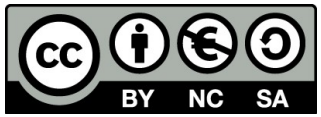
Tema 7:

Diseño segmentado 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.
- ✓ Conflictos estructurales.
- ✓ Conflictos de datos.
- ✓ Conflictos de control.
- ✓ Comparativa: monociclo vs. multiciclo vs. segmentado.
- ✓ Microarquitecturas avanzadas.
- ✓ 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.*

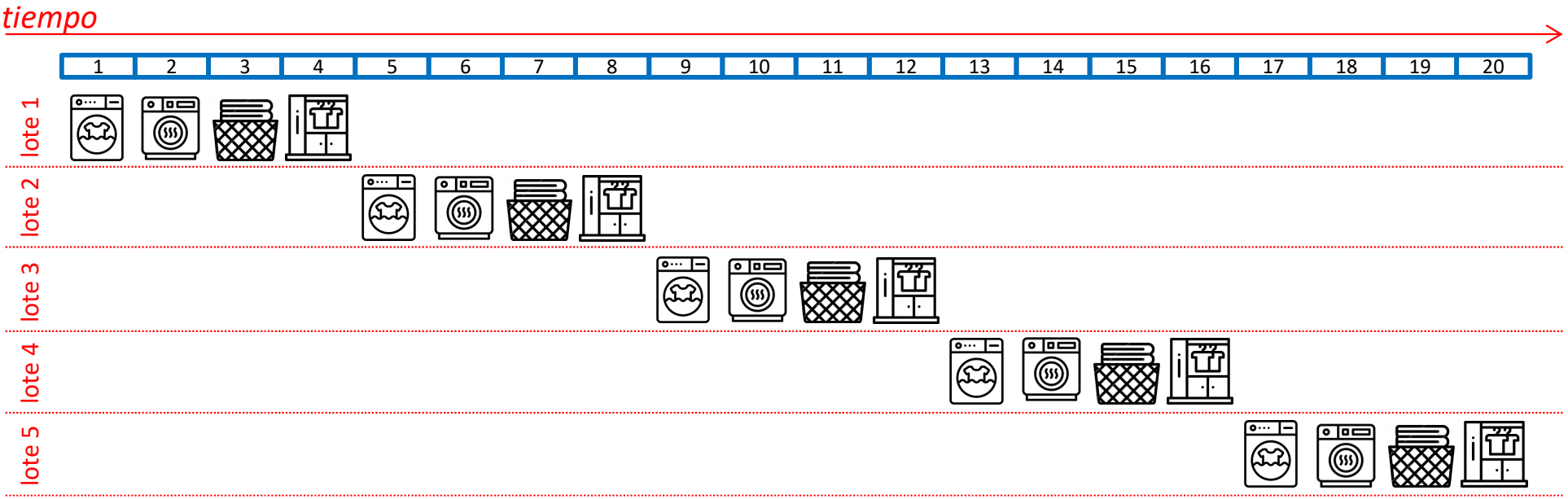
Introducción

- Ningún procesador moderno es monociclo.
  - Esta microarquitectura solo se usó en los primeros computadores.
- Ningún procesador contemporáneo es multiciclo.
  - Esta microarquitectura estuvo en uso hasta finales de los 80s:
    - Mainframes: IBM/360, DEC VAX
    - Microprocesadores: 8088/86 (IBM PC), 68000 (Apple Macintosh), Z80 (Spectrum)
  - En la actualidad solo existe en microcontroladores de bajas prestaciones:
    - 8051, 68HC11, PIC-16
- A partir de los 90s, todos los procesadores se segmentan.
  - Los procesadores actuales presentan microarquitecturas aún más avanzadas pero sustentadas sobre el concepto de segmentación.

Introducción

Concepto de segmentación (i)

- En un hogar lo habitual es tener una lavandería secuencial:
  - 4 etapas de duración similar: lavado, secado, planchado y guardado.

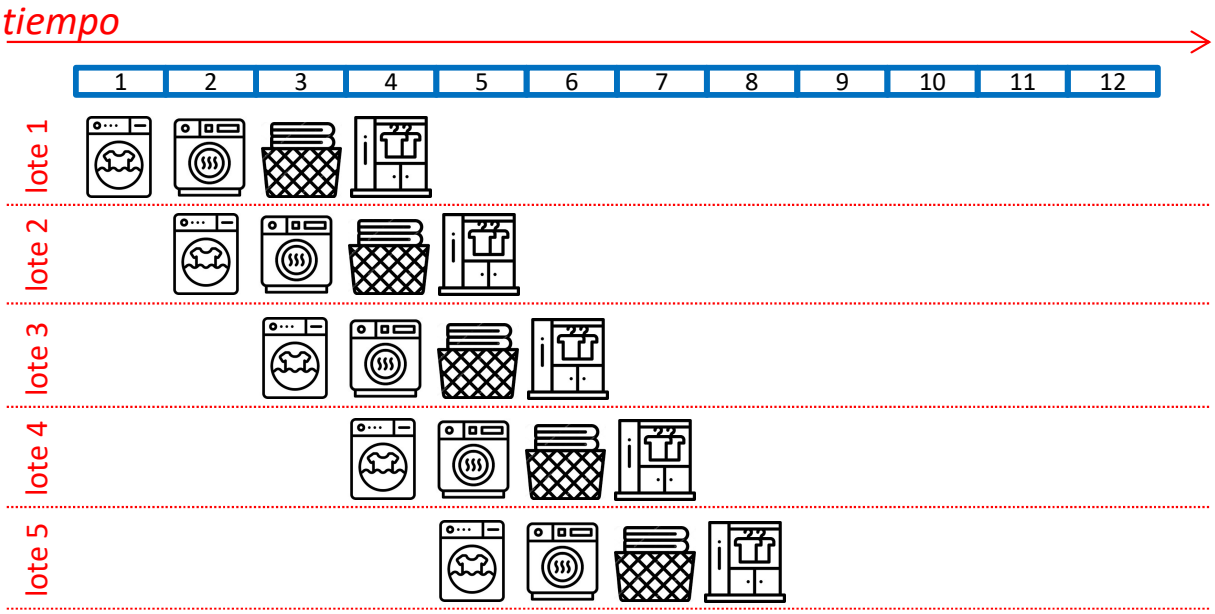


- Cada electrodoméstico está el 75% del tiempo inactivo.
- 1 lote de ropa se procesa en 4 unidades de tiempo.
- 5 lotes se procesan en 4×5 = 20 unidades de tiempo.
- n lotes se procesan en 4·n unidades de tiempo.

Introducción

Concepto de segmentación (ii)

- En una lavandería industrial el proceso es más eficiente:
  - Se empieza con un nuevo lote aunque el anterior no haya finalizado



$$Speedup = \frac{4 \cdot n}{4 + (n - 1)}$$

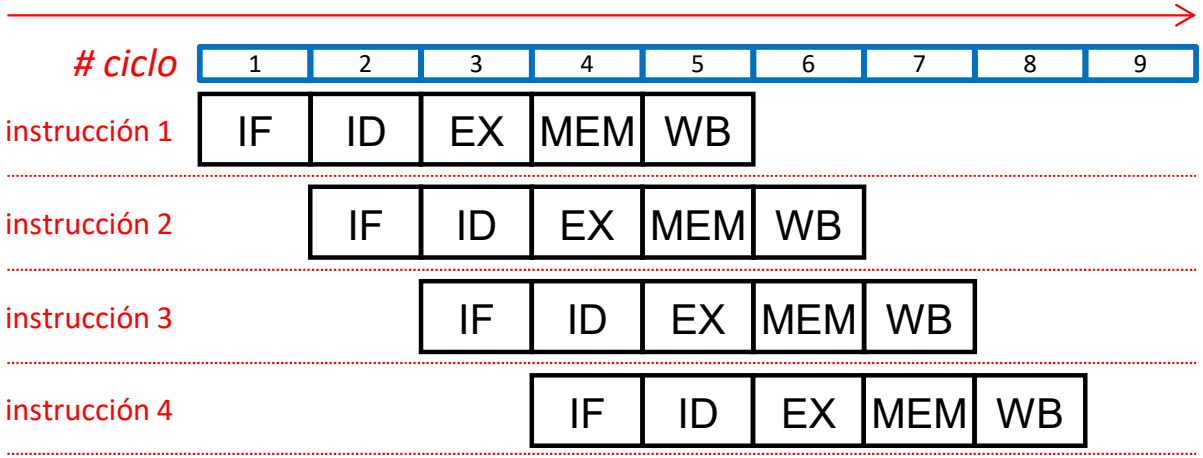
$$\lim_{n \to \infty} \frac{4 \cdot n}{4 + (n - 1)} = 4$$

- Ahora los electrodomésticos están en uso el 100% de su tiempo.
- 1 lote de ropa sigue procesándose en 4 unidades de tiempo.
- 5 lotes se procesan ahora en 4 + (5-1) = 8 unidades de tiempo.
- n lotes se procesan en 4 + (n-1) unidades de tiempo.

Introducción

Concepto de segmentación (iii)

- Un procesador segmentado (*pipelined*) se comporta como la lavandería industrial solapando la ejecución de varias instrucciones.

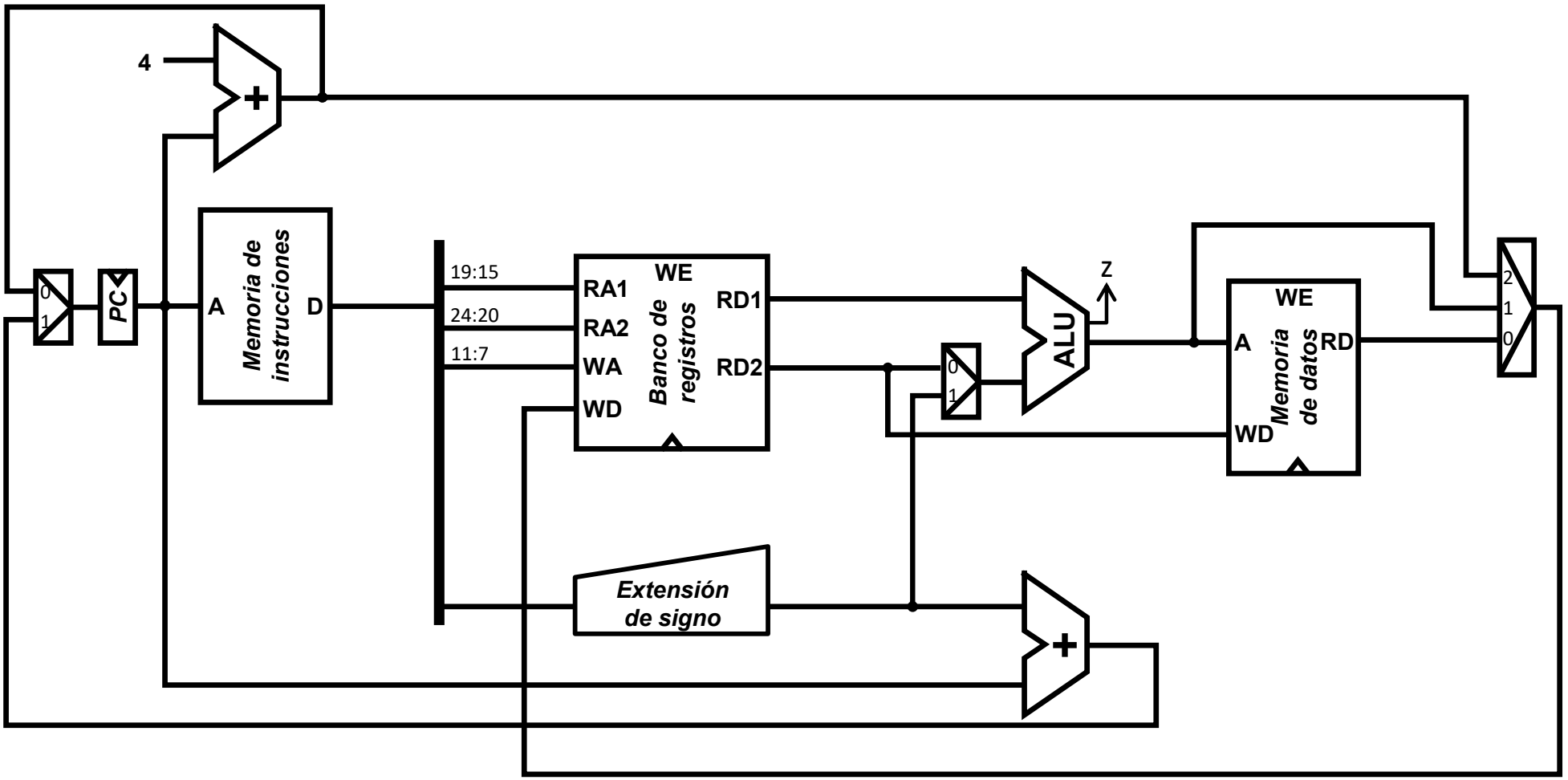


  - En cada ciclo lanza una nueva instrucción sin esperar a que termine la anterior.
  - Cada instrucción pasa por 5 etapas tardando 5 ciclos en ejecutarse:
    - Se dice que la latencia (*latency*) de este procesador es de 5 ciclos.
- El tiempo de ejecución de un programa será mucho menor porque:
  - Varias instrucciones se ejecutan simultáneamente.
  - El tiempo de ciclo podrá ser corto (como en el procesador multiciclo).
  - De manera ideal CPI = 1 (como en el procesador monociclo).

Diseño de la ruta de datos

Ruta de datos RISC-V reducido (i)

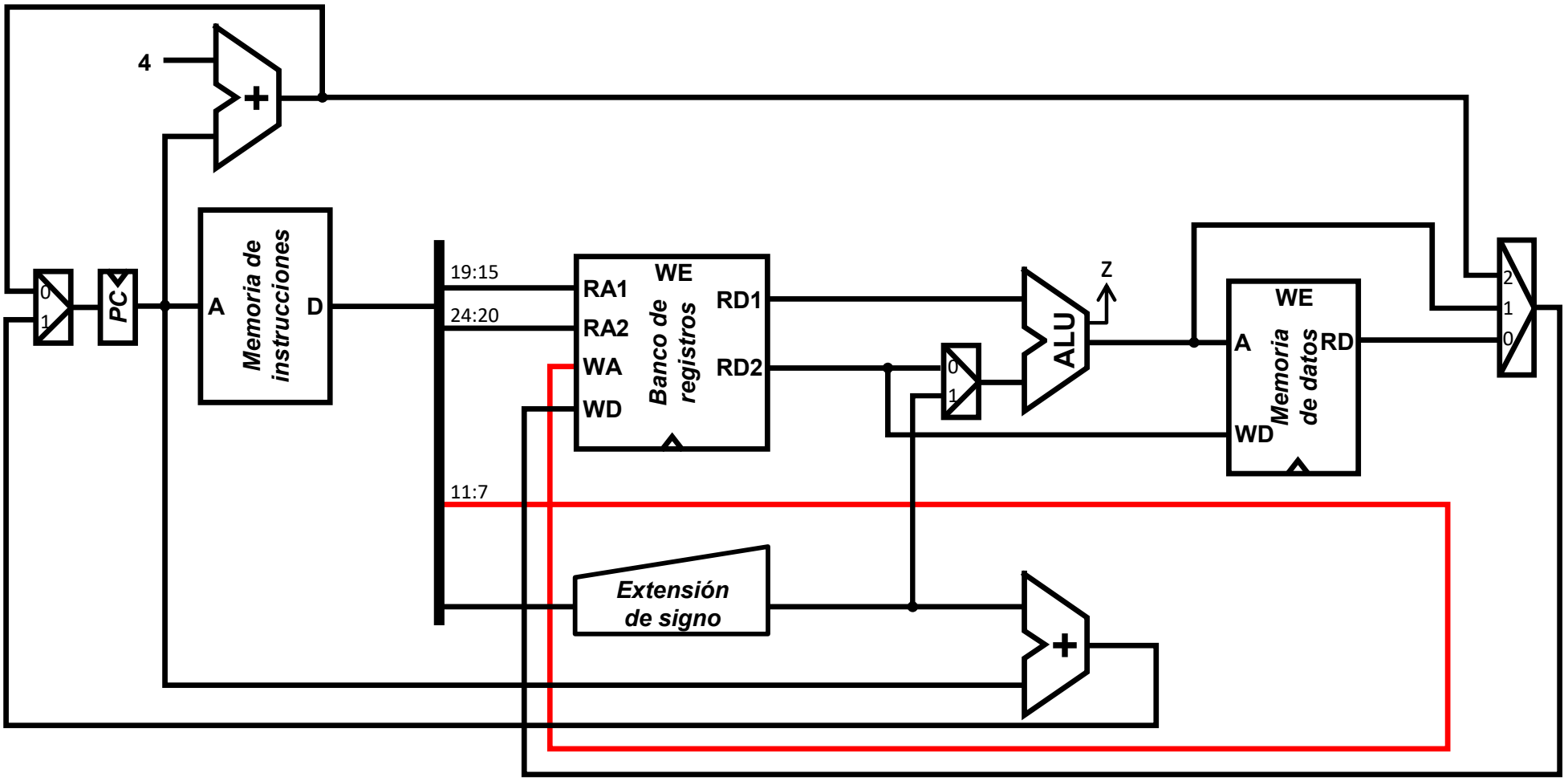
- La ruta de datos del procesador segmentado es:
  - ruta de datos del procesador monociclo + registros de segmentación



Diseño de la ruta de datos

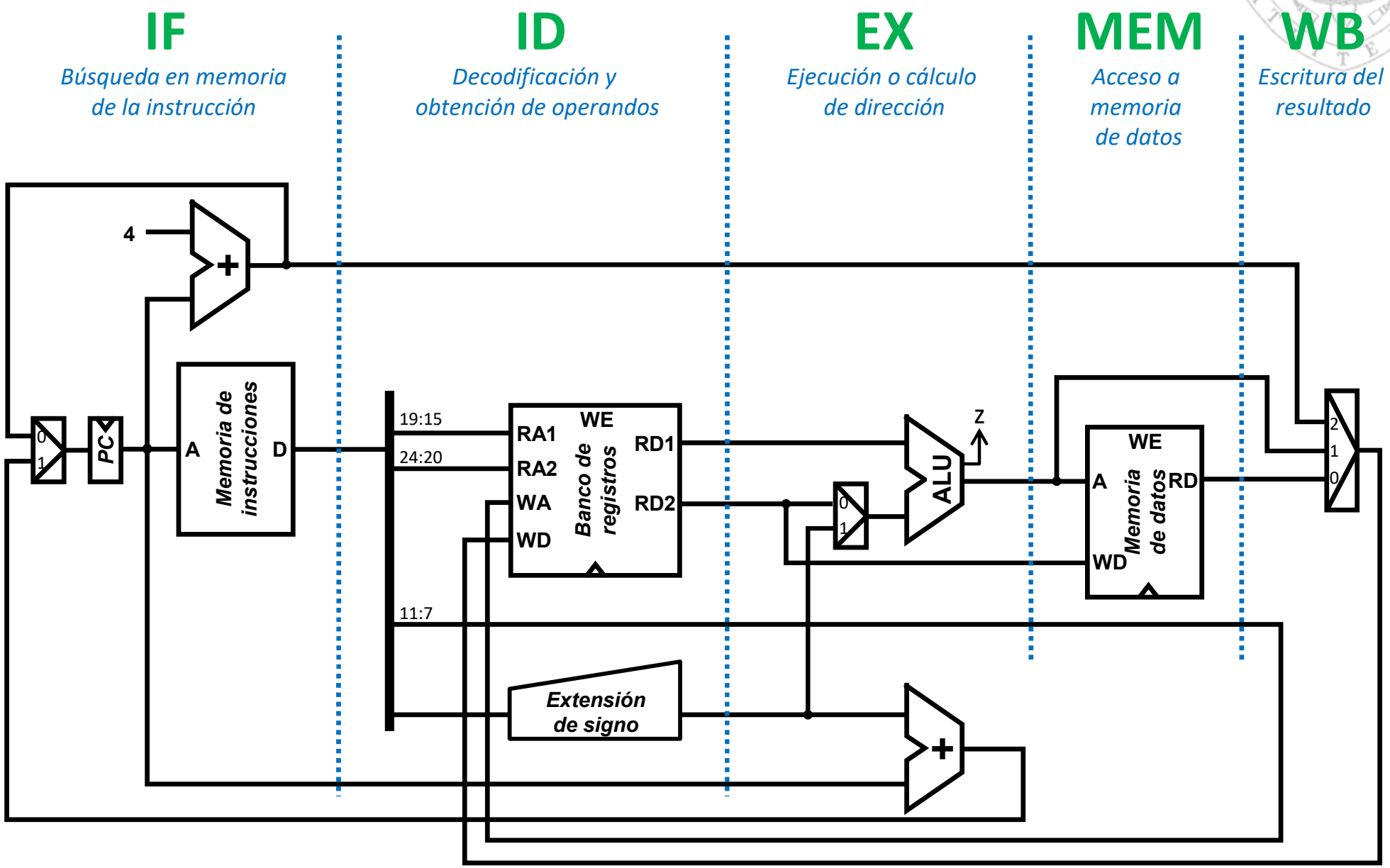
Ruta de datos RISC-V reducido (ii)

- La ruta de datos del procesador segmentado es:
  - ruta de datos del procesador monociclo + registros de segmentación



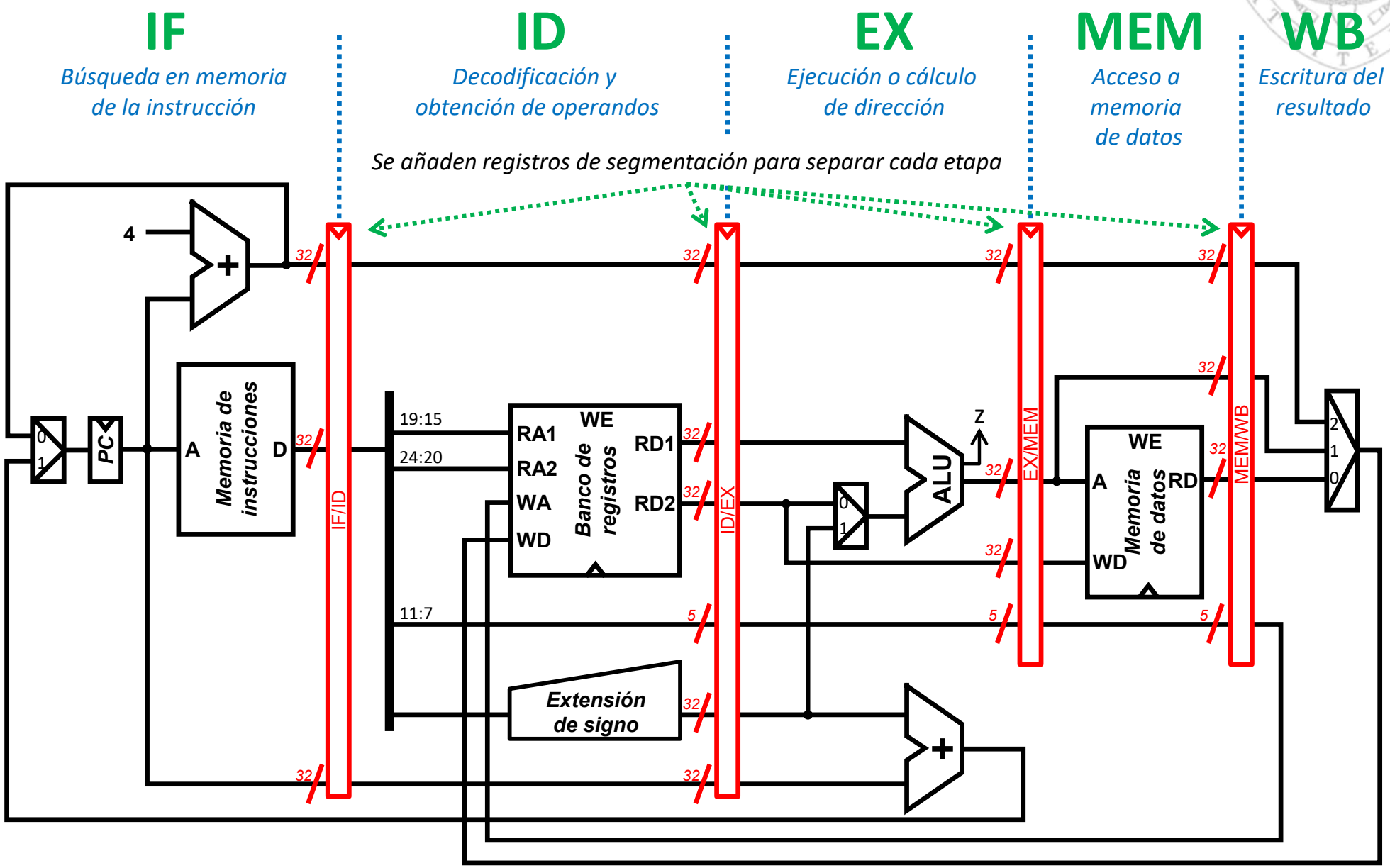
Diseño de la ruta de datos

Ruta de datos RISC-V reducido (iii)



Diseño de la ruta de datos

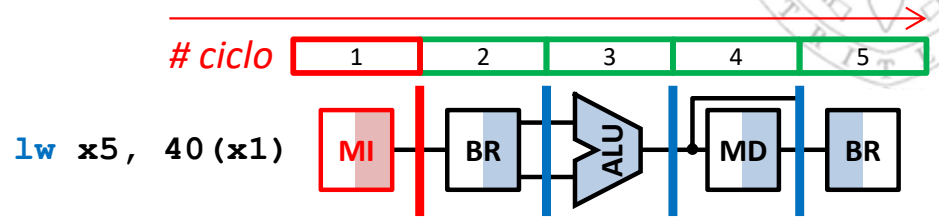
Ruta de datos RISC-V reducido (iv)

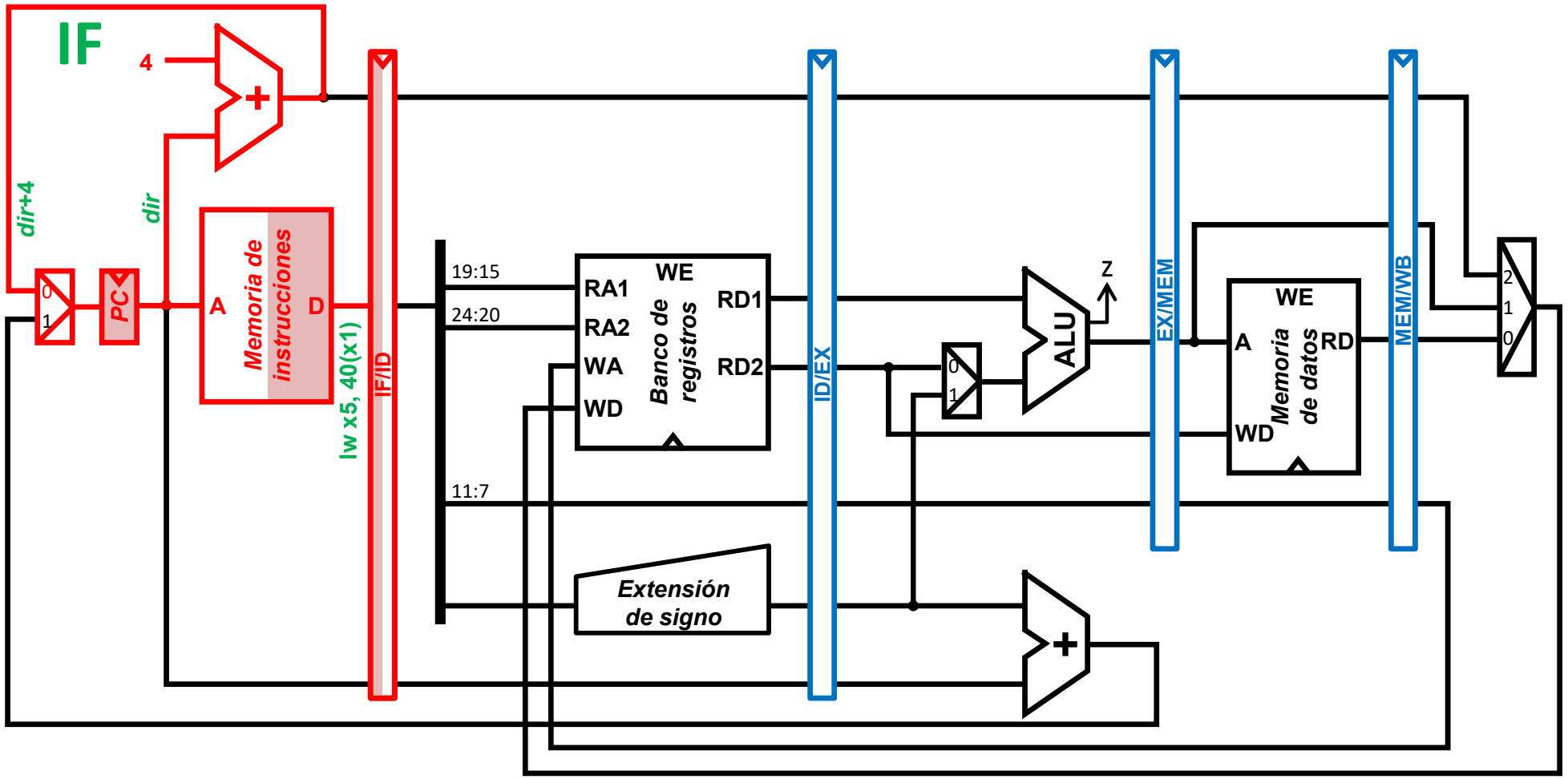


Diseño de la ruta de datos

Instrucción `lw`: etapa IF

La instrucción **`lw`** *se ejecuta en 5 ciclos usando recursos en todas las etapas*

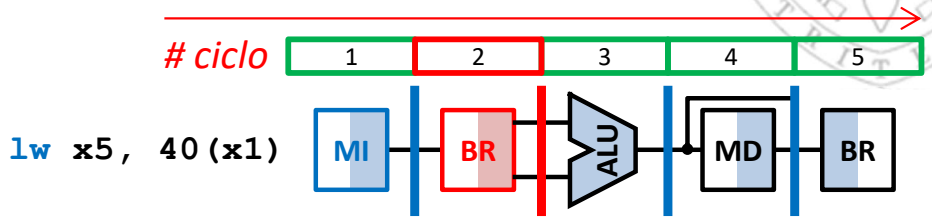


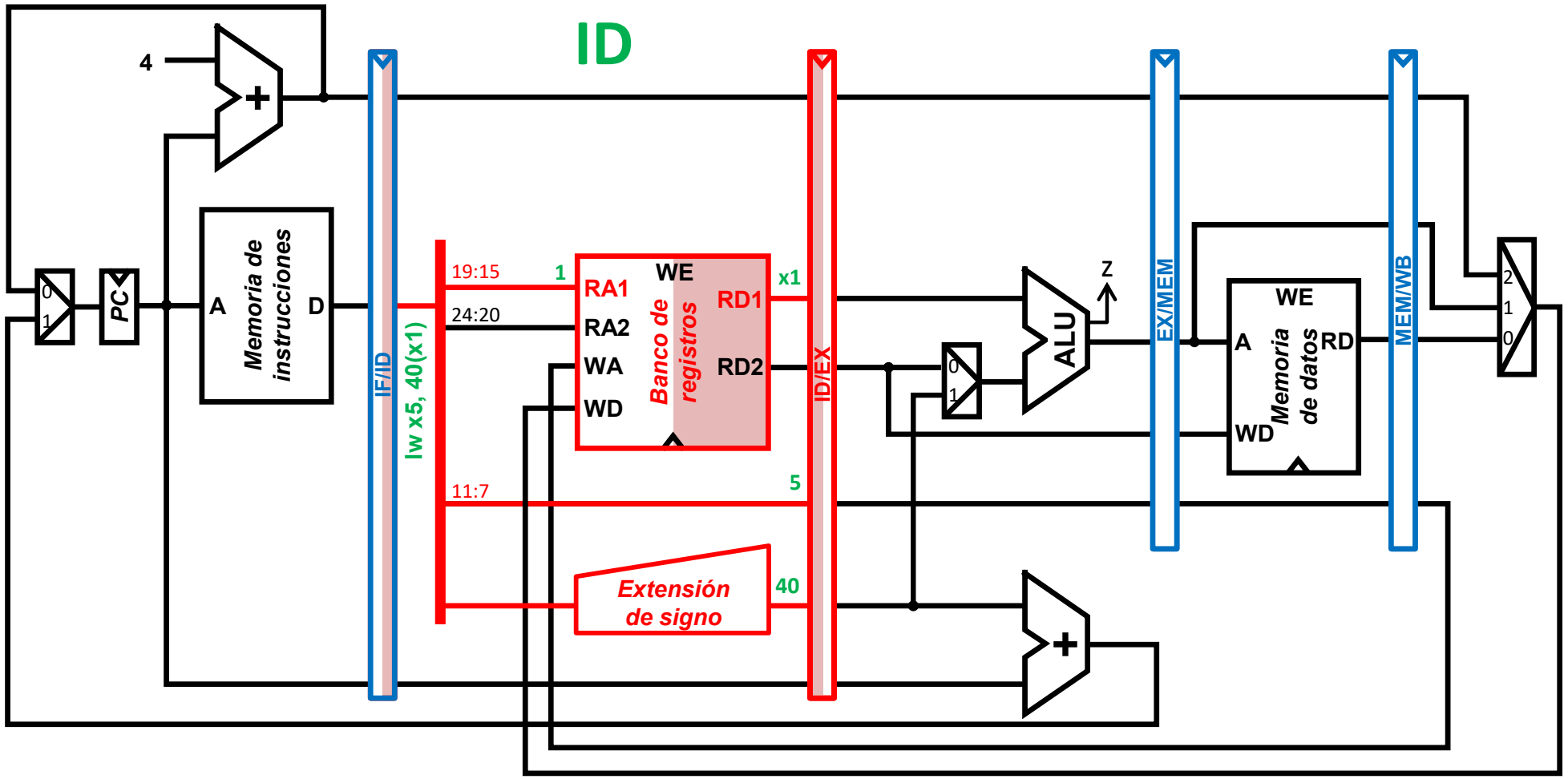


Diseño de la ruta de datos

Instrucción `lw`: etapa ID

La instrucción **`lw`** *se ejecuta en* *5 ciclos* *usando recursos en todas las etapas*

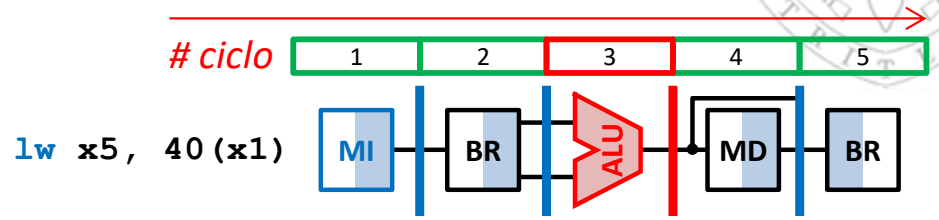


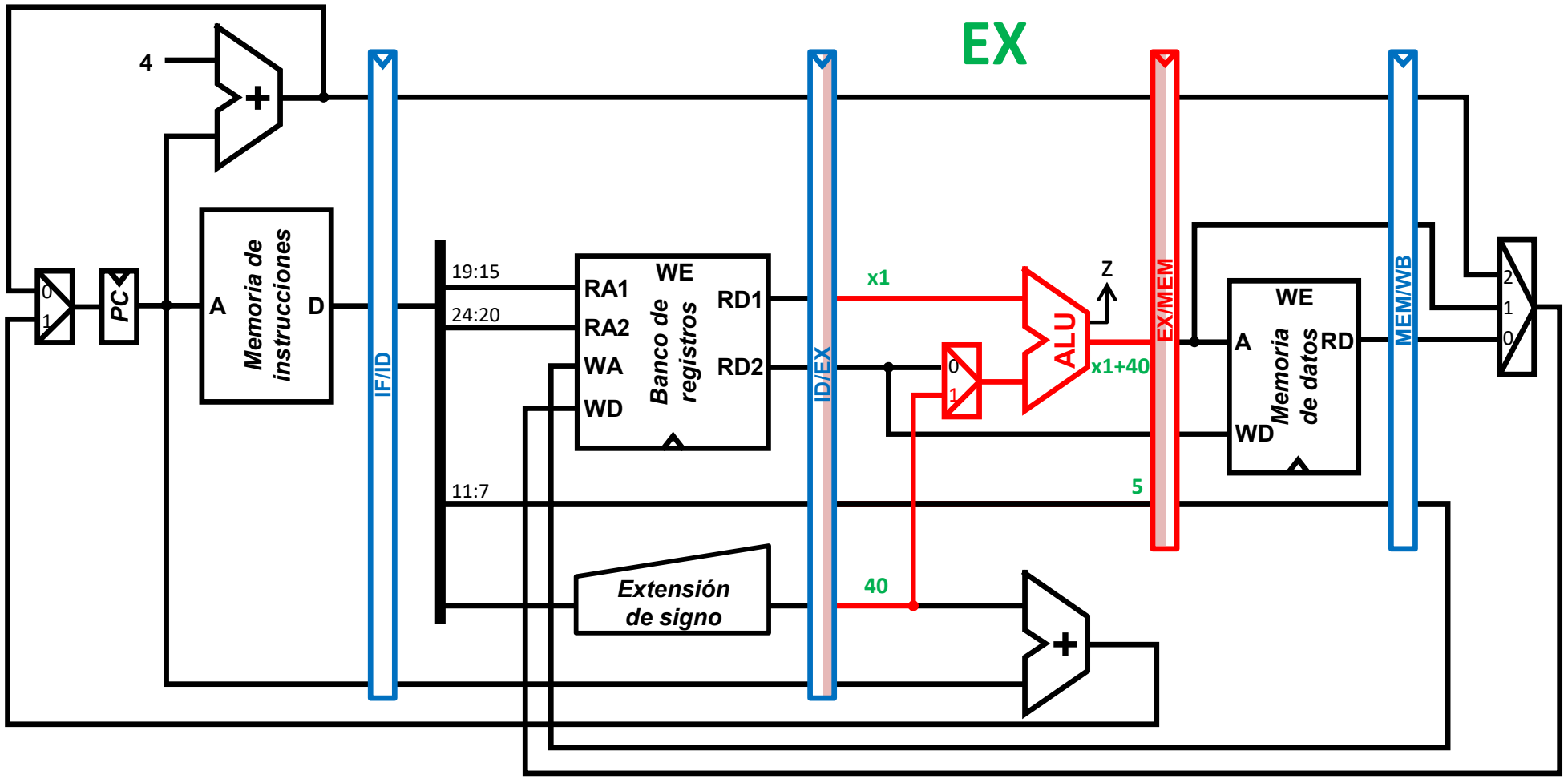


Diseño de la ruta de datos

Instrucción `lw`: etapa EX

La instrucción **`lw`** *se ejecuta en 5 ciclos usando recursos en todas las etapas*

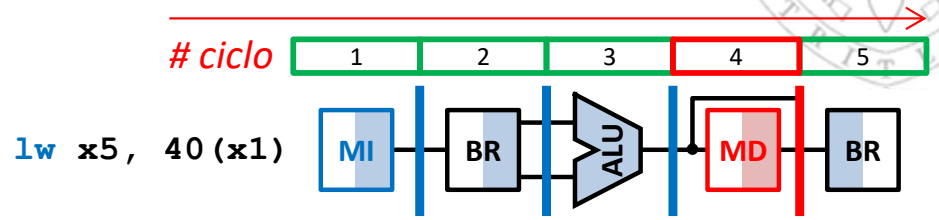


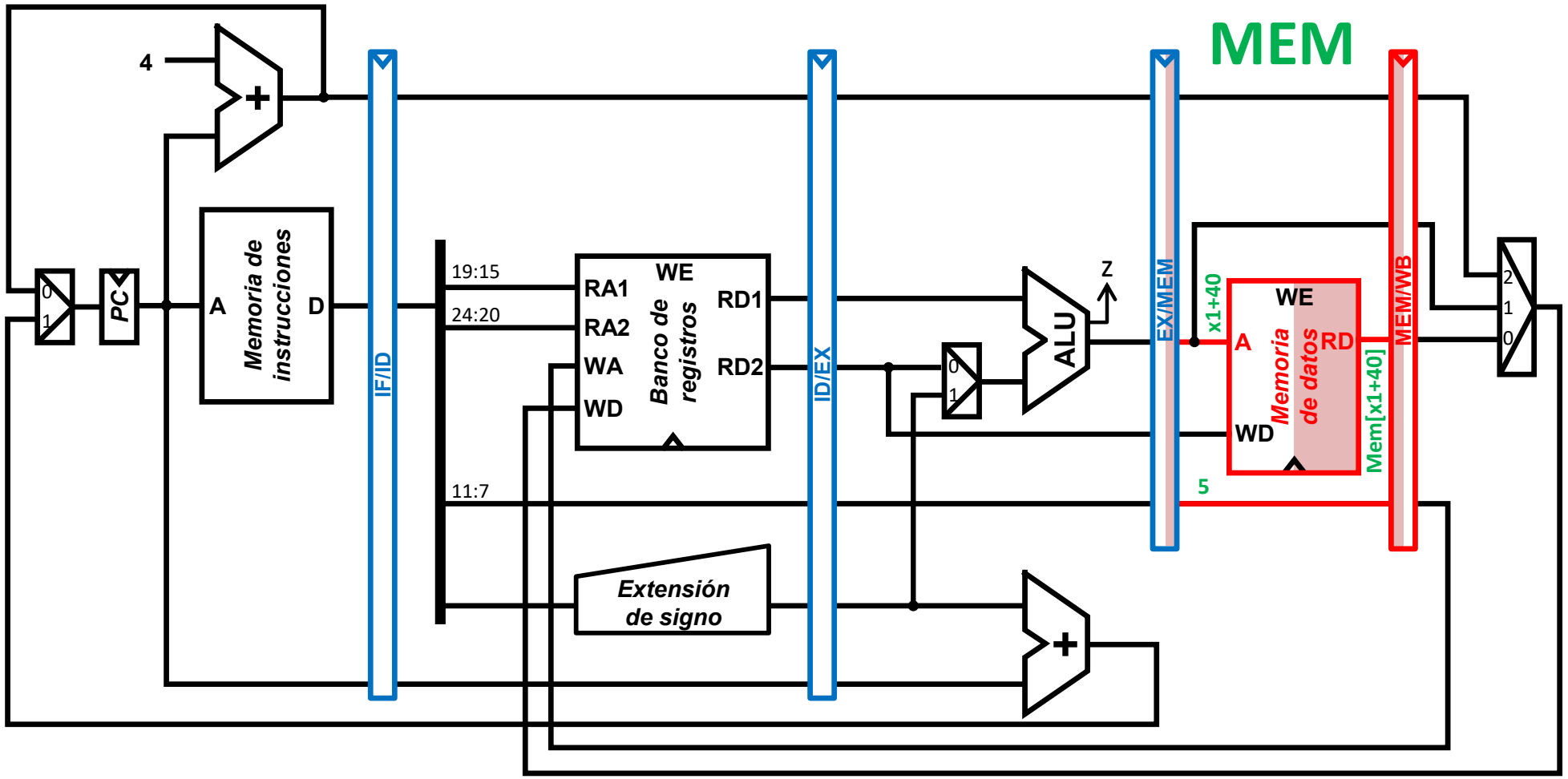


Diseño de la ruta de datos

Instrucción `lw`: etapa MEM

La instrucción **`lw`** *se ejecuta en 5 ciclos usando recursos en todas las etapas*

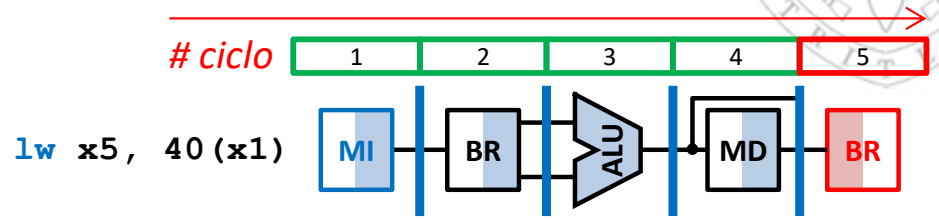


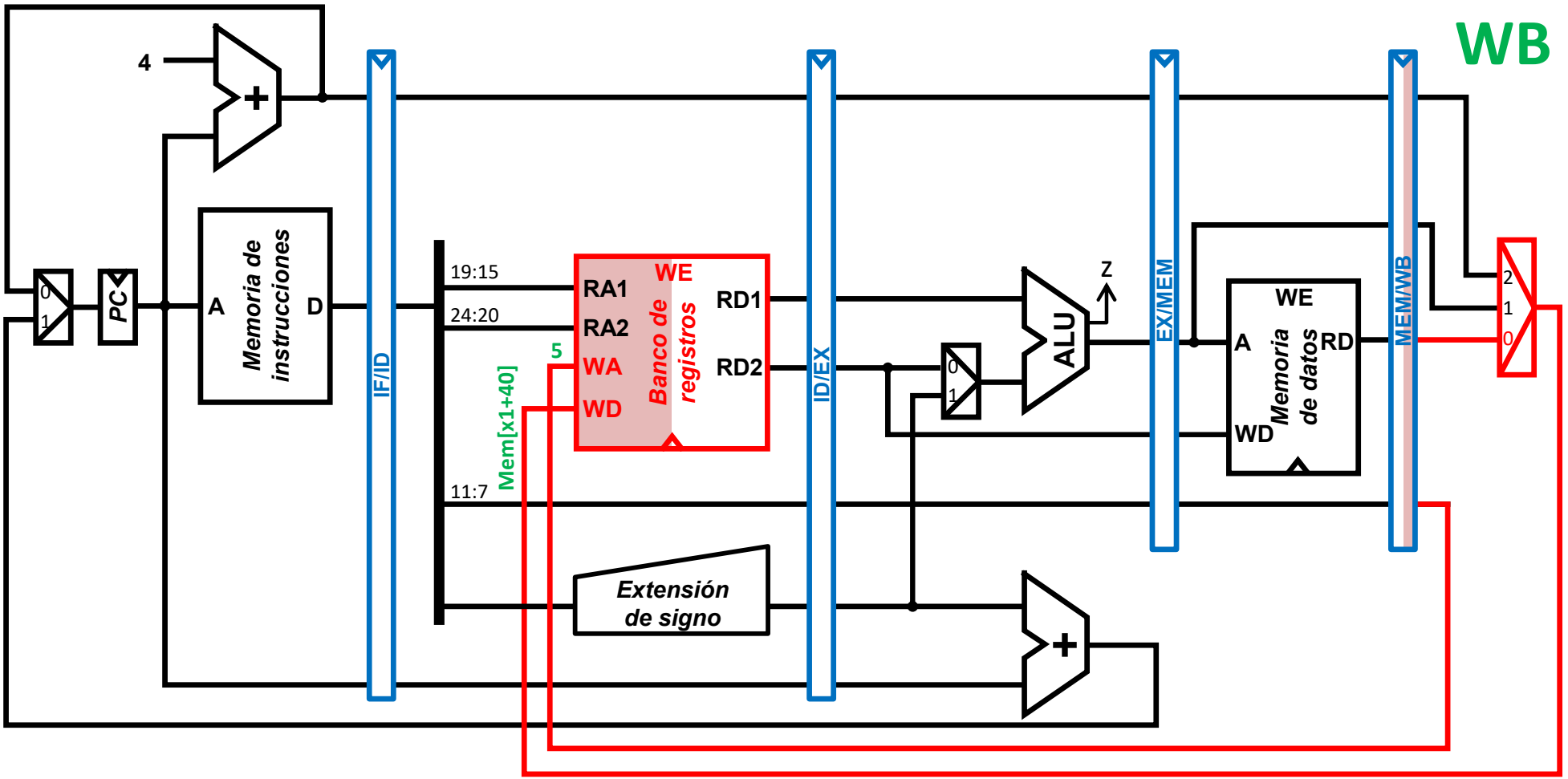


Diseño de la ruta de datos

Instrucción `lw`: etapa WB

La instrucción **`lw`** *se ejecuta en 5 ciclos usando recursos en todas las etapas*

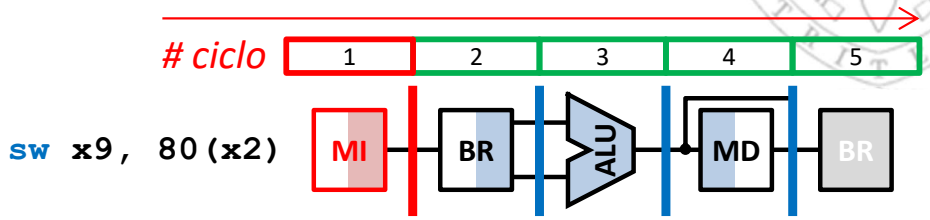


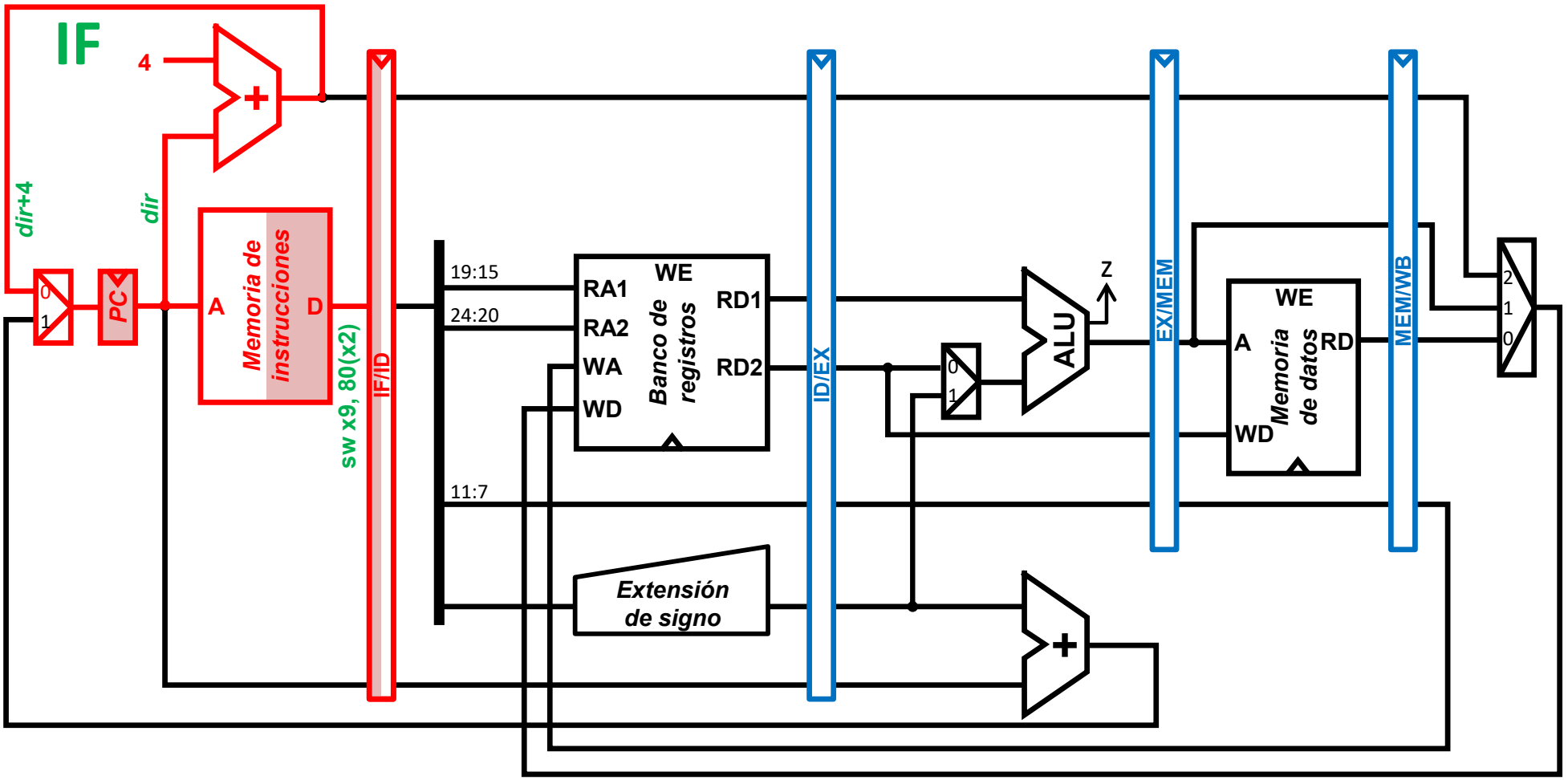


Diseño de la ruta de datos

Instrucción **sw**: etapa IF

La instrucción **sw** *se ejecuta en 5 ciclos sin usar el BR en la etapa WB*

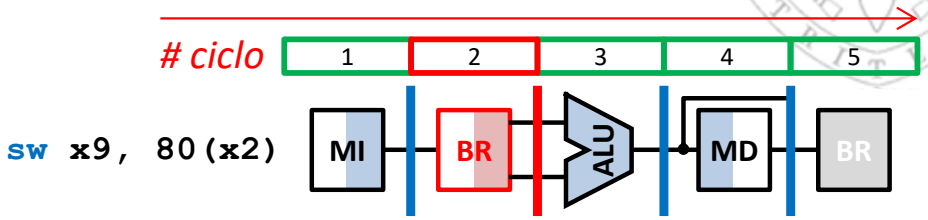


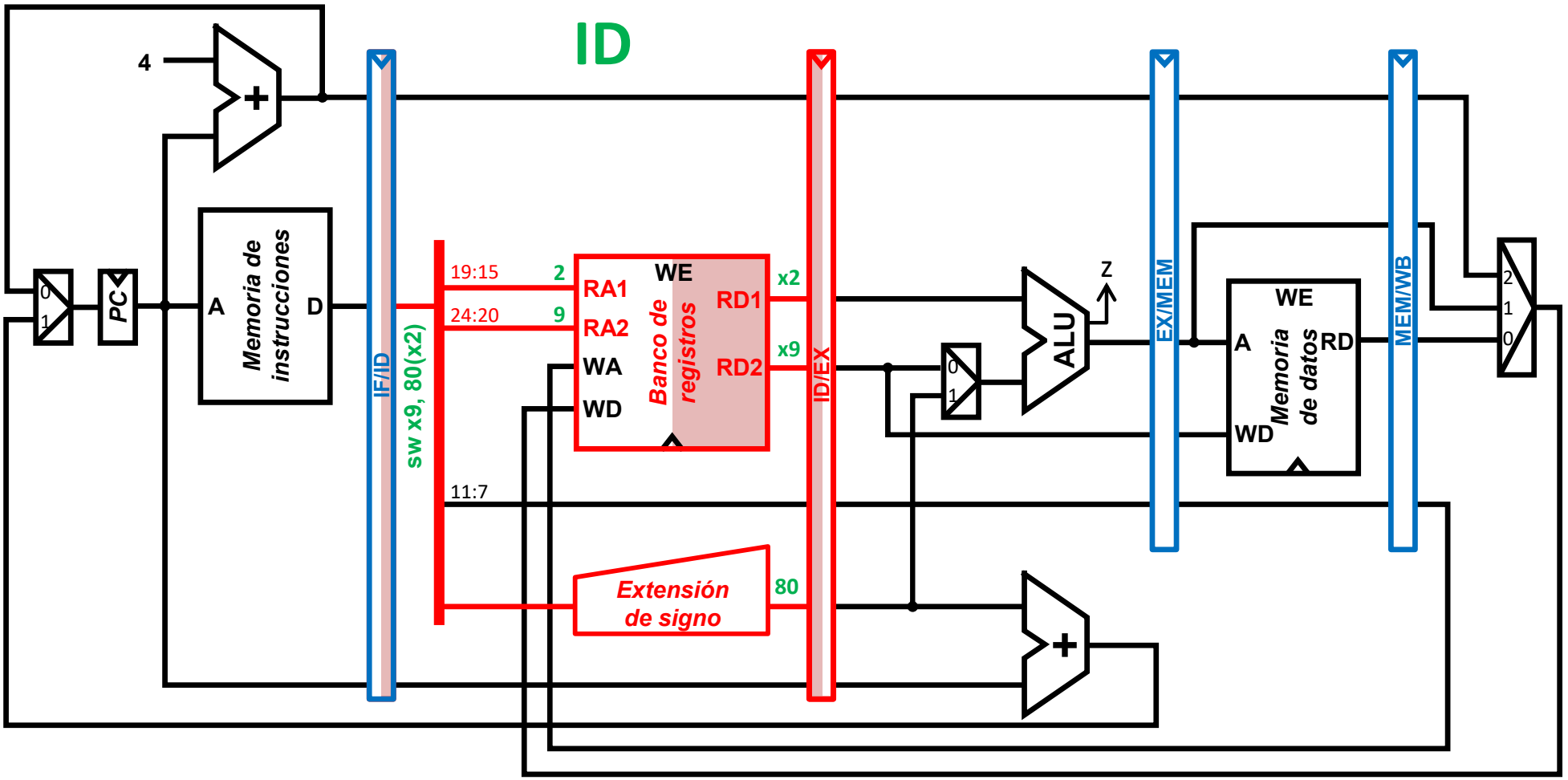


Diseño de la ruta de datos

Instrucción **sw**: etapa ID

La instrucción **sw** *se ejecuta en* <u>*5 ciclos*</u>
sin usar el BR en la etapa WB

