Tema 8:

Excepciones

Fundamentos de computadores II

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

Contenidos

- ✓ Introducción.
- ✓ Registros de control y estado.
- ✓ Instrucciones privilegiadas.
- ✓ Rutinas de tratamiento de excepción.
- ✓ Procesador monociclo con gestión de excepciones.
- ✓ Procesador multiciclo con gestión de excepciones.
- ✓ Procesador segmentado con gestión de excepciones.
- ✓ Interrupciones.
- ✓ 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

- ¿Qué sucede si el RISC-V de repertorio reducido ejecuta un programa que por error contiene...
  - ... una instrucción del repertorio completo?
  - ... una instrucción de salto a una dirección de la sección de datos?
  - ... una instrucción de salto a una dirección no múltiplo de 4?
  - ... una instrucción de `lw/sw` de un dato en una dirección no múltiplo de 4?

- El resultado será impredecible ya que, según el caso, lee de memoria...
  - ... un código de instrucción no previsto en el diseño y lo ejecuta.
  - ... un valor mezcla de dos datos y lo procesa.

- Por ello, todos los procesadores implementan mecanismos para:
  - Detectar por HW estos casos.
  - Permitir que el SW realice una acción correctora cuando ocurren.

Introducción

Excepciones vs. interrupciones (i)

- *Trap*: suceso imprevisto que altera la ejecución de un programa.
  - Excepción (*exception*): la causa del suceso es interna al procesador.
    - Es síncrona al programa: ocurre al ejecutar instrucciones del programa.
    - El procesador detecta la ocurrencia del *trap*.
    - Instrucción ilegal, error de alineamiento, etc...
  - Interrupción (*interrupt*): la causa del suceso es externa al procesador.
    - Es asíncrona al programa: ocurre de manera independiente al programa.
    - El procesador tiene una entrada por la que se le indica la ocurrencia del *trap*.
    - Petición de E/S, alarma de sobrecalentamiento, *tick* del RTC, etc...
- Los *traps* se tratan como llamadas a función no programadas:
  - Cuando el procesador lo detecta, salta automáticamente a una dirección predeterminada en donde el programador ha debido ubicar una función SW.
  - La función efectúa una cierta acción y vuelve al programa o lo aborta.
  - Estas funciones típicamente forman parte del sistema operativo:
    - RTE: Rutina de tratamiento de excepción (*exception service-routine handler*)
    - RTI: Rutina de tratamiento de interrupción (*interrupt service-routine/handler*)

Introducción

Excepciones vs. interrupciones (ii)

- Para que la rutina de tratamiento SW pueda realizar la acción apropiada en cada caso, debe conocer el *trap* concreto generado por el HW.

- Existen 2 métodos básicos (modos de servicio) para ello:
  - Directo: el procesador almacena un valor distinto para cada *trap* en un registro "especial" y salta a una única dirección.
    - En dicha dirección se ubica una única rutina común para todos los *traps*.
    - La rutina lee el registro para determinar la acción a realizar: el valor almacenado actúa como parámetro que la rutina.
  - Vectorizado: el procesador salta a una dirección distinta para cada tipo de *trap* generado.
    - Existen tantas rutinas de tratamiento como *traps* diferentes.
    - Cada rutina realiza la acción que corresponde según el *trap* generado.

Registros de control y estado

- Un RISC-V puede disponer de hasta 4096 CSR (Control and Status Registers) de 32 bits para control del sistema.
  - Son distintos de los registros convencionales **`x0`**-**`x31`** del banco de registros.
  - Para operar con ellos, es necesario copiarlos en registros convencionales usando instrucciones privilegiadas.

- Ejemplos de CSR son:

# Reg.	Alias	Descripción
`0xf11`	`mvendorid`	identificador del fabricante*
`0xf12`	`marchid`	identificador de la microarquitectura*
`0xf13`	`mimpid`	identificador de la implementación*
`0x301`	`misa`	repertorio de instrucciones base y extensiones implementadas*
`0xb00`	`mcycle`	número de ciclos transcurridos desde un cierto instante
`0xb02`	`minstret`	número de instrucciones ejecutadas desde un cierto instante

(*) registros de solo lectura

Registros de control y estado

Para gestión de excepciones (i)

- Los CSR principales para la gestión de excepciones son:

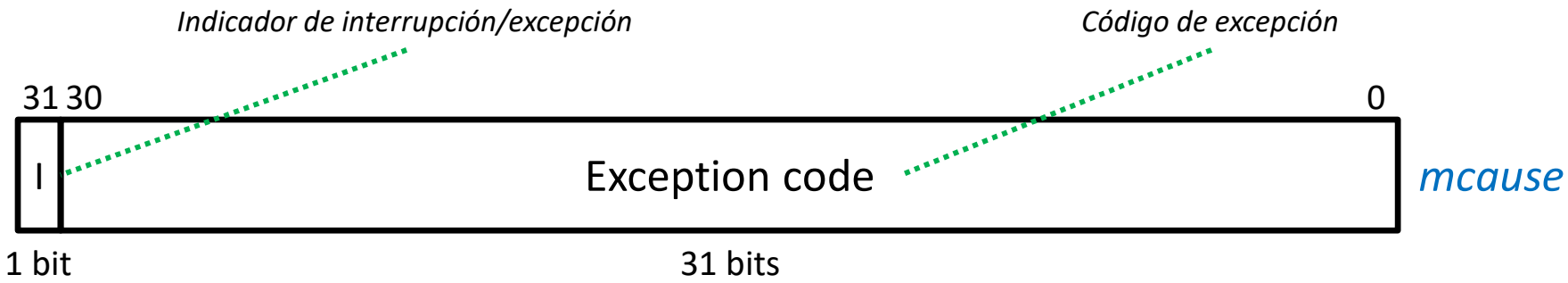
# Reg.	Alias	Descripción
0x305	mtvec	machine trap-vector – dirección base de la rutina de tratamiento
0x342	mcause	machine trap cause – identificador de la causa de la excepción
0x341	mepc	machine exception program counter – dirección de la instrucción que ha provocado la excepción
0x340	mscratch	machine scratch – puntero a una pila auxiliar de la rutina de tratamiento
0x343	mtval	machine trap value – dirección de memoria que ha causado la excepción

- Si en un RISC-V se dispara una excepción, el procesador:
  - Cancela la ejecución de la instrucción en curso (causante de la excepción).
  - Salva la dirección de la instrucción en **mepc**.
  - Si la excepción se produce al acceder a un dato, salva su dirección en **mtval**.
  - Salva un código que identifica la excepción en **mcause**.
  - Salta a la dirección almacenada en **mtvec**.

Registros de control y estado

Para gestión de excepciones (ii)

- El registro **`mcause`** está estructurado en 2 campos:



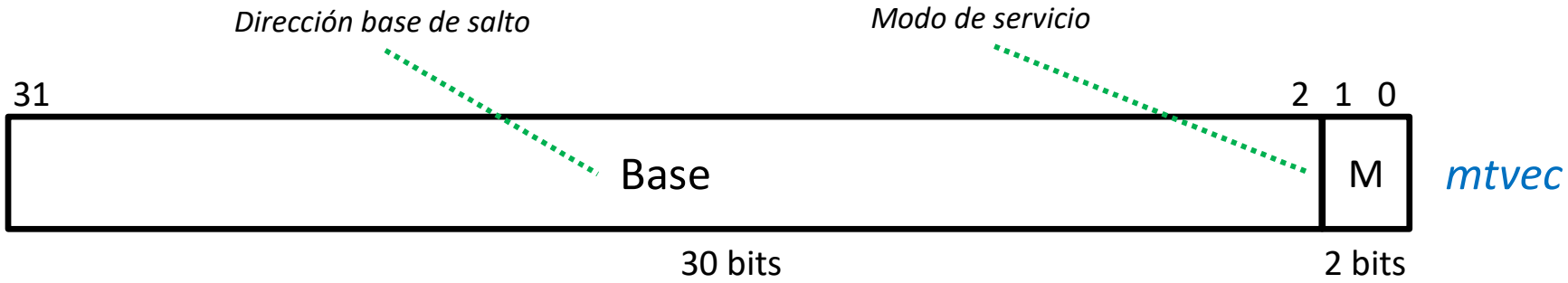
- Indica el tipo de excepción/interrupción generada, por ejemplo:

I	Exception Code	Tipo de trap
0	0x0000 (0)	Dirección de instrucción desalineada
0	0x0002 (2)	Instrucción ilegal
0	0x0004 (4)	Dirección de carga desalineada
0	0x0006 (6)	Dirección de almacenaje desalineada
0	0x000b (11)	Ejecución de la instrucción **`ecall`**
1	0x0007 (7)	Interrupción del temporizador
1	0x000b (11)	Interrupción externa

Registros de control y estado

Para gestión de excepciones (iii)

- El registro **`mtvec`** también está estructurado en 2 campos:



- Define la dirección de salto y el modo de servicio

M	Descripción	Salto realizado si trap
00	Modo directo	*if* (excepción | interrupción) *then* (PC ← PC + Base)
01	Modo vectorizado	*if* (excepción) *then* (PC ← PC + Base) *elsif* (interrupción) *then* (PC ← PC + Base + ($mcause_{30:0}$ << 2))

Instrucciones privilegiadas

Para acceso a CSR

- Permiten copiar un CSR en un registro convencional a la vez que modifican el valor almacenado en el CSR leído.

Instrucción	Operación	Descripción
`csrrw rd, csr, rs1`	rd ← csr csr ← rs1	control and status register read/write intercambia csr y registro
`csrrs rd, csr, rs1`	rd ← csr csr ← csr | rs1	control and status register read/set copia csr y pone bits a 1 usando mascara en registro
`csrrc rd, csr, rs1`	rd ← csr csr ← csr & ~rs1	control and status register read/clear copia csr y pone bits a 0 usando mascara en registro
`csrrwi rd, csr, imm5b`	rd ← csr csr ← zExt(imm)	control and status register read/write immediate copia csr y carga una constante
`csrrsi rd, csr, imm5b`	rd ← csr csr ← csr | zExt(imm)	control and status register read/set immediate copia csr pone bits a 1 usando máscara constante
`csrrci rd, csr, imm5b`	rd ← csr csr ← csr & ~zExt(imm)	control and status register read/clear immediate copia csr pone bits a 0 usando máscara constante

Instrucciones privilegiadas

Otras

Instrucción	Operación	Descripción
`mret`	PC ← mepc	machine return retorna de una excepción
`ecall`	genera excepción 11	enviroment call llamada al sistema operativo

- Un RISC-V puede operar a distintos niveles de privilegio llamados modos.
  - Entre otras cosas, determinan si el procesador puede ejecutar instrucciones privilegiadas y sobre qué subconjunto de CSR.
  - En procesador solo puede estar en un modo en cada instante.
- El máximo nivel de privilegio en un RISC-V es el M-mode: modo máquina.
  - Todos los RISC-V disponen de este modo y es el modo tras *reset*.
  - Según implementación, pueden existir otros:
    - S-mode: Modo supervisor, modo en el que típicamente corre el SO.
    - U-mode: Modo usuario, modo en el que típicamente corren las aplicaciones.
  - El cambio de modo se efectúa cuando se genera un *trap* o se ejecuta una instrucción privilegiada que modifique un cierto CSR (**`mstatus`**).

Instrucciones privilegiadas

Pseudo-instrucciones

- Existen porque es muy común cambiar/leer/escribir un CSR sin necesitar intercambiarlo con un registro convencional.

Instrucción	Operación	Traducción	Descripción
`csrr rd, csr`	rd ← csr	`csrrs rd, csr, x0`	csr read copia csr en registro
`csrw csr, rs1`	csr ←rs1	`csrrw x0, csr, rs1`	csr write copia registro en csr
`csrwi csr, imm5b`	csr ← zExt(imm)	`csrrwi x0, csr, imm`	csr write immediate copia constante en csr
`csrsi csr, imm5b`	csr ← csr | zExt(imm)	`csrrwi x0, csr, imm`	csr set immediate pone a 1 bits del csr
`csrci csr, imm5b`	csr ← csr & ~zExt(imm)	`csrrwi x0, csr, imm`	csr clear immediate pone a 0 bits del csr

Instrucciones privilegiadas

Formatos de instrucción (i)

- Todas las instrucciones privilegiadas son de formato tipo I y tienen el mismo código de operación:

31 — 20	19 — 15	14 — 12	11 — 7	6 — 0	
$imm_{11:0}$	rs1	funct3	rd	op	*tipo-I*
000000000000	00000	000	00000	1110011	`ecall`
001100000010	00000	000	00000	1110011	`mret`
csr	rs1	fucnt3	rd	1110011	`csrrX`
csr	imm	funct3	rd	1110011	`csrrXi`

Instrucciones privilegiadas

Formatos de instrucción (ii)

- Para distinguirlas, tienen diferente código de función:

Instrucciones privilegiadas

op	funct3	Instrucción	Tipo
1110011	001	`csrrw`	I
	010	`csrrs`	I
	011	`csrrc`	I
	101	`csrrwi`	I
	110	`csrrsi`	I
	111	`csrrci`	I

Rutinas de tratamiento de excepción

- El procesador, cuando detecta una excepción, salta a la rutina de tratamiento de forma automática.
  - El programa en ejecución es interrumpido sin su conocimiento.

- Por ello, la rutina de tratamiento de excepción debe:
  - Apilar todos los registros que use en la región de memoria apuntada por el CSR **`mscratch`**.
    - Dado que es una situación imprevista, deberá apilar tanto los registros preservados como los temporales.
    - Si llama a otras funciones, deberá apilar también el registro **`ra`**.
    - Usando una región de memoria diferente para no alterar la pila del programa.
  - Leer **`mcause`** para conocer la excepción que se ha producido.
  - Actuar en consecuencia.
- Si la excepción permite continuar el programa deberá, además:
  - Desapilar los registros salvados.
  - Volver al programa interrumpido ejecutando **`mret`**.

Rutinas de tratamiento de excepción

Ejemplo

ASM

```
rte:
 csrrw  t0, mscratch, t0
 add    t0, t0, -8
 sw     t1, 4(t0)
 sw     t2, 0(t0)
 csrr   t1, mcause
 li     t2, 2
 bne    t1, t2, other
illegalop:
 csrr   t2, mepc
 addi   t2, t2, 4
 csrw   mepc, t2
 j      done
other:
 j      .
done:
 lw     t1, 4(t0)
 lw     t2, 0(t0)
 add    t0, t0, 8
 csrrw  t0, mscratch, t0
 mret
```

Intercambia `t0` *y* `mscratch`

Apila contexto

Chequea si la causa de excepción fue instrucción ilegal (código 2)

Si fue esa la causa: salta la instrucción modificando `mepc`

Si fue otra causa: aborta el programa

Desapila contexto

Intercambia `t0` *y* `mscratch`

Retorna al programa

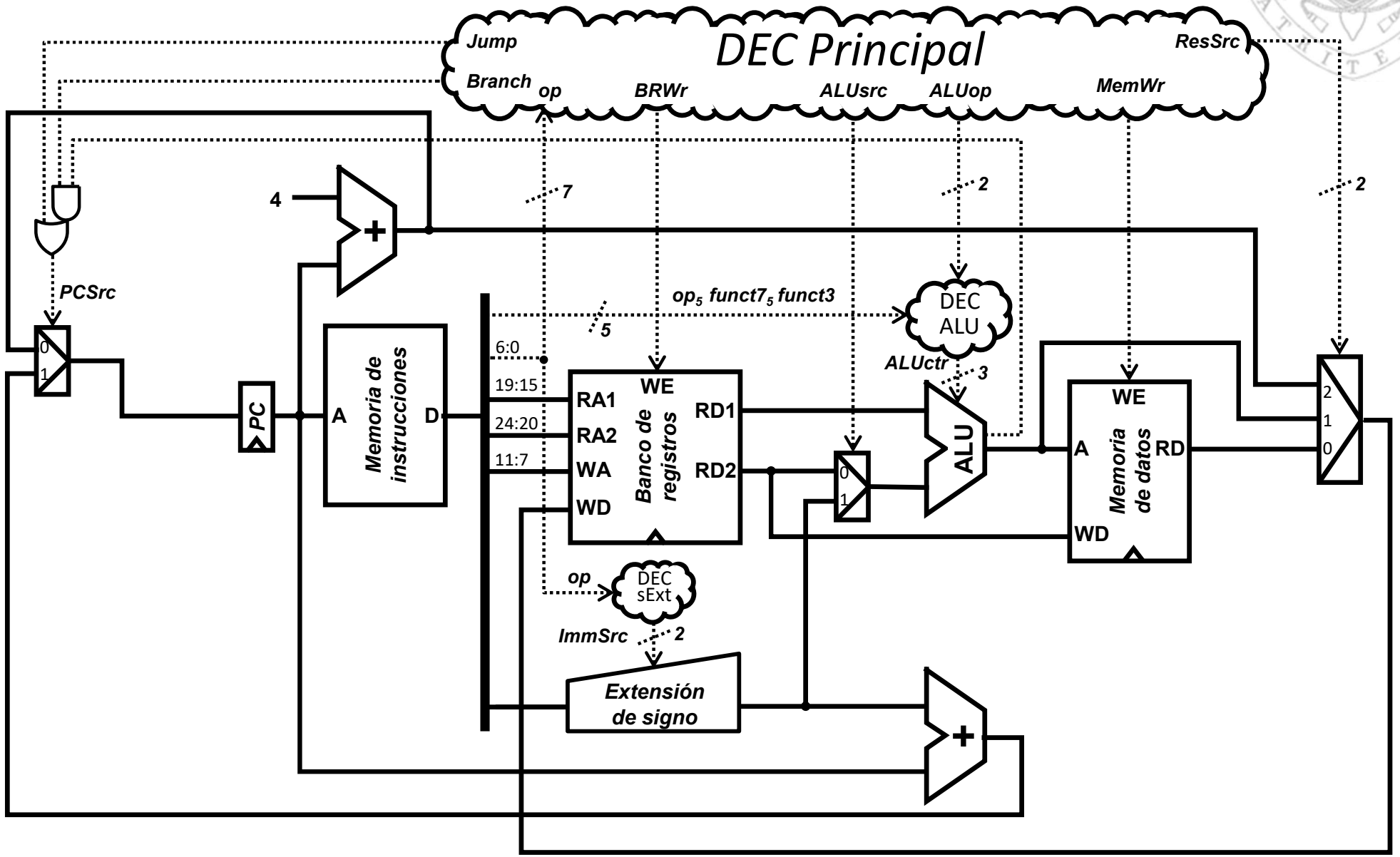
RISC-V de arquitectura reducida

Con gestión elemental de excepciones

- Se ampliarán las microarquitecturas RISC-V de repertorio reducido para gestionar las excepciones siguientes:
  - Acceso no alineado a memoria de instrucciones (causa 0).
  - Instrucción ilegal por código de operación desconocido (causa 2).
    - Por ejemplo, si ejecuta: **`lui`**, **`auipc`**, **`jalr`**, **`mul`**…
  - Instrucción ilegal por operación aritmético-lógica no implementada (causa 2).
    - Por ejemplo, si ejecuta: **`xor`**, **`sll`**, **`sra`**, **`xori`**…
  - Acceso no alineado a memoria de datos (lectura: causa 4, escritura: causa 6).
- Se añadirán 3 CSR con funcionalidad reducida:
  - **`mepc`**
  - **`mcause`** que solo podrá leerse.
  - **`mtvec`** que tendrá valor fijo **`0x1c000000`** y solo podrá leerse.
- Se añadirán 2 instrucciones privilegiadas con funcionalidad reducida:
  - **`csrrw`** limitado a los CSR **`mepc`** y **`mcause`**
  - **`mret`**

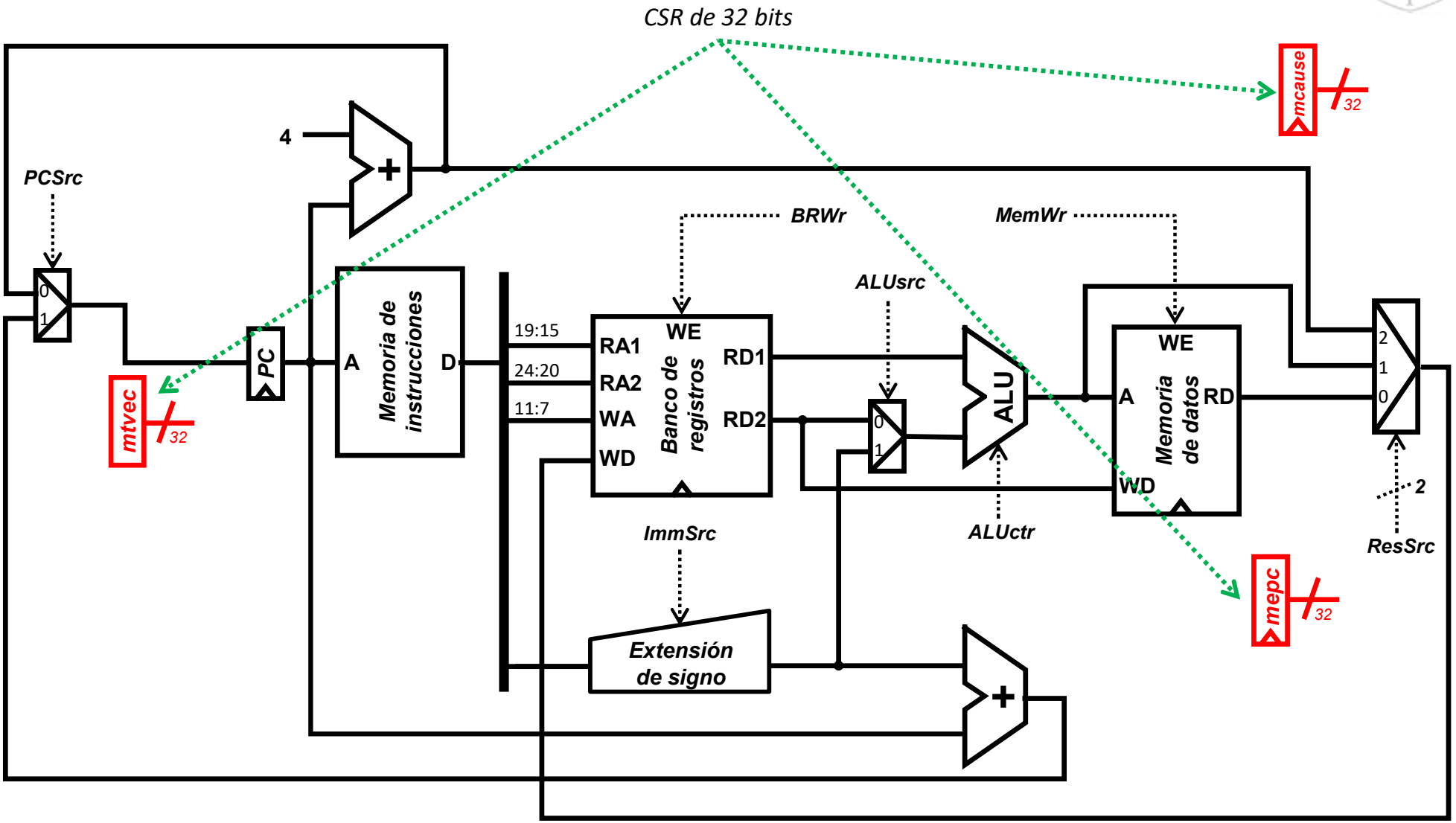
Procesador monociclo

Ruta de datos + controlador originales



Procesador monociclo

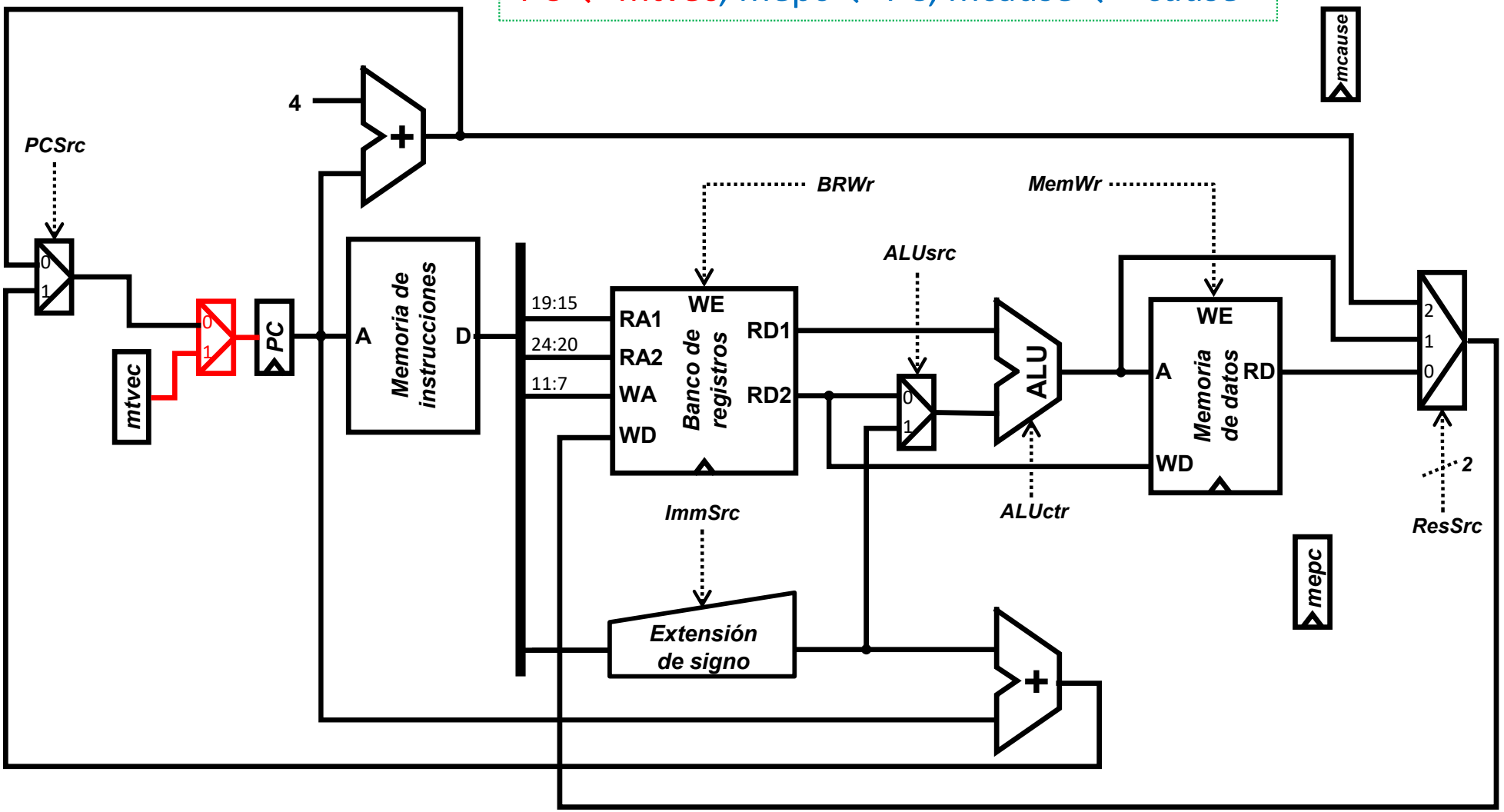
Ruta de datos + gestión de excepciones (i)



Procesador monociclo

Ruta de datos + gestión de excepciones (ii)

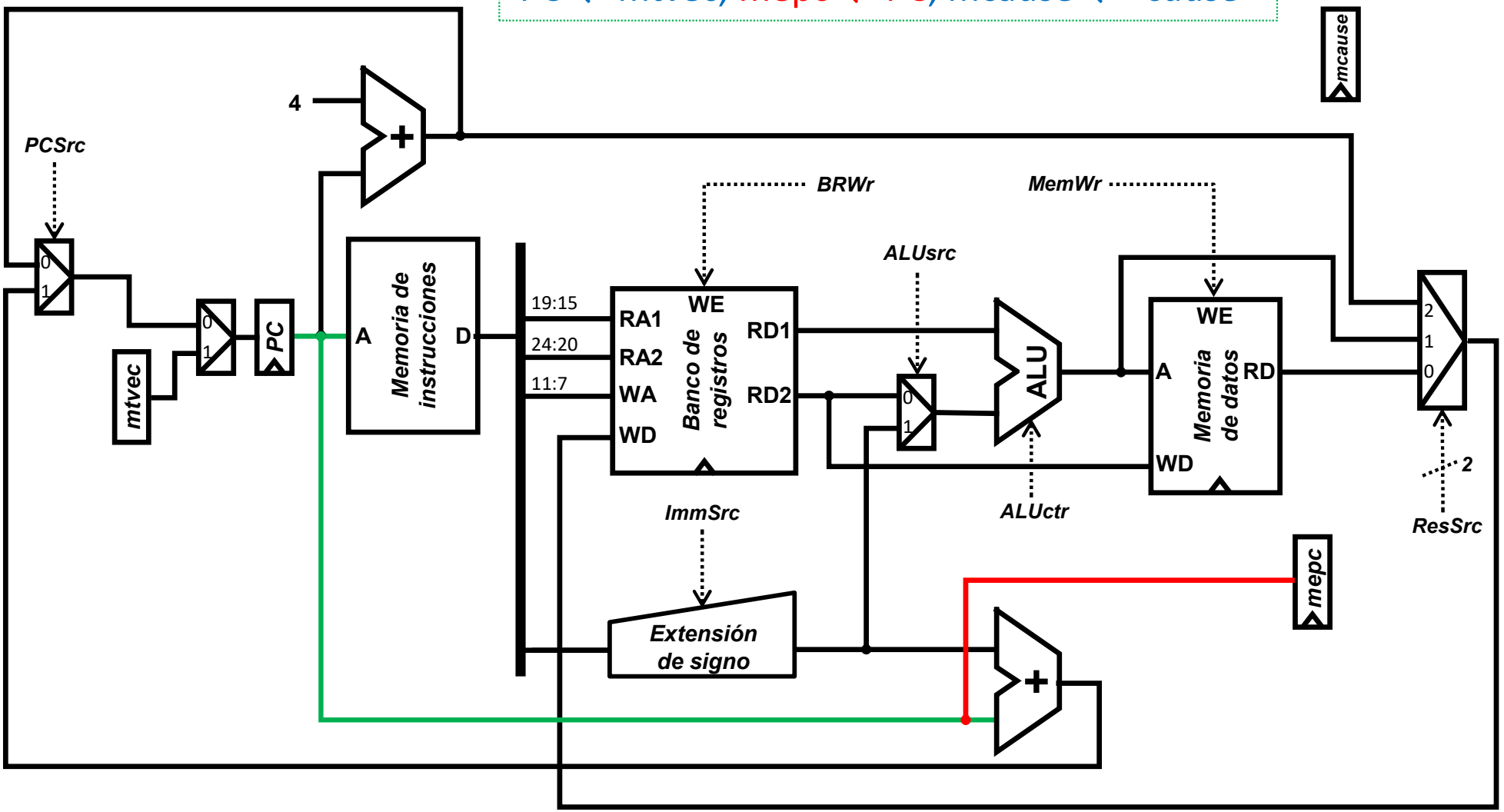
si excepción:
PC ← mtvec, mepc ← PC, mcause ← *"cause"*



Procesador monociclo

Ruta de datos + gestión de excepciones (ii)

