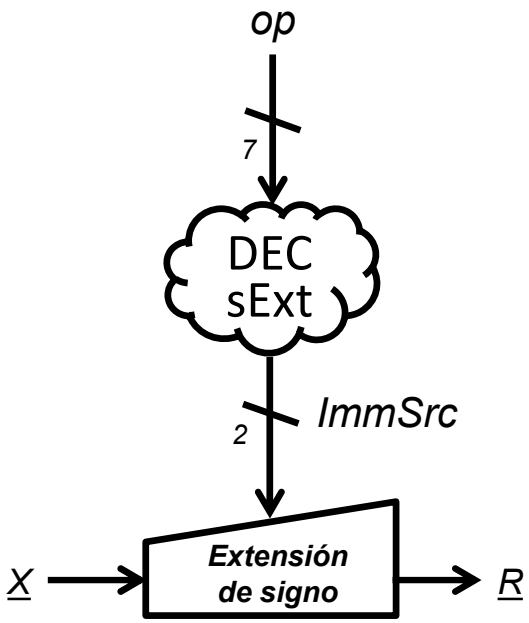


Tabla de verdad

Op	ImmSrc
0000011 (lw)	00 (tipo-I)
0100011 (sw)	01 (tipo-S)
0010011 (tipo-I)	00 (tipo-I)
0110011 (tipo-R)	–
1100011 (beq)	10 (tipo-B)
1101111 (jal)	11 (tipo-J)

Diseño del controlador

Diseño del controlador: DEC local al Extensor de Signo

- Esté subcircuito indica al Extensor de Signo el tipo de extensión que deber realizar.

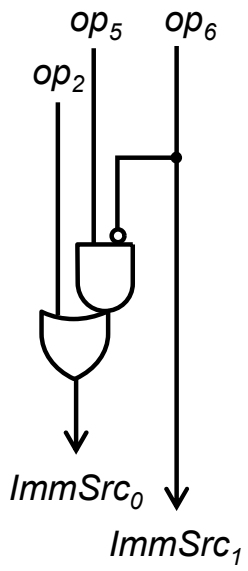


Tabla de verdad

Op	ImmSrc
0000011 (lw)	00 (tipo-I)
0100011 (sw)	01 (tipo-S)
0010011 (tipo-I)	00 (tipo-I)
0110011 (tipo-R)	–
1100011 (beq)	10 (tipo-B)
1101111 (jal)	11 (tipo-J)

Diseño del controlador

Diseño del controlador: DEC principal

- Este subcircuito gobierna el comportamiento general del procesador.

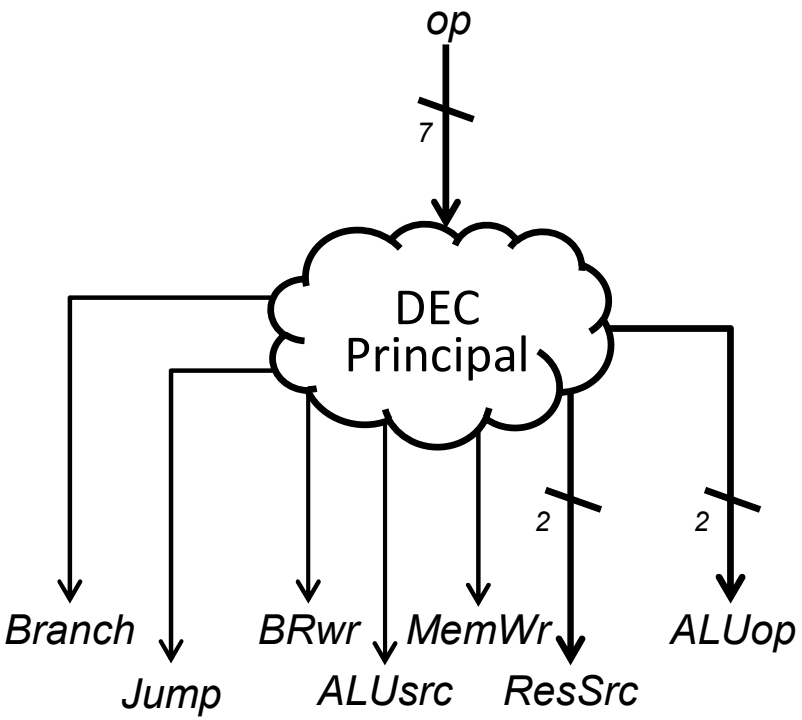


Tabla de verdad

op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

Diseño del controlador

Diseño del controlador: DEC principal

- Este subcircuito gobierna el comportamiento general del procesador.

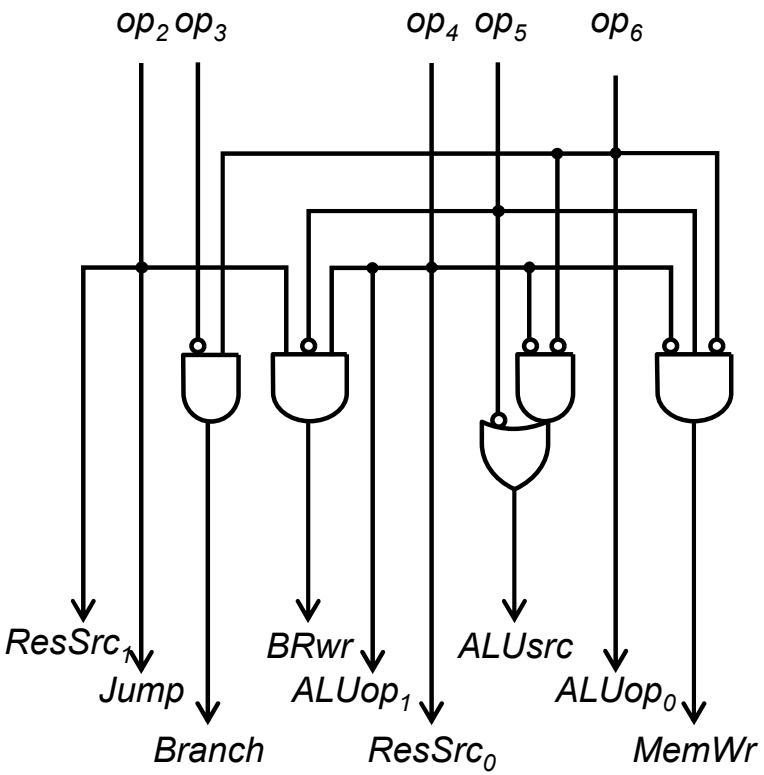
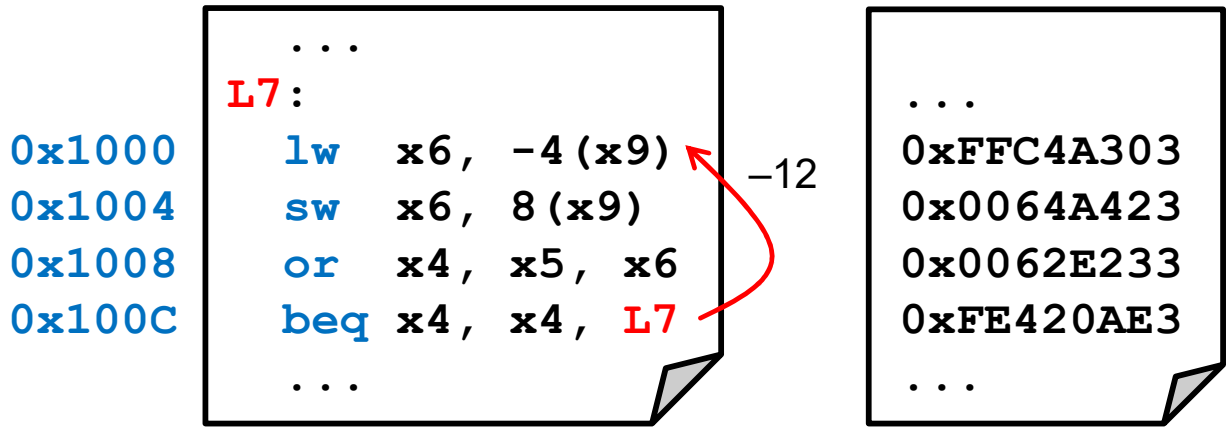


Tabla de verdad

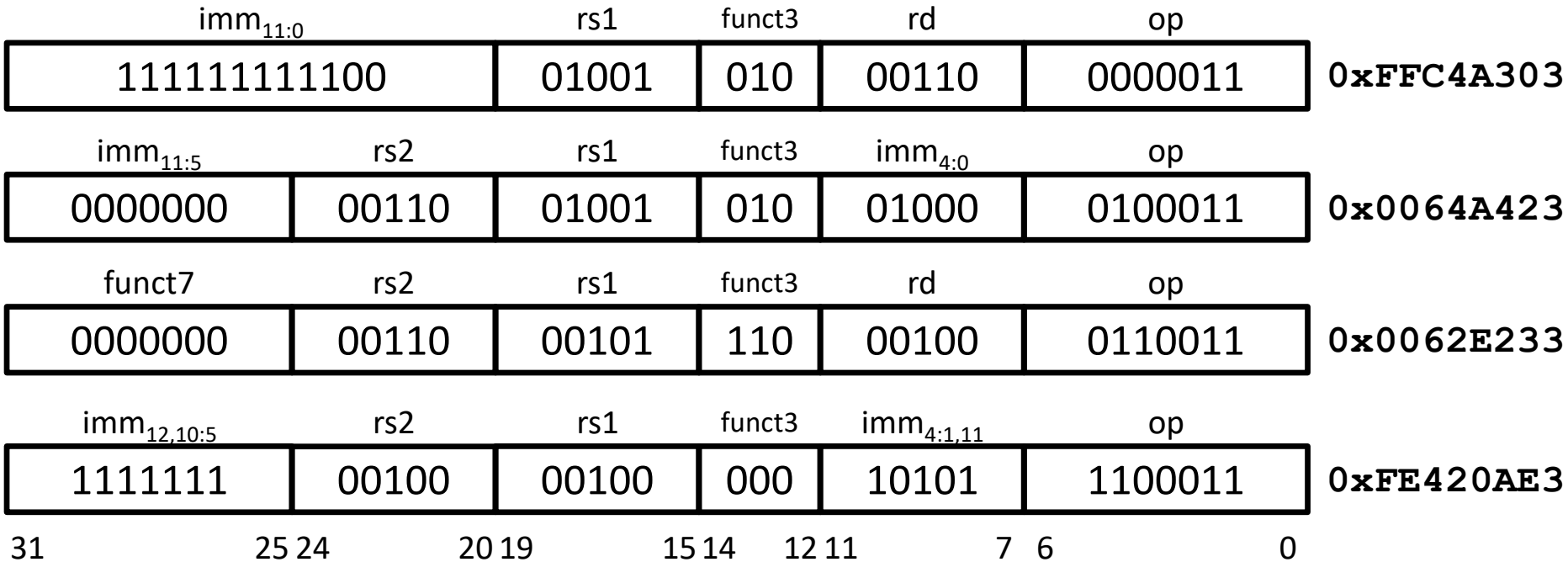
op	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc
0000011 (lw)	0	0	1	1	00 (sumar)	0	00
0100011 (sw)	0	0	0	1	00 (sumar)	1	–
0010011 (tipo-I)	0	0	1	1	10 (operar)	0	01
0110011 (tipo-R)	0	0	1	0	10 (operar)	0	01
1100011 (beq)	1	0	0	0	01 (restar)	0	–
1101111 (jal)	0	1	1	–	–	0	10

Procesador monociclo

Simulación (i)