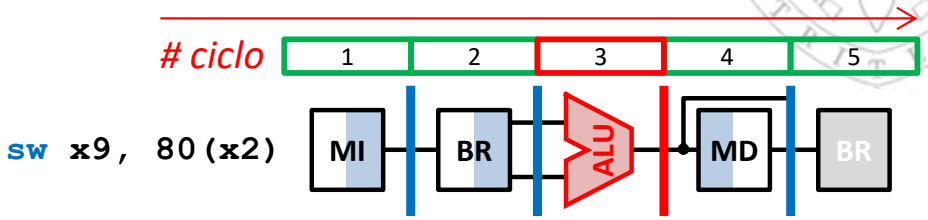


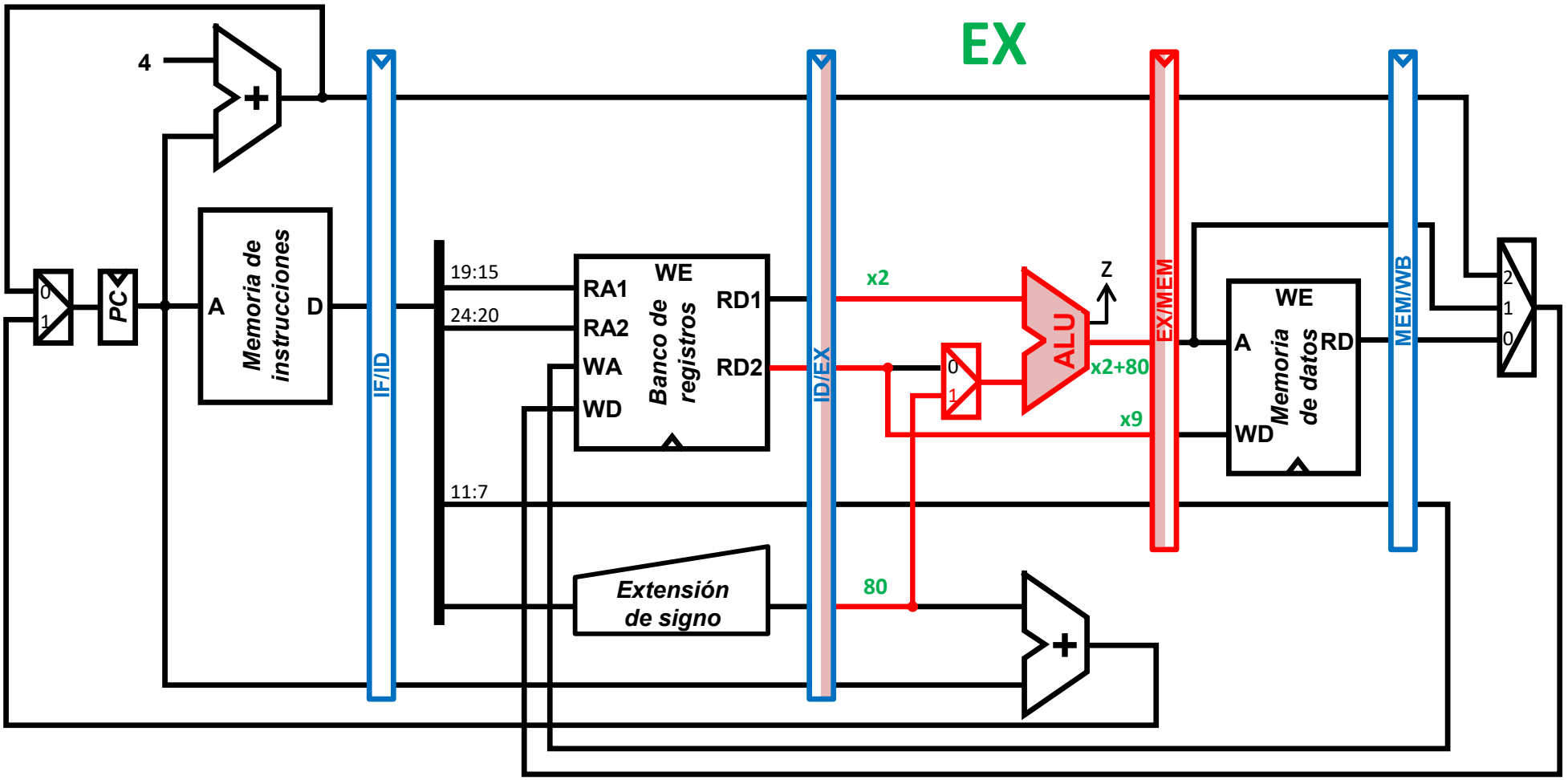


Diseño de la ruta de datos

Instrucción **sw**: etapa EX

La instrucción **sw** *se ejecuta en 5 ciclos sin usar el BR en la etapa WB*

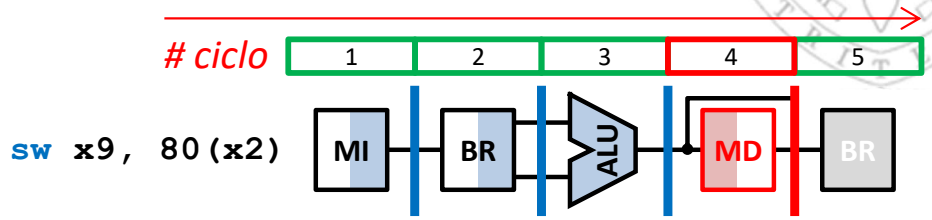


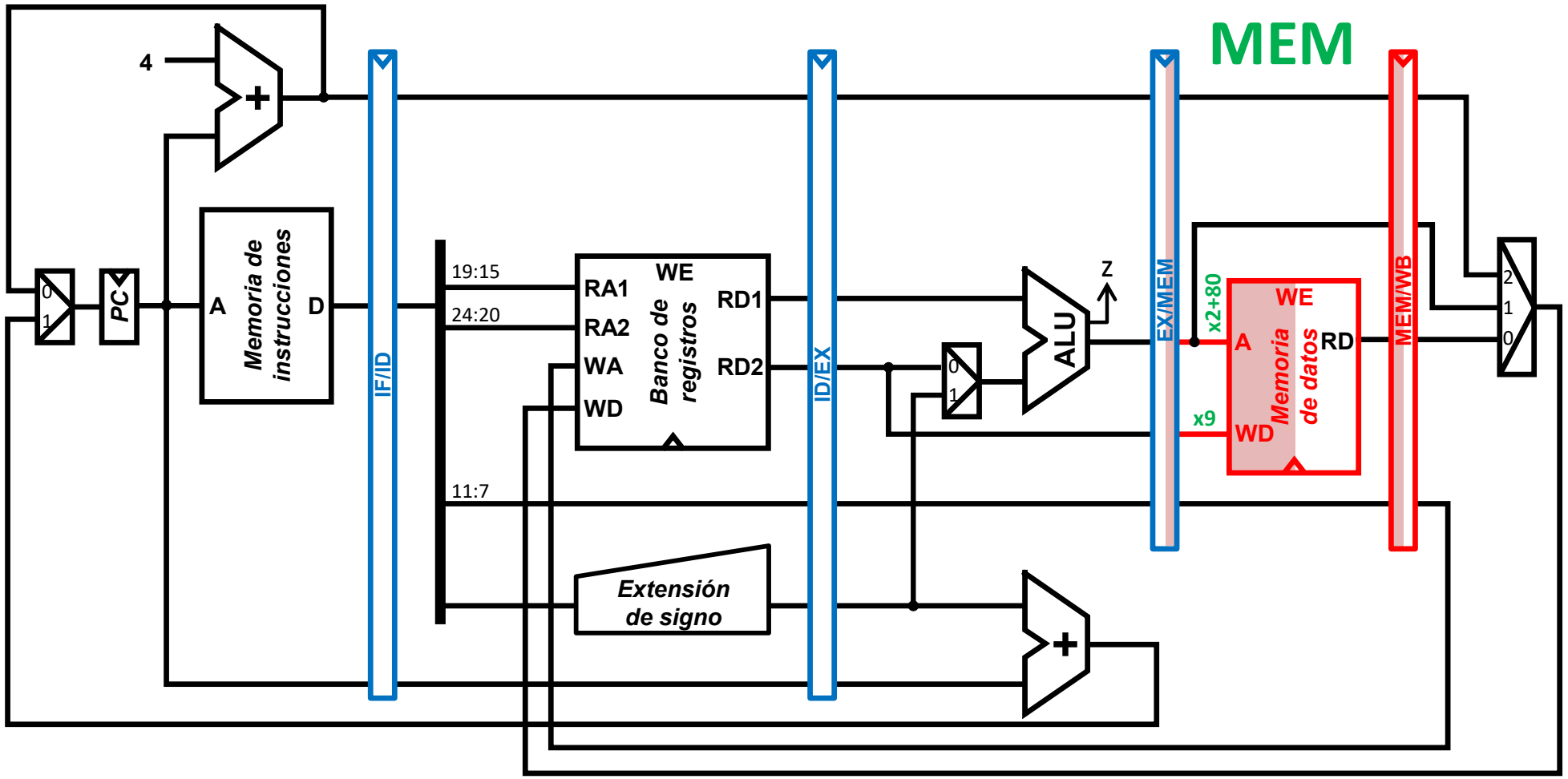


Diseño de la ruta de datos

Instrucción **sw**: etapa MEM

La instrucción **sw** *se ejecuta en* *5 ciclos* *sin usar el BR en la etapa WB*

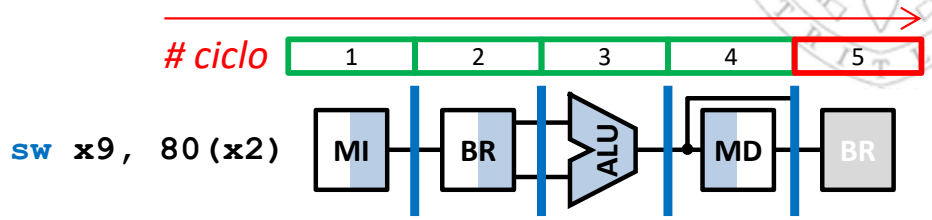


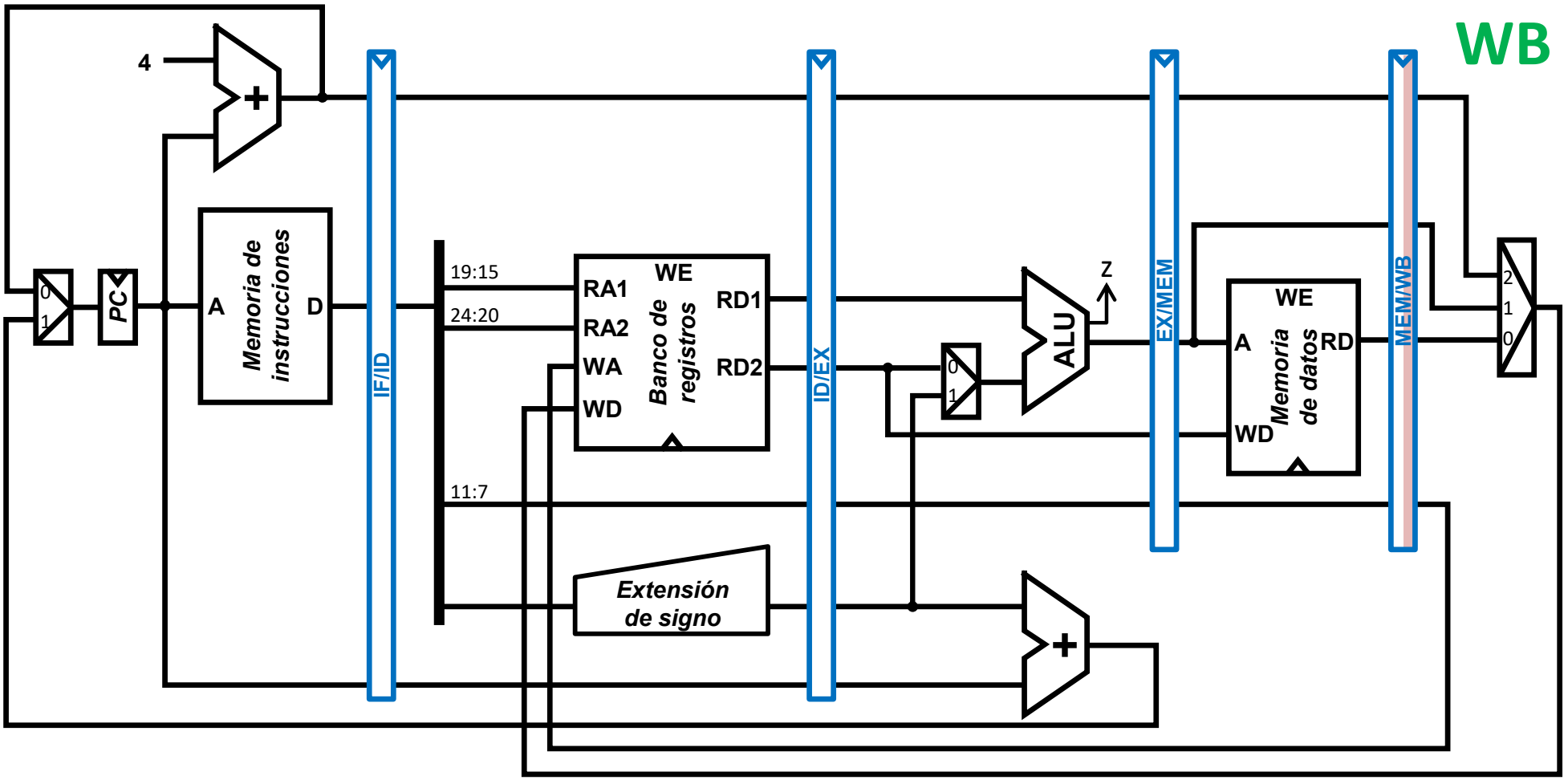


Diseño de la ruta de datos

Instrucción **sw**: etapa WB

La instrucción **sw** *se ejecuta en* *5 ciclos* *sin usar el BR en la etapa WB*

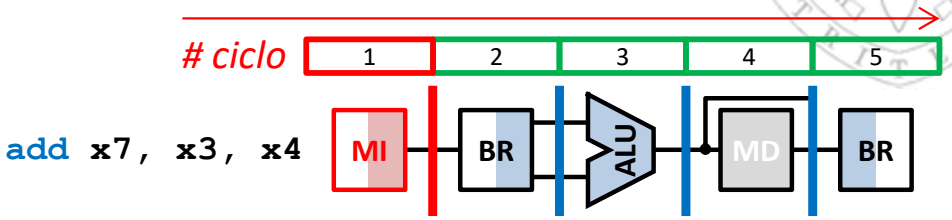


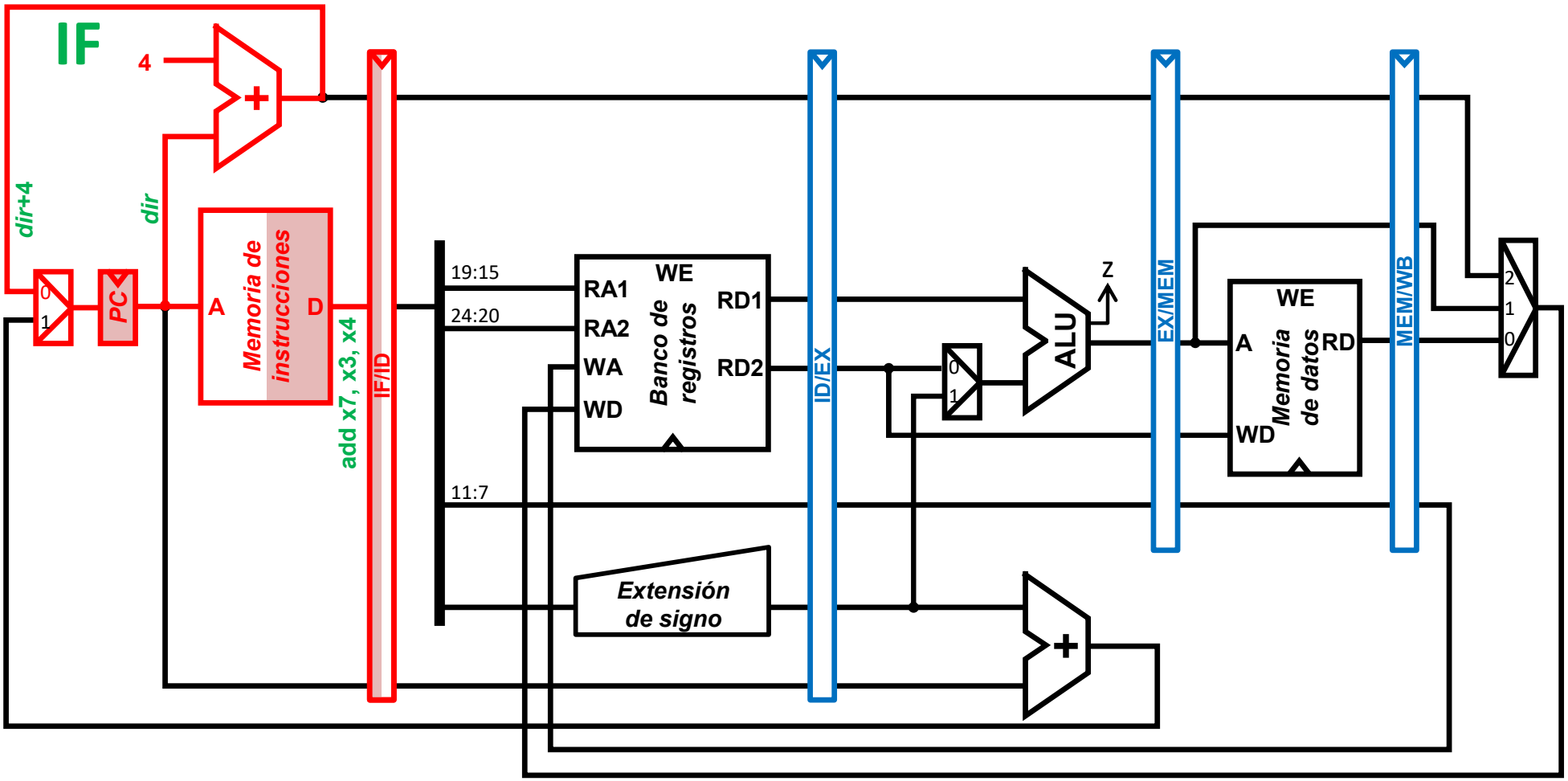


Diseño de la ruta de datos

Instrucción tipo **add**: etapa IF

La instrucción **add** *se ejecuta en 5 ciclos sin usar la memoria en la etapa MEM*

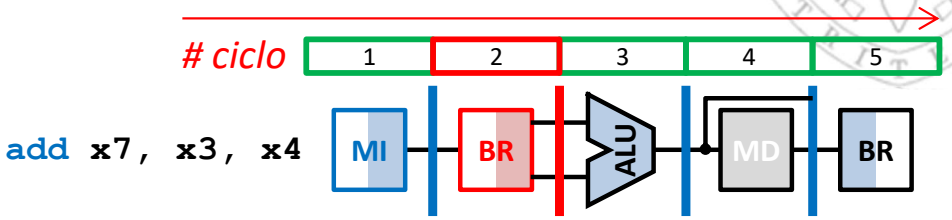


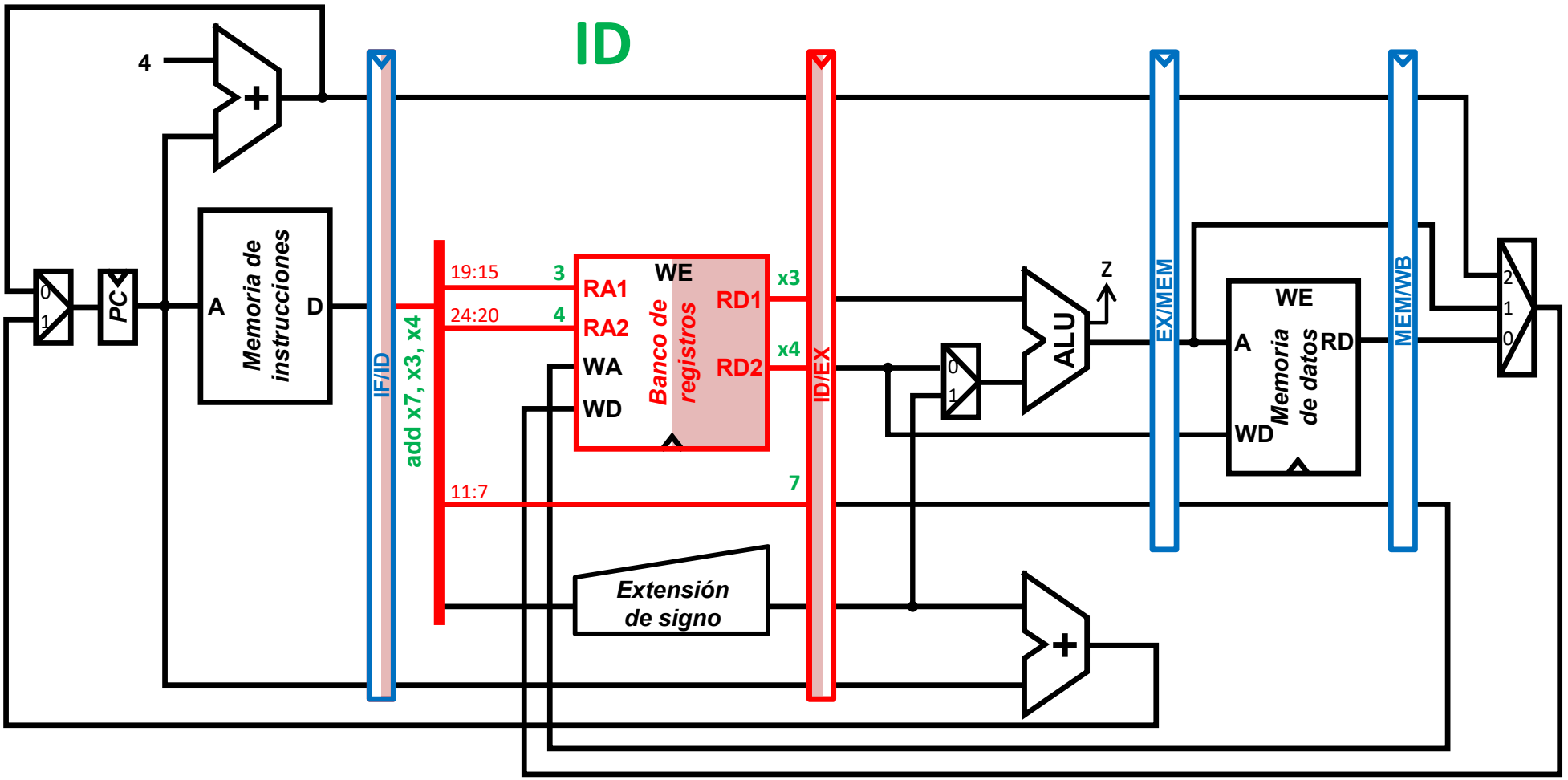


Diseño de la ruta de datos

Instrucción tipo **add**: etapa ID

La instrucción **add** *se ejecuta en* *5 ciclos* *sin usar la memoria en la etapa MEM*

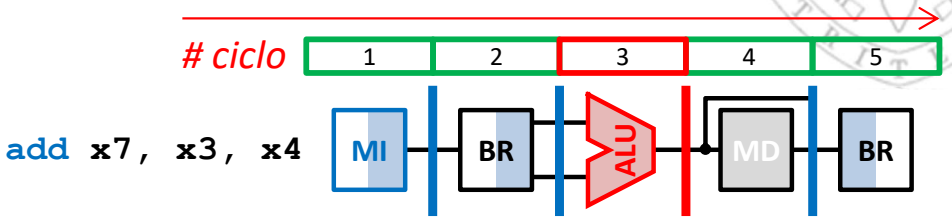


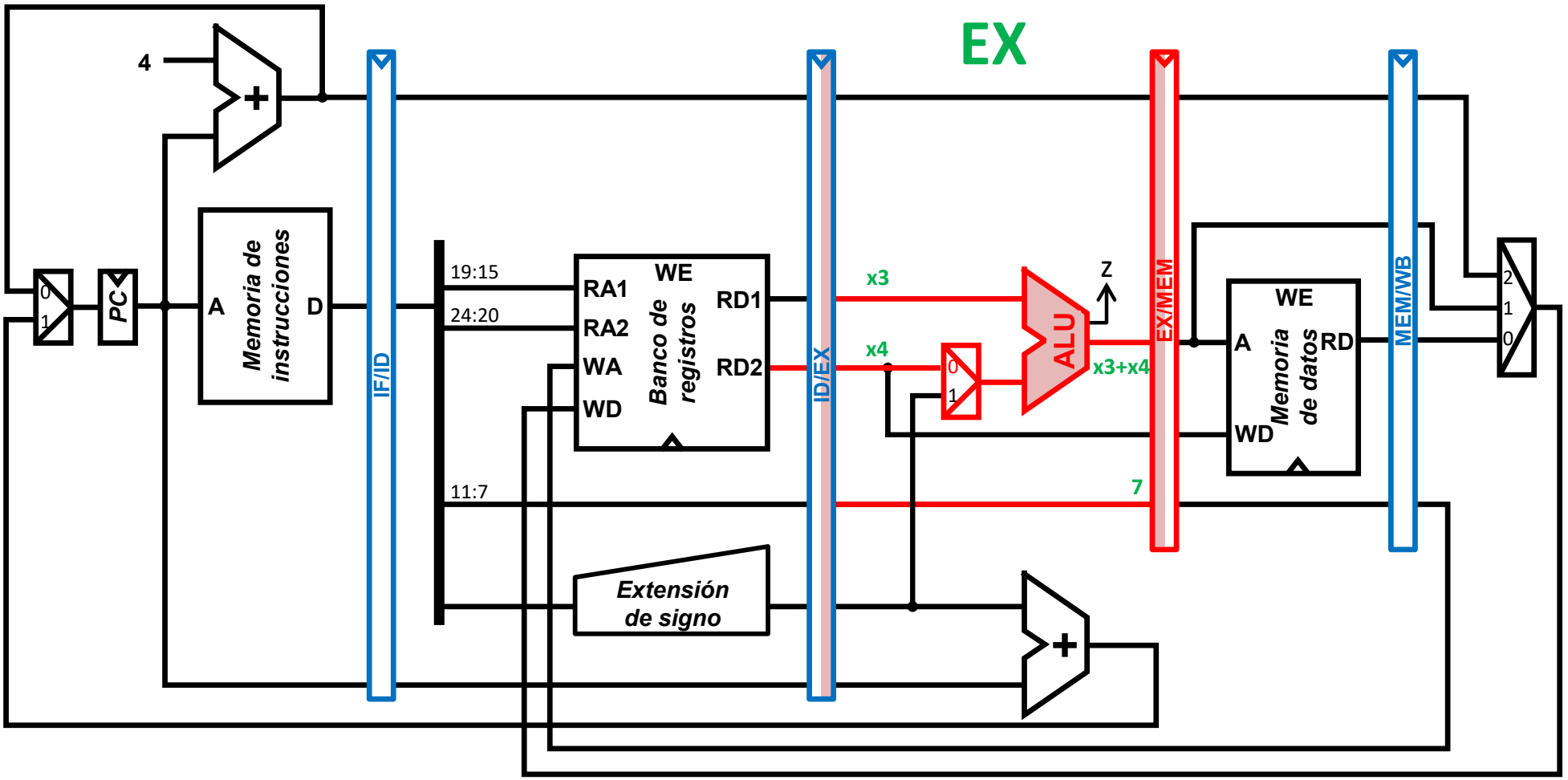


Diseño de la ruta de datos

Instrucción tipo **add**: etapa EX

La instrucción **add** *se ejecuta en* *5 ciclos* *sin usar la memoria en la etapa MEM*

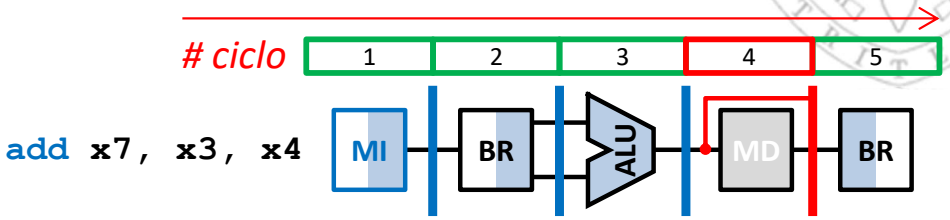


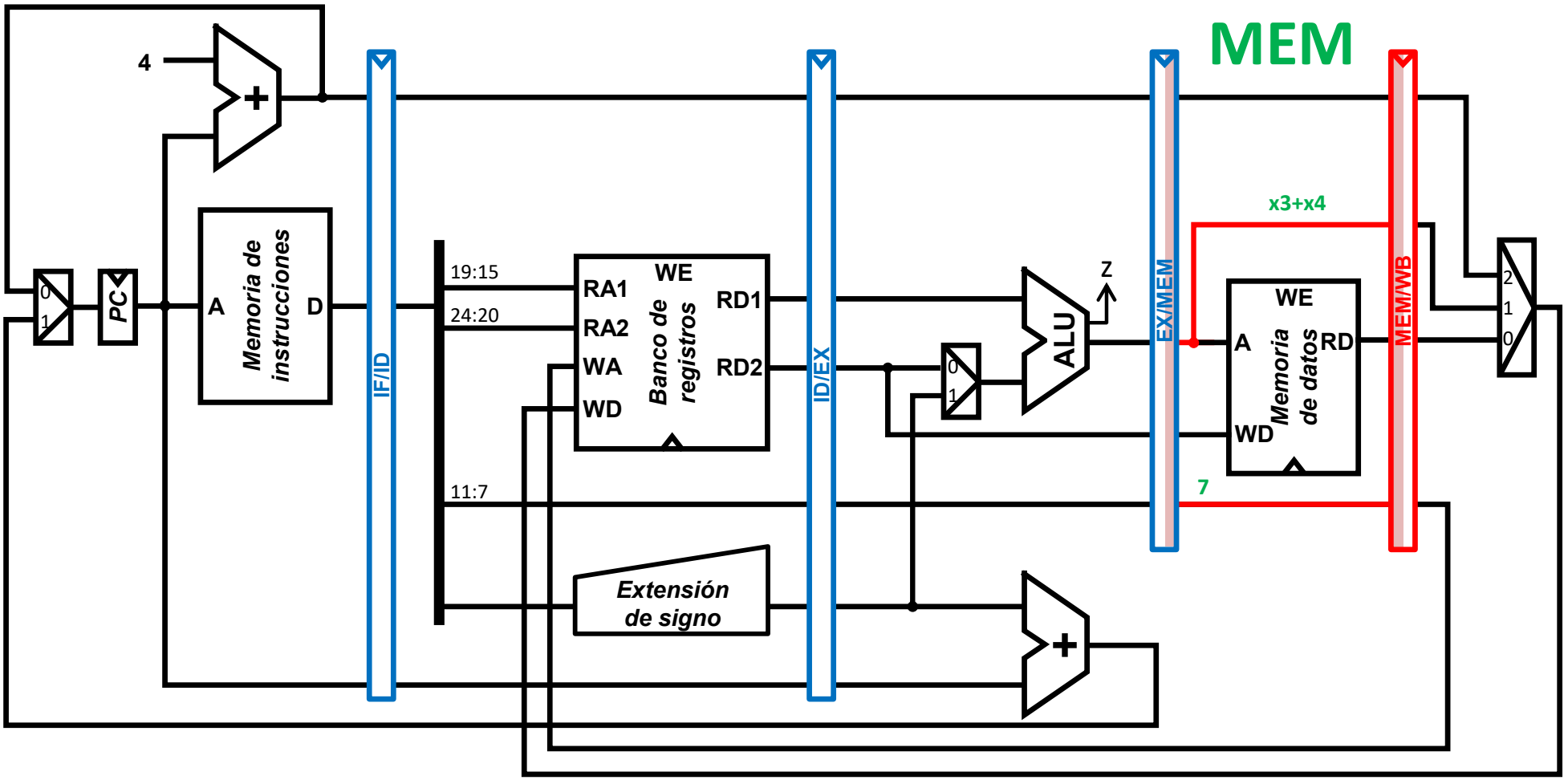


Diseño de la ruta de datos

Instrucción tipo **add**: etapa MEM

La instrucción **add** *se ejecuta en 5 ciclos sin usar la memoria en la etapa MEM*

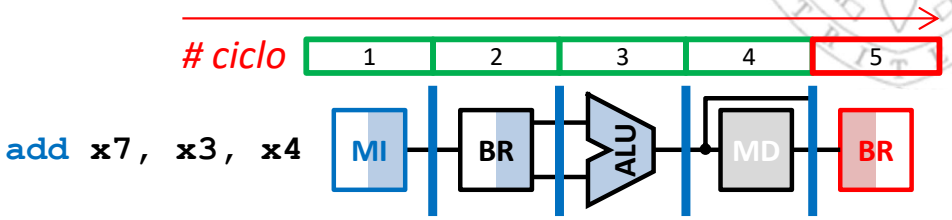


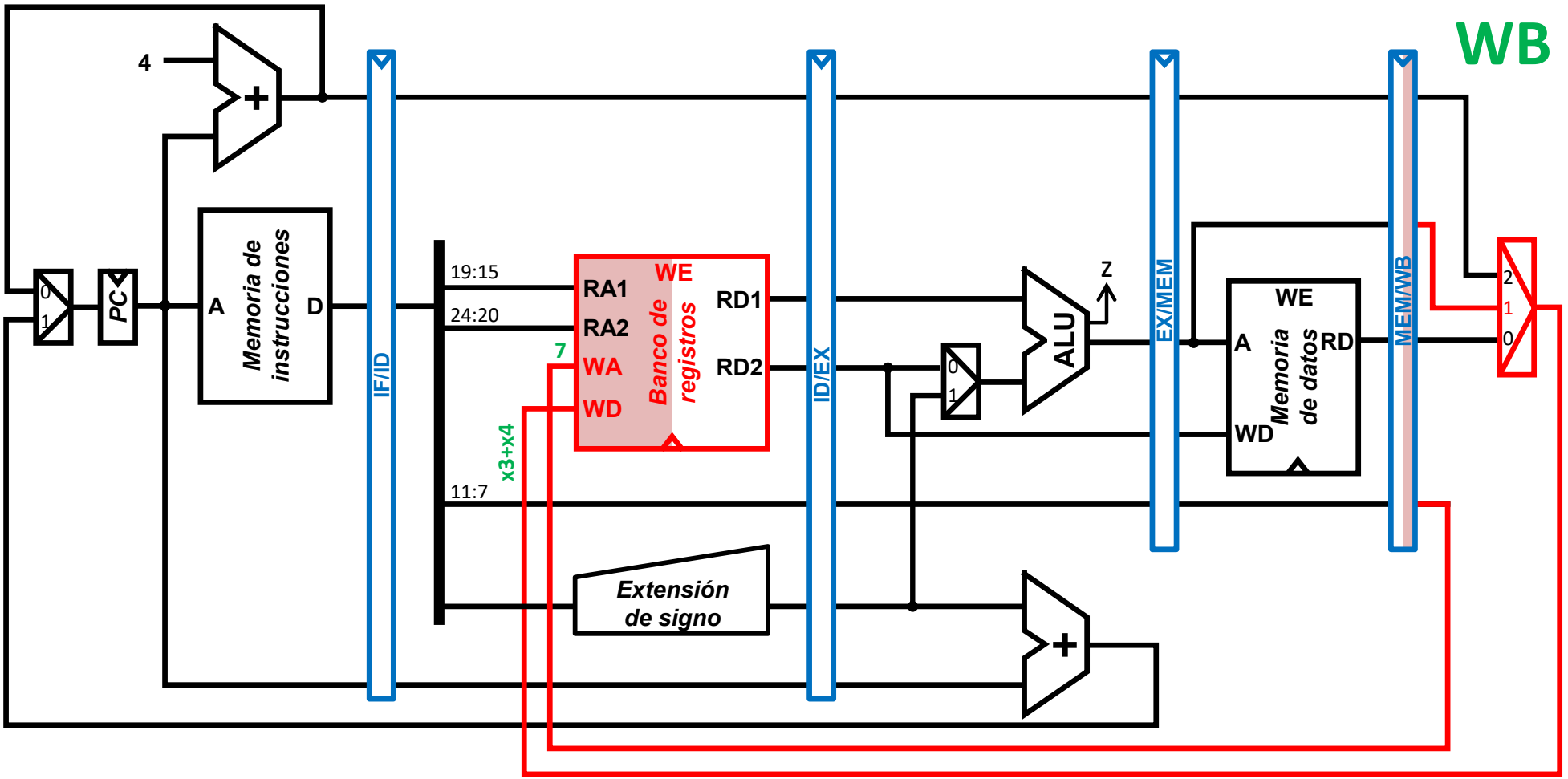


Diseño de la ruta de datos

Instrucción tipo **add**: etapa WB

La instrucción **add** *se ejecuta en 5 ciclos sin usar la memoria en la etapa MEM*

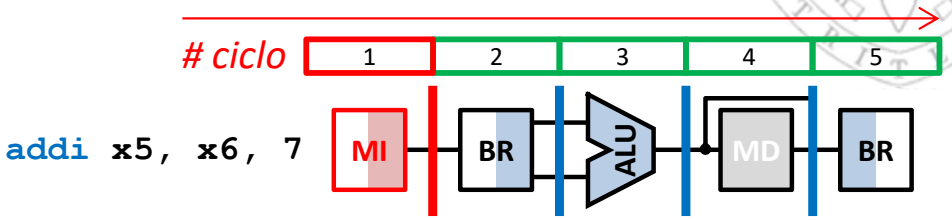


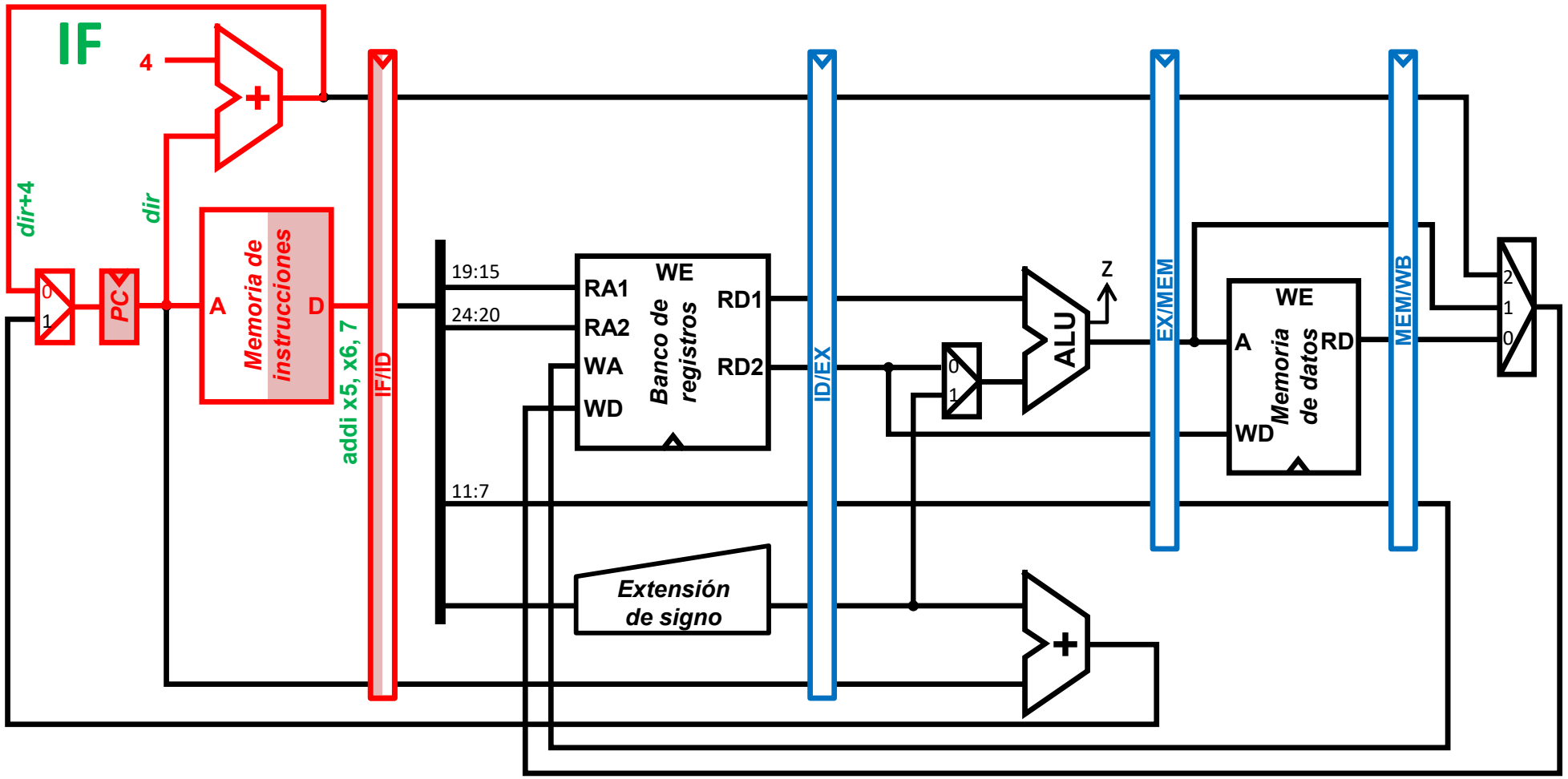


Diseño de la ruta de datos

Instrucción tipo **addi**: etapa IF

La instrucción **addi** *se ejecuta en* *5 ciclos* *sin usar la memoria en la etapa MEM*

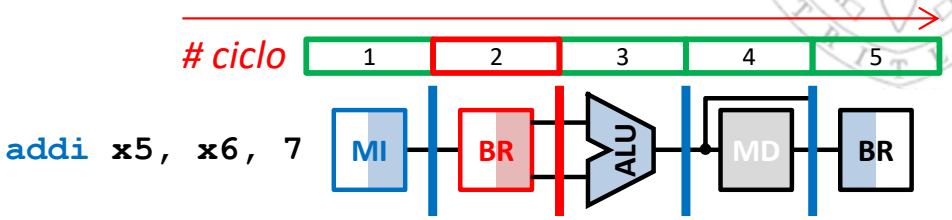


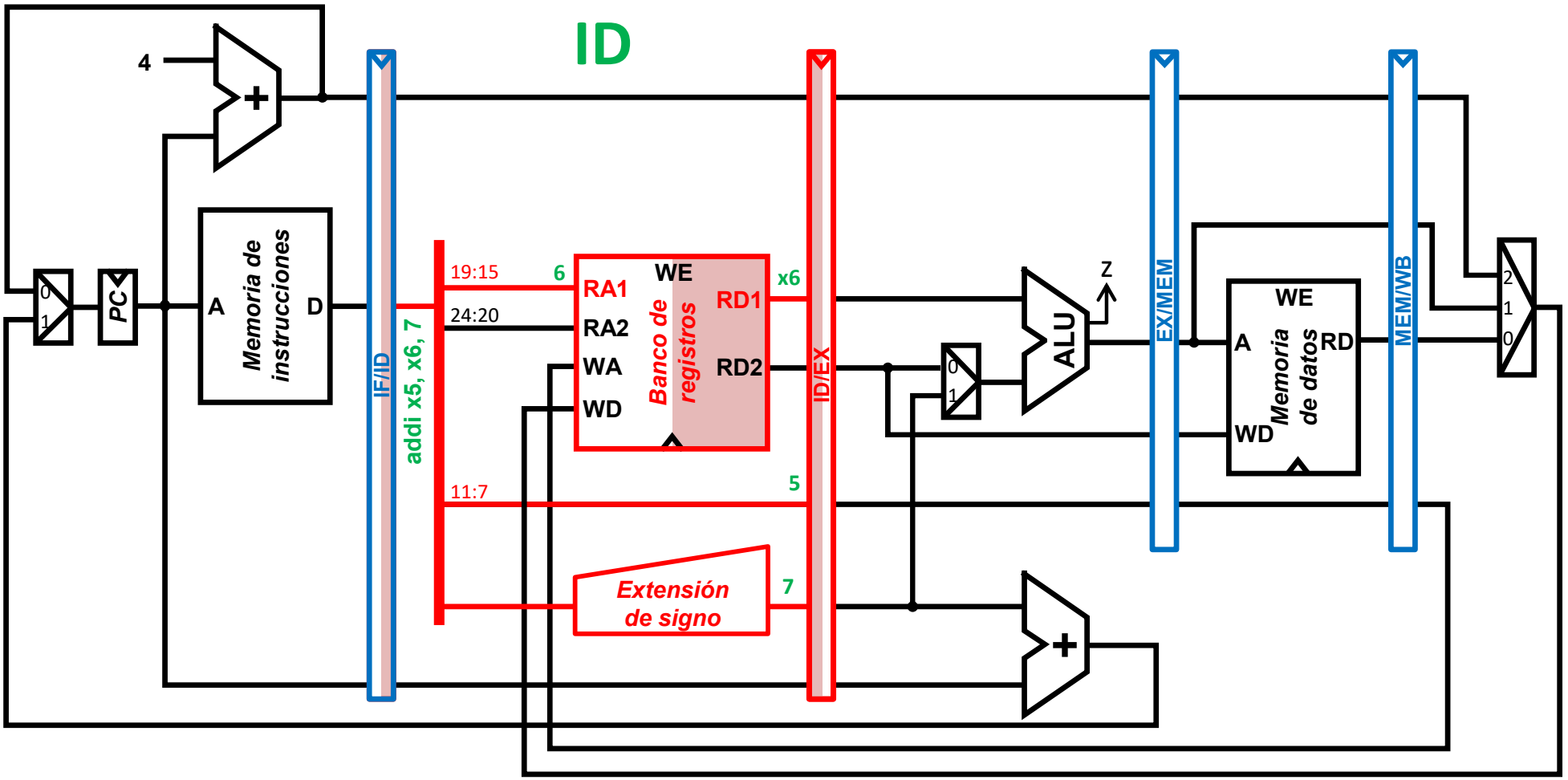


Diseño de la ruta de datos

Instrucción tipo **addi**: etapa ID

La instrucción **addi** *se ejecuta en* *5 ciclos* *sin usar la memoria en la etapa MEM*

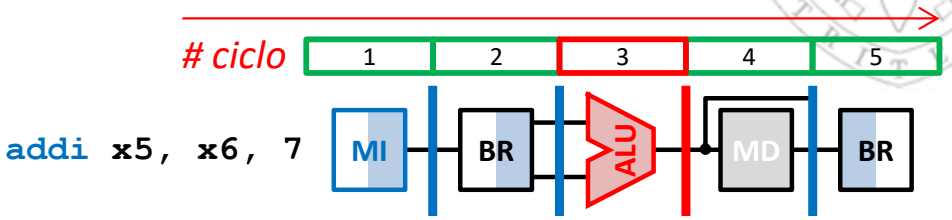


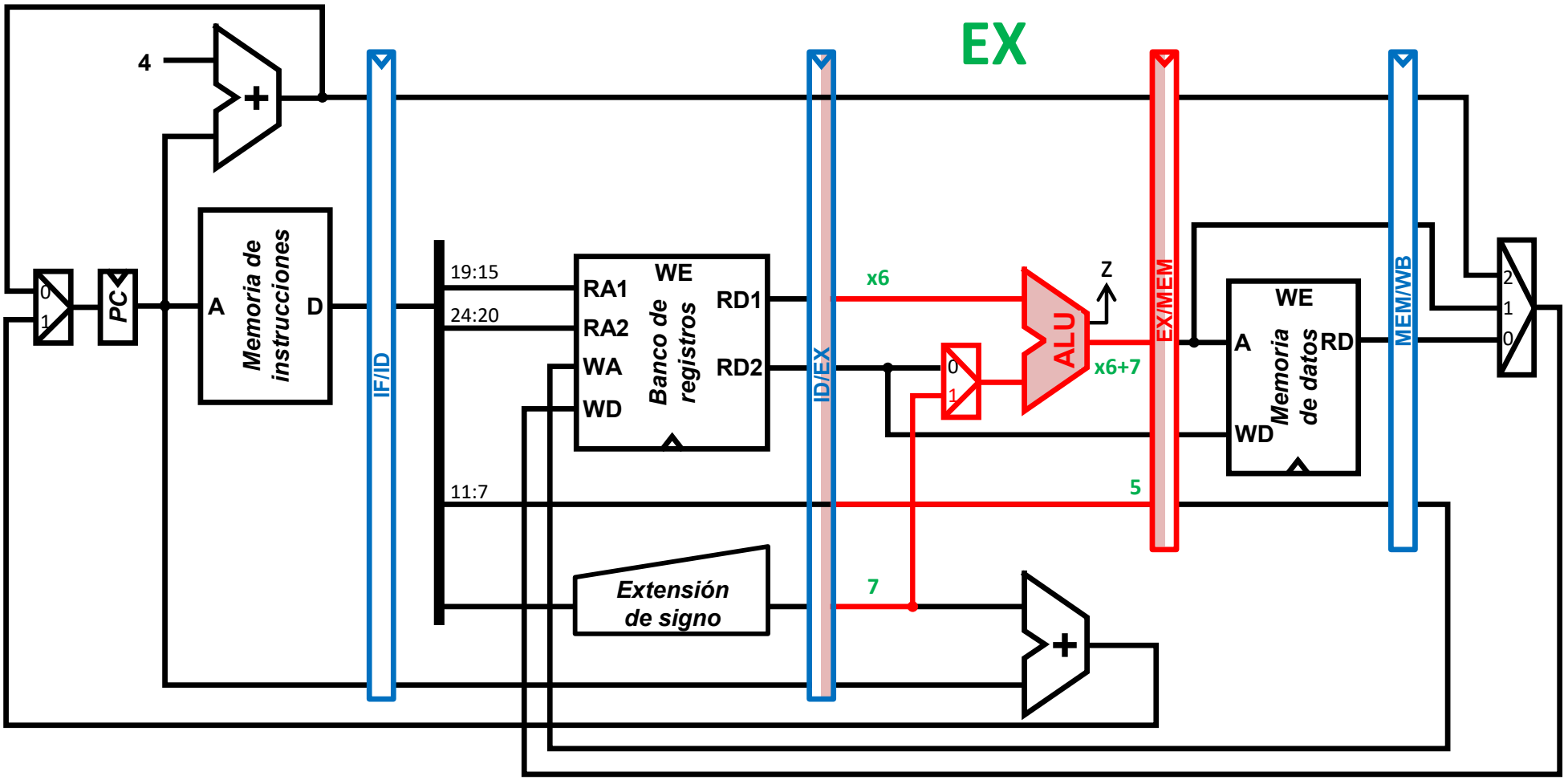


Diseño de la ruta de datos

Instrucción tipo **addi**: etapa EX

La instrucción **addi** *se ejecuta en 5 ciclos sin usar la memoria en la etapa MEM*

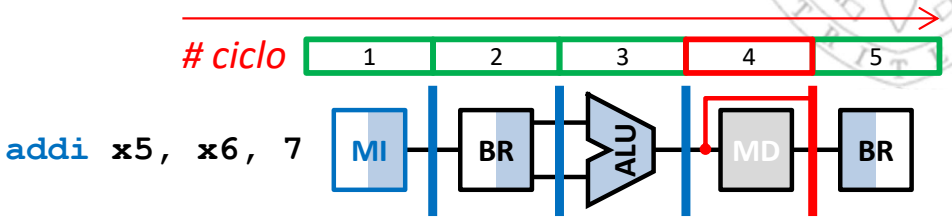


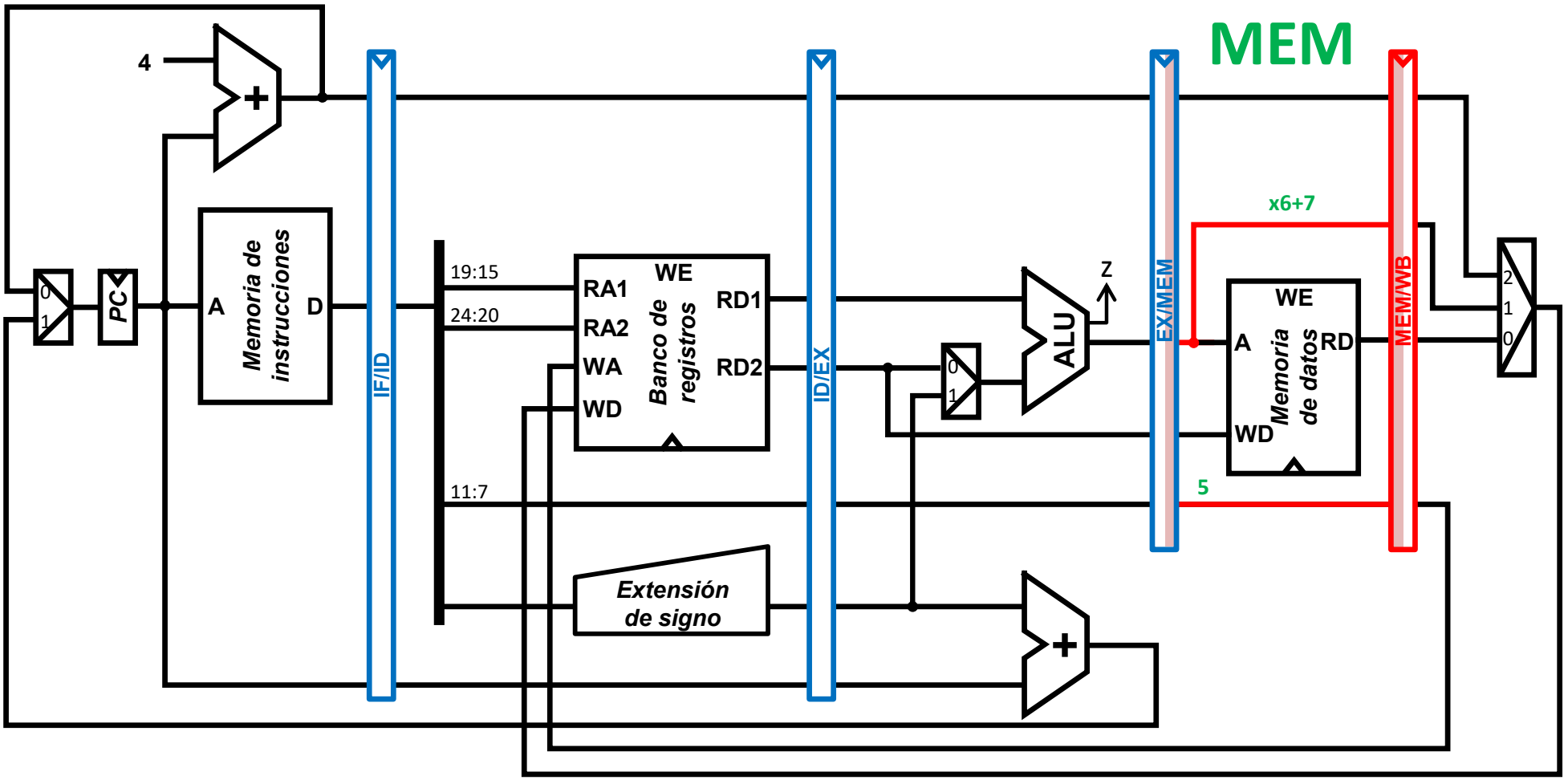


Diseño de la ruta de datos

Instrucción tipo **addi**: etapa MEM

La instrucción **addi** *se ejecuta en* *5 ciclos* *sin usar la memoria en la etapa MEM*

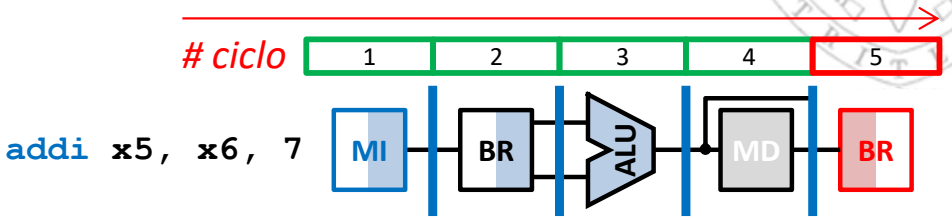


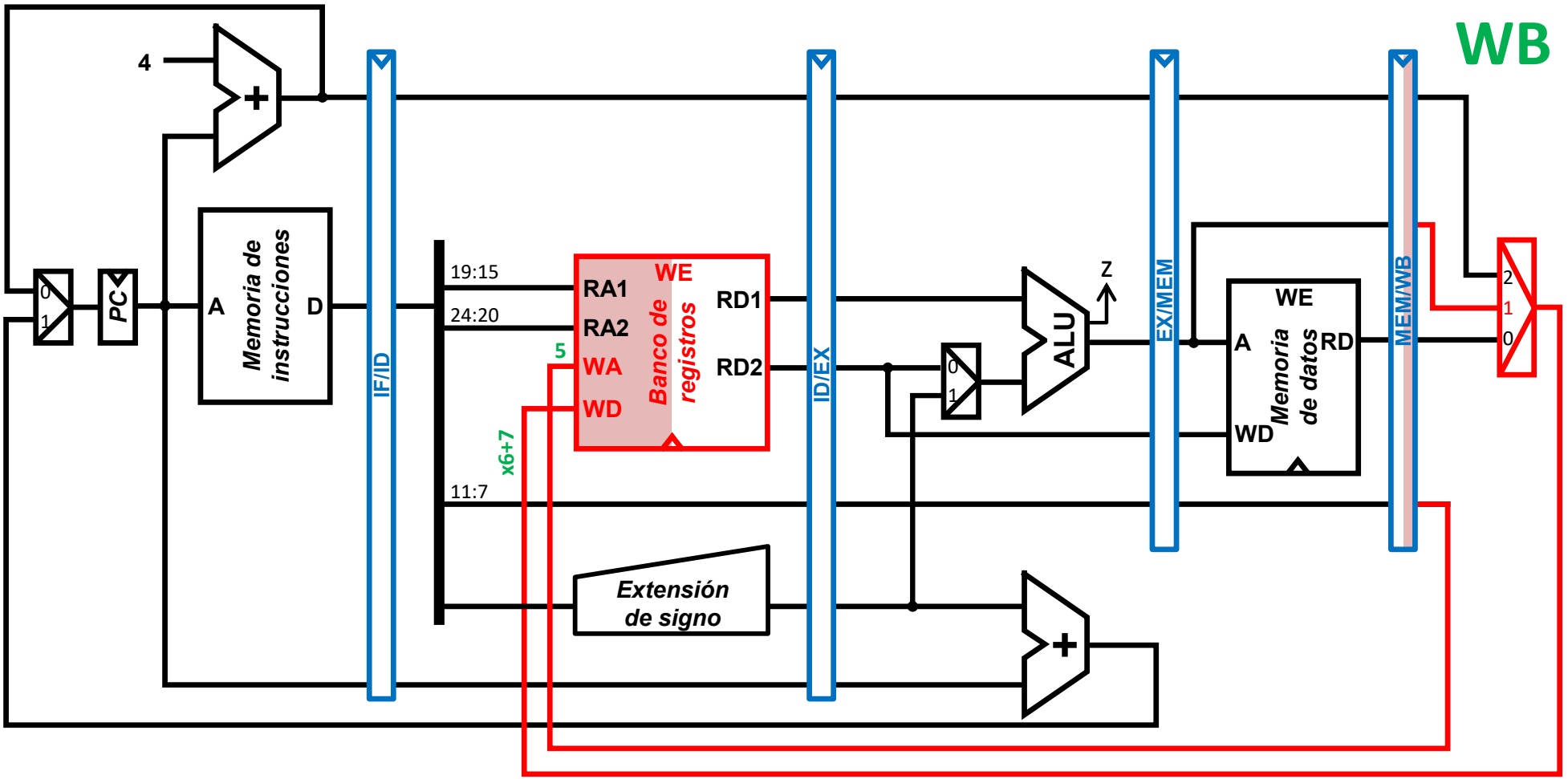


Diseño de la ruta de datos

Instrucción tipo **addi**: etapa WB

La instrucción **addi** *se ejecuta en* *5 ciclos* *sin usar la memoria en la etapa MEM*