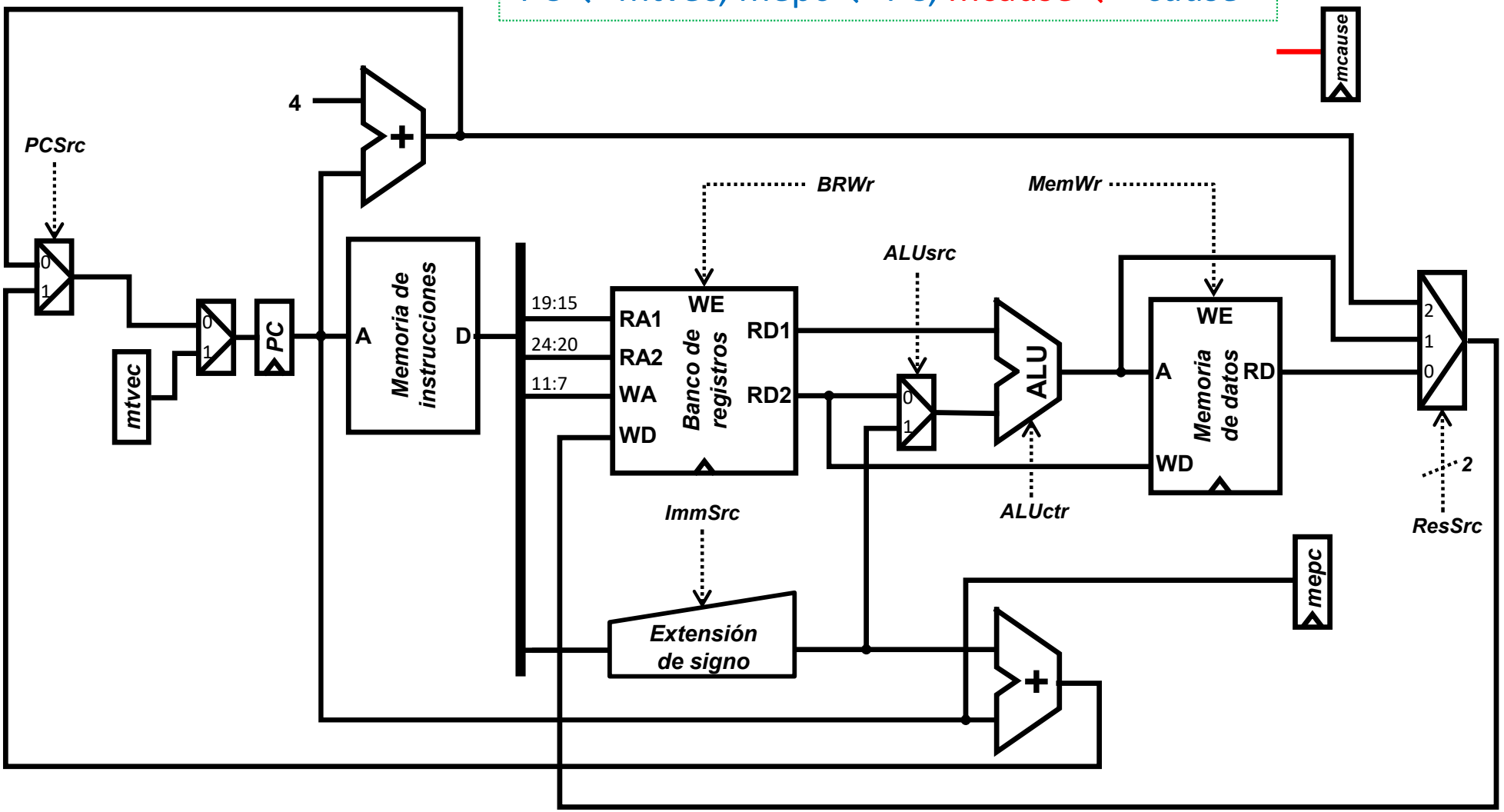
si excepción:
PC ← mtvec, mepc ← PC, mcause ← *"cause"*



Procesador monociclo

Ruta de datos + gestión de excepciones (ii)

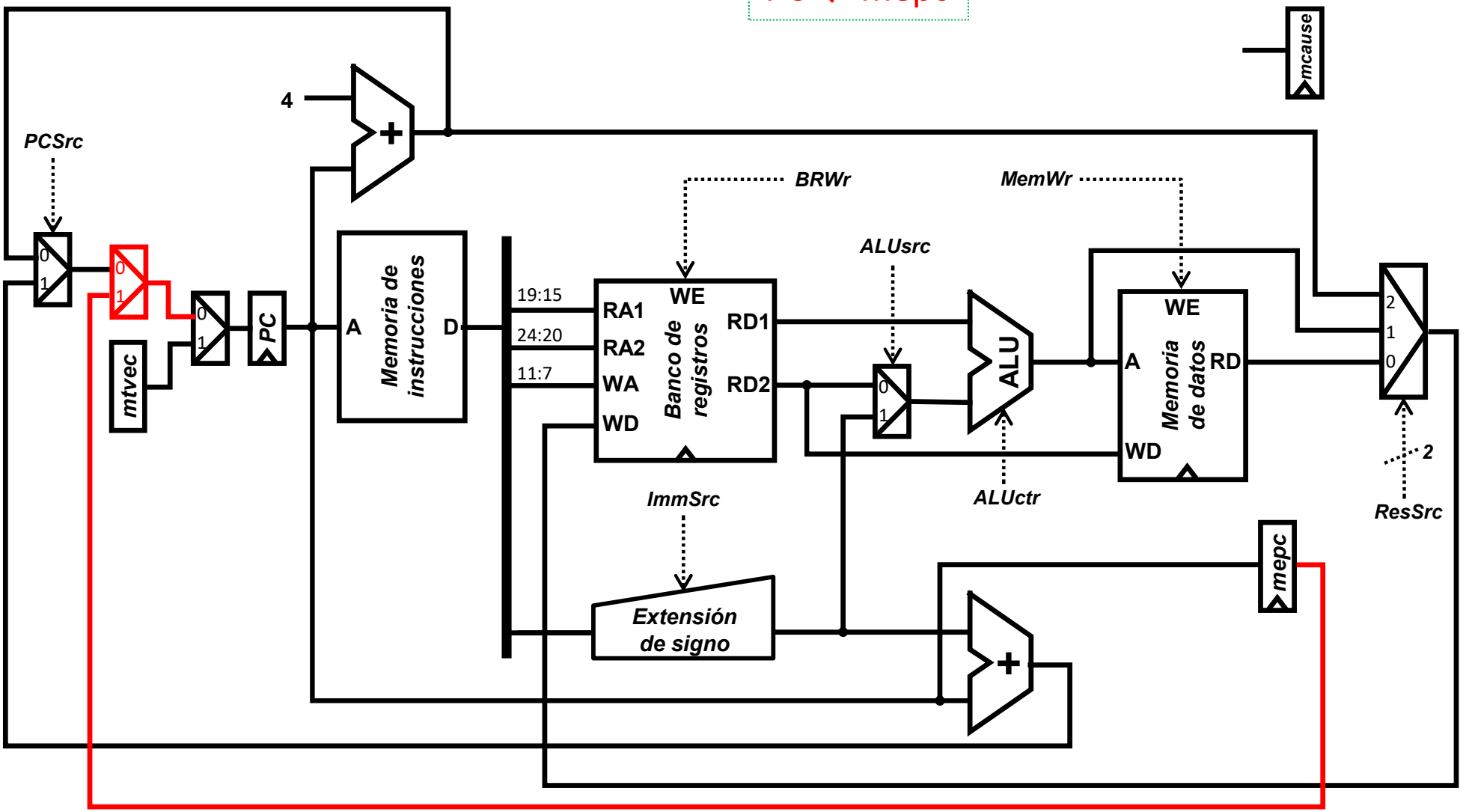
si excepción:
PC ← mtvec, mepc ← PC, mcause ← "*cause*"



Procesador monociclo

Ruta de datos + gestión de excepciones (iii)

mret
PC ← mepc

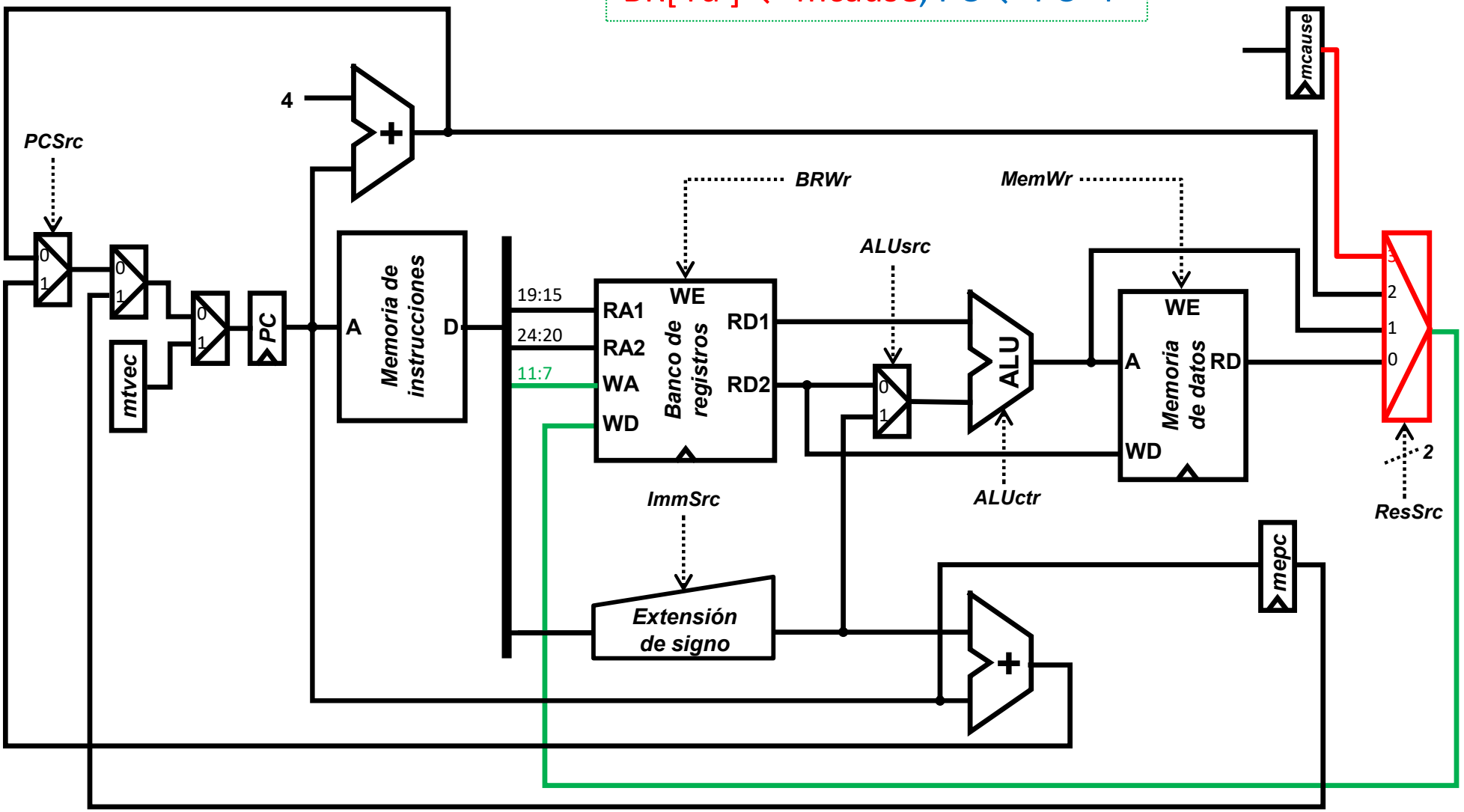


Procesador monociclo

Ruta de datos + gestión de excepciones (iv)

```
csrrw rd, mcause, rs1
```

BR[rd] ← mcause, PC ← PC+4

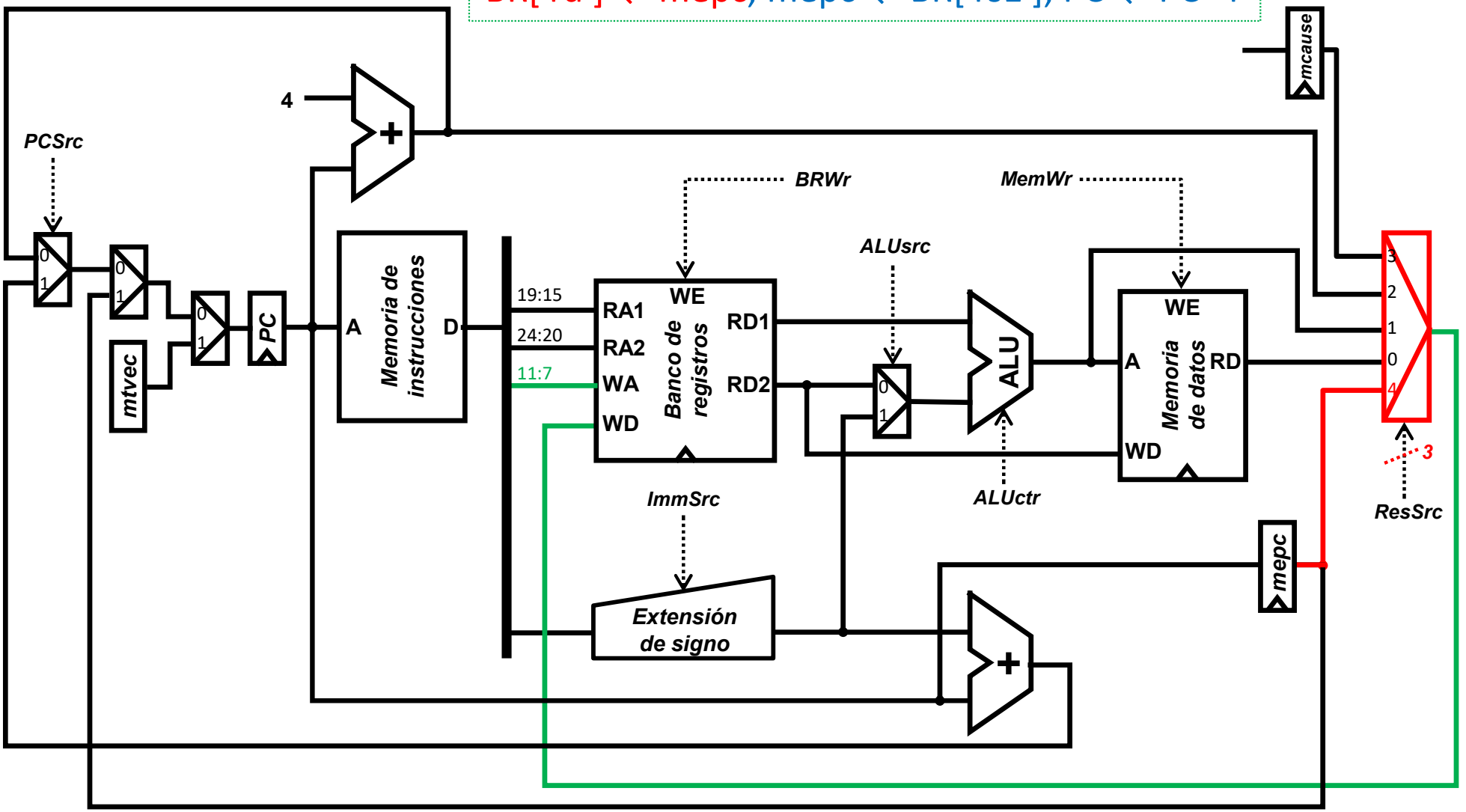


Procesador monociclo

Ruta de datos + gestión de excepciones (v)

```
csrrw rd, mepc, rs1
```

BR[rd] ← mepc, mepc ← BR[rs1], PC ← PC+4

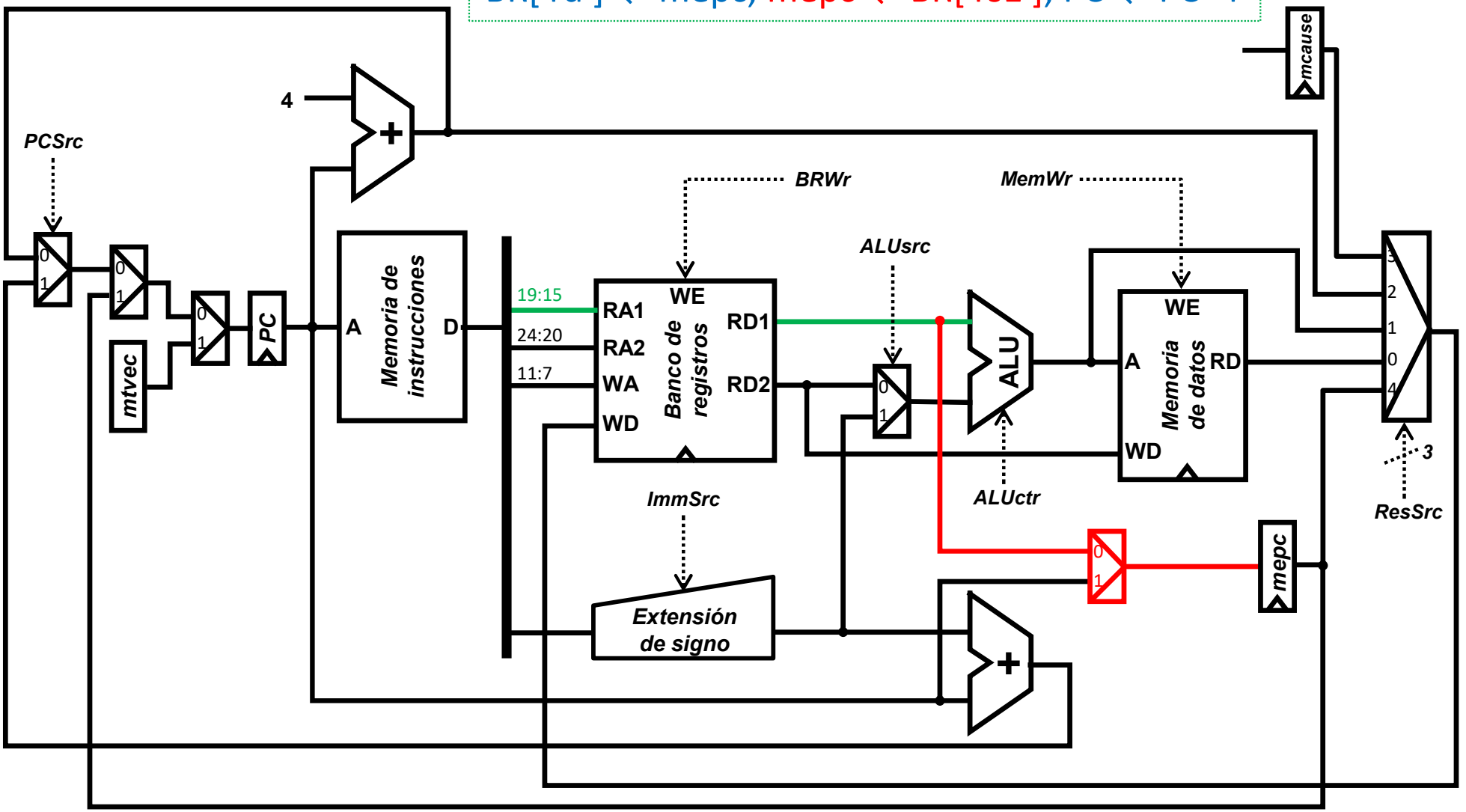


Procesador monociclo

Ruta de datos + gestión de excepciones (v)

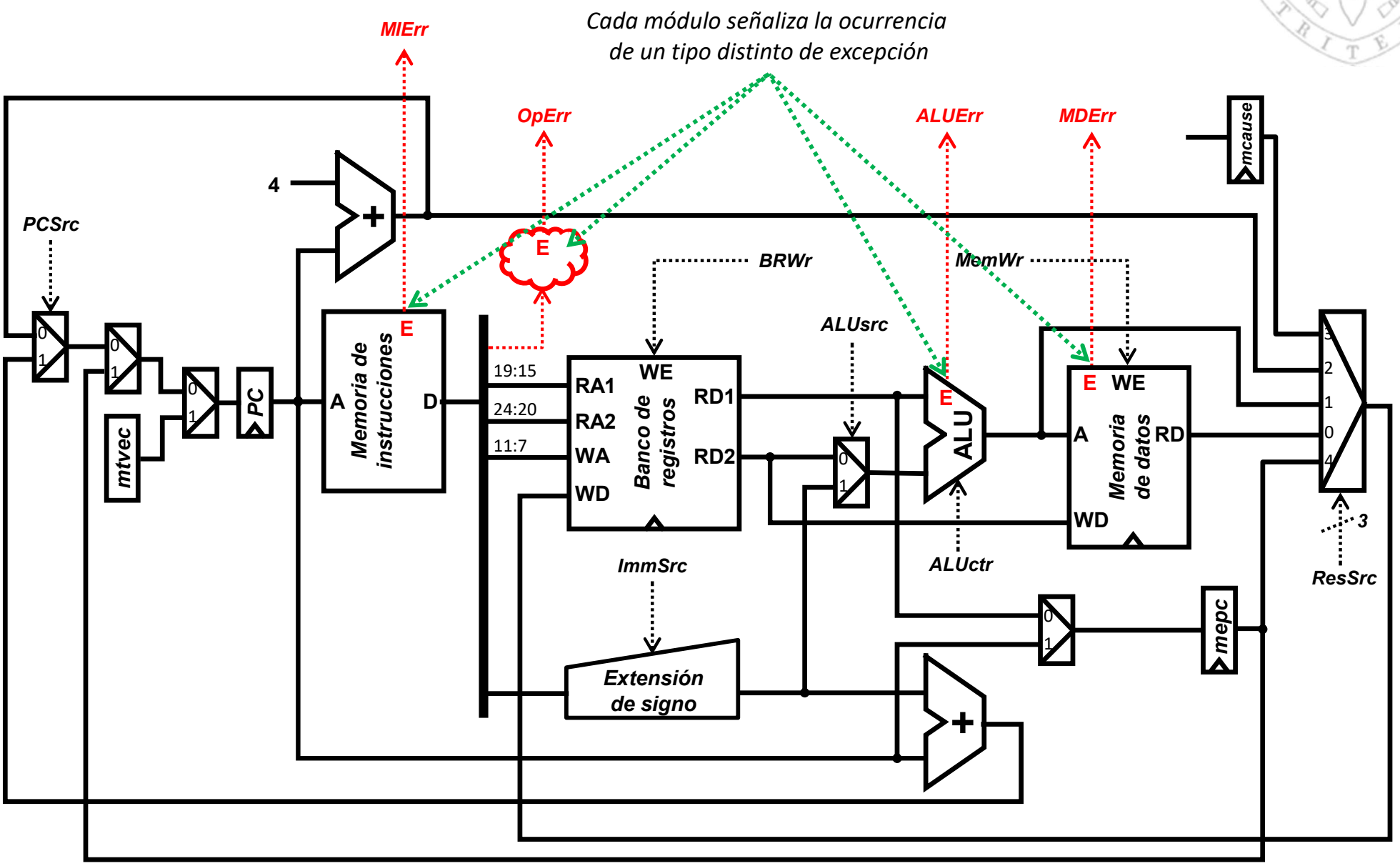
```
csrrw rd, mepc, rs1
```

BR[rd] ← mepc, mepc ← BR[rs1], PC ← PC+4



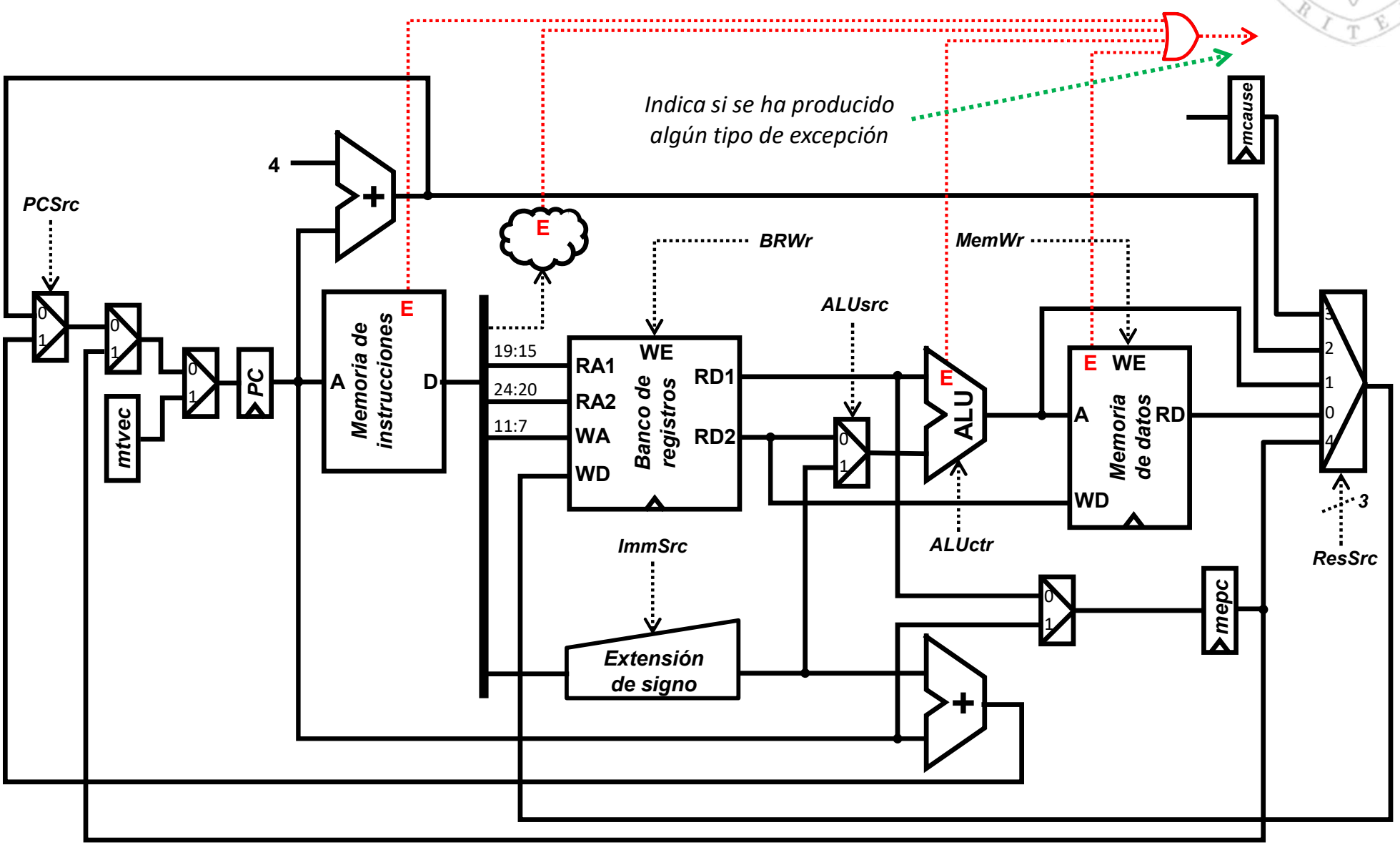
Procesador monociclo

Controlador de excepciones (i)



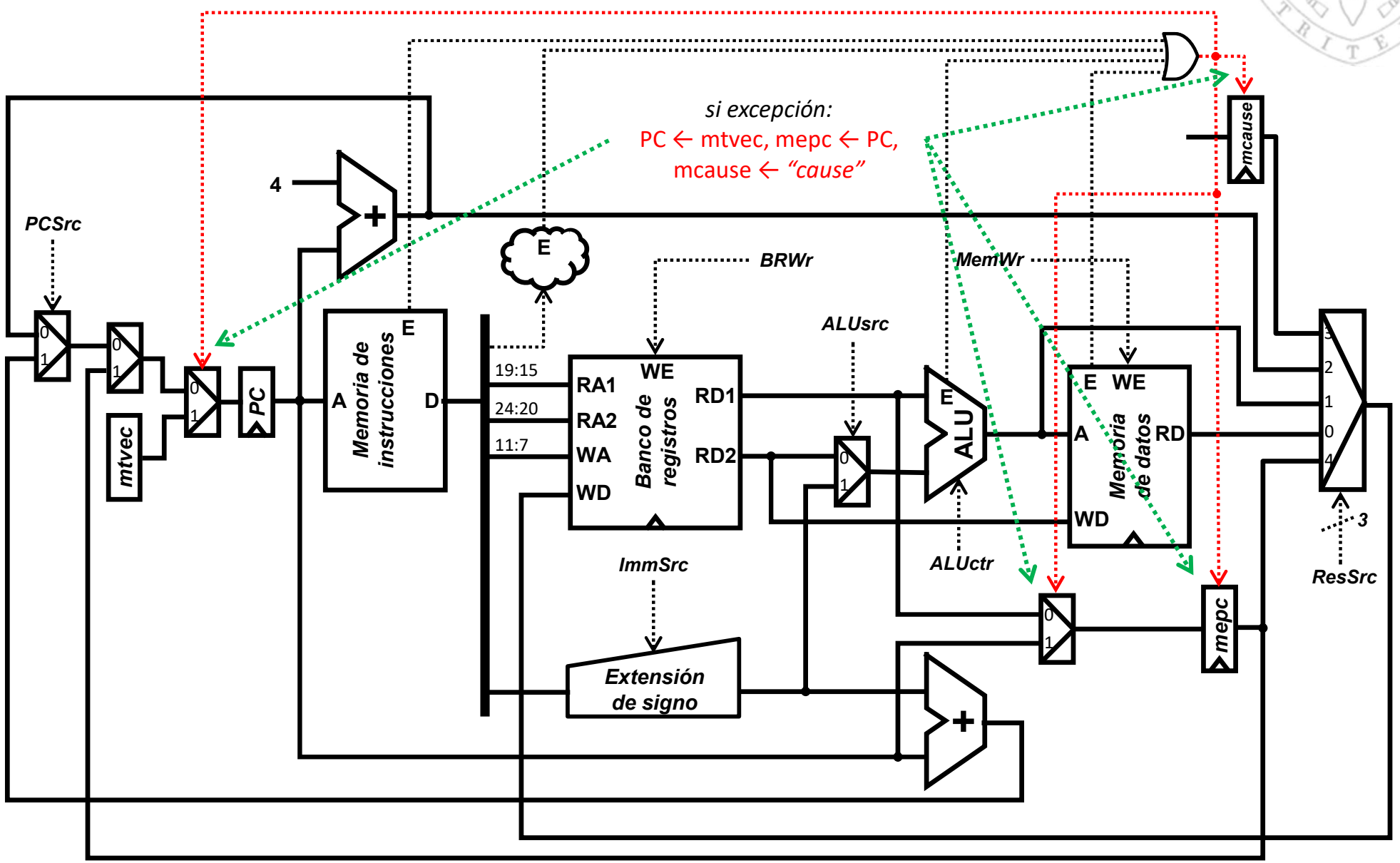
Procesador monociclo

Controlador de excepciones (ii)