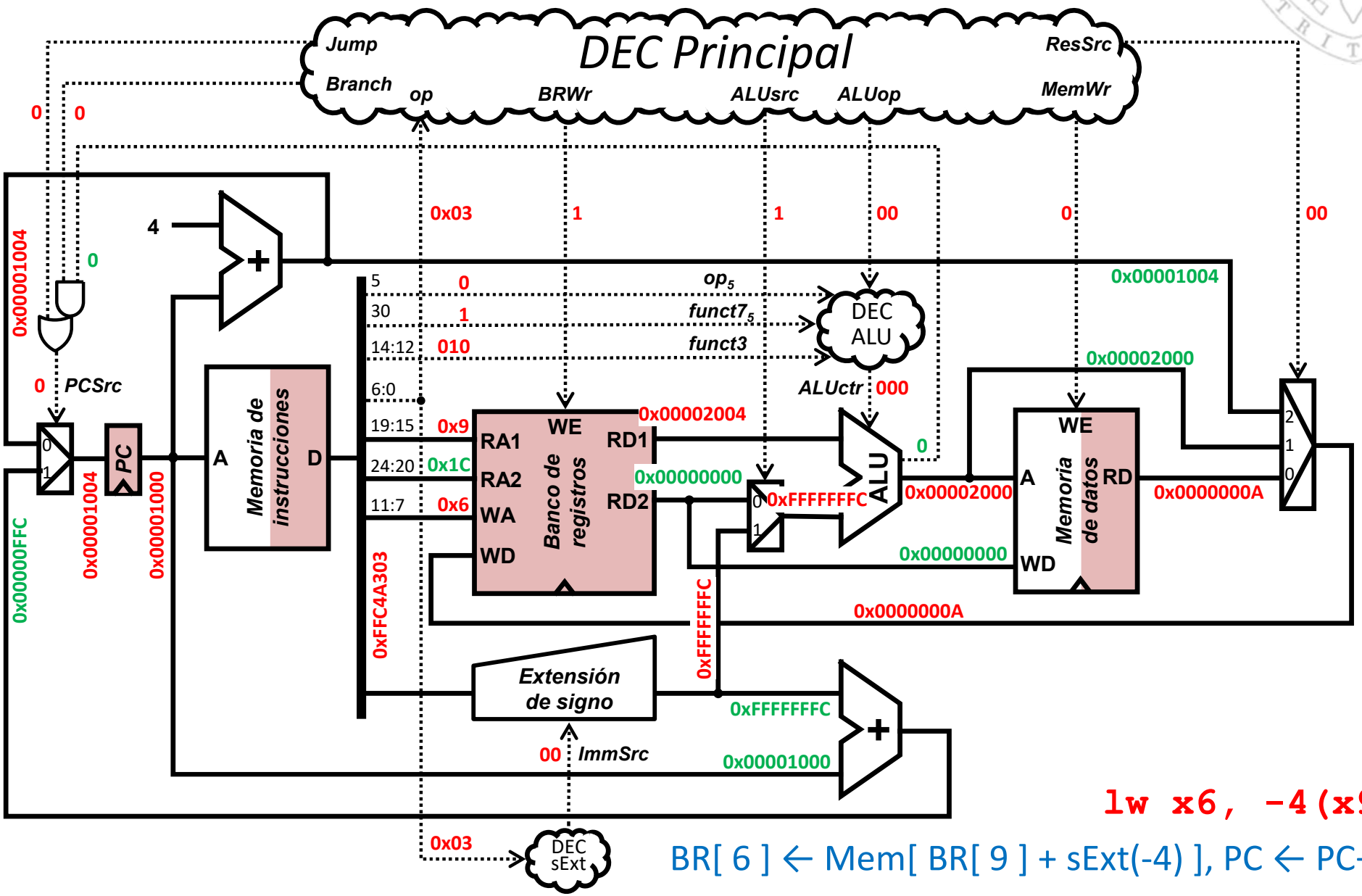


x5	6
x9	0x2004
Mem[0x2000]	10
PC	0x1000



Procesador monociclo

Simulación: 1er ciclo

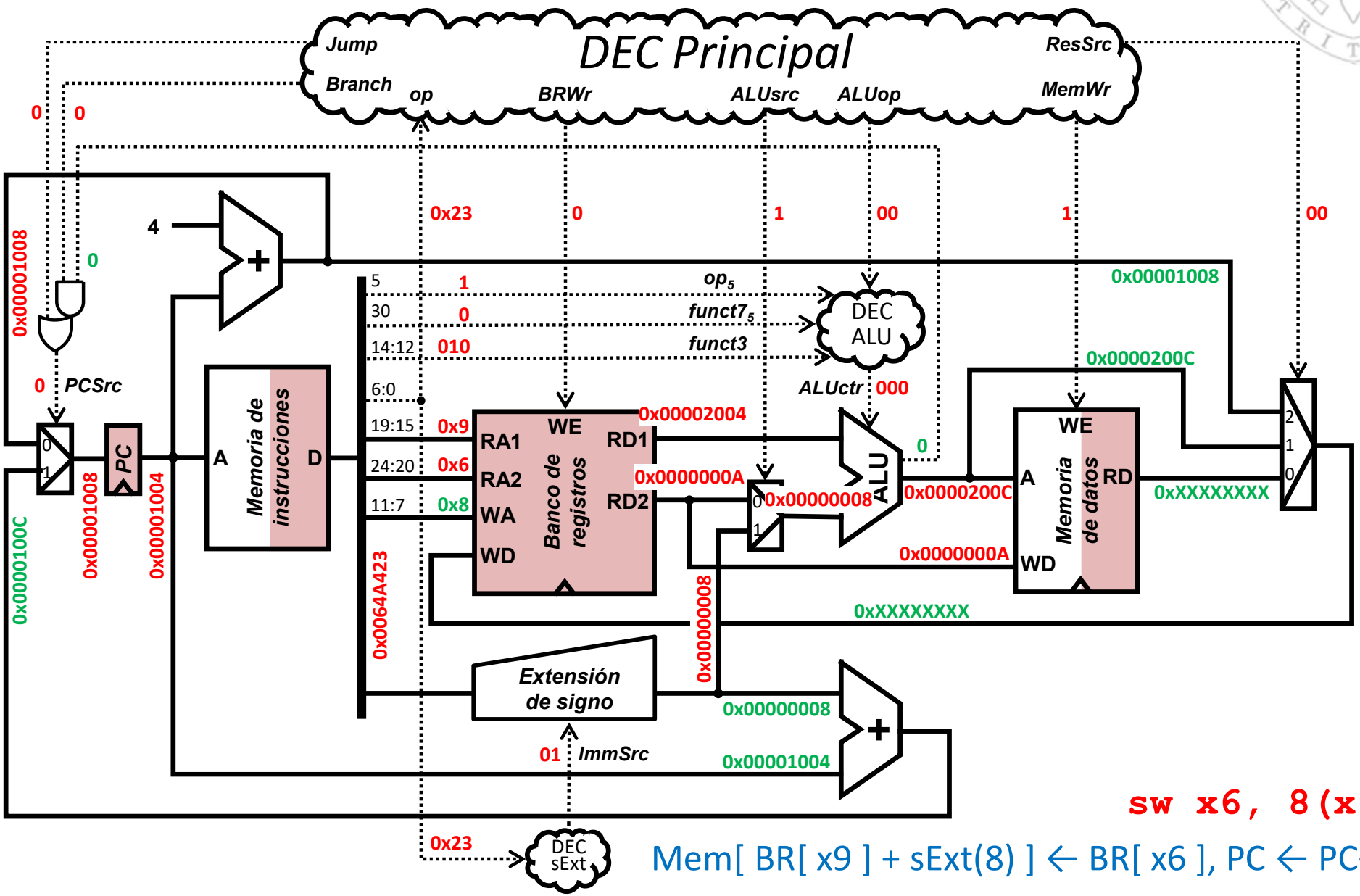


`lw x6, -4(x9)`

BR[6] ← Mem[BR[9] + sExt(-4)], PC ← PC+4

Procesador monociclo

Simulación: 2do. ciclo

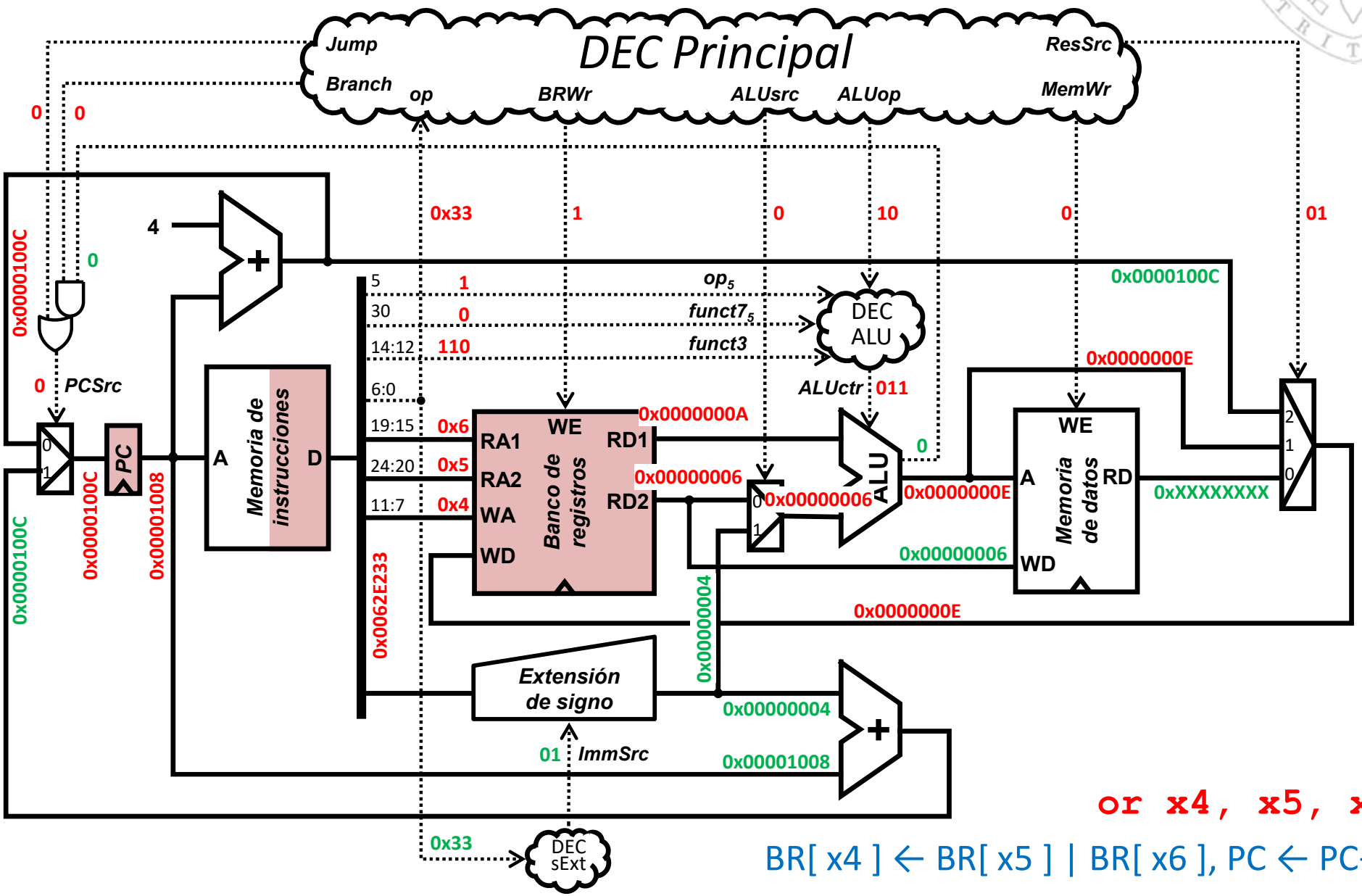


`sw x6, 8(x9)`

Mem[BR[x9] + sExt(8)] ← BR[x6], PC ← PC+4

Procesador monociclo

Simulación: 3er. ciclo

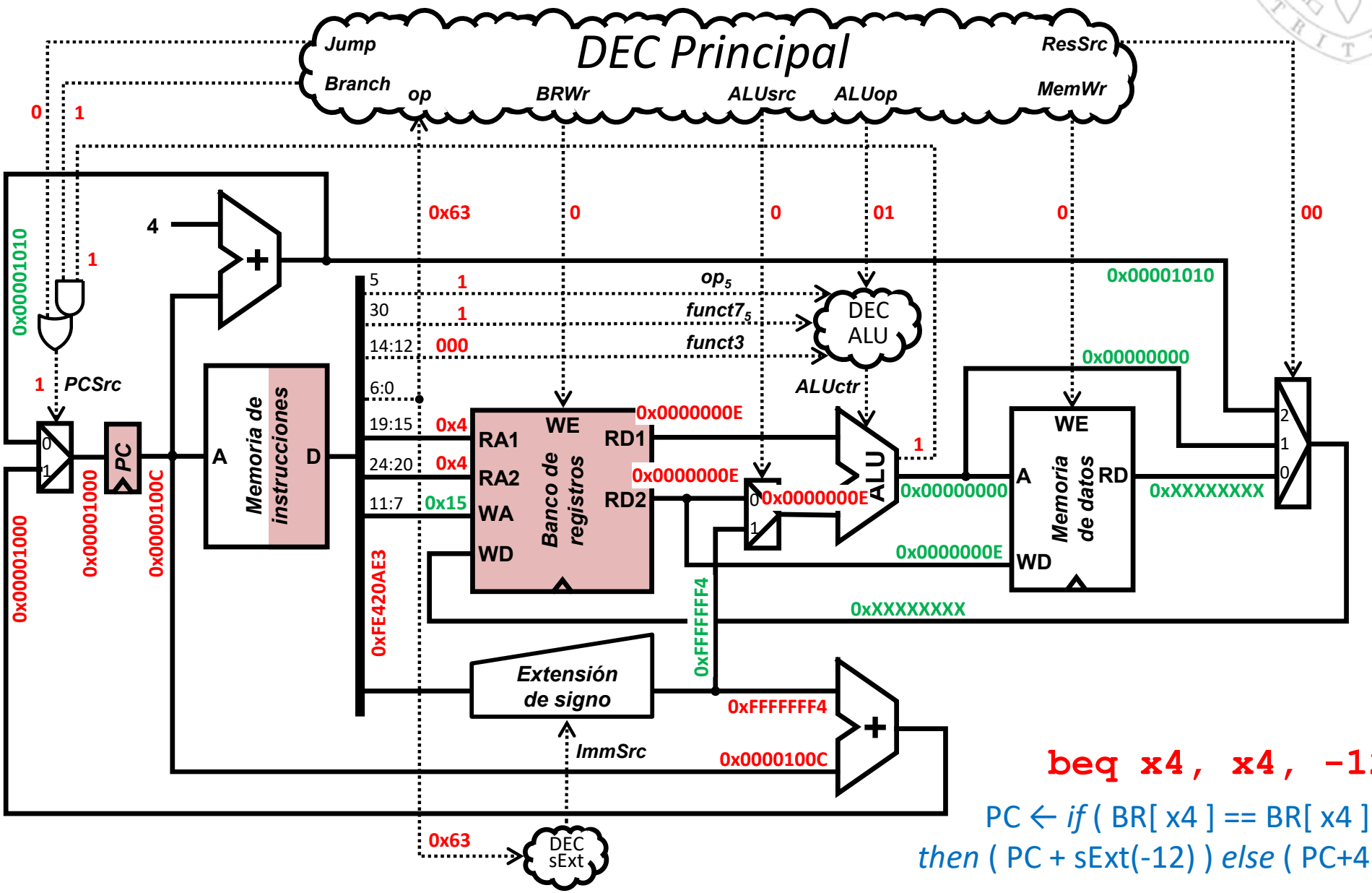



```
or x4, x5, x6
```

BR[x4] ← BR[x5] | BR[x6], PC ← PC+4

Procesador monociclo

Simulación: 4o. ciclo



`beq x4, x4, -12`

PC ← *if* (BR[x4] == BR[x4])
then (PC + sExt(-12)) *else* (PC+4)

Procesador monociclo

Coste y tiempo de ciclo (CMOS 90 nm)

$$area = 59181\ \mu m^2$$

$$t_{clk} = 27.6\ ns$$

$$f_{clk} = \frac{1}{t_{clk}} = \frac{1}{27.6 \cdot 10^{-9}\text{s}} = 36.2\ \text{MHz}$$

transferencia entre registros	instr.	camino crítico
PC ← PC+4	varias	9692 ps
BR[rd] ← Mem[BR[rs1] + sExt(imm)]	`lw`	**27616 ps**
Mem[BR[rs1] + sExt(imm)] ← BR[rs2]	`sw`	26661 ps
BR[rd] ← BR[rs1] *op* sExt(imm)	tipo I	19116 ps
BR[rd] ← BR[rs1] *op* BR[rs2]	tipo R	18928 ps
PC ← *if* (BR[rs1] = BR[rs2]) *then* (PC + sExt(imm)) *else* (PC+4)	`beq`	18547 ps
BR[rd] ← PC+4	`jal`	10073 ps
PC ← PC + sExt(imm)		17033 ps
	max.	**27616 ps**