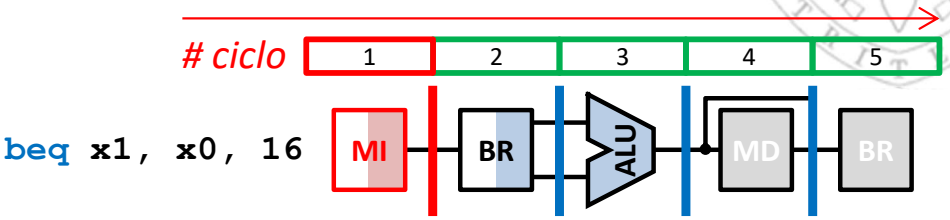


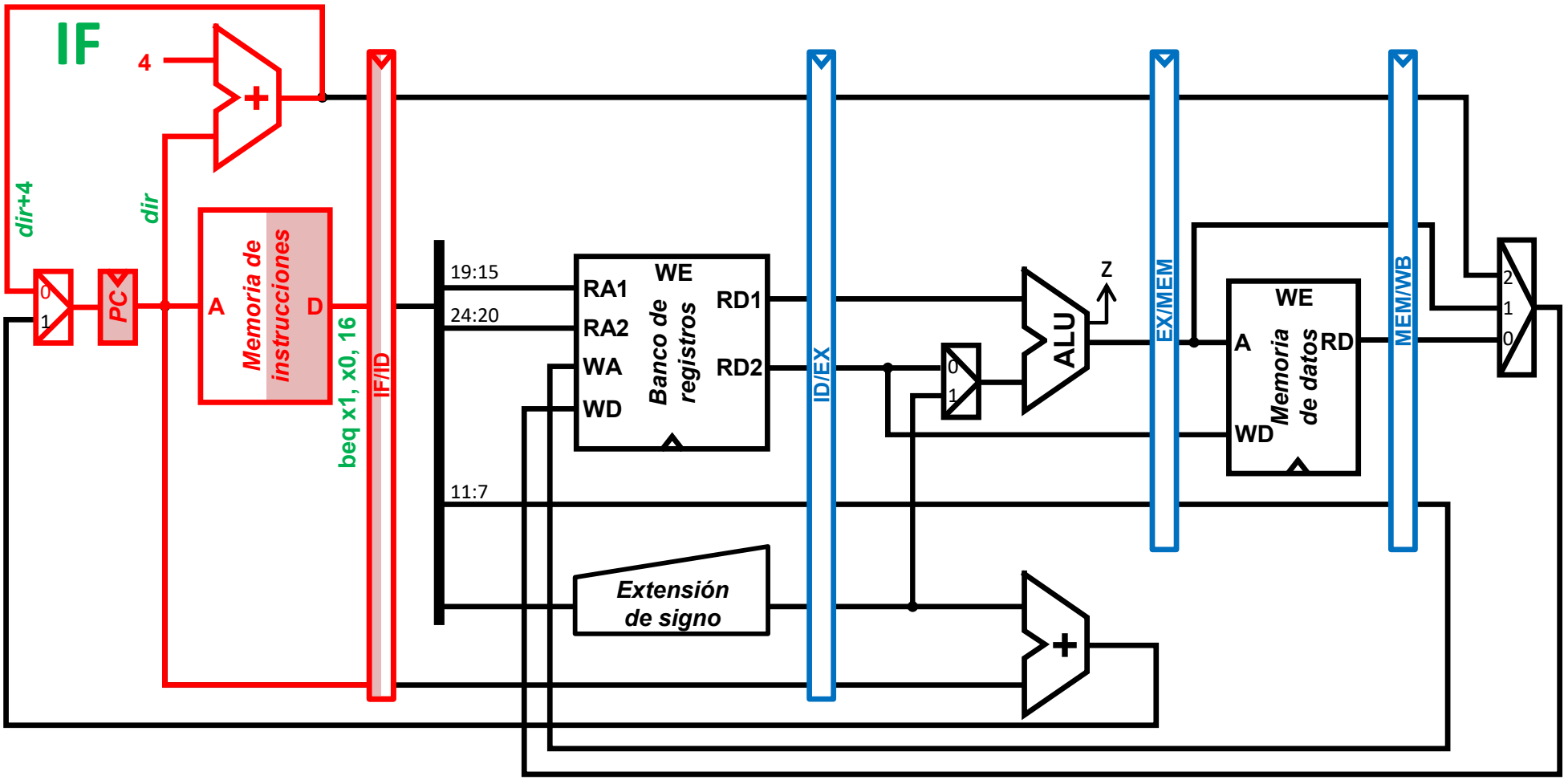


Diseño de la ruta de datos

Instrucción **beq**: etapa IF

La instrucción **beq** *se ejecuta en 5 ciclos sin usar la memoria en la etapa MEM ni el BR en la etapa WB*

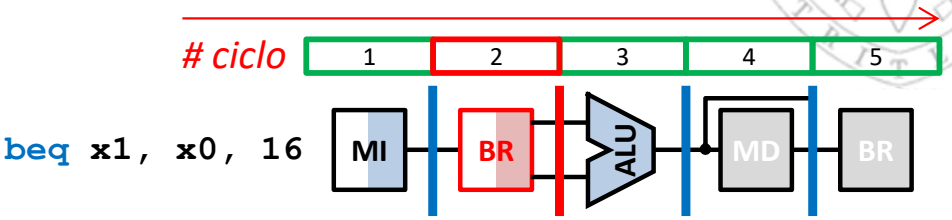


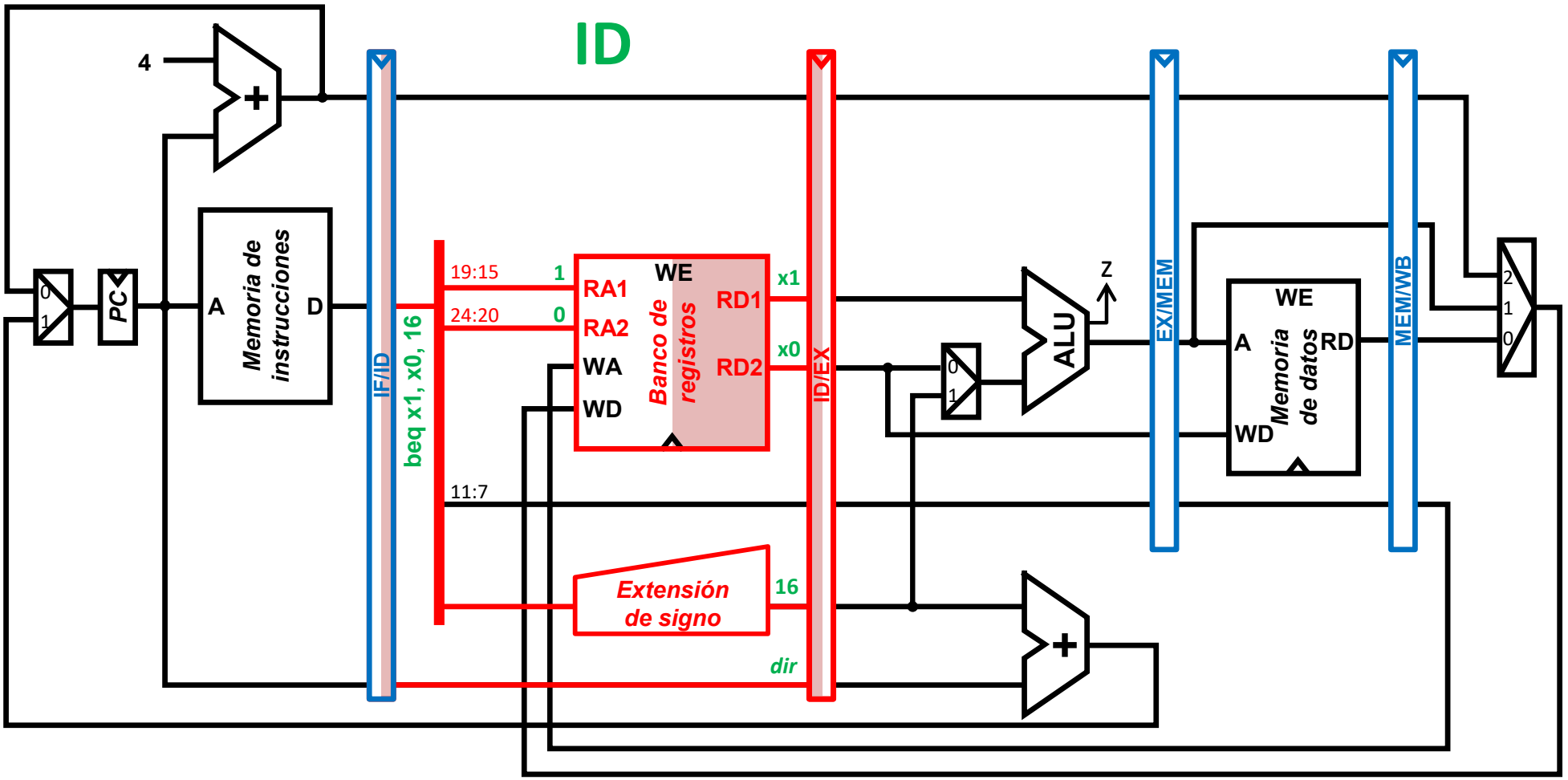


Diseño de la ruta de datos

Instrucción **beq**: etapa ID

La instrucción **beq** *se ejecuta en 5 ciclos sin usar la memoria en la etapa MEM ni el BR en la etapa WB*

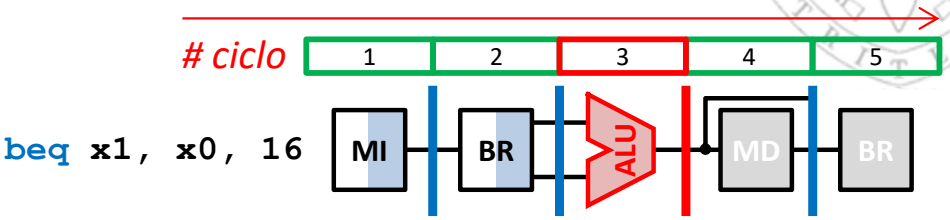


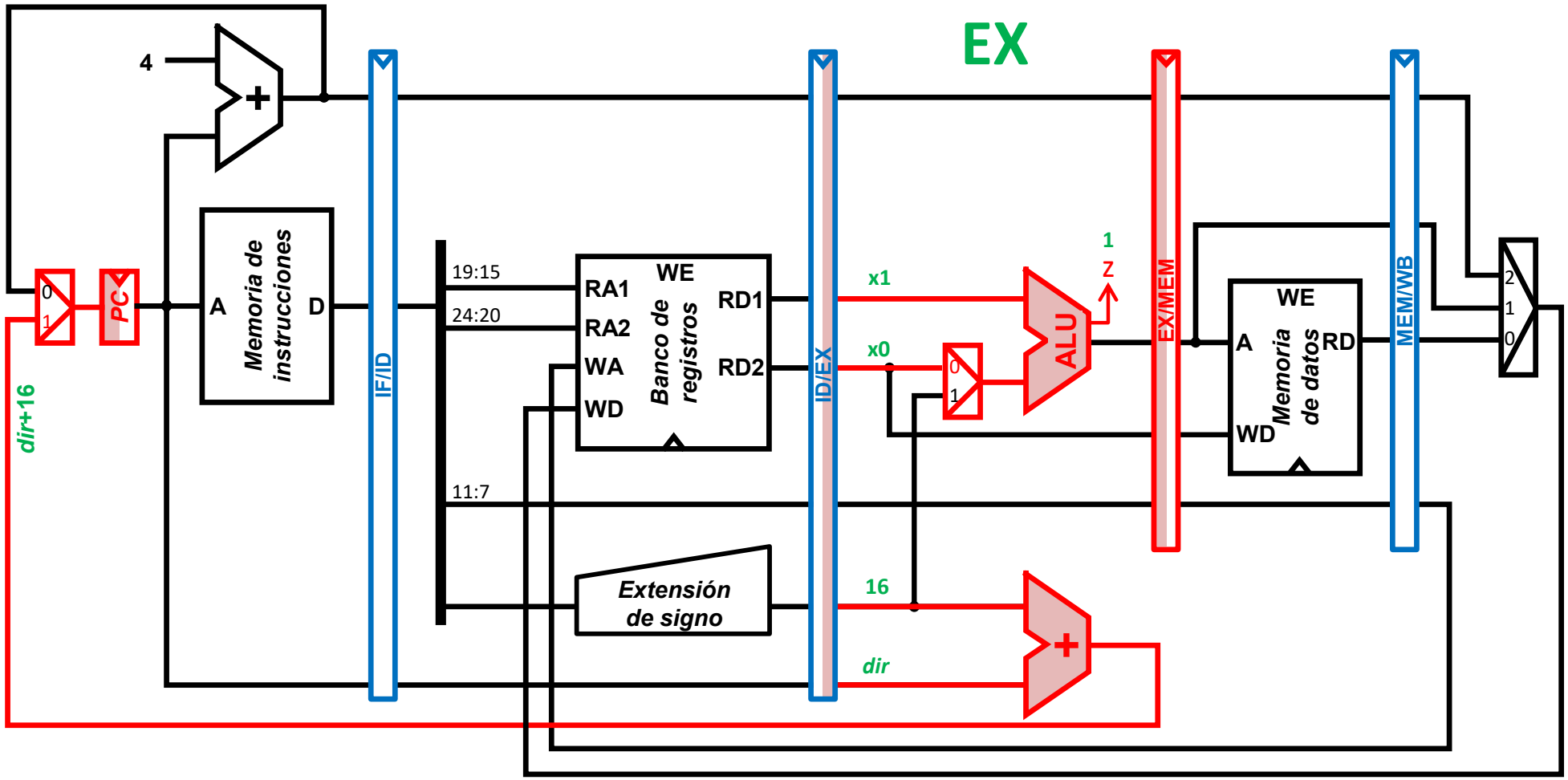


Diseño de la ruta de datos

Instrucción `beq`: etapa EX

La instrucción **`beq`** *se ejecuta en 5 ciclos sin usar la memoria en la etapa MEM ni el BR en la etapa WB*

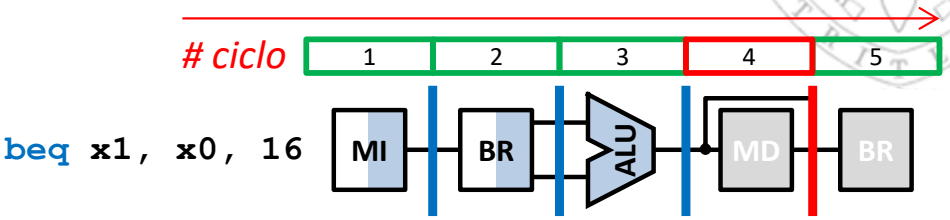


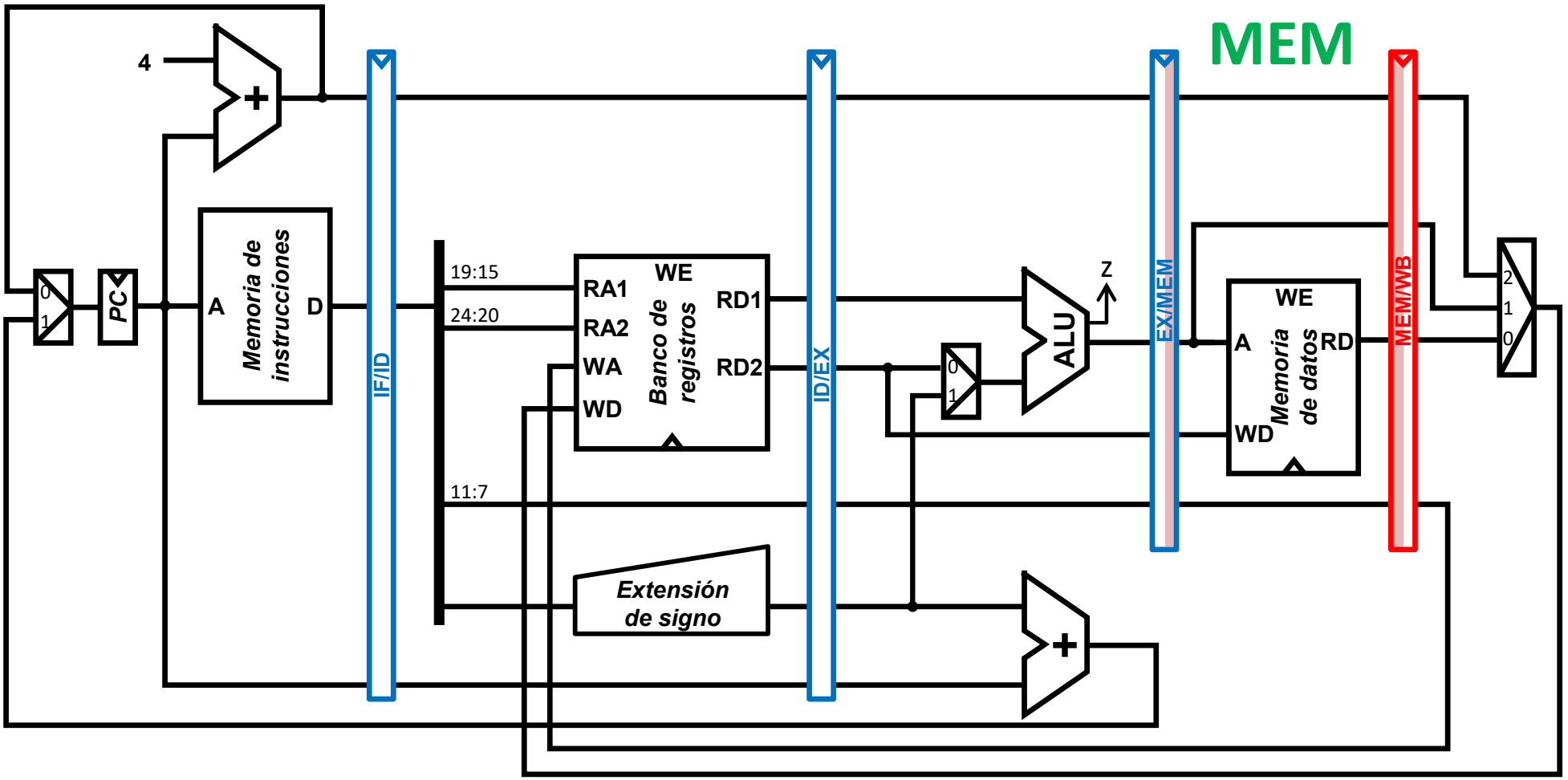


Diseño de la ruta de datos

Instrucción `beq`: etapa MEM

La instrucción **`beq`** *se ejecuta en 5 ciclos sin usar la memoria en la etapa MEM ni el BR en la etapa WB*

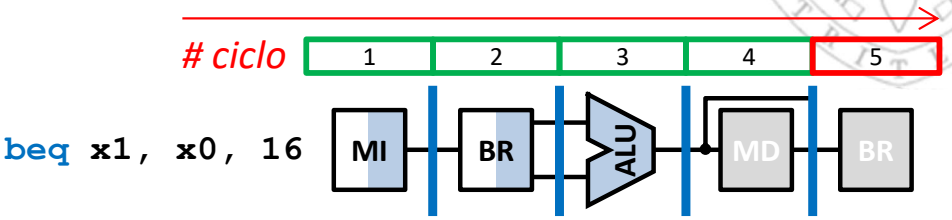


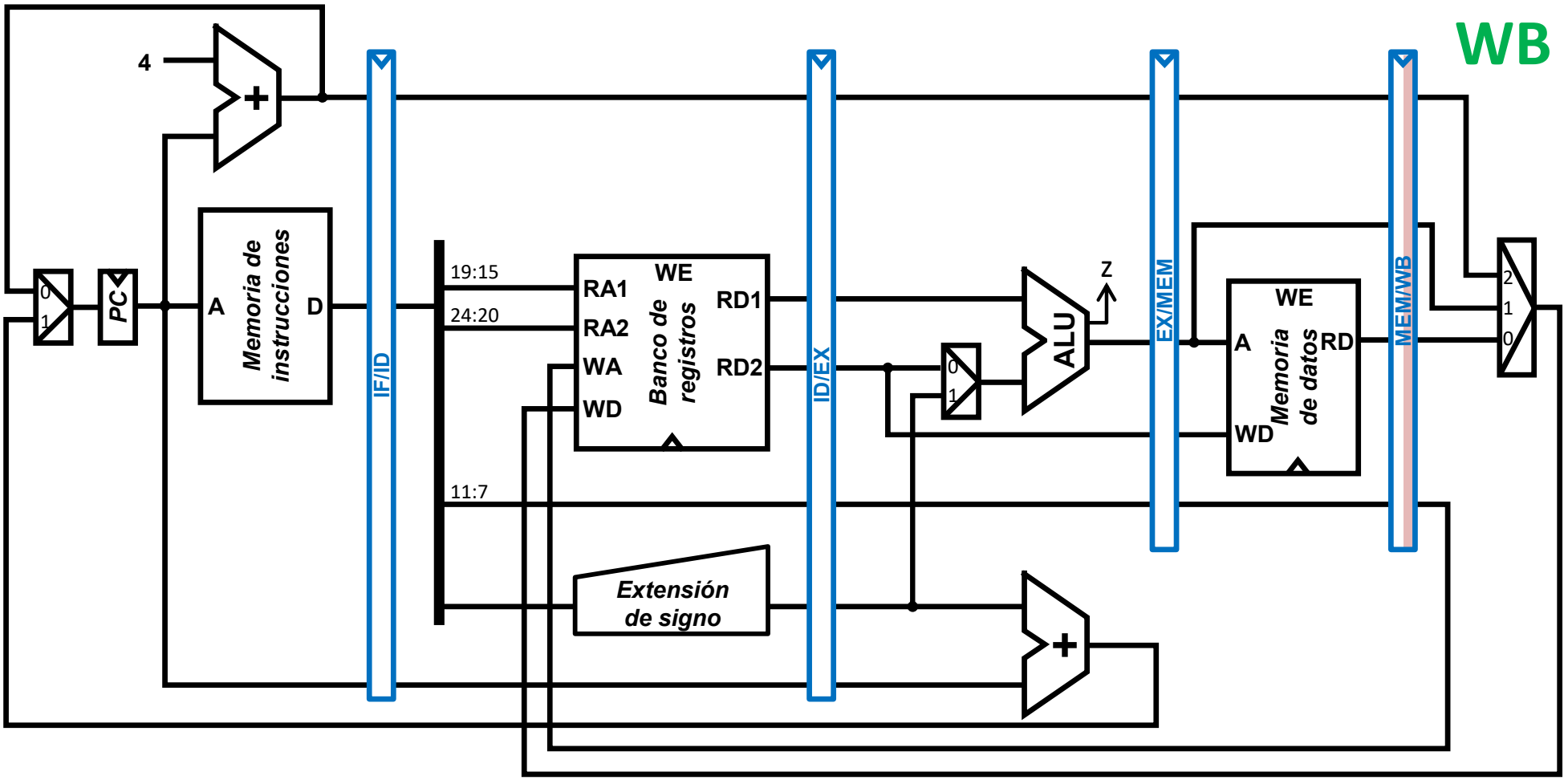


Diseño de la ruta de datos

Instrucción **beq**: etapa WB

La instrucción **beq** *se ejecuta en 5 ciclos sin usar la memoria en la etapa MEM ni el BR en la etapa WB*

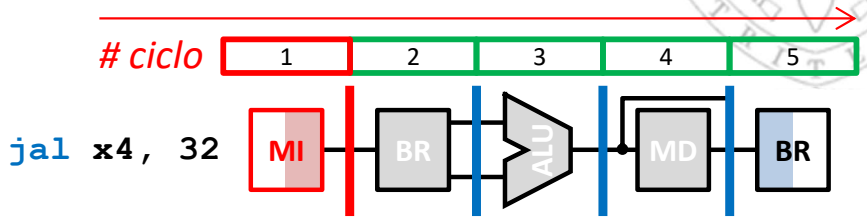


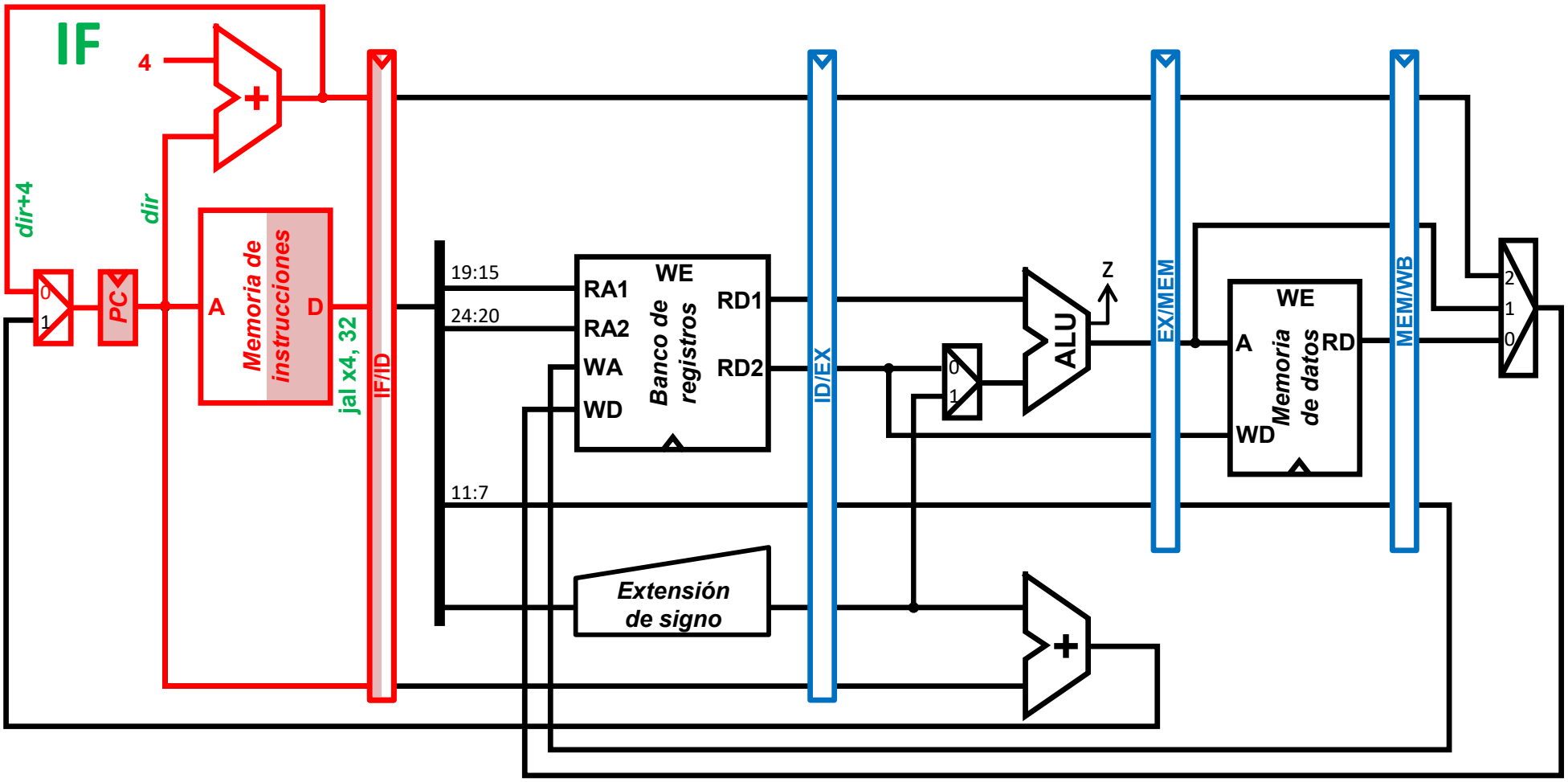


Diseño de la ruta de datos

Instrucción `jal`: etapa IF

La instrucción **`jal`** *se ejecuta en 5 ciclos sin usar el BR en la etapa ID, ni la ALU en la etapa EX, ni la memoria en la etapa MEM*

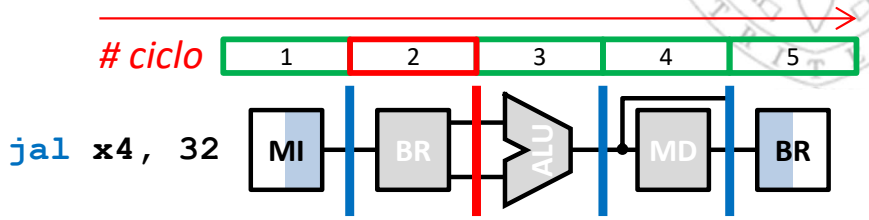


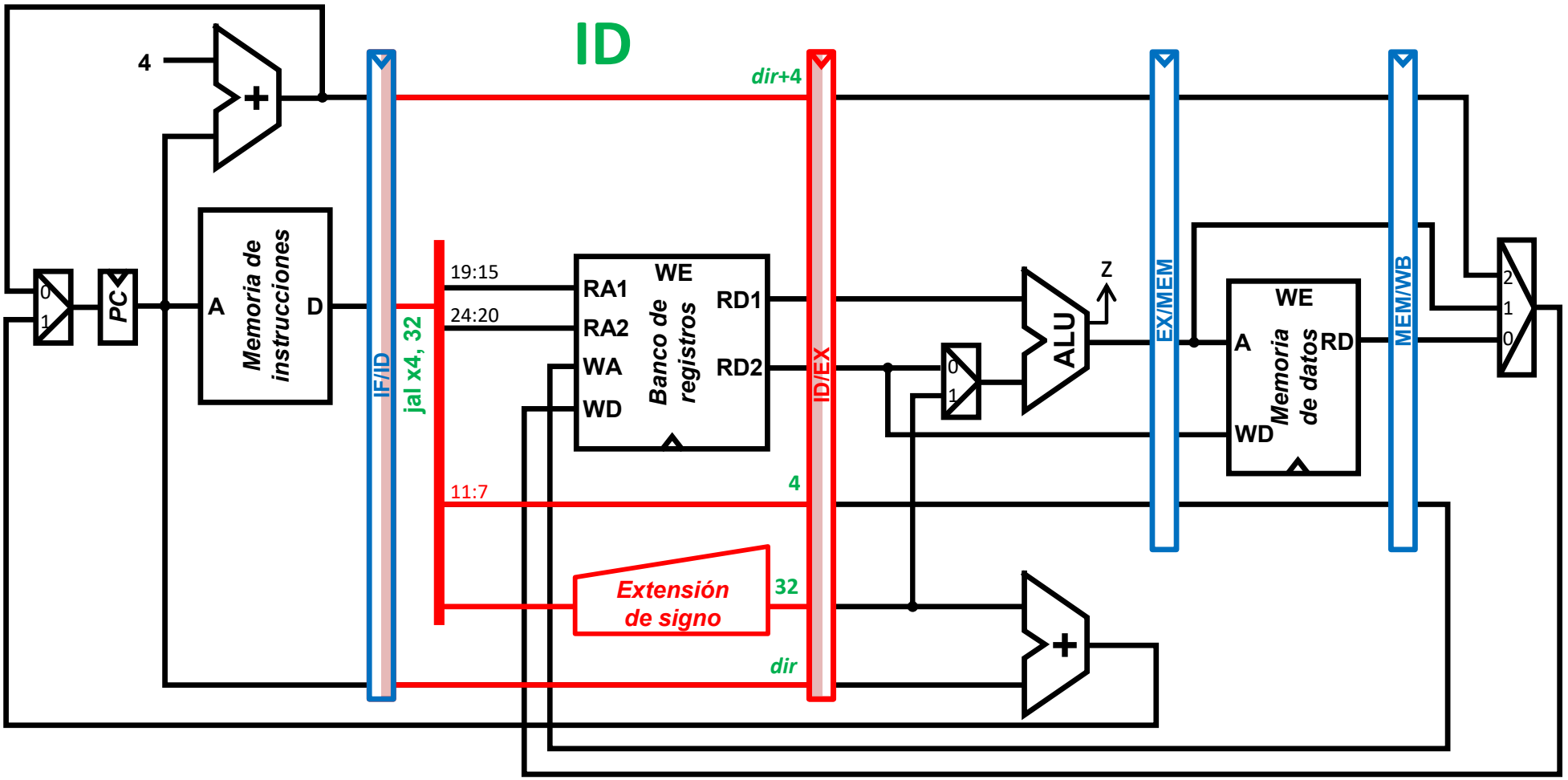


Diseño de la ruta de datos

Instrucción `jal`: etapa ID

La instrucción **`jal`** *se ejecuta en 5 ciclos sin usar el BR en la etapa ID, ni la ALU en la etapa EX, ni la memoria en la etapa MEM*

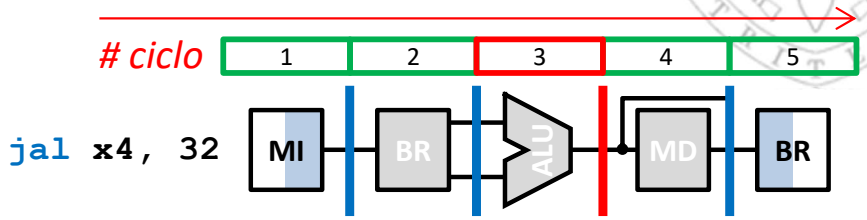


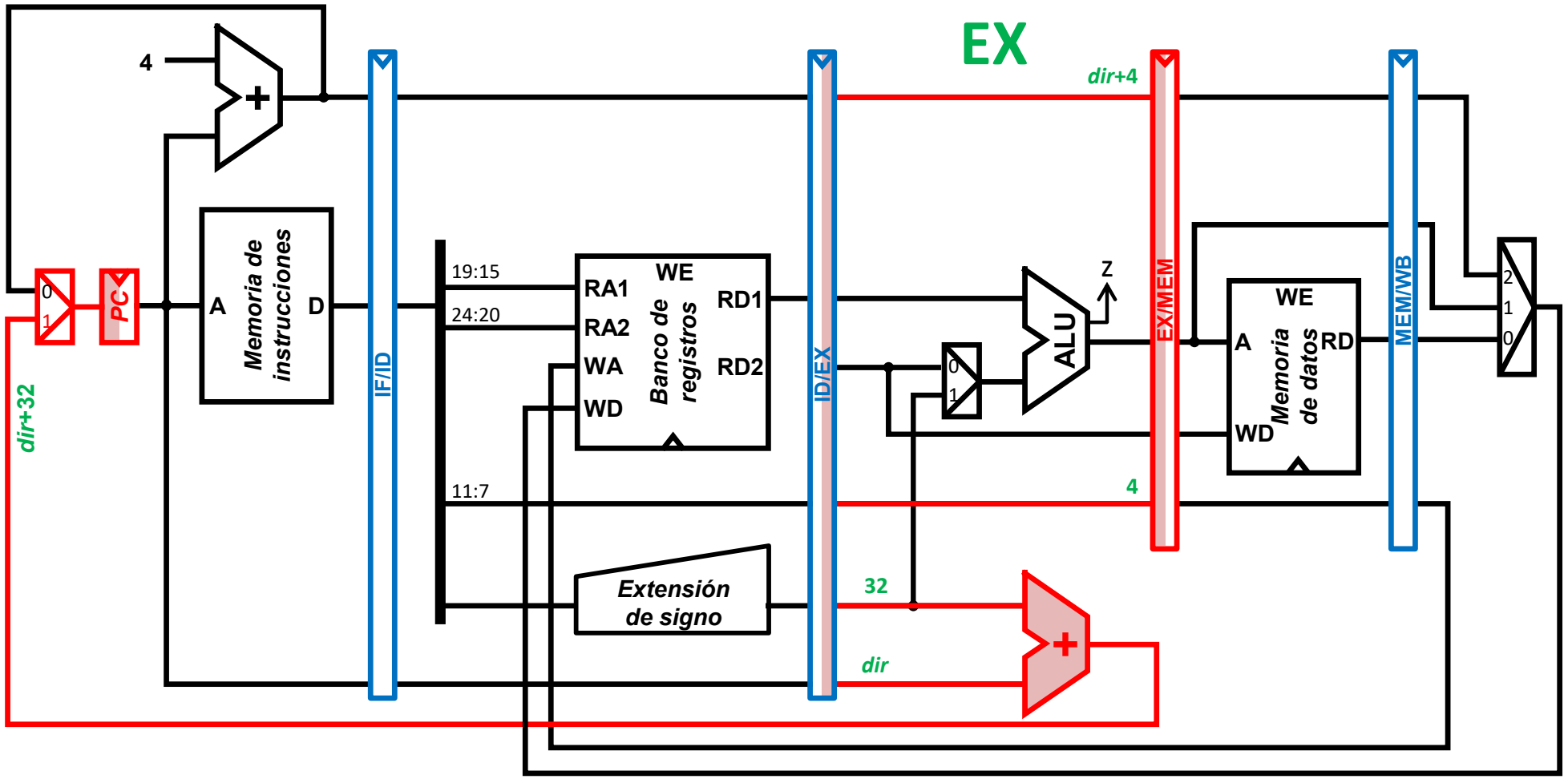


Diseño de la ruta de datos

Instrucción `jal`: etapa EX

La instrucción **`jal`** *se ejecuta en 5 ciclos*
sin usar el BR en la etapa ID,
ni la ALU en la etapa EX,
ni la memoria en la etapa MEM

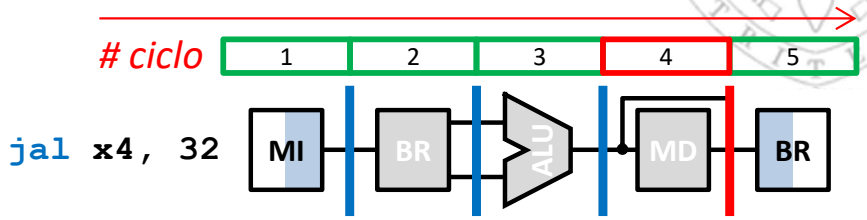


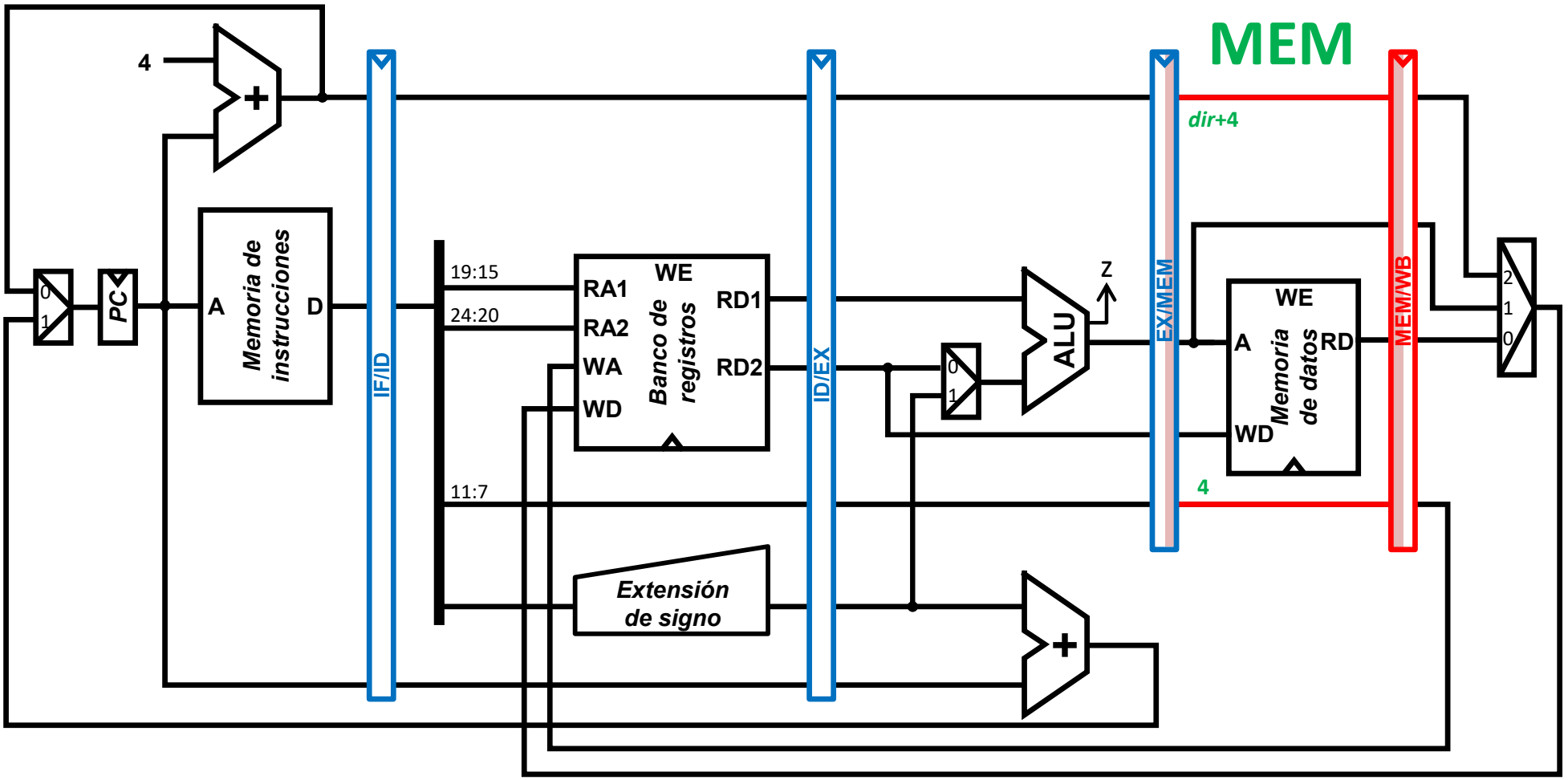


Diseño de la ruta de datos

Instrucción `jal`: etapa MEM

La instrucción **`jal`** *se ejecuta en* *5 ciclos*
sin usar el BR en la etapa ID,
ni la ALU en la etapa EX,
ni la memoria en la etapa MEM

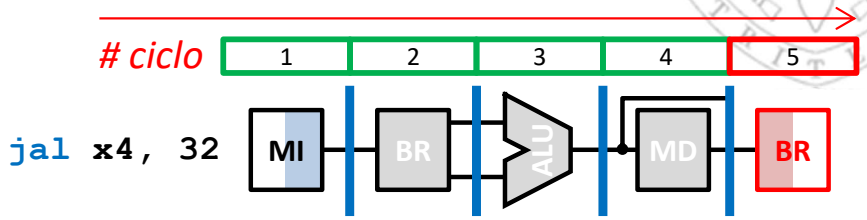


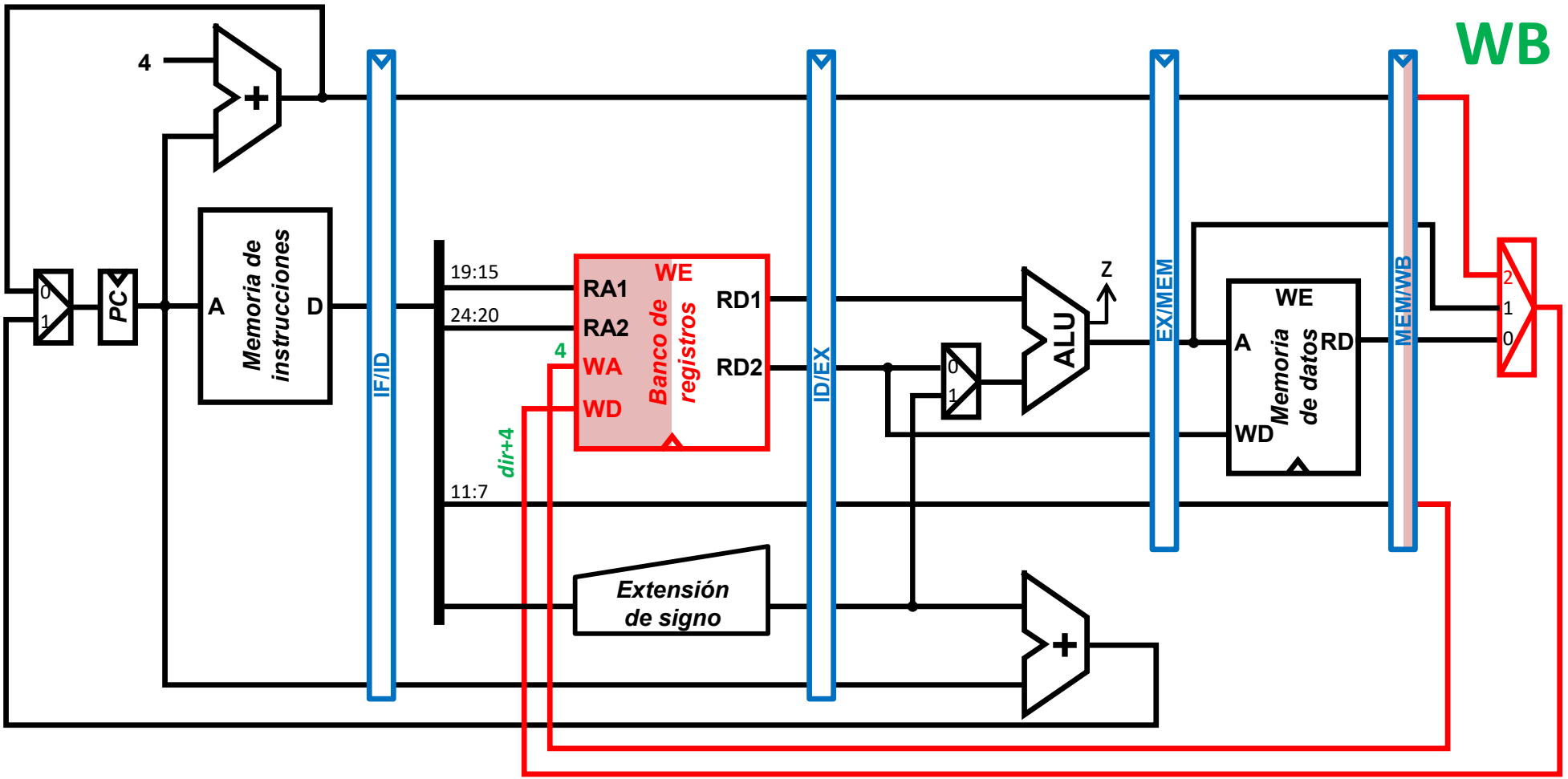


Diseño de la ruta de datos

Instrucción `jal`: etapa WB

La instrucción **`jal`** *se ejecuta en 5 ciclos sin usar el BR en la etapa ID, ni la ALU en la etapa EX, ni la memoria en la etapa MEM*

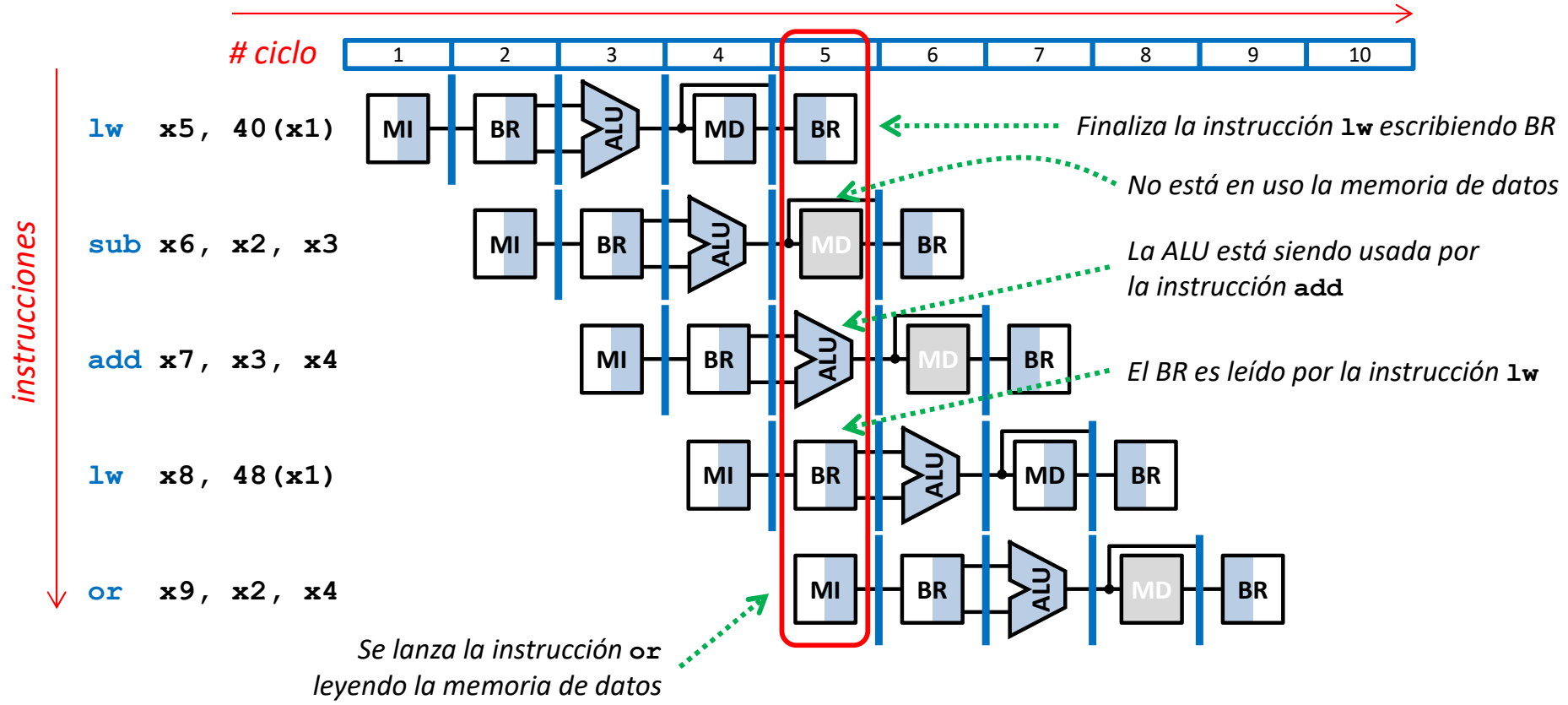




Diseño de la ruta de datos

Diagrama de ejecución (i)

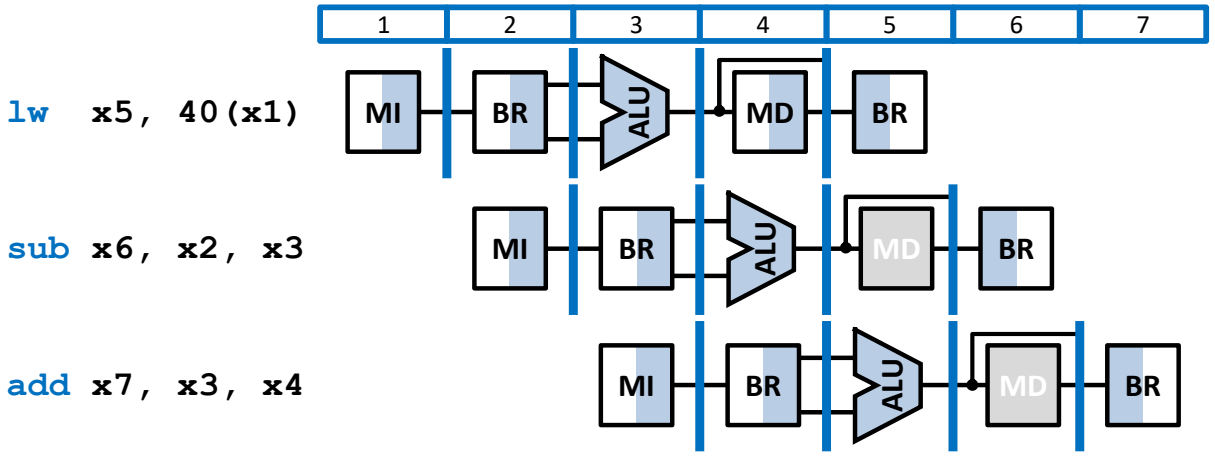
- Los diagramas de ejecución permiten visualizar la ejecución de un programa en el pipeline:
  - Para un ciclo dado, visualiza las instrucciones que están en ejecución, en qué etapa del pipe está cada una y los recursos que se utilizan.



Diseño de la ruta de datos

Diagrama de ejecución (ii)

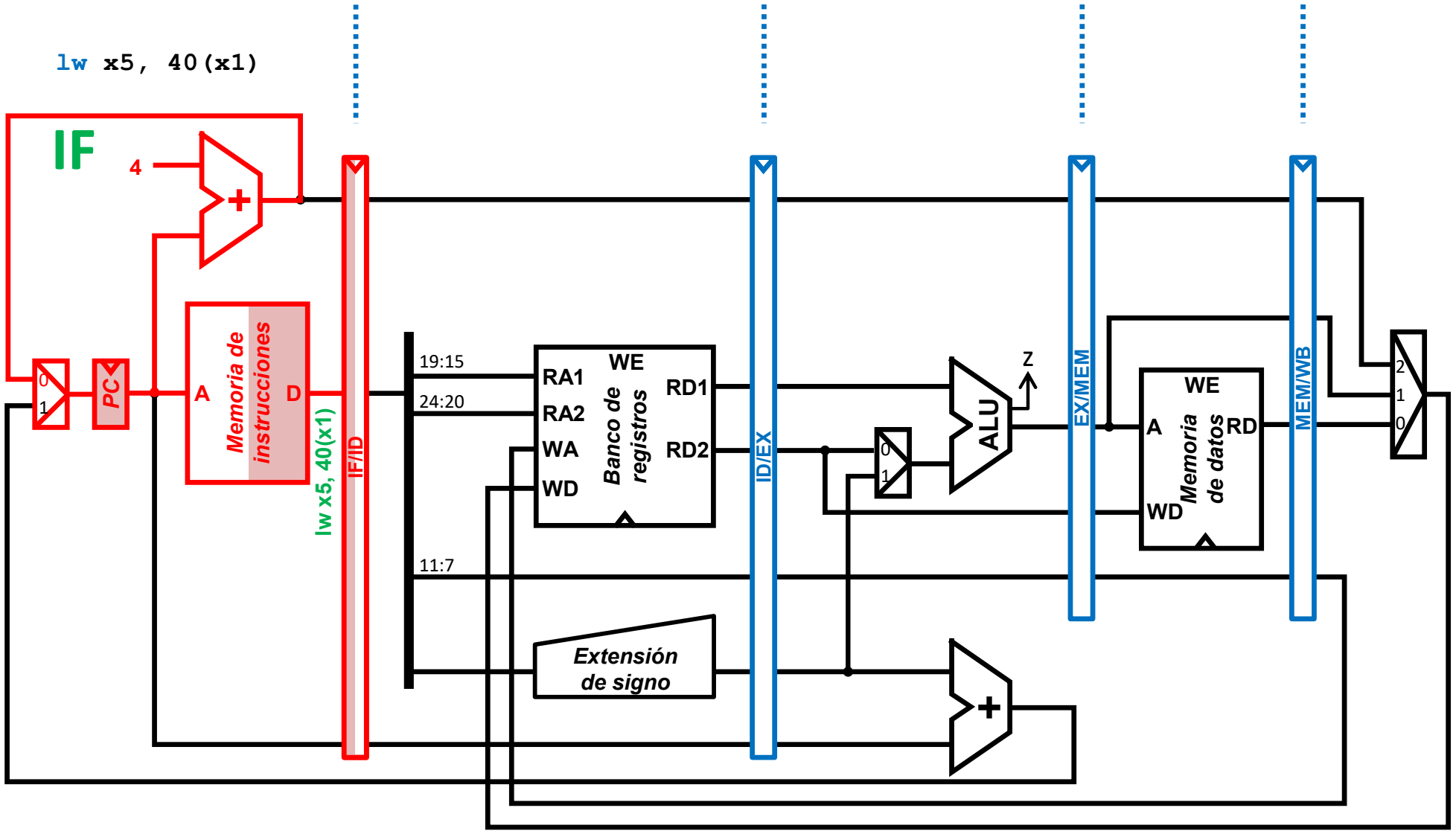
- Alternativamente, es común usar diagramas de ejecución simplificados que muestran, en cada ciclo, la etapa en que está cada instrucción.