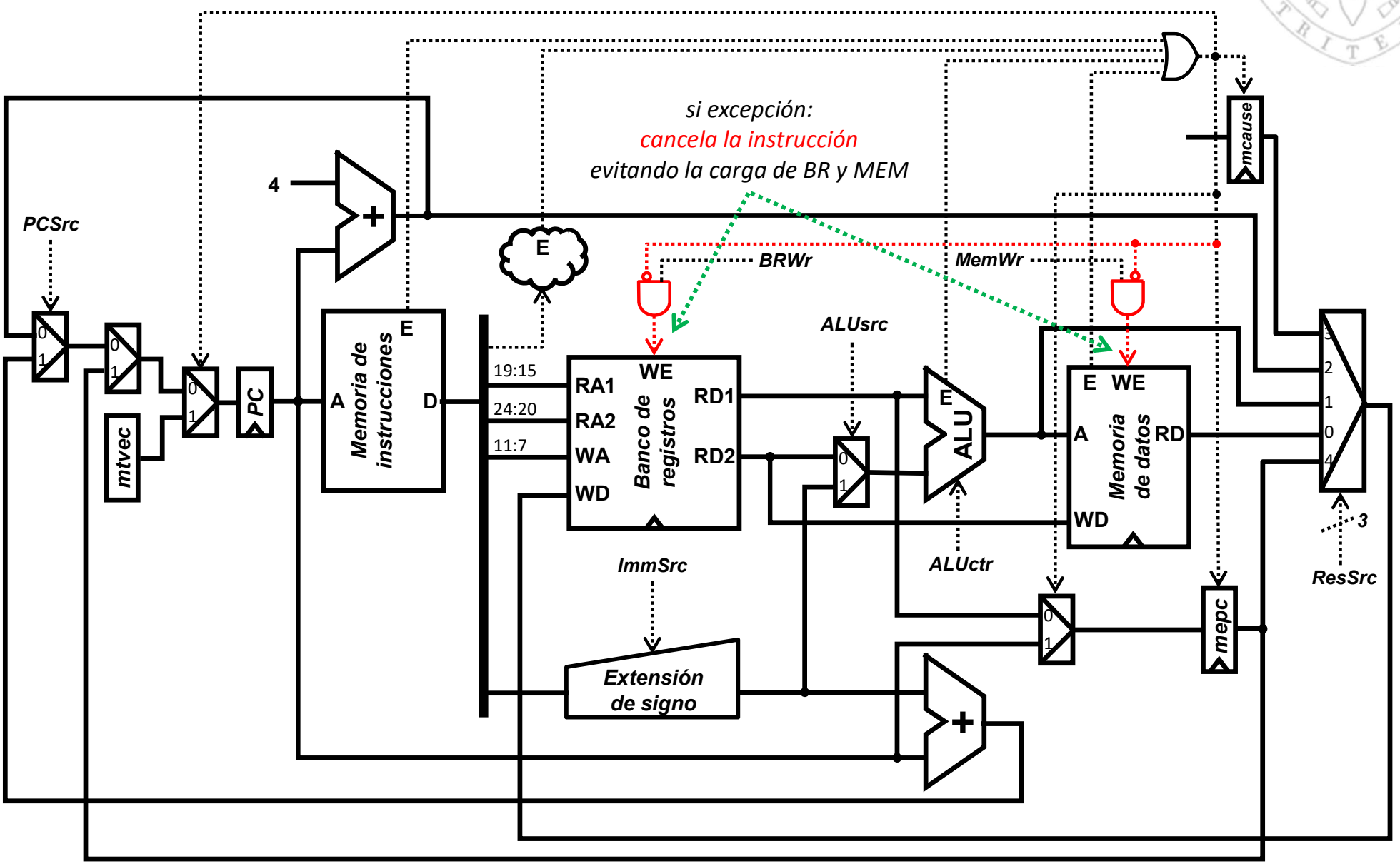
Procesador monociclo

Controlador de excepciones (iii)



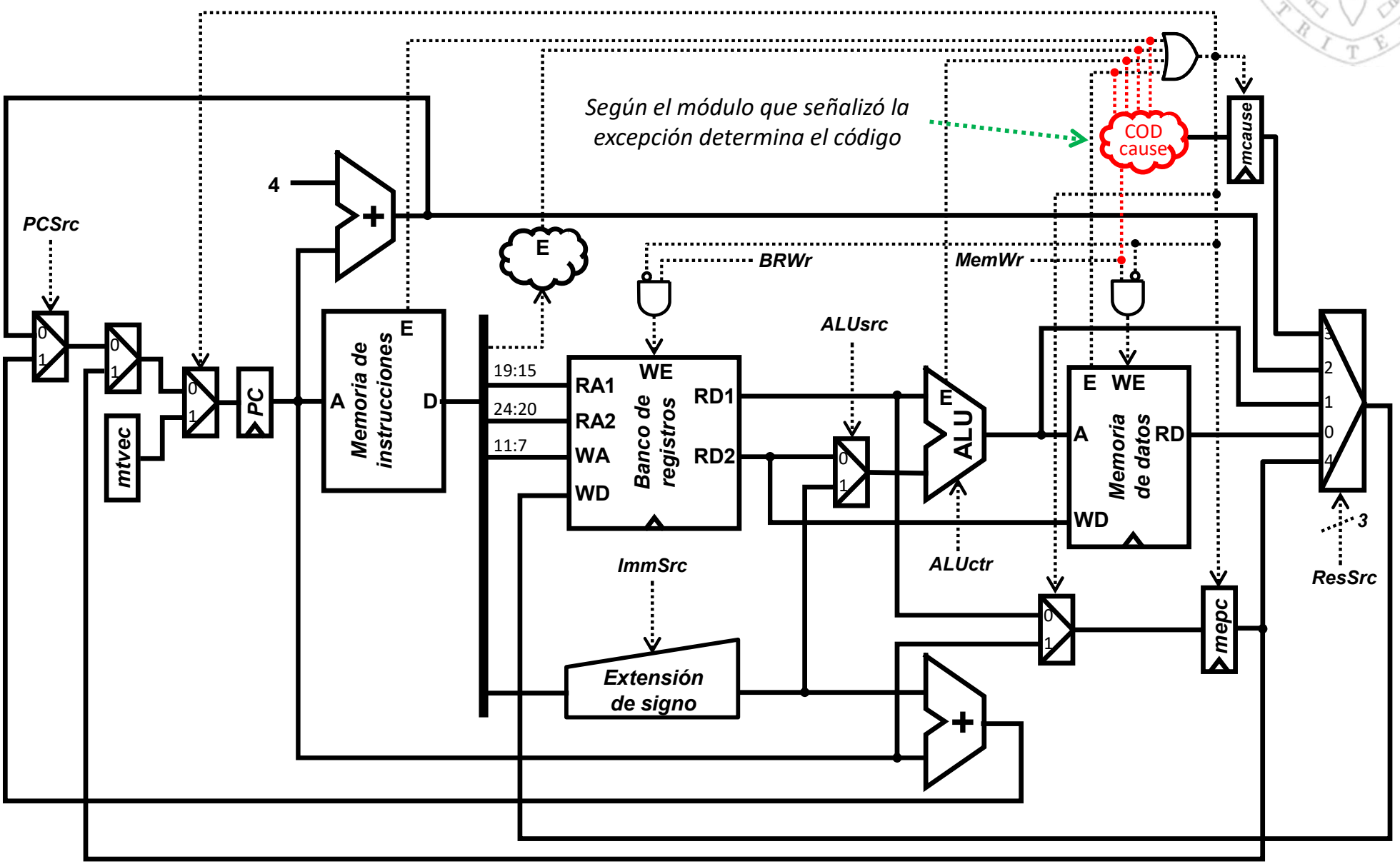
Procesador monociclo

Controlador de excepciones (iv)



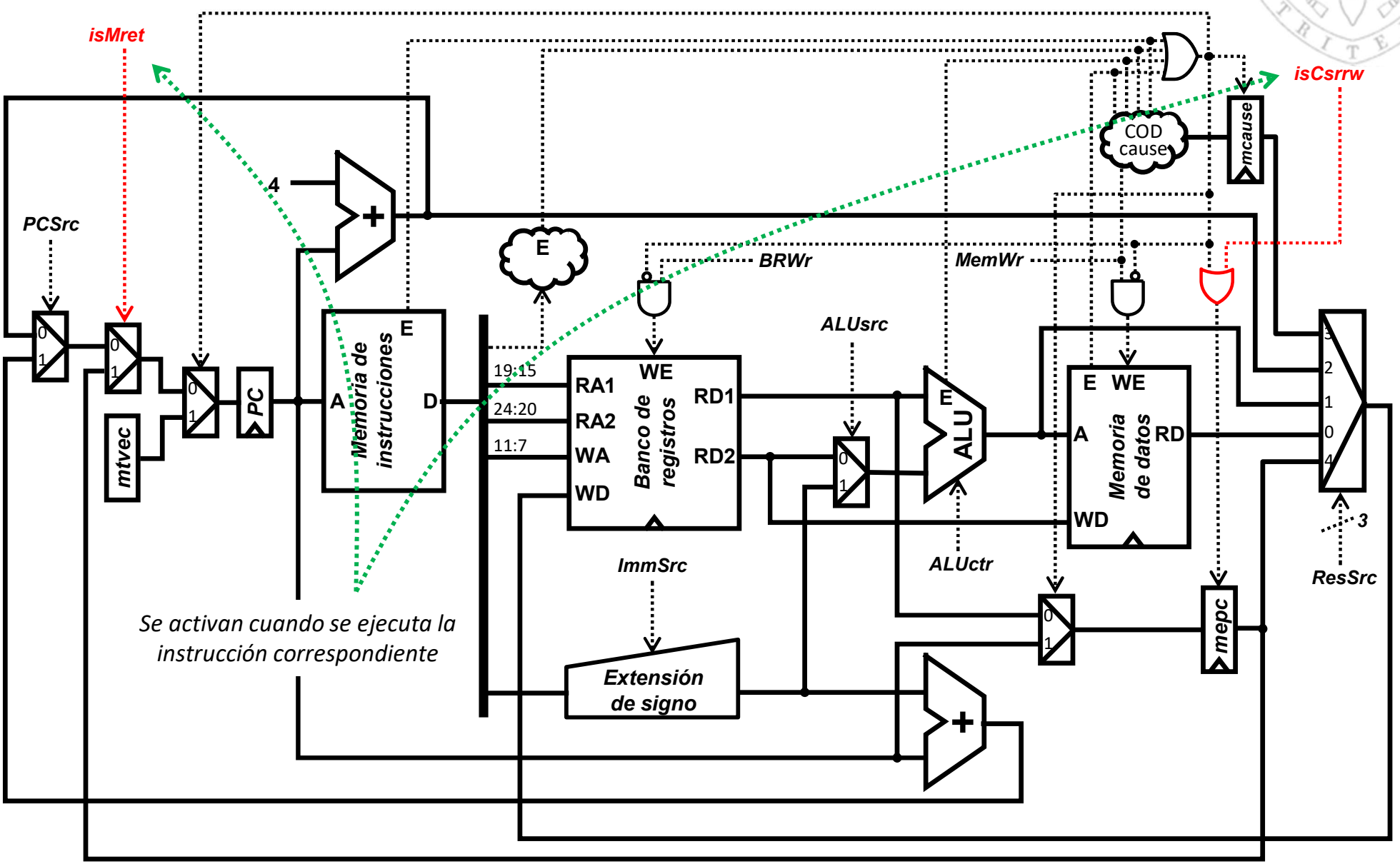
Procesador monociclo

Controlador de excepciones (v)



Procesador monociclo

Controlador de excepciones (v)



Procesador monociclo

Controlador de excepciones: COD de causa

- Este subcircuito codifica la causa de la excepción.
  - Estableciendo, además, una prioridad entre ellas en caso de que una misma instrucción genere varias.

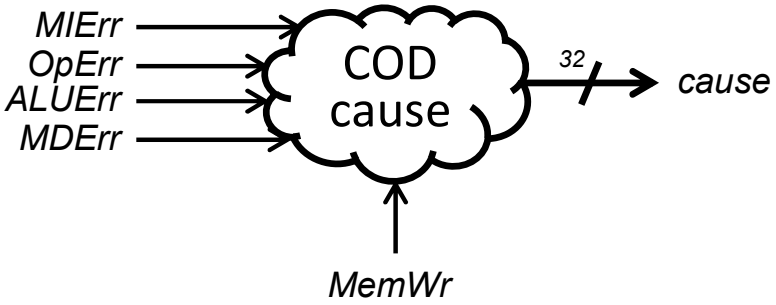
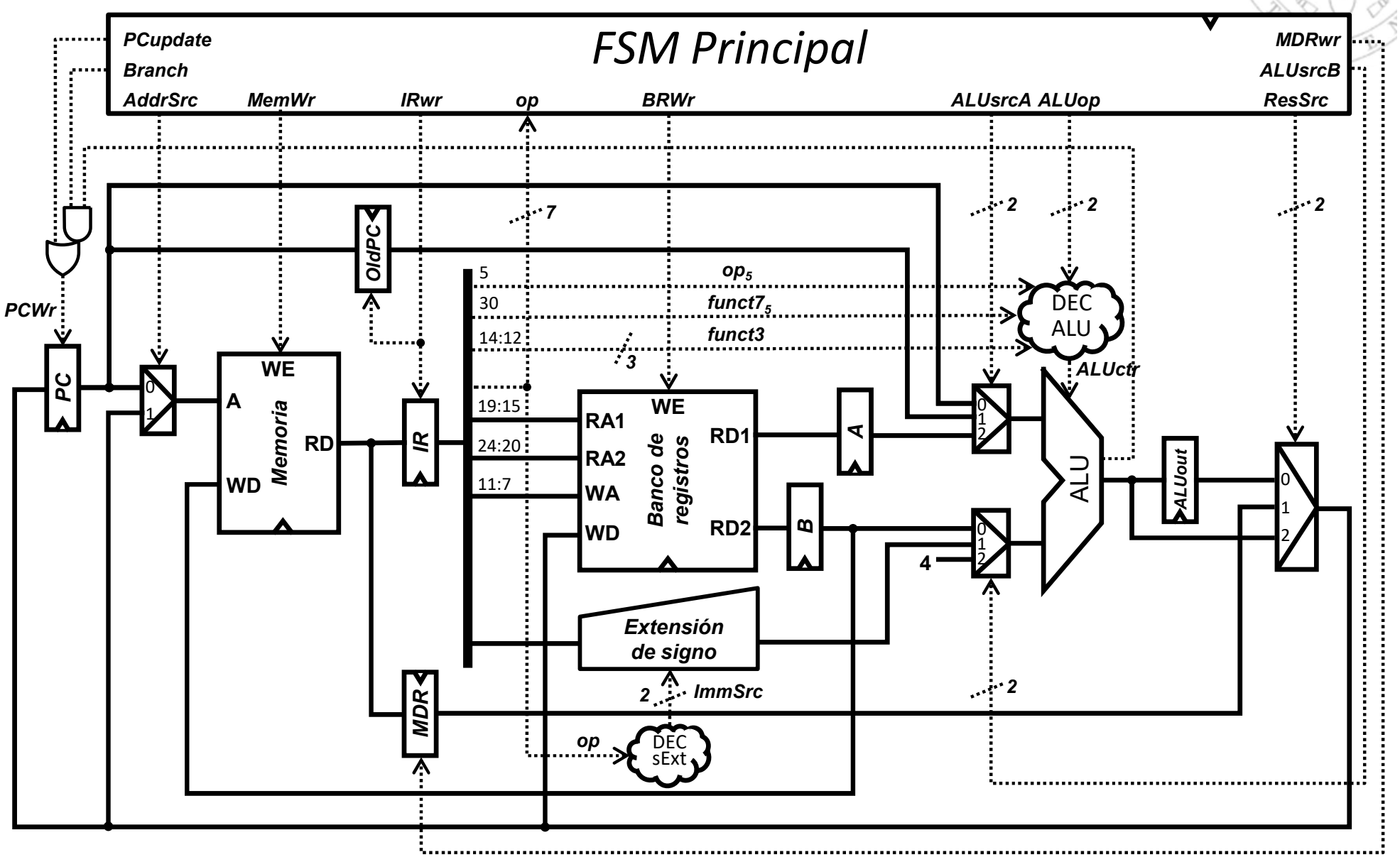


Tabla de verdad

MIErr	OpErr	ALUErr	MDErr	MemWr	cause
1	X	X	X	X	0x0000
0	1	X	X	X	0x0002
0	0	1	X	X	0x0002
0	0	0	1	0	0x0004
0	0	0	1	1	0x0006
0	0	0	0	0	–

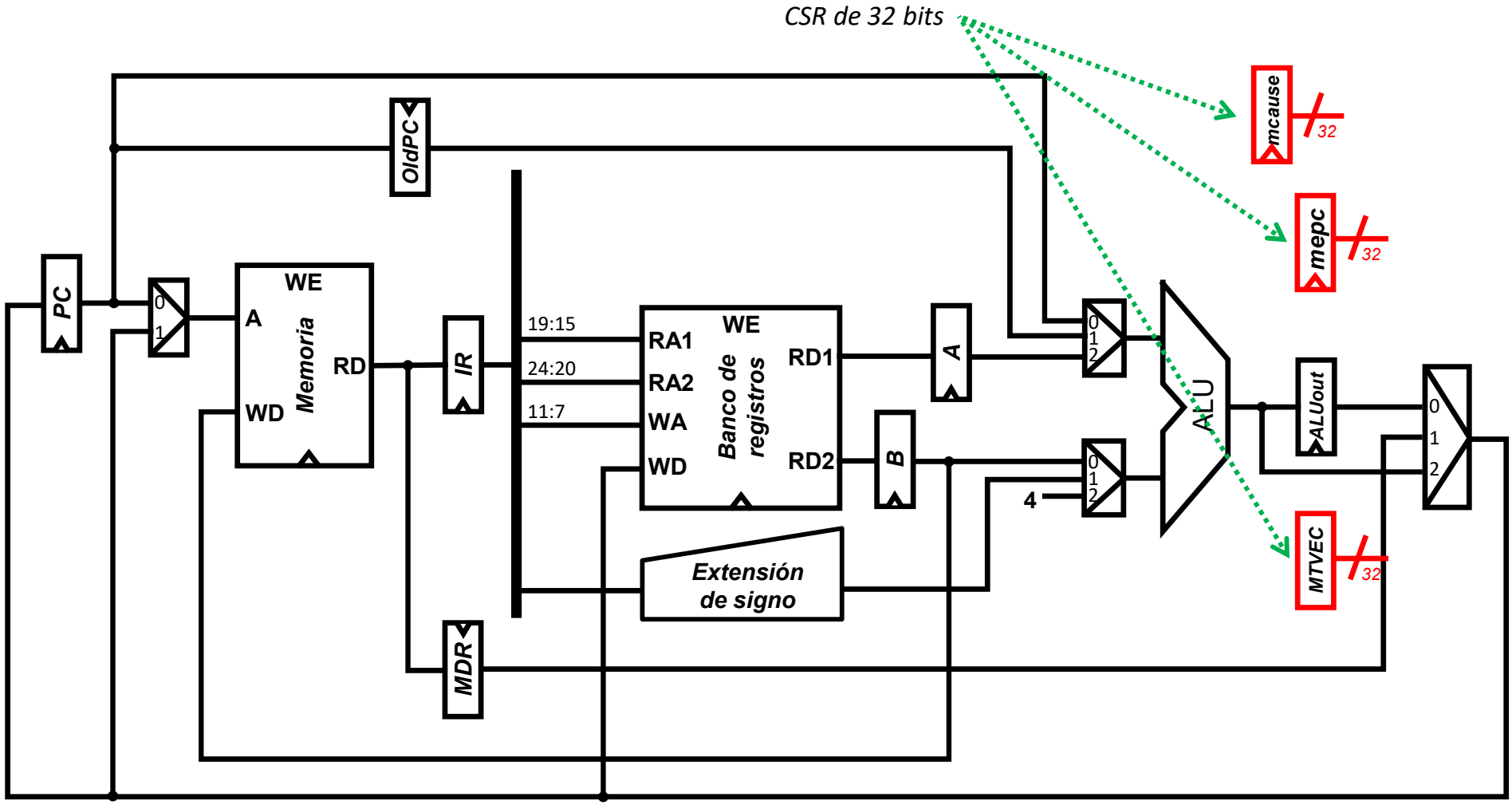
Procesador multiciclo

Ruta de datos + controlador optimizado originales



Procesador multiciclo

Ruta de datos + gestión de excepciones (i)

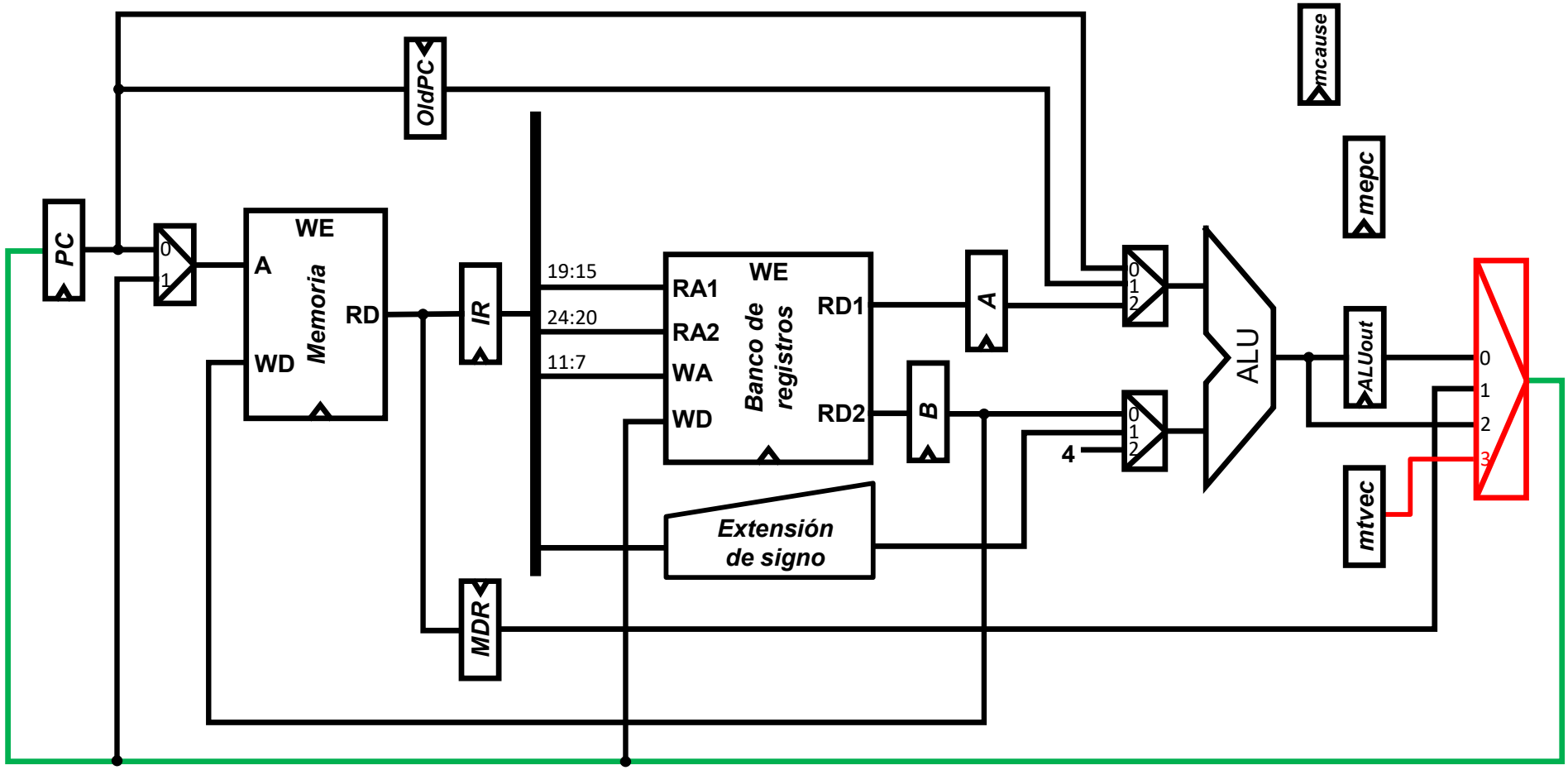


Procesador multiciclo

Ruta de datos + gestión de excepciones (ii)

si excepción:

PC ← mtvec, mepc ← PC, mcause ← "*cause*"

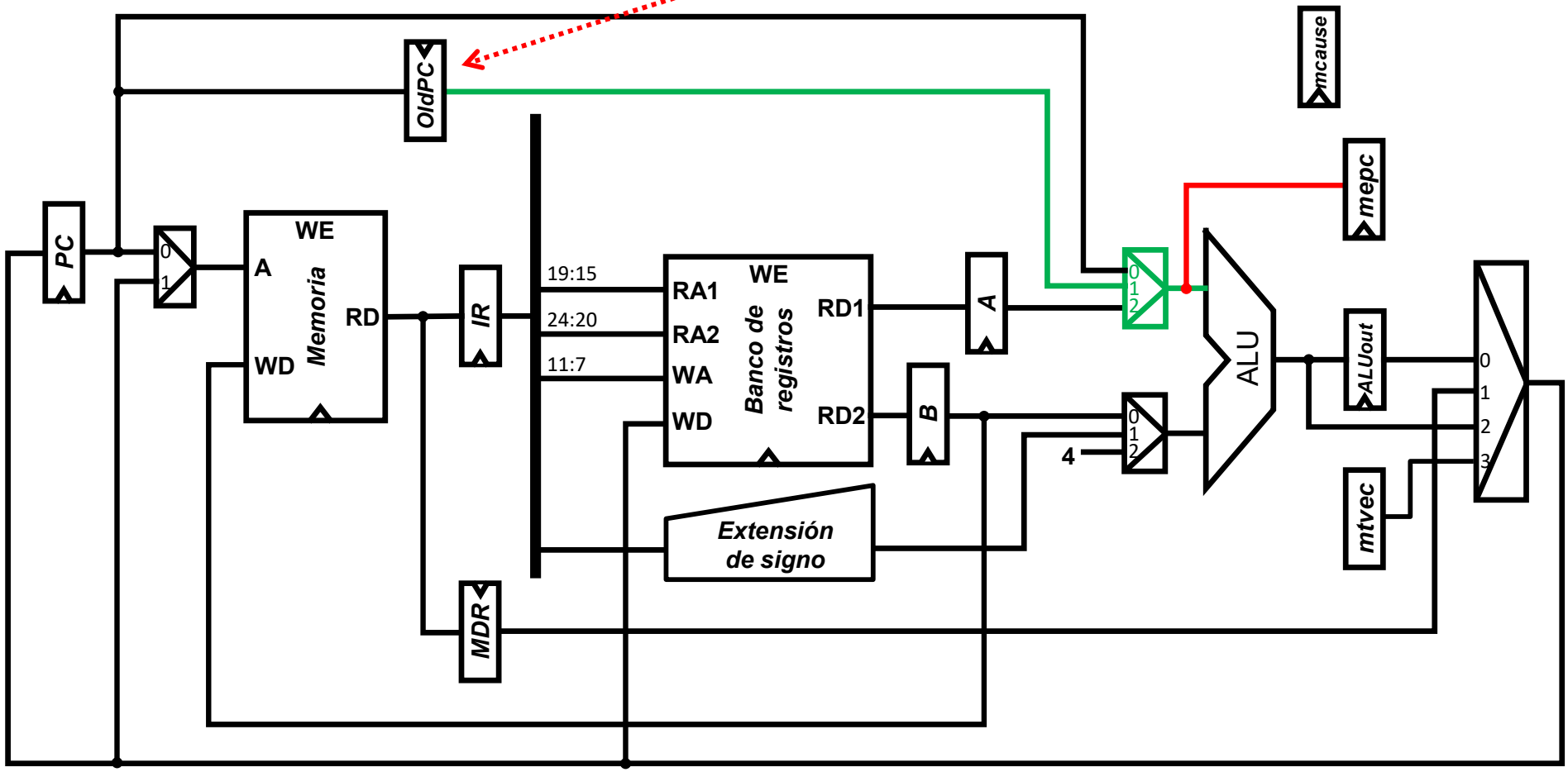


Procesador multiciclo

Ruta de datos + gestión de excepciones (iii)

si excepción:
PC ← mtvec, mepc ← PC, mcause ← *"cause"*

la dirección de la instrucción en ejecución está almacenada en OldPC

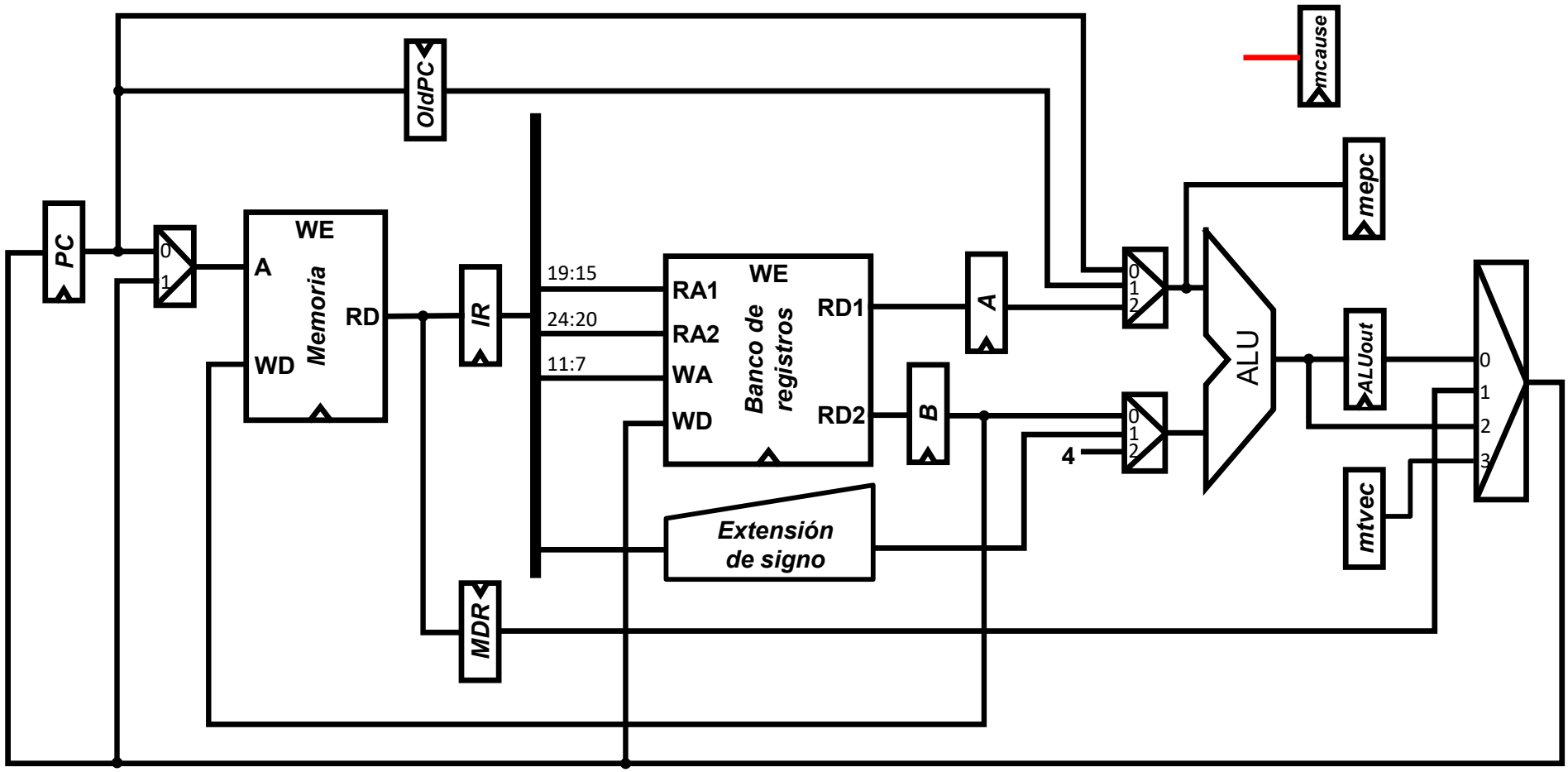


Procesador multiciclo

Ruta de datos + gestión de excepciones (iv)

si excepción:

PC ← mtvec, mepc ← PC, mcause ← "*cause*"