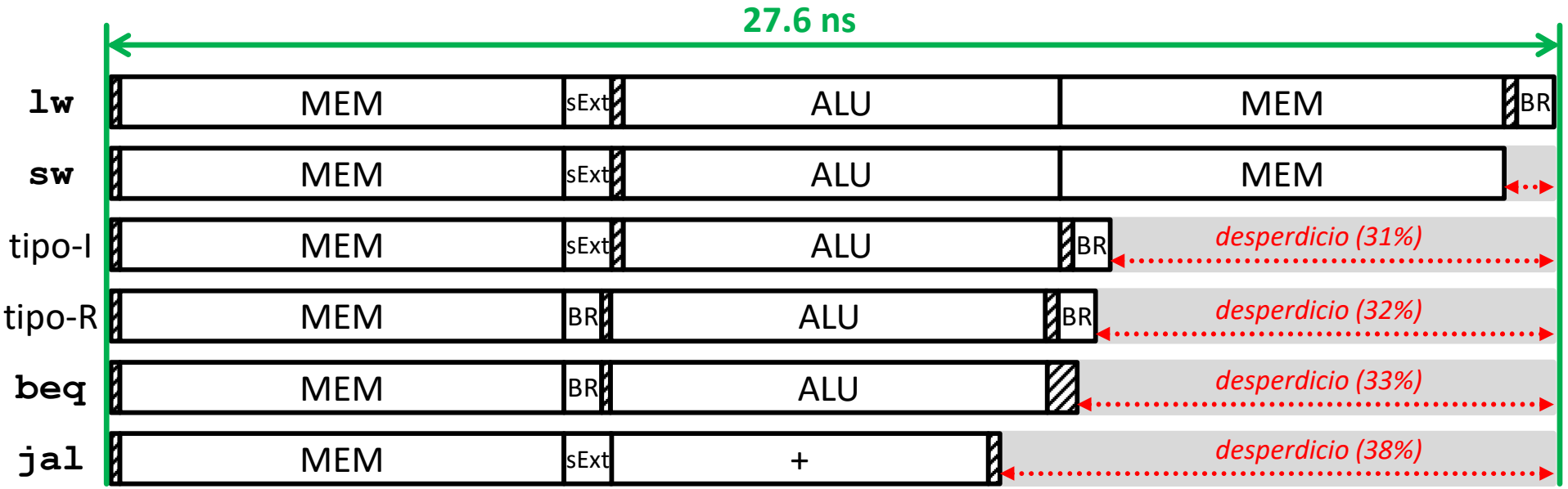
```
   ...
L7:
   lw   x6, -4(x9)   ·····> 1 ciclo
   sw   x6, 8(x9)    ·····> 1 ciclo
   or   x4, x5, x6   ·····> 1 ciclo
   beq  x4, x4, L7   ·····> + 1 ciclo
   ...                      4 ciclos
```

$$t_{ejec} = 4 \times 27.6\ ns = 110.4\ ns$$

Procesador monociclo

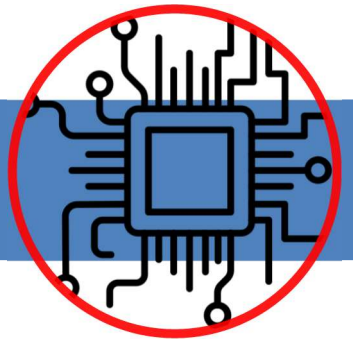
Conclusiones

- La implementación monociclo tiene algunos problemas:
  - El tiempo de ciclo viene dado por la instrucción más lenta
    - Todas las instrucciones tardan lo mismo en ejecutarse con independencia de su complejidad: se desperdicia tiempo en la ejecución de instrucciones rápidas.
    - En repertorios reales existen algunas instrucciones muy largas: memorias lentas, operaciones aritméticas complejas, modos de direccionamiento complejos...
  - No es posible reutilizar hardware:
    - Requiere una ALU y 2 sumadores, memoria de instrucciones y datos separada...



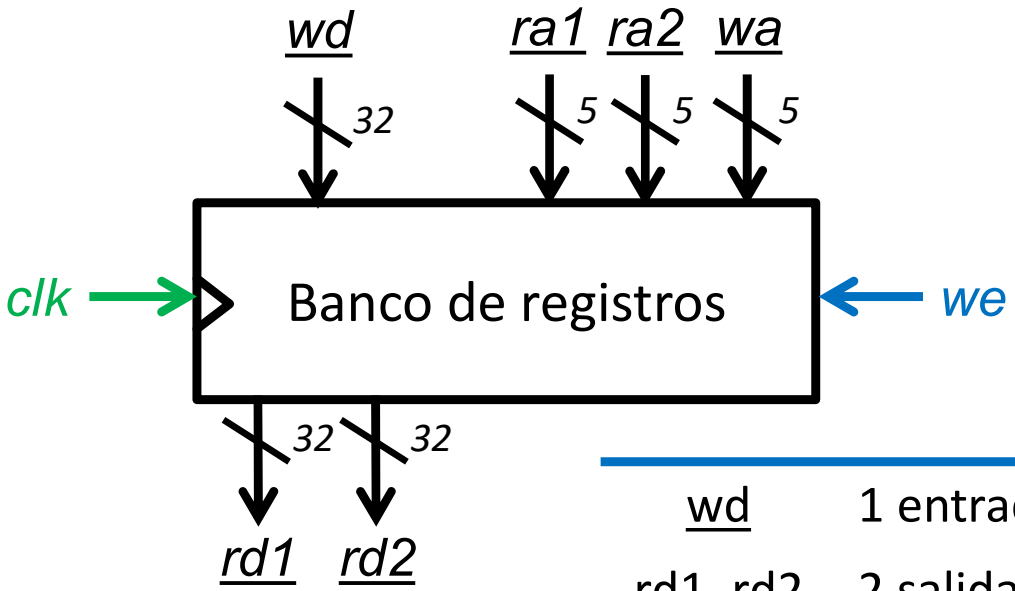
- Diseño del Banco de Registros.
- Diseño de la Memoria.
- Diseño de la ALU.
- Diseño del Extensor de Signo.
- Cálculo del coste.
- Cálculo del tiempo de ciclo.

Apéndice tecnológico



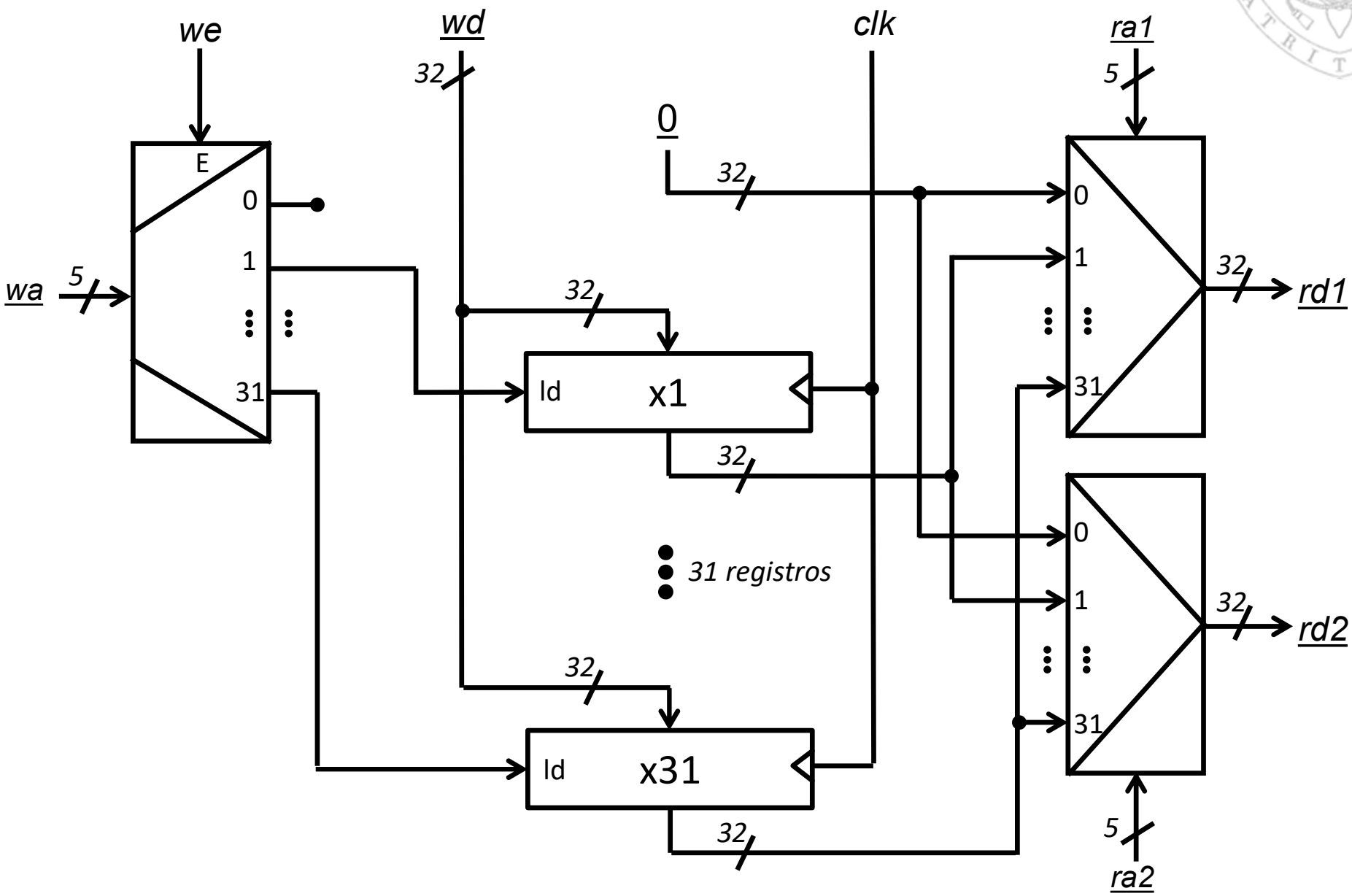
Diseño del Banco de Registros

- El Banco de Registros contiene 32 registros de datos de 32 bits.