	1	2	3	4	5	6	7
`lw x5, 40(x1)`	IF	ID	EX	M	WB		
`sub x6, x2, x3`		IF	ID	EX	M	WB	
`add x7, x3, x4`			IF	ID	EX	M	WB

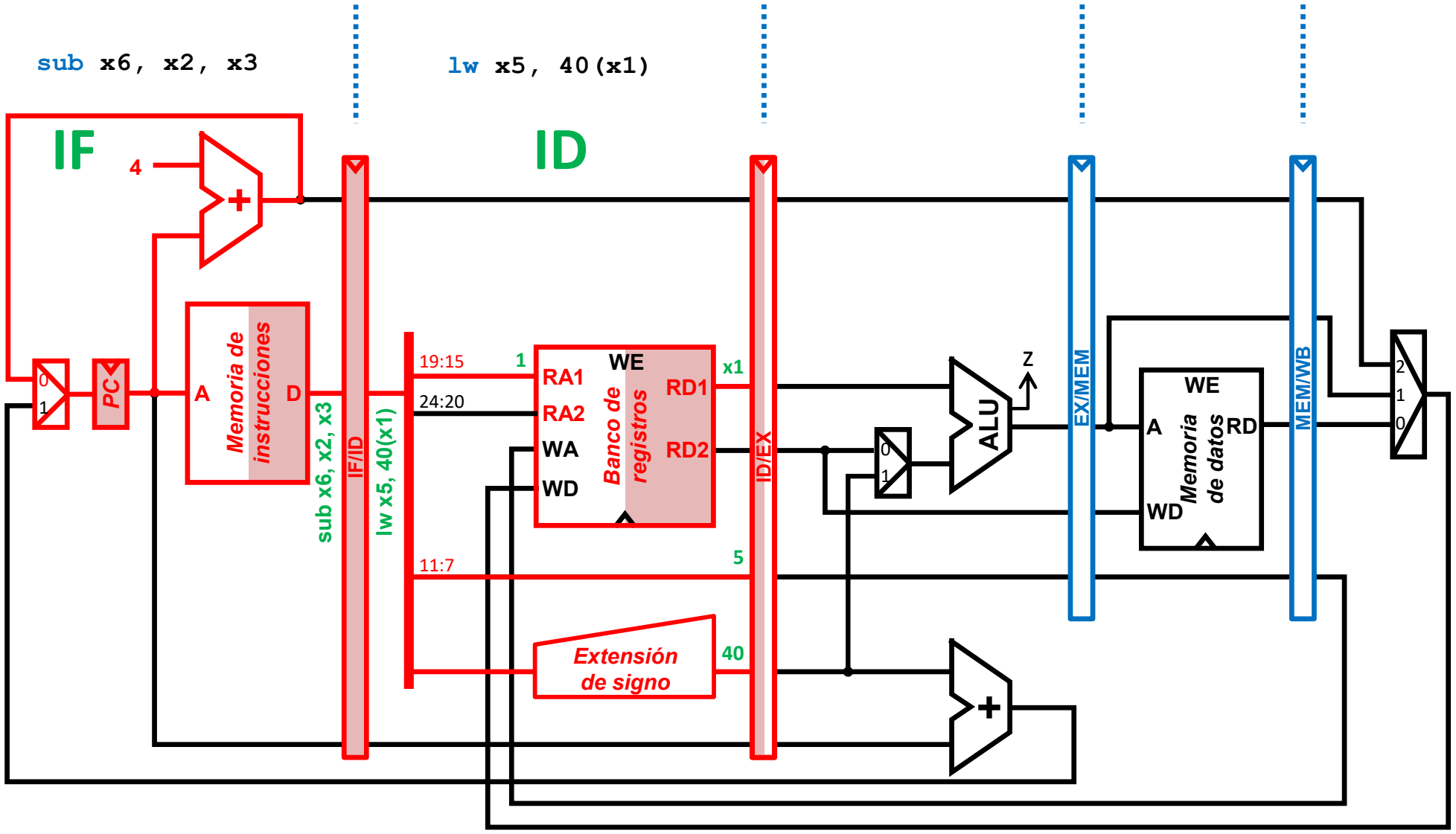
Diseño de la ruta de datos

Simulación: 1er. ciclo



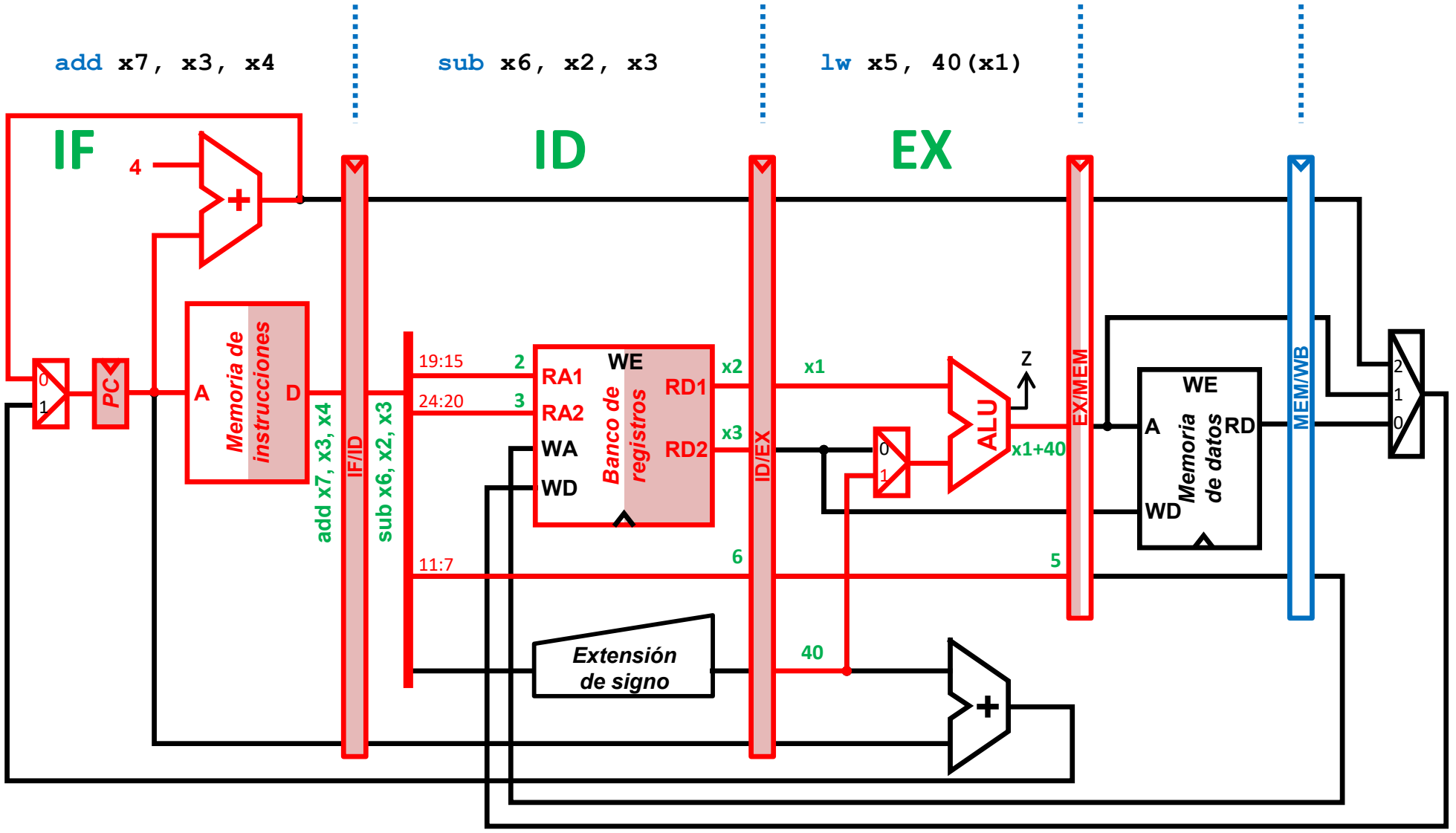
Diseño de la ruta de datos

Simulación: 2do. ciclo



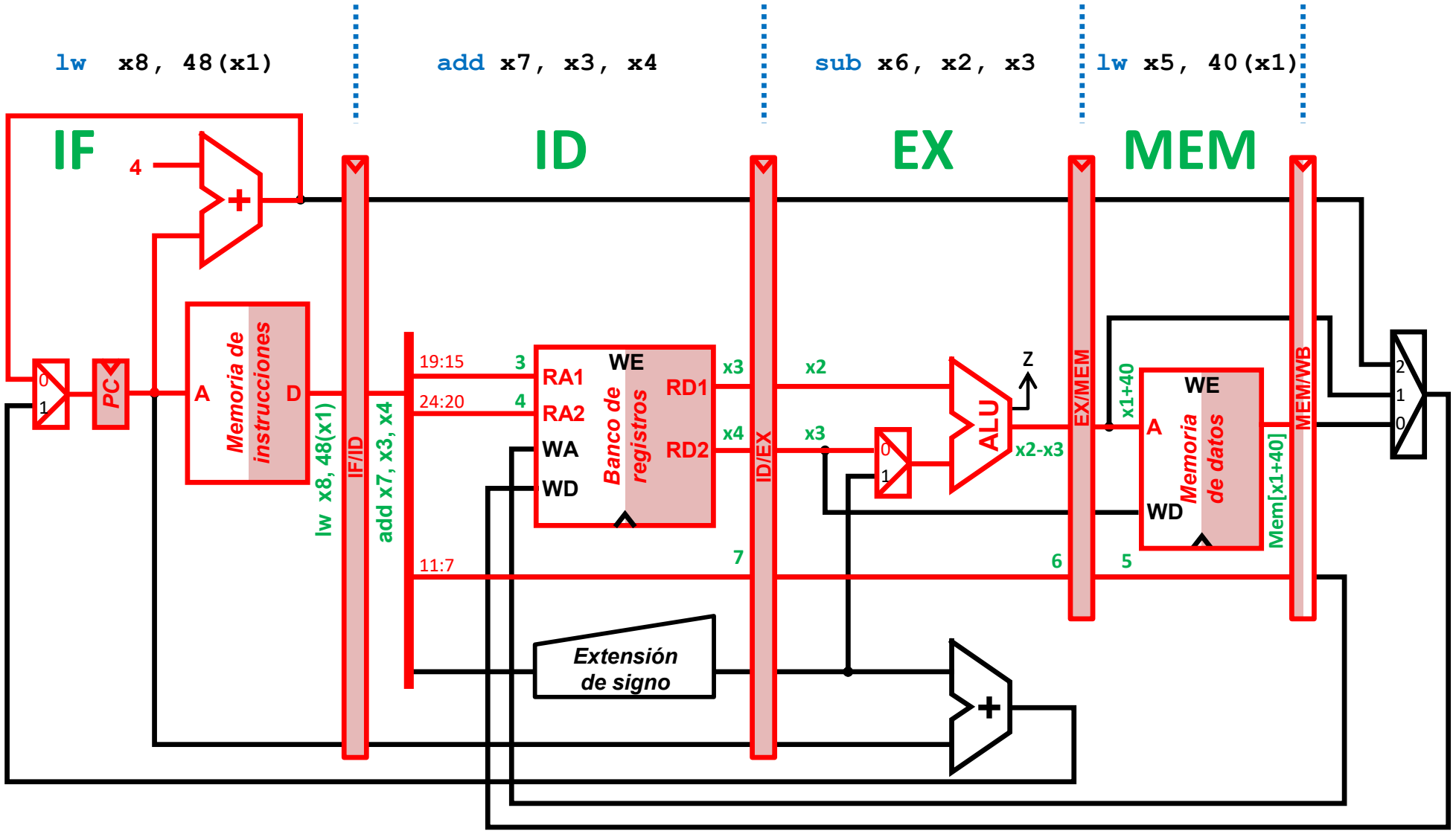
Diseño de la ruta de datos

Simulación: 3er. ciclo



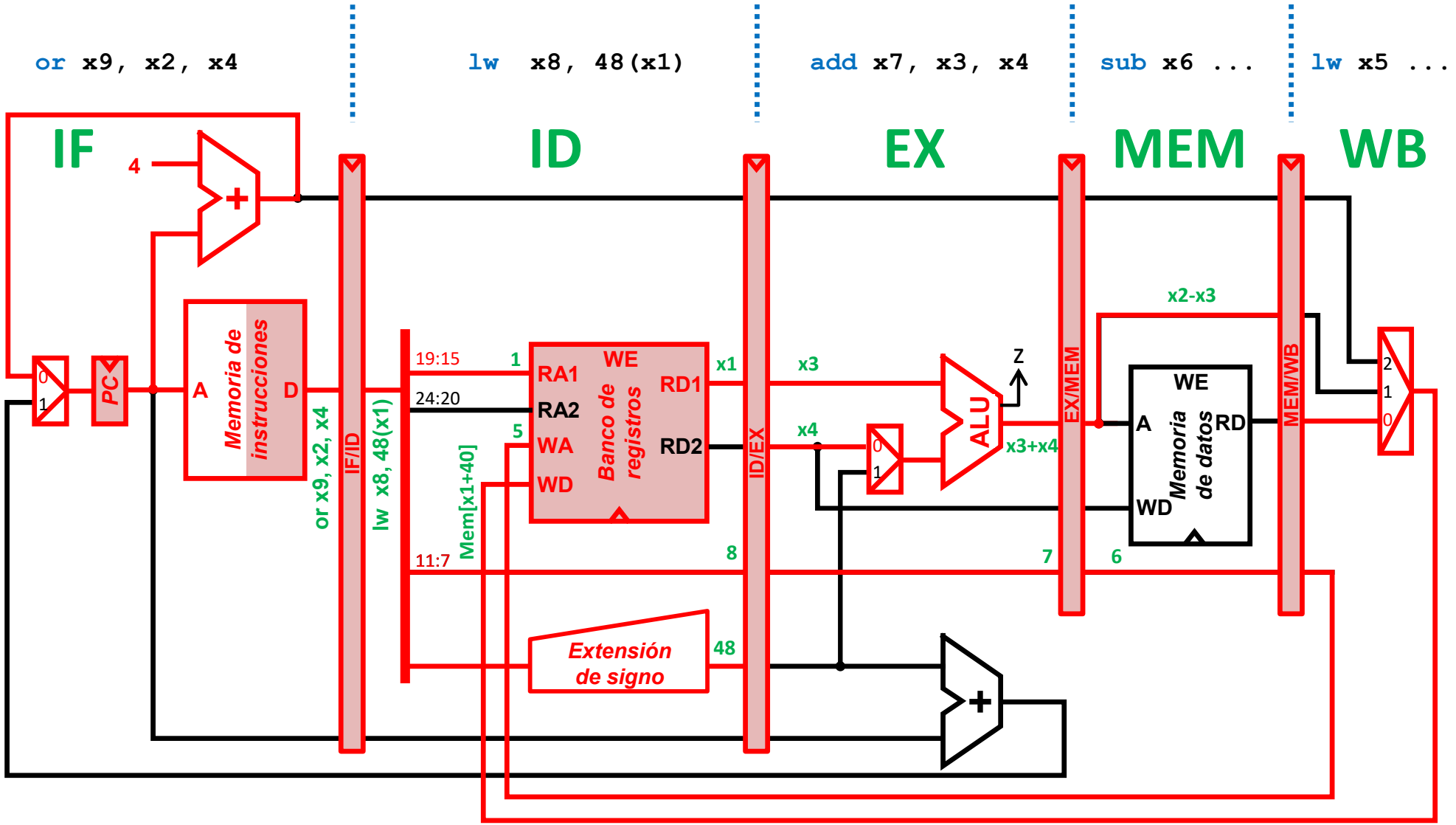
Diseño de la ruta de datos

Simulación: 4o. ciclo



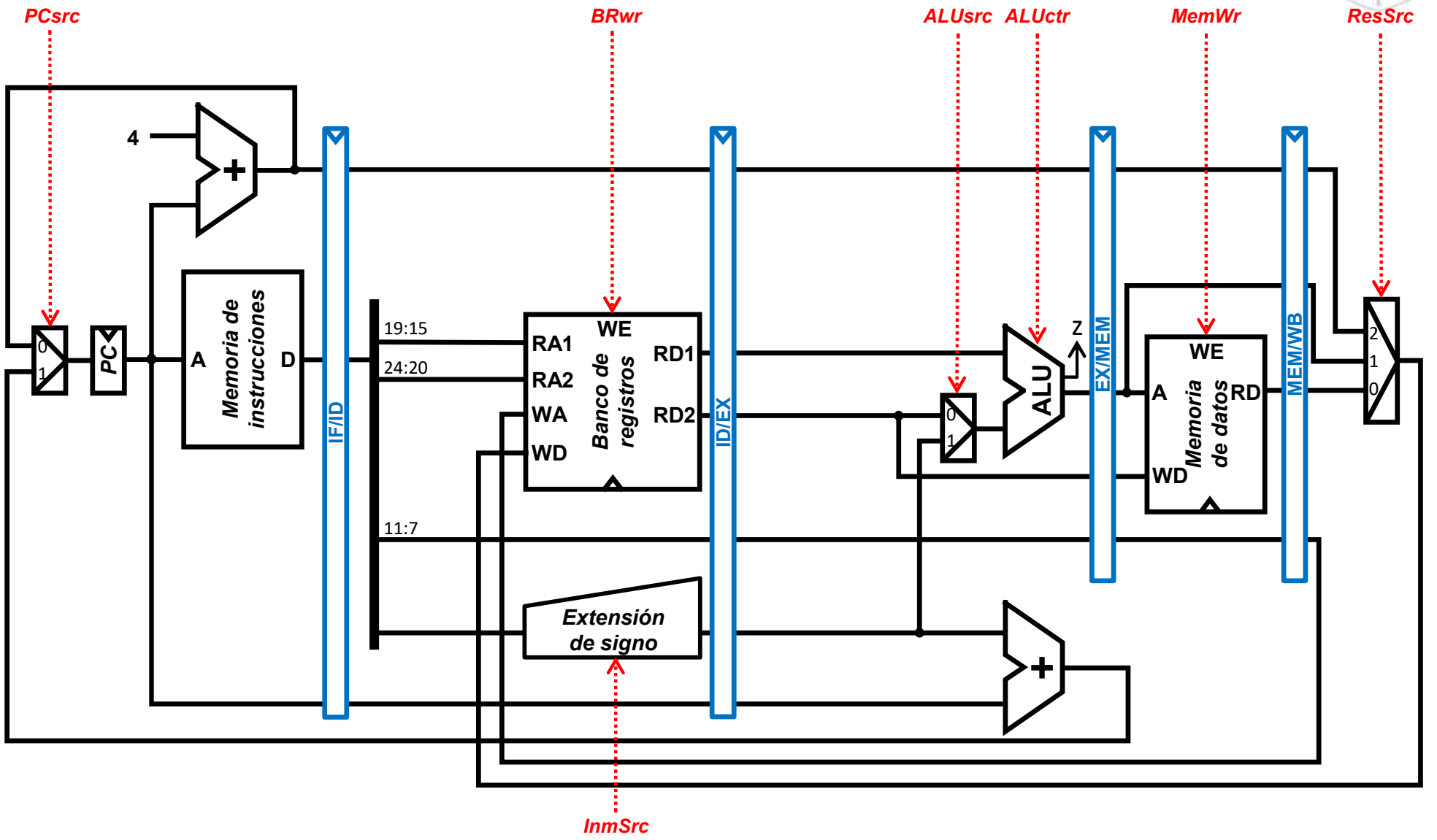
Diseño de la ruta de datos

Simulación: 5o. ciclo



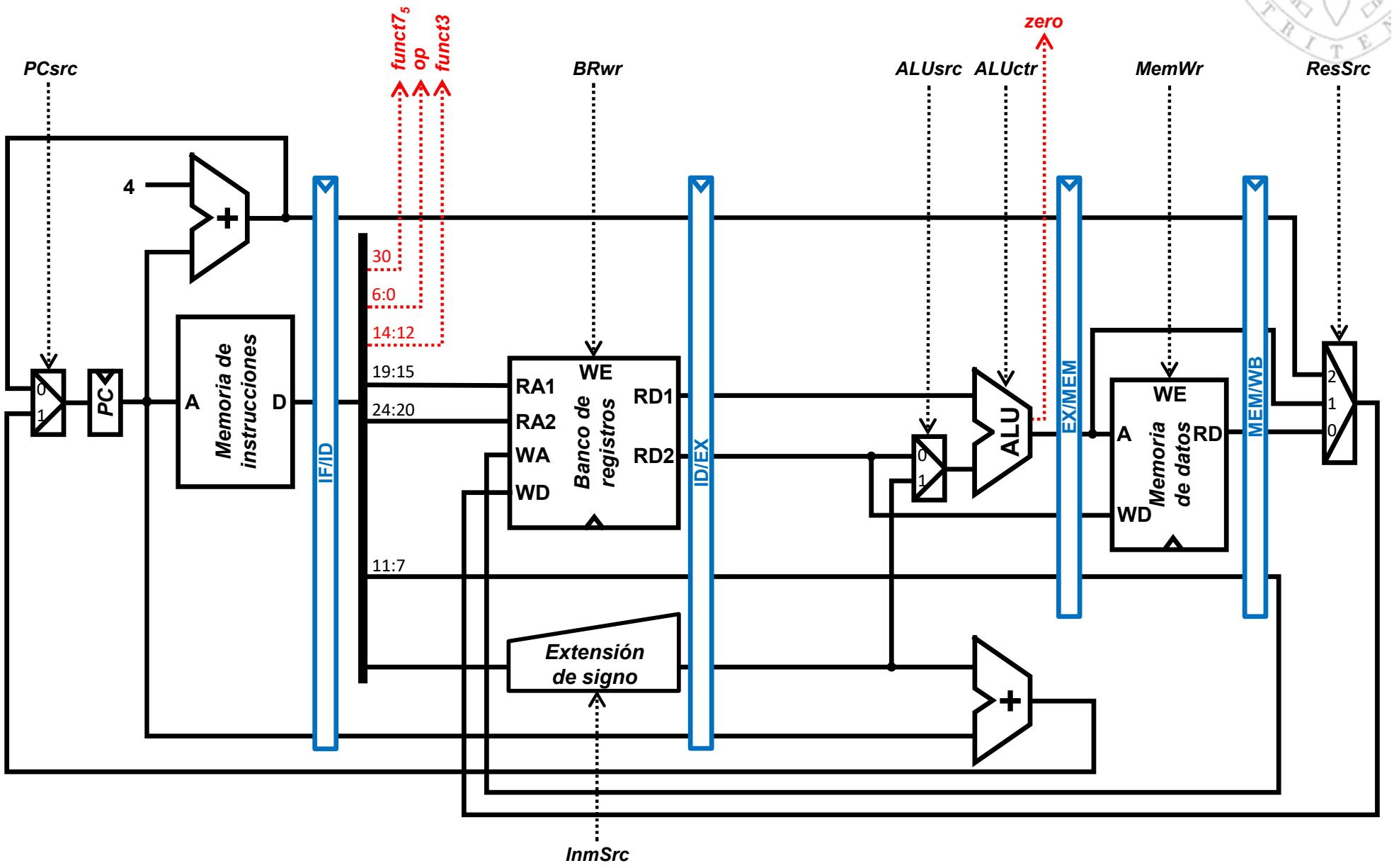
Diseño de la ruta de datos

Señales de control



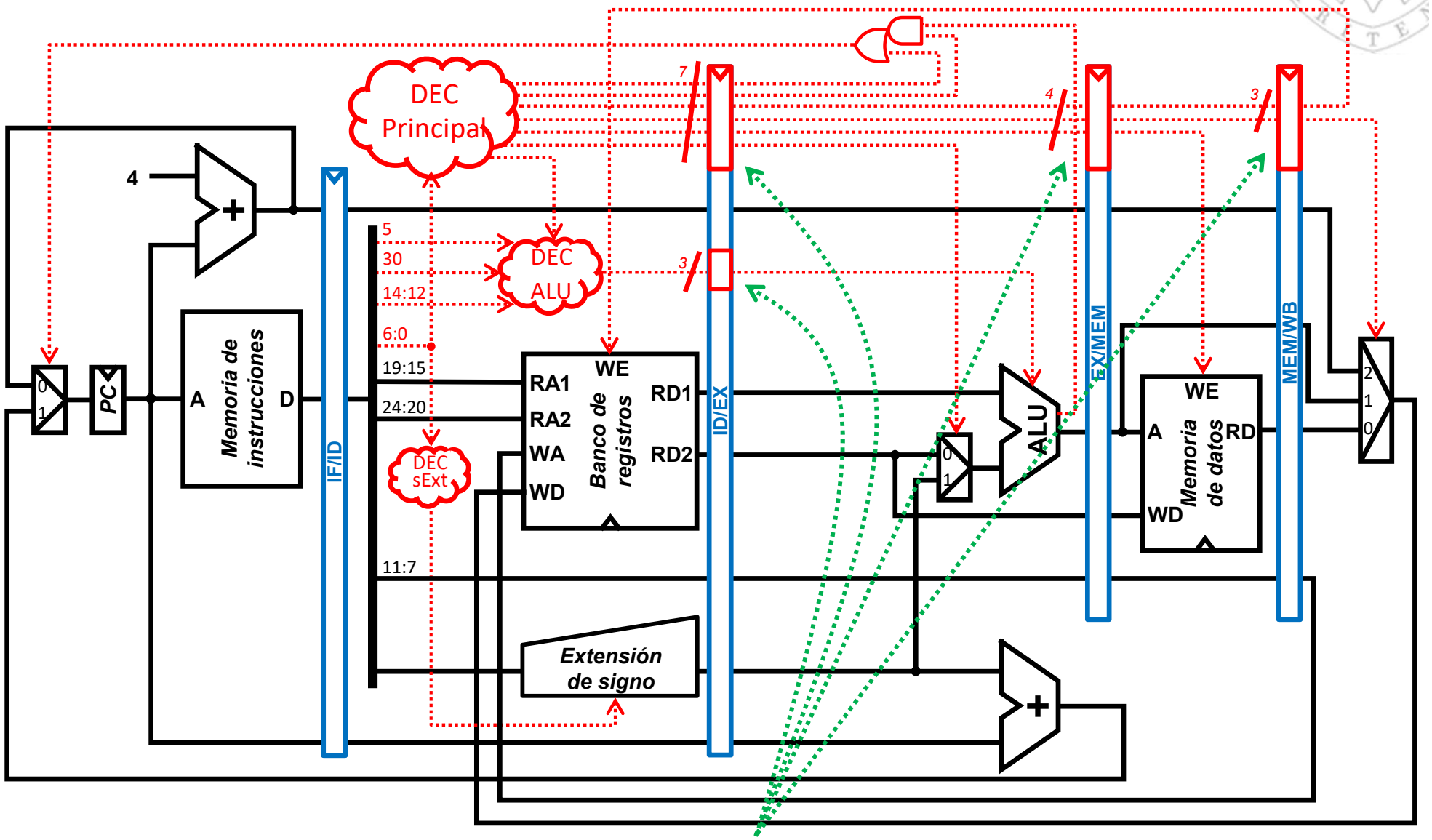
Diseño de la ruta de datos

Señales de estado



Diseño del controlador

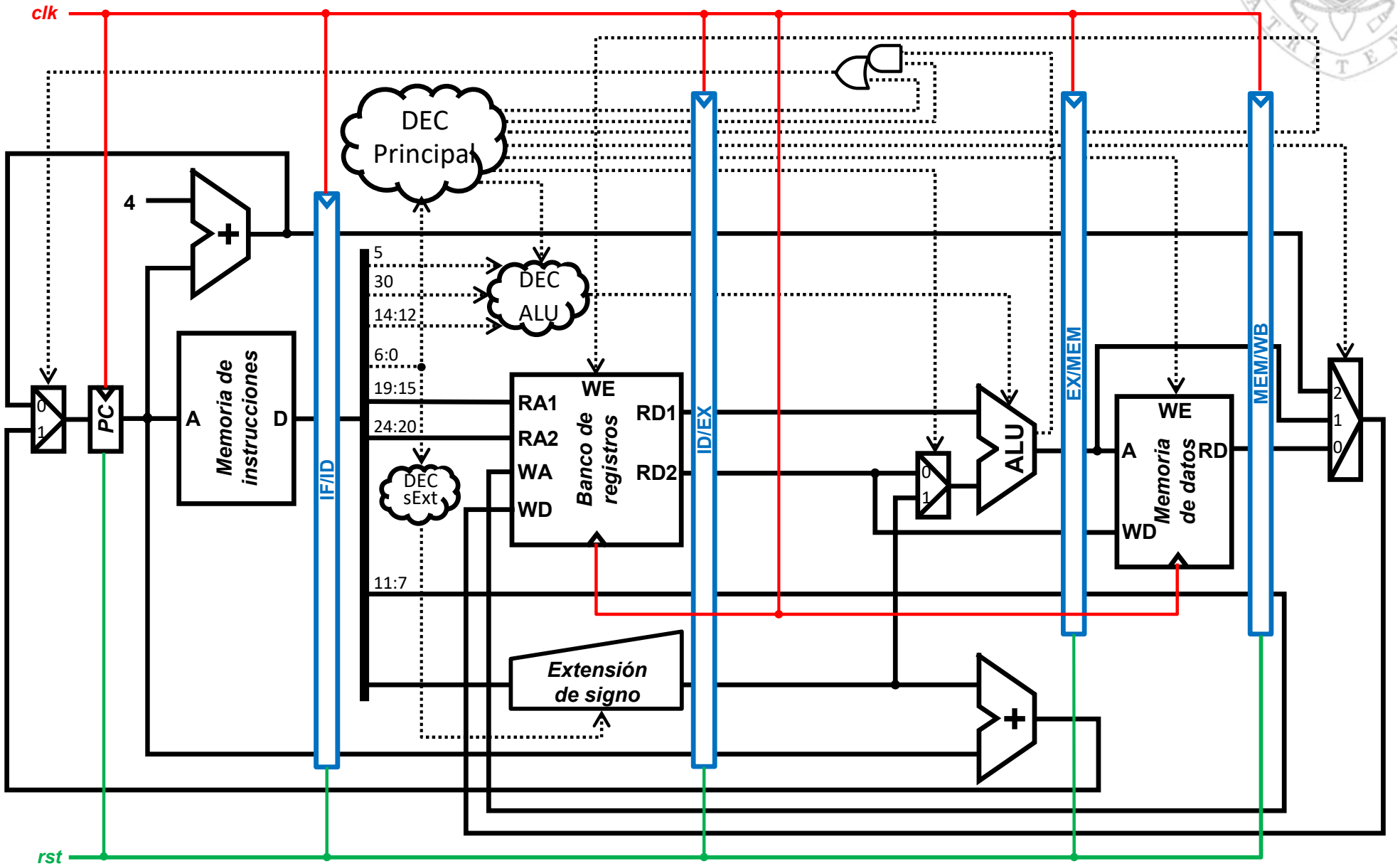
Conexión con el controlador



Se extienden los registros de segmentación para transmitir de etapa en etapa las señales de control que necesita cada una

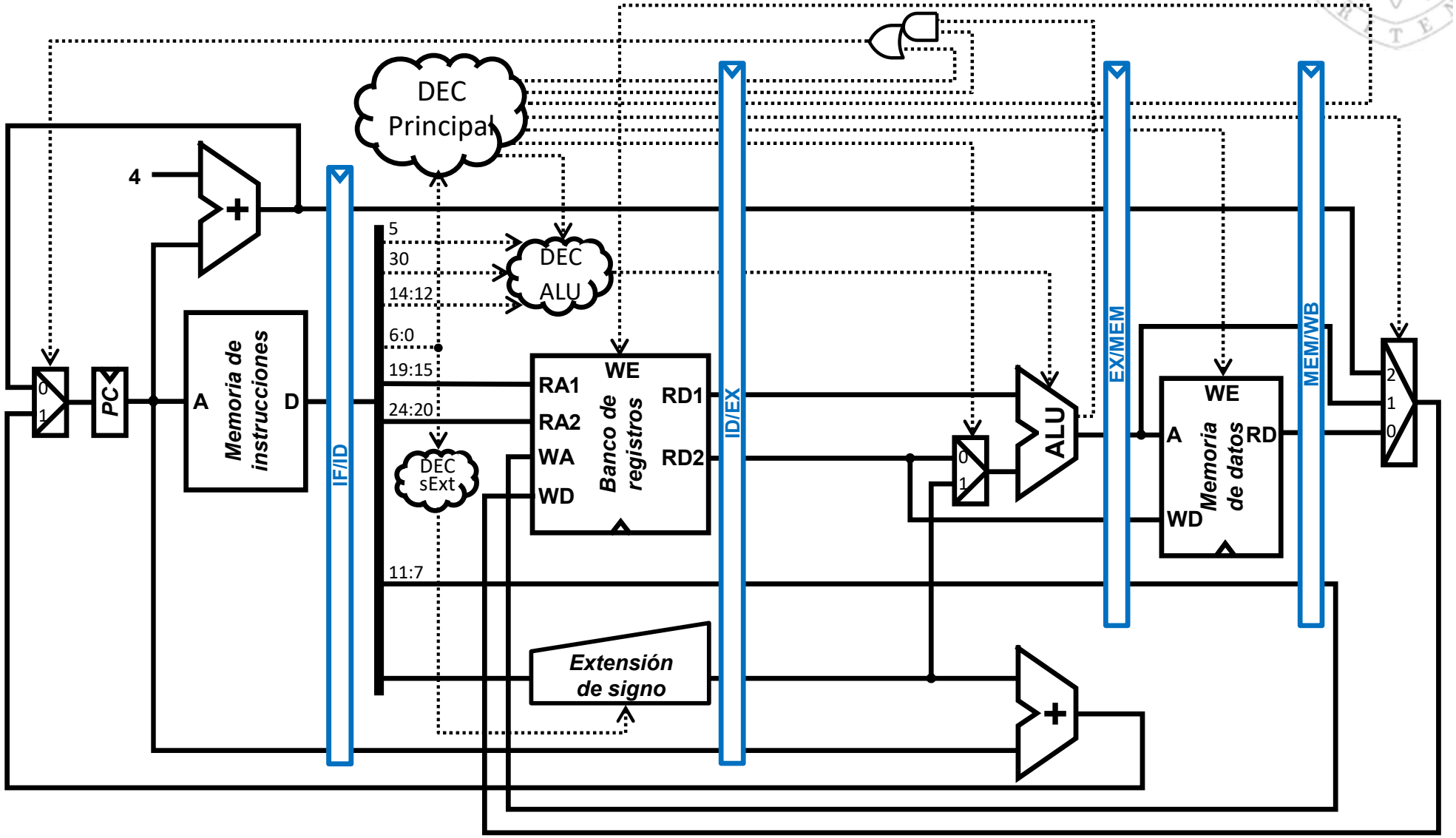
Procesador segmentado

Conexión del reloj y el reset



Procesador segmentado

Estructura del sistema completo



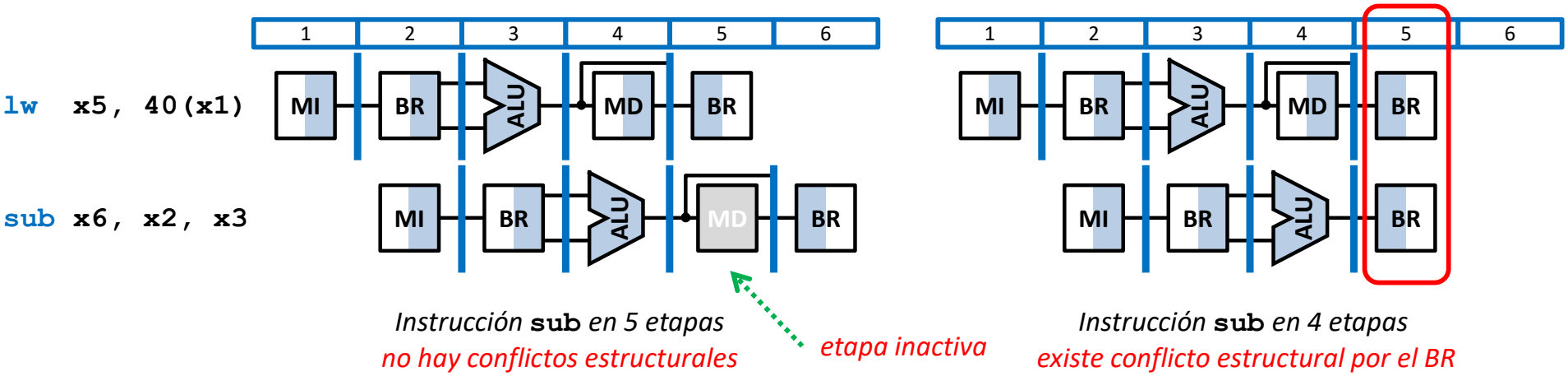
Conflictos

- En un procesador segmentado pueden aparecer conflictos (*hazards*) entre las instrucciones que están simultáneamente en ejecución.
  - No suceden en procesadores monociclo y multiciclo porque una instrucción no comienza a ejecutarse hasta que la anterior no ha finalizado.

- Tipos de conflictos :
  - Estructurales: una instrucción necesita usar un recurso hardware que está siendo utilizado por una instrucción lanzada con anterioridad.
  - De datos: una instrucción debe leer el dato de un registro que todavía no ha sido escrito por una instrucción lanzada con anterioridad.
  - De control: debe leerse de memoria la siguiente instrucción a ejecutar pero su dirección todavía no ha sido decidida/calculada por una instrucción de salto lanzada con anterioridad.

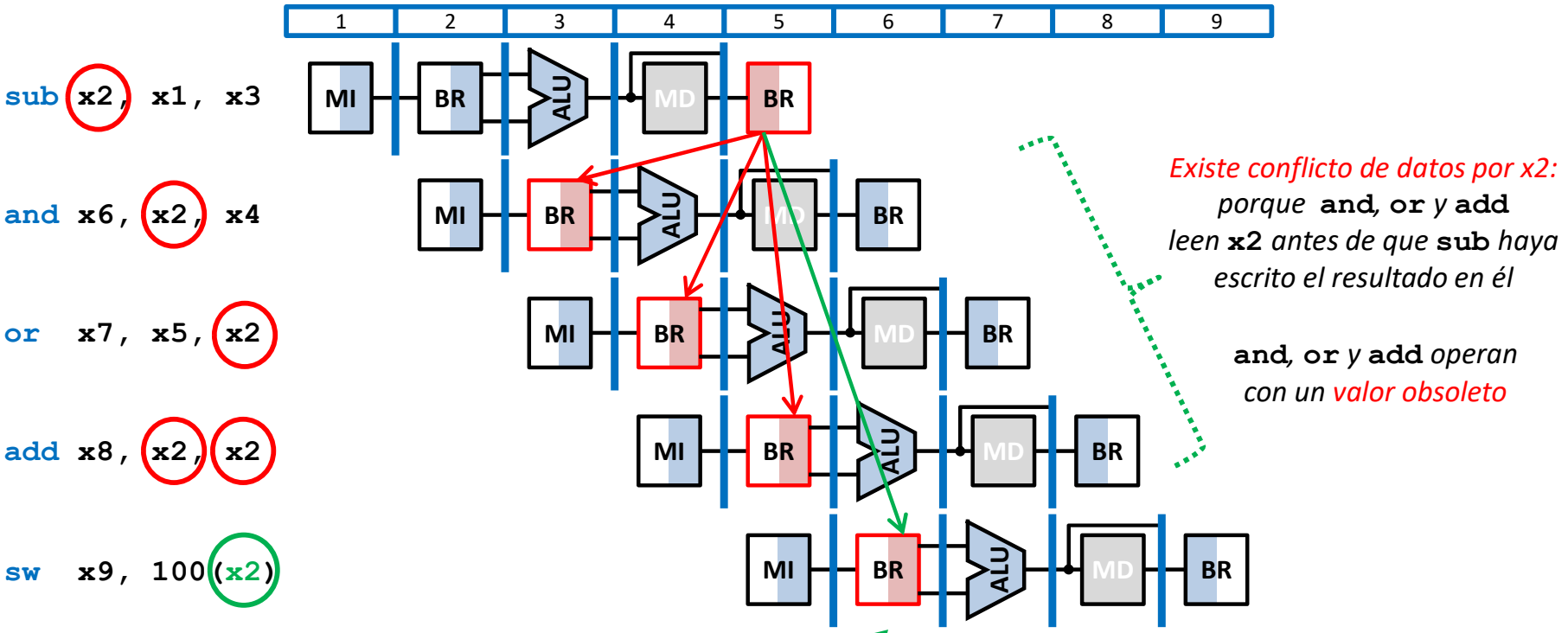
Conflictos estructurales

- Este procesador segmentado carece de conflictos estructurales porque:
  - No hay recursos compartidos.
    - Simultáneamente puede incrementarse el PC (etapa IF), así como calcular la dirección efectiva (etapa EX) y la condición de salto (etapa EX).
  - La memoria está dividida.
    - Simultáneamente puede leerse instrucciones (etapa IF) y datos (etapa MEM).
  - El banco de registros tiene triple puerto.
    - Simultáneamente pueden leerse 2 registros (etapa ID) y escribirse 1 (etapa WB).
  - Todas las instrucciones pasan por las 5 etapas.
    - Añadiendo etapas inactivas cuando sea necesario para evitar los conflictos.



Conflictos de datos

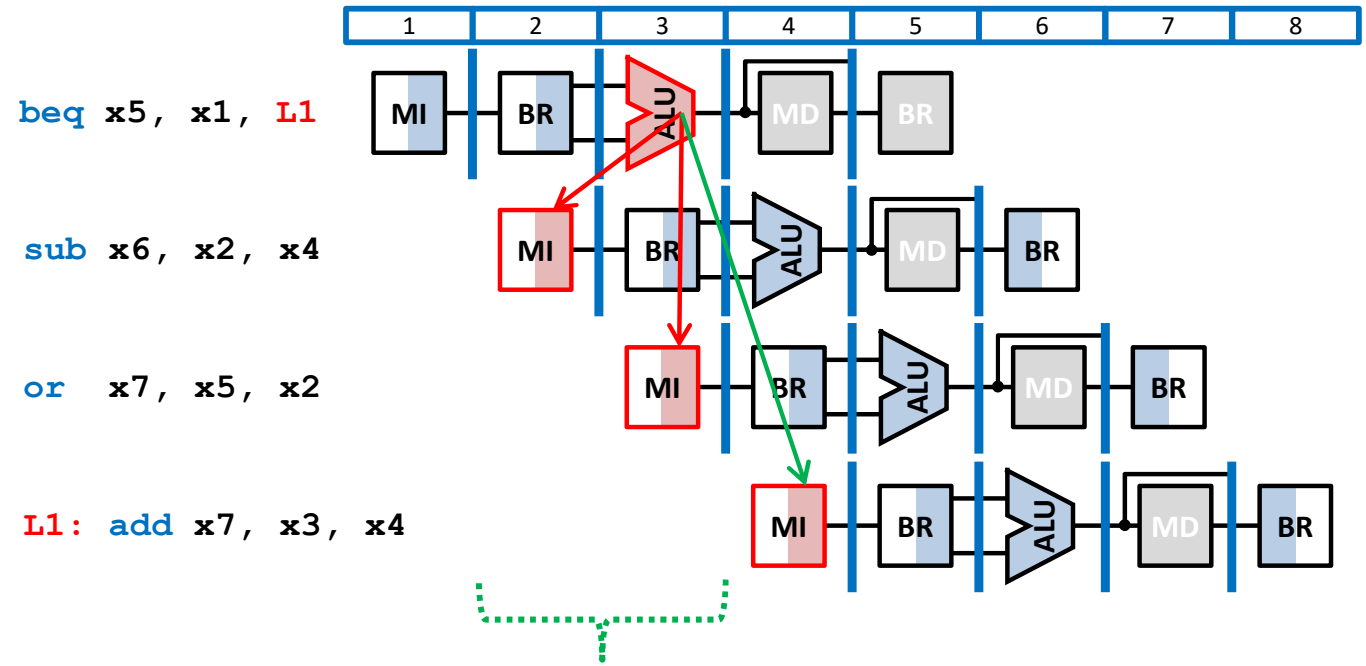
- Este procesador segmentado tiene conflictos de datos al ejecutar cualquier instrucción que lea un registro que sea escrito por cualquiera de las 3 instrucciones anteriores.



Conflictos de control

- Este procesador segmentado tiene conflictos de control al ejecutar instrucciones de salto, porque debe leerse la siguiente instrucción:
  - Antes de haber decidido si saltar o no (instrucción **`beq`**)
  - Antes de haber calculado la dirección de la instrucción a la que saltar en caso de hacerlo (instrucciones **`beq`** y **`jal`**)

```
   ...
   beq  x5, x1, L1
   sub  x6, x2, x4
   or   x7, x5, x2
   ...
L1:
   add  x7, x3, x4
...
```

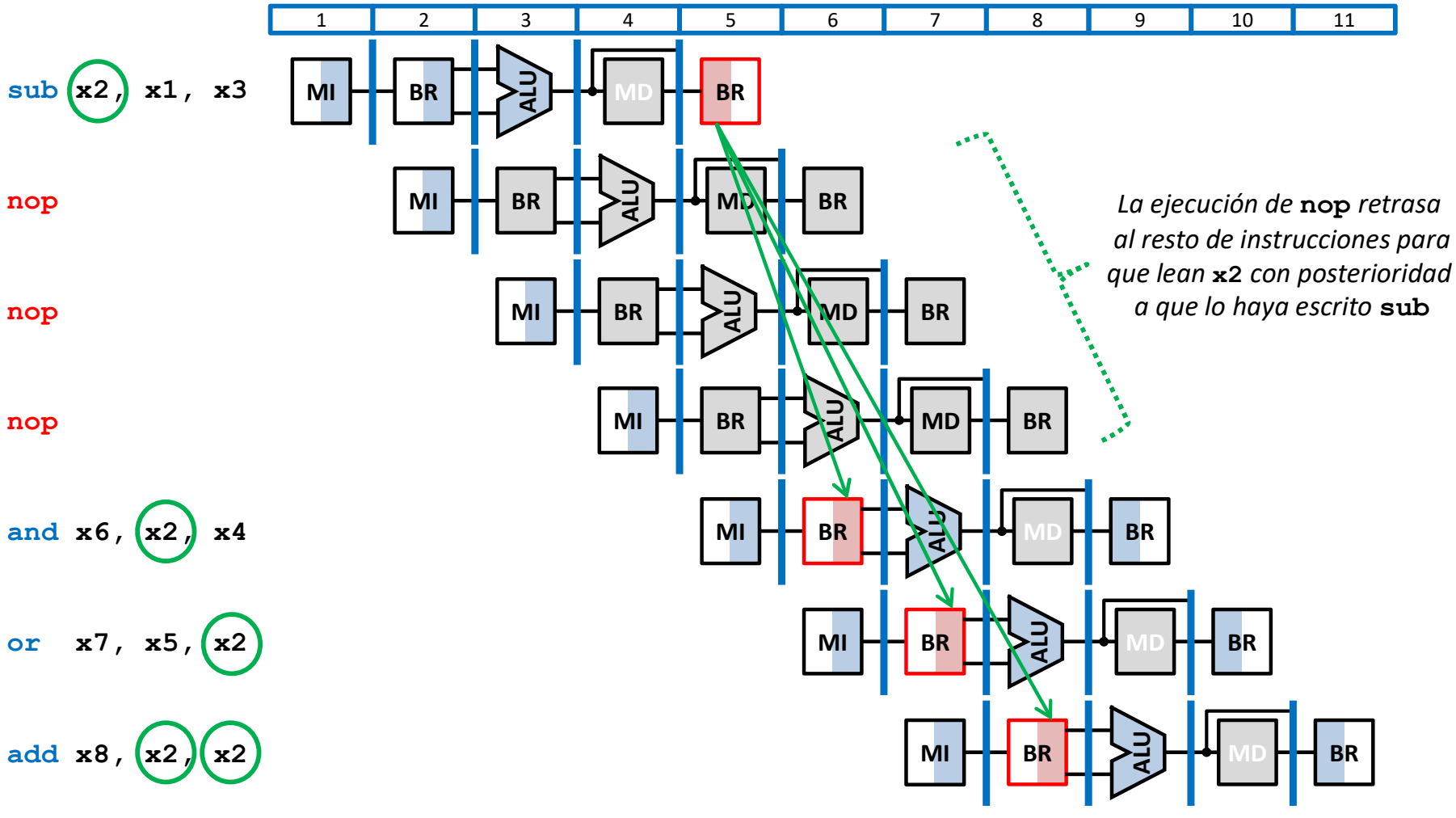


Existe conflicto de control porque se lanzan instrucciones antes de que **`beq`** *resuelva si se salta o no*

Conflictos de datos

Solución SW: inserción de nop (i)

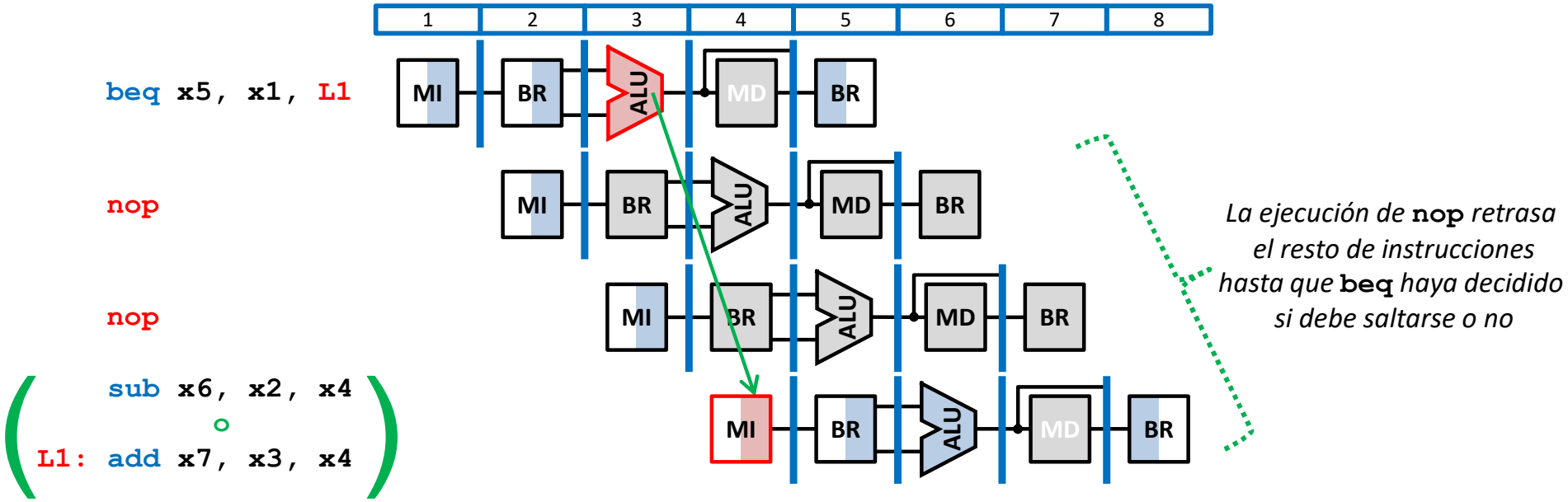
- Pueden resolverse por software insertando 1, 2 ó 3 instrucciones **nop** (según el caso) entre la instrucción que escribe el registro y la que lo lee.



Conflictos de control

Solución SW: inserción de nop (ii)

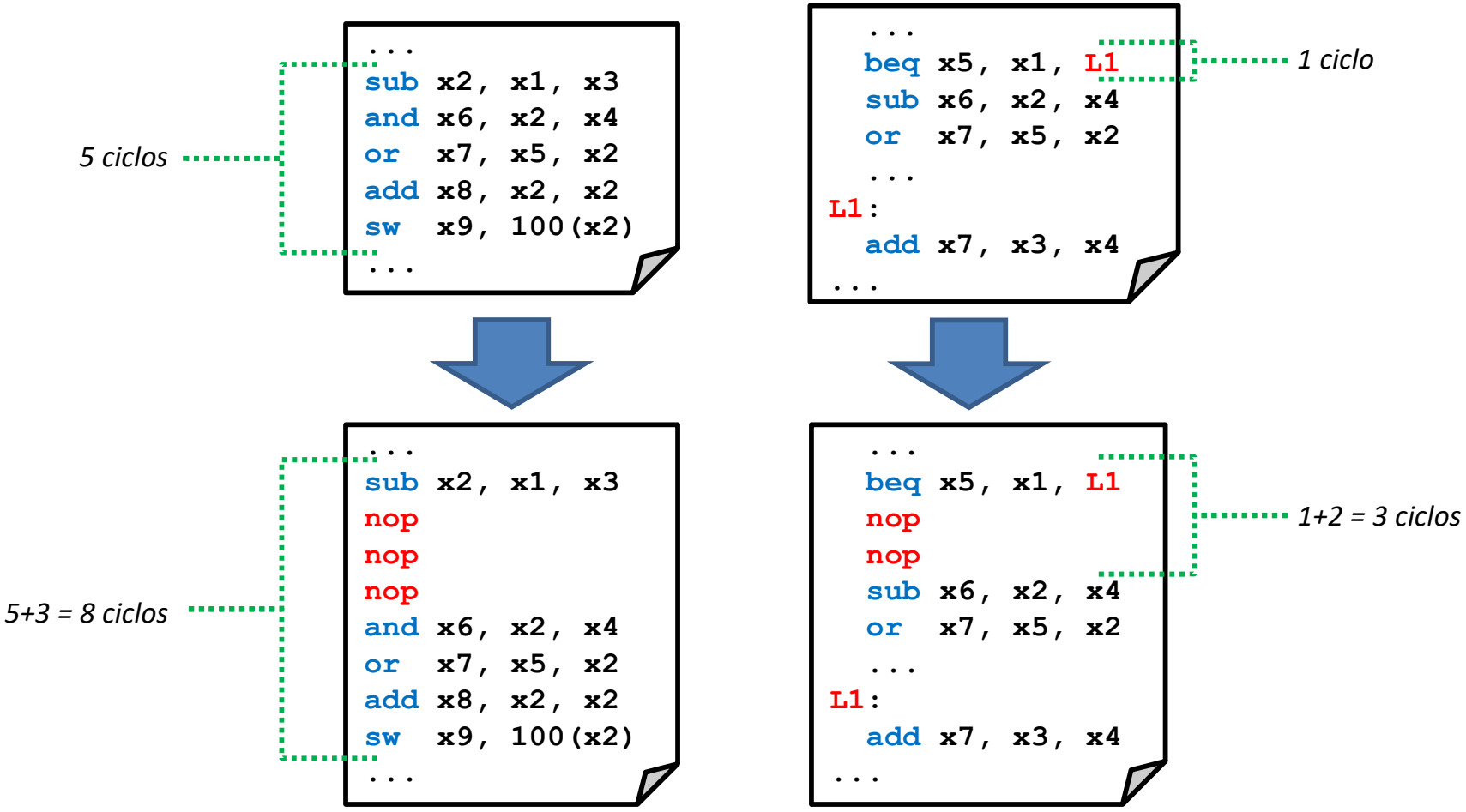
- Pueden resolverse por software insertando 2 instrucciones **nop** después de cada instrucción de salto.



Conflictos de datos y control

Solución SW: inserción de nop (iii)

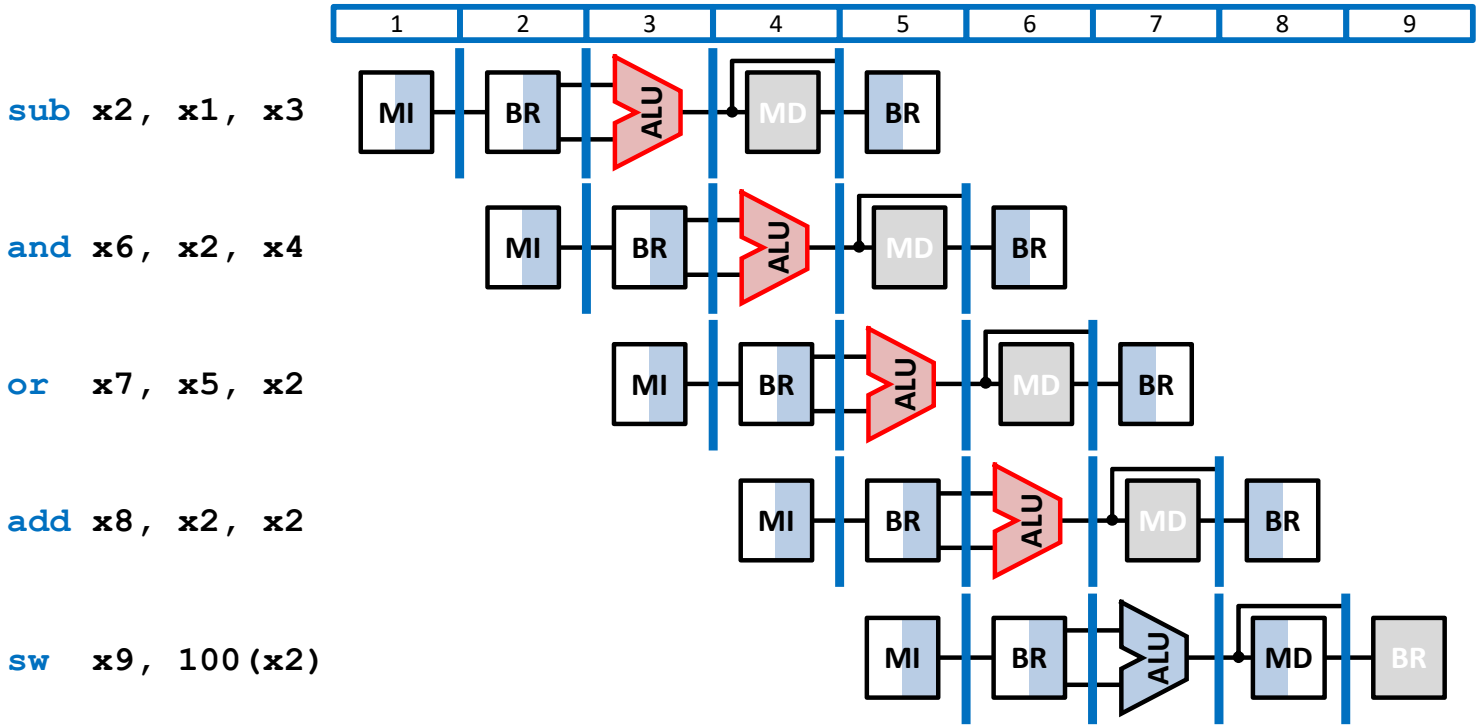
- La solución software tiene importantes desventajas:
  - Complica la programación porque requiere insertar **nop** adecuadamente.
  - La ejecución de cada **nop** penaliza el tiempo de ejecución en 1 ciclo.



Conflictos de datos

Solución HW: anticipación (i)

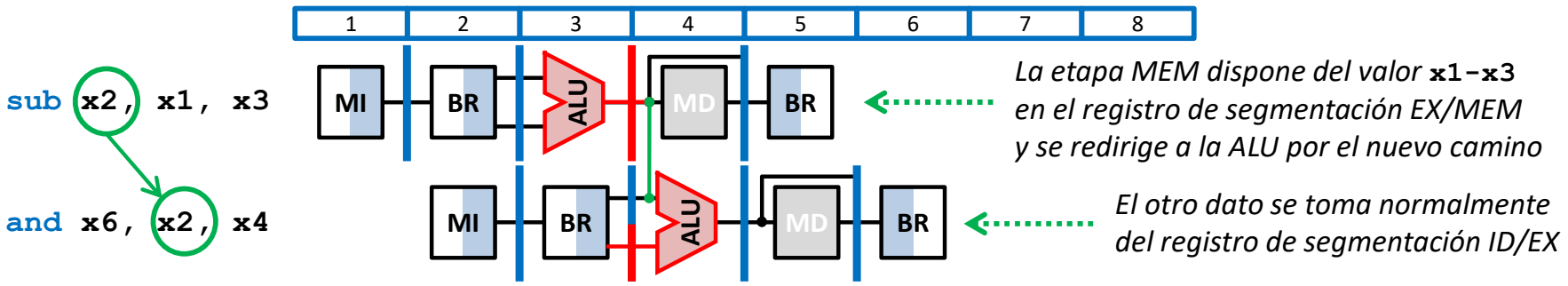
- Existe una solución hardware que evita la penalización ya que:
  - La instrucción **`sub`** usa la ALU en el ciclo 3 para calcular la resta.
  - Las instrucciones siguientes necesitan el dato en los ciclos 4, 5 y 6.
  - El dato está disponible desde el ciclo 4, y puede anticiparse (*forward*) sin necesidad de esperar a que pueda leerse del BR.