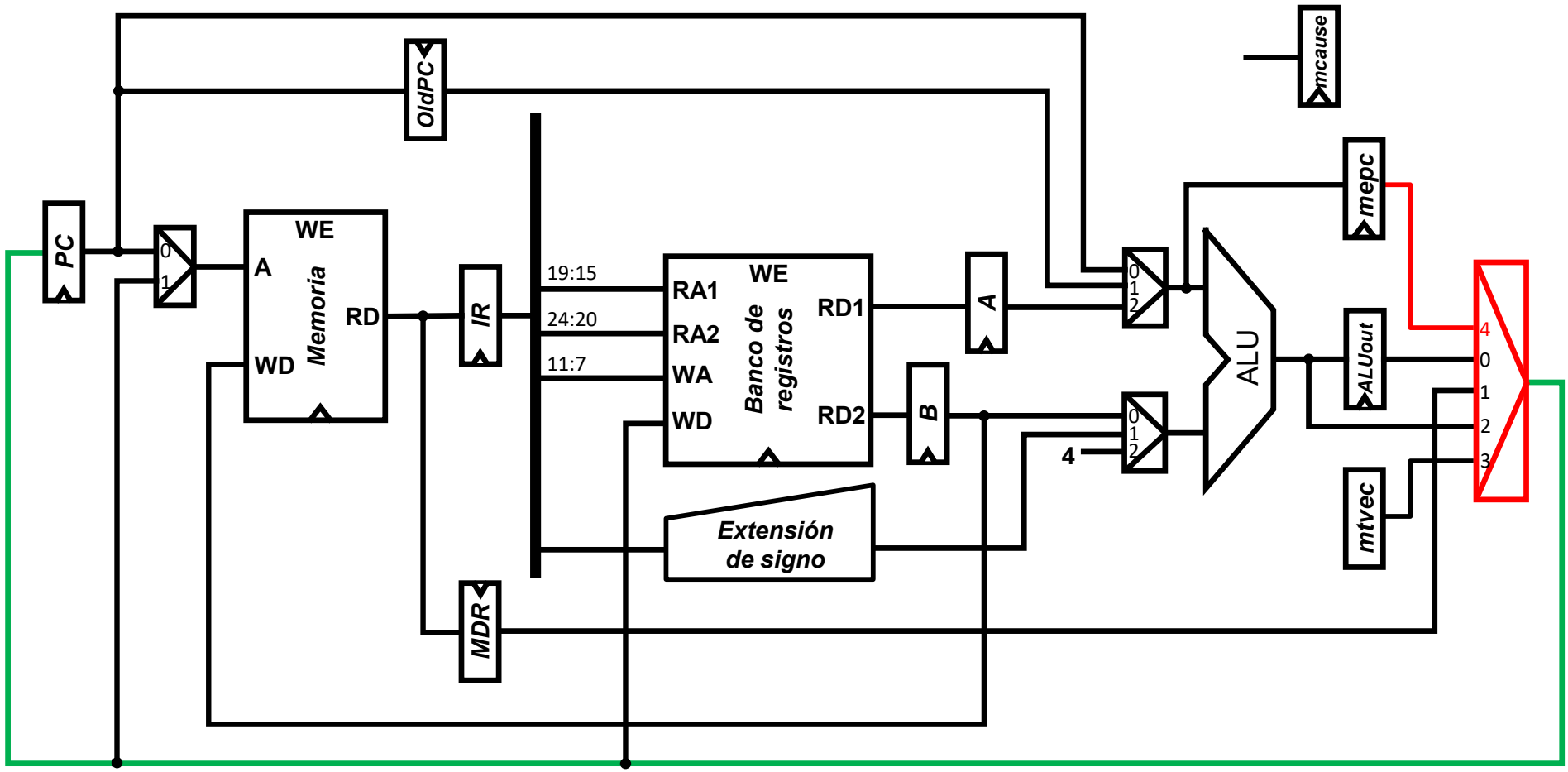


Procesador multiciclo

Ruta de datos + gestión de excepciones (v)

mret

PC ← mepc

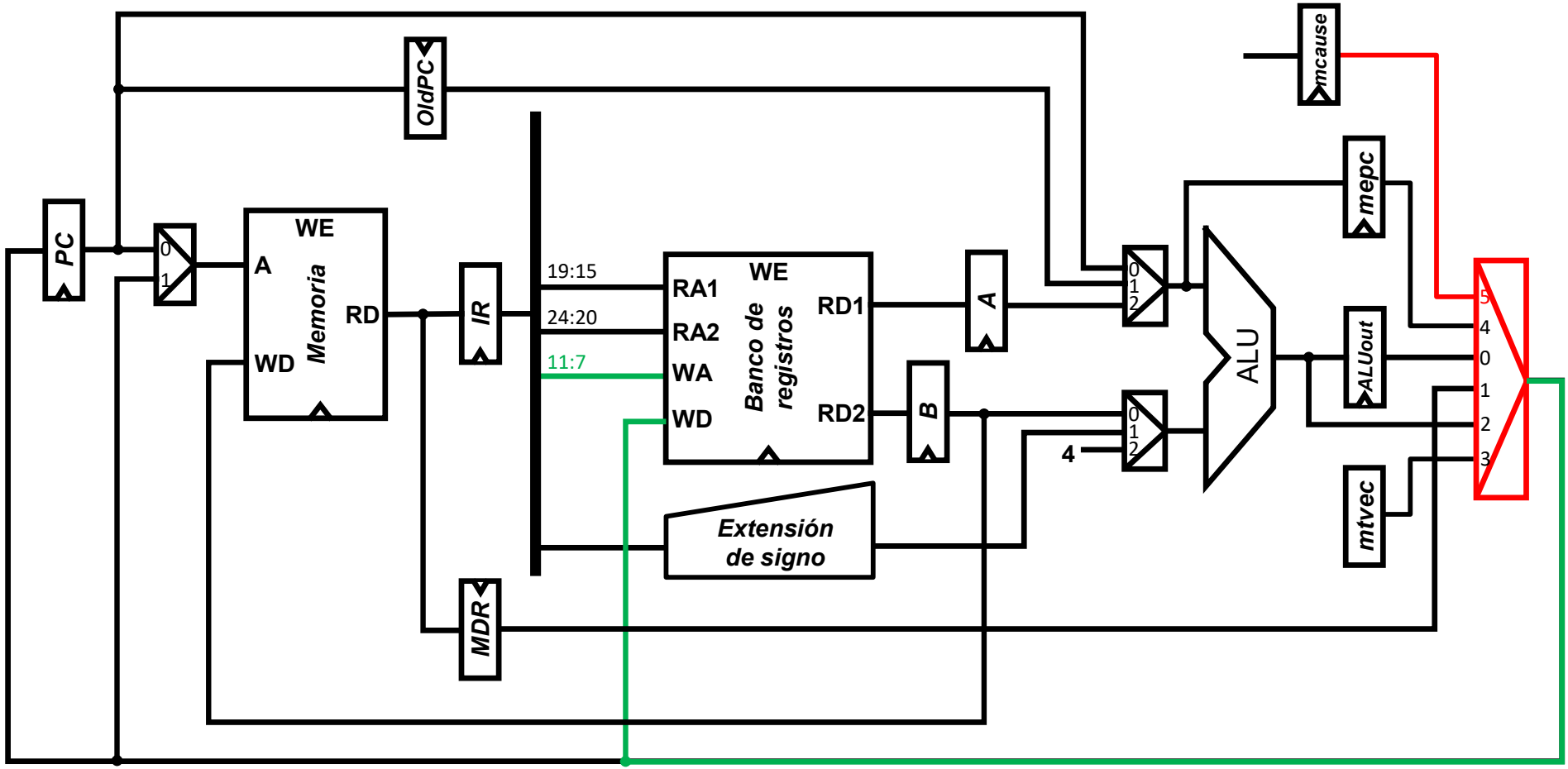


Procesador multiciclo

Ruta de datos + gestión de excepciones (vi)

```
csrrw rd, mcause, rs1
```

BR[rd] ← mcause, PC ← PC+4

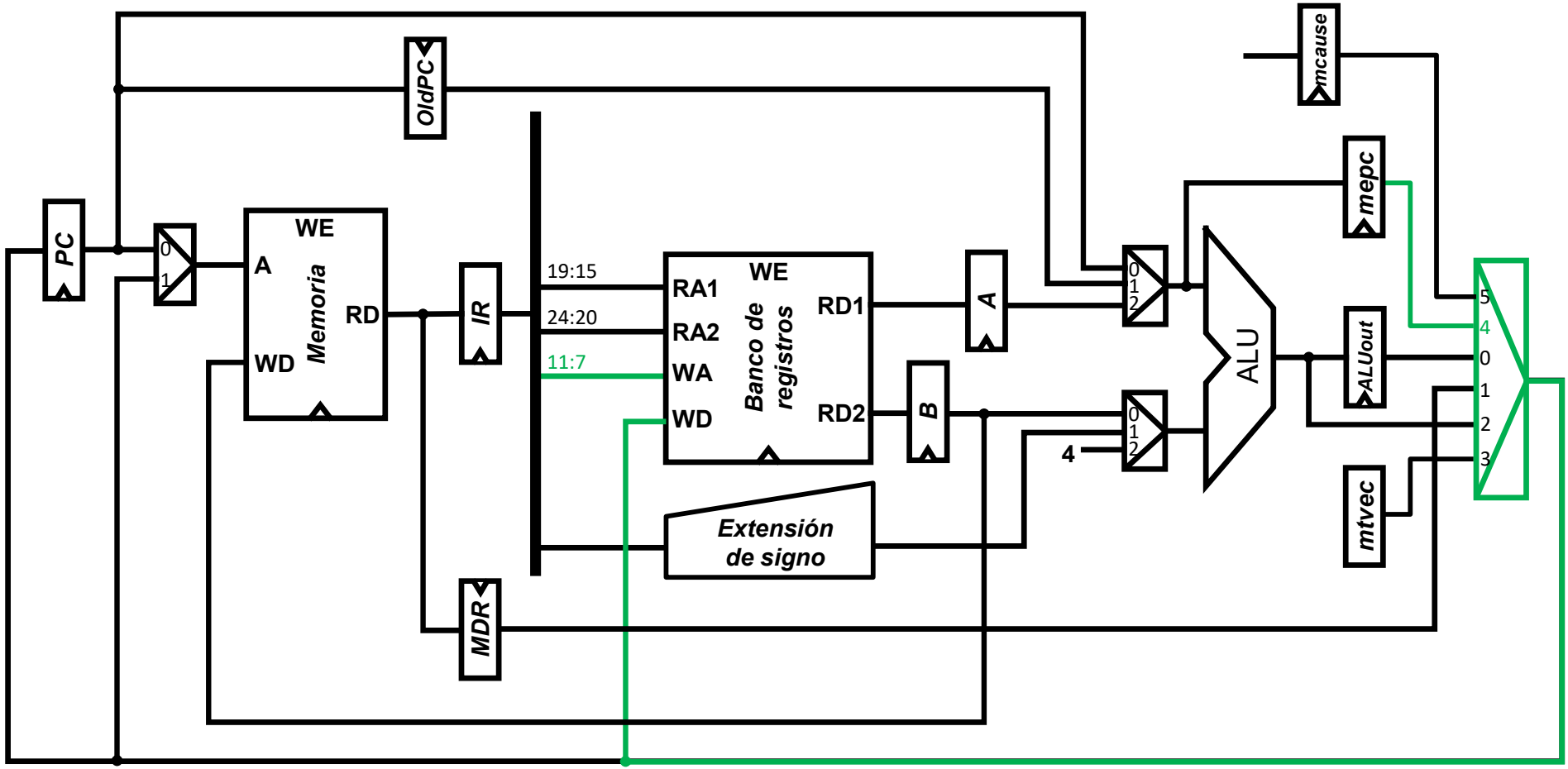


Procesador multiciclo

Ruta de datos + gestión de excepciones (vii)

```
csrrw rd, mepc, rs1
```

BR[rd] ← mepc, mepc ← BR[rs1], PC ← PC+4



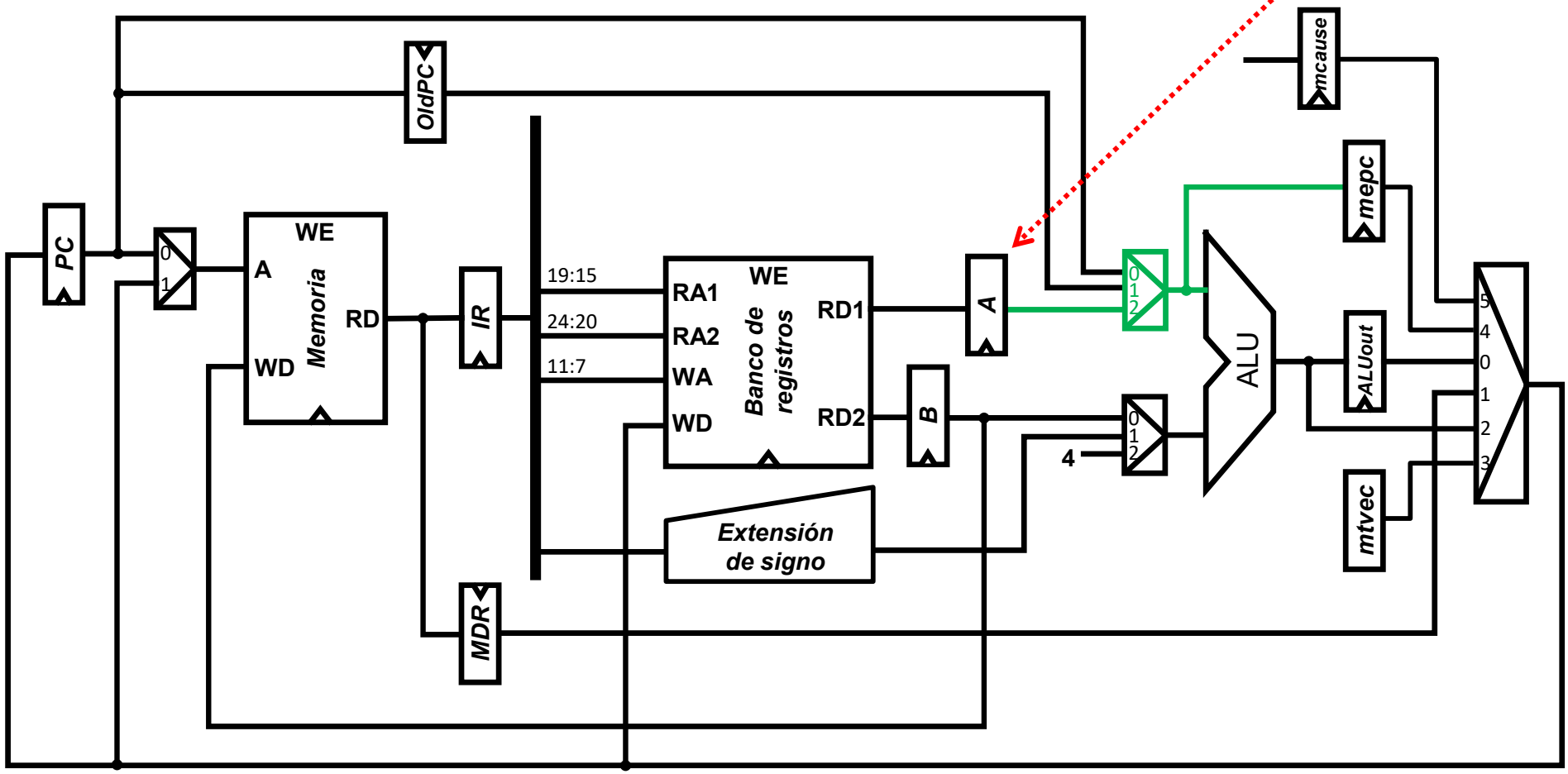
Procesador multiciclo

Ruta de datos + gestión de excepciones (viii)

```
csrrw rd, mepc, rs1
```

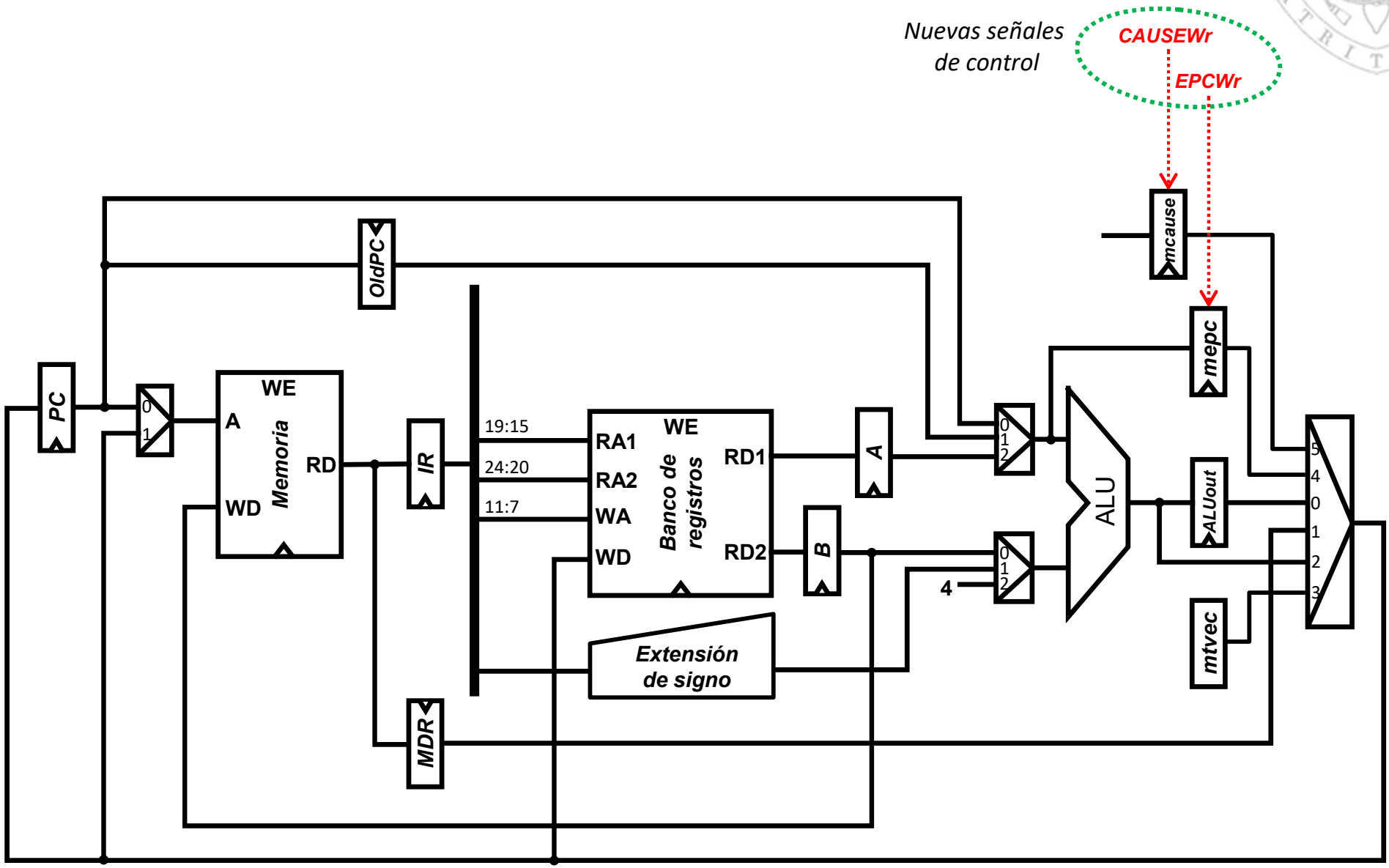
BR[rd] ← mepc, mepc ← BR[rs1], PC ← PC+4

el operando fuente está almacenado en A



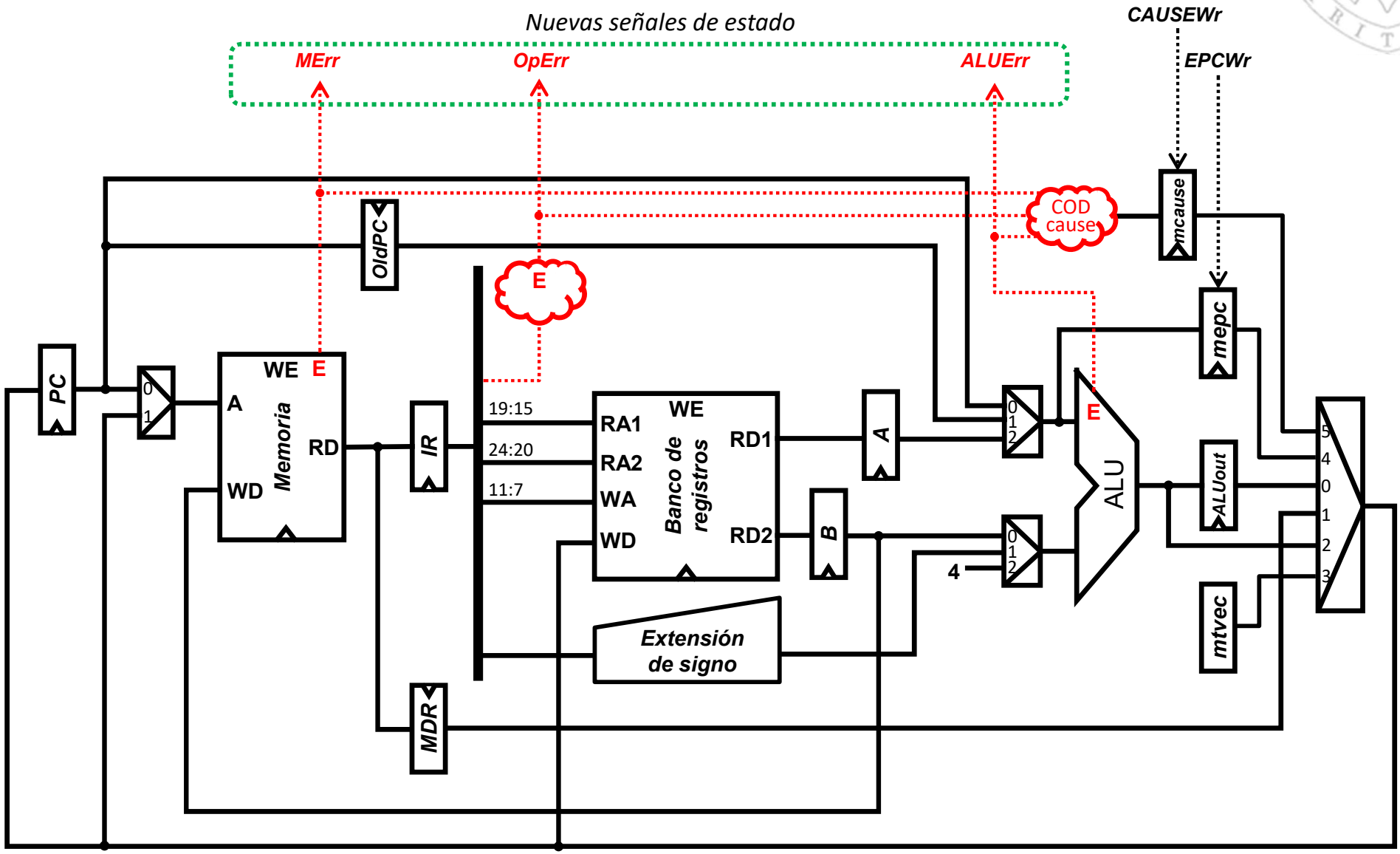
Procesador multiciclo

Ruta de datos + gestión de excepciones (ix)



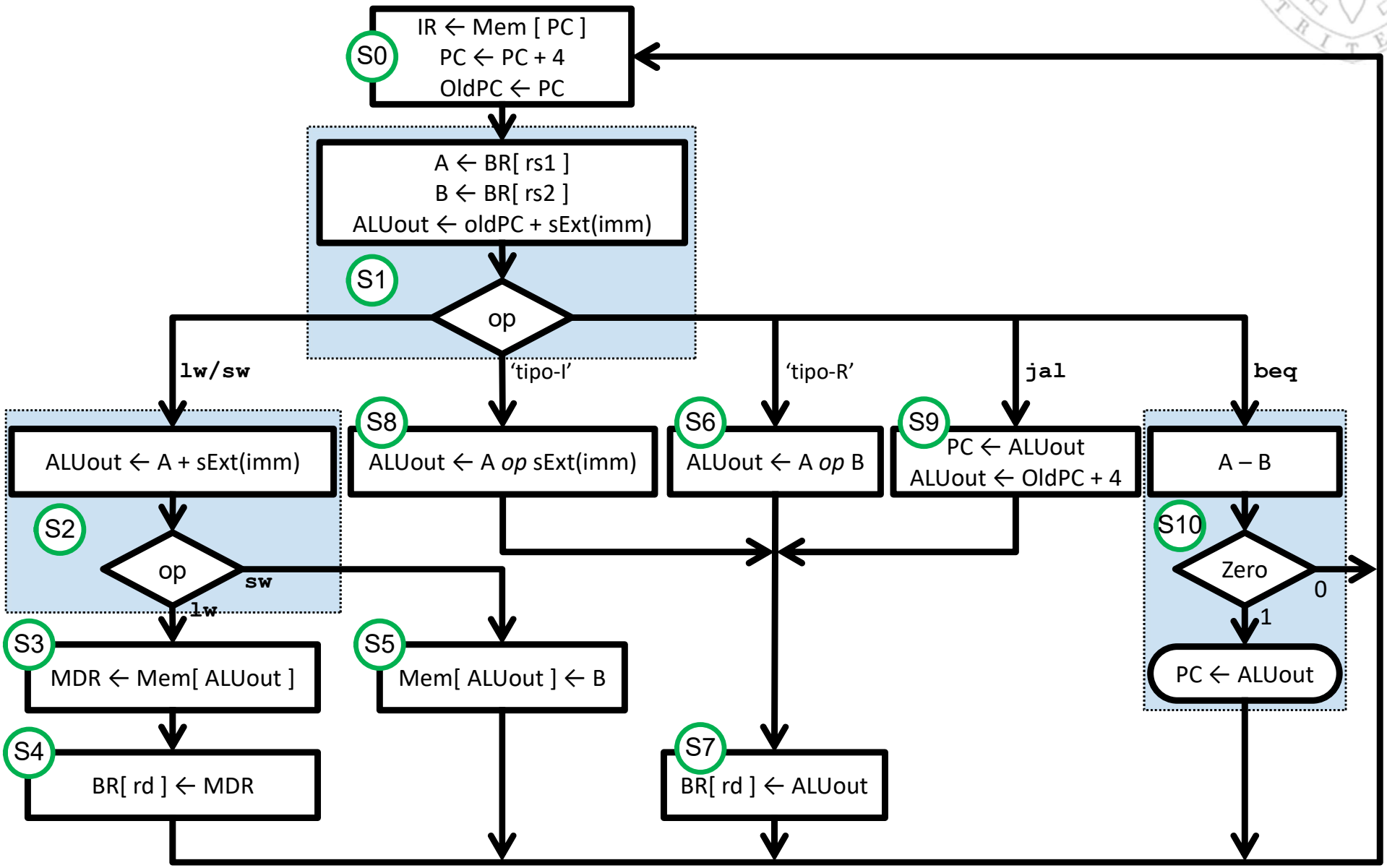
Procesador multiciclo

Ruta de datos + gestión de excepciones (x)



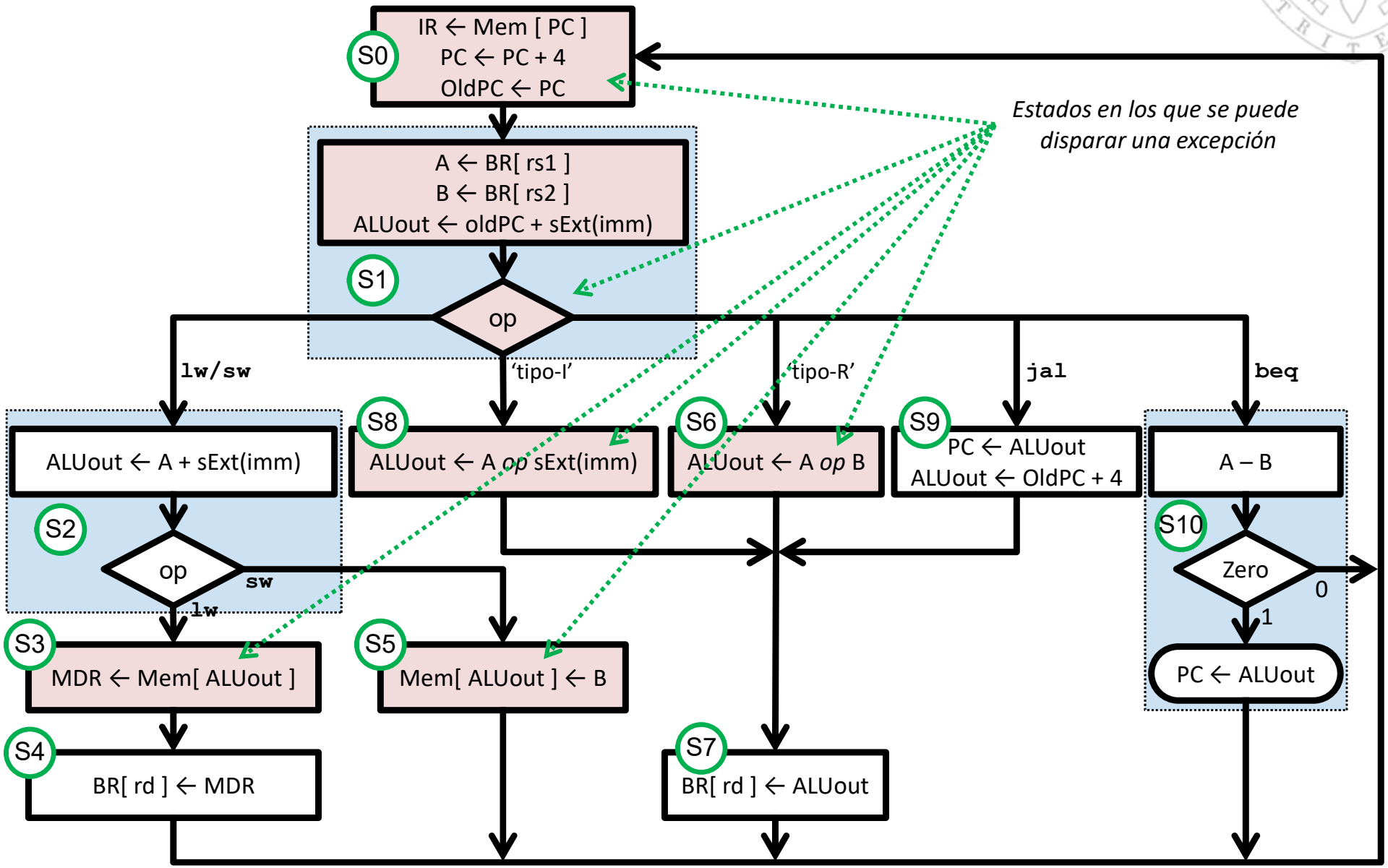
Procesador multiciclo

Diagrama ASM original de la FSM principal (i)