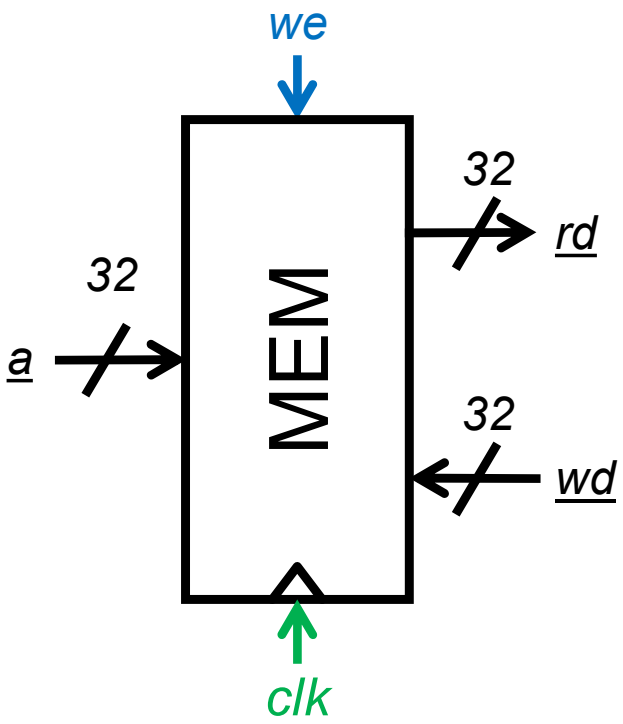


wd	1 entrada de datos de 32 bits
rd1, rd2	2 salidas de datos de 32 bits
wa	1 entrada de dirección de escritura de 5 bits
ra1, ra2	2 entradas de dirección de lectura de 5 bits
we	1 entrada de capacitación de escritura
clk	1 entrada de reloj

Diseño del Banco de Registros



Diseño de la Memoria

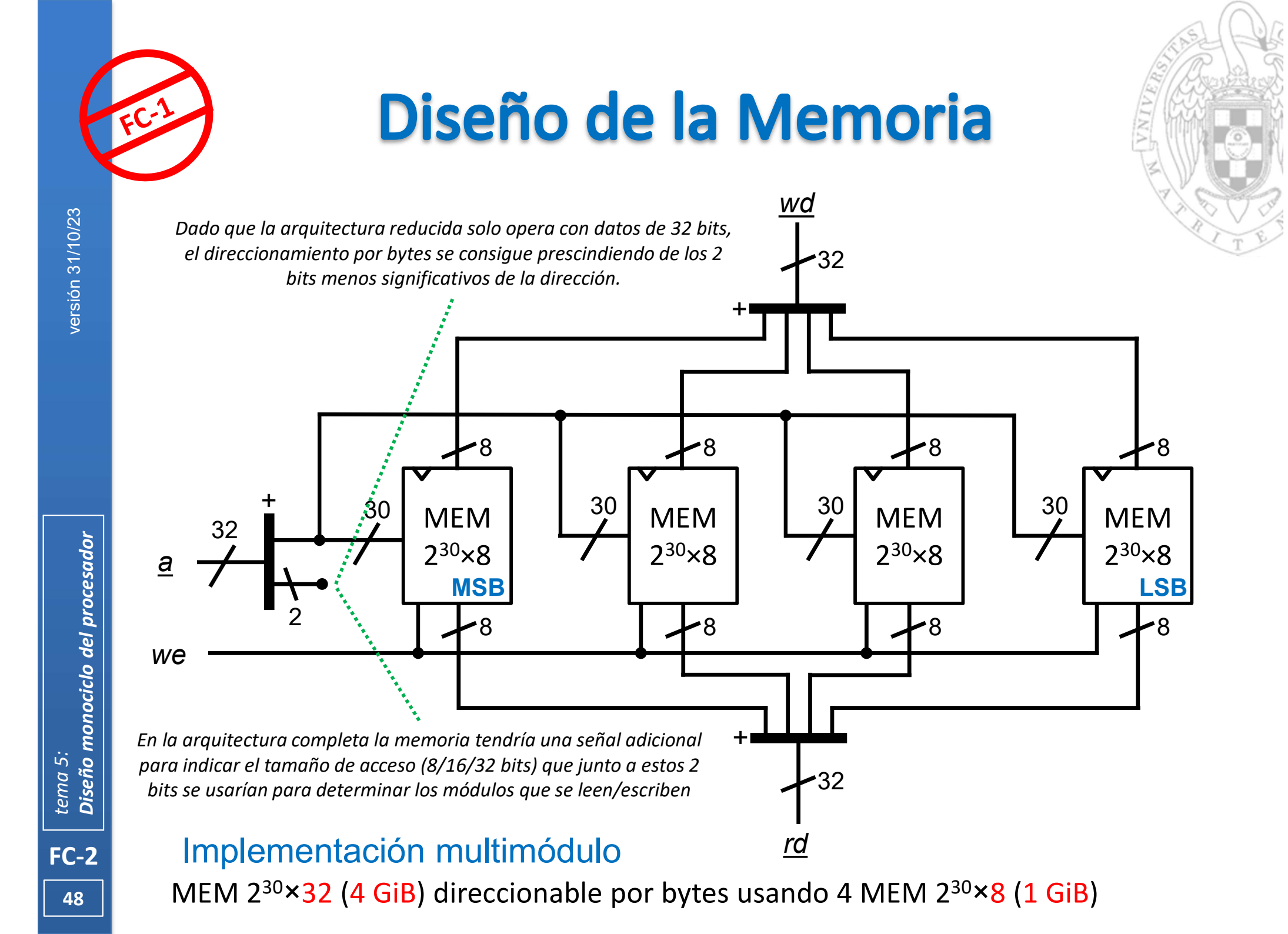


RAM $2^{32}\times8 = 2^{30}\times32$
(2^{30} palabras de 32 bits)

wd	1 entrada de datos de 32 bits
rd	1 salidas de datos de 32 bits
a	1 entrada de dirección de 32 bits
we	1 entrada de capacitación de escritura
clk	1 entrada de reloj

Memoria con doble puerto de datos,
direccionable por bytes
y con ordenamiento little-endian

Diseño de la Memoria



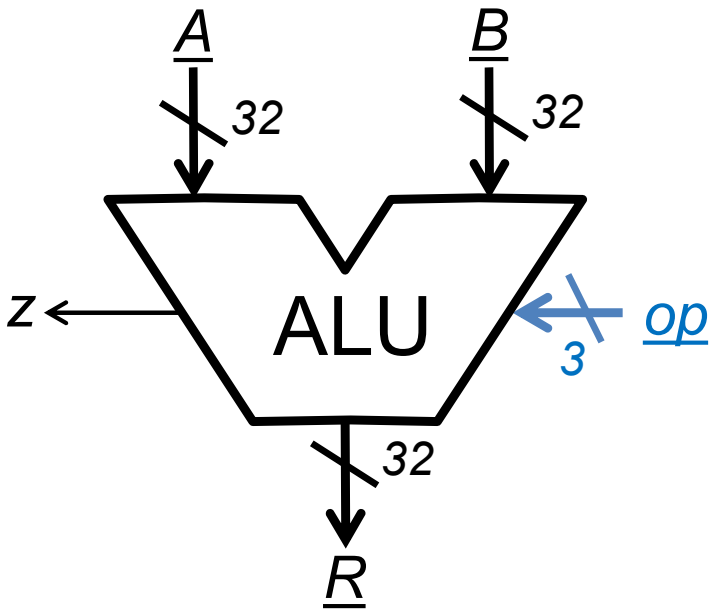
Dado que la arquitectura reducida solo opera con datos de 32 bits, el direccionamiento por bytes se consigue prescindiendo de los 2 bits menos significativos de la dirección.

En la arquitectura completa la memoria tendría una señal adicional para indicar el tamaño de acceso (8/16/32 bits) que junto a estos 2 bits se usarían para determinar los módulos que se leen/escriben

Implementación multimódulo

MEM 2^{30}×32 (4 GiB) direccionable por bytes usando 4 MEM 2^{30}×8 (1 GiB)

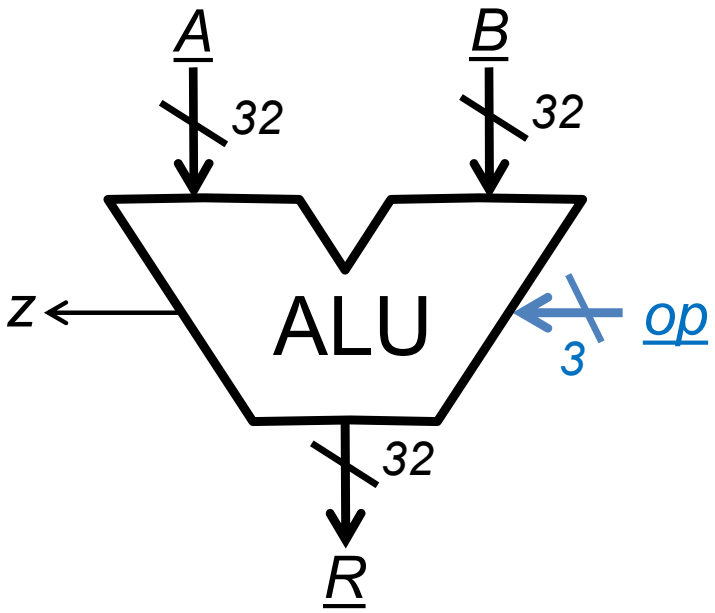
Diseño de la ALU



A, B	2 entradas de datos de 32 bits
op	1 entrada de selección de operación
R	1 salida de datos de 32 bits
z	1 salida de detección de cero

- La ALU es un módulo combinacional que realiza:
  - El cálculo de direcciones efectivas de memoria en instrucciones **`lw`**/**`sw`**.
  - Todas las operaciones aritmético-lógicas del instrucciones tipo I / tipo R.
  - La comparación de operandos en la instrucción **`beq`**.
  - En la ruta de datos multiciclo, también se usará para para incrementar el PC y realizar el cálculo de direcciones de salto en instrucciones **`beq`**/**`jal`**

Diseño de la ALU



A, B	2 entradas de datos de 32 bits
op	1 entrada de selección de operación
R	1 salida de datos de 32 bits
z	1 salida de detección de cero