Conflictos de datos

Solución HW: anticipación (ii)

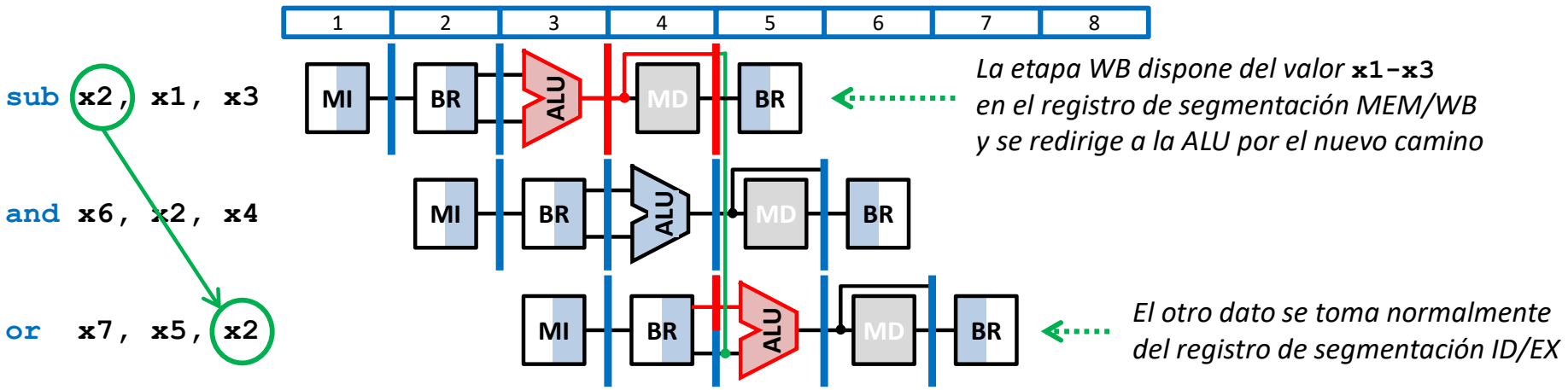
- Cada conflicto se resuelve de manera distinta:
  - **`sub - and`**: se añaden caminos para anticipar el dato desde la etapa MEM a cualquiera de las entradas de la ALU (etapa EX)



Conflictos de datos

Solución HW: anticipación (ii)

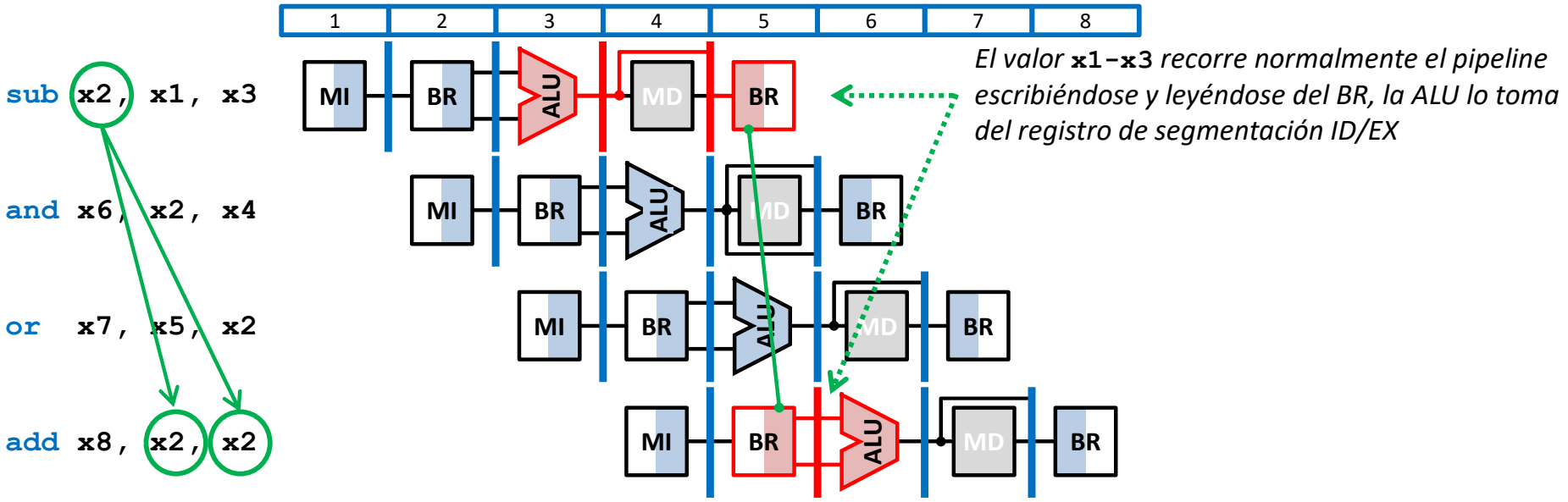
- Cada conflicto se resuelve de manera distinta:
  - **`sub - and`**: se añaden caminos para anticipar el dato desde la etapa MEM a cualquiera de las entradas de la ALU (etapa EX)
  - **`sub - or`**: se añaden caminos para anticipar el dato desde la etapa WB a cualquiera de las entradas de la ALU (etapa EX)



Conflictos de datos

Solución HW: anticipación (ii)

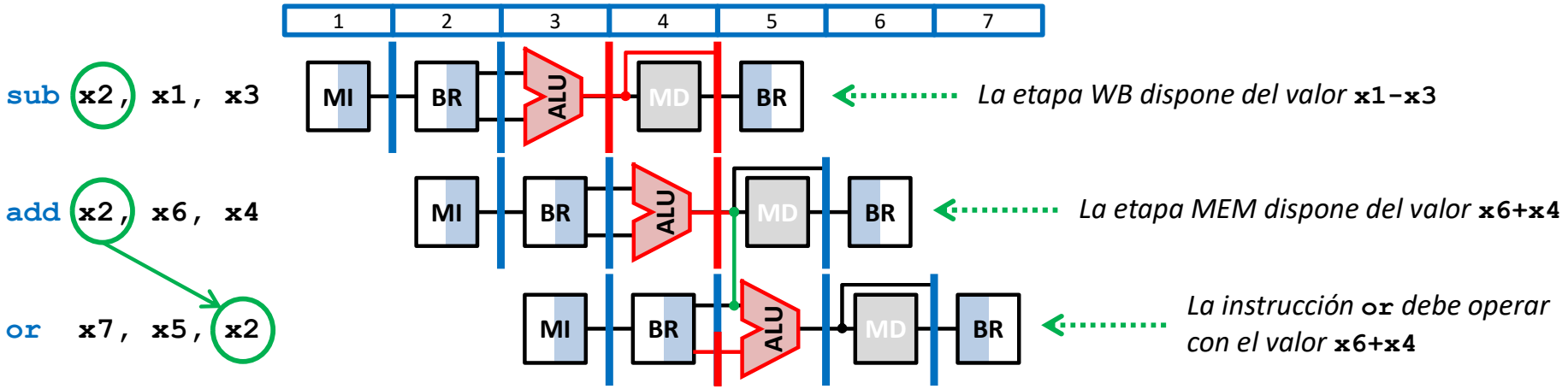
- Cada conflicto se resuelve de manera distinta:
  - **sub - and**: se añaden caminos para anticipar el dato desde la etapa MEM a cualquiera de las entradas de la ALU (etapa EX)
  - **sub - or**: se añaden caminos para anticipar el dato desde la etapa WB a cualquiera de las entradas de la ALU (etapa EX)
  - **sub - add**: se resuelve escribiendo el BR al final de primera mitad del ciclo de reloj para que pueda leerse durante la segunda mitad.



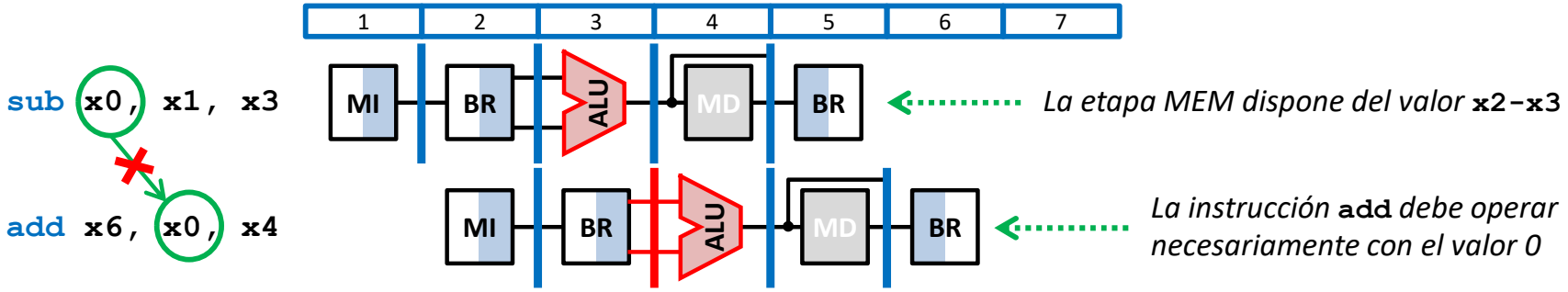
Conflictos de datos

Solución HW: anticipación (iii)

- En caso de que un registro pueda anticiparse desde MEM y WB:
  - deberá hacerlo desde la etapa MEM por disponer ésta del valor más actualizado del registro que causa el conflicto.



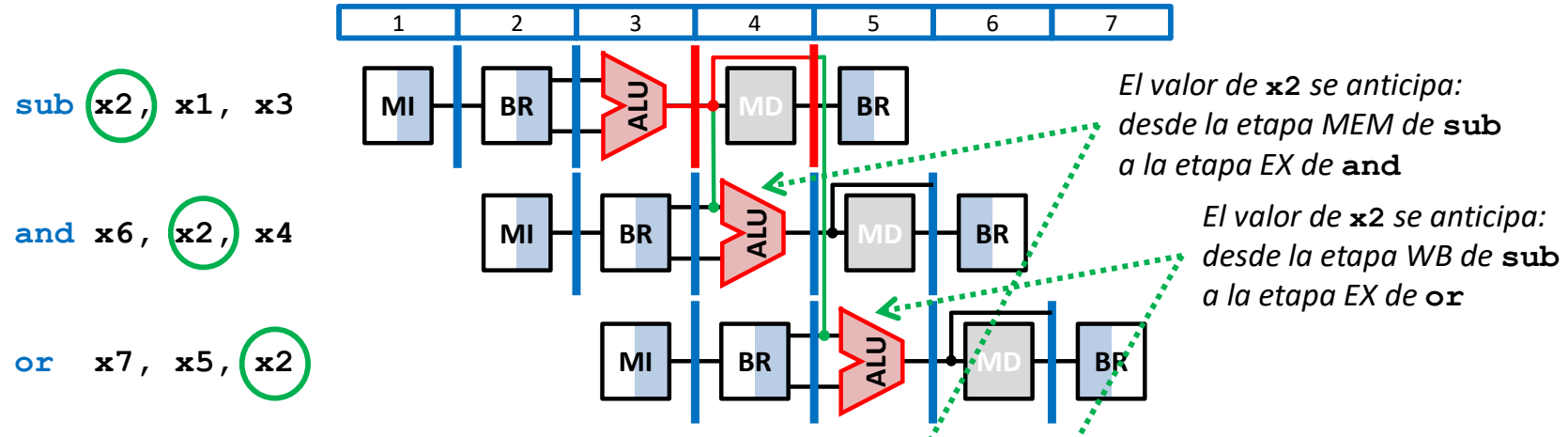
- Además, el registro **x0** nunca deberá anticiparse:



Conflictos de datos

Solución HW: anticipación (iv)

- En diagramas de ejecución simplificados, las anticipaciones se indican mediante dependencias entre etapas:



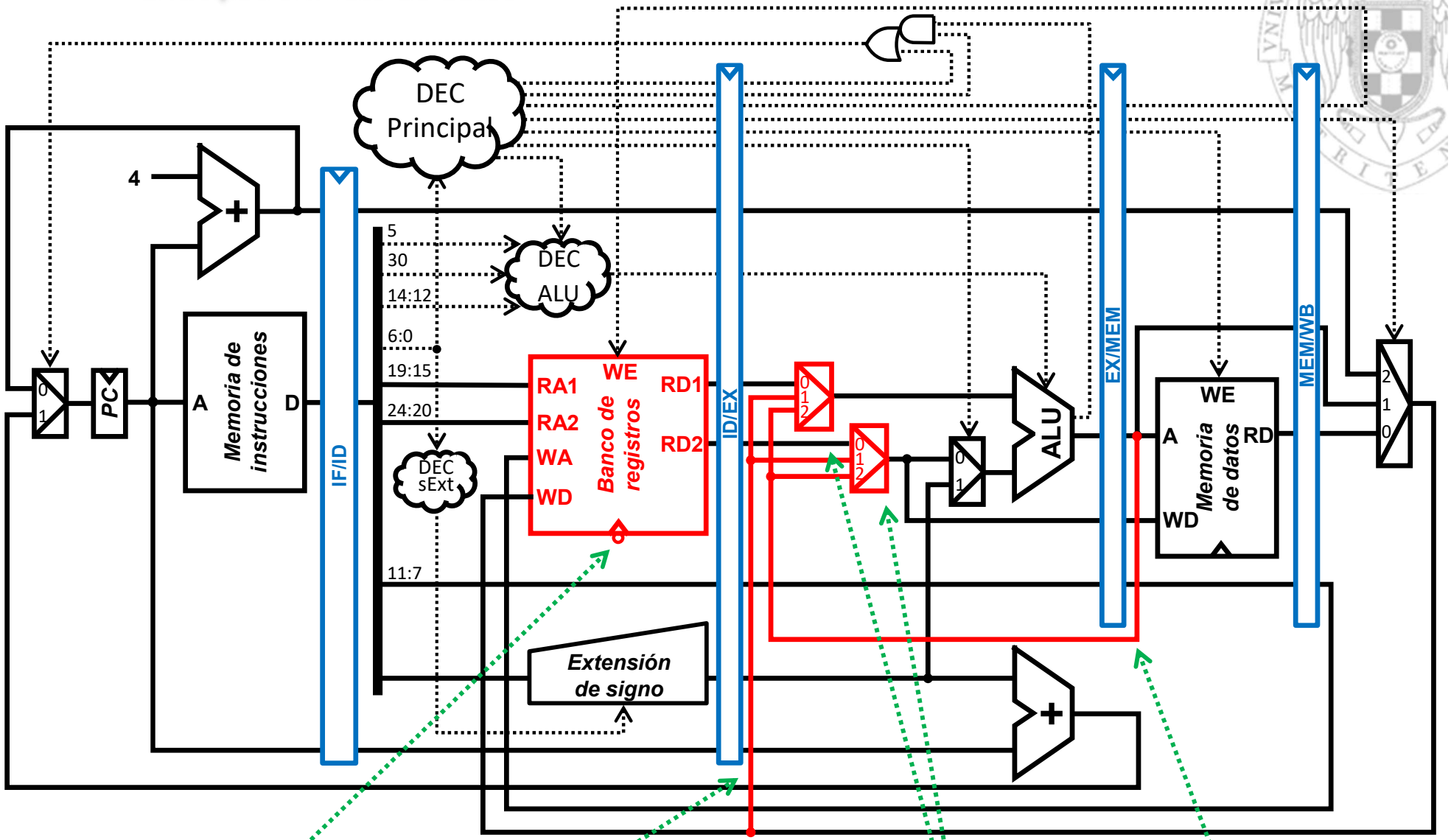
	1	2	3	4	5	6	7
sub x2, x1, x3	IF	ID	EX	M	WB		
and x6, x2, x4		IF	ID	EX	M	WB	
or x7, x5, x2			IF	ID	EX	M	WB

x2

x2

Procesador segmentado

+ anticipación: ruta de datos



Al invertir la entrada de reloj, el BR carga a flanco de bajada (mitad del ciclo a flanco de subida)

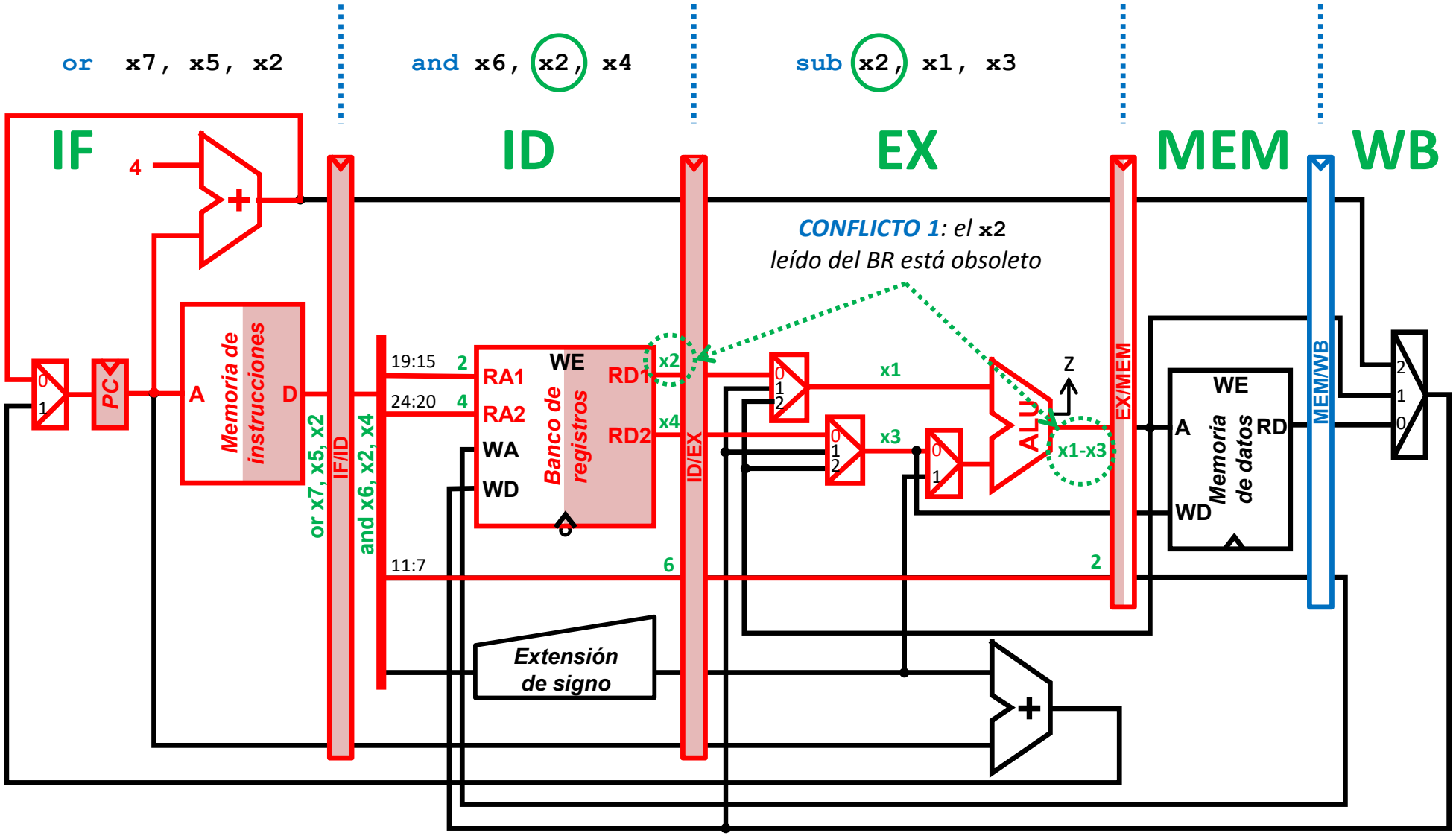
Se añade camino de anticipación desde la etapa WB a la EX

Se añade camino de anticipación desde la etapa MEM a la EX

Se añaden MUX a las entradas de la ALU

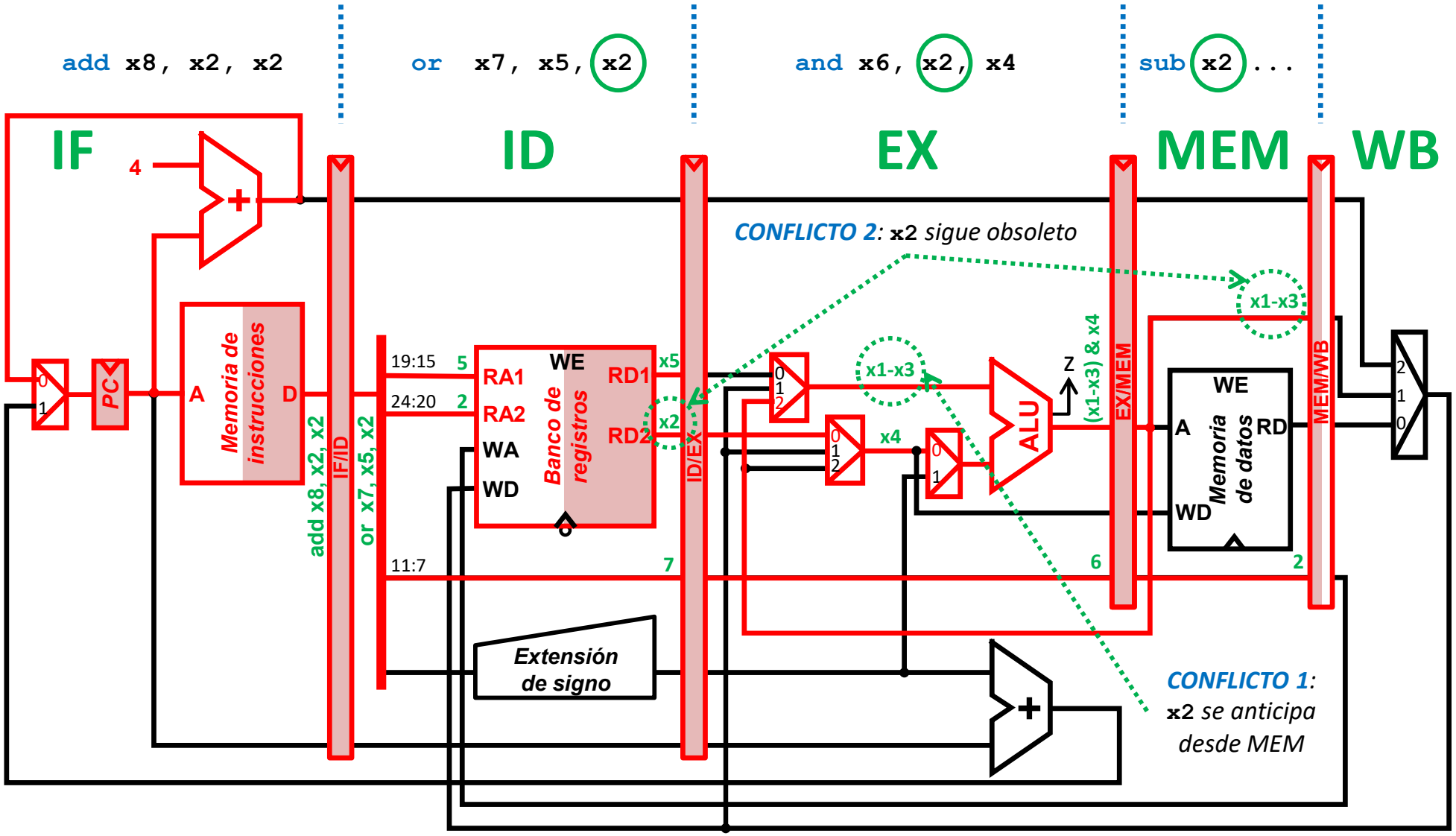
Procesador segmentado

Simulación de anticipación: 3er. ciclo



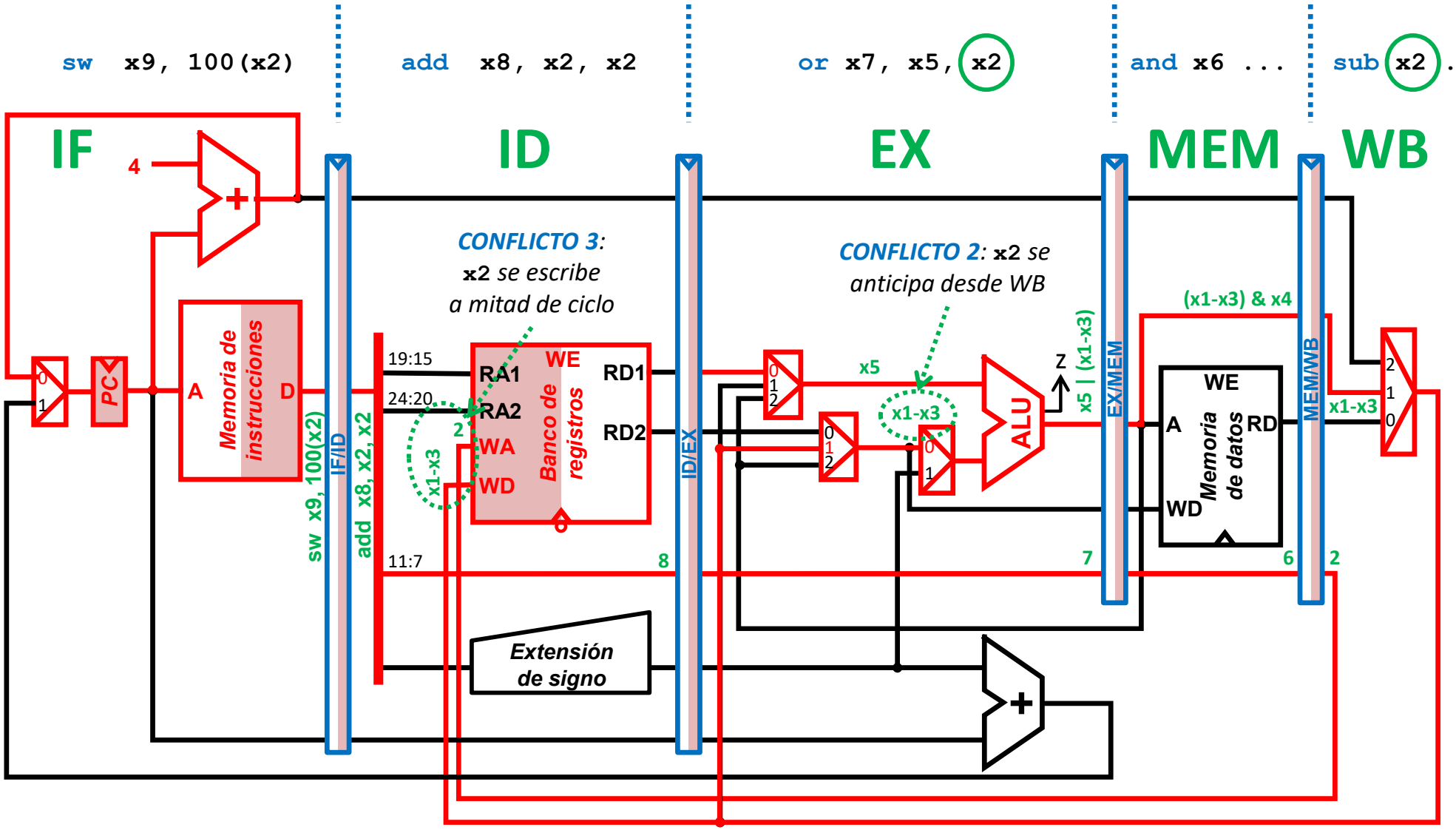
Procesador segmentado

Simulación de anticipación: 4o. ciclo



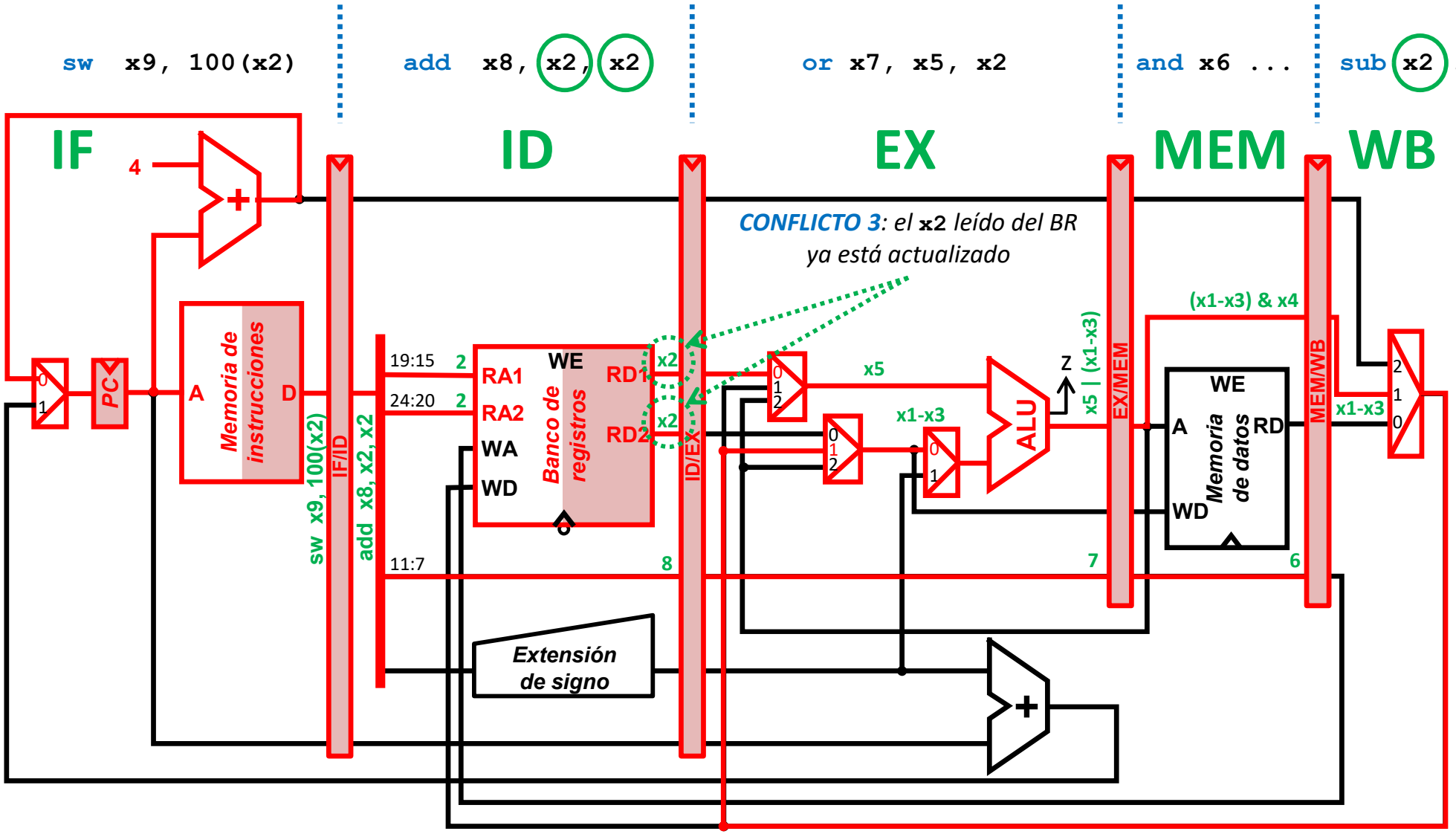
Procesador segmentado

Simulación de anticipación: 5o. ciclo (1ª mitad)



Procesador segmentado

Simulación de anticipación: 5o. ciclo (2ª mitad)



Procesador segmentado

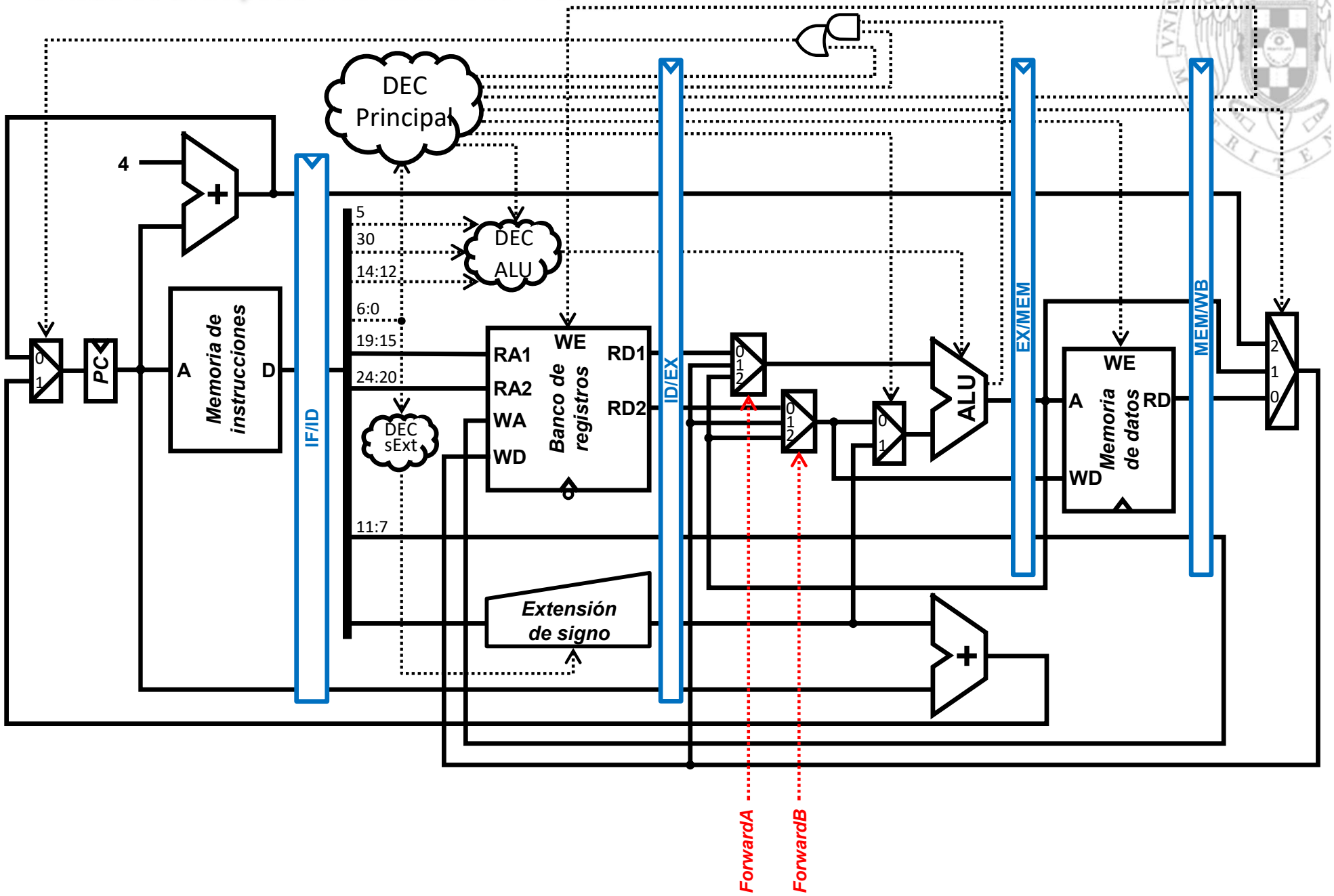
Unidad de anticipación

- La unidad de anticipación es un circuito combinacional que en todo ciclo controla los MUX de anticipación para que la ALU opere con datos:
  - Leídos del BR y disponibles en el registro de segmentación ID/EX.
  - Disponibles en registros de segmentación de etapas posteriores (EX/MEM o MEM/WB)

- Para realizar su función necesita conocer:
  - Rs1E: número del registro fuente 1 de la instrucción en la etapa EX.
  - Rs2E: número del registro fuente 2 de la instrucción en la etapa EX.
  - RdM: número del registro destino de la instrucción en la etapa MEM.
  - BRwrM: si la instrucción en la etapa MEM escribe en el BR.
  - RdW: número del registro destino de la instrucción en la etapa WB.
  - BRweW: Si la instrucción en la etapa WR escribe en el BR.

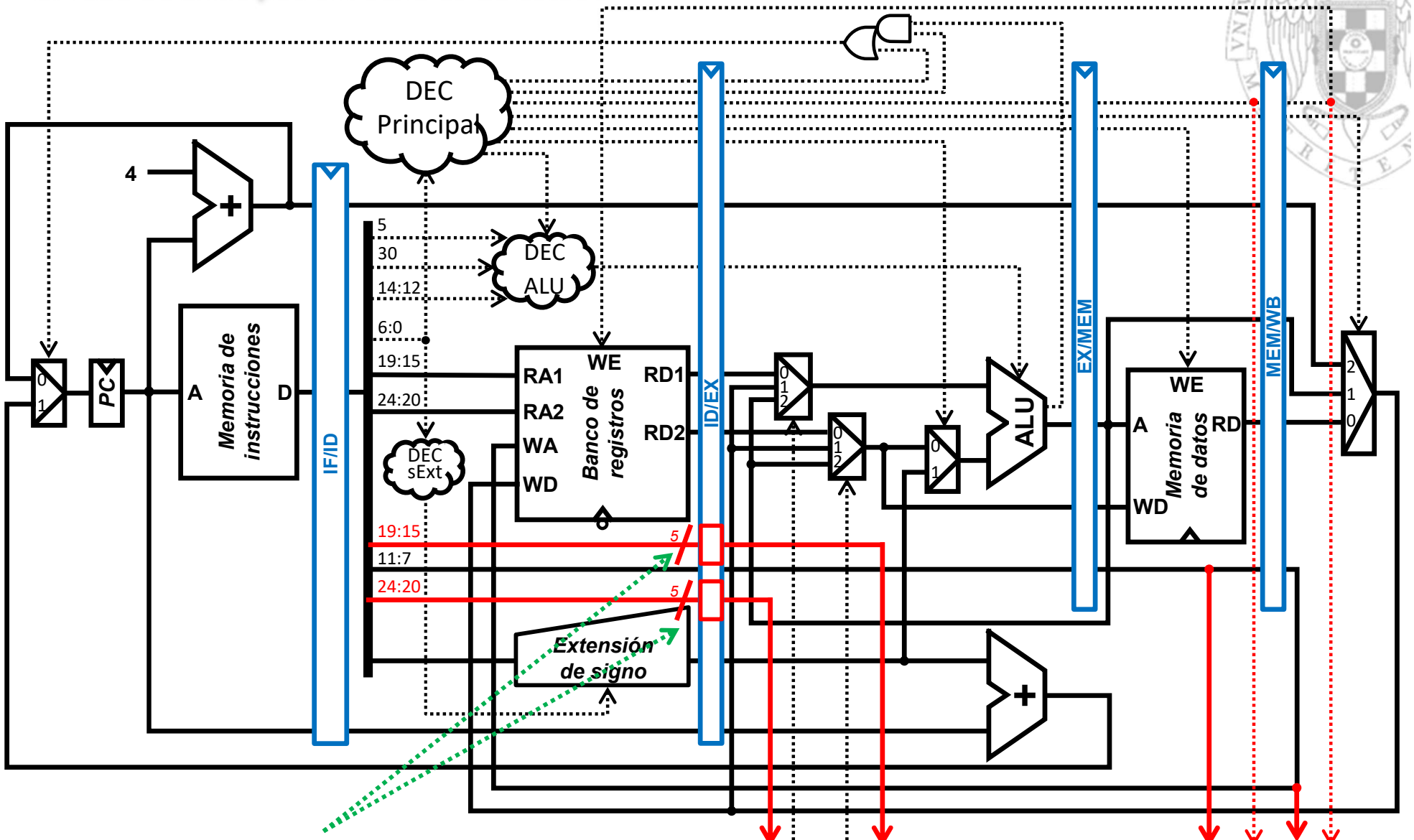
Procesador segmentado

+ Unidad de anticipación: señales de control



Procesador segmentado

+ Unidad de anticipación: señales de estado



Se extiende el registro de segmentación para transmitir a la etapa EX el número de cada registro fuente

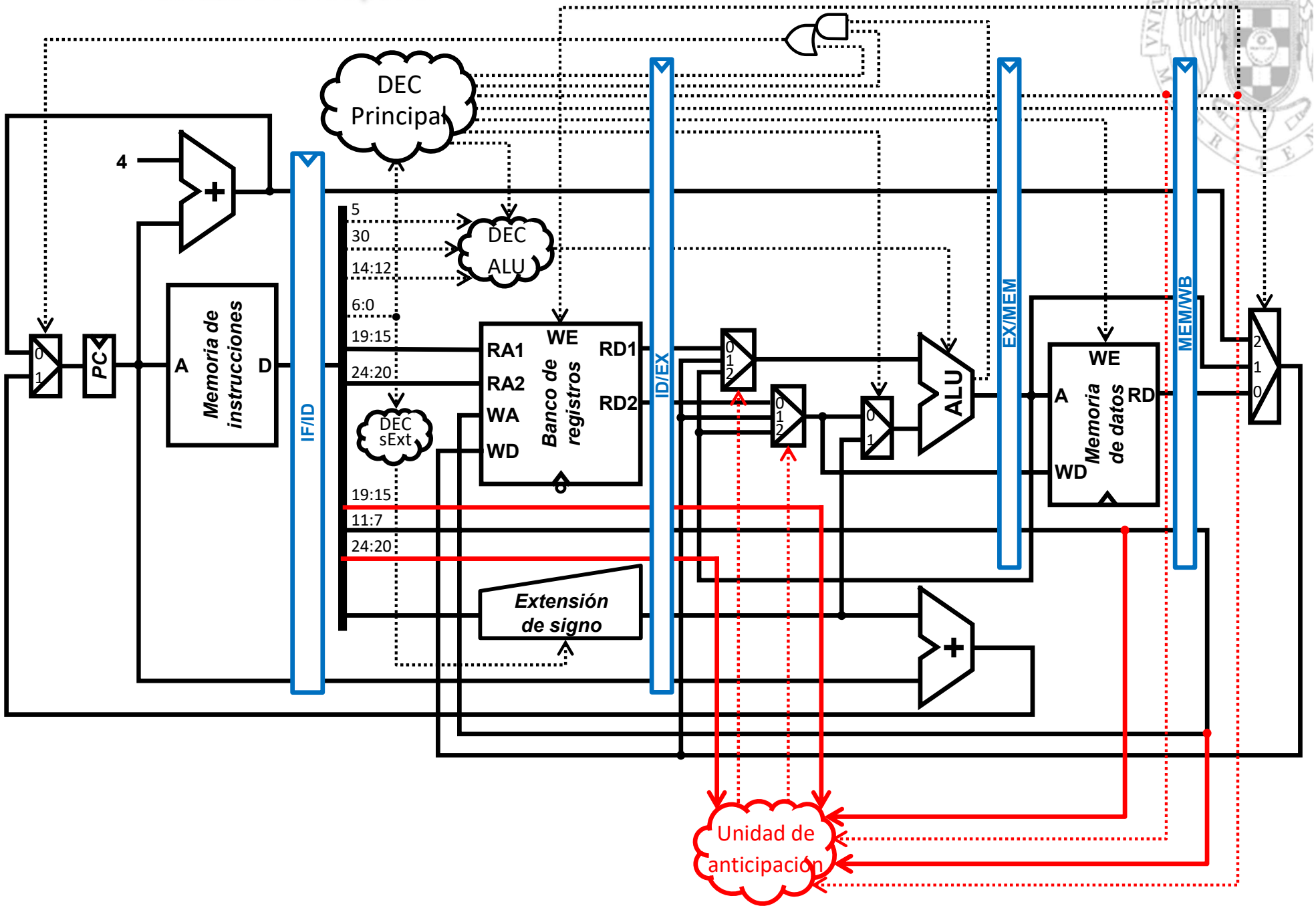
Registros fuente de la instrucción en la etapa EX

Ídem de la etapa MEM

Registro destino y señal de escritura en el BR de la instrucción en la etapa WB

Procesador segmentado

+ Unidad de anticipación



Procesador segmentado

Diseño de la unidad de anticipación (i)

- Un dato debe anticiparse por la entrada A de la ALU:
  - Desde la etapa MEM, si el registro destino en la etapa MEM (RdM) se escribirá (BRwrM) y coincide con el registro fuente en la etapa EX (Rs1E).

ForwardA ← *if* (BRwrM & (Rs1E = RdM)) *then* (10) ⟵ *Anticipar desde MEM*

Procesador segmentado

Diseño de la unidad de anticipación (i)

- Un dato debe anticiparse por la entrada A de la ALU:
  - Desde la etapa MEM, si el registro destino en la etapa MEM (RdM) se escribirá (BRwrM) y coincide con el registro fuente en la etapa EX (Rs1E).
  - Desde la etapa WB, si el registro destino en la etapa WB (RdW) se escribirá (BRwrW) y coincide con el registro fuente de la etapa EX (Rs1E) .
    - Esta condición solo se comprueba si la anterior no se cumple, porque cuando el dato puede anticiparse desde ambas etapas, se debe tomar de la etapa MEM.

```
ForwardA ← if (      BRwrM & (Rs1E = RdM) )   then  ( 10 )   <..... Anticipar desde MEM
           elsif(    BRwrW & (Rs1E = RdW) )   then  ( 01 )   <..... Anticipar desde WB
```

Procesador segmentado

Diseño de la unidad de anticipación (i)

- Un dato debe anticiparse por la entrada A de la ALU:
  - Desde la etapa MEM, si el registro destino en la etapa MEM (RdM) se escribirá (BRwrM) y coincide con el registro fuente en la etapa EX (Rs1E).
  - Desde la etapa WB, si el registro destino en la etapa WB (RdW) se escribirá (BRwrW) y coincide con el registro fuente de la etapa EX (Rs1E) .
    - Esta condición solo se comprueba si la anterior no se cumple, porque cuando el dato puede anticiparse desde ambas etapas, se debe tomar de la etapa MEM.
  - No debe anticiparse el registro x0 porque tiene valor constante 0.

ForwardA ← *if* ((Rs1E ≠ 0) & BRwrM & (Rs1E = RdM)) *then* (10) ⟵ *Anticipar desde MEM*
elsif((Rs1E ≠ 0) & BRwrW & (Rs1E = RdW)) *then* (01) ⟵ *Anticipar desde WB*

Procesador segmentado

Diseño de la unidad de anticipación (i)

- Un dato debe anticiparse por la entrada A de la ALU:
  - Desde la etapa MEM, si el registro destino en la etapa MEM (RdM) se escribirá (BRwrM) y coincide con el registro fuente en la etapa EX (Rs1E).
  - Desde la etapa WB, si el registro destino en la etapa WB (RdW) se escribirá (BRwrW) y coincide con el registro fuente de la etapa EX (Rs1E) .
    - Esta condición solo se comprueba si la anterior no se cumple, porque cuando el dato puede anticiparse desde ambas etapas, se debe tomar de la etapa MEM.
  - No debe anticiparse el registro x0 porque tiene valor constante 0.
- En el resto de casos no debe anticiparse.

ForwardA ← *if* ((Rs1E ≠ 0) & BRwrM & (Rs1E = RdM)) *then* (10) ⟵ *Anticipar desde MEM*
elsif((Rs1E ≠ 0) & BRwrW & (Rs1E = RdW)) *then* (01) ⟵ *Anticipar desde WB*
else (00) ⟵ *No anticipar*

- Ídem para la unidad de anticipación de datos por la entrada B de la ALU
  - Únicamente cambiando RS1E por RS2E.

Procesador segmentado

Diseño de la unidad de anticipación (ii)

ForwardA ← *if* ((Rs1E ≠ 0) & BRwrM & (Rs1E = RdM)) *then* (10)
elsif((Rs1E ≠ 0) & BRwrW & (Rs1E = RdW)) *then* (01)
else (00)

Tabla de verdad

Rs1E ≠ 0	BRwrM	BRwrW	Rs1E = RdM	Rs1E = RdW	ForwardA
0	X	X	X	X	00 (no anticipar)
1	0	1	X	0	00 (no anticipar)
1	0	1	X	1	01 (anticipar WB)
1	1	X	0	X	00 (no anticipar)
1	1	X	1	X	10 (anticipar MEM)

Procesador segmentado

Diseño de la unidad de anticipación (ii)

ForwardA ← *if* ((Rs1E ≠ 0) & BRwrM & (Rs1E = RdM)) *then* (10)
elsif((Rs1E ≠ 0) & BRwrW & (Rs1E = RdW)) *then* (01)
else (00)

ForwardB ← *if* ((Rs2E ≠ 0) & BRwrM & (Rs2E = RdM)) *then* (10)
elsif((Rs2E ≠ 0) & BRwrW & (Rs2E = RdW)) *then* (01)
else (00)

Tabla de verdad

Rs1E ≠ 0	BRwrM	BRwrW	Rs1E = RdM	Rs1E = RdW	ForwardA
0	X	X	X	X	00 (no anticipar)
1	0	1	X	0	00 (no anticipar)
1	0	1	X	1	01 (anticipar WB)
1	1	X	0	X	00 (no anticipar)
1	1	X	1	X	10 (anticipar MEM)

Tabla de verdad

Rs2E ≠ 0	BRwrM	BRwrW	Rs2E = RdM	Rs2E = RdW	ForwardB
0	X	X	X	X	00 (no anticipar)
1	0	1	X	0	00 (no anticipar)
1	0	1	X	1	01 (anticipar WB)
1	1	X	0	X	00 (no anticipar)
1	1	X	1	X	10 (anticipar MEM)

Procesador segmentado

Diseño de la unidad de anticipación (iii)

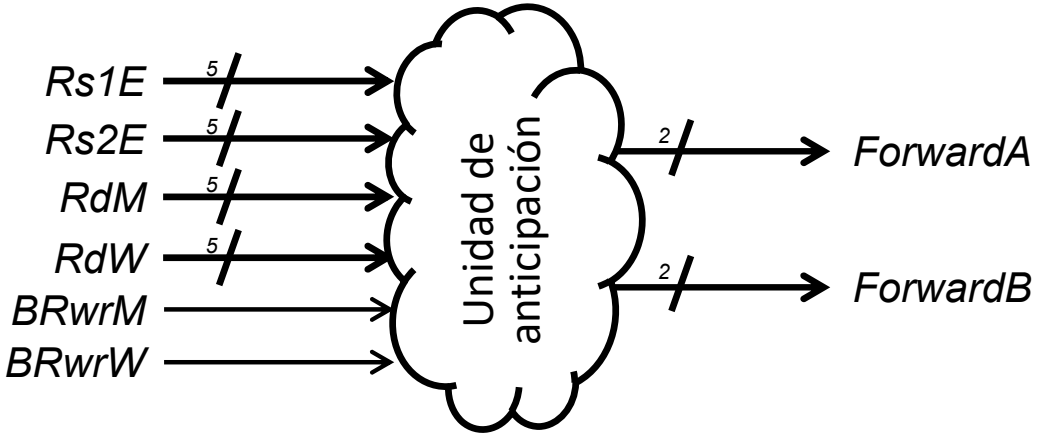


Tabla de verdad

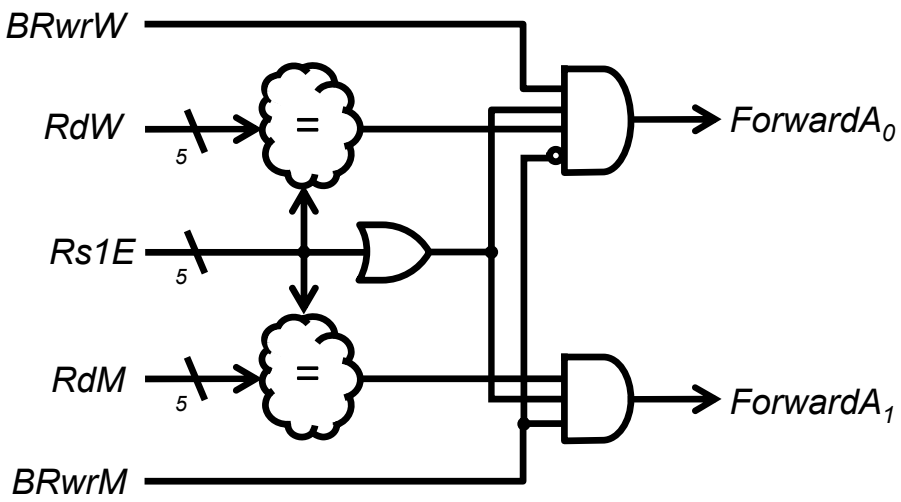
Rs1E ≠ 0	BRwrM	BRwrW	Rs1E = RdM	Rs1E = RdW	ForwardA
0	X	X	X	X	00 (no anticipar)
1	0	1	X	0	00 (no anticipar)
1	0	1	X	1	01 (anticipar WB)
1	1	X	0	X	00 (no anticipar)
1	1	X	1	X	10 (anticipar MEM)

Tabla de verdad

Rs2E ≠ 0	BRwrM	BRwrW	Rs2E = RdM	Rs2E = RdW	ForwardB
0	X	X	X	X	00 (no anticipar)
1	0	1	X	0	00 (no anticipar)
1	0	1	X	1	01 (anticipar WB)
1	1	X	0	X	00 (no anticipar)
1	1	X	1	X	10 (anticipar MEM)

Procesador segmentado

Diseño de la unidad de anticipación (iv)



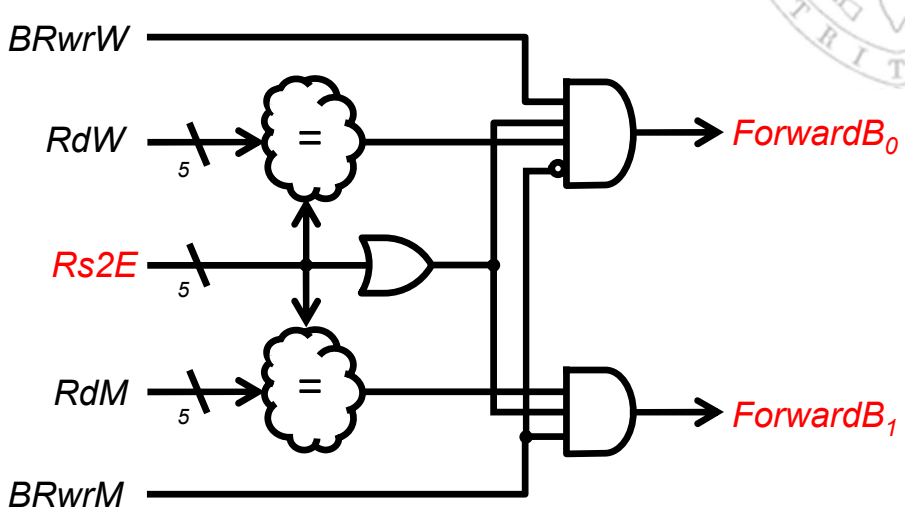


Tabla de verdad

Rs1E ≠ 0	BRwrM	BRwrW	Rs1E = RdM	Rs1E = RdW	ForwardA
0	X	X	X	X	00 (no anticipar)
1	0	1	X	0	00 (no anticipar)
1	0	1	X	1	01 (anticipar WB)
1	1	X	0	X	00 (no anticipar)
1	1	X	1	X	10 (anticipar MEM)

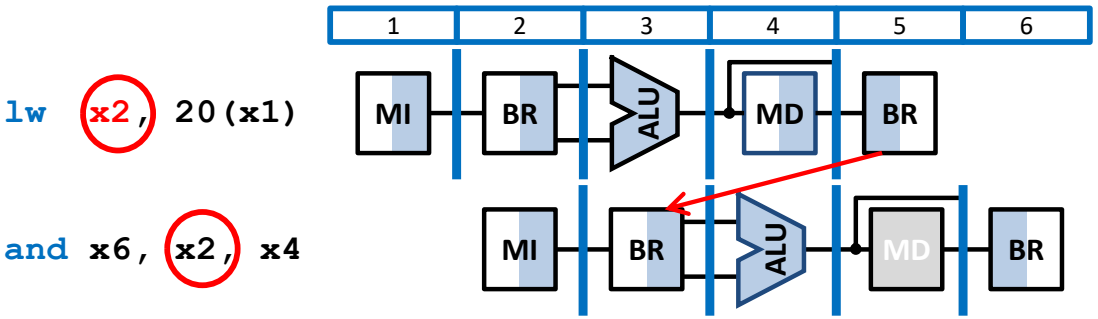
Tabla de verdad

Rs2E ≠ 0	BRwrM	BRwrW	Rs2E = RdM	Rs2E = RdW	ForwardB
0	X	X	X	X	00 (no anticipar)
1	0	1	X	0	00 (no anticipar)
1	0	1	X	1	01 (anticipar WB)
1	1	X	0	X	00 (no anticipar)
1	1	X	1	X	10 (anticipar MEM)

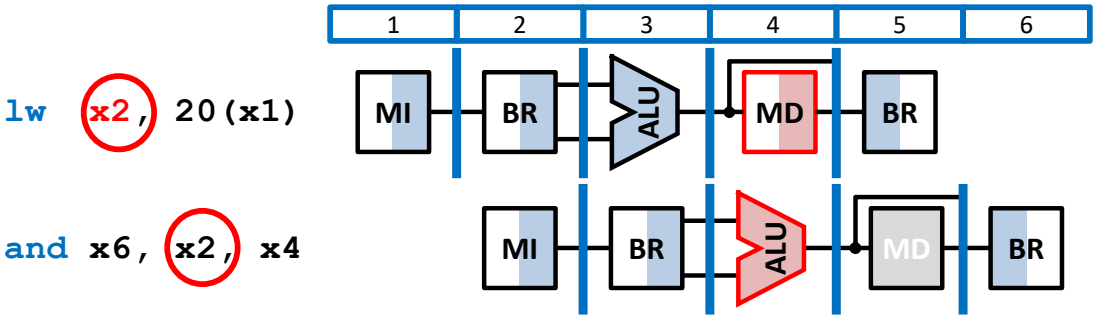
Conflictos de datos

Solución HW: conflicto `lw`

- Existe un conflicto de datos que requiere un tratamiento especial:
  - Cuando a una instrucción **`lw`** que carga un registro le sigue otra instrucción que lee ese mismo registro.



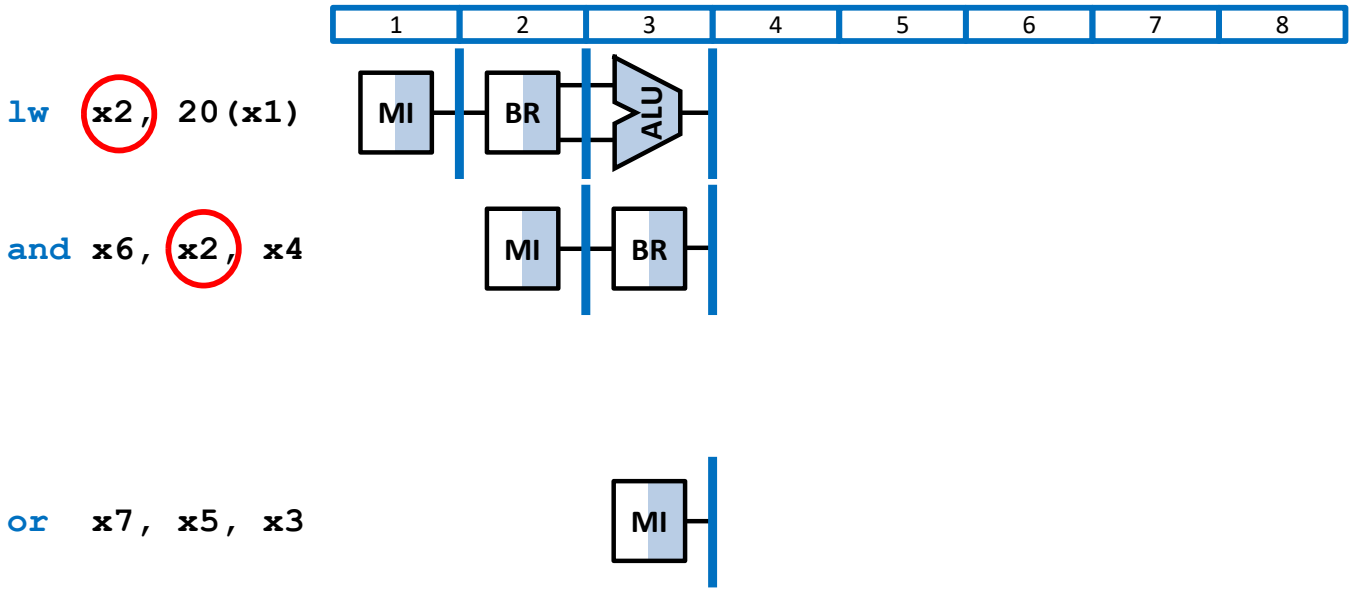
- El dato requerido no puede anticiparse porque:
  - La instrucción **`lw`** lee el dato de memoria en el ciclo 4.
  - La instrucción siguiente necesita el dato en ese mismo ciclo.



Conflictos de datos

Solución HW: conflicto `lw`, parada (i)

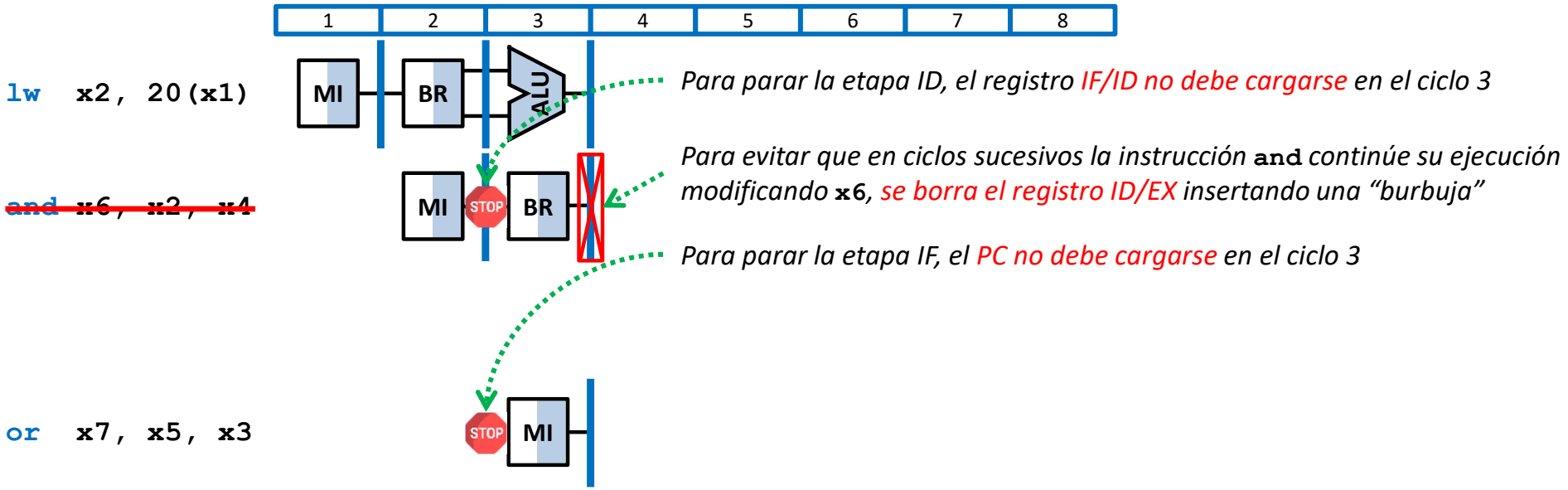
- La solución consiste en parar (*stall*) el pipeline un ciclo para retrasar la instrucción que necesita el dato (y sucesivas) y este pueda anticiparse.
  - Ciclo 3: **`and`** (está en ID) se detecta el conflicto.