Procesador multiciclo

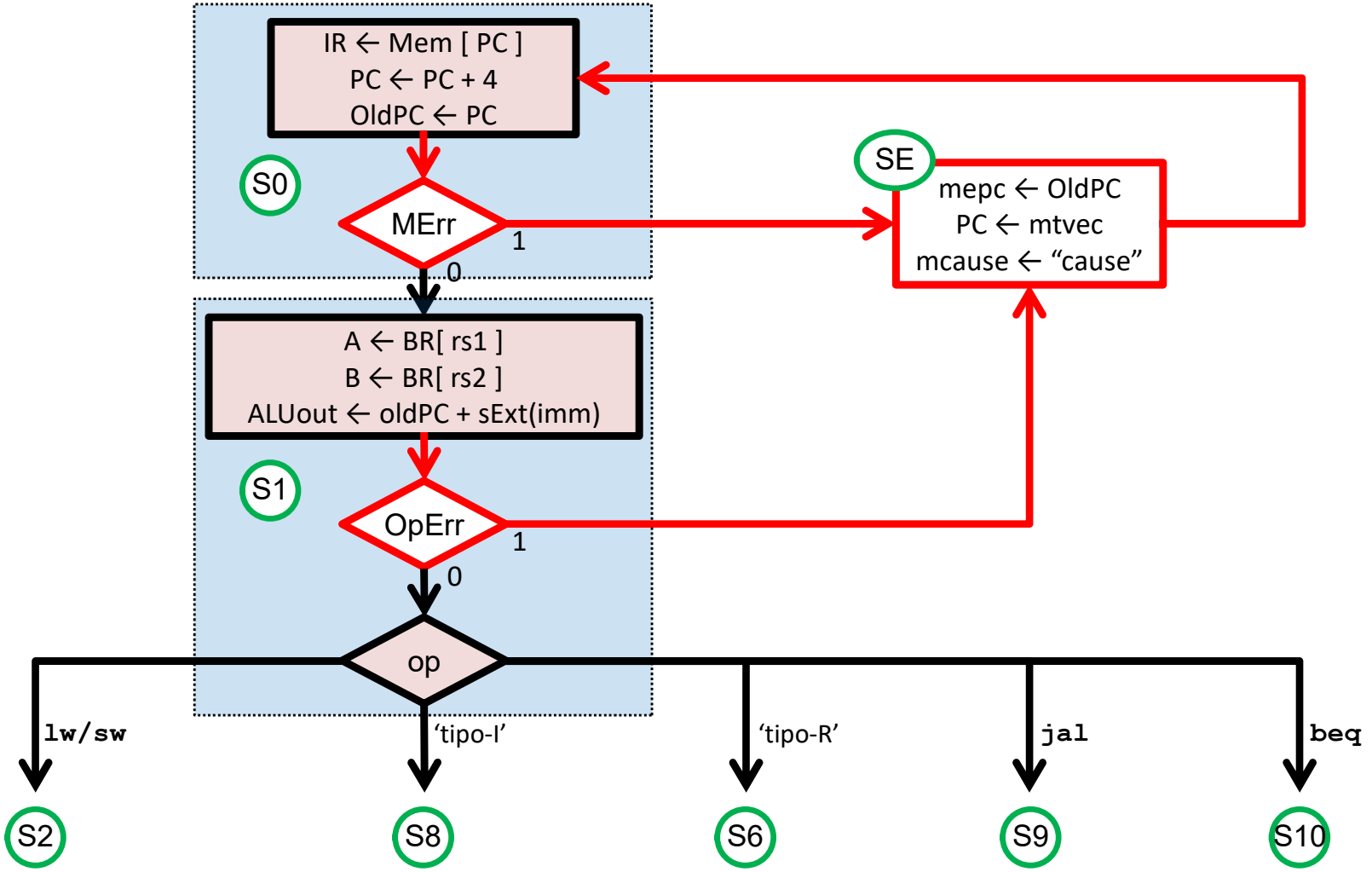
Diagrama ASM original de la FSM principal (i)



Procesador multiciclo

ASM de la FSM principal + gestión de excepciones (i)

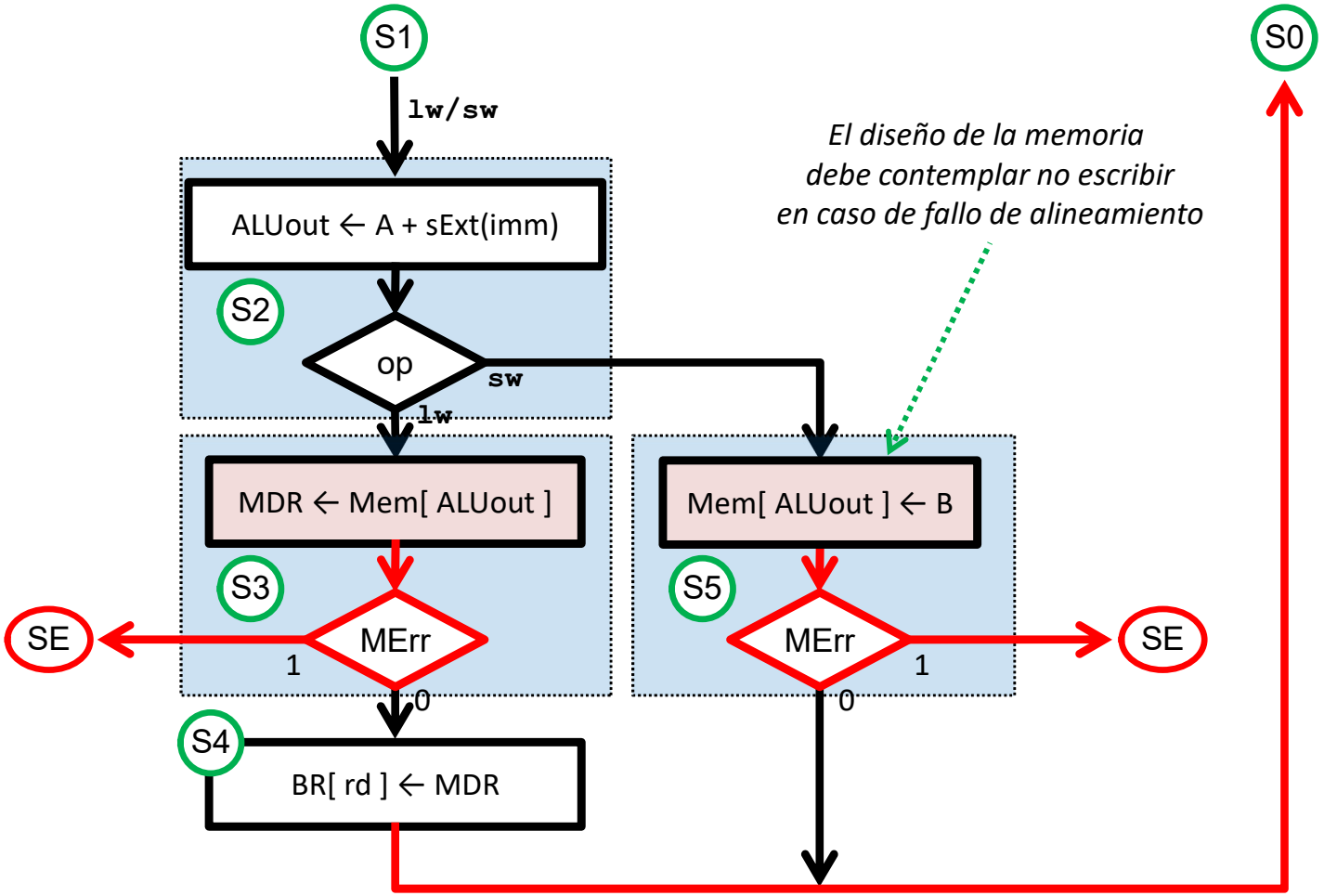
- Se añade estado SE al que se salta en caso de excepción.
  - SE actualiza CSR y cancela la instrucción en curso saltando a S0.



Procesador multiciclo

ASM de la FSM principal + gestión de excepciones (ii)

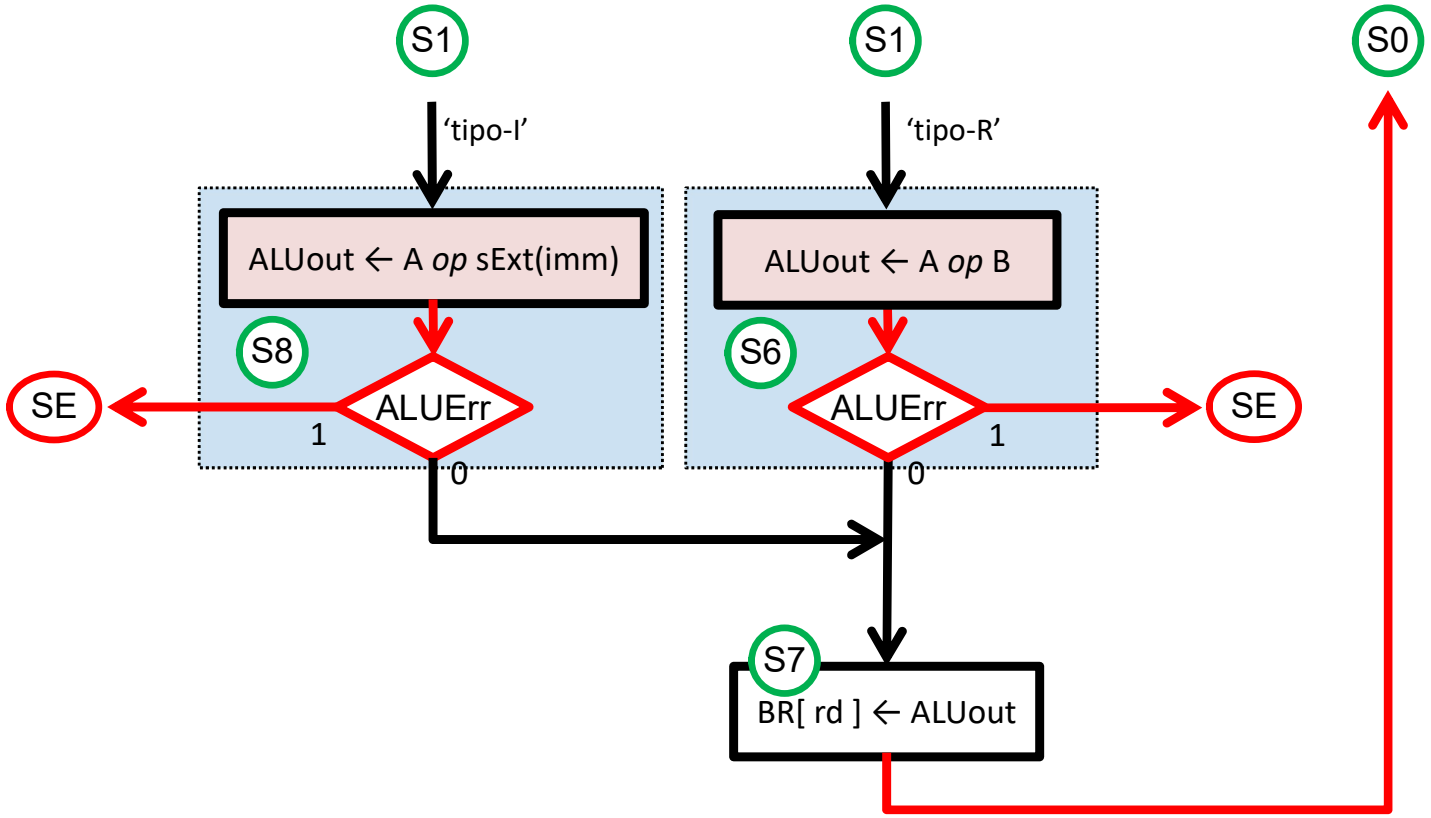
- Todos los estados en los que puede dispararse una excepción chequean la señal de estado correspondiente para saltar o no a SE.



Procesador multiciclo

ASM de la FSM principal + gestión de excepciones (iii)

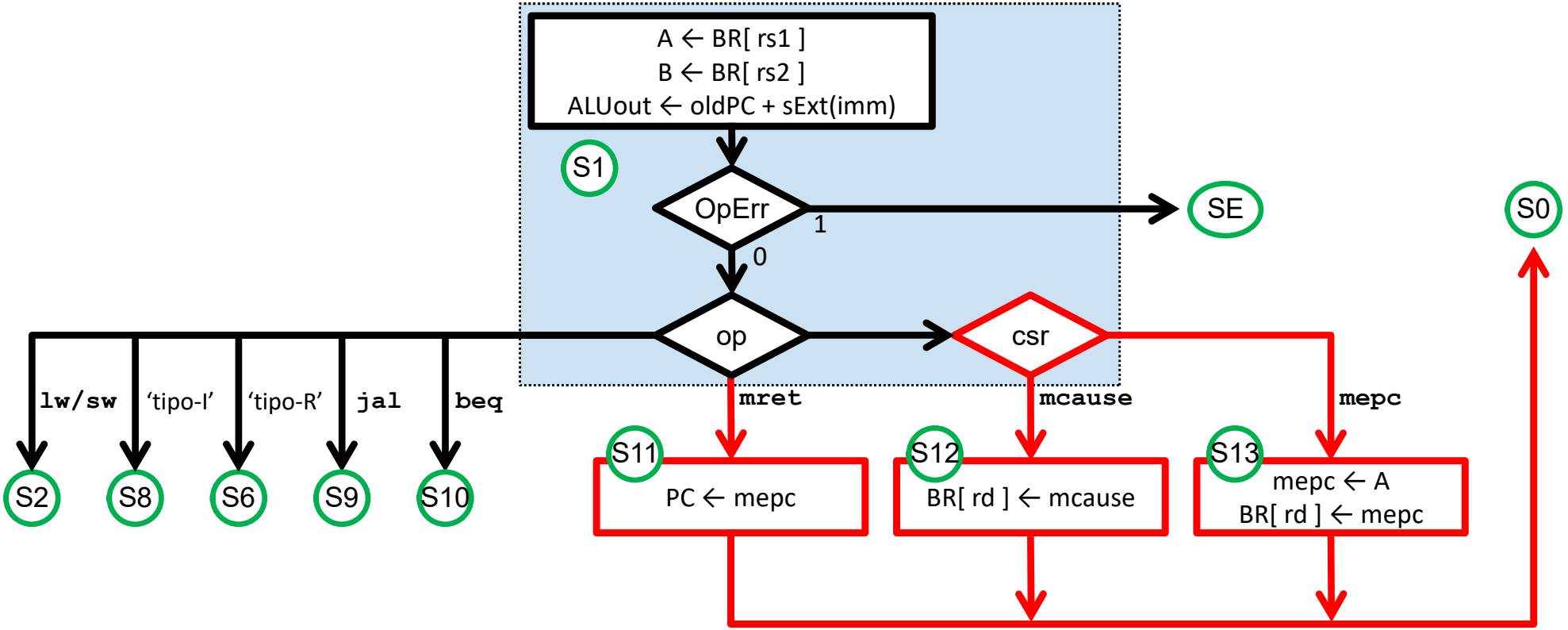
- Todos los estados en los que puede dispararse una excepción chequean la señal de estado correspondiente para saltar o no a SE.



Procesador multiciclo

ASM de la FSM principal + gestión de excepciones (iv)

- Se añaden estados para soportar las nuevas instrucciones:
  - Ninguna de ellas puede disparar excepción.



Procesador segmentado

Consideraciones previas

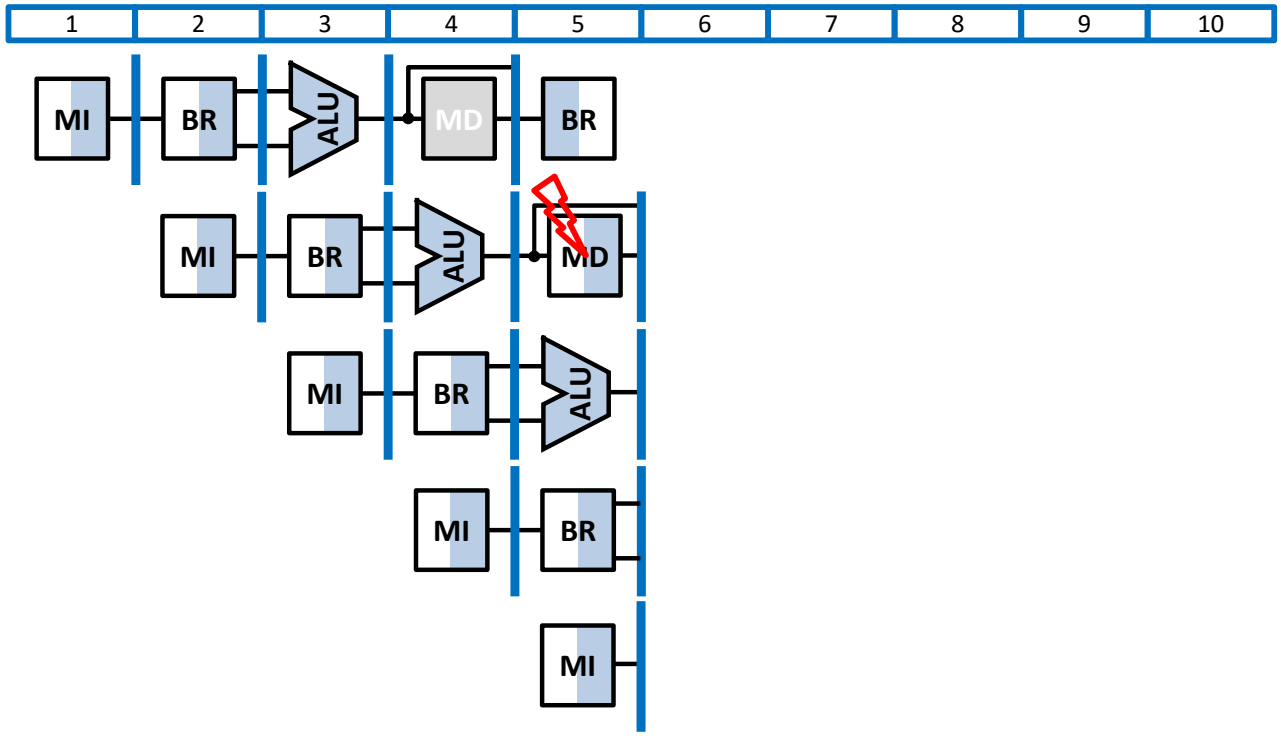
- Las excepciones son saltos tomados a una dirección implícita, por ello en RISC-V se tratan de manera análoga a los conflictos de control, así:
  - Las instrucciones anteriores a la que genera la excepción se completan normalmente.
  - La instrucción que dispara la excepción y todas las que se lanzaron con posterioridad se descartan.
  - Se lanza la instrucción ubicada en la dirección implícita.
  - Pero, a diferencia de los saltos que se efectúan en la etapa EX, las excepciones se tratan cuando la instrucción causante está en la etapa MEM.

Procesador segmentado

Excepción simple (i)

- El tratamiento de excepción por acceso no alineado a datos será:
  - Ciclo 5: **`lw`** (está en MEM) genera excepción.

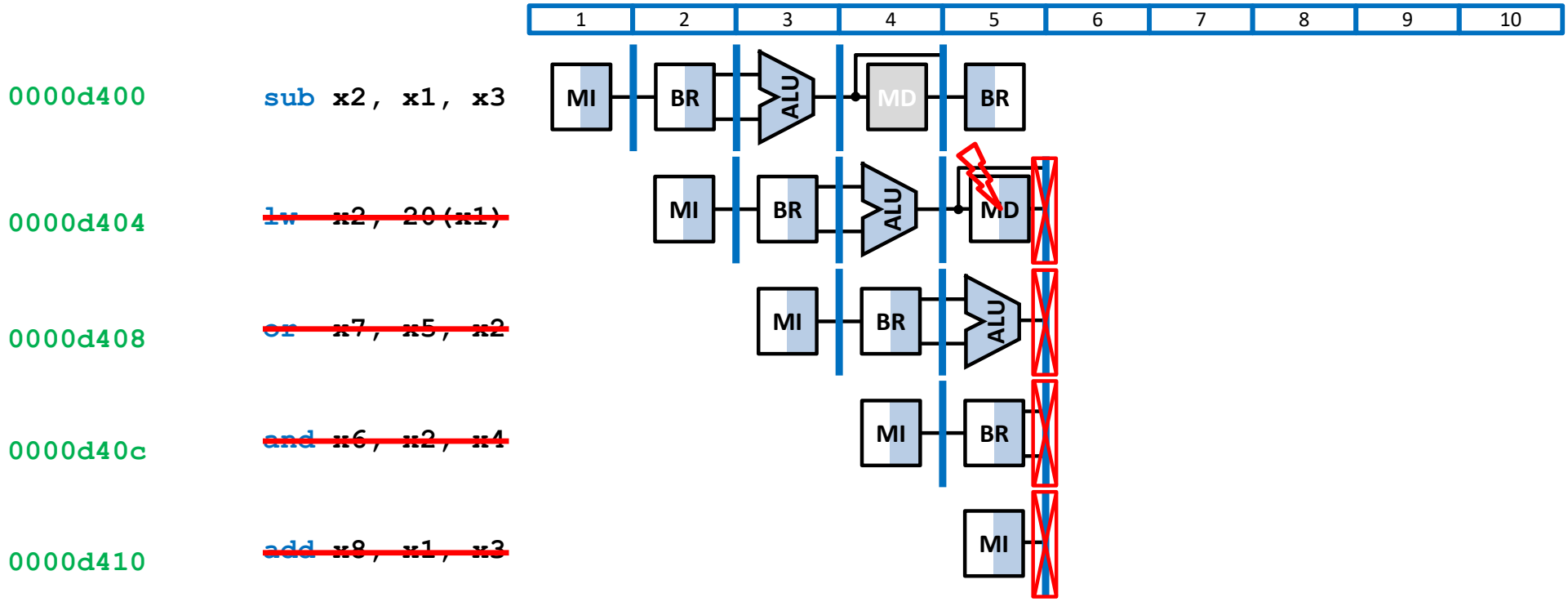
```
0000d400    sub x2, x1, x3
0000d404    lw  x2, 20(x1)
0000d408    or  x7, x5, x2
0000d40c    and x6, x2, x4
0000d410    add x8, x1, x3
```



Procesador segmentado

Excepción simple (i)

- El tratamiento de excepción por acceso no alineado a datos será:
  - Ciclo 5: **`lw`** (está en MEM) genera excepción. Se descartan **`lw`** y siguientes.
    - **`mepc`** = 0x0000d404 y **`cause`** = 4

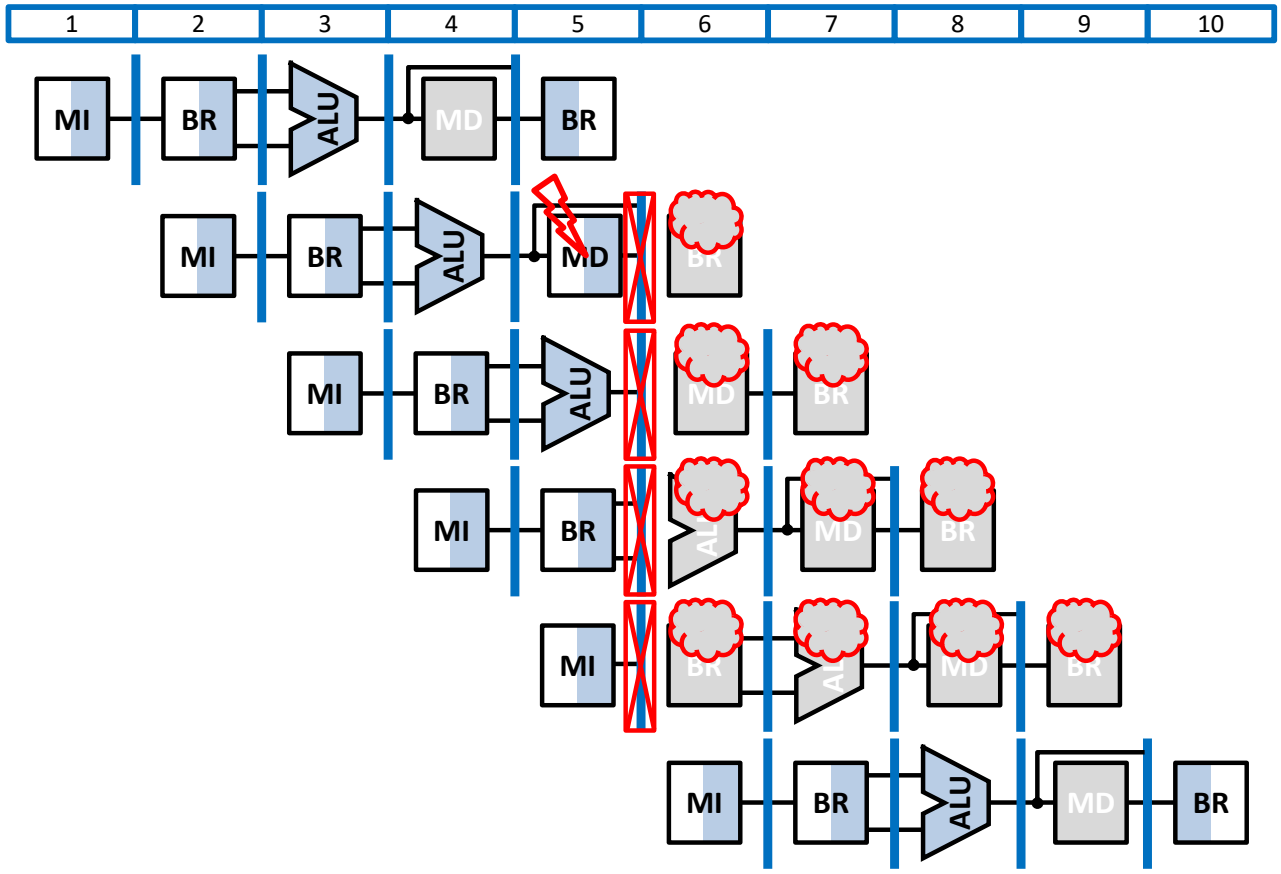


Procesador segmentado

Excepción simple (i)

- El tratamiento de excepción por acceso no alineado a datos será:
  - Ciclo 5: **`lw`** (está en MEM) genera excepción. Se descartan **`lw`** y siguientes.
    - **`mepc`** = 0x0000d404 y **`cause`** = 4
  - Ciclo 6: se lanza la instrucción cuya dirección almacena **`mtvec`**.

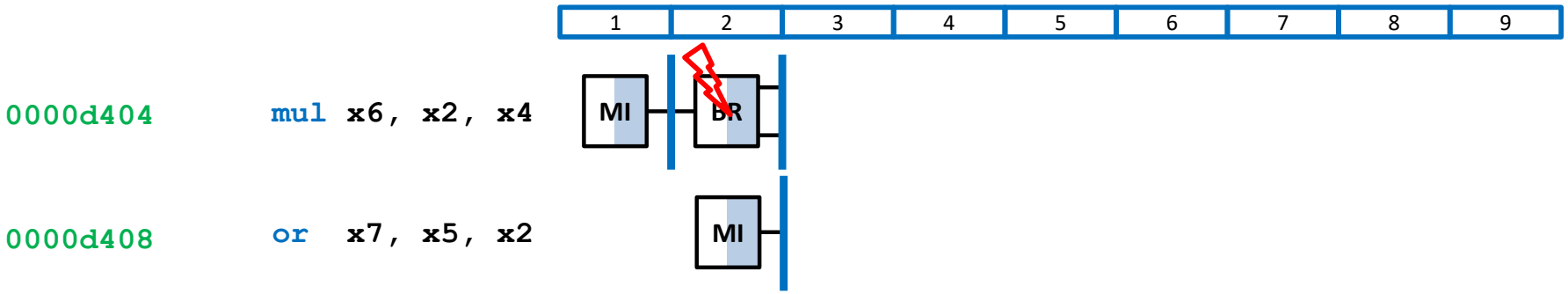
```
0000d400        sub x2, x1, x3
0000d404        lw  x3, 20(x1)
0000d408        or  x7, x5, x2
0000d40c        and x6, x2, x4
0000d410        add x8, x1, x3
1c000000  RTE:  add sp, sp, -8
```



Procesador segmentado

Excepción simple (ii)

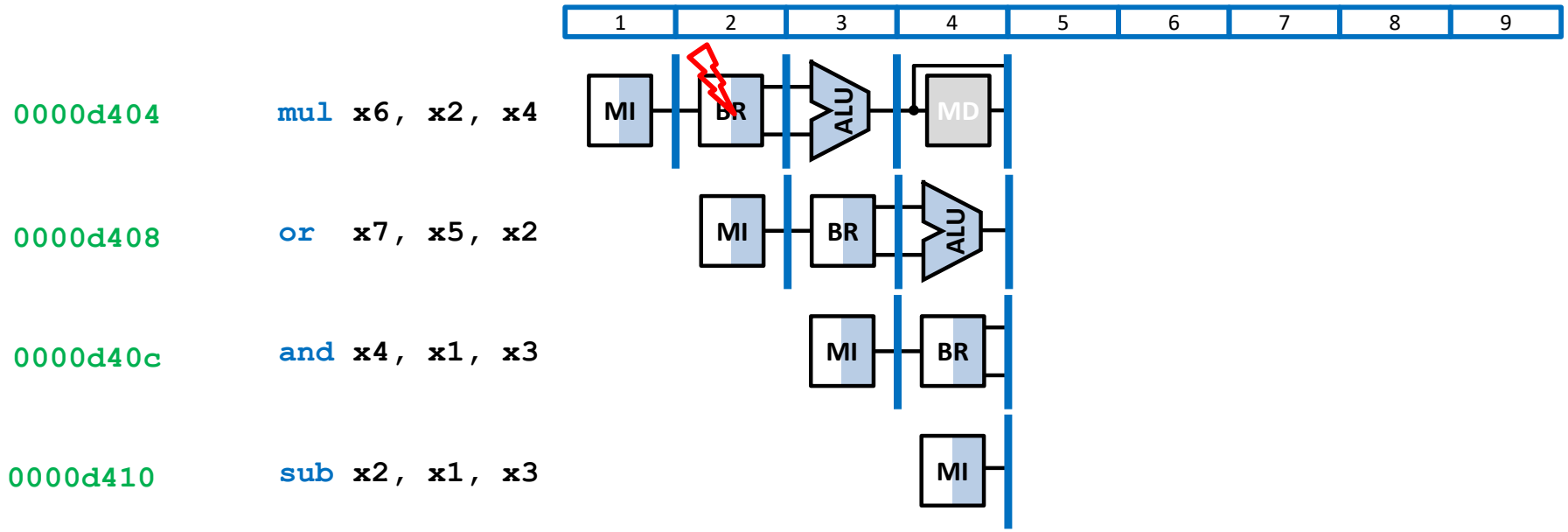
- Con independencia de cuándo se produzca la excepción, siempre se tratará cuando la instrucción desencadenante llegue a MEM:
  - Ciclo 2: **`mul`** (está en ID) genera excepción.



Procesador segmentado

Excepción simple (ii)

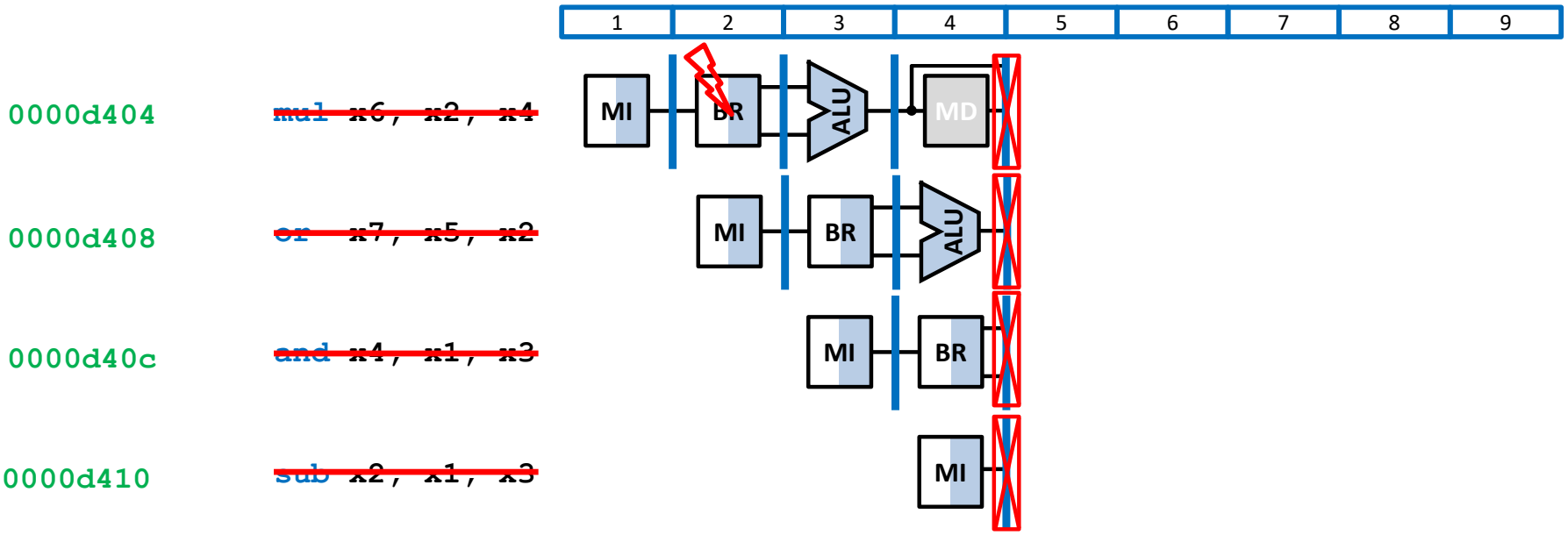
- Con independencia de cuándo se produzca la excepción, siempre se tratará cuando la instrucción desencadenante llegue a MEM:
  - Ciclo 2: **`mul`** (está en ID) genera excepción.
  - Ciclo 4: **`mul`** (está en MEM).