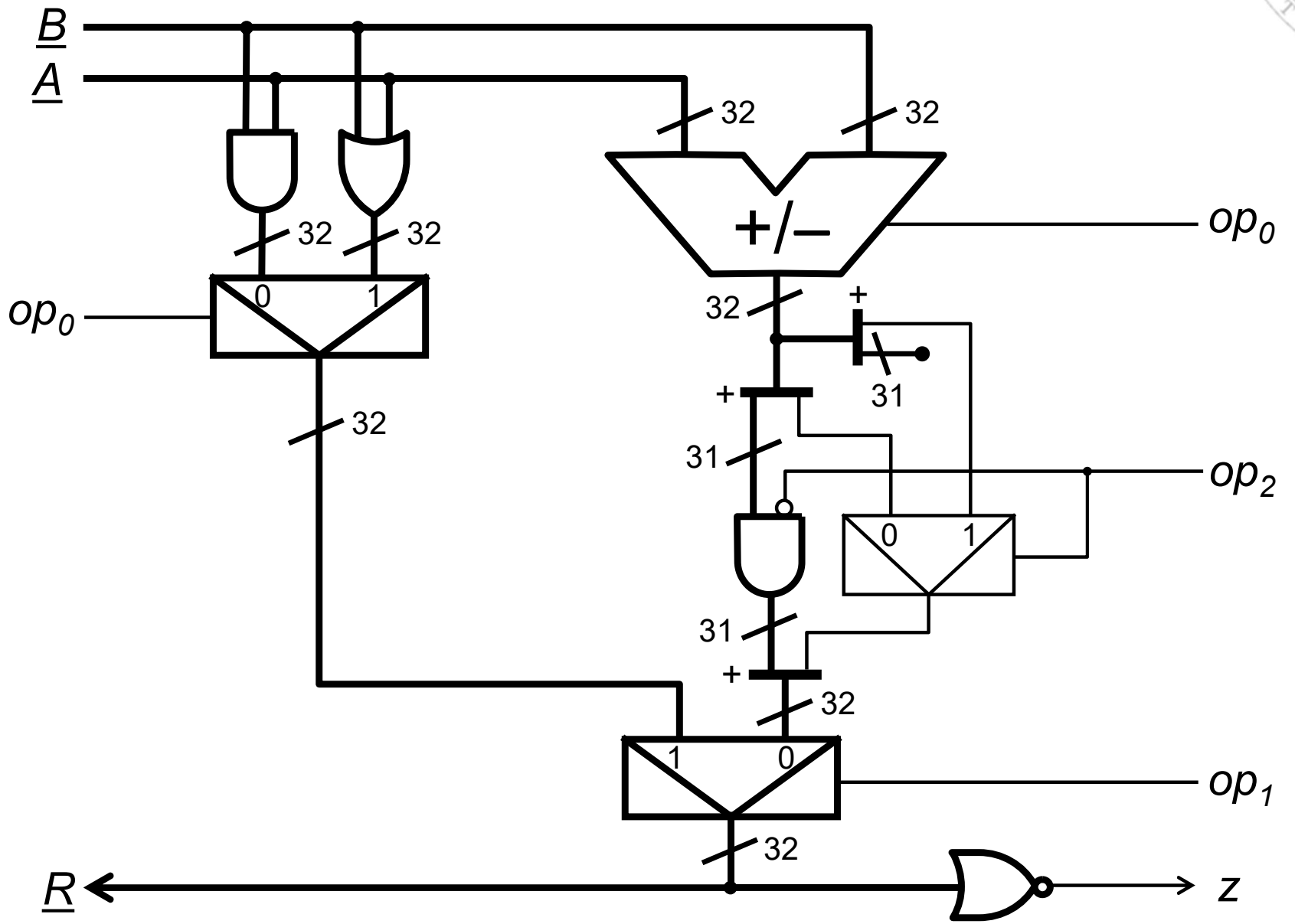
operaciones aritméticas

op_2	op_1	op_0	R
0	0	0	A + B
0	0	1	A – B
1	0	0	–
1	0	1	*if* (A<B) *then* 1 *else* 0

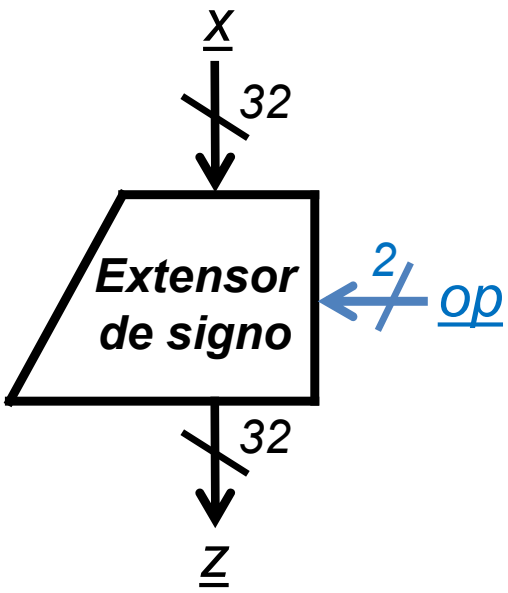
operaciones lógicas

op_2	op_1	op_0	R
0	1	0	A & B
0	1	1	A | B
1	1	0	–
1	1	1	–

Diseño de la ALU



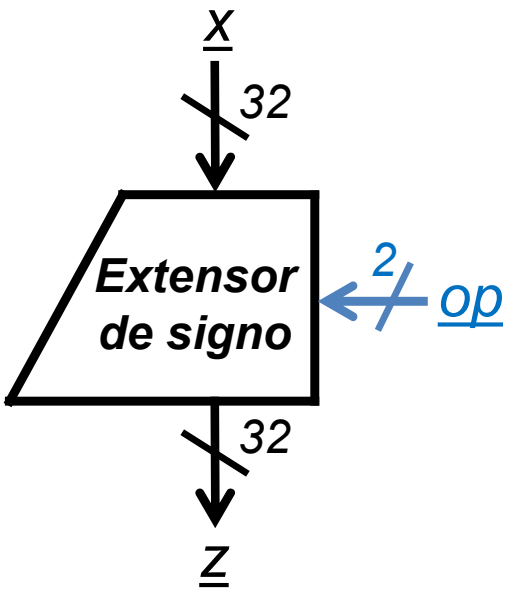
Diseño del Extensor de Signo



x	1 entrada de datos de 32 bits (instrucción)
op	1 entrada de selección de operación
z	1 salida de datos de 32 bits (operando inmediato)

- El Extensor de Signo es un módulo combinacional que construye el operando inmediato de 32 bits a partir de información contenida en los campos imm de la instrucción:
  - Para cada tipo de instrucción el campo imm tiene diferente tamaño y posición dentro de la instrucción.
  - Por ello, aparte de extender el signo, debe reordenar bits y completar con 0 en el caso de direcciones de salto.

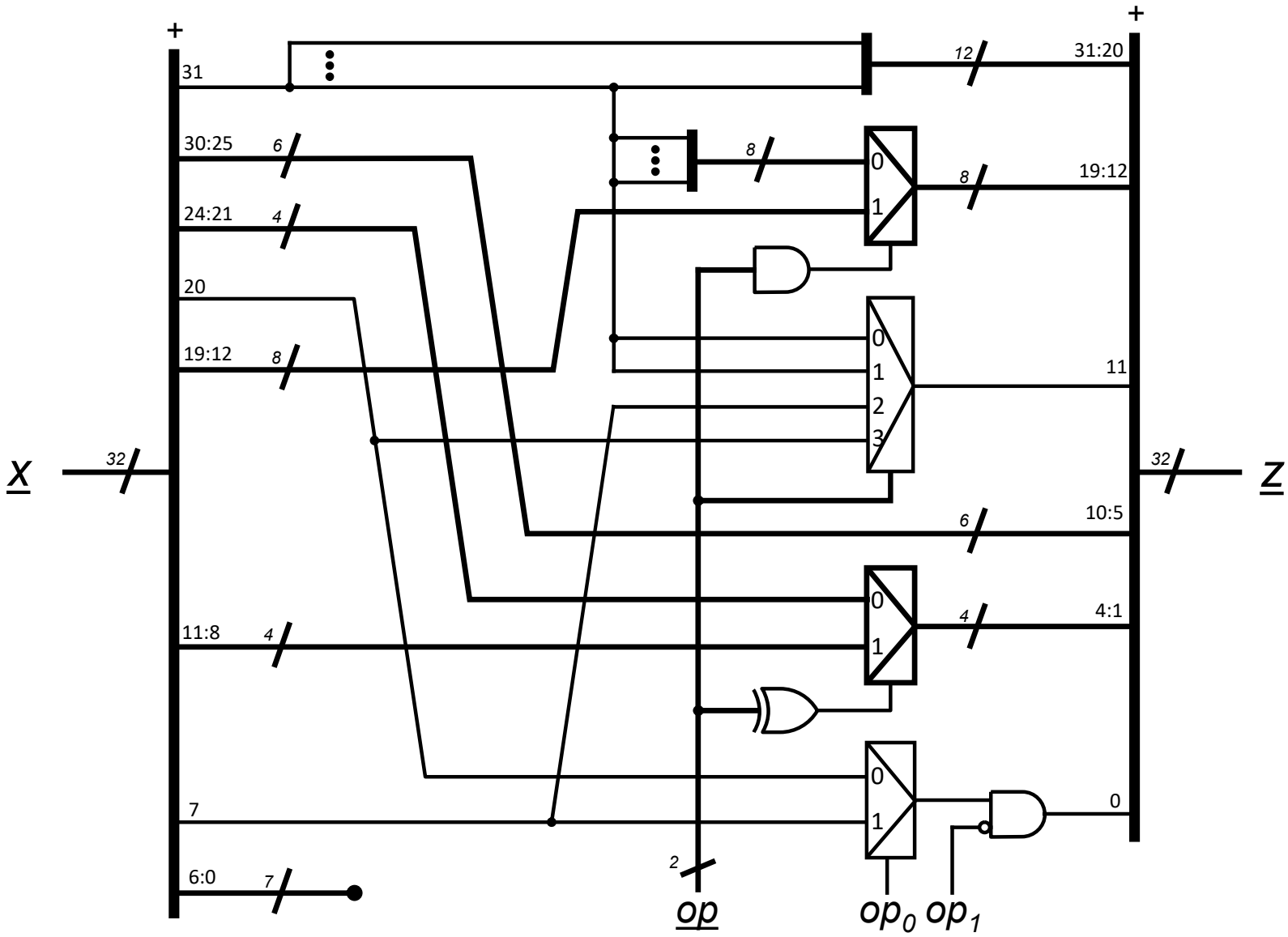
Diseño del Extensor de Signo



x	1 entrada de datos de 32 bits (instrucción)
op	1 entrada de selección de operación
z	1 salida de datos de 32 bits (operando inmediato)

op	z_{31} z_{30} z_{29} z_{28} z_{27} z_{26} z_{25} z_{24} z_{23} z_{22} z_{21} z_{20} z_{19} z_{18} z_{17} z_{16} z_{15} z_{14} z_{13} z_{12} z_{11} z_{10} z_9 z_8 z_7 z_6 z_5 z_4 z_3 z_2 z_1 z_0
00	x_{31} x_{30} x_{29} x_{28} x_{27} x_{26} x_{25} x_{24} x_{23} x_{22} x_{21} x_{20}
01	x_{31} x_{30} x_{29} x_{28} x_{27} x_{26} x_{25} x_{11} x_{10} x_9 x_8 x_7
10	x_{31} x_7 x_{30} x_{29} x_{28} x_{27} x_{26} x_{25} x_{11} x_{10} x_9 x_8 0
11	x_{31} x_{31} x_{31} x_{31} x_{31} x_{31} x_{31} x_{31} x_{31} x_{31} x_{31} x_{31} x_{19} x_{18} x_{17} x_{16} x_{15} x_{14} x_{13} x_{12} x_{20} x_{30} x_{29} x_{28} x_{27} x_{26} x_{25} x_{24} x_{23} x_{22} x_{21} 0

Diseño del Extensor de Signo



Cálculo del coste y tiempo de ciclo

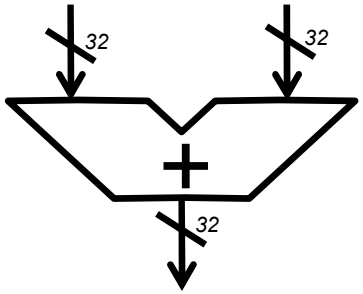
- Para calcular el coste del procesador es la suma del coste de cada uno de los módulos que lo componen.
  - El coste de cada módulo se calcula sumando el coste de sus celdas.

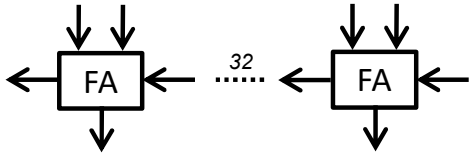
- El tiempo de ciclo del procesador es el máximo de los caminos críticos de las transferencias entre registros que realiza el procesador.
  - El camino crítico de una transferencia entre registros es el camino de datos de mayor retardo de todos los implicados en dicha transferencia.
  - En el procesador monociclo, en cada ciclo se ejecuta una instrucción que implica 1 ó 2 transferencias entre registros: la de actualización del PC y la operativa propia de cada instrucción.