Conflictos de datos

Solución HW: conflicto `lw`, parada (i)

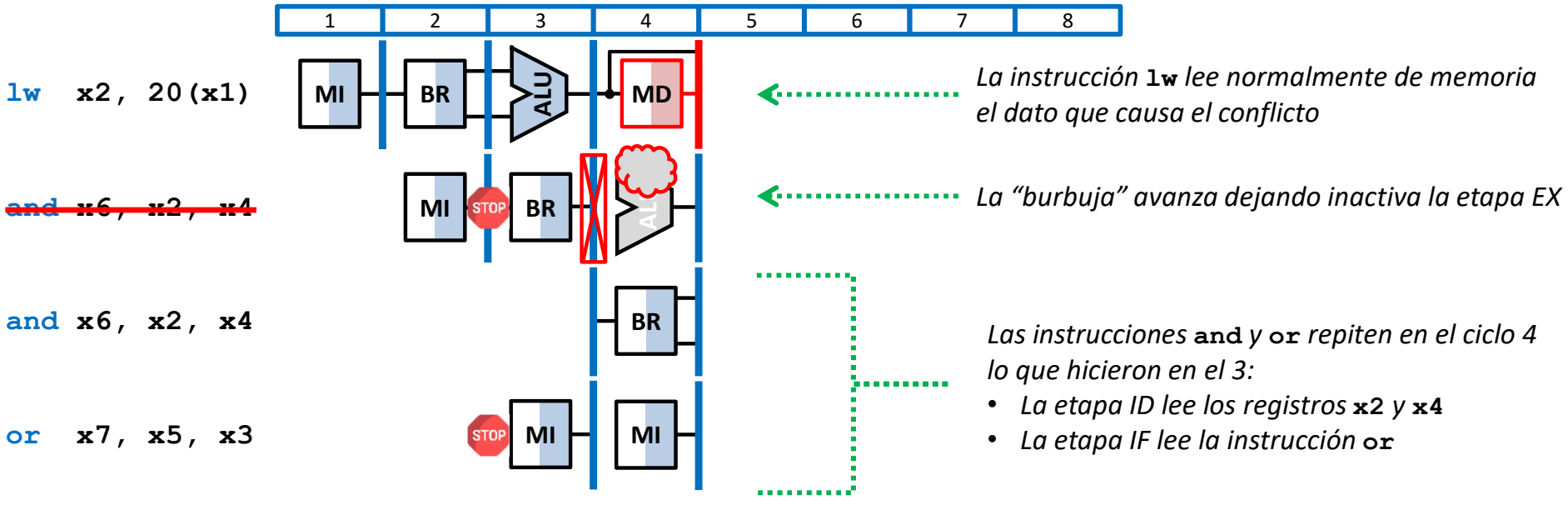
- La solución consiste en parar (*stall*) el pipeline un ciclo para retrasar la instrucción que necesita el dato (y sucesivas) y este pueda anticiparse.
  - Ciclo 3: **`and`** (está en ID) se detecta el conflicto. Las instrucciones **`and`** y **`or`** se paran y se inserta una "burbuja" de no operación en la etapa EX.



Conflictos de datos

Solución HW: conflicto `lw`, parada (i)

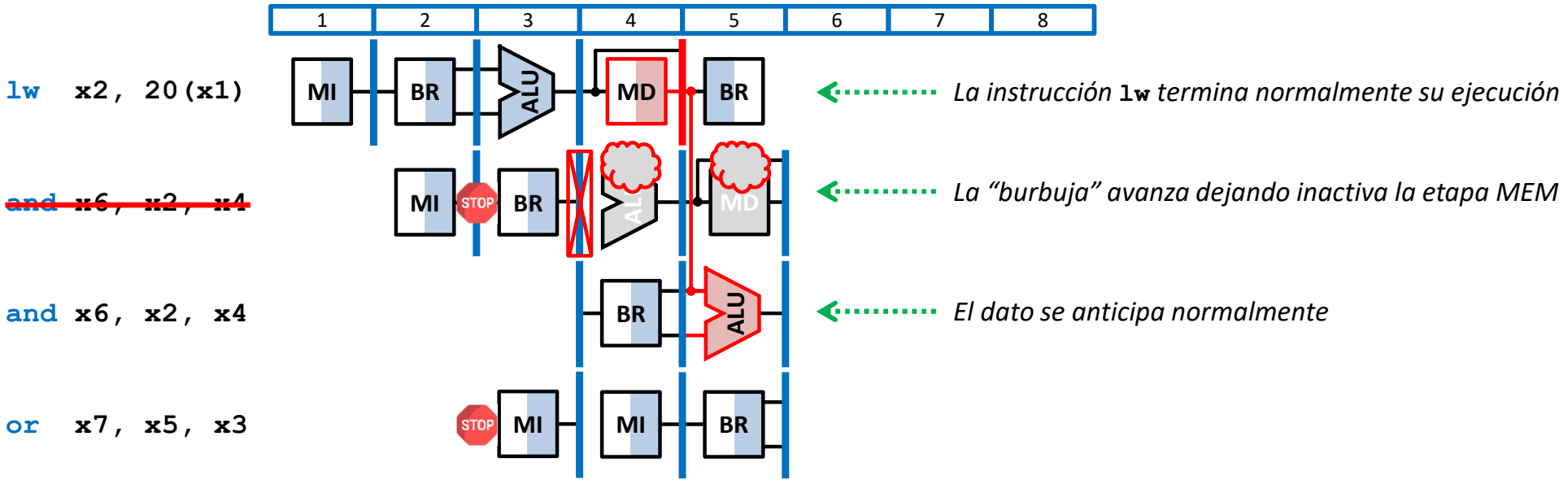
- La solución consiste en parar (*stall*) el pipeline un ciclo para retrasar la instrucción que necesita el dato (y sucesivas) y este pueda anticiparse.
  - Ciclo 3: **`and`** (está en ID) se detecta el conflicto. Las instrucciones **`and`** y **`or`** se paran y se inserta una "burbuja" de no operación en la etapa EX.
  - Ciclo 4: la instrucción **`lw`** lee el dato de memoria, **`and`** y **`or`** se reanudan.



Conflictos de datos

Solución HW: conflicto `lw`, parada (i)

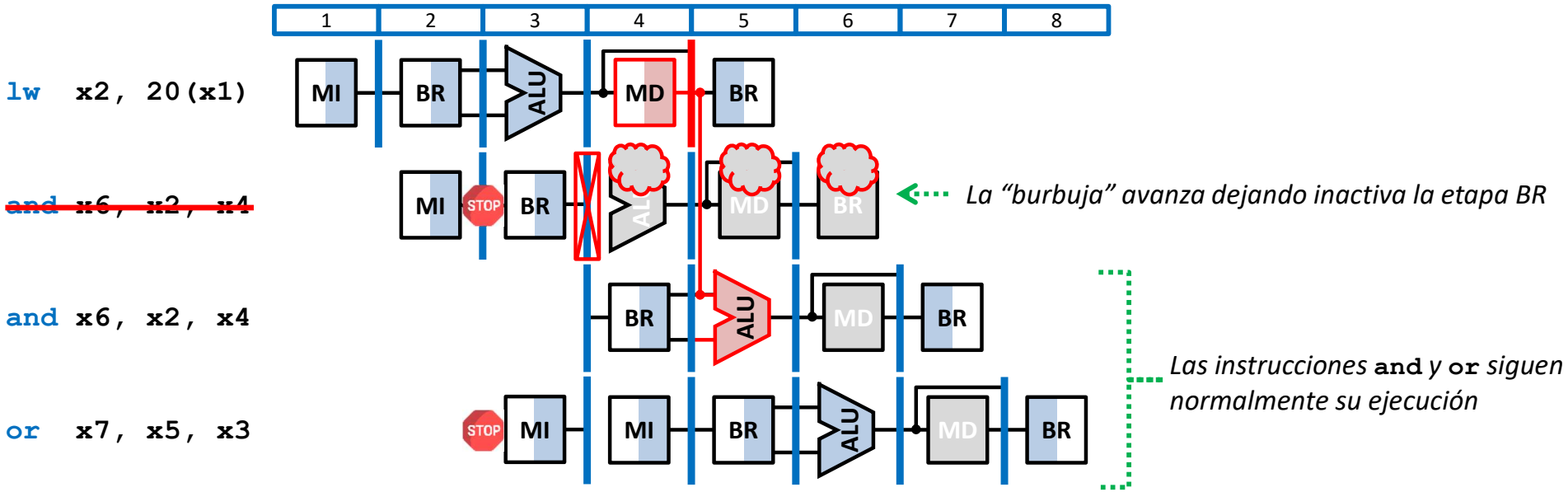
- La solución consiste en parar (*stall*) el pipeline un ciclo para retrasar la instrucción que necesita el dato (y sucesivas) y este pueda anticiparse.
  - Ciclo 3: **`and`** (está en ID) se detecta el conflicto. Las instrucciones **`and`** y **`or`** se paran y se inserta una "burbuja" de no operación en la etapa EX.
  - Ciclo 4: la instrucción **`lw`** lee el dato de memoria, **`and`** y **`or`** se reanudan.
  - Ciclo 5: el dato se anticipa desde la etapa WB de **`lw`** a la etapa EX de **`and`**.



Conflictos de datos

Solución HW: conflicto `lw`, parada (i)

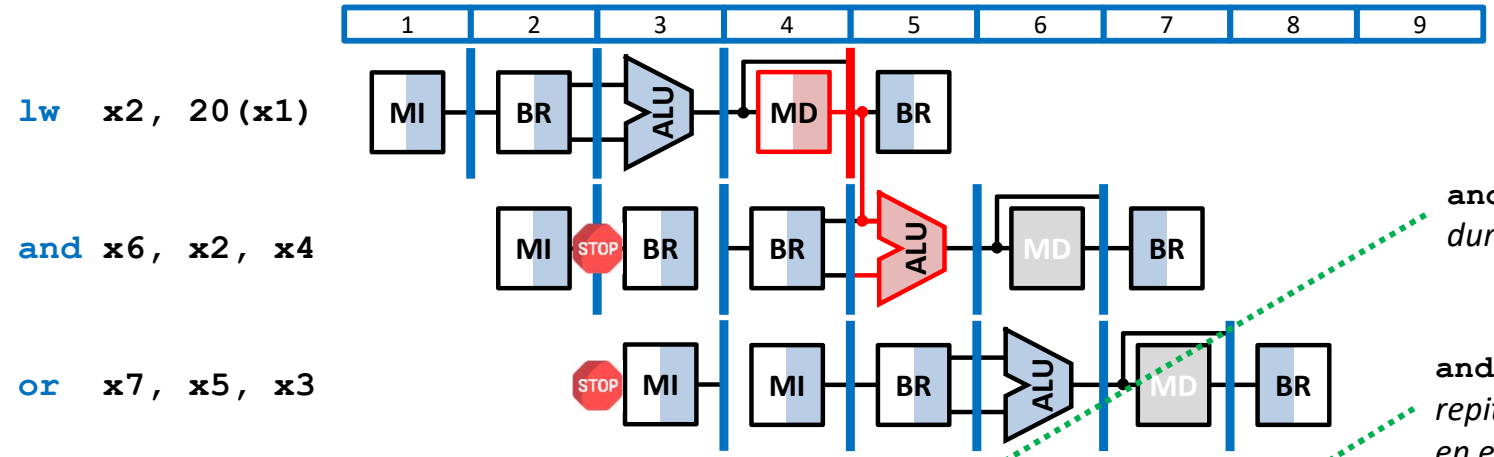
- La solución consiste en parar (*stall*) el pipeline un ciclo para retrasar la instrucción que necesita el dato (y sucesivas) y este pueda anticiparse.
  - Ciclo 3: **`and`** (está en ID) se detecta el conflicto. Las instrucciones **`and`** y **`or`** se paran y se inserta una "burbuja" de no operación en la etapa EX.
  - Ciclo 4: la instrucción **`lw`** lee el dato de memoria, **`and`** y **`or`** se reanudan.
  - Ciclo 5: el dato se anticipa desde la etapa WB de **`lw`** a la etapa EX de **`and`**.
  - Ciclos sucesivos: el pipeline avanza normalmente.
  - Penaliza la ejecución del programa en un ciclo por conflicto **`lw`**.



Conflictos de datos

Solución HW: conflicto `lw`, parada (ii)

- En diagramas de ejecución simplificados, las paradas se indican marcando las etapas que paran (las "burbujas" quedan implícitas):



`and` y `or` se paran durante un ciclo

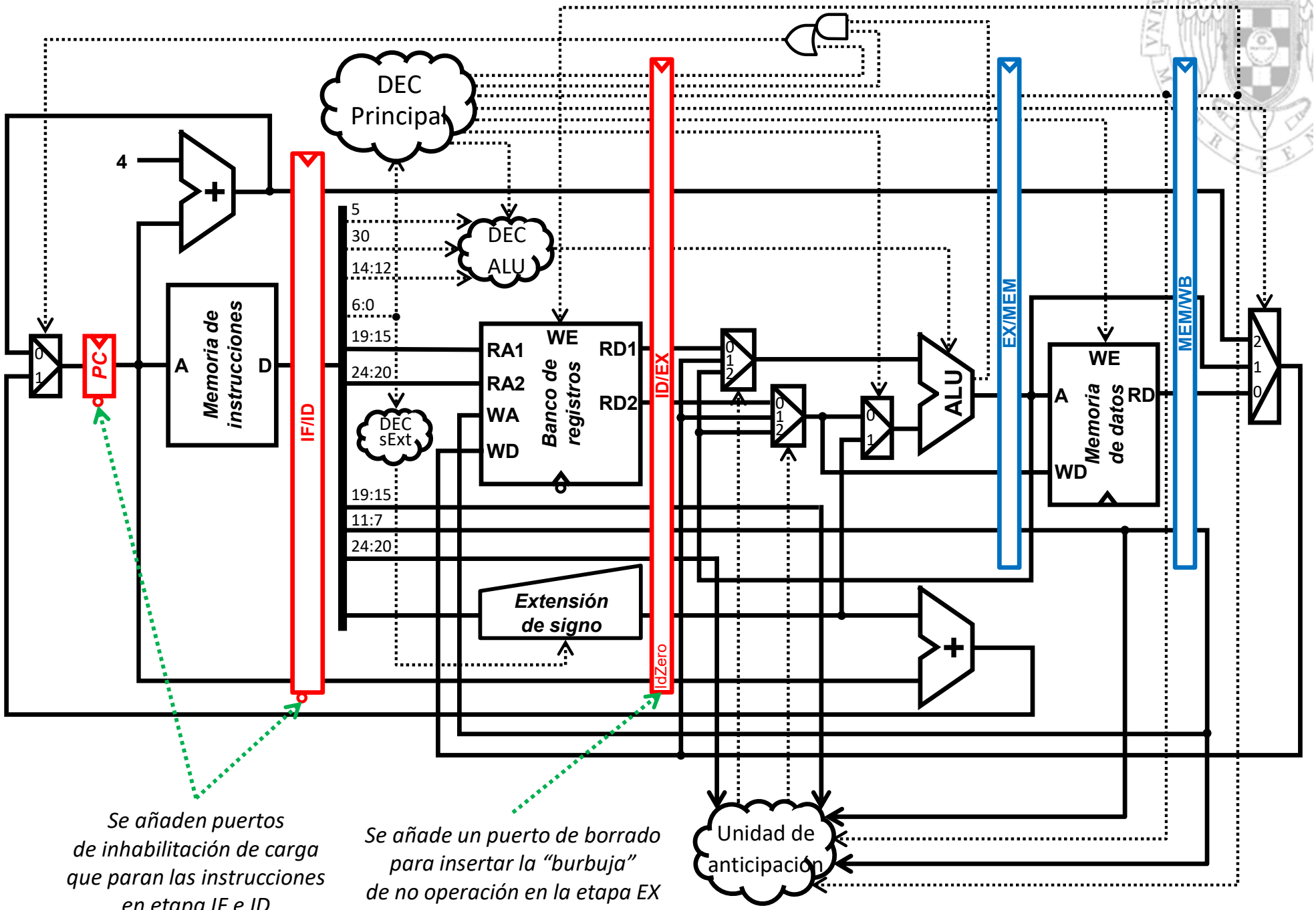
`and` y `or` se reanudan repitiendo lo que hicieron en el ciclo anterior

	1	2	3	4	5	6	7	8	9
`lw x2, 20(x1)`	IF	ID	EX	M	WB x2				
`and x6, x2, x4`		IF	IDp	ID	EX	M	WB		
`or x7, x5, x3`			IFp	IF	ID	EX	M	WB	
`add x8, x2, x2`					IF	ID	EX	M	WB

La siguiente instrucción se lanza tras reanudarse `and` y `or`

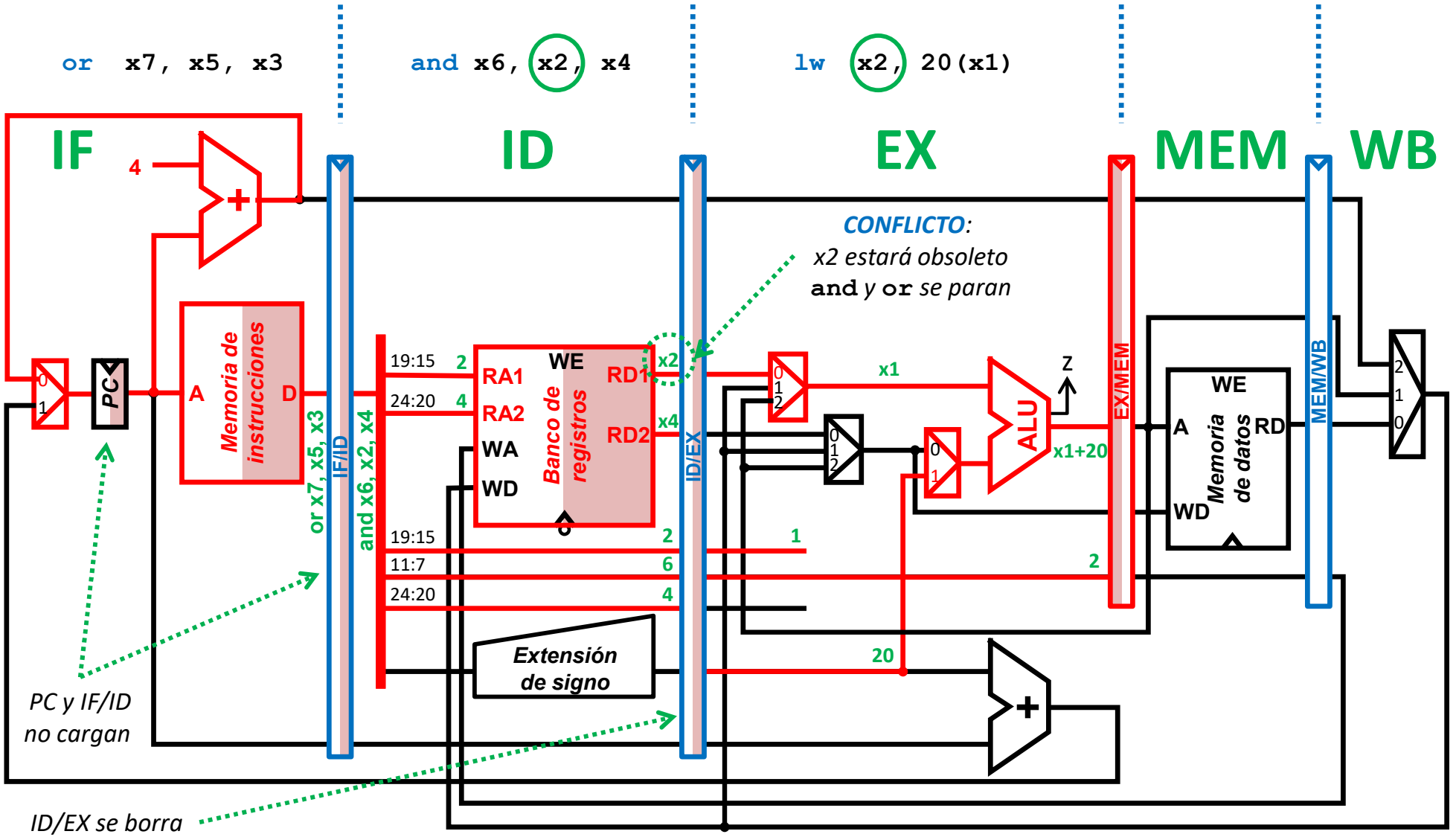
Procesador segmentado

+ conflicto `lw`: ruta de datos



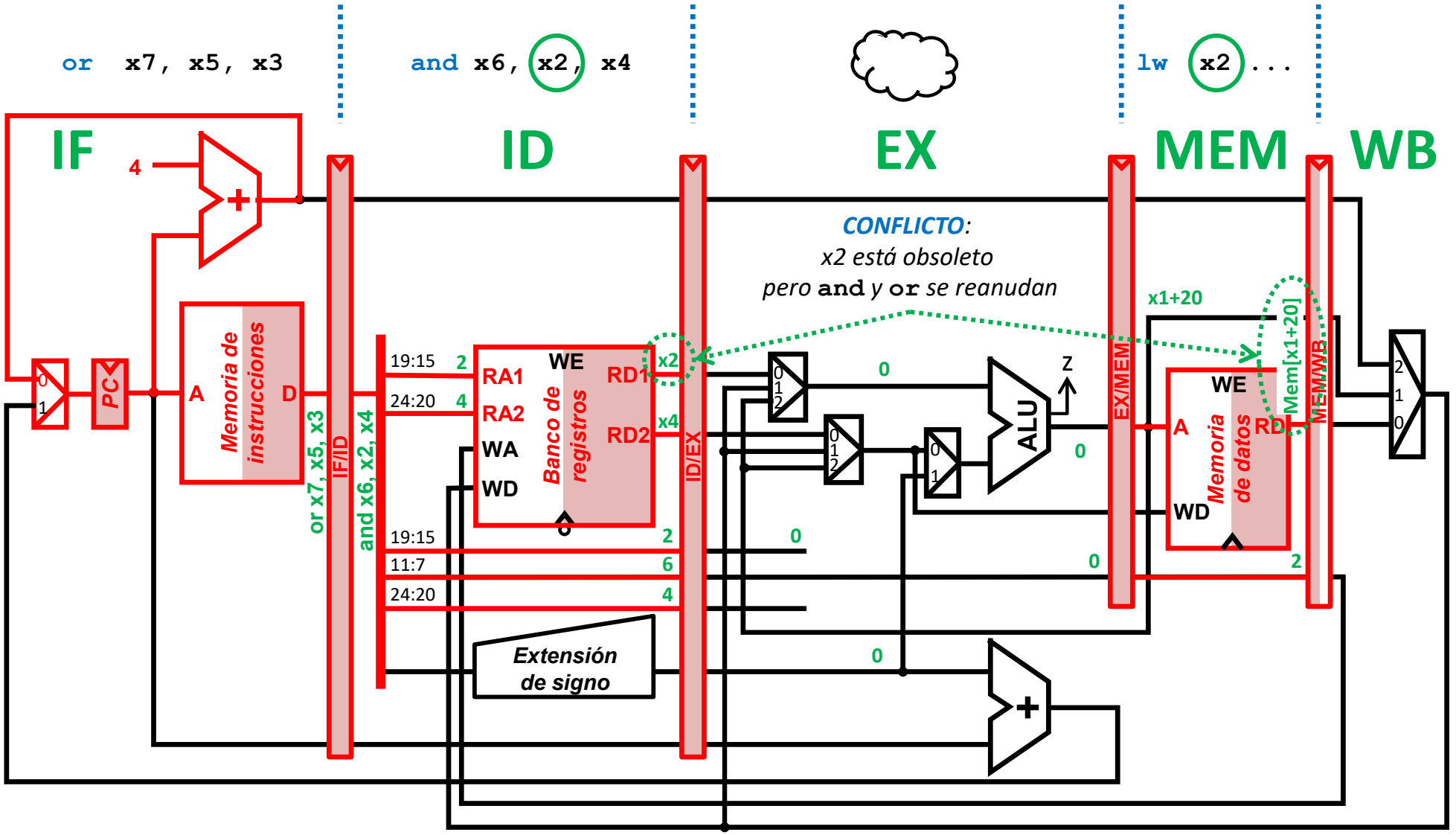
Procesador segmentado

Simulación de parada: 3er. ciclo



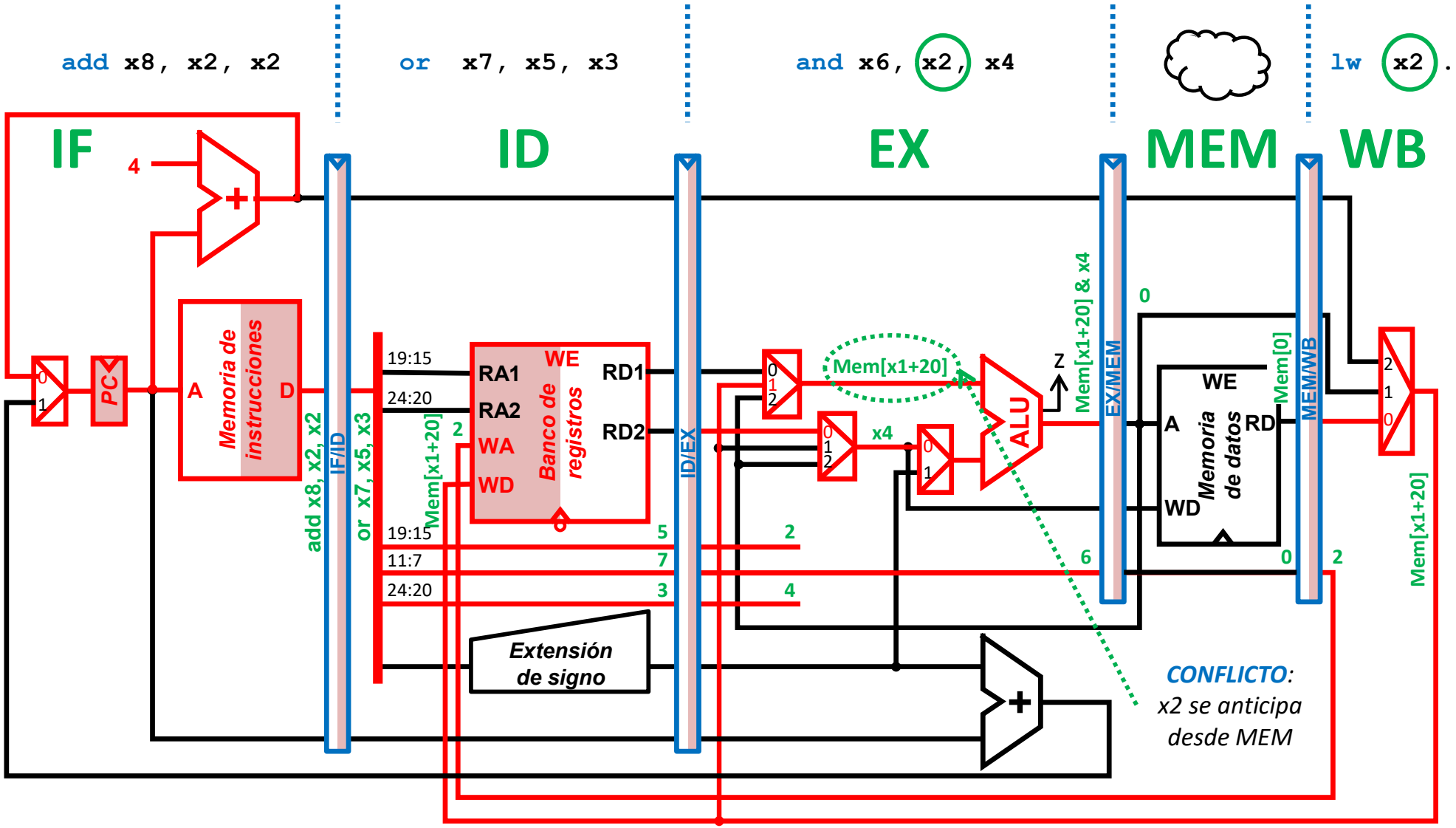
Procesador segmentado

Simulación de parada: 4o. ciclo



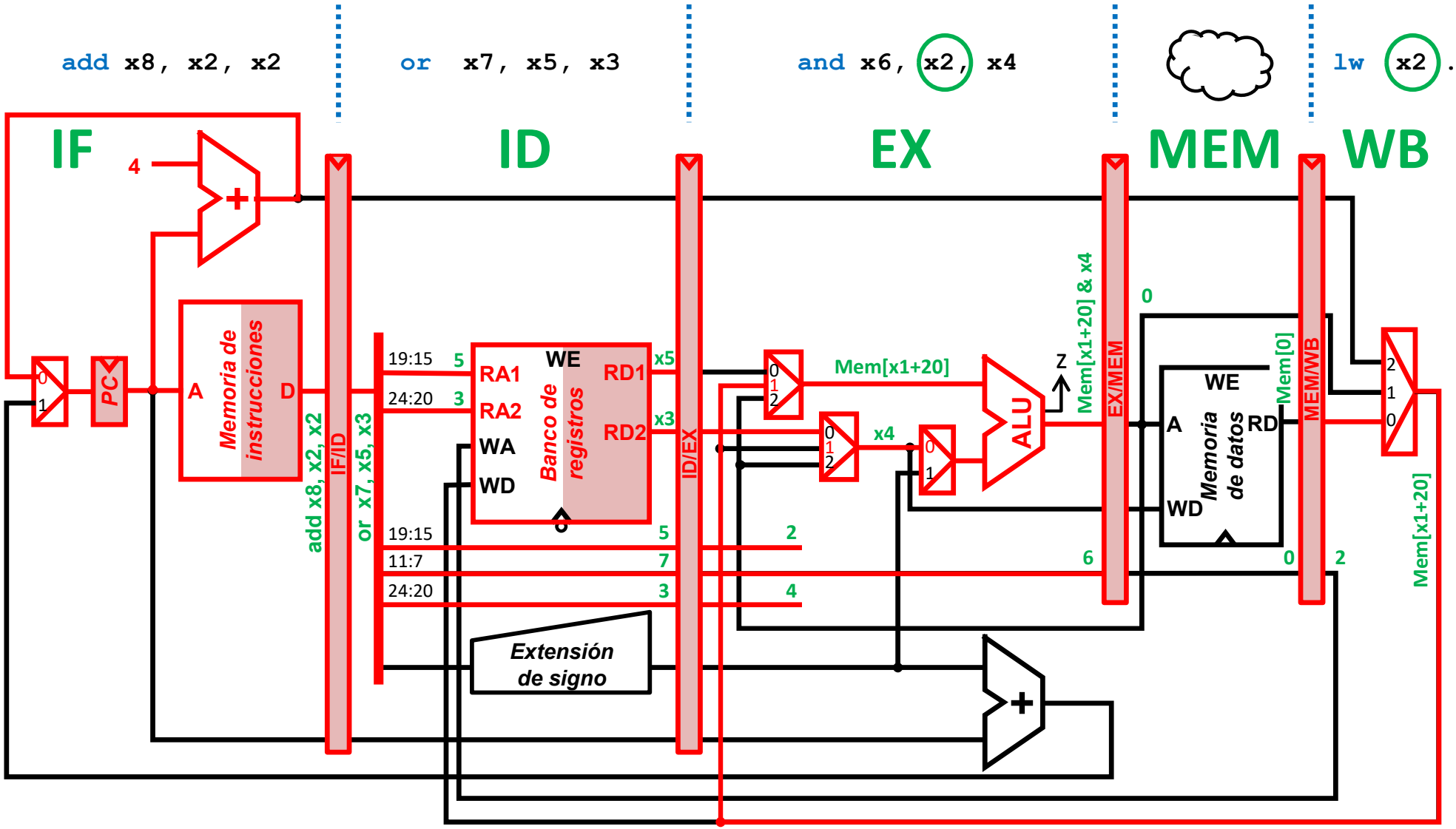
Procesador segmentado

Simulación de parada: 5o. ciclo (1ª mitad)



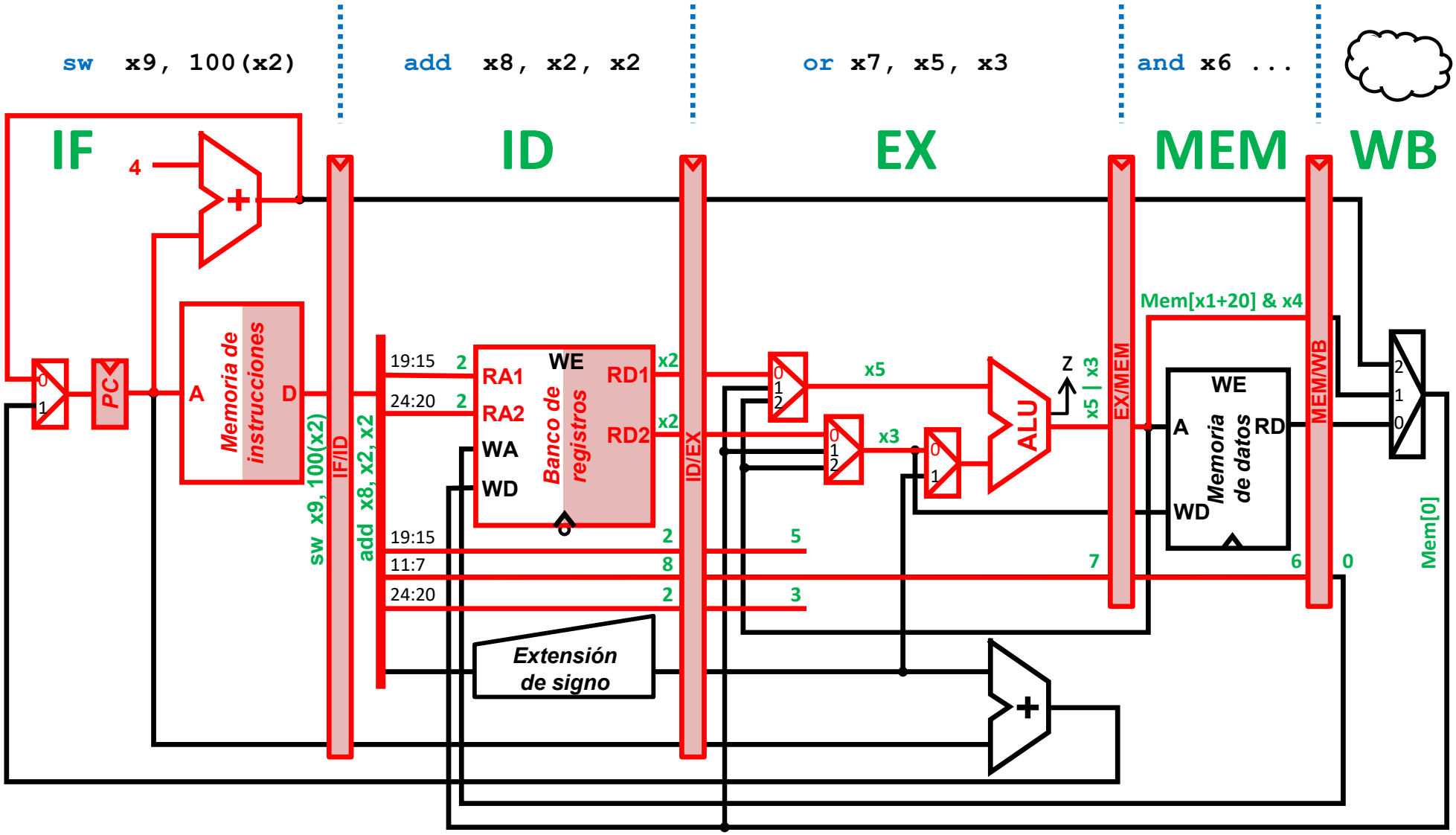
Procesador segmentado

Simulación de parada: 5o. ciclo (2ª mitad)



Procesador segmentado

Simulación de parada: 6o. ciclo



Procesador segmentado

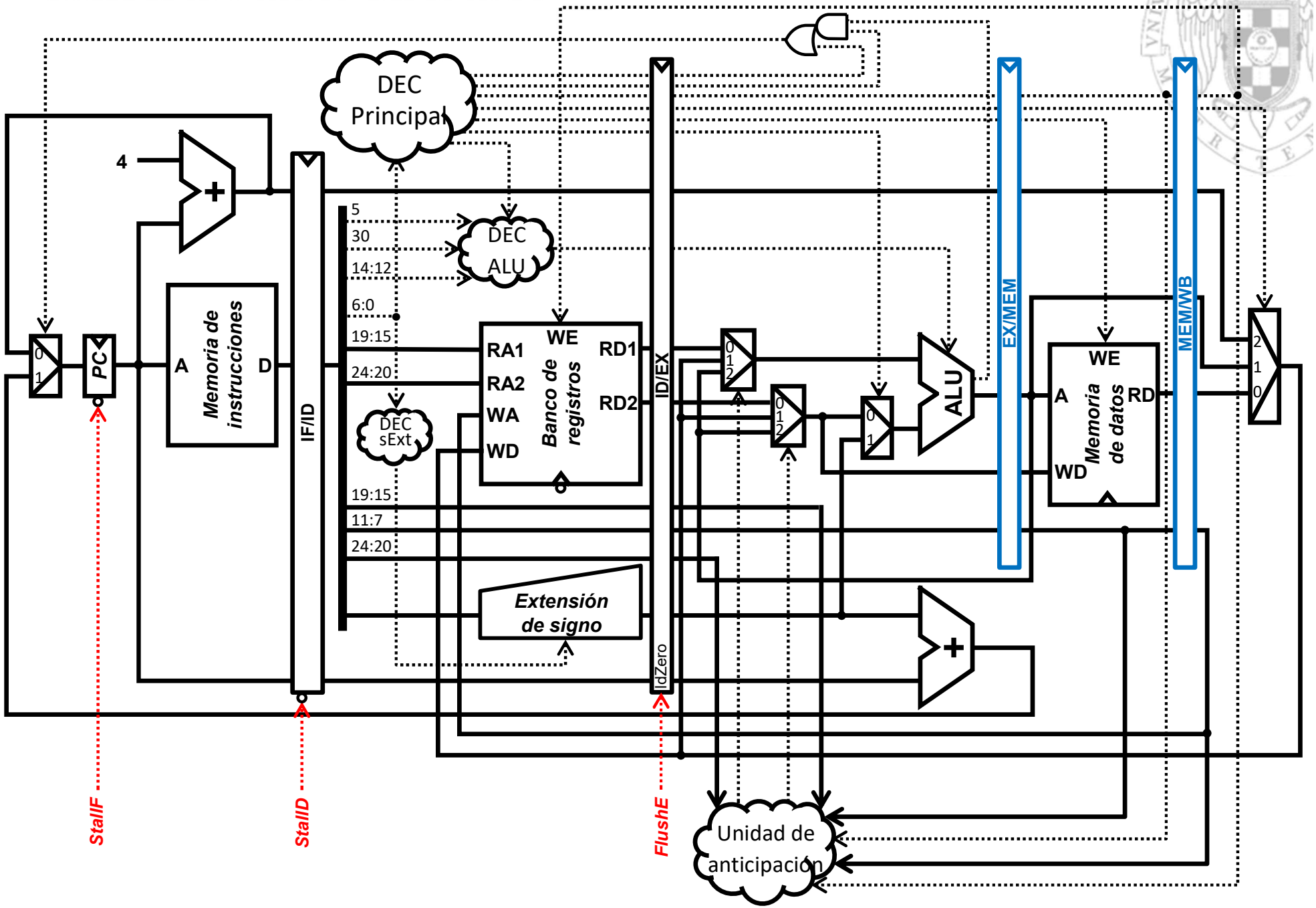
Unidad de conflictos

- La unidad de conflictos es un circuito combinacional que en todo ciclo determina si las etapas IF e ID del pipeline deben pararse, controlando si:
  - El PC y el registro de segmentación IF/ID deben cargar o no.
  - Debe borrarse el registro de segmentación ID/EX.

- Para realizar su función necesita conocer:
  - Si en la etapa EX hay una instrucción **`lw`**.
    - Comprobando que que ResSrcE = 0 y BRwrE = 1 (solo **`lw`** satisface esta condición)
  - RdE: número del registro destino de la instrucción **`lw`** en la etapa EX.
  - Rs1D: número del registro fuente 1 de la instrucción en la etapa ID.
  - Rs2D: número del registro fuente 2 de la instrucción en la etapa ID.

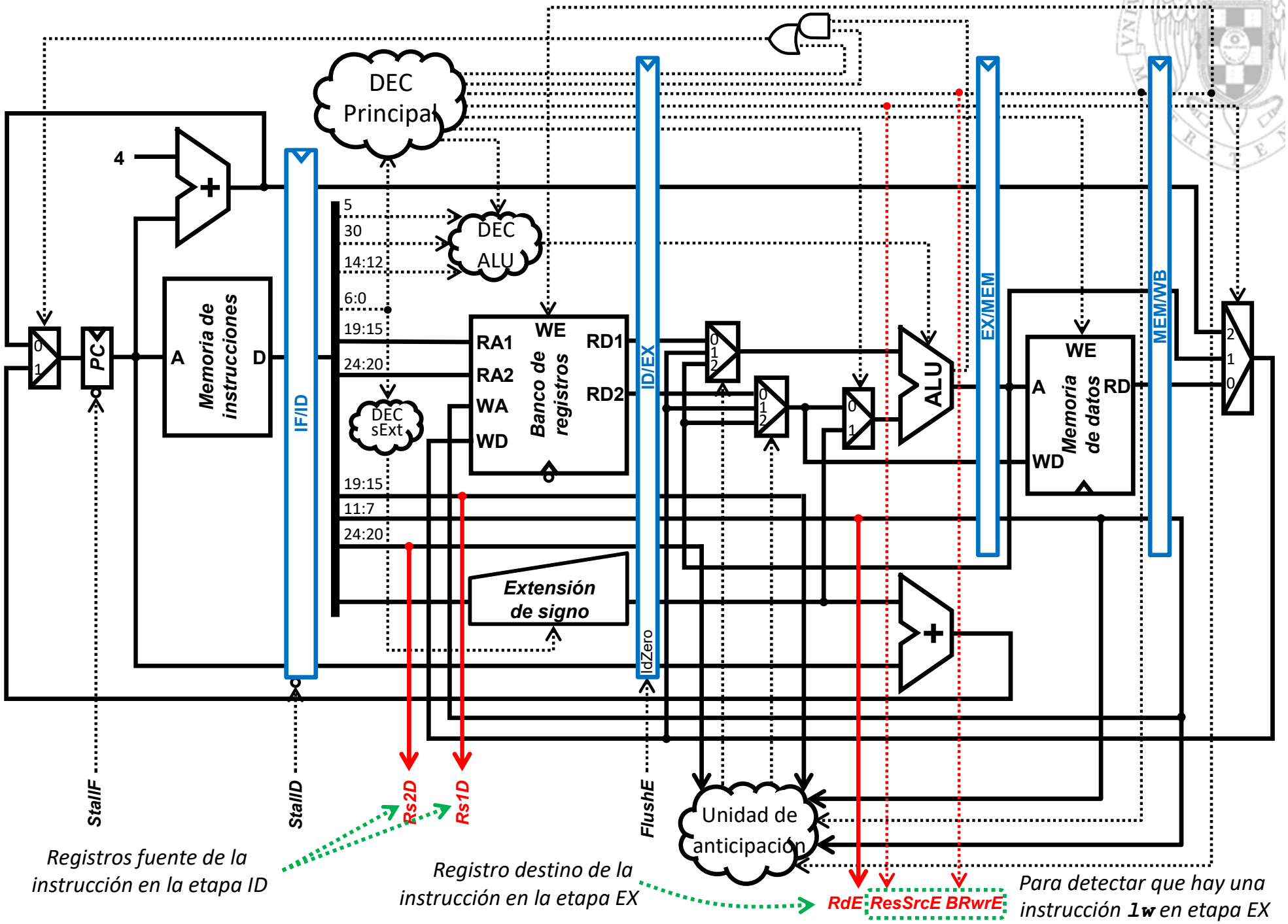
Procesador segmentado

+ Unidad de conflictos: señales de control



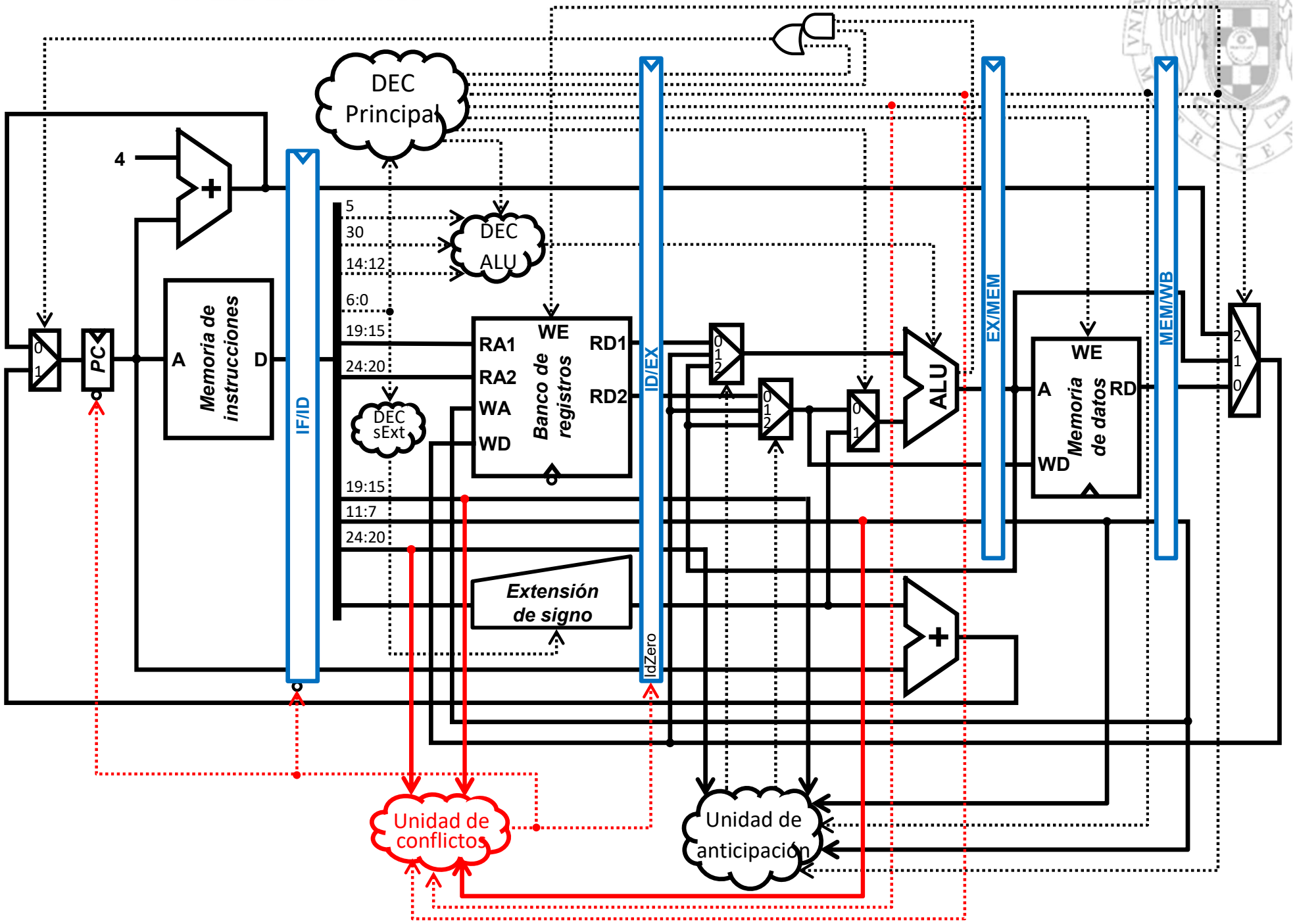
Procesador segmentado

+ Unidad de conflictos: señales de estado



Procesador segmentado

+ Unidad de conflictos



Procesador segmentado

Diseño de la unidad de conflictos (i)

- El pipeline debe pararse durante un ciclo por un conflicto **`lw`** si:
  - Hay una instrucción **`lw`** en la etapa EX
    - Comprobando que ResSrcE = 0 y BRwrE = 1 (solo **`lw`** satisface esta combinación)

Stall ← *if* ((ResSrcE = 0) & BRwrE) *then* (1) ⇠ *Parar el pipeline*

Procesador segmentado

Diseño de la unidad de conflictos (i)

- El pipeline debe pararse durante un ciclo por un conflicto **`lw`** si:
  - Hay una instrucción **`lw`** en la etapa EX
    - Comprobando que ResSrcE = 0 y BRwrE = 1 (solo **`lw`** satisface esta combinación)
  - El registro destino en la etapa EX (RdE) coincide con uno de los registros fuente de la etapa ID (Rs1D y/o Rs2D).

Stall ← *if* ((ResSrcE = 0) & BRwrE & ((Rs1D = RdE) | (Rs2D = RdE))) *then* (1) ⇠ *Parar el pipeline*

Procesador segmentado

Diseño de la unidad de conflictos (i)

- El pipeline debe pararse durante un ciclo por un conflicto **`lw`** si:
  - Hay una instrucción **`lw`** en la etapa EX
    - Comprobando que ResSrcE = 0 y BRwrE = 1 (solo **`lw`** satisface esta combinación)
  - El registro destino en la etapa EX (RdE) coincide con uno de los registros fuente de la etapa ID (Rs1D y/o Rs2D).
- En el resto de casos no debe pararse.

Stall ← *if* ((ResSrcE = 0) & BRwrE & ((Rs1D = RdE) | (Rs2D = RdE))) *then* (1) ⇠ *Parar el pipeline*
else (0) ⇠ *No parar el pipeline*

Procesador segmentado

Diseño de la unidad de conflictos (i)

- El pipeline debe pararse durante un ciclo por un conflicto `lw` si:
  - Hay una instrucción **`lw`** en la etapa EX
    - Comprobando que ResSrcE = 0 y BRwrE = 1 (solo **`lw`** satisface esta combinación)
  - El registro destino en la etapa EX (RdE) coincide con uno de los registros fuente de la etapa ID (Rs1D y/o Rs2D).
- En el resto de casos no debe pararse.
- En caso de parada hay que:
  - Inhabilitar la carga del PC y del registro de segmentación ID/IF.

Stall ← *if* ((ResSrcE = 0) & BRwrE & ((Rs1D = RdE) | (Rs2D = RdE))) *then* (1) ⇠ *Parar el pipeline*
else (0) ⇠ *No parar el pipeline*
StallF ← Stall
StallD ← Stall

Procesador segmentado

Diseño de la unidad de conflictos (i)

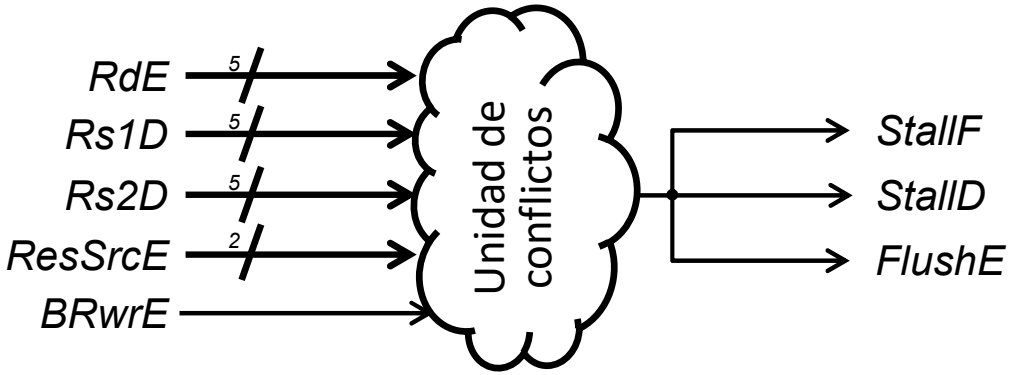
- El pipeline debe pararse durante un ciclo por un conflicto `lw` si:
  - Hay una instrucción **`lw`** en la etapa EX
    - Comprobando que ResSrcE = 0 y BRwrE = 1 (solo **`lw`** satisface esta combinación)
  - El registro destino en la etapa EX (RdE) coincide con uno de los registros fuente de la etapa ID (Rs1D y/o Rs2D).
- En el resto de casos no debe pararse.
- En caso de parada hay que:
  - Inhabilitar la carga del PC y del registro de segmentación ID/IF.
  - Borrar el registro de segmentación ID/EX.

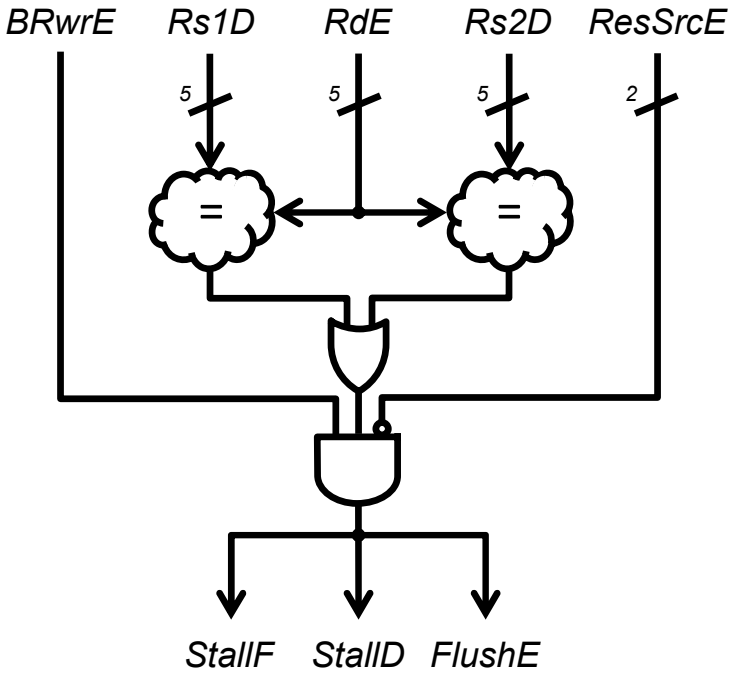
Stall ← *if* ((ResSrcE = 0) & BRwrE & ((Rs1D = RdE) | (Rs2D = RdE))) *then* (1) ⟵ *Parar el pipeline*
else (0) ⟵ *No parar el pipeline*
StallF ← Stall
StallD ← Stall
FlushE ← Stall

Procesador segmentado

Diseño de la unidad de conflictos (ii)

Stall ← *if* ((ResSrcE = 0) & BRwrE & ((Rs1D = RdE) | (Rs2D = RdE))) *then* (1)
else (0)
StallF ← Stall
StallD ← Stall
FlushE ← Stall





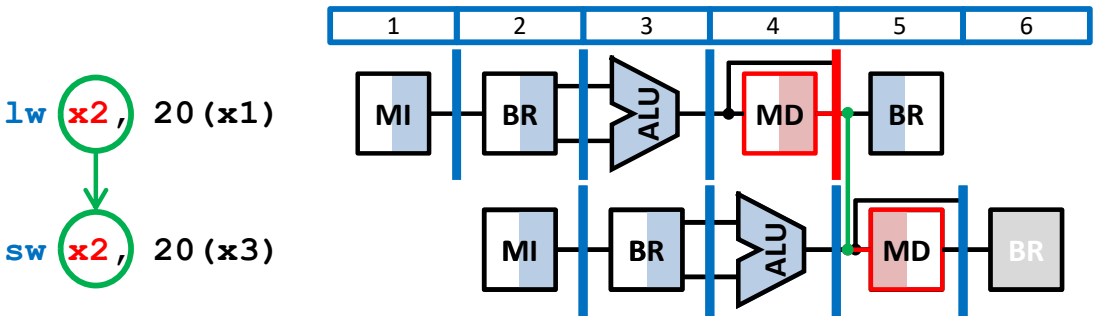
Conflictos de datos

Solución HW: optimizaciones adicionales (i)

- La solución presentada realiza en algunos casos paradas innecesarias.

- Cuando `x0` es el registro destino de la carga de memoria.
  - Al carecer de sentido instrucciones como `lw x0,20(x1)` no merece la pena añadir lógica a la unidad de conflictos para tratar estos casos.

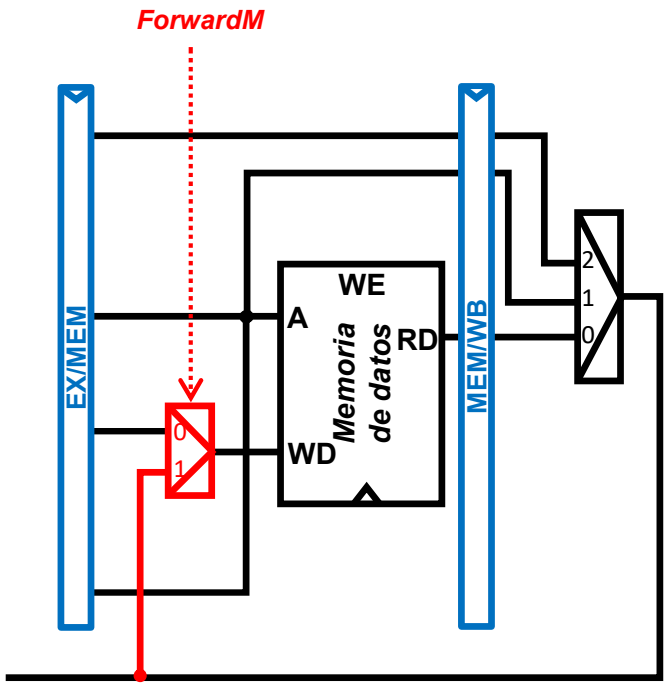
- Cuando a la instrucción `lw` le sigue una instrucción `sw` que almacena en memoria el mismo registro cargado por la primera.
  - Es un caso es más común porque, por ejemplo, se usa para copiar de arrays.
  - El dato, disponible desde el ciclo 5, podría anticiparse sin necesidad de parar.



Conflictos de datos

Solución HW: optimizaciones adicionales (ii)

- Para evitar la penalización en el caso **`lw`** → **`sw`** bastaría con:
  - Añadir un MUX en la entrada de datos de la memoria que permita anticipar el dato desde la etapa WB a la etapa MEM.
  - Rediseñar las unidades de anticipación y de conflictos.