Procesador segmentado

Excepción simple (ii)

- Con independencia de cuándo se produzca la excepción, siempre se tratará cuando la instrucción desencadenante llegue a MEM:
  - Ciclo 2: **`mul`** (está en ID) genera excepción.
  - Ciclo 4: **`mul`** (está en MEM). Se descartan **`mul`** y siguientes.
    - **`mepc`** = 0x0000d404 y **`cause`** = 2

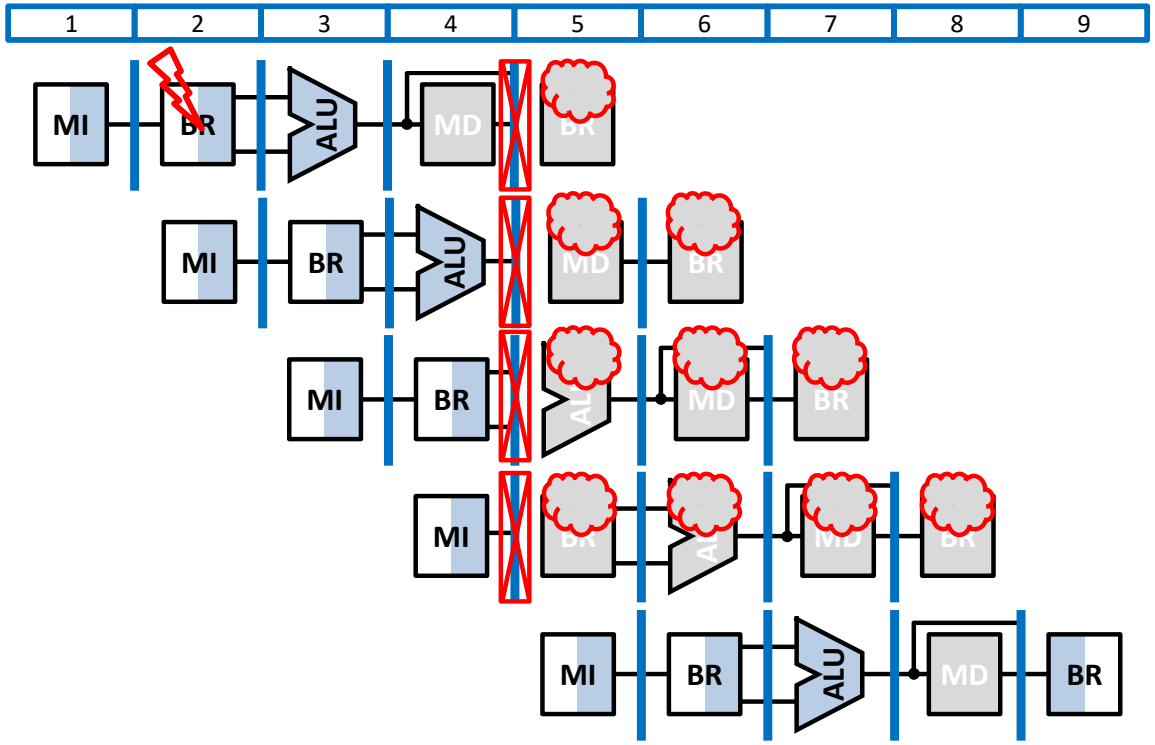


Procesador segmentado

Excepción simple (ii)

- Con independencia de cuándo se produzca la excepción, siempre se tratará cuando la instrucción desencadenante llegue a MEM:
  - Ciclo 2: **`mul`** (está en ID) genera excepción.
  - Ciclo 4: **`mul`** (está en MEM). Se descartan **`mul`** y siguientes.
    - **`mepc`** = 0x0000d404 y **`cause`** = 2
  - Ciclo 6: se lanza la instrucción cuya dirección almacena **`mtvec`**.

```
0000d404    mul x6, x2, x4
0000d408    or  x7, x5, x2
0000d40c    and x4, x1, x3
0000d410    sub x2, x1, x3

1c000000 RTE: add sp, sp, -8
```

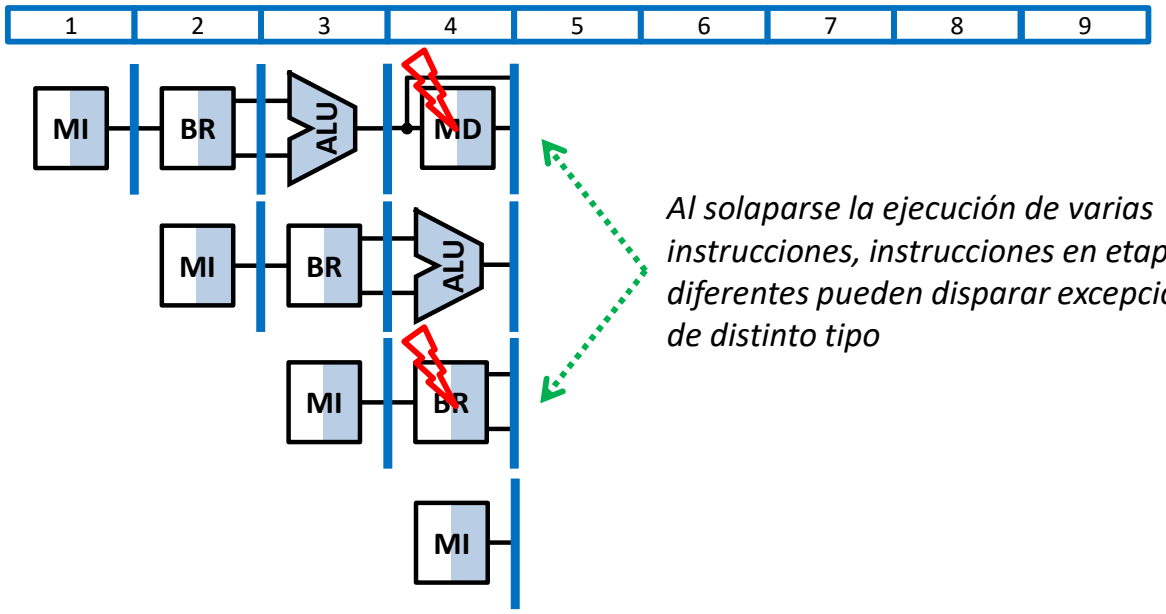


Procesador segmentado

Excepciones múltiples simultáneas

- Esto permite que si existen excepciones simultáneas, se trate la de la instrucción lanzada en primer lugar al ser la que llega antes a MEM:
  - Ciclo 4: las instrucciones **`lw`** (está en MEM) y **`mul`** (está en ID) generan excepciones.

```
0000d404        lw   x2, 20(x1)
0000d408        or   x7, x5, x2
0000d40c        mul  x6, x2, x4
0000d410        add  x8, x2, x2
```

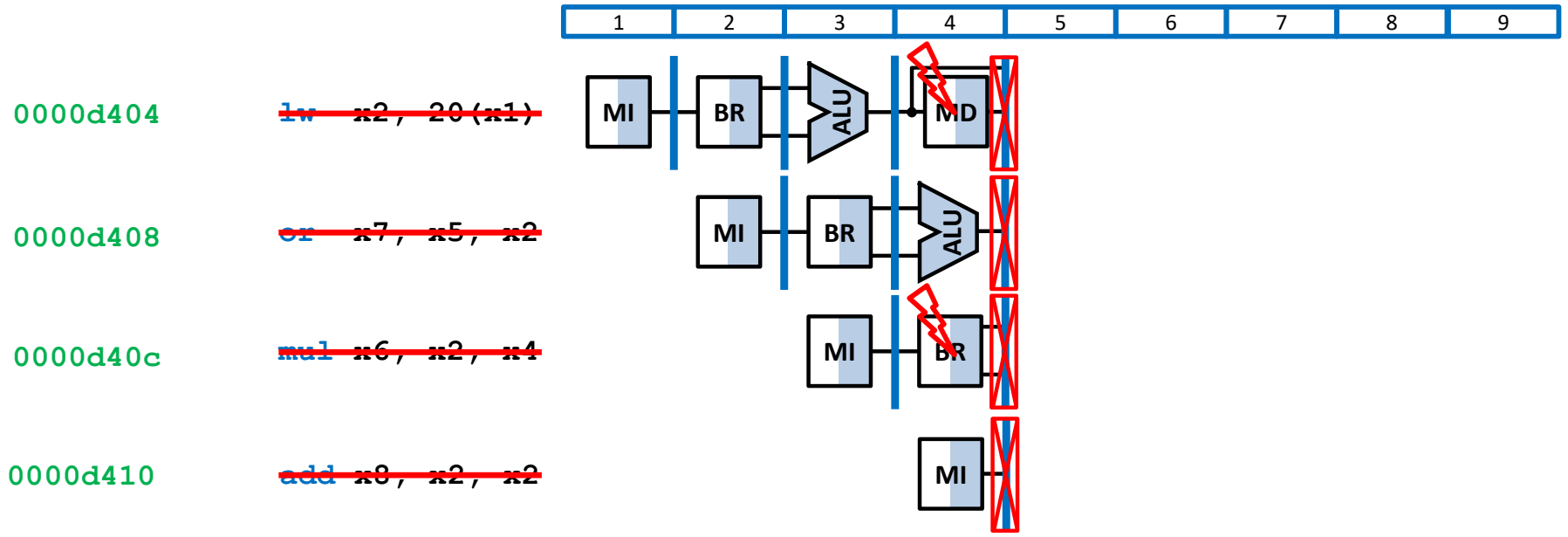


Al solaparse la ejecución de varias instrucciones, instrucciones en etapas diferentes pueden disparar excepciones de distinto tipo

Procesador segmentado

Excepciones múltiples simultáneas

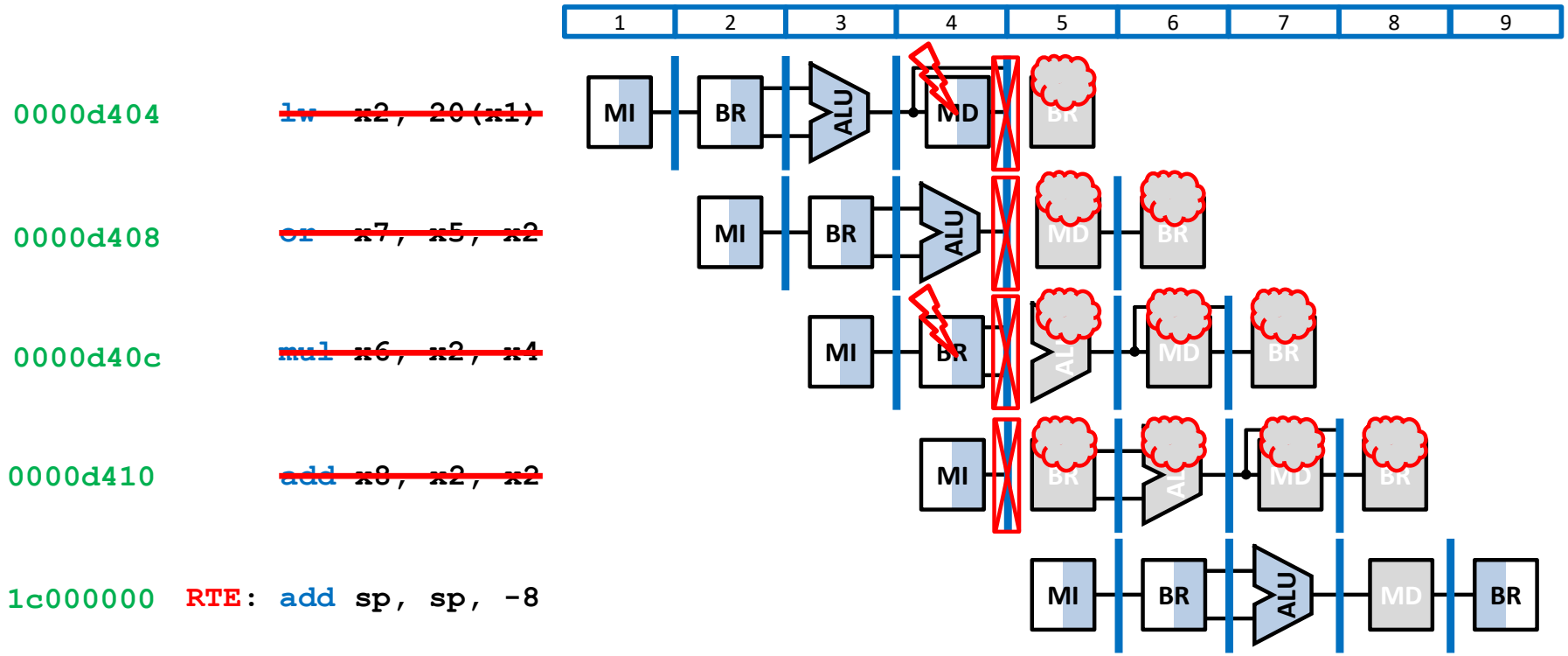
- Esto permite que si existen excepciones simultáneas, se trate la de la instrucción lanzada en primer lugar al ser la que llega antes a MEM:
  - Ciclo 4: las instrucciones **`lw`** (está en MEM) y **`mul`** (está en ID) generan excepciones. Se descarta **`lw`** y todas las que le siguen.
    - **`mepc`** = 0x0000d404 y **`cause`** = 4



Procesador segmentado

Excepciones múltiples simultáneas

- Esto permite que si existen excepciones simultáneas, se trate la de la instrucción lanzada en primer lugar al ser la que llega antes a MEM:
  - Ciclo 4: las instrucciones **`lw`** (está en MEM) y **`mul`** (esta en ID) generan excepciones. Se descarta **`lw`** y todas las que le siguen.
    - **`mepc`** = 0x0000d404 y **`cause`** = 4
  - Ciclo 5: se lanza la instrucción cuya dirección almacena **`mtvec`**.

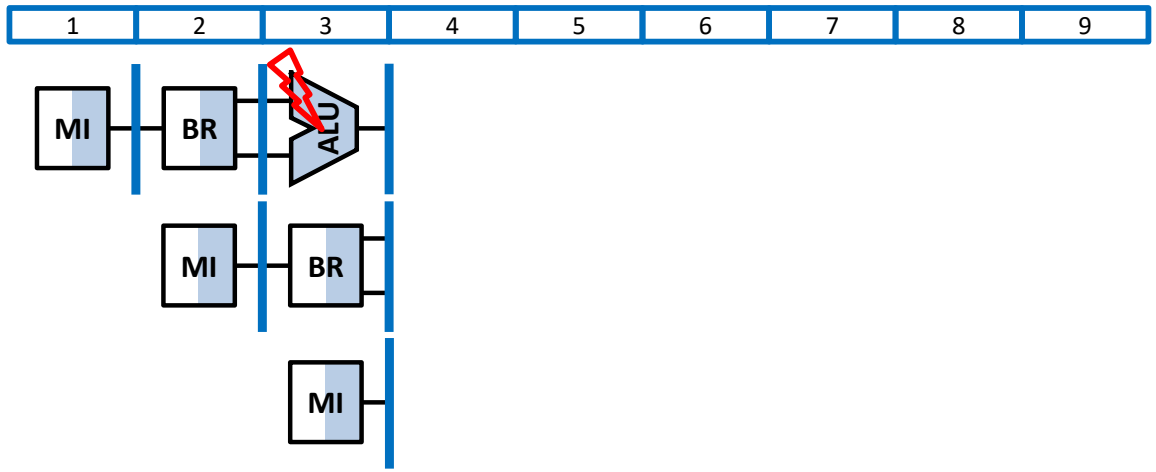


Procesador segmentado

Excepciones múltiples no simultáneas en orden

- Ídem si las excepciones no son simultáneas.
  - Ciclo 3: **`xor`** (está en EX) genera excepción.

```
0000d404        xor x5, x1, x3
0000d408        sra x6, x2, x4
0000d40c        or  x7, x5, x2
```

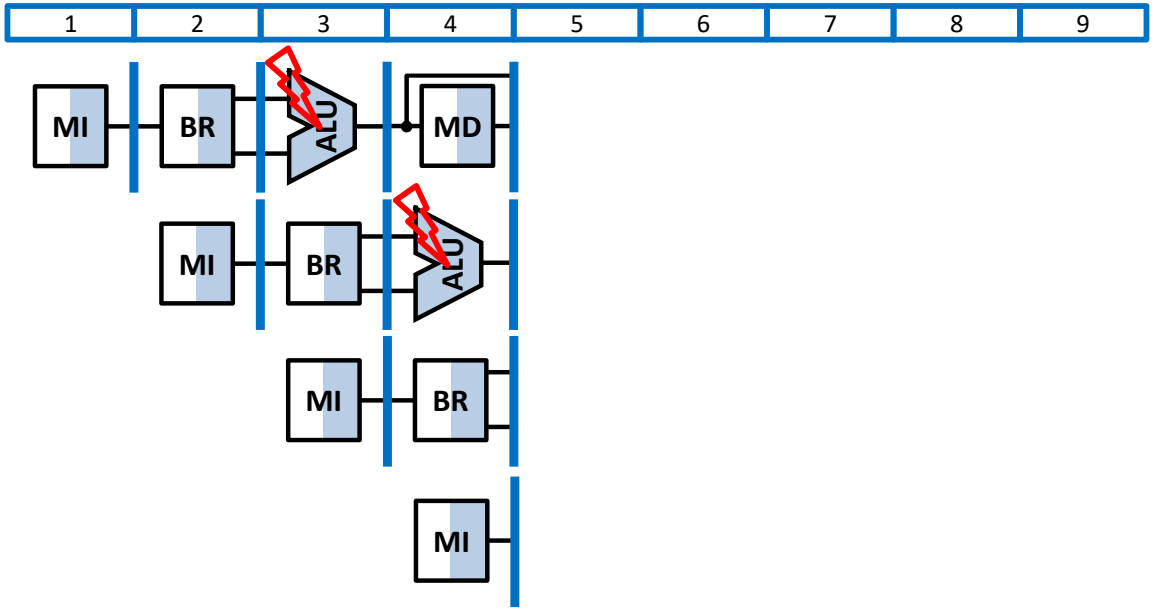


Procesador segmentado

Excepciones múltiples no simultáneas en orden

- Ídem si las excepciones no son simultáneas.
  - Ciclo 3: **`xor`** (está en EX) genera excepción.
  - Ciclo 4: **`sra`** (está en EX) también genera excepción

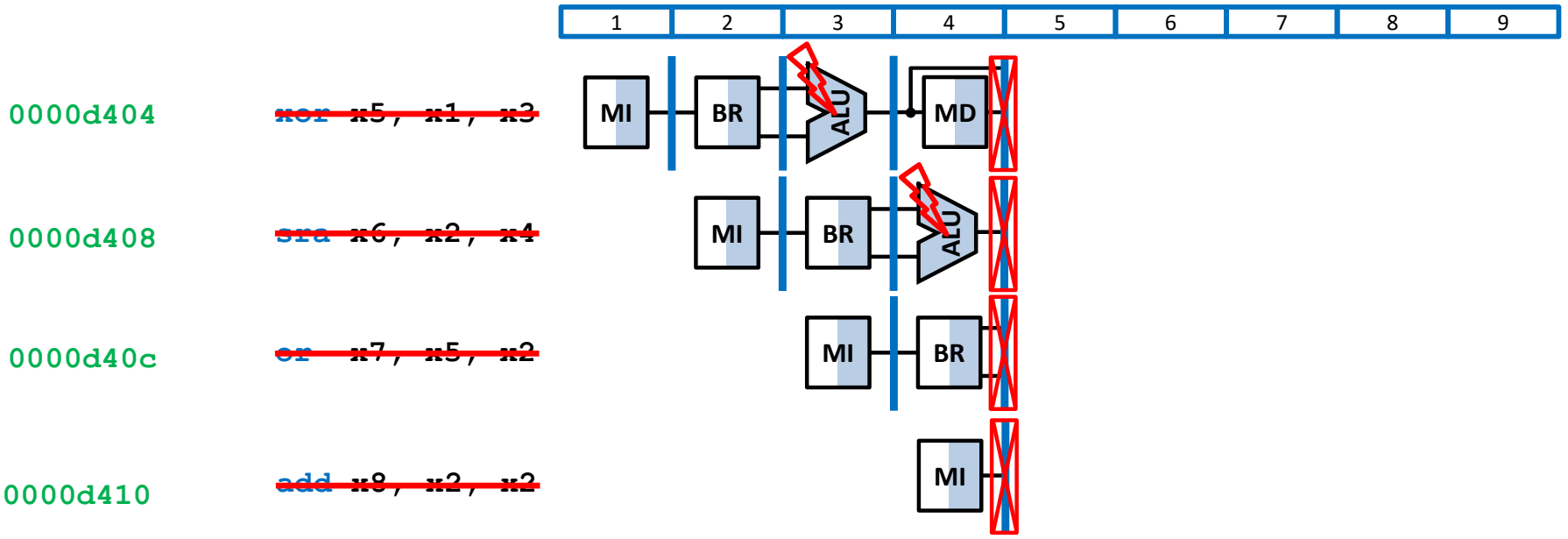
```
0000d404        xor x5, x1, x3
0000d408        sra x6, x2, x4
0000d40c        or  x7, x5, x2
0000d410        add x8, x2, x2
```



Procesador segmentado

Excepciones múltiples no simultáneas en orden

- Ídem si las excepciones no son simultáneas.
  - Ciclo 3: **`xor`** (está en EX) genera excepción.
  - Ciclo 4: **`sra`** (está en EX) también genera excepción, pero se descarta **`xor`** (está en MEM) y siguientes.
    - **`mepc`** = 0x0000d404 y **`cause`** = 2

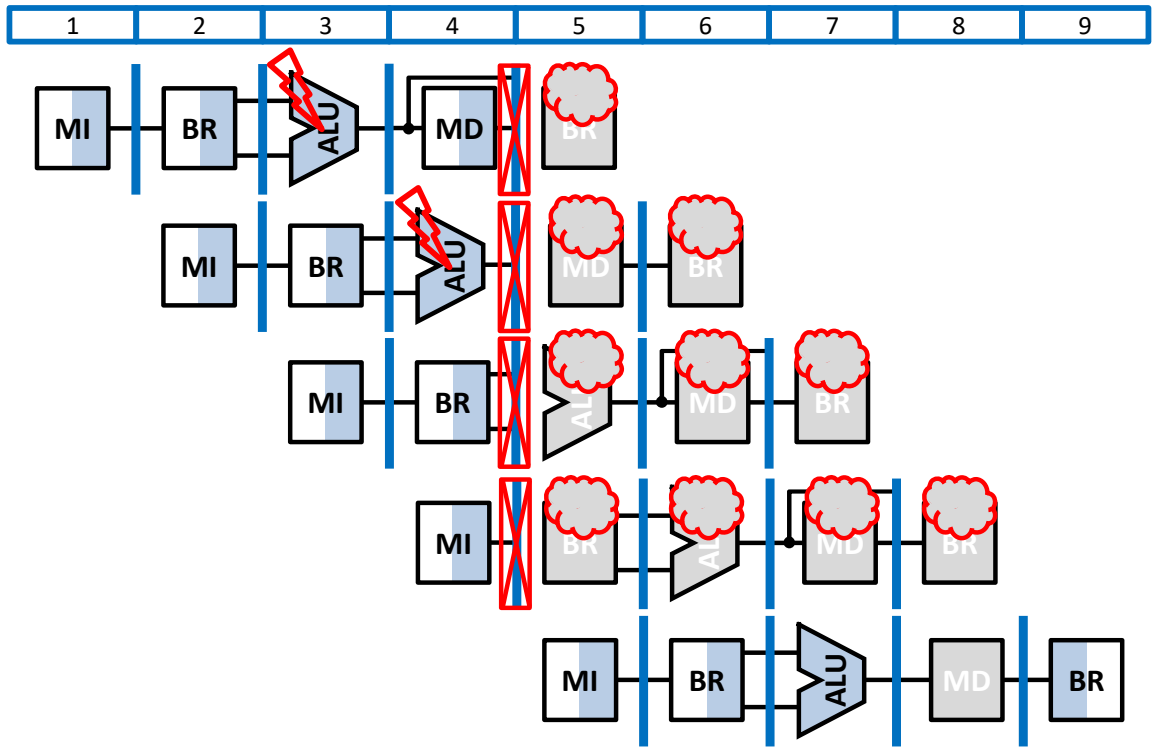


Procesador segmentado

Excepciones múltiples no simultáneas en orden

- Ídem si las excepciones no son simultáneas.
  - Ciclo 3: **`xor`** (está en EX) genera excepción.
  - Ciclo 4: **`sra`** (está en EX) también genera excepción, pero se descarta **`xor`** (está en MEM) y siguientes.
    - **`mepc`** = 0x0000d404 y **`cause`** = 2
  - Ciclo 5: se lanza la instrucción cuya dirección almacena **`mtvec`**.

```
0000d404    xor  x5, x1, x3
0000d408    sra  x6, x2, x4
0000d40c    or   x7, x5, x2
0000d410    add  x8, x2, x2
1c000000 RTE: add  sp, sp, -8
```

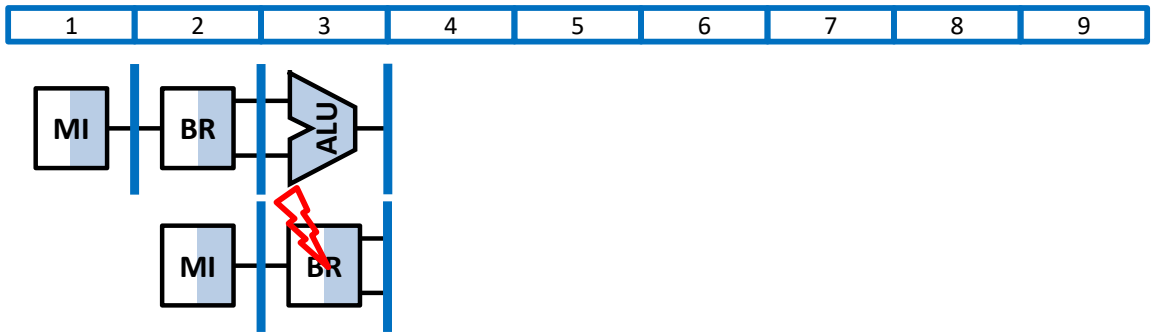


Procesador segmentado

Excepciones múltiples no simultáneas fuera de orden

- Ídem en el caso de excepciones fuera de orden:
  - Ciclo 3: **`mul`** (está en ID) genera excepción.

```
0000d404        lw   x2, 20(x1)
0000d408        mul  x6, x2, x4
```

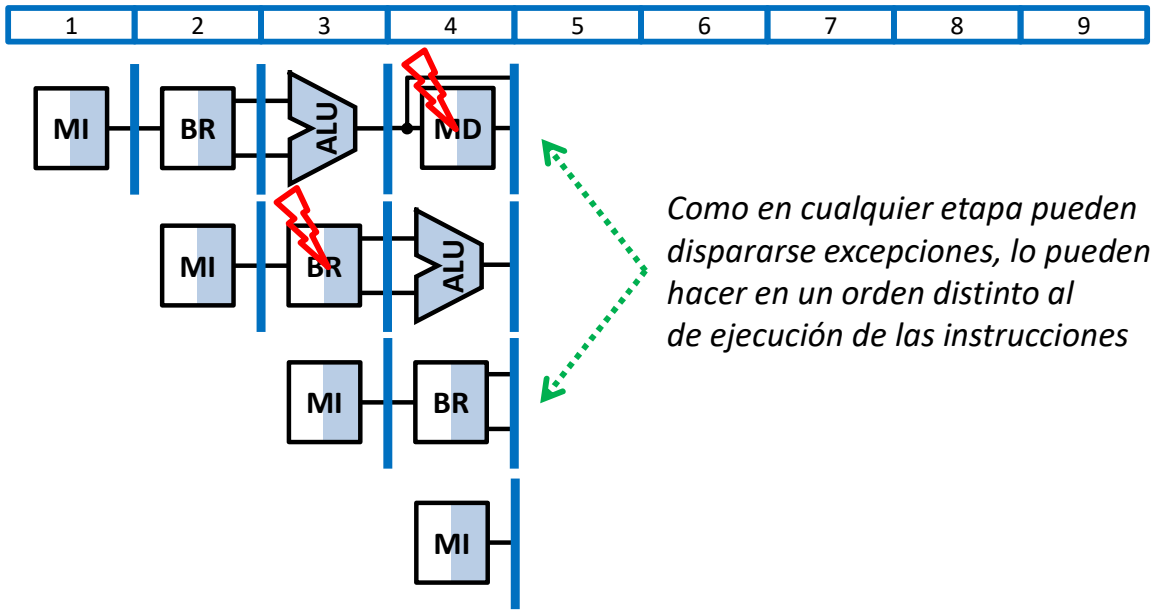


Procesador segmentado

Excepciones múltiples no simultáneas fuera de orden

- Ídem en el caso de excepciones fuera de orden:
  - Ciclo 3: **`mul`** (está en ID) genera excepción.
  - Ciclo 4: **`lw`** (está en MEM) genera excepción después que **`mul`** aún siendo una instrucción previa.

```
0000d404        lw   x2, 20(x1)
0000d408        mul  x6, x2, x4
0000d40c        or   x7, x5, x2
0000d410        add  x8, x2, x2
```

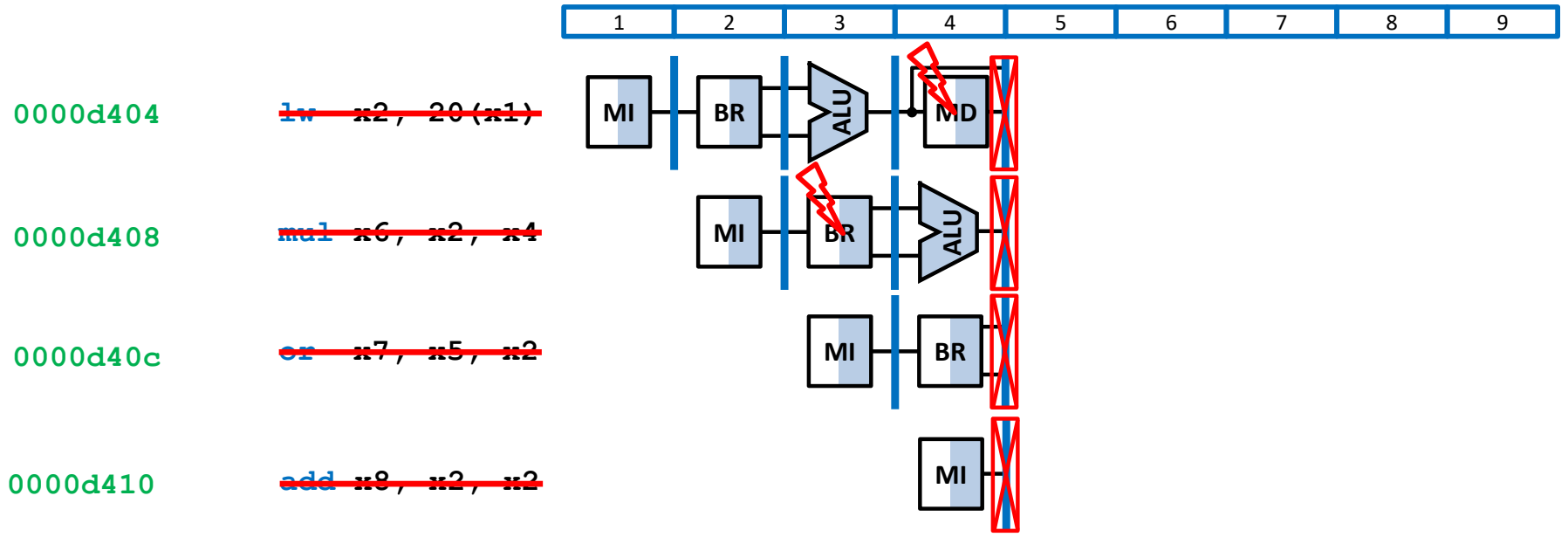


Como en cualquier etapa pueden dispararse excepciones, lo pueden hacer en un orden distinto al de ejecución de las instrucciones

Procesador segmentado

Excepciones múltiples no simultáneas fuera de orden

- Ídem en el caso de excepciones fuera de orden:
  - Ciclo 3: **`mul`** (está en ID) genera excepción.
  - Ciclo 4: **`lw`** (está en MEM) genera excepción después que **`mul`** aún siendo una instrucción previa. Se descarta **`lw`** y siguientes.
    - **`mepc`** = 0x0000d404 y **`cause`** = 4

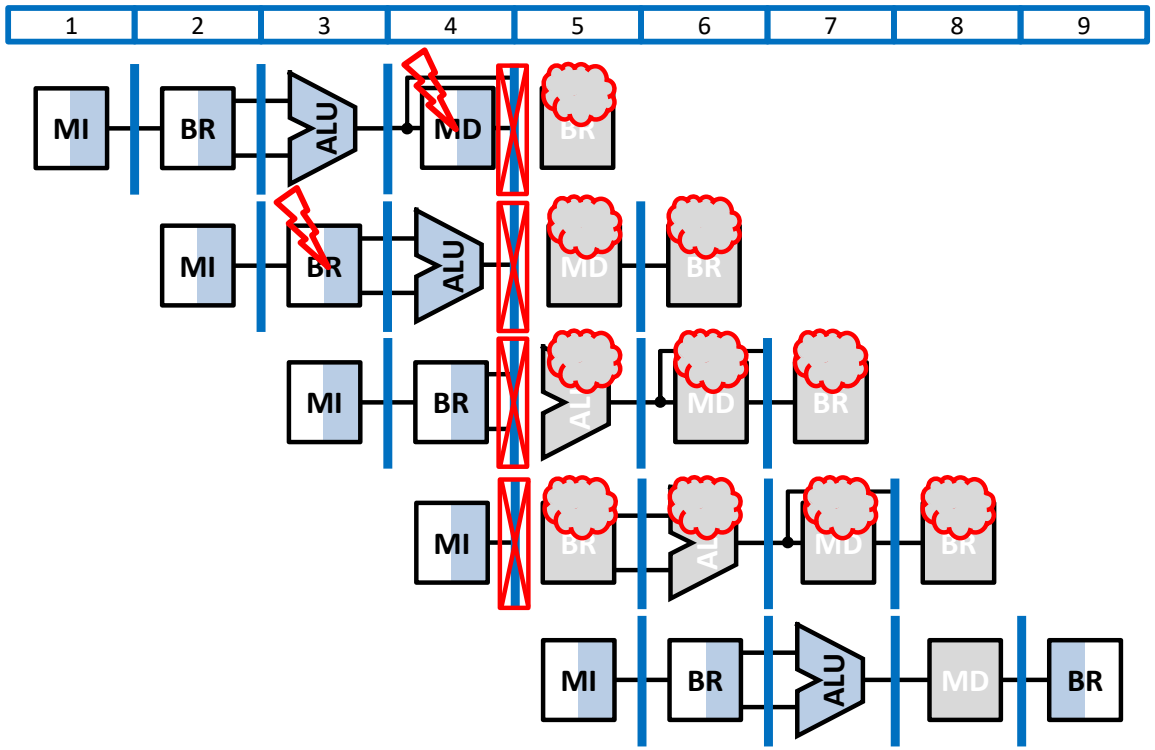


Procesador segmentado

Excepciones múltiples no simultáneas fuera de orden

- Ídem en el caso de excepciones fuera de orden:
  - Ciclo 3: **`mul`** (está en ID) genera excepción.
  - Ciclo 4: **`lw`** (está en MEM) genera excepción después que **`mul`** aún siendo una instrucción previa. Se descarta **`lw`** y siguientes.
    - **`mepc`** = 0x0000d404 y **`cause`** = 4
  - Ciclo 5: se lanza la instrucción cuya dirección almacena **`mtvec`**.