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

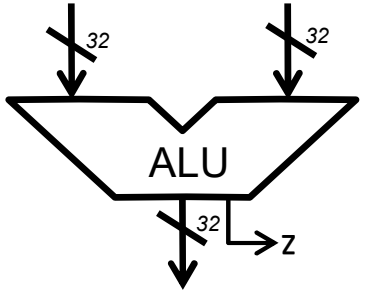
Cálculo del coste y tiempo de ciclo

CMOS 90 nm

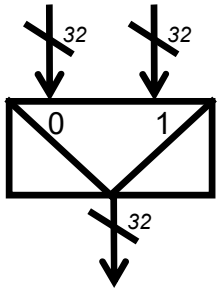


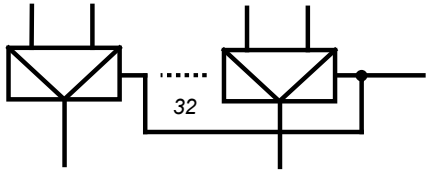


área: 32×29.49 = **944 μm²**
retardo: 32×226 = **7232 ps**

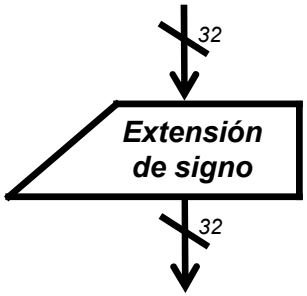


área: 3052 μm²
retardo: 8360 ps

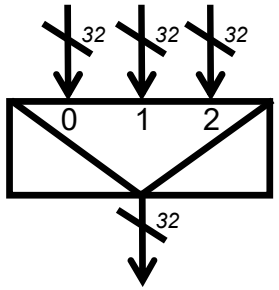


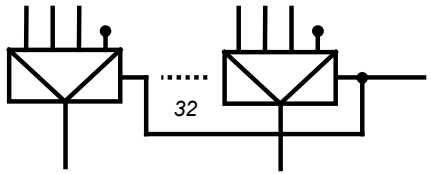


área: 32×11.05 = **354 μm²**
retardo: 223 ps

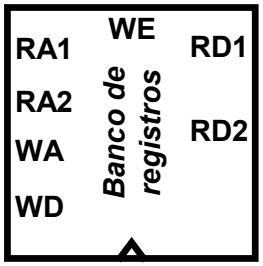


área: 202 μm²
retardo: 460 ps





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



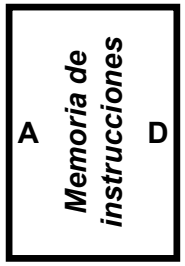
área: 51405 μm²
retardo lectura: 723 ps
setup escritura: 705 ps
(debido al DEC de dirección)

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

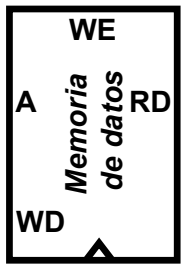
Cálculo del coste y tiempo de ciclo

CMOS 90 nm

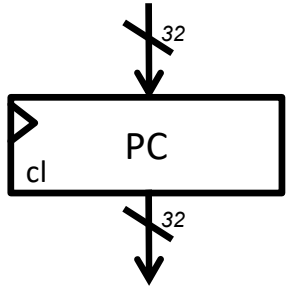
Comportamiento idealizado: retardo comparable al de la ALU
(para que pueda leerse en un ciclo de reloj)