Debe anticipar si: hay una instrucción **`lw`** *en la etapa WB, una* **`sw`** *en MEM y el registro destino de la primera coincide con el fuente de la segunda*

ForwardM ← *if* ((ResSrcW = 0) & BRwrW
& MemWrM
& (RdW = Rs2M)) *then* (1)
else (0)

Stall ← *if* ((ResSrcE = 0) & BRwrE
& ((Rs1D = RdE) | (Rs2D = RdE))
& !MemWrD) *then* (1)
else (0)

No debe parar si hay una instrucción **`sw`** *en la etapa ID*

Conflictos de datos

Solución HW+SW: reordenación de código

- Dado un programa en ensamblador, las paradas por conflicto de datos con instrucciones **`lw`** son inevitables por HW.
  - Pero pueden evitarse por SW reordenando el código para que a una instrucción **`lw`** nunca le siga otra que utilice el registro que carga.
  - Esta es una de las optimizaciones que aplican los compiladores.

```
...
int a, b, c;
int d, e;
...
c = a + b;
e = d + b;
...
```

a → *x1* *d* → *x4*
b → *x2* *e* → *x5*
c → *x3*

Asignación de variables a registros

compilación directa

```
...
lw   x1, 0(x31)
lw   x2, 4(x31)
add  x3, x1, x2
sw   x3, 8(x31)
lw   x4, 12(x31)
add  x5, x4, x2
sw   x5, 16(x31)
...
```

2 paradas

compilación optimizada

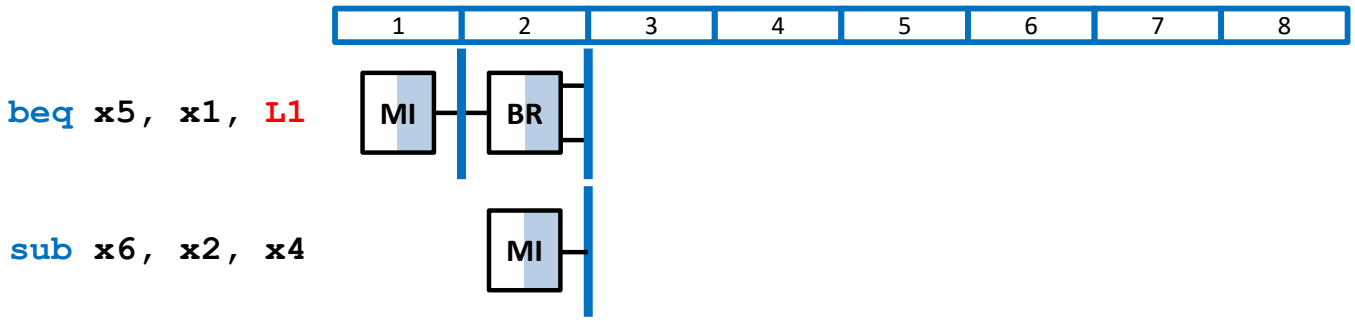
```
...
lw   x1, 0(x31)
lw   x2, 4(x31)
lw   x4, 12(x31)
add  x3, x1, x2
sw   x3, 8(x31)
add  x5, x4, x2
sw   x5, 16(x31)
...
```

0 paradas

Conflictos de control

Solución HW: parada

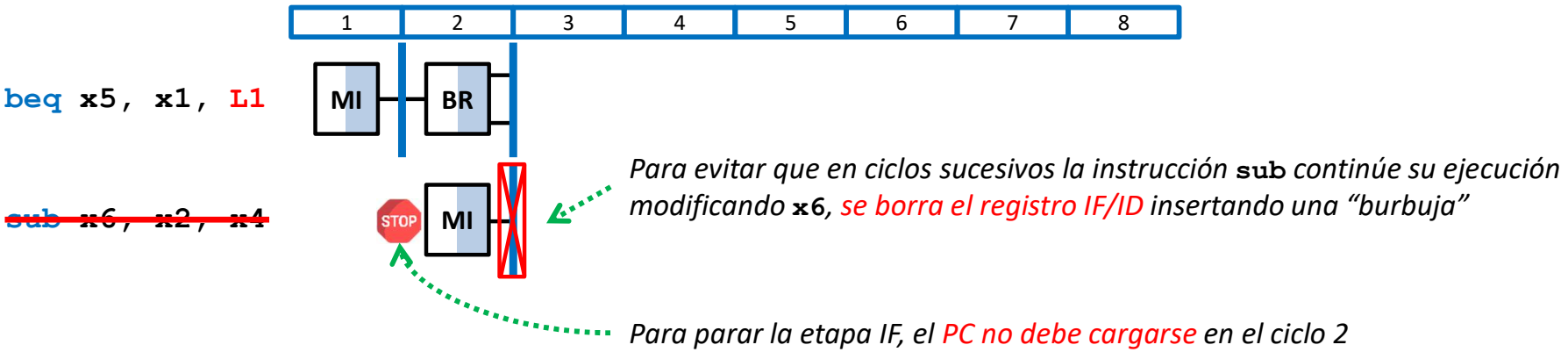
- Una solución consiste en parar el pipeline durante 2 ciclos para retrasar la ejecución de nuevas instrucciones hasta que el salto esté resuelto.
  - Ciclo 2: **`beq`** (está en ID) se detecta el conflicto.



Conflictos de control

Solución HW: parada

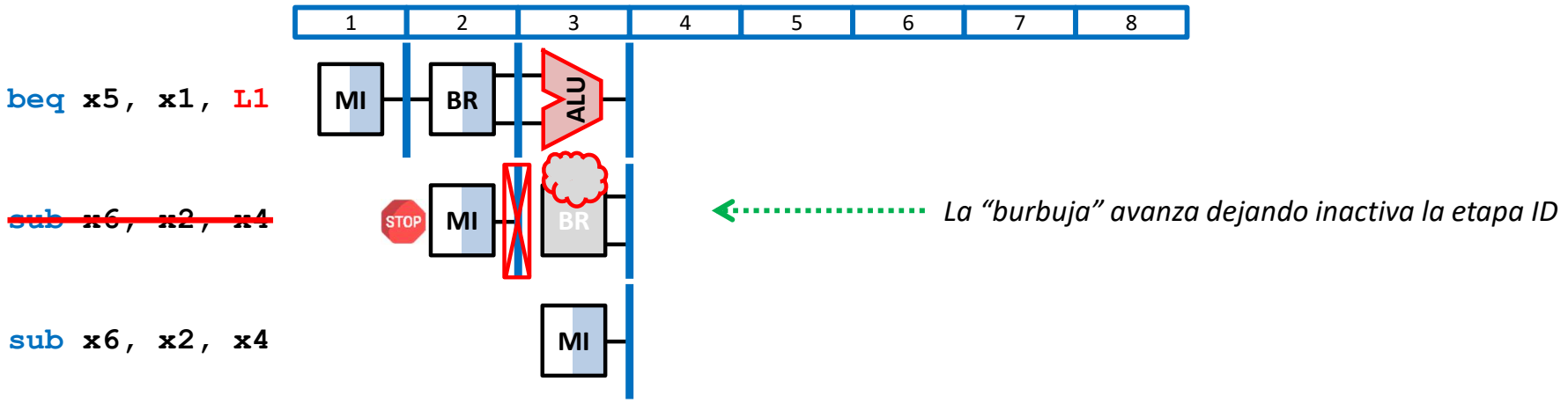
- Una solución consiste en parar el pipeline durante 2 ciclos para retrasar la ejecución de nuevas instrucciones hasta que el salto esté resuelto.
  - Ciclo 2: **`beq`** (está en ID) se detecta el conflicto. La instrucción **`sub`** se para y se inserta una "burbuja" de no operación en la etapa ID.



Conflictos de control

Solución HW: parada

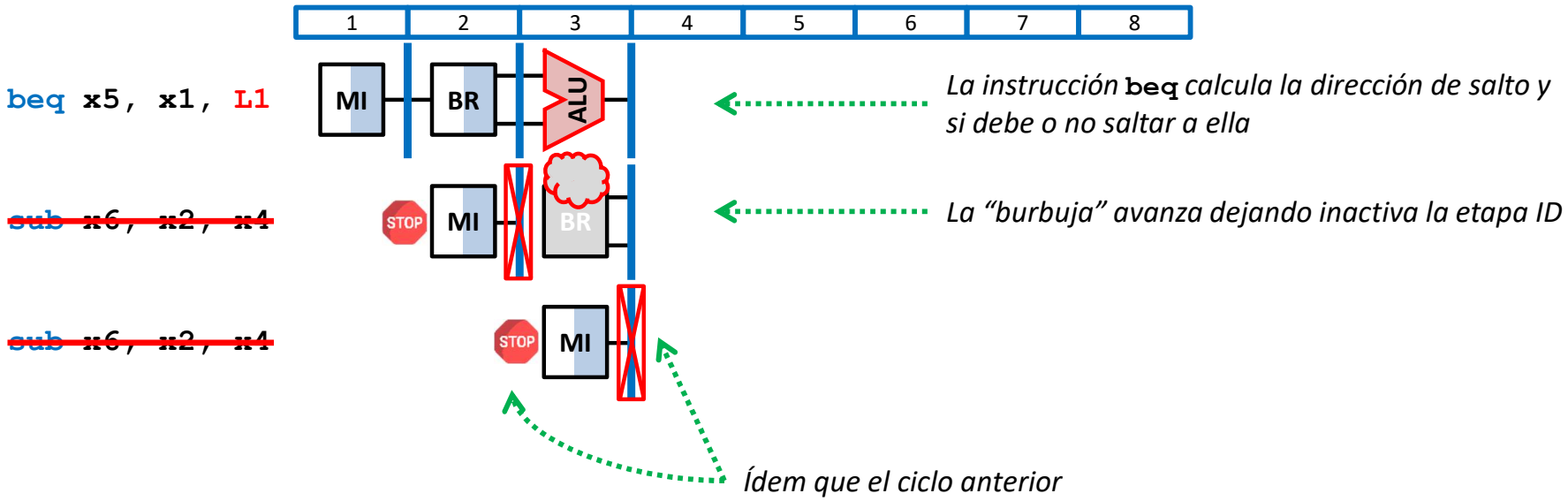
- Una solución consiste en parar el pipeline durante 2 ciclos para retrasar la ejecución de nuevas instrucciones hasta que el salto esté resuelto.
  - Ciclo 2: **`beq`** (está en ID) se detecta el conflicto. La instrucción **`sub`** se para y se inserta una "burbuja" de no operación en la etapa ID.
  - Ciclo 3: **`beq`** (está en EX) resuelve el salto, pero el conflicto continua.



Conflictos de control

Solución HW: parada

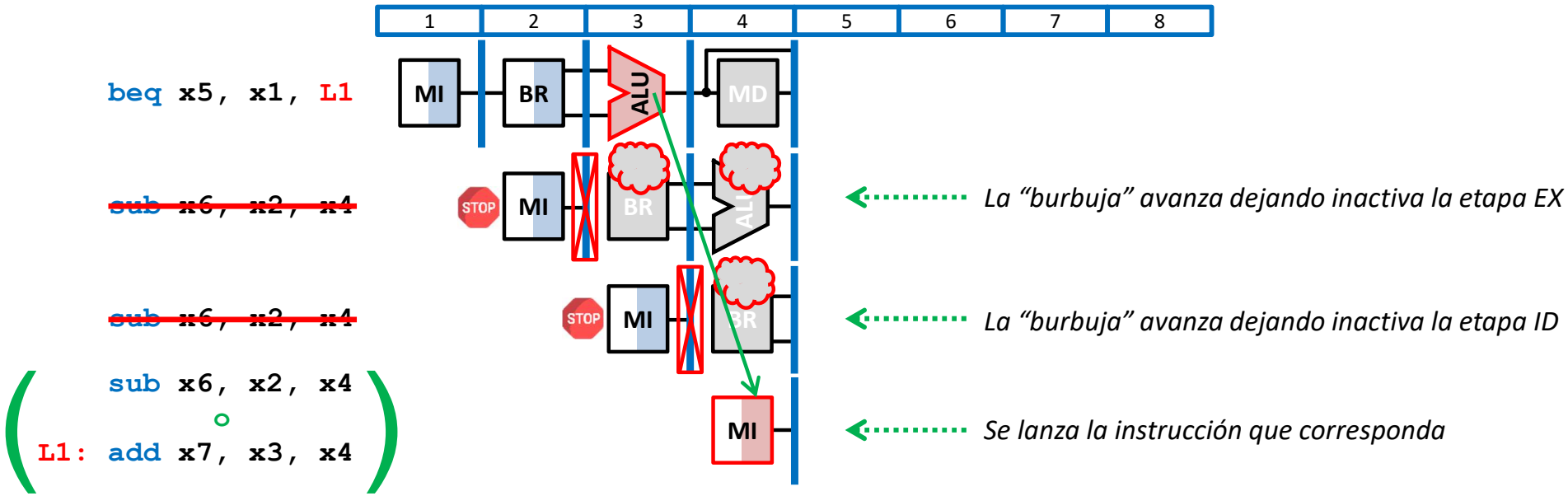
- Una solución consiste en parar el pipeline durante 2 ciclos para retrasar la ejecución de nuevas instrucciones hasta que el salto esté resuelto.
  - Ciclo 2: **`beq`** (está en ID) se detecta el conflicto. La instrucción **`sub`** se para y se inserta una "burbuja" de no operación en la etapa ID.
  - Ciclo 3: **`beq`** (está en EX) resuelve el salto, pero el conflicto continua. La instrucción **`sub`** permanece parada y se inserta otra "burbuja".



Conflictos de control

Solución HW: parada

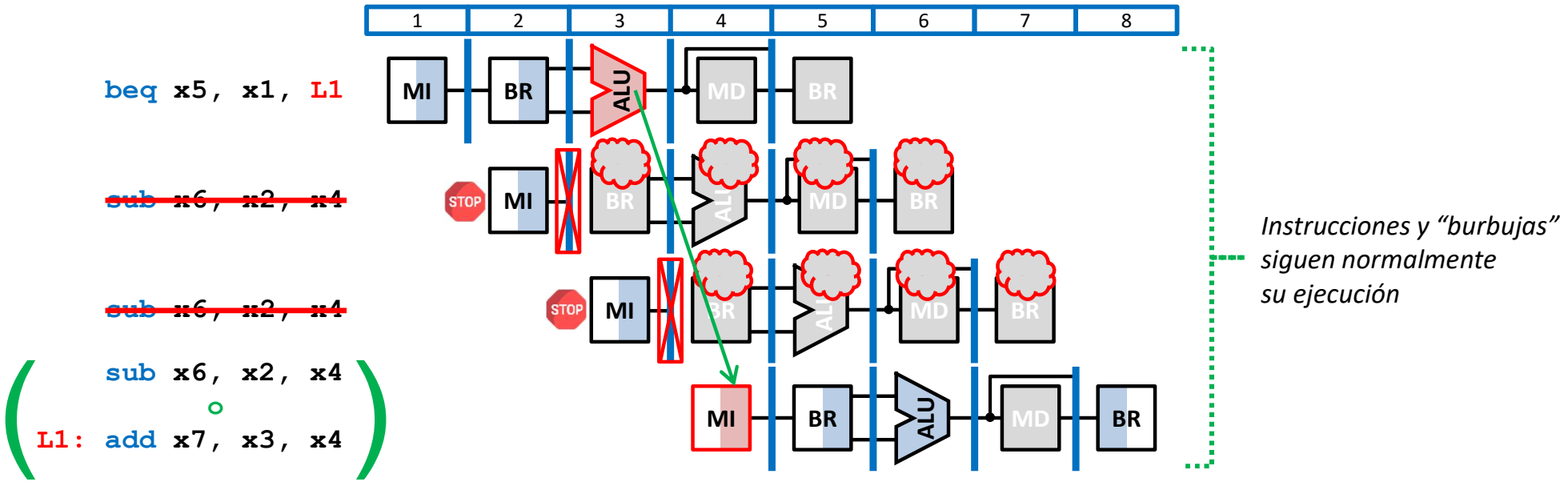
- Una solución consiste en parar el pipeline durante 2 ciclos para retrasar la ejecución de nuevas instrucciones hasta que el salto esté resuelto.
  - Ciclo 2: **beq** (está en ID) se detecta el conflicto. La instrucción **sub** se para y se inserta una "burbuja" de no operación en la etapa ID.
  - Ciclo 3: **beq** (está en EX) resuelve el salto, pero el conflicto continua. La instrucción **sub** permanece parada y se inserta otra "burbuja".
  - Ciclo 4: **beq** (está en MEM) se lanza la instrucción que corresponda.



Conflictos de control

Solución HW: parada

- Una solución consiste en parar el pipeline durante 2 ciclos para retrasar la ejecución de nuevas instrucciones hasta que el salto esté resuelto.
  - Ciclo 2: **beq** (está en ID) se detecta el conflicto. La instrucción **sub** se para y se inserta una "burbuja" de no operación en la etapa ID.
  - Ciclo 3: **beq** (está en EX) resuelve el salto, pero el conflicto continua. La instrucción **sub** permanece parada y se inserta otra "burbuja".
  - Ciclo 4: **beq** (está en MEM) se lanza la instrucción que corresponda.
  - Penaliza la ejecución del programa en dos ciclos por instrucción de salto.



Conflictos de control

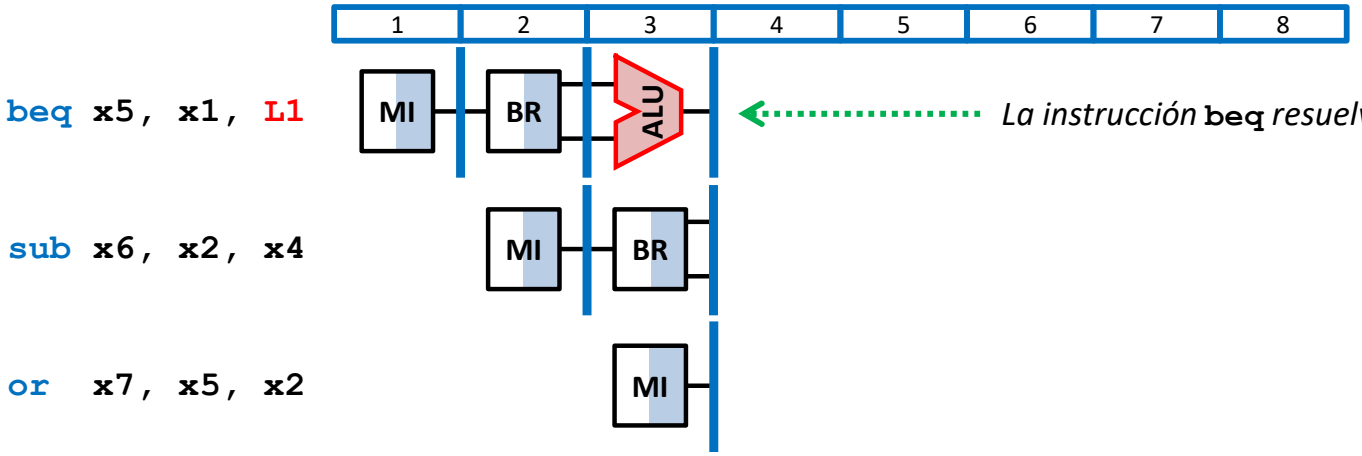
Solución HW: predicción de salto (i)

- Para instrucciones **`beq`** existe una solución mejor que consiste en predecir que el salto no se tomará.
  - La instrucción **`beq`** y las que le siguen en secuencia se lanzan normalmente.
  - Cuando se conoce la dirección/decisión de salto (**`beq`** está en etapa EX):
    - Si no hay que saltar, no se hace nada.
    - Si hay que saltar, se descartan (*flush*) las 2 últimas instrucciones lanzadas, insertando "burbujas" de no operación en las etapas ID y EX.
  - En el ciclo siguiente (**`beq`** está en etapa MEM) se lanza la instrucción que corresponda.
  - Penaliza la ejecución del programa en dos ciclos por instrucción de salto tomado, no hay penalización si el salto no se toma.

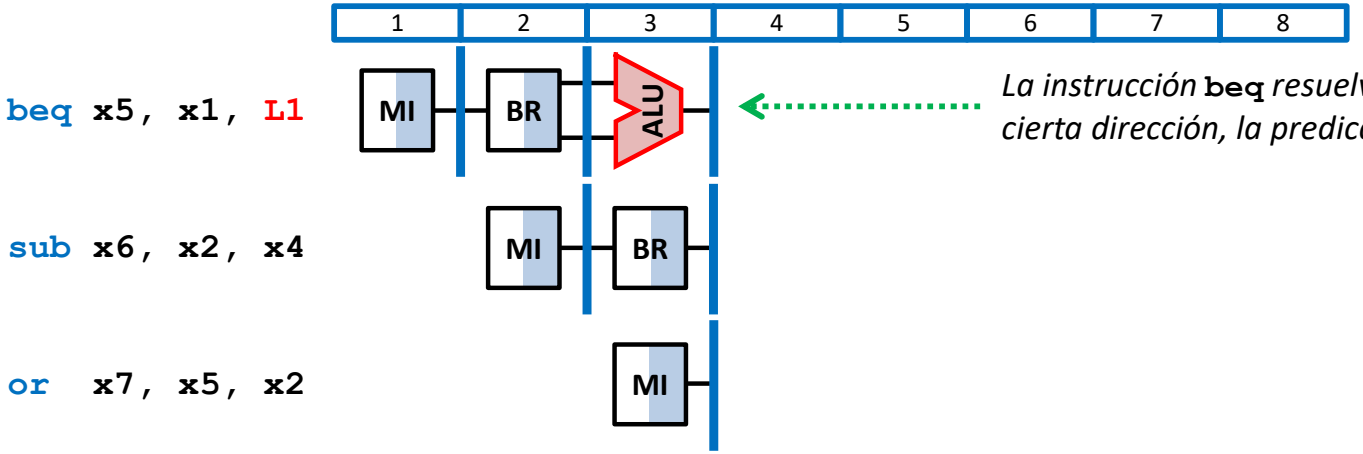
- La opción complementaria, predecir que el salto se toma, es mucho más compleja porque requiere también predecir la dirección a la que saltar.

Conflictos de control

Solución HW: predicción de salto (ii)



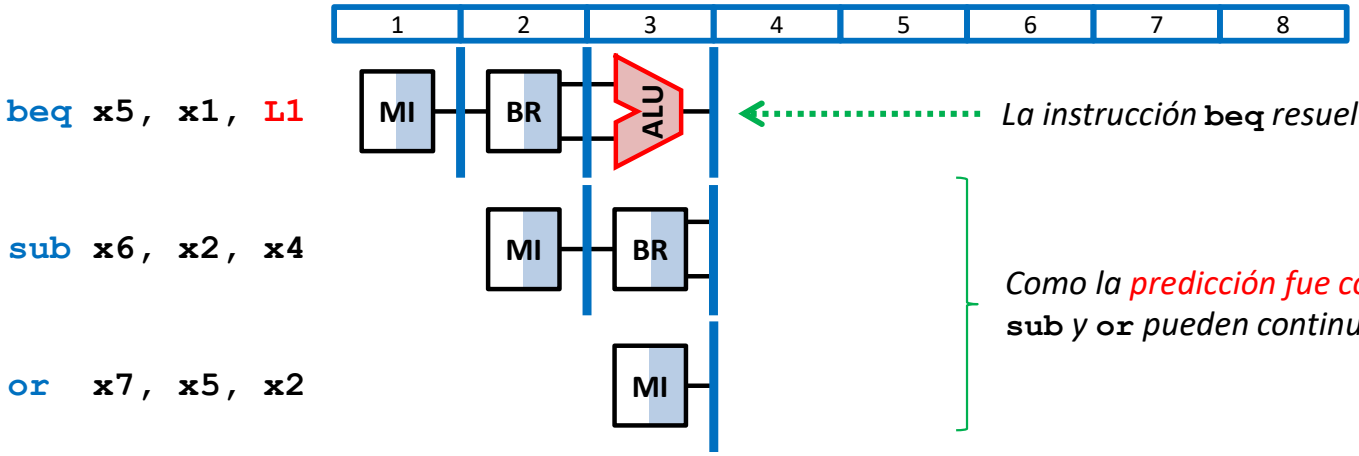
La instrucción **beq** *resuelve que no hay que saltar*



La instrucción **beq** *resuelve que hay que saltar a una cierta dirección, la predicción falló*

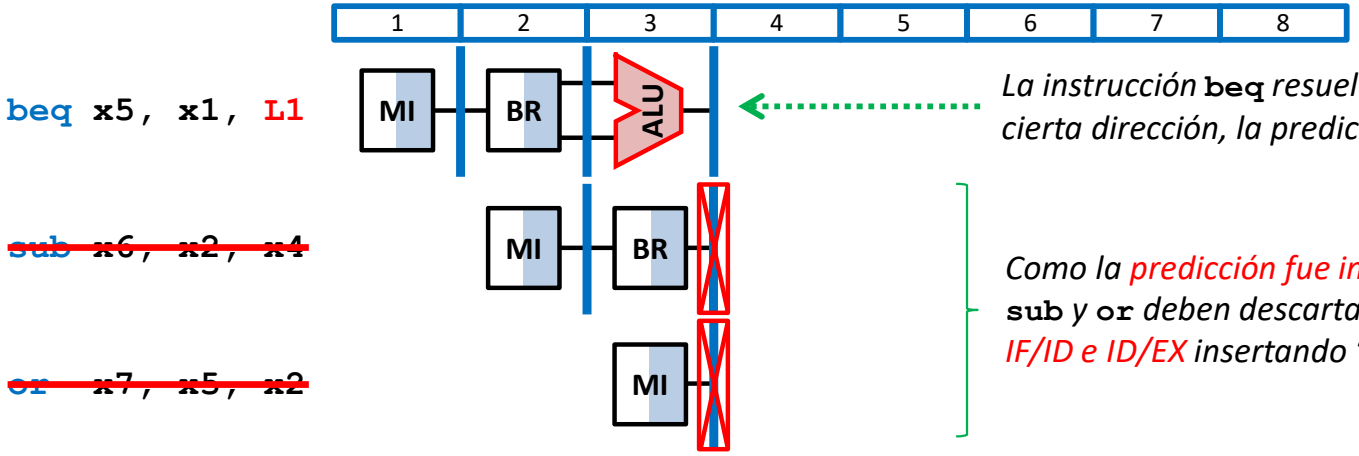
Conflictos de control

Solución HW: predicción de salto (ii)



La instrucción **beq** *resuelve que no hay que saltar*

Como la predicción fue correcta, las instrucciones **sub** *y* **or** *pueden continuar su ejecución*

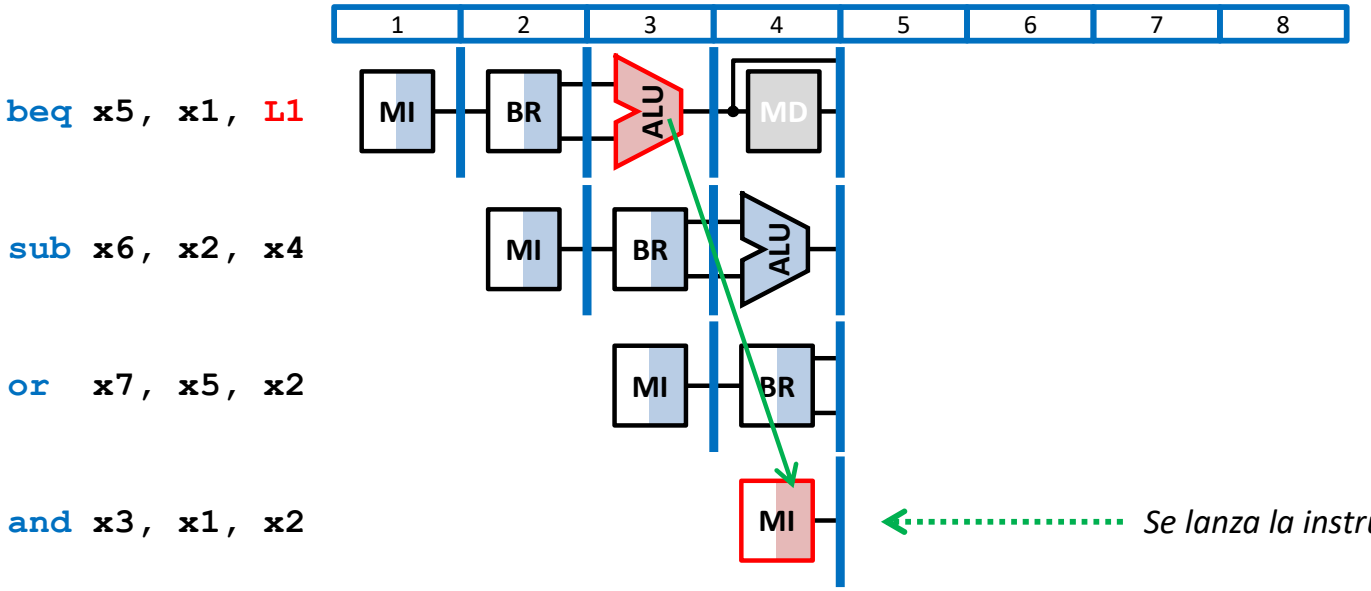


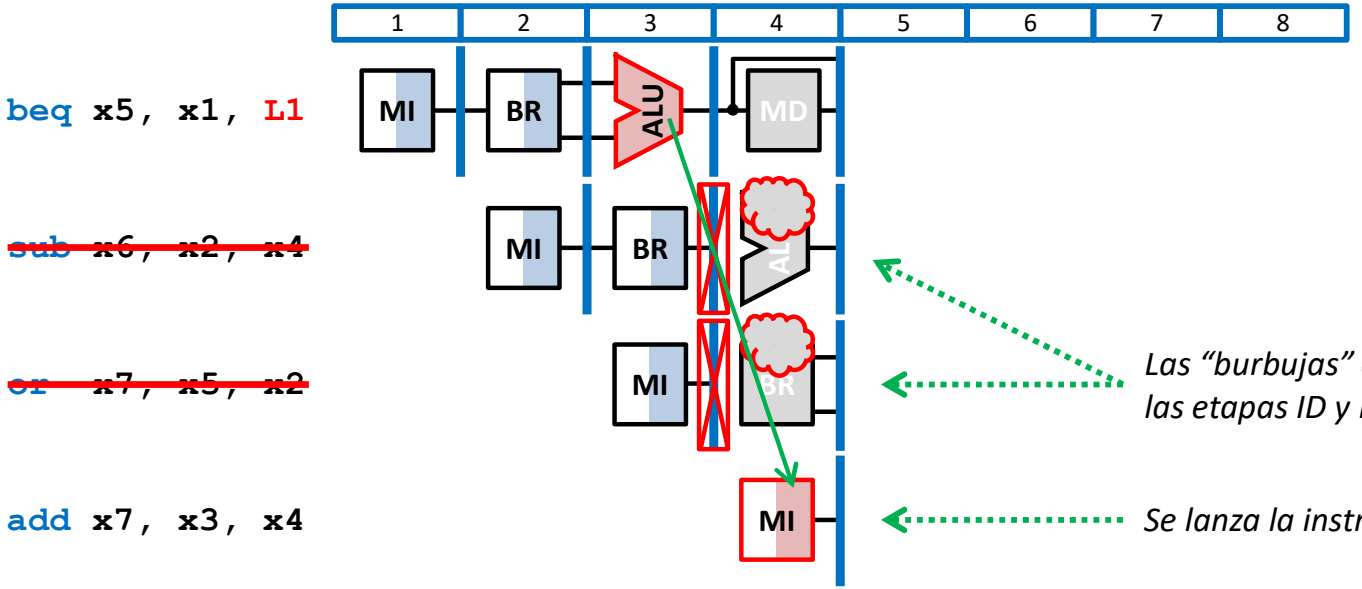
La instrucción **beq** *resuelve que hay que saltar a una cierta dirección, la predicción falló*

Como la predicción fue incorrecta, las instrucciones **sub** *y* **or** *deben descartarse: se borran los registros IF/ID e ID/EX insertando "burbujas"*

Conflictos de control

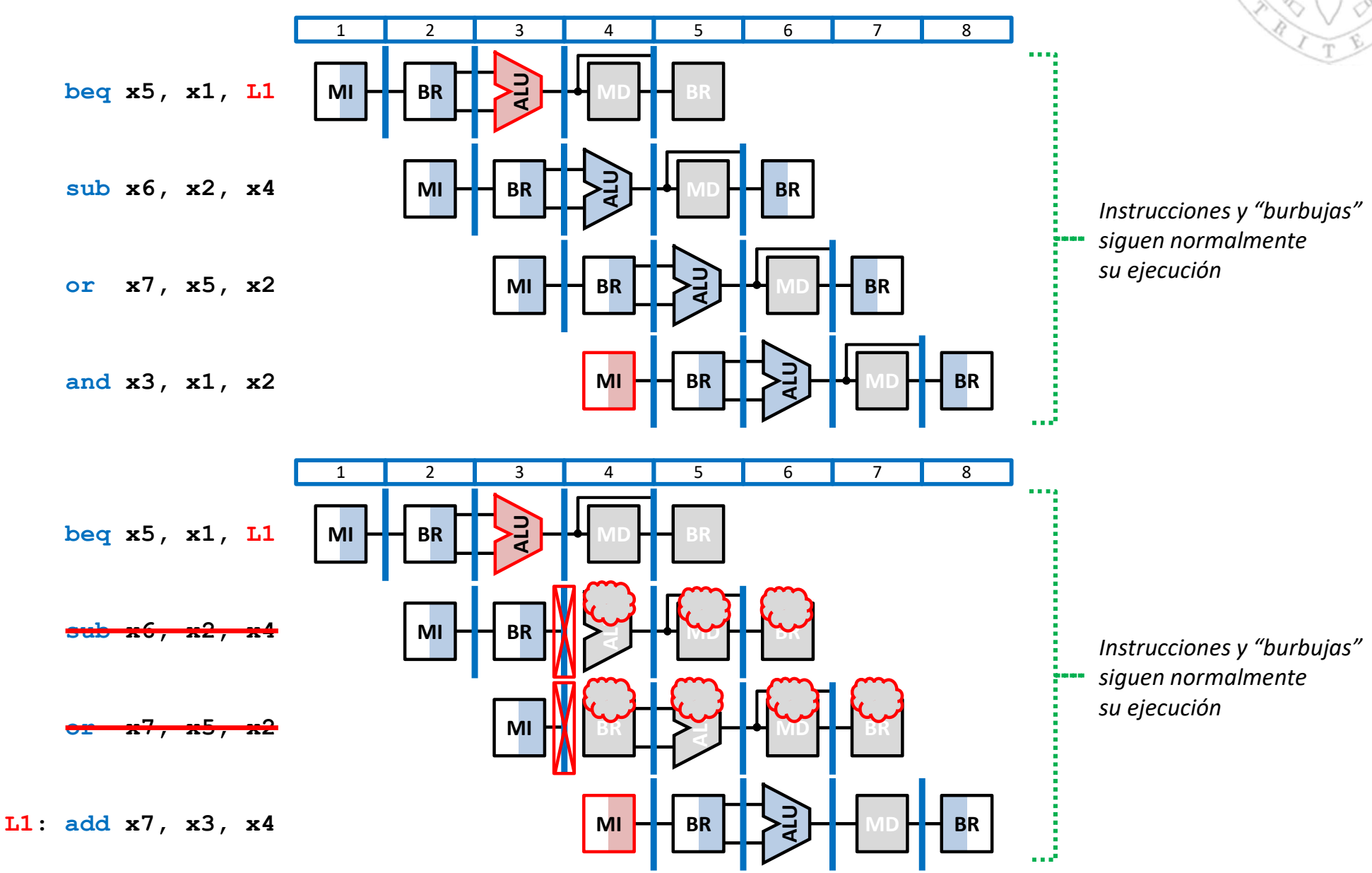
Solución HW: predicción de salto (ii)





Conflictos de control

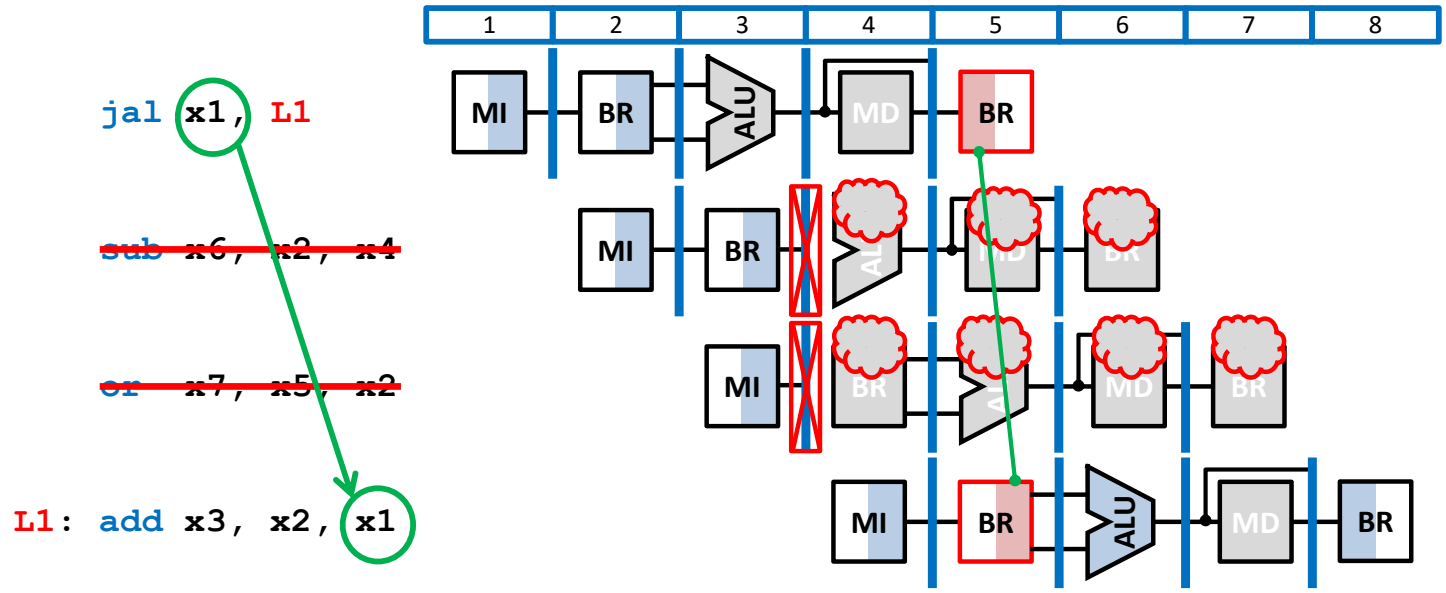
Solución HW: predicción de salto (ii)



Conflictos de control

Solución HW: predicción de salto (iii)

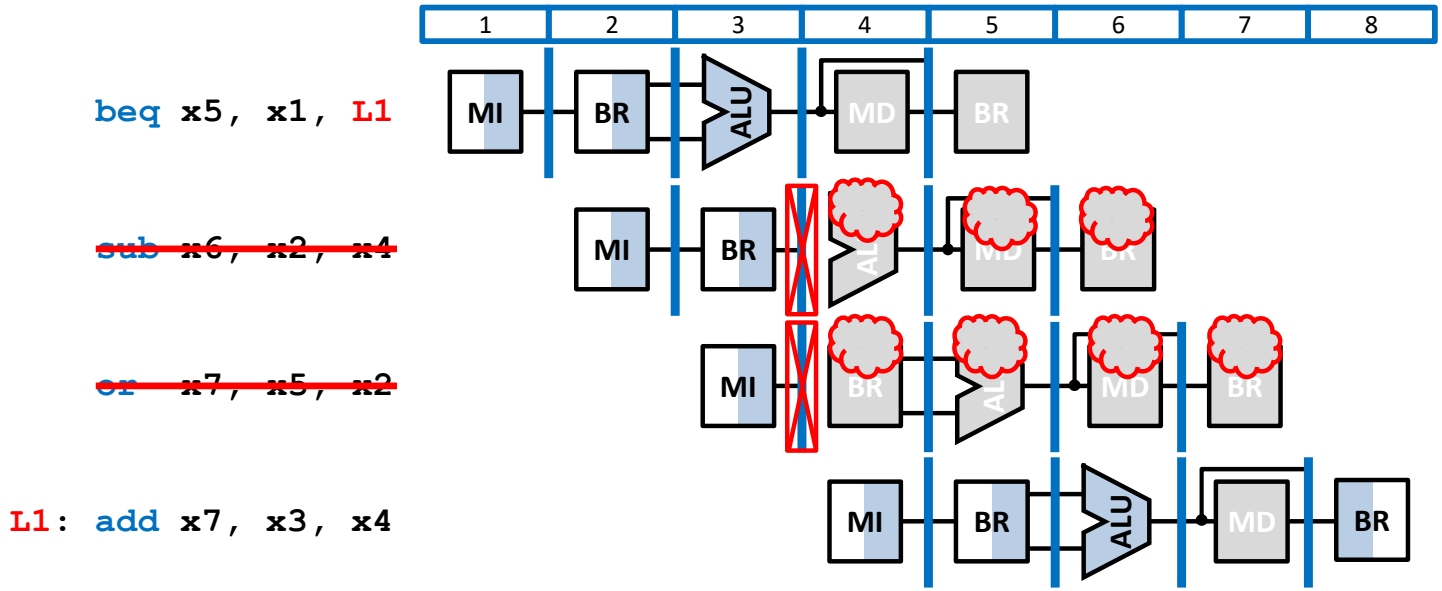
- Para instrucciones **`jal`** (que siempre saltan) la predicción de salto no tomado siempre falla, pero se usa porque:
  - Su penalización es la misma que en el caso de parada.
  - No requiere lógica adicional a la necesaria para **`beq`**.
  - Implícitamente resuelve un conflicto de datos no estudiado.
    - La instrucción **`jal`** almacena PC+4 en **`x1`** durante la etapa WB, un valor que no pasa por la ALU y por tanto no puede ser anticipado en la ruta datos diseñada.
    - Gracias al retraso de 2 ciclos, el valor actualizado de **`x1`** puede leerse del BR.



Conflictos de control

Solución HW: predicción de salto (iv)

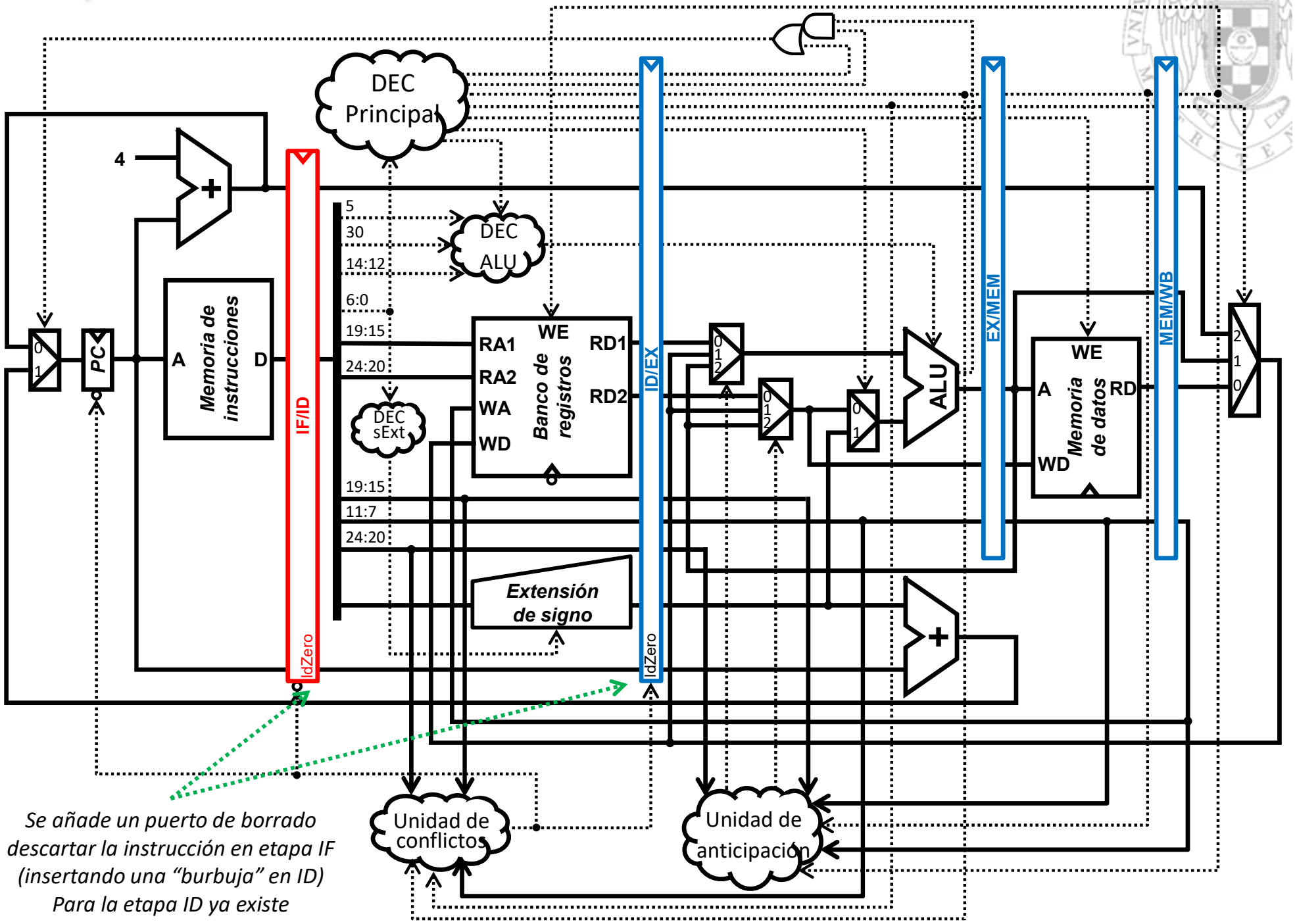
- En diagramas de ejecución simplificados se marcan explícitamente las instrucciones descartadas:



	1	2	3	4	5	6	7	8
beq x5, x1, L1	IF	ID	EX	M	WB			
sub x6, x2, x4		IF	ID	X				
or x7, x5, x2			IF	X				
L1: add x7, x3, x4				IF	ID	EX	M	WB

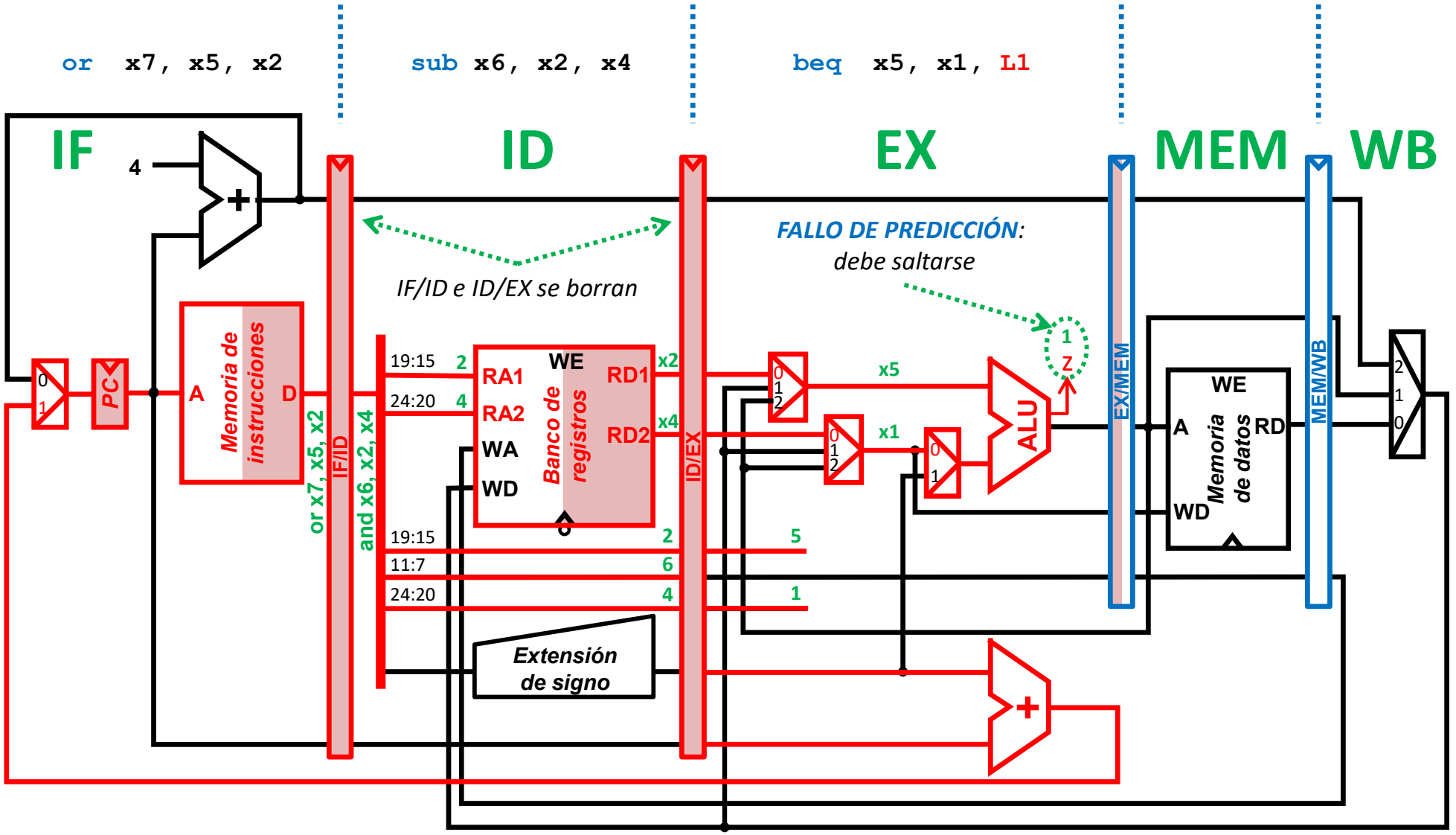
Procesador segmentado

+ predicción de salto: ruta de datos



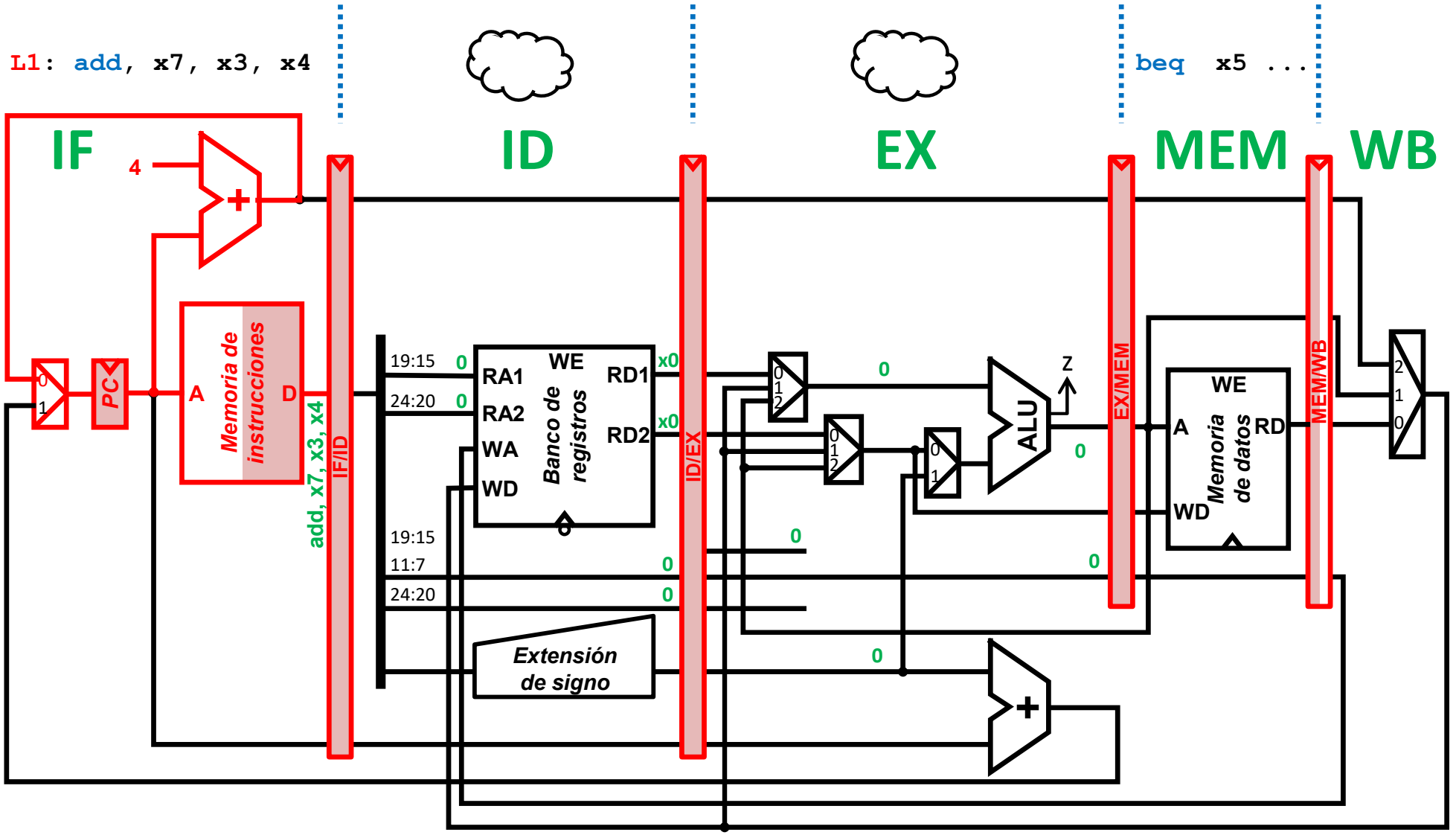
Procesador segmentado

Simulación de predicción de salto incorrecta: 3er. ciclo



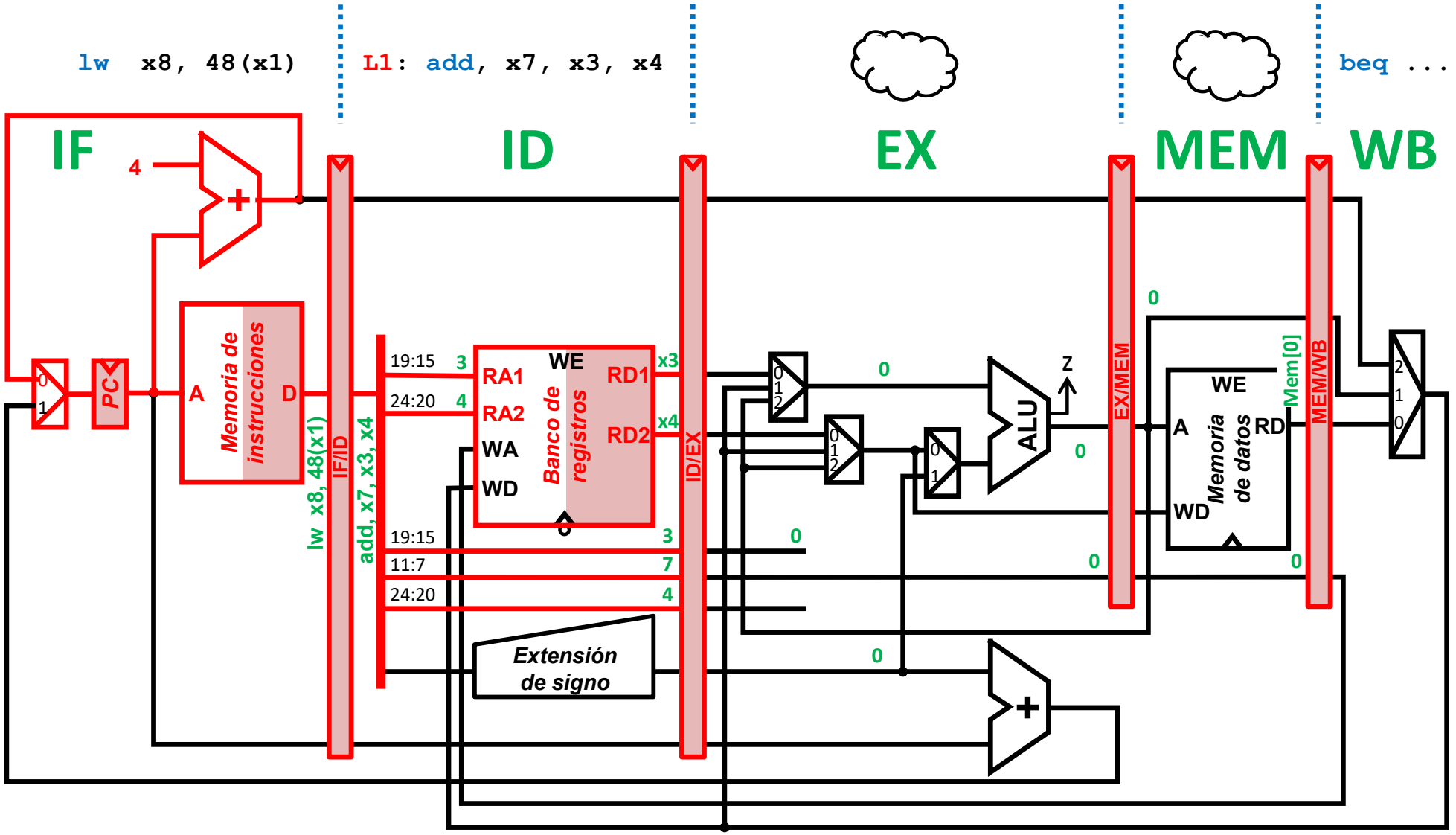
Procesador segmentado

Simulación de predicción de salto incorrecta: 4o. ciclo



Procesador segmentado

Simulación de predicción de salto incorrecta: 5o. ciclo



Procesador segmentado

Diseño de la unidad de conflictos ampliada (i)

- La unidad de conflictos se amplía para que, en caso de que el salto deba tomarse, se descarten las instrucciones de las etapas IF e ID:
  - Borrando los registros de segmentación IF/ID e ID/EX.

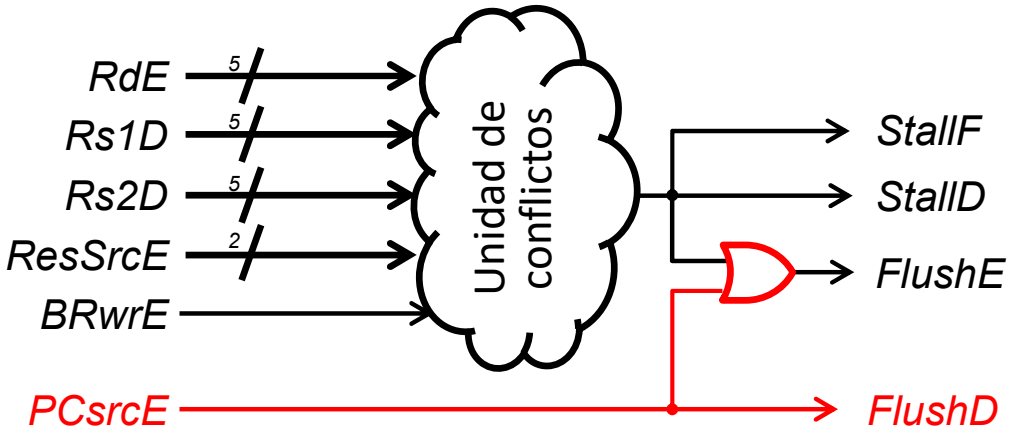
- Para realizar su función necesita conocer:
  - PCsrcE: solo se activa si la instrucción en la etapa EX es de salto y ha resuelto saltar.