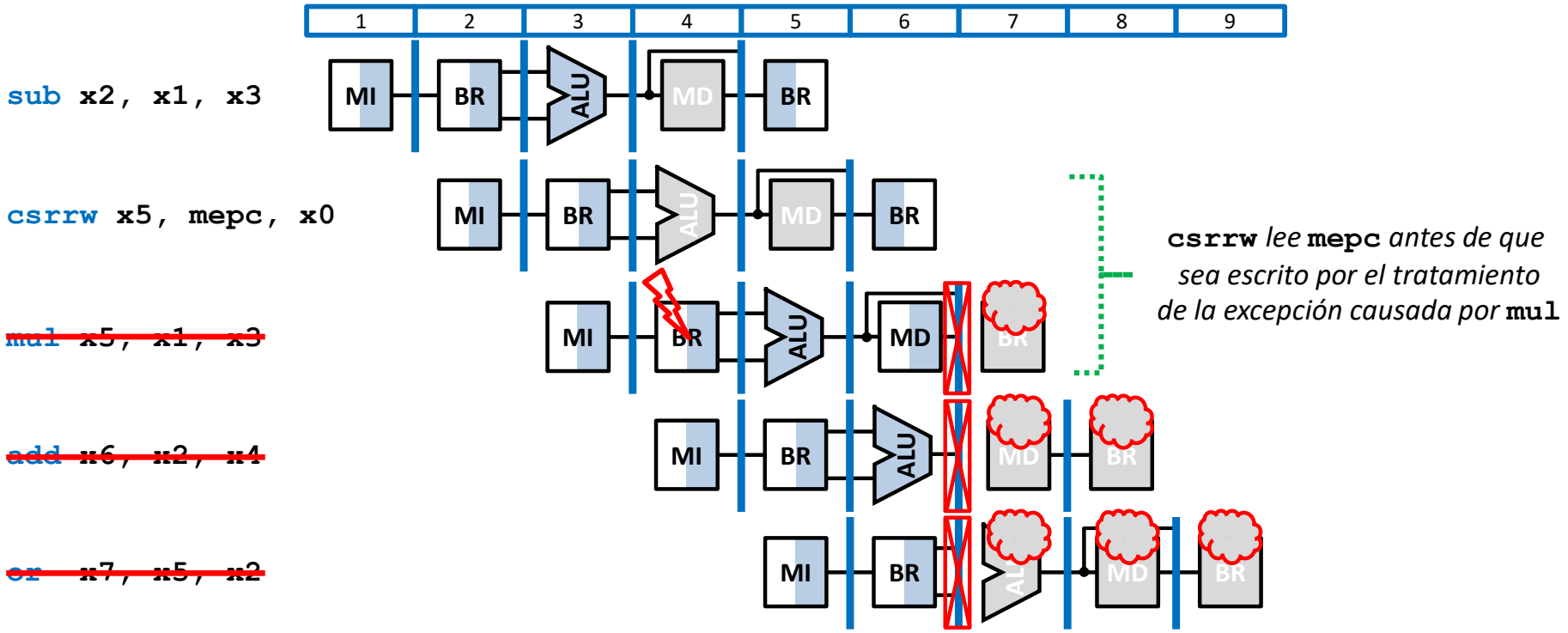
```
0000d404   lw  x2, 20(x1)
0000d408   mul x6, x2, x4
0000d40c   or  x7, x5, x2
0000d410   add x8, x2, x2

1c000000 RTE: add sp, sp, -8
```



Procesador segmentado

Consistencia de CSR

- Además, dado que el tratamiento de la excepción se realiza en la etapa MEM de la instrucción desencadenante se asegura que:
  - Ni esta instrucción ni posteriores modifican el BR o la memoria.
  - Todas las instrucciones anteriores terminan normalmente.
    - Si hubieran leído los registros **`mepc`** o **`cause`** habrían leído los valores previos.



Procesador segmentado

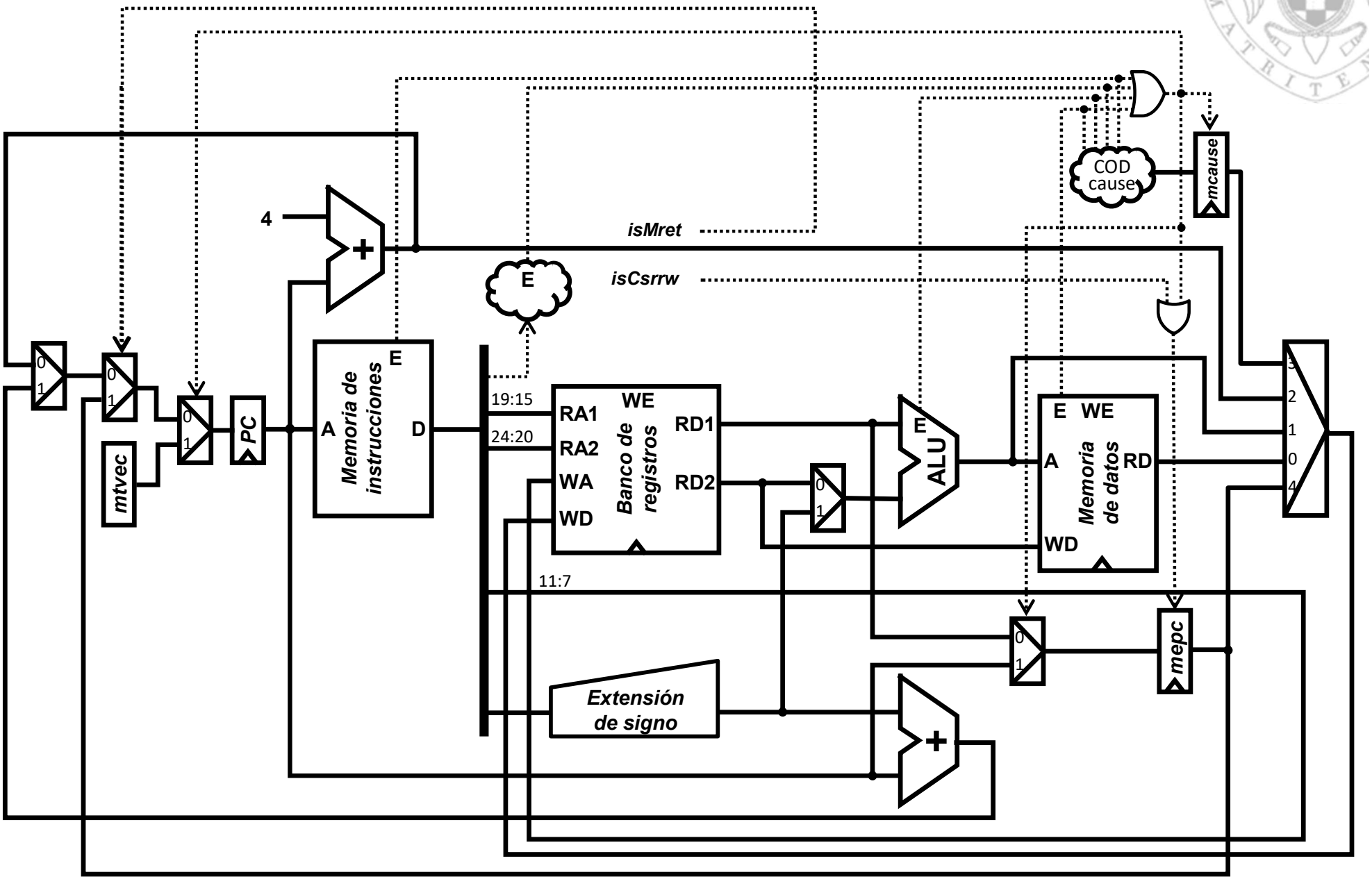
Excepciones precisas vs. imprecisas

- Este mecanismo de tratamiento se conoce como excepciones precisas y replica lo que ocurre en un procesador no segmentado, ya que:
  - Se trata siempre primero la excepción generada por la instrucción que se encuentra antes en el código.
  - Se completan las instrucciones anteriores a la que dispara la excepción.
  - Dejan sin efecto la instrucción que genera la excepción y todas las lanzadas que se encuentran después en el código.
  - Se almacenan con exactitud la dirección que la instrucción que generó la excepción y su causa.

- Existen procesadores segmentados en donde no todas las condiciones anteriores se cumplen, se dice que tienen excepciones imprecisas.

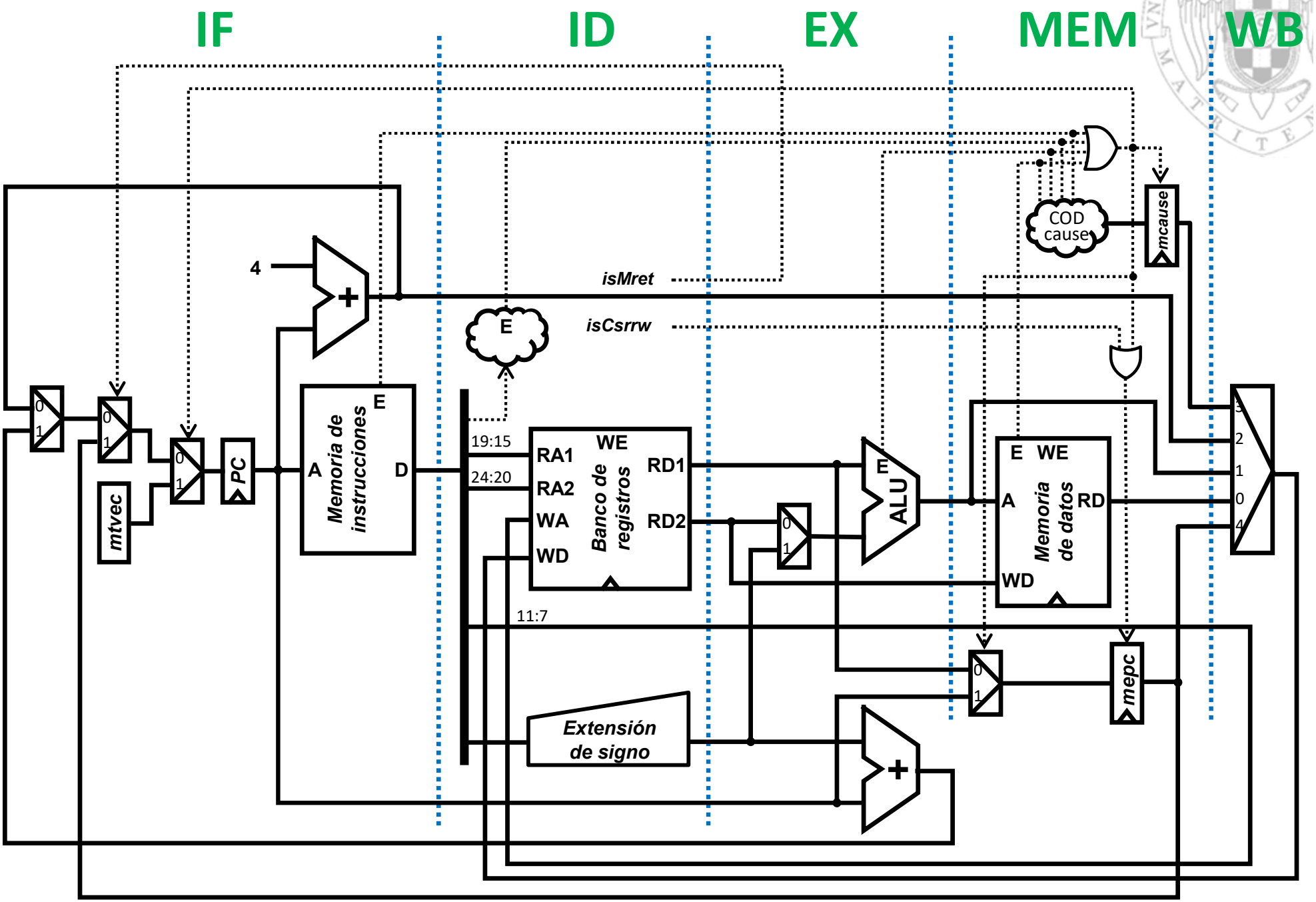
Procesador segmentado

Ruta de datos monociclo + gestión de excepciones (i)



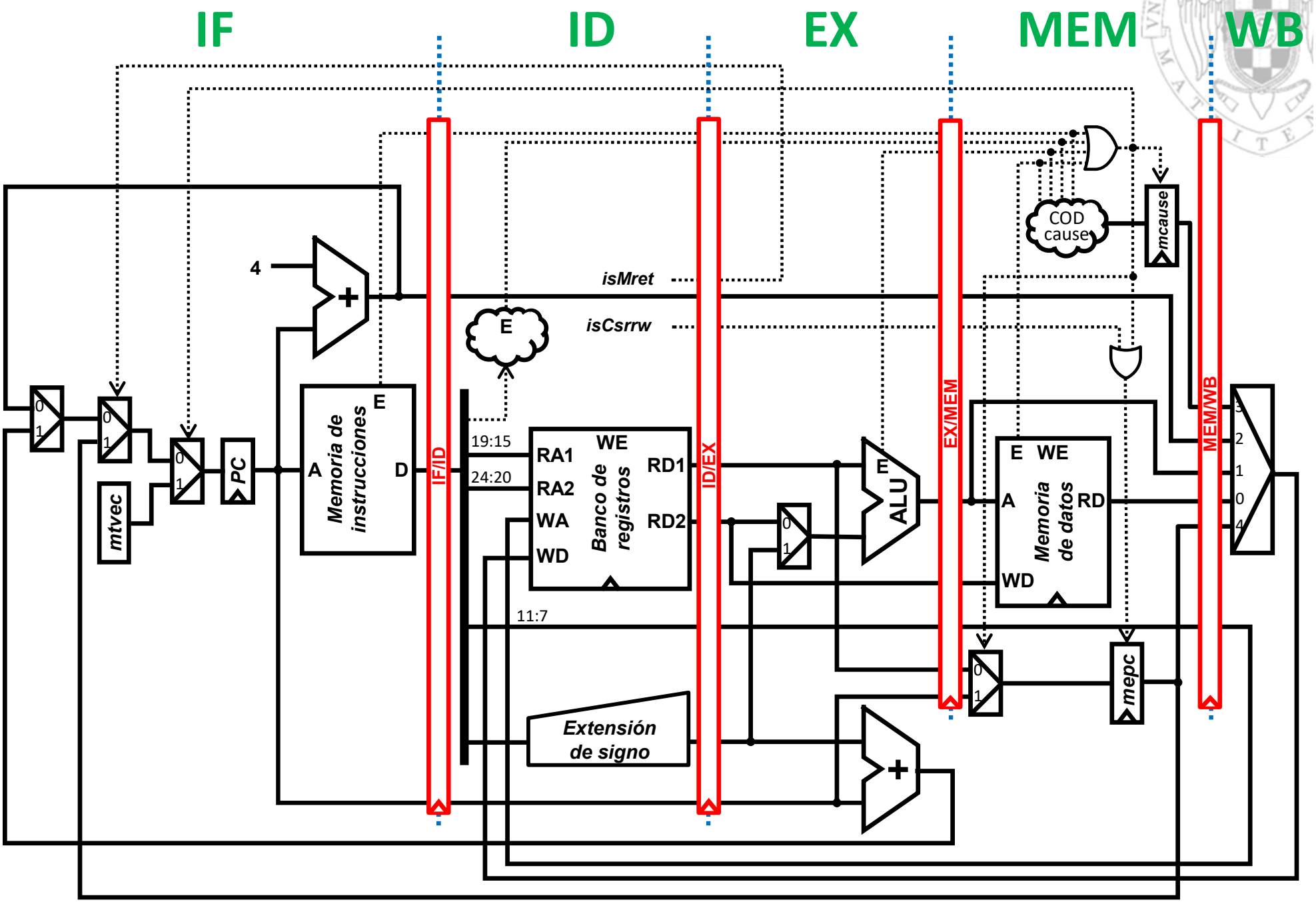
Procesador segmentado

Ruta de datos monociclo + gestión de excepciones (ii)



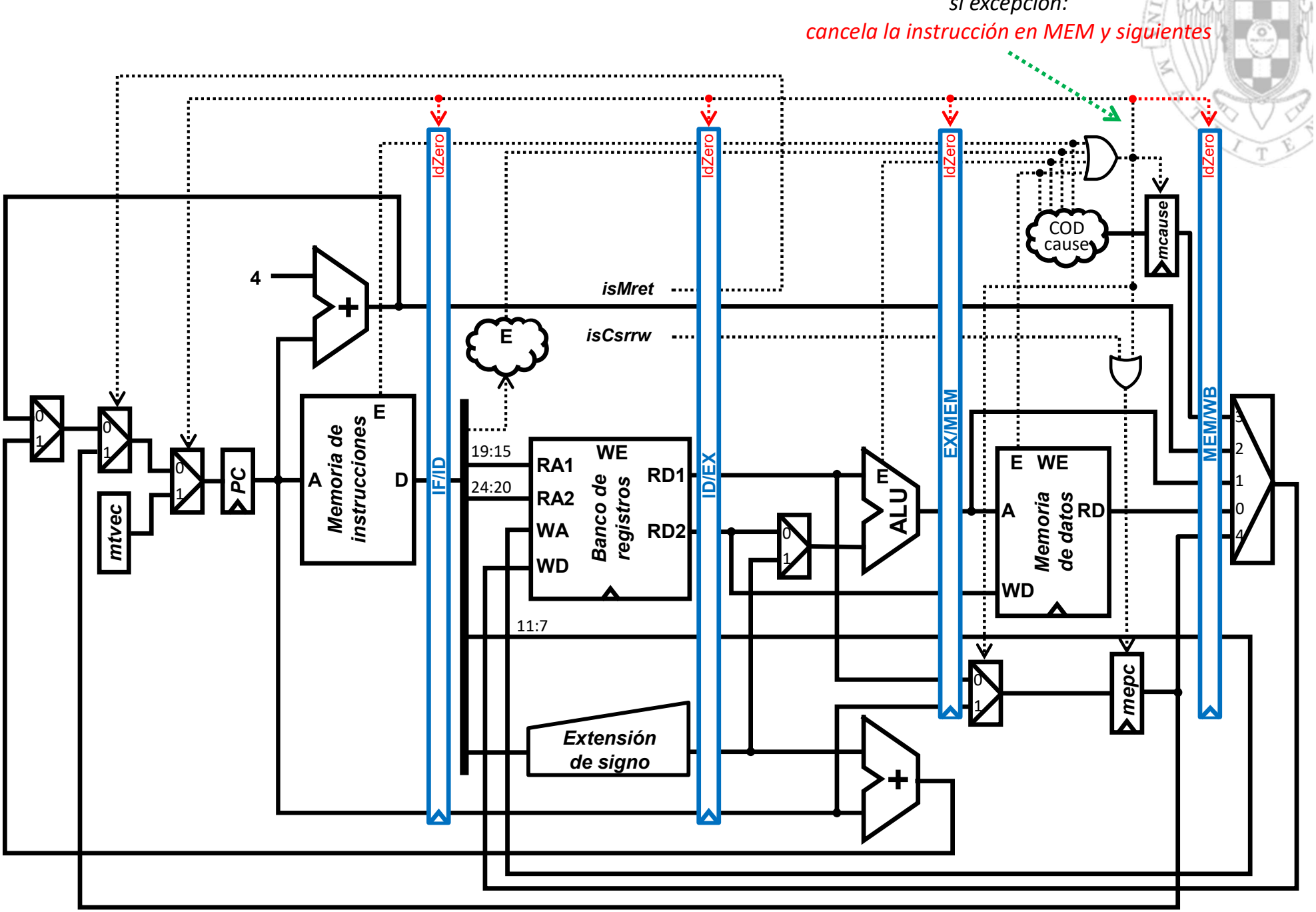
Procesador segmentado

Ruta de datos segmentada + gestión de excepciones (i)



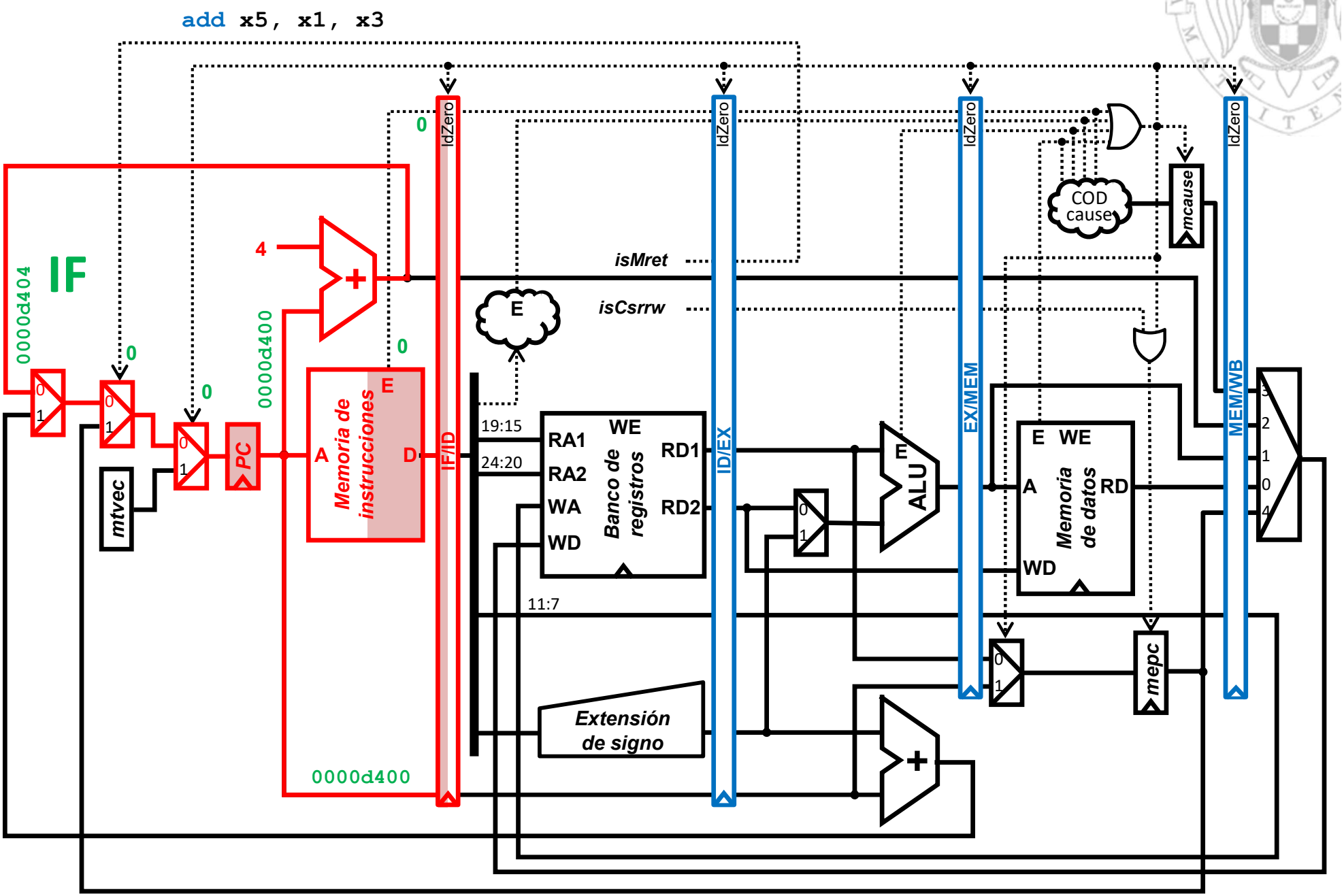
Procesador segmentado

Ruta de datos segmentada + gestión de excepciones (ii)