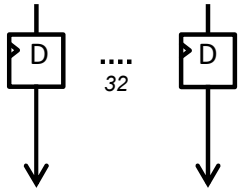


área: –
tiempo de acceso: **8500 ps**



área: –
tiempo de acceso: **8500 ps**





área: 32×32.26 = **1032 μm²**
retardo CLK→Q: 1×167 = **167 ps**
setup: **0 ps**

área: **56 μm²**
retardo: **490 ps**

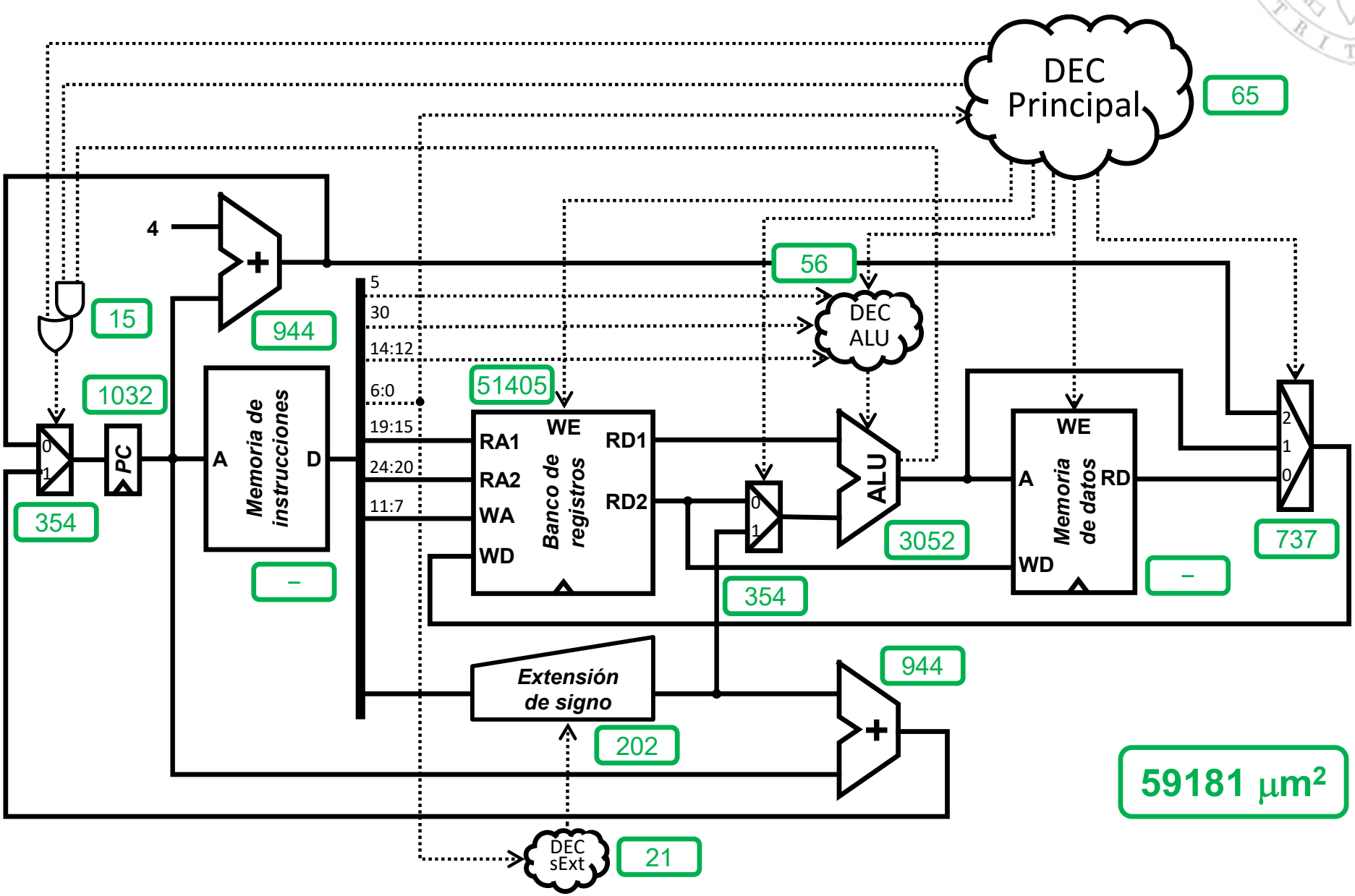
área: **65 μm²**
retardo: **451 ps**

área: **21 μm²**
retardo: **451 ps**

área: **15 μm²**
retardo: **351 ps**

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

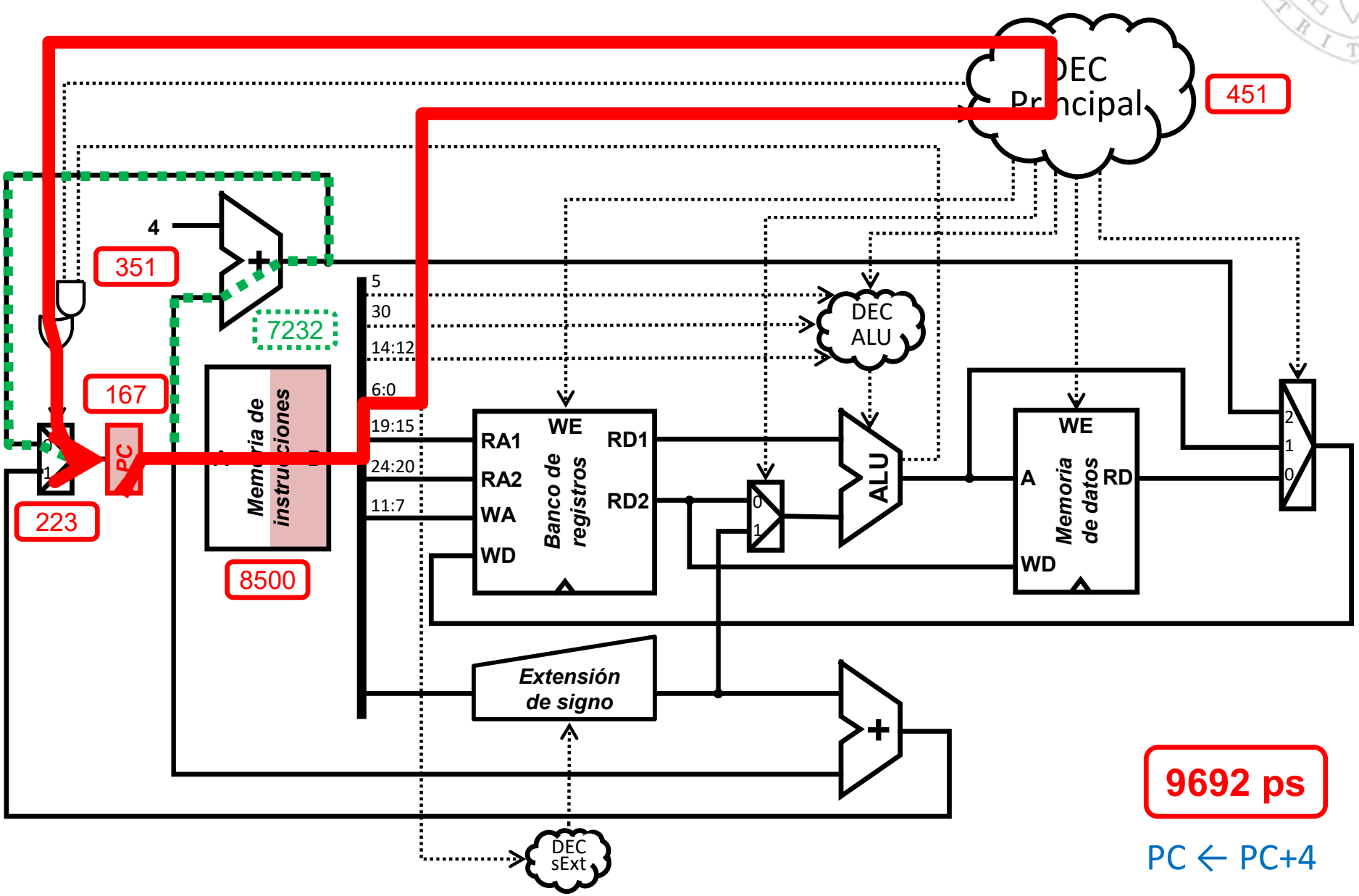
Cálculo del coste



59181 µm²

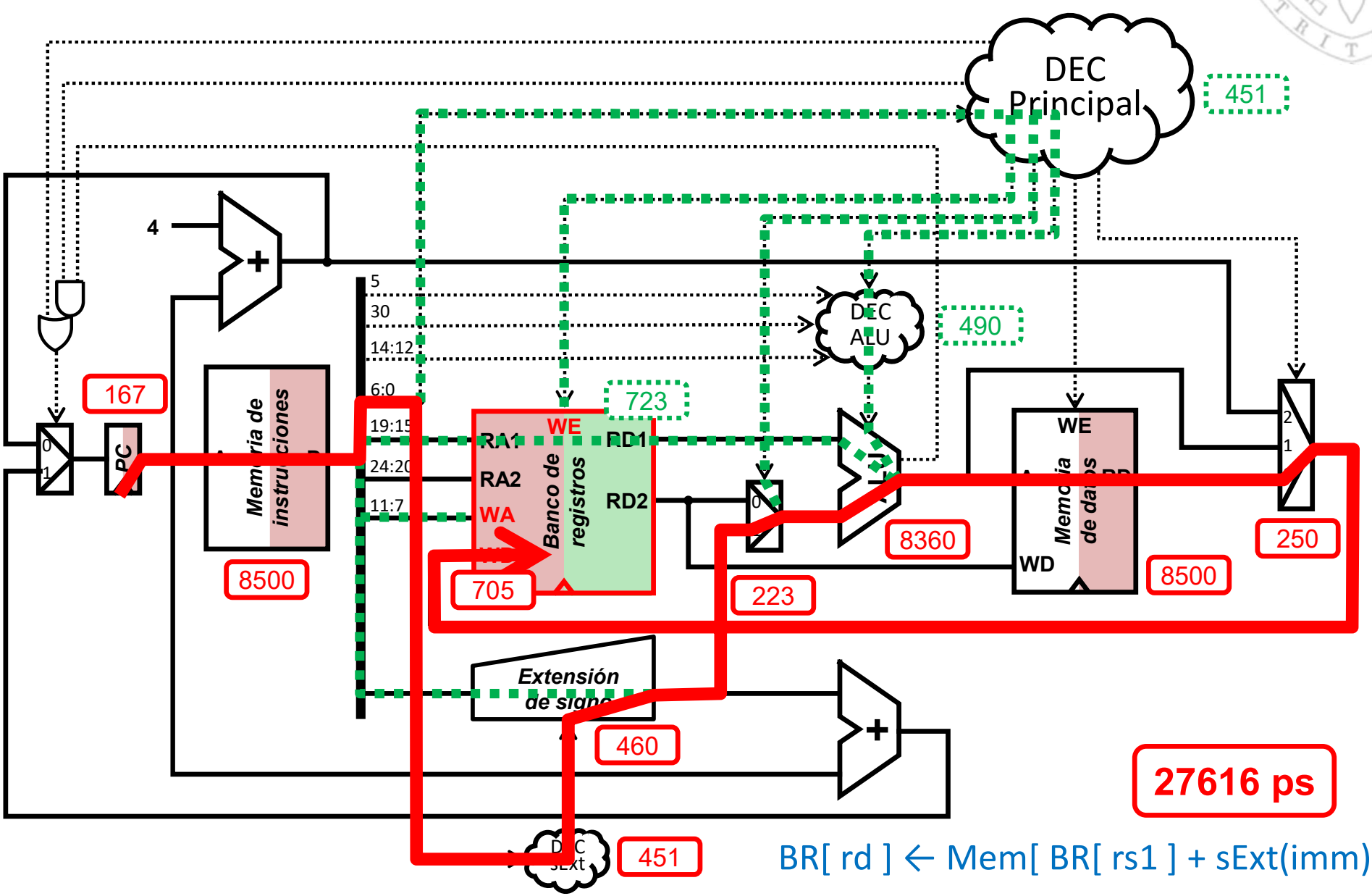
Cálculo del tiempo de ciclo

Incremento del PC: camino crítico



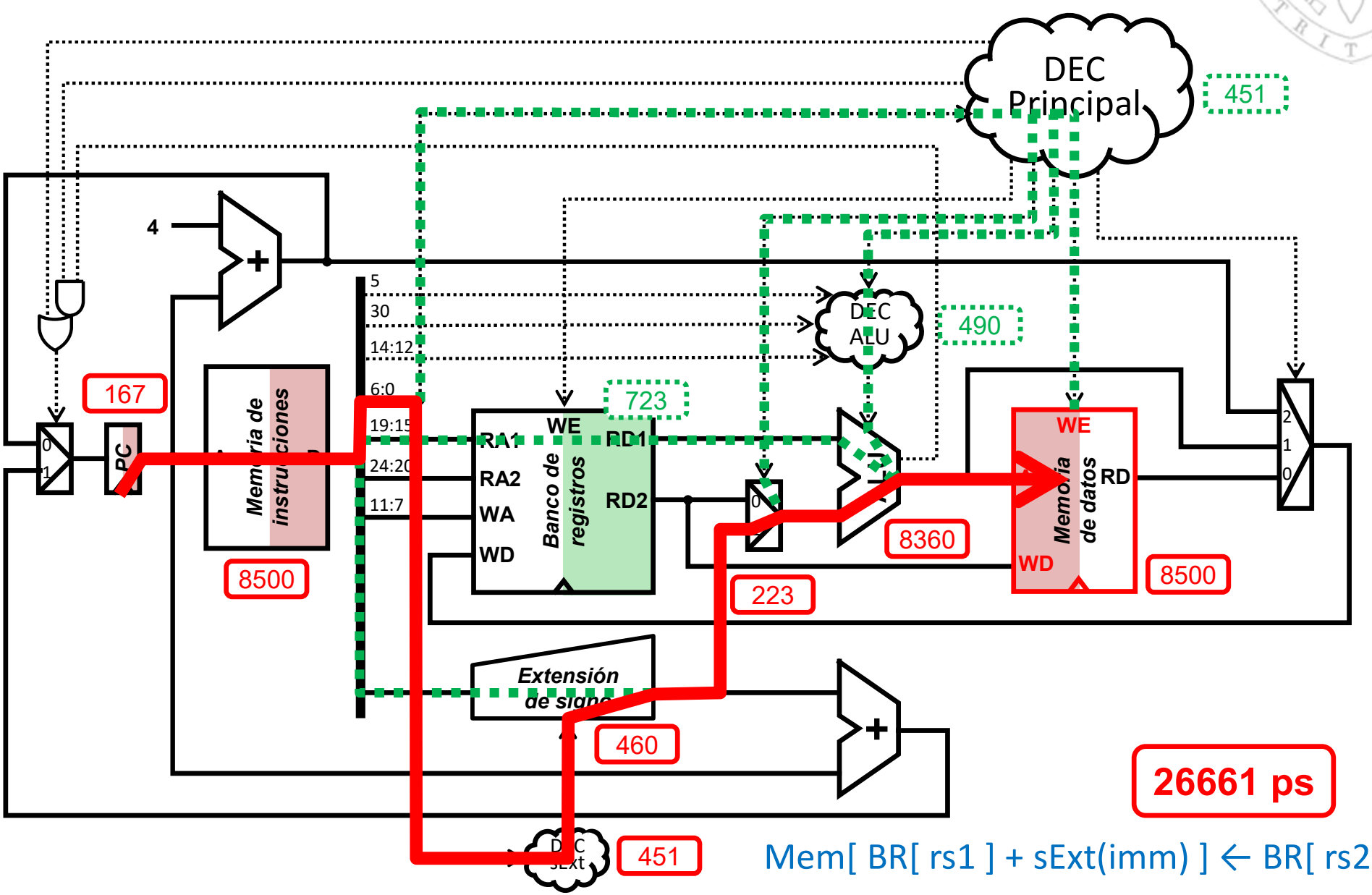
Cálculo del tiempo de ciclo

Instrucción `lw`: camino crítico



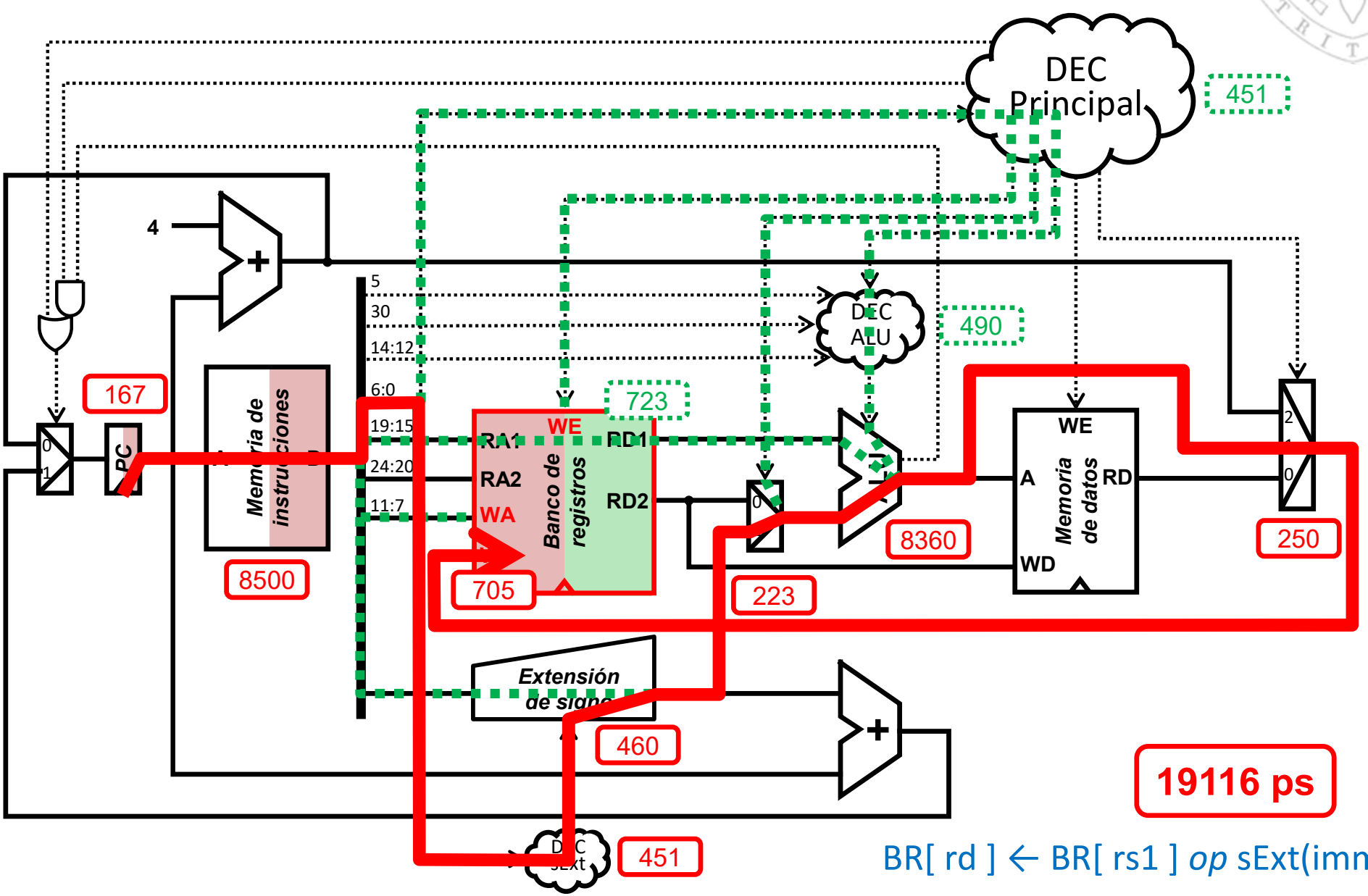