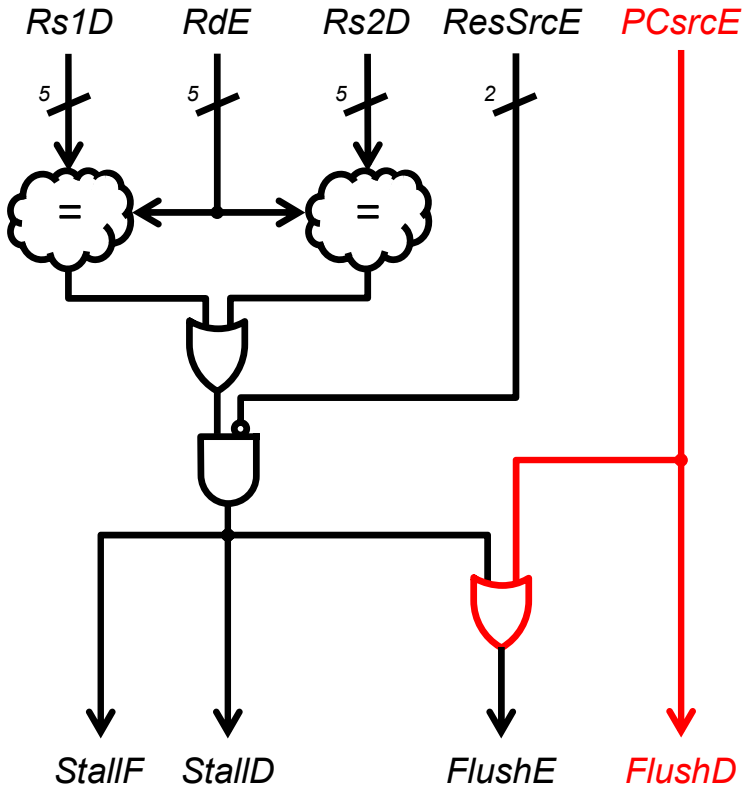
Stall ← *if* ((ResSrcE = 0) & BRwrE & ((Rs1D = RdE) | (Rs2D = RdE))) *then* (1)
else (0)
StallF ← Stall
StallD ← Stall
FlushE ← Stall | PCsrcE ⇠ *Borra el registro de segmentación ID/EX*
FlushD ← PCsrcE ⇠ *Borra el registro de segmentación IF/ID*

Procesador segmentado

Diseño de la unidad de conflictos ampliada (ii)

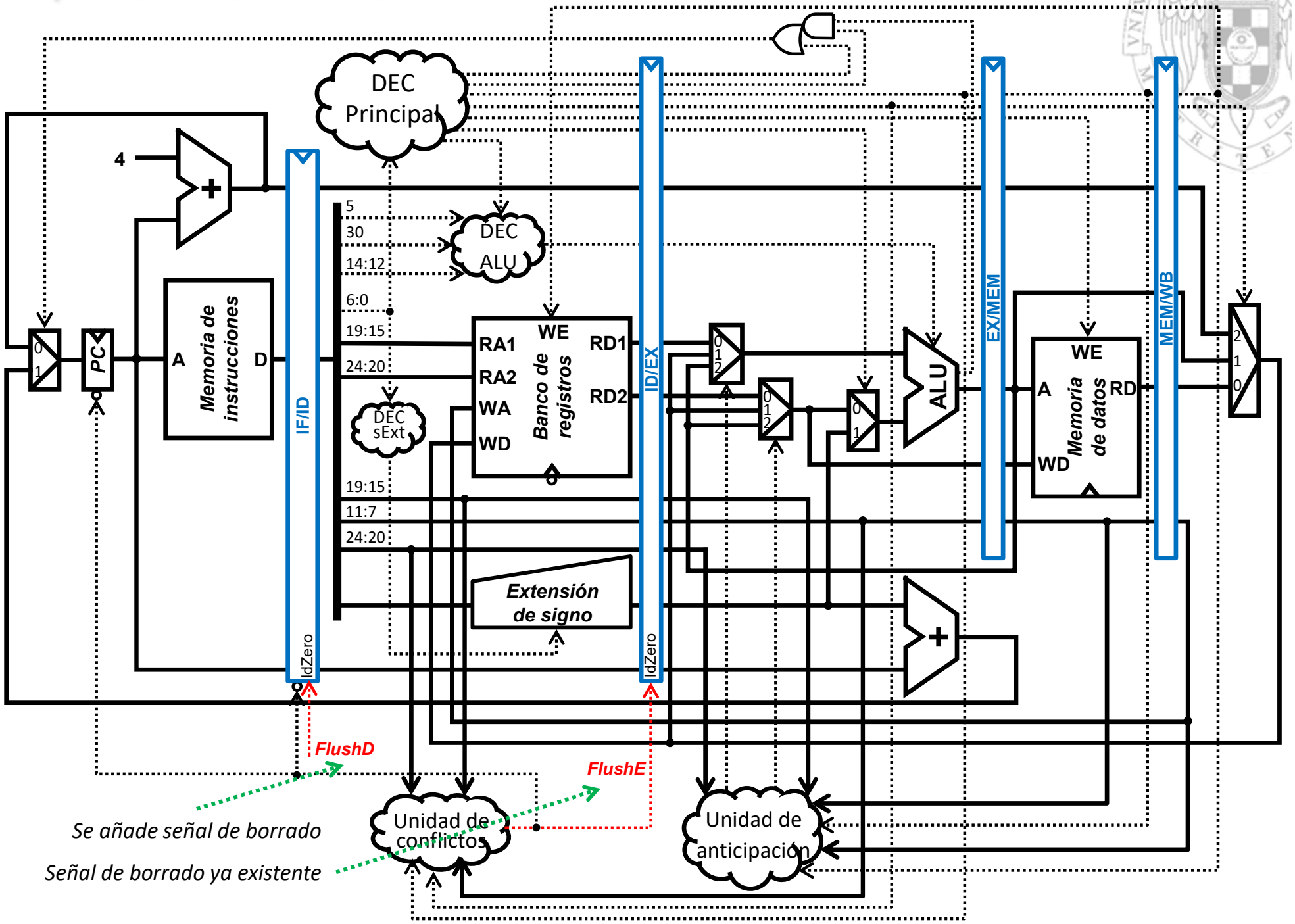
Stall ← *if* ((ResSrcE = 0) & BRwrE & ((Rs1D = RdE) | (Rs2D = RdE))) *then* (1)
else (0)
StallF ← Stall
StallD ← Stall
FlushE ← Stall | PCsrcE
FlushD ← PCsrcE





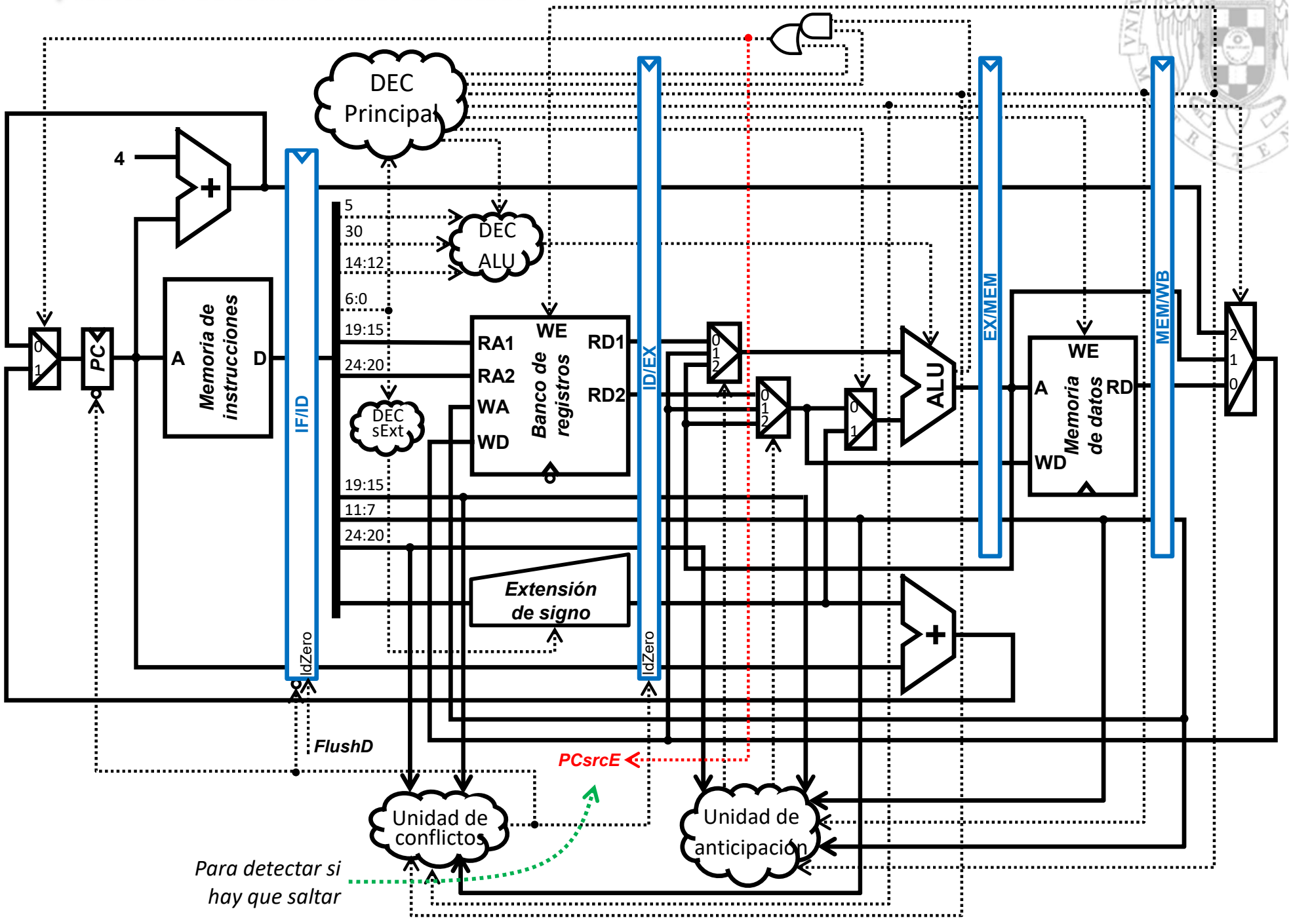
Procesador segmentado

+ predicción de salto: señales de control



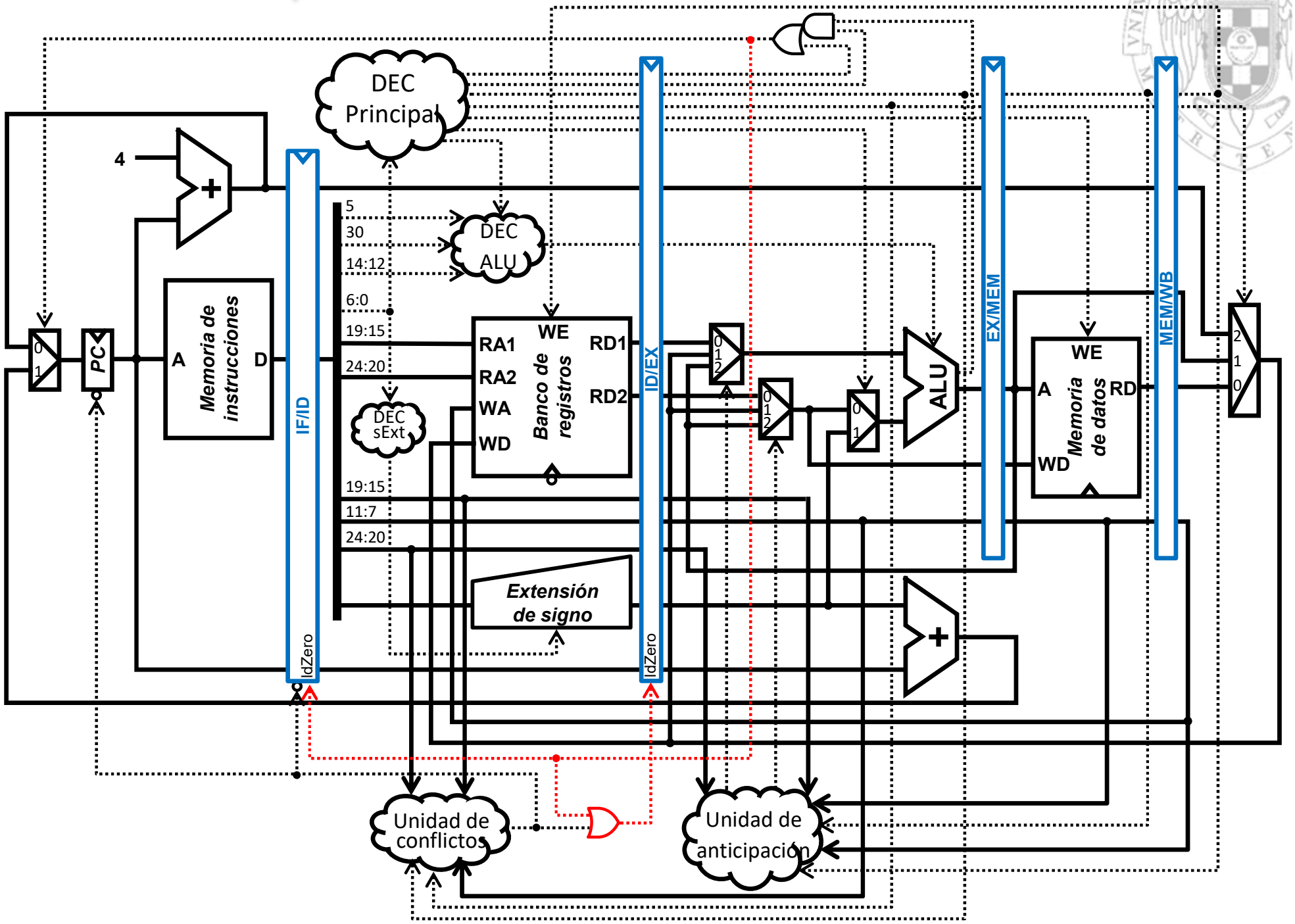
Procesador segmentado

+ predicción de salto: señales de estado



Procesador segmentado

Con resolución completa de conflictos



Procesador segmentado

Resolución de conflictos SW vs. HW

Tipo de Conflicto	Instrucciones implicadas	Penalización (ciclos) SW	Penalización (ciclos) HW	*Solución HW implementada*
Estructural	*(no existe este tipo de conflicto)*	-	-	-
Datos	tipo `add/i` → resto	1, 2 ó 3	0	*anticipación*
	`lw` → tipo `add/i`	1, 2 ó 3	1	*parada + anticipación* *(evitable reordenando código)*
	`lw` → `lw`	1, 2 ó 3	1	*parada + anticipación* *(evitable reordenando código)*
	`lw` → `sw`	1, 2 ó 3	1	*parada + anticipación* *(evitable optimizando anticipación)*
Control	`beq`	2	0 ó 2	*predicción de salto*
	`jal`	2	2	*predicción de salto*

Procesador segmentado

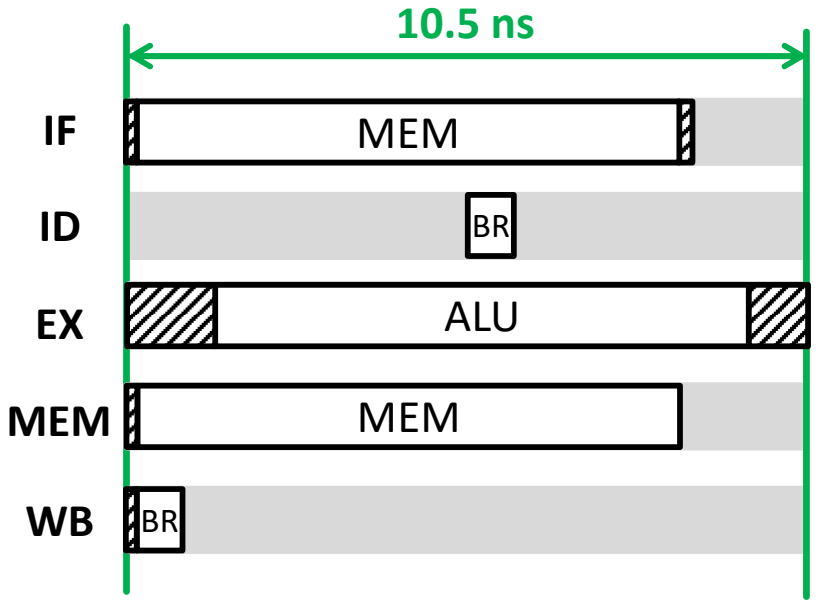
Coste y tiempo de ciclo (CMOS 90 nm)

$$area = 77018\ \mu m^2$$

$$t_{clk} = 10.5\ \text{ns}$$

$$f_{clk} = \frac{1}{t_{clk}} = \frac{1}{10.5 \cdot 10^{-6}\text{s}} = 95\ \text{MHz}$$

etapa	camino crítico
IF	8890 ps
ID	½t_{clk}+723 ps
EX	**10541 ps**
MEM	8667 ps
WB	1122 ps
máx.	**10541 ps**



- CPI segmentado ideal: sin penalización por conflictos (CPI = 1).

$$CPI = 1$$
$$t_{ejec} = 10^8 \cdot 1 \cdot 10.5\ ns = 1.05\ s$$
$$MIPS = 10^8/(10^6 \cdot 1.05\ s) = 95.2\ Minst/s$$

Procesador segmentado

Métricas de rendimiento

- Sea un programa que ejecuta 10^8 instrucciones (100 millones) tal que:
  - 25% de las instrucciones son de tipo `lw`
    - 40% le sigue una instrucción que necesita el valor cargado: parada de 1 ciclo.
  - 10% de las instrucciones son de tipo `sw`
  - 11% de las instrucciones son de tipo `beq`
    - 50% de los saltos se toman: salto mal predicho y 2 instrucciones se descartan.
  - 2% de las instrucciones son de tipo `jal`
  - 52% de las instrucciones son aritmético-lógicas
- CPI segmentado real: con penalización por conflictos (CPI > 1).
  - `lw`: 1/2 ciclos, `beq`: 1/3 ciclos, `jal`: 3 ciclos, *resto*: 1 ciclo

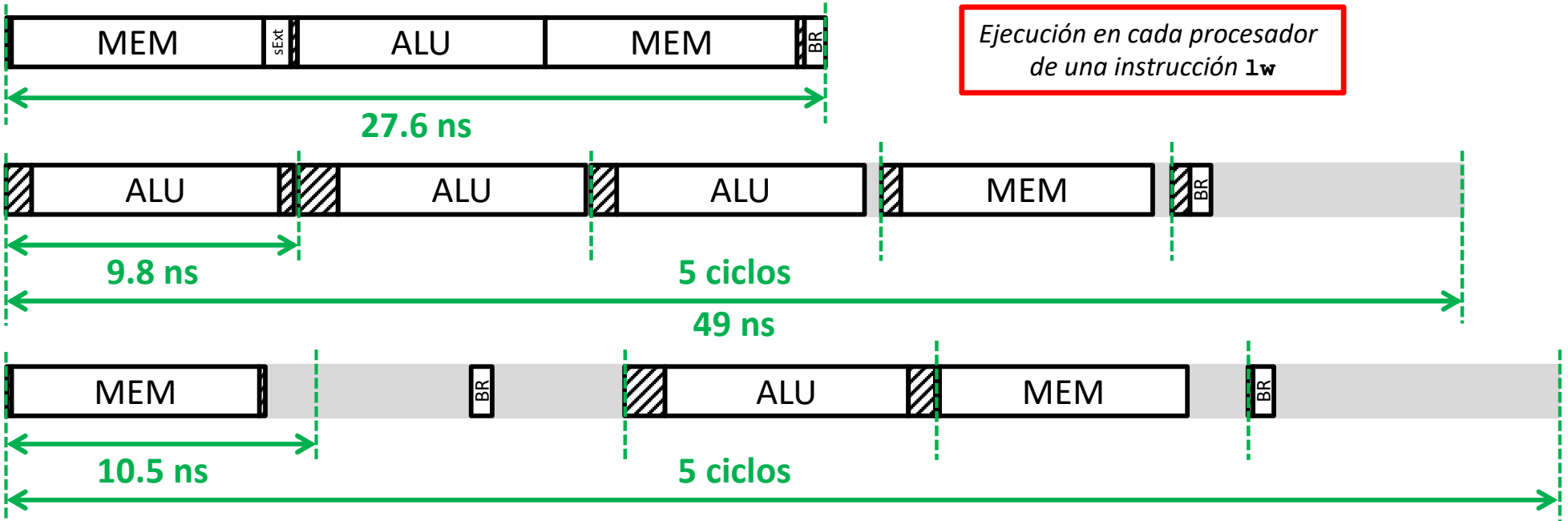
$$
\begin{aligned}
CPI &= 0.25 \cdot (0.6 \cdot 1 + 0.4 \cdot 2) && \leftarrow \text{Instrucciones } \texttt{lw} \\
&\quad + 0.10 \cdot 1 && \leftarrow \text{Instrucciones } \texttt{sw} \\
&\quad + 0.11 \cdot (0.5 \cdot 1 + 0.5 \cdot 3) && \leftarrow \text{Instrucciones } \texttt{beq} \\
&\quad + 0.02 \cdot 3 && \leftarrow \text{Instrucciones } \texttt{jal} \\
&\quad + 0.52 \cdot 1 = \mathbf{1.25} && \leftarrow \text{Instrucciones aritmético-lógicas} \\
t_{ejec} &= 10^8 \cdot 1.25 \cdot 10.5\ ns = \mathbf{1.31\ s} \\
MIPS &= 10^8/(10^6 \cdot 1.43\ s) = \mathbf{76.2\ Minst/s}
\end{aligned}
$$

Comparativa

RISC-V reducido: monociclo vs. multiciclo vs. segmentado

- El procesador segmentado es el más caro pero tiene mejor rendimiento

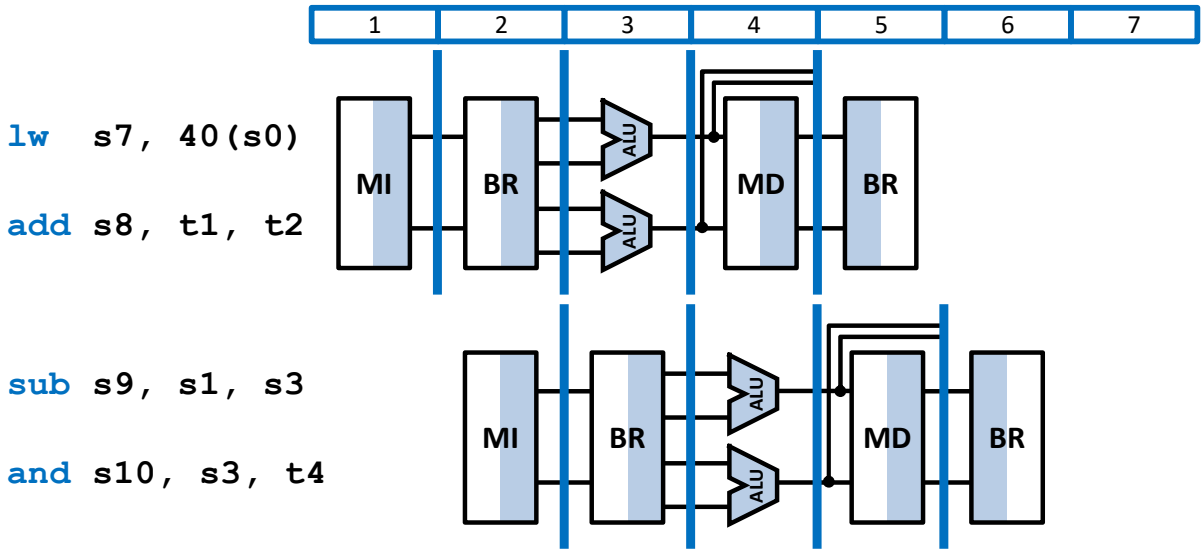
Procesador	t_{clk} (ns)	CPI	Coste (µm²)	t_{ejec} (s)	↑ coste	Speedup
Monociclo	27.6	1	59181	2.76	1	1
Multiciclo	9.8	4.14	65626	4.06	1.12	0.68
Segmentado	10.5	1.25	77018	1.31	1.30	2.11



Microarquitecturas avanzadas

Procesadores superescalares

- Un procesador superescalar contiene varias copias de la ruta de datos:
  - Ejecuta en paralelo varias instrucciones del mismo programa/hebra.
  - Un procesador superescalar de 2 vías:
    - Tiene 2 ALUs, y el BR y la memoria tienen duplicados sus puertos.
    - Lanza 2 instrucciones por ciclo (CPI ideal = ½).

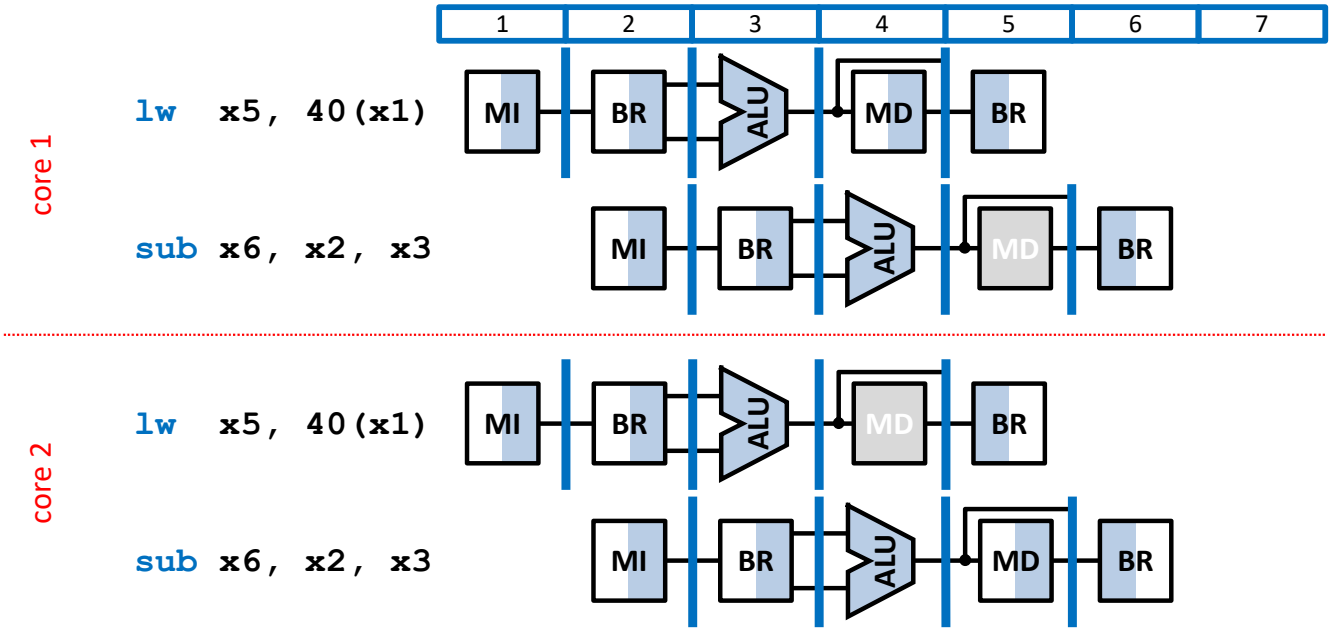


  - La probabilidad de conflicto de datos/control es mayor:
    - Para reducirla, ejecuta las instrucciones en un orden distinto al que están escritas en el programa (fuera de orden) asegurando que el resultado sea el mismo.

Microarquitecturas avanzadas

Procesadores multicore

- Un procesador multicore contiene varias copias del procesador completo:
  - Ejecuta en paralelo varias instrucciones de distintos programas/hebras.
  - Un procesador dual core:
    - Tiene 2 procesadores segmentados completos que comparten memoria.
    - Lanza 2 instrucciones por ciclo (CPI ideal = ½).



  - Los cores pueden ser a su vez superescalares:
    - Un dual core superscalar de 2 vías, lanza 4 instrucciones por ciclo (CPI ideal = ¼).

Microarquitecturas avanzadas

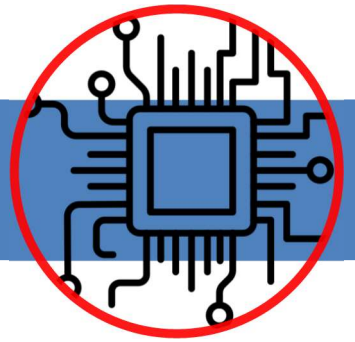
Procesadores Intel

Microprocesador	Año	f_{clk} (MHz)	# Etapas	# Vias	Fuera de orden	# cores
486	1989	25	5	1	No	1
Pentium	1993	66	5	2	No	1
Pentium Pro	1997	200	10	3	Sí	1
Pentium 4 Willamette	2001	2000	22	3	Sí	1
Pentium 4 Prescott	2004	3600	31	3	Sí	1
Core	2006	3600	14	4	Sí	2
Core i7 Nehalem	2008	3600	14	4	Sí	2-4
Core Westmere	2010	3730	14	4	Sí	6
Core i7 Ivy Bridge	2012	3400	14	4	Sí	6
Core Broadwell	2014	3700	14	4	Sí	10
Core i9 Skylake	2016	3100	14	4	Sí	14
Ice Lake	2018	4200	14	4	Sí	16

fuente: A.A. Patterson & J.L. Hennesy, Computer Organization and Design, RISC-V Edition (2nd. edition). (2021)

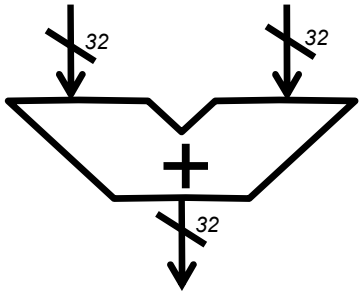
- Cálculo del coste.
- Cálculo del tiempo de ciclo.

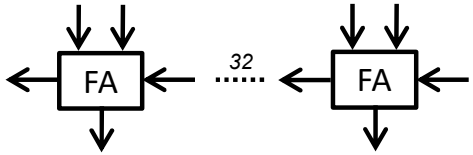
Apéndice tecnológico



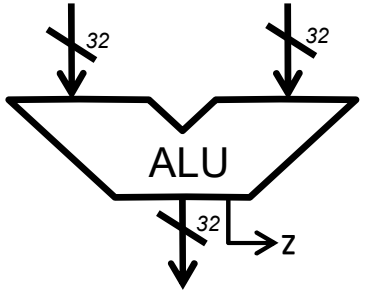
Cálculo del coste y tiempo de ciclo

CMOS 90 nm

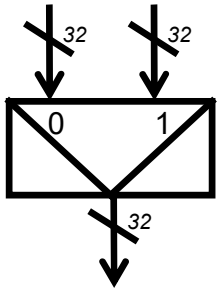


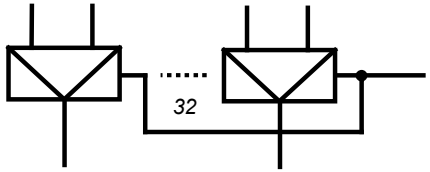


área: 32×29.49 = **944 μm²**
retardo: 32×226 = **7232 ps**

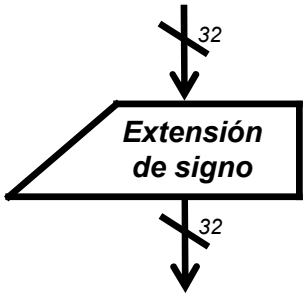


área: 3052 μm²
retardo: 8360 ps

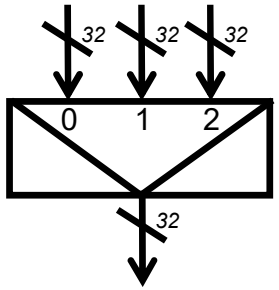


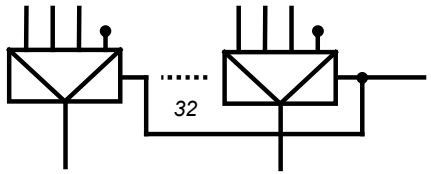


área: 32×11.05 = **354 μm²**
retardo: 223 ps

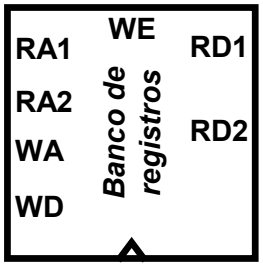


área: 202 μm²
retardo: 460 ps





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



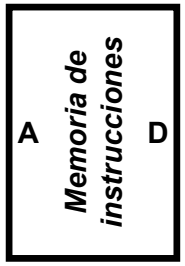
área: 51405 μm²
retardo lectura: 723 ps
setup escritura: 705 ps
(debido al DEC de dirección)

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

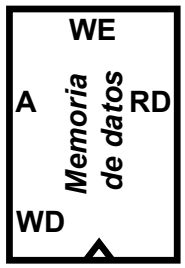
Cálculo del coste y tiempo de ciclo

CMOS 90 nm

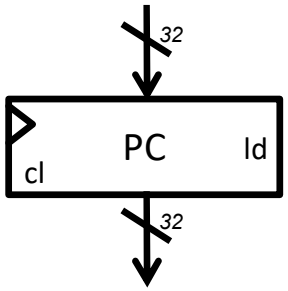
Comportamiento idealizado: retardo comparable al de la ALU
(para que pueda leerse en un ciclo de reloj)

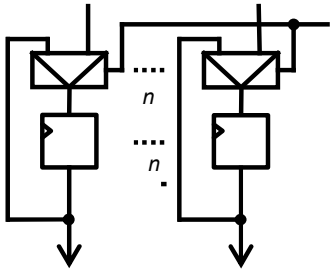


área: –
tiempo de acceso: **8500 ps**



área: –
tiempo de acceso: **8500 ps**





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

área: **56 μm²**
retardo: **490 ps**

área: **65 μm²**
retardo: **451 ps**

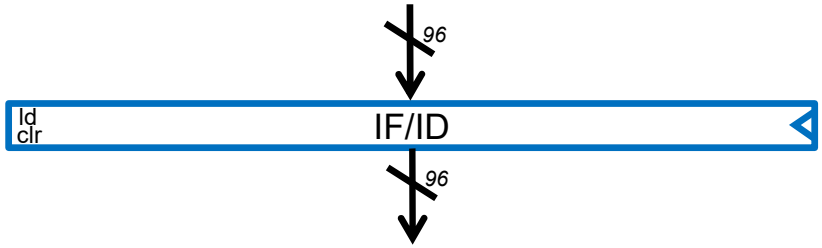
área: **21 μm²**
retardo: **451 ps**

área: **15 μm²**
retardo: **351 ps**

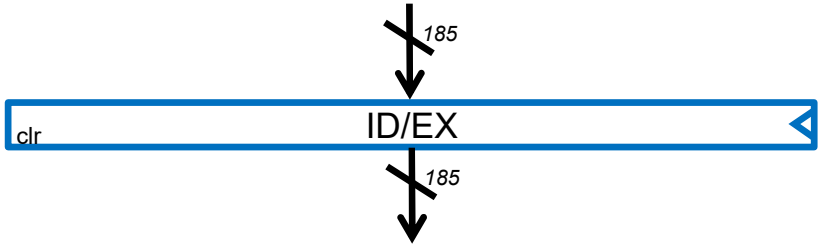
Biblioteca de celdas CMOS 90 nm, fuente: Synopsys (SAED EDK 90 nm)

Cálculo del coste y tiempo de ciclo

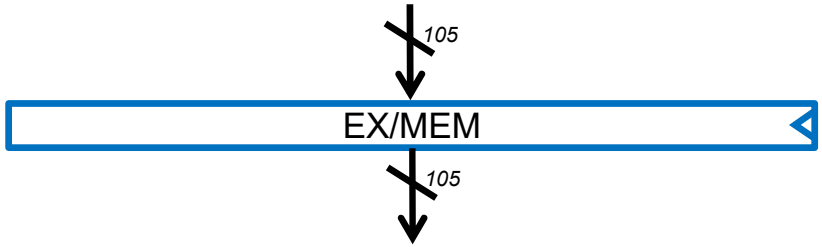
CMOS 90 nm



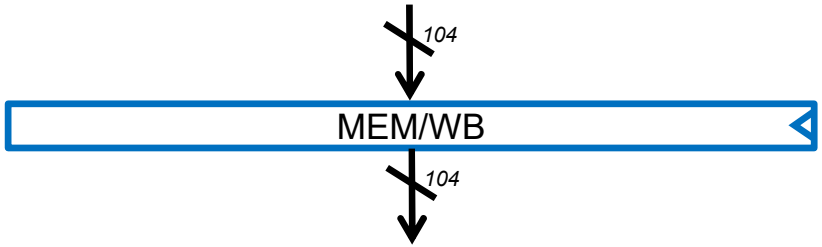
área: 96×11.05 + 96×32.26 = **4158 μm²**
retardo CLK→Q: **167 ps**
setup: 1×223 = **223 ps** (debido a MUX de carga)



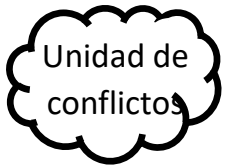
área: 185×32.26 = **5968 μm²**
retardo CLK→Q: 1×167 = **167 ps**
setup: **0 ps**



área: 105×24.88 = **2612 μm²**
retardo CLK→Q: **167 ps**
setup: **0 ps**



área: 104×24.88 = **2588 μm²**
retardo CLK→Q: **167 ps**
setup: **0 ps**

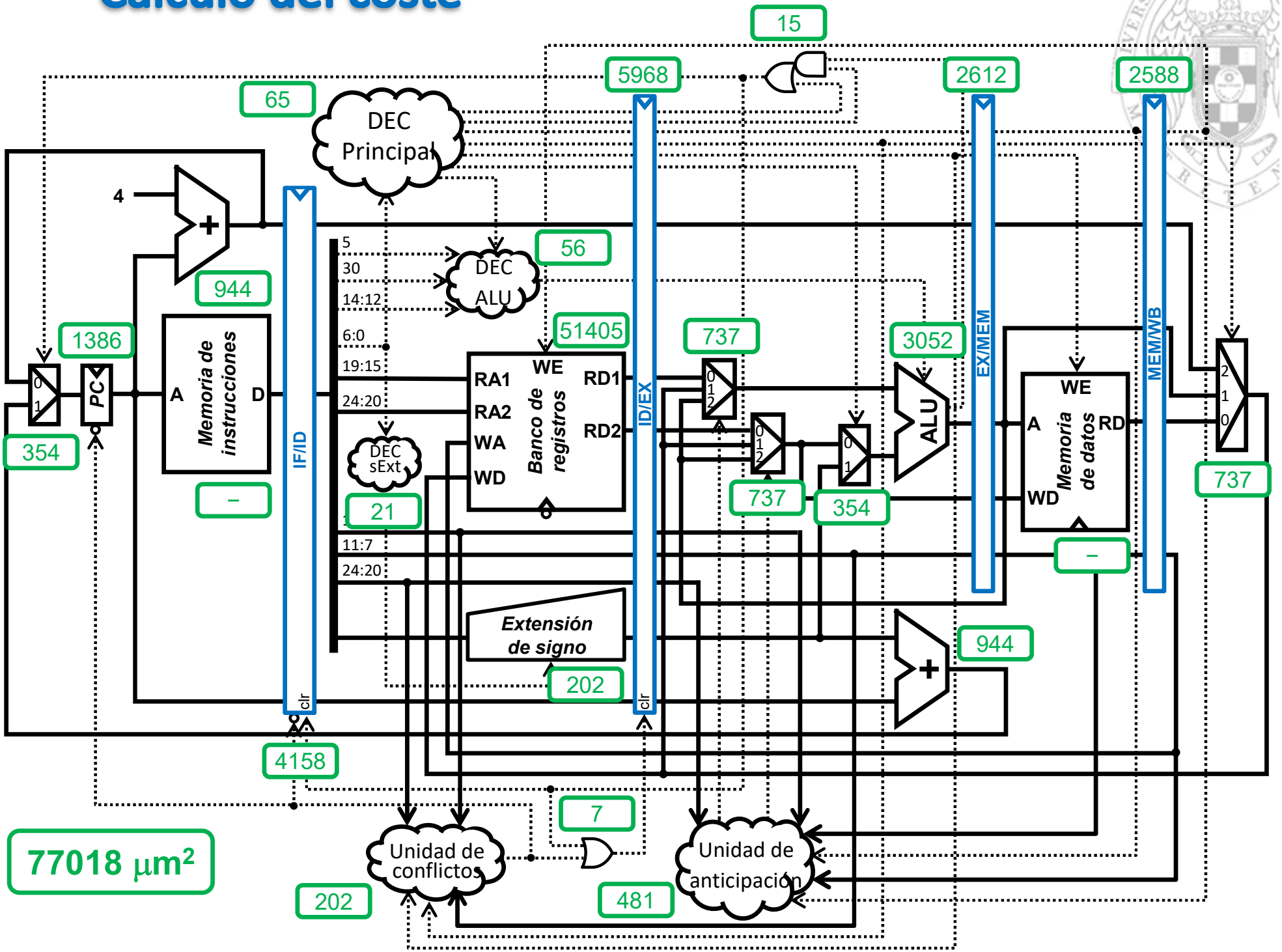


área: **195 μm²**
retardo: **881 ps**

área: **7 μm²**
retardo: **171 ps**

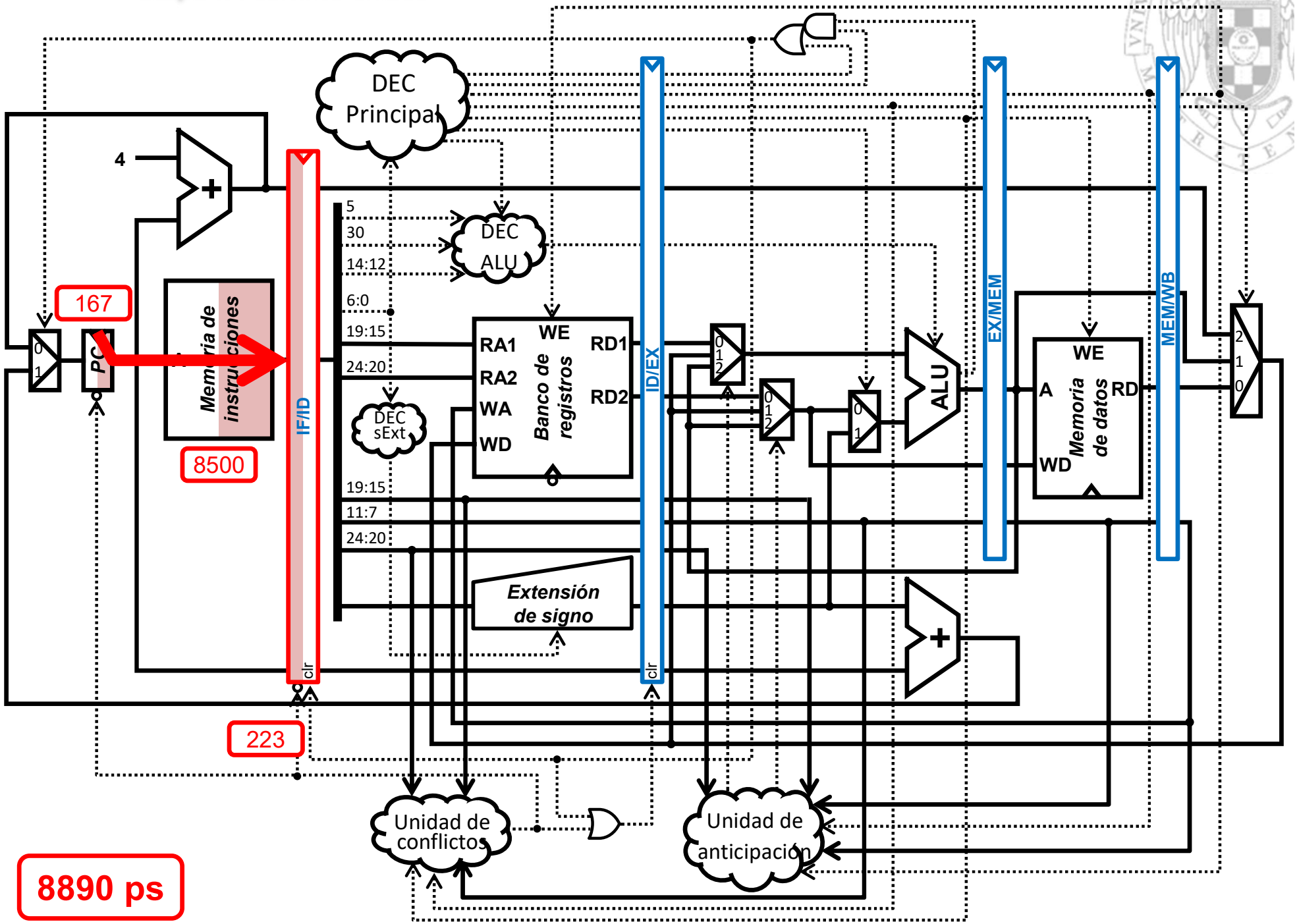
área: **418 μm²**
retardo: **744 ps**

Cálculo del coste



Cálculo del tiempo de ciclo

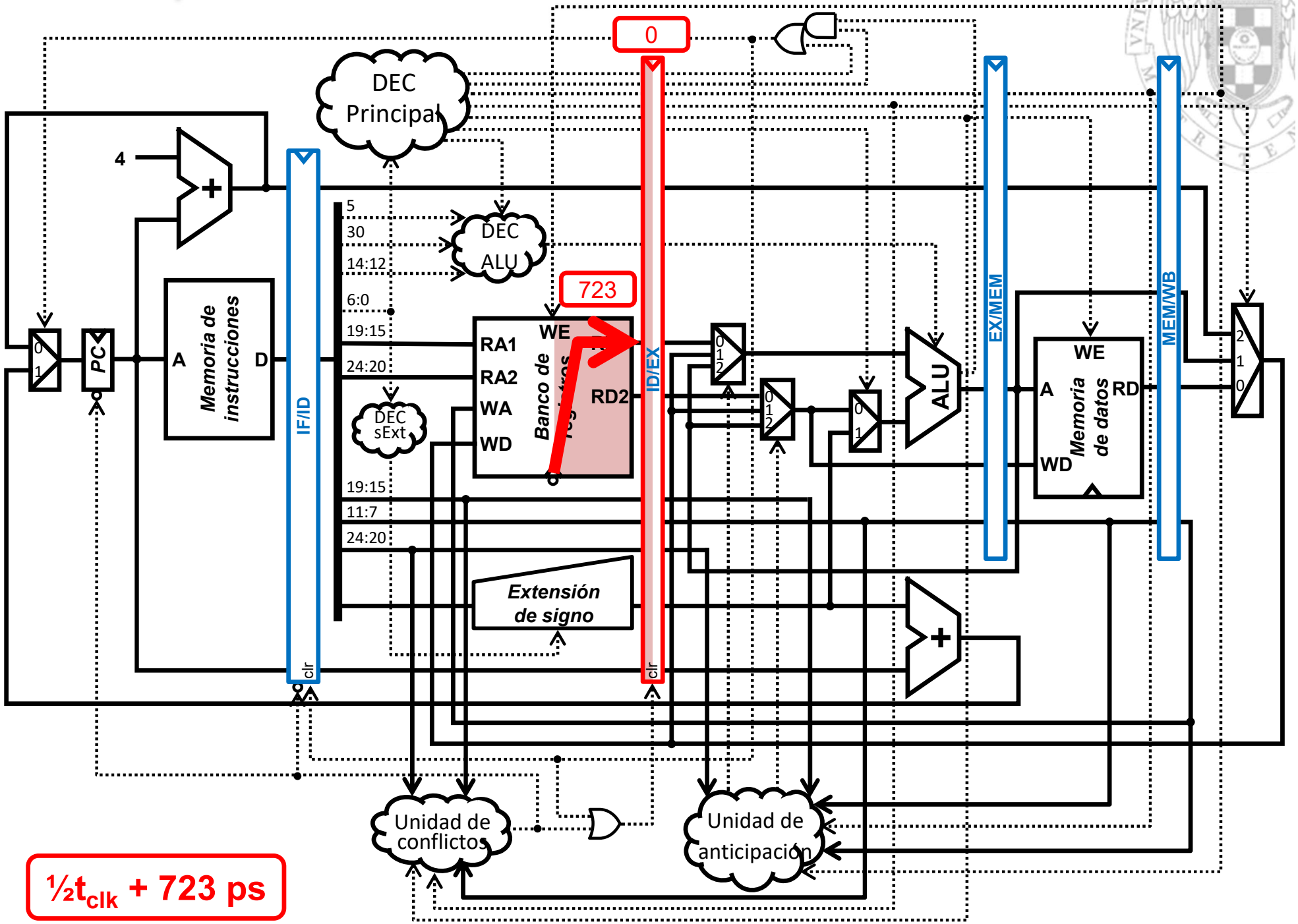
Etapa IF: camino crítico



8890 ps

Cálculo del tiempo de ciclo

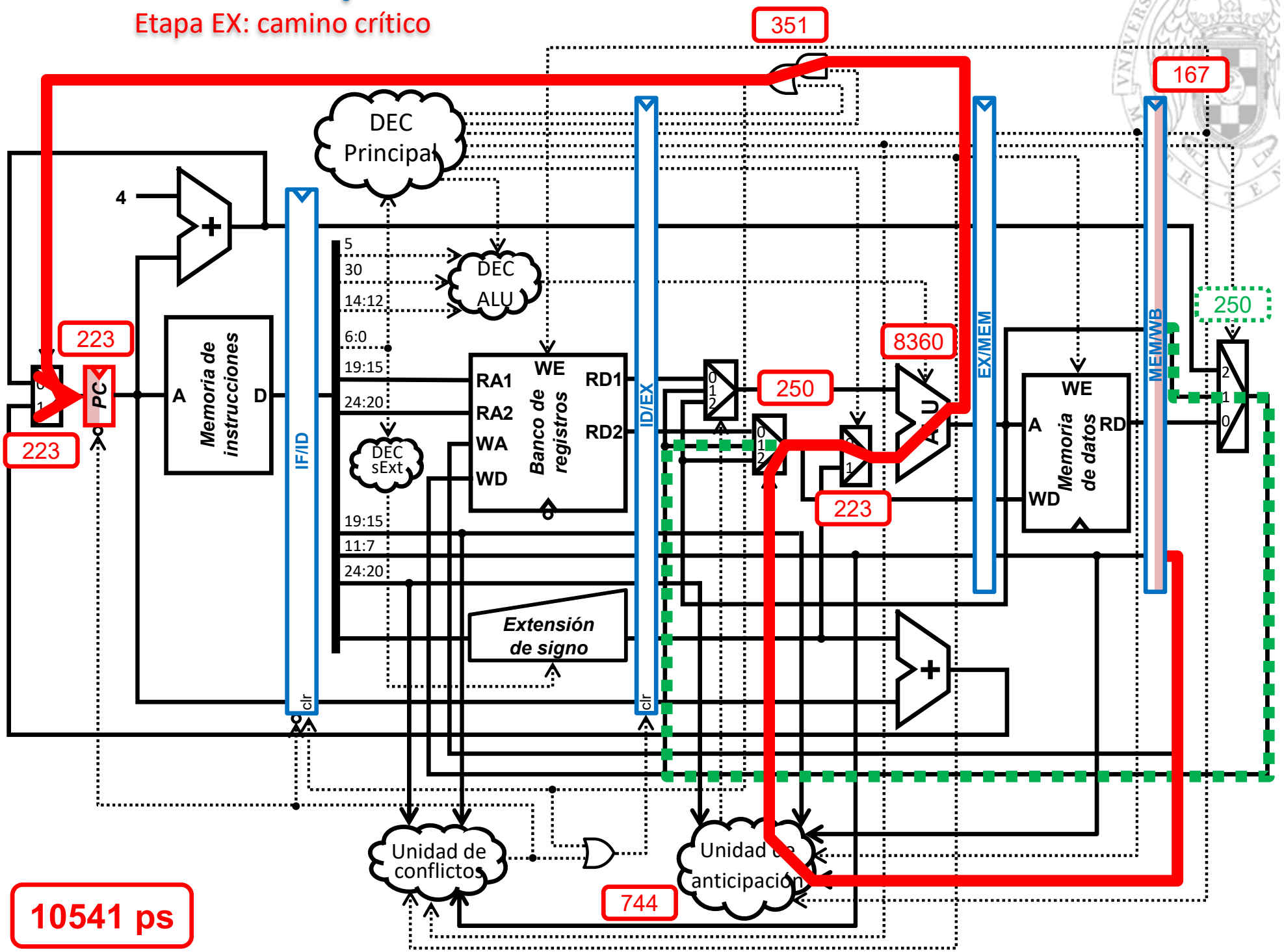
Etapa ID: camino crítico



$\frac{1}{2}t_{clk}$ + 723 ps

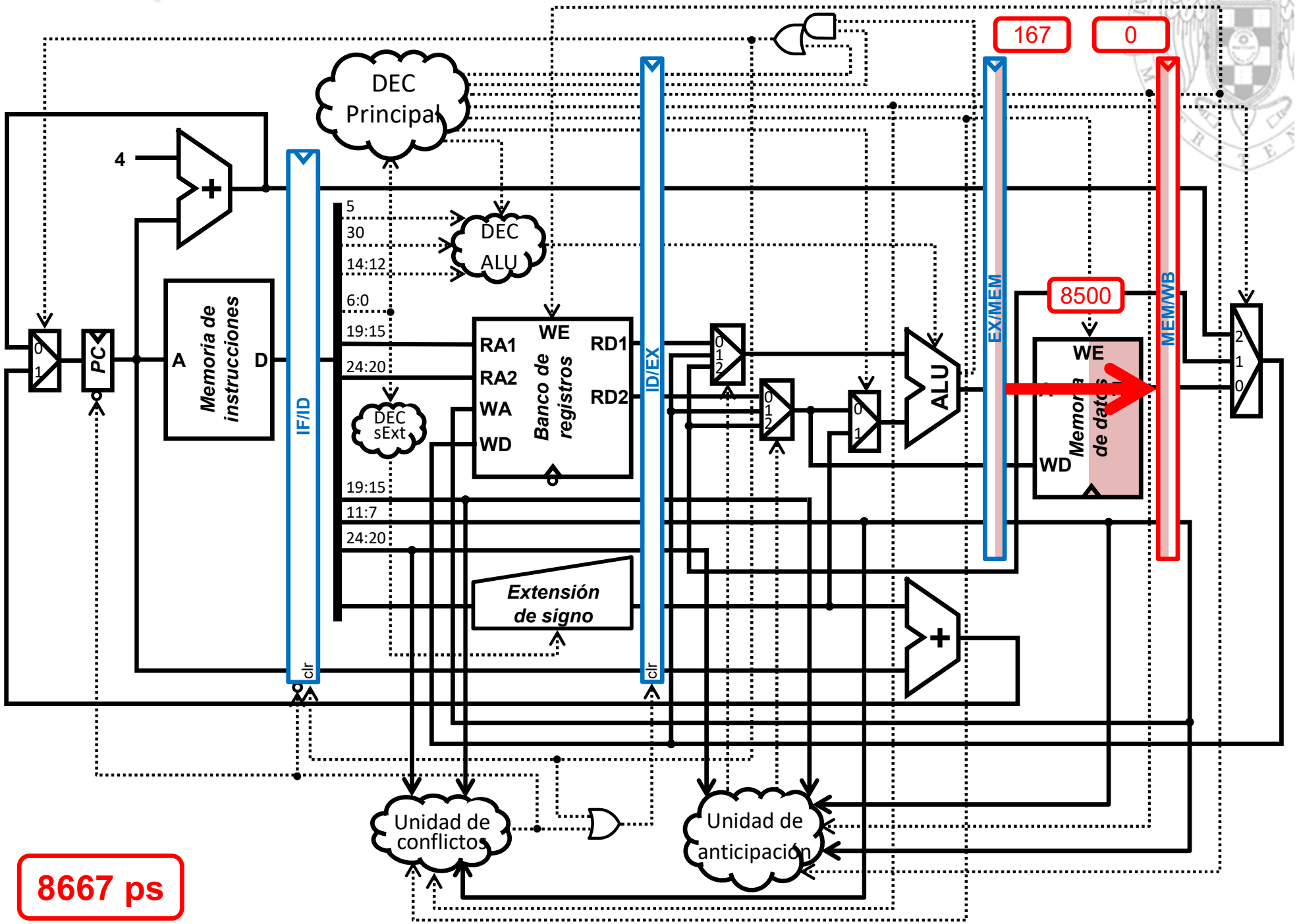
Cálculo del tiempo de ciclo

Etapa EX: camino crítico



Cálculo del tiempo de ciclo

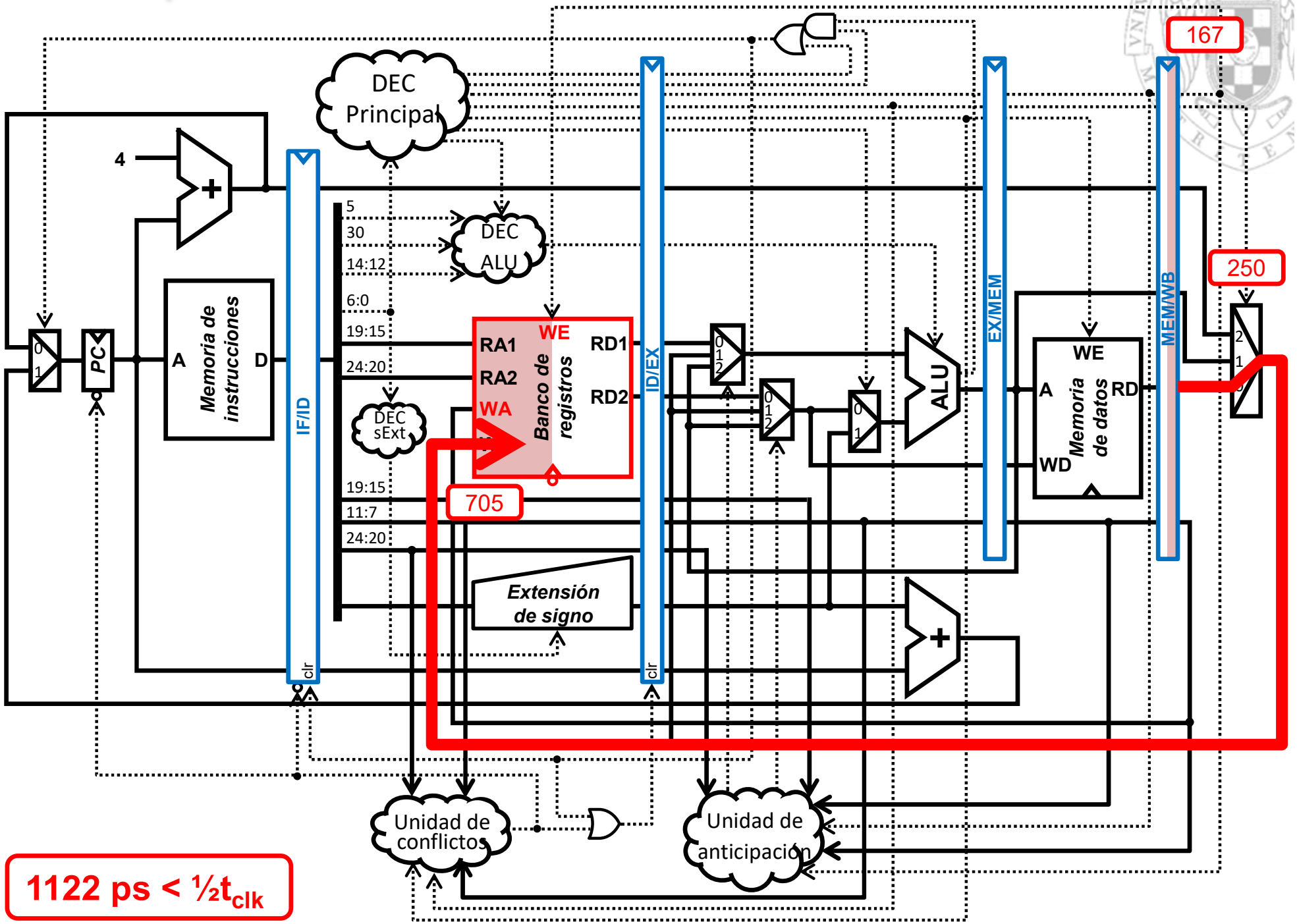
Etapa MEM: camino crítico



8667 ps

Cálculo del tiempo de ciclo

Etapa WB: 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/