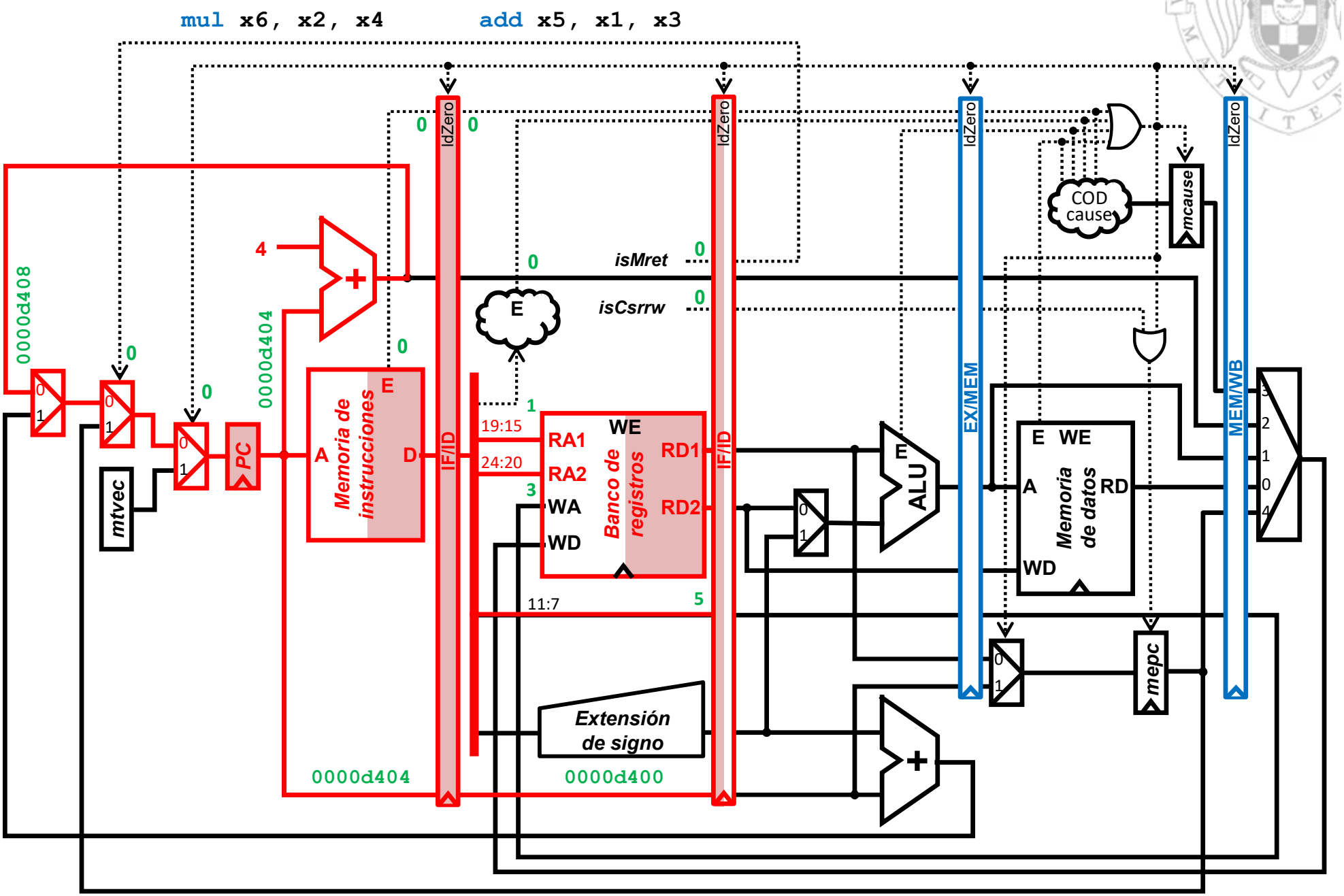
Procesador segmentado

Excepción por instrucción ilegal: 1er. ciclo



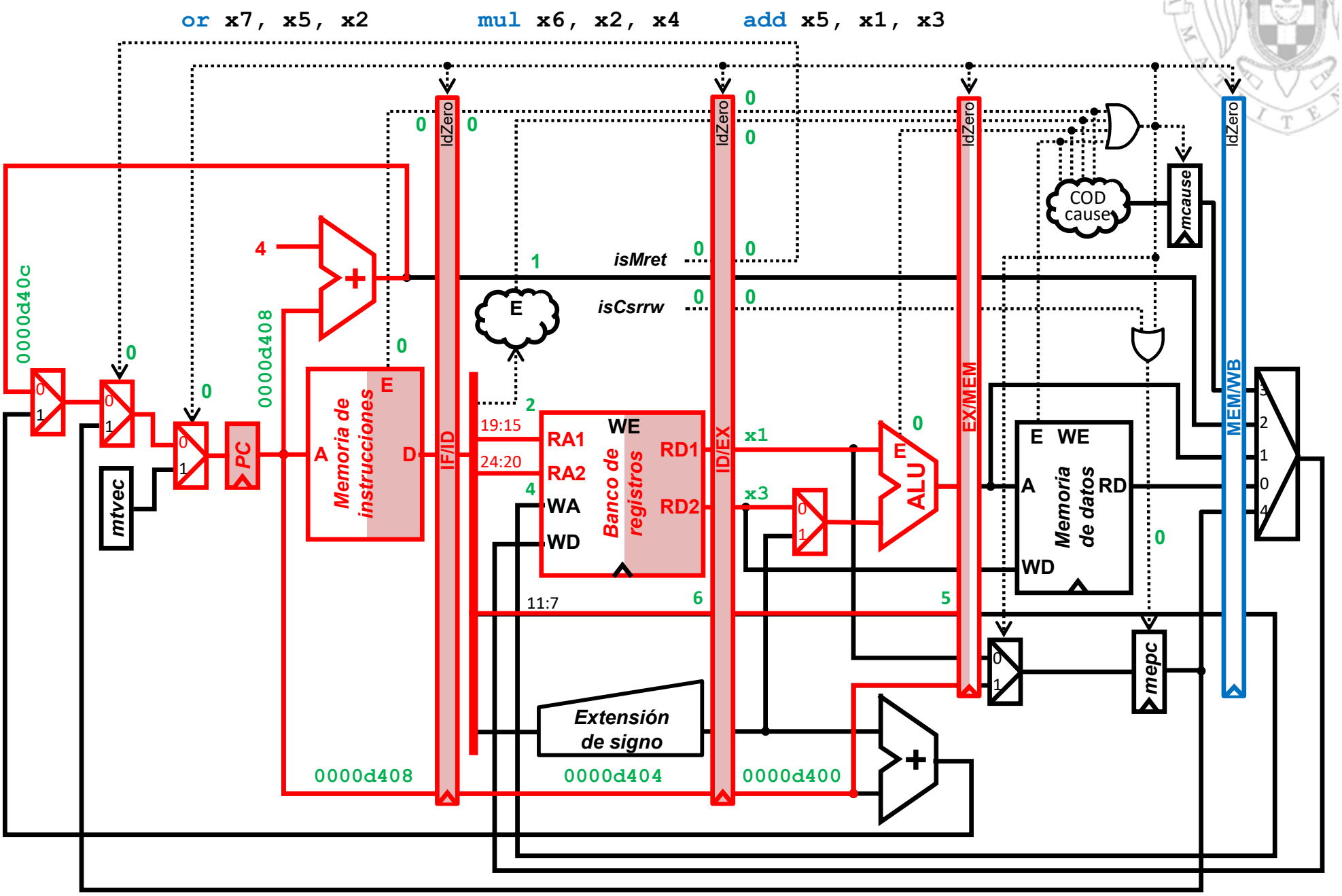
Procesador segmentado

Excepción por instrucción ilegal: 2do. ciclo



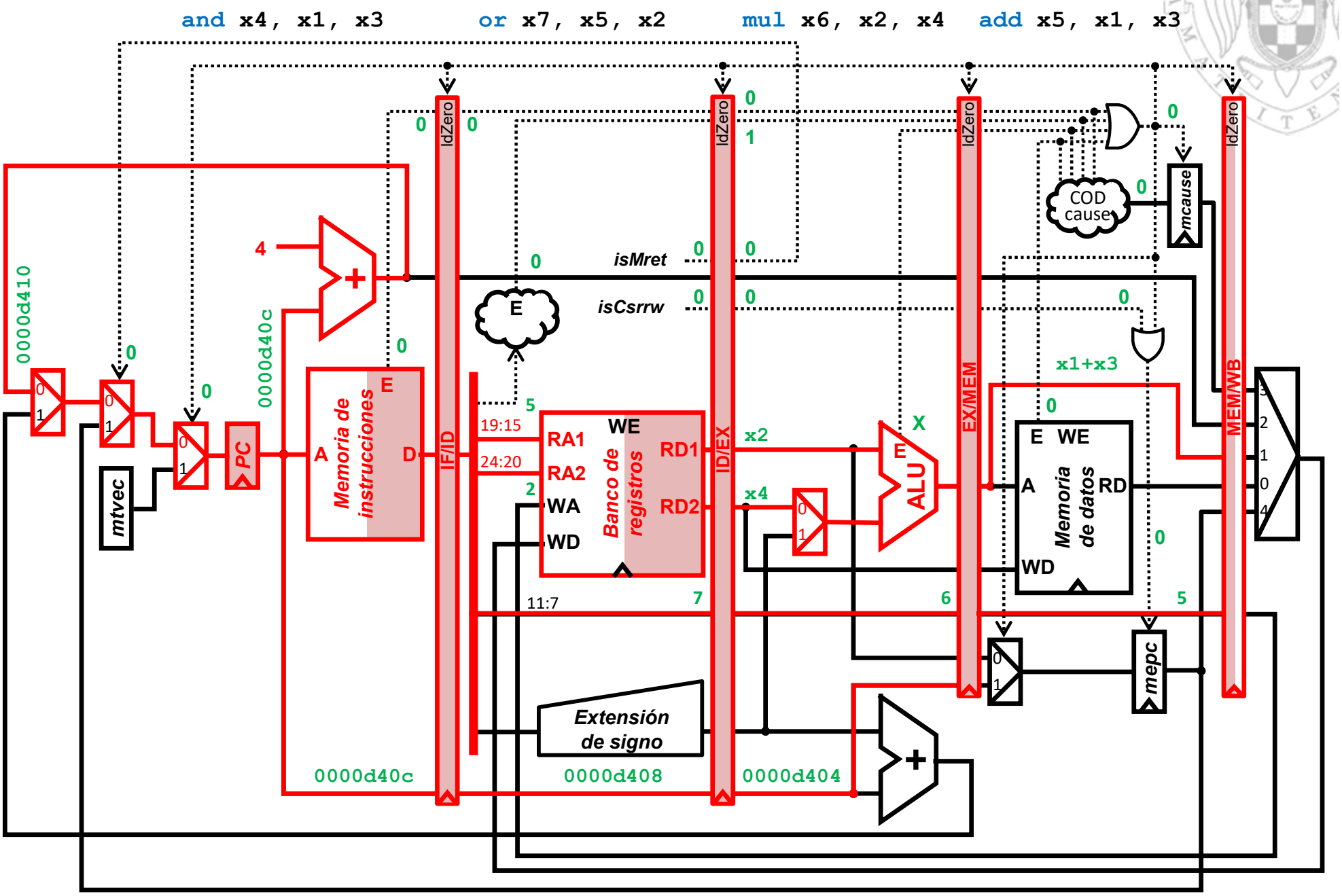
Procesador segmentado

Excepción por instrucción ilegal: 3er. ciclo



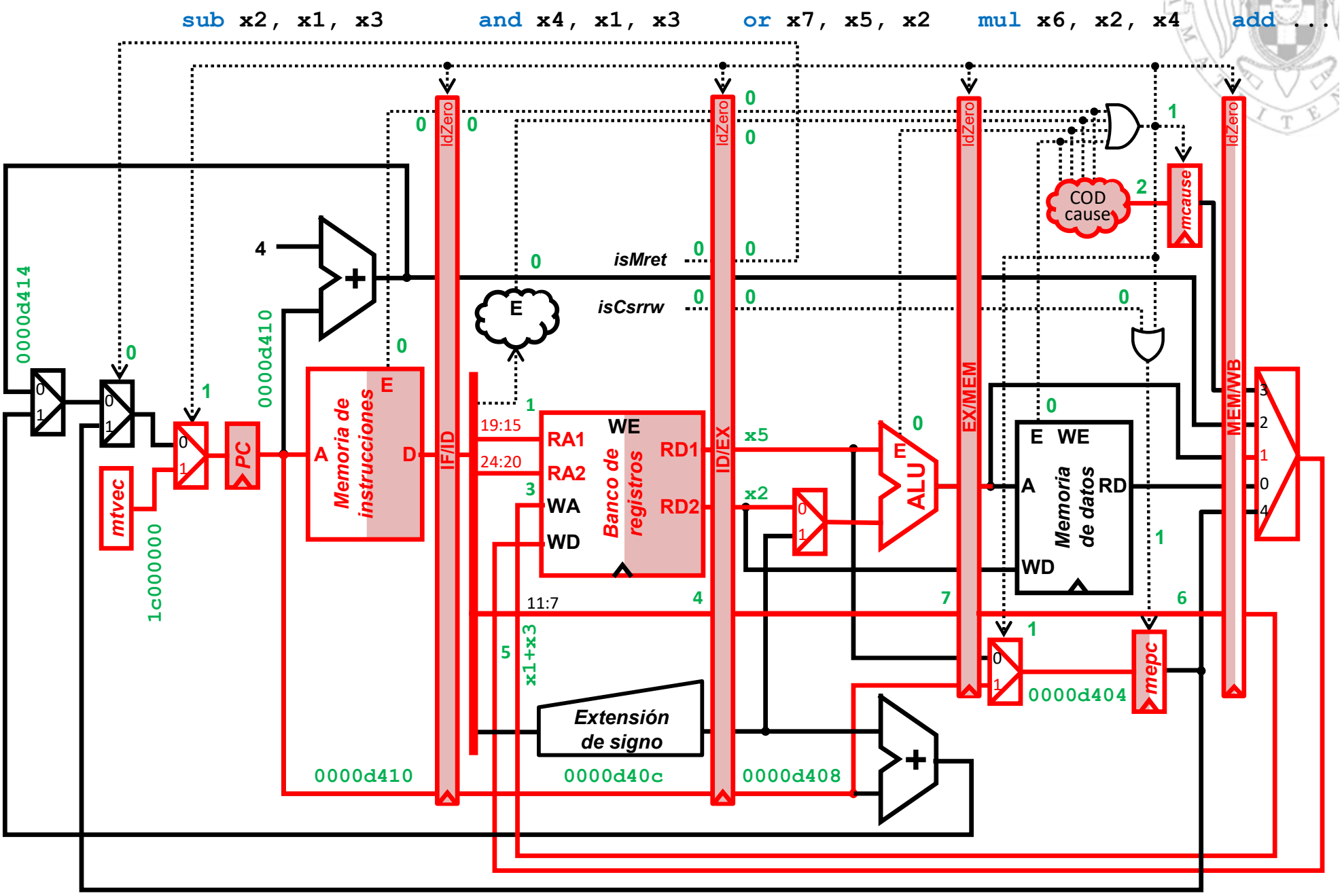
Procesador segmentado

Excepción por instrucción ilegal: 4o. ciclo



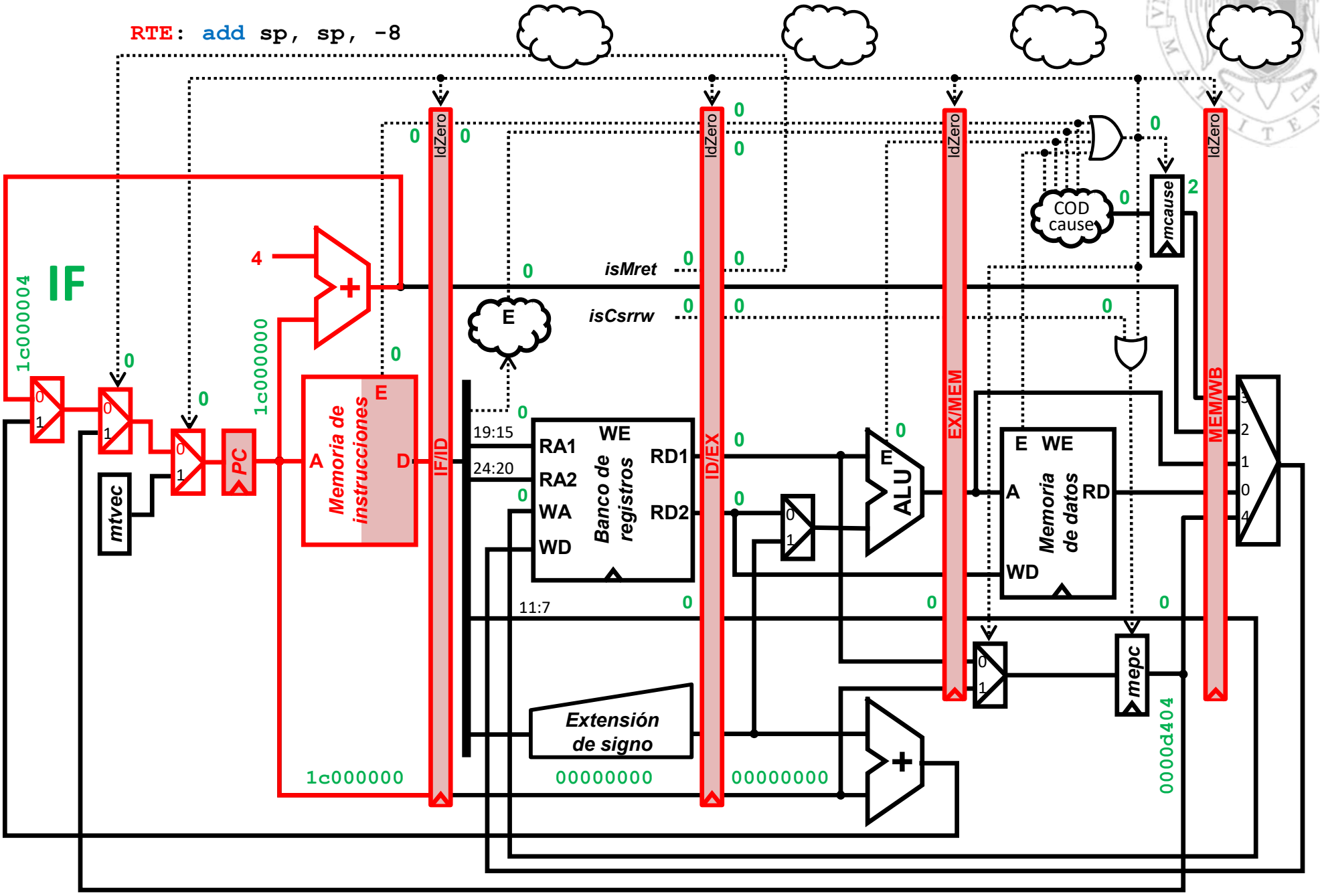
Procesador segmentado

Excepción por instrucción ilegal: 5o. ciclo



Procesador segmentado

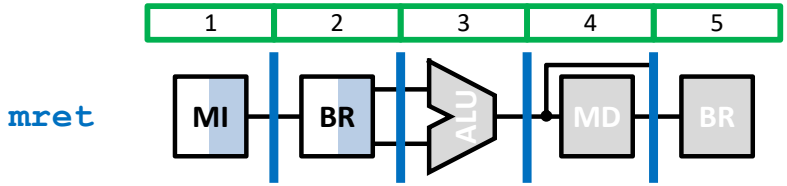
Excepción por instrucción ilegal: 6o. ciclo



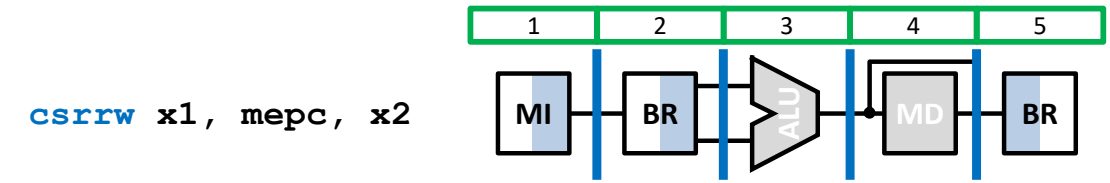
Procesador segmentado

Nuevas instrucciones

- La instrucción **`mret`** se ejecuta en 5 ciclos y como cualquier instrucción de salto:
  - Sin usar la memoria en la etapa MEM ni el BR en la etapa WB.
  - Tomando el salto la etapa EX (pero sin necesidad de usar la ALU).

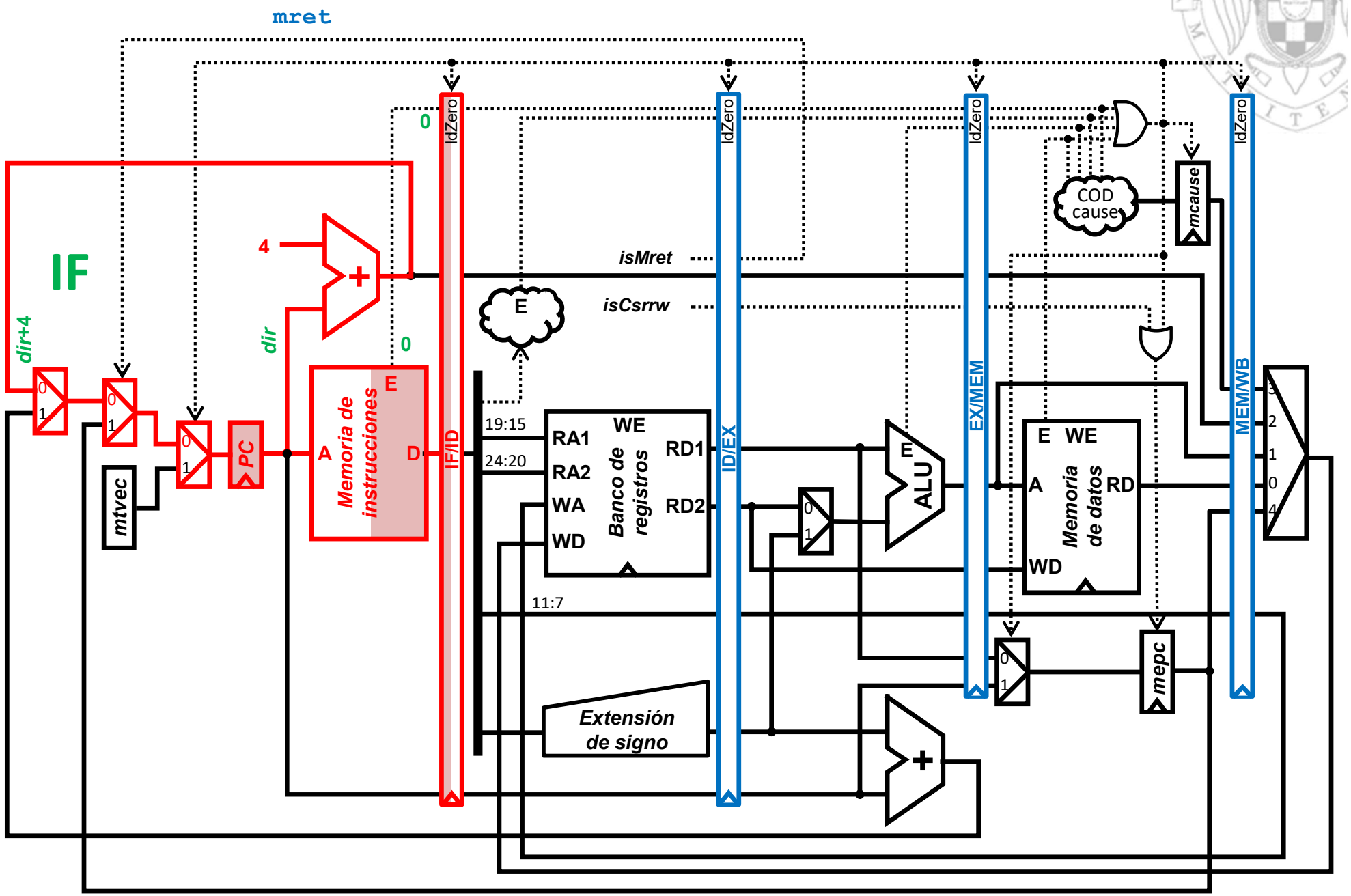


- La instrucción **`csrrw`** se ejecuta en 5 ciclos:
  - Sin usar la ALU en la etapa EX ni la memoria en la etapa MEM.
  - Como cualquier otra instrucción, lee el BR en ID y lo escribe en WB.
  - El CSR lo actualiza en la etapa MEM.



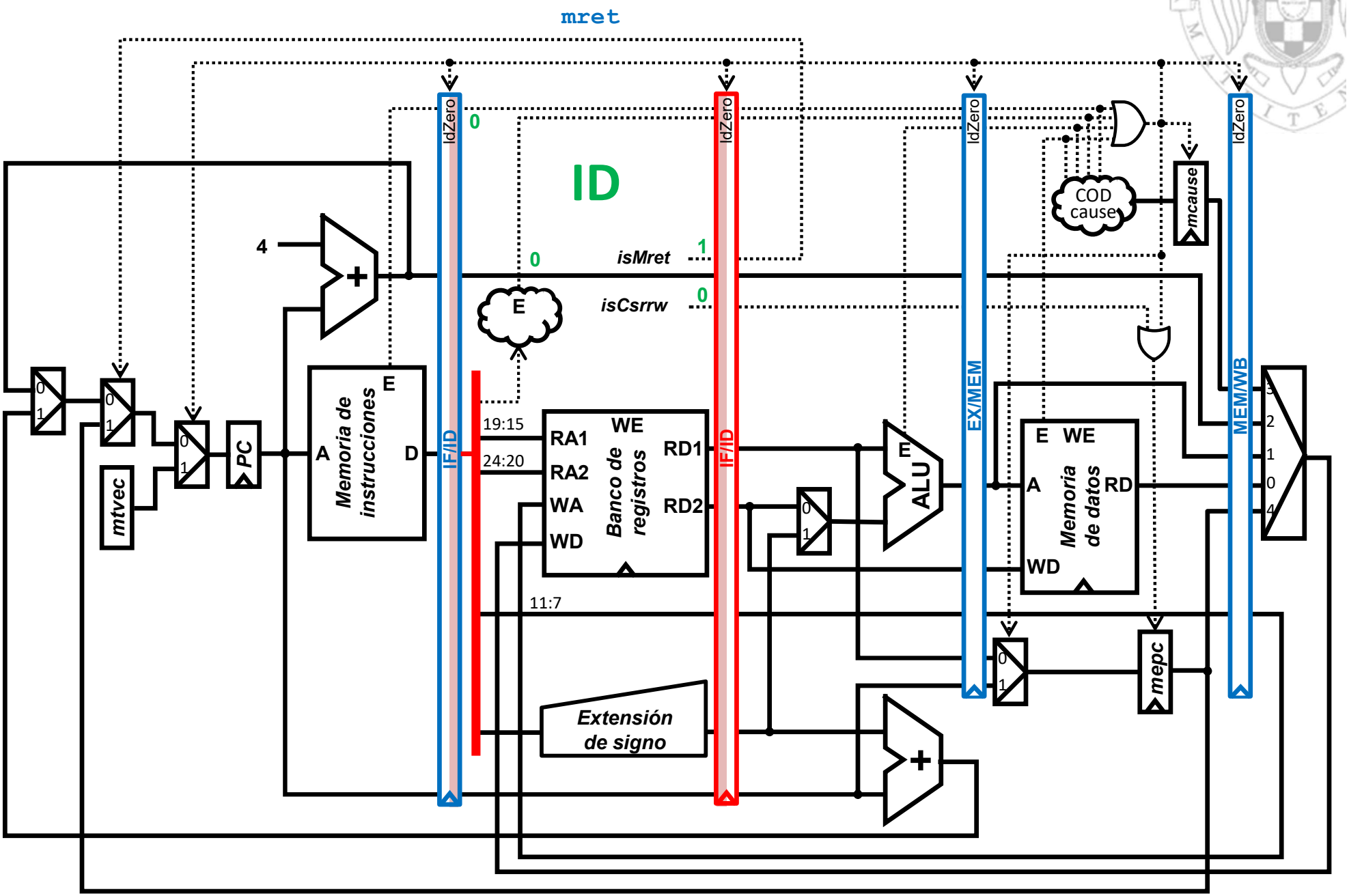
Procesador segmentado

Instrucción `mret`: etapa IF



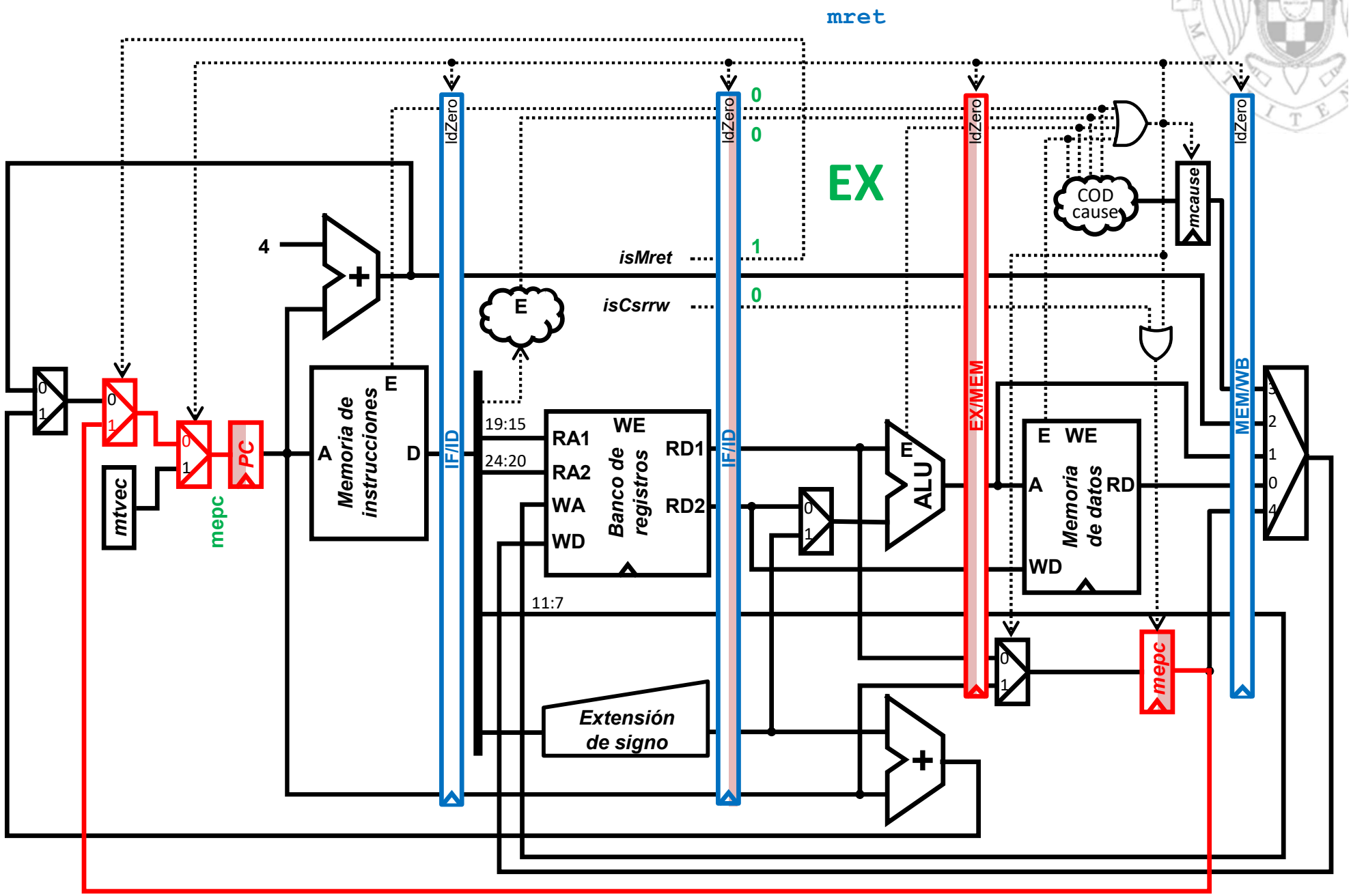
Procesador segmentado

Instrucción `mret`: etapa ID



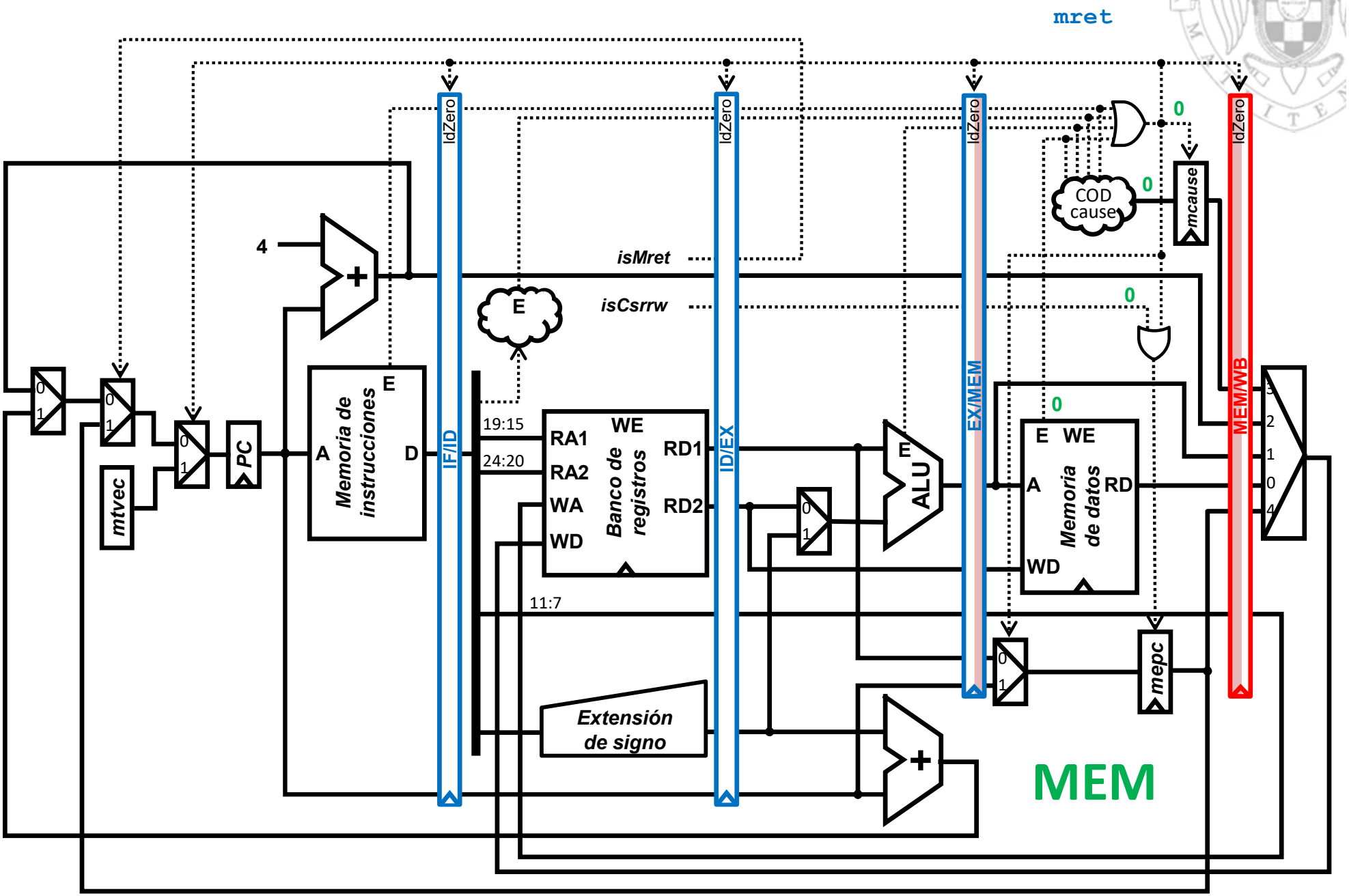
Procesador segmentado