Cálculo del tiempo de ciclo

Instrucción **sw**: camino crítico



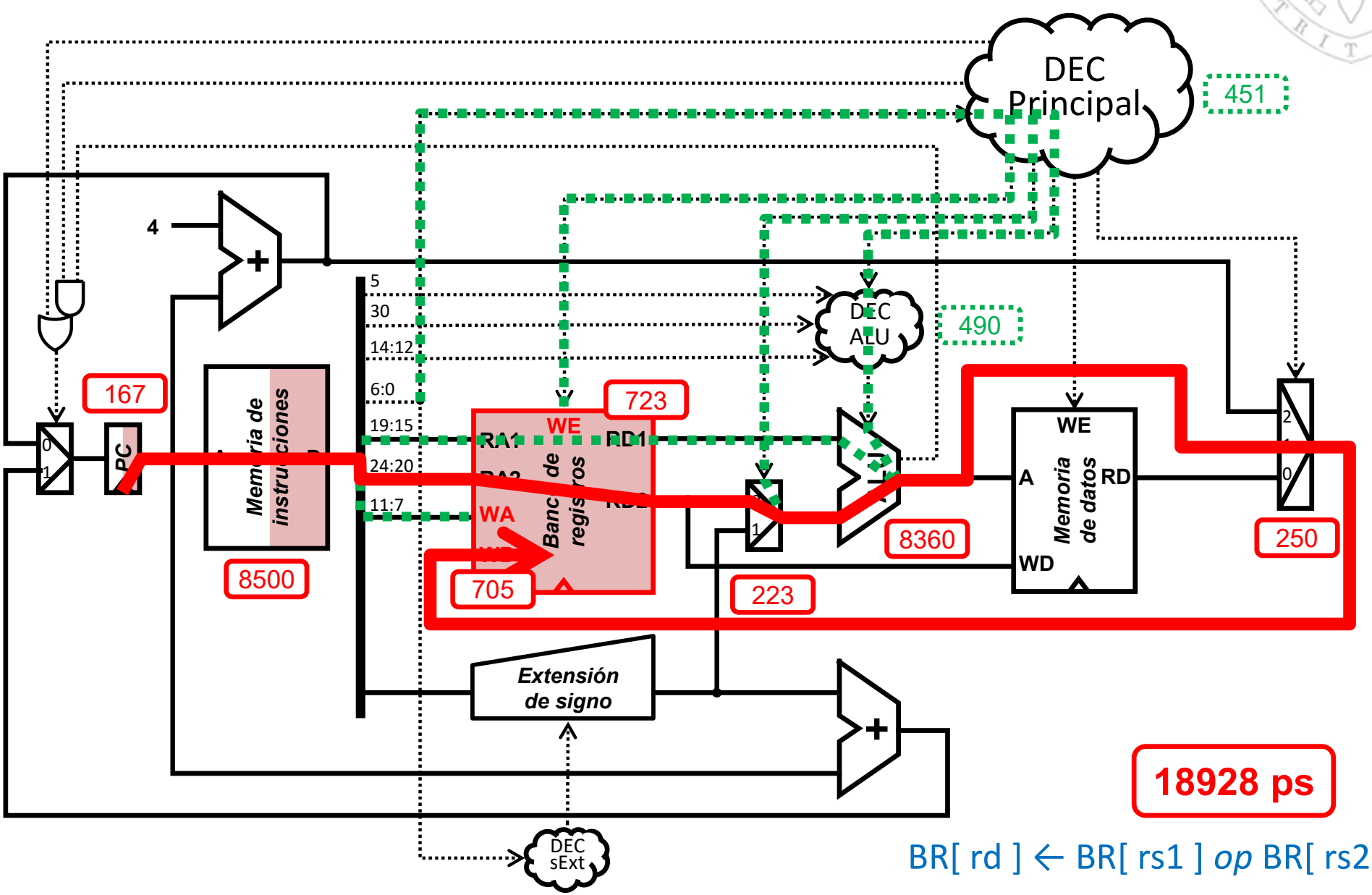
Cálculo del tiempo de ciclo

Instrucciones tipo **addi**: camino crítico



Cálculo del tiempo de ciclo

Instrucciones tipo **add**: camino crítico

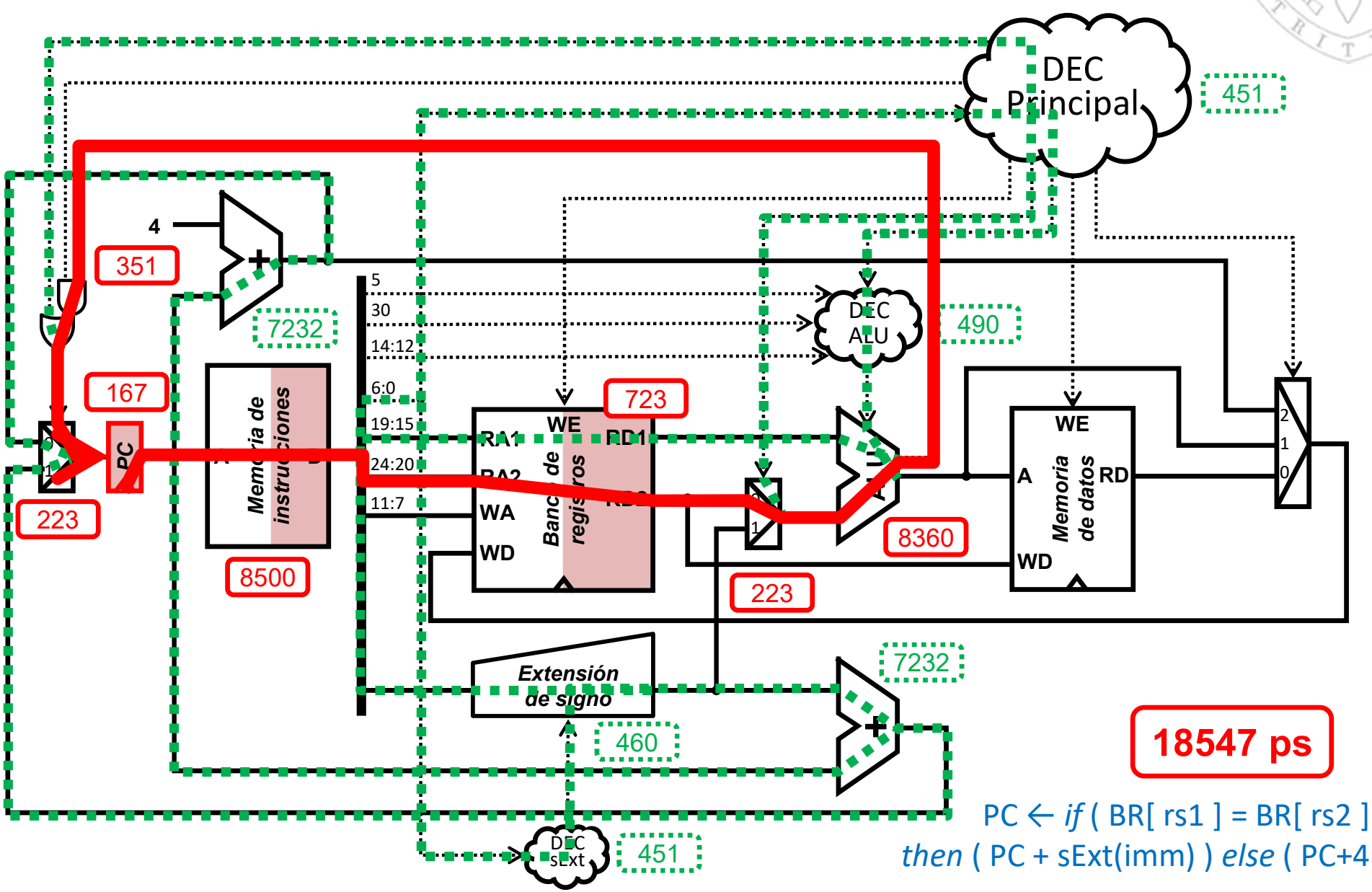


18928 ps

BR[rd] ← BR[rs1] *op* BR[rs2]

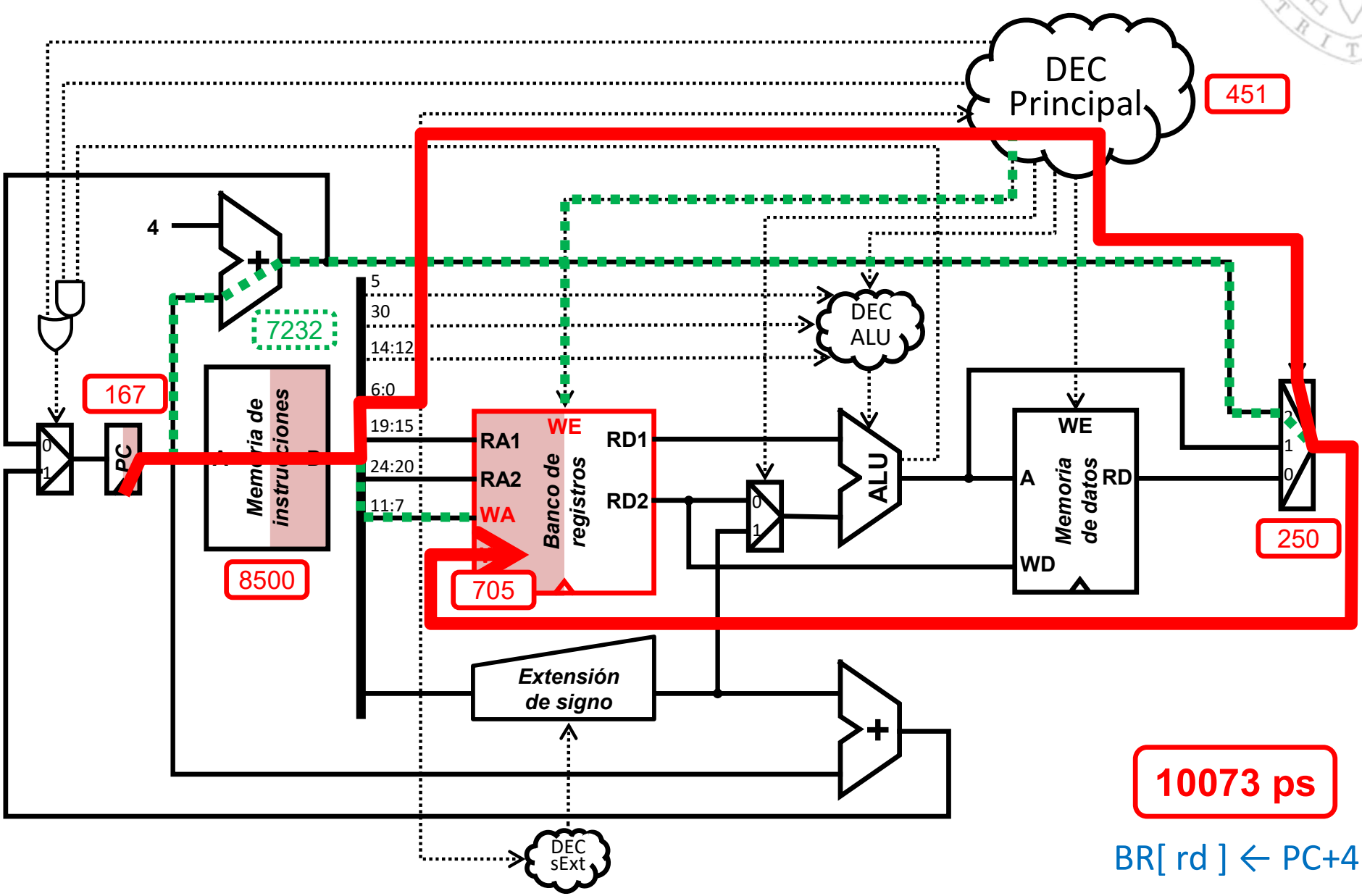
Cálculo del tiempo de ciclo

Instrucción **beq**: camino crítico



Cálculo del tiempo de ciclo

Instrucción `jal` (almacenaje dir. retorno): camino crítico

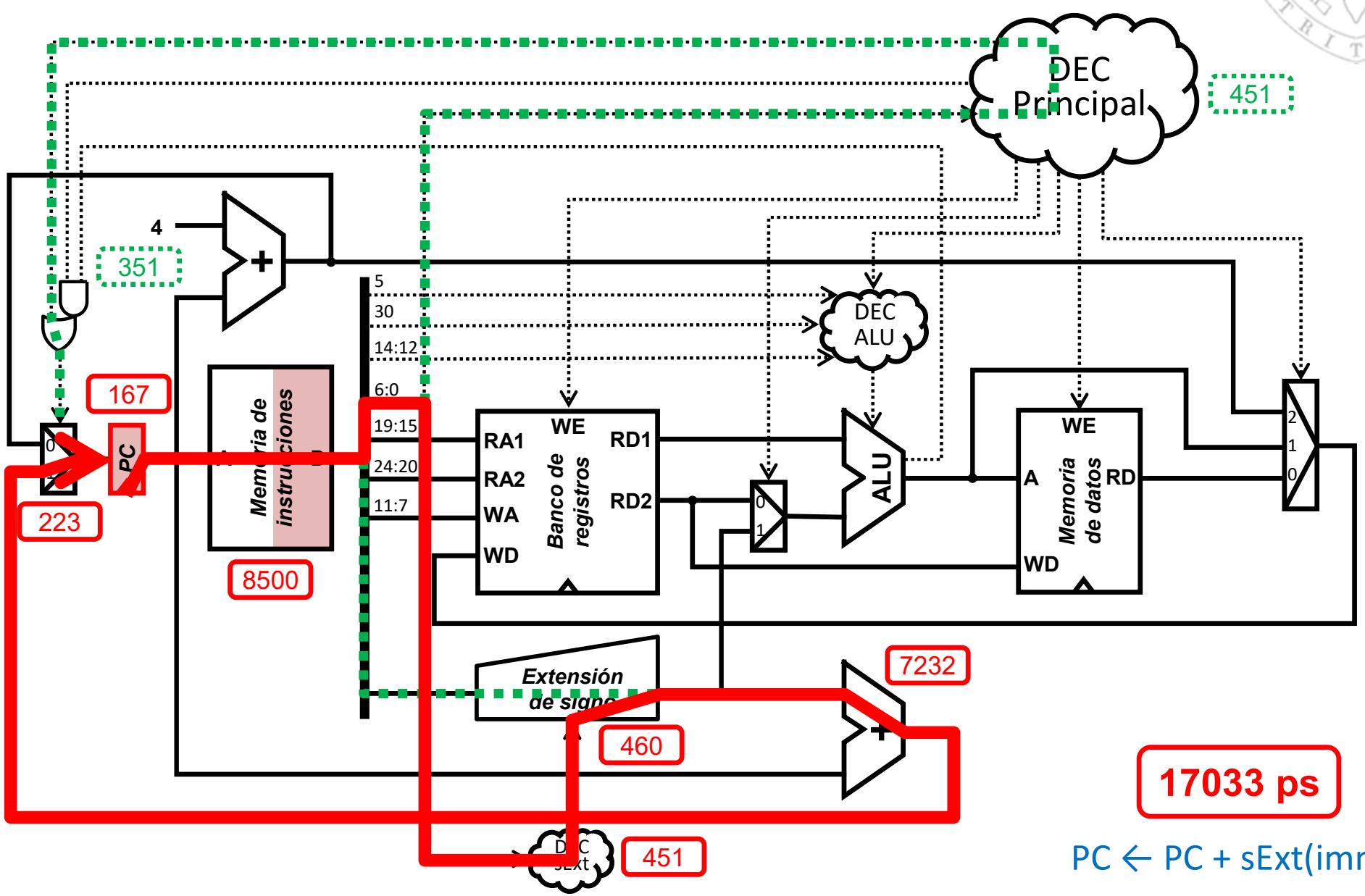


10073 ps

BR[rd] ← PC+4

Cálculo del tiempo de ciclo

Instrucción `jal` (actualización PC): camino crítico



17033 ps

PC ← PC + sExt(imm)

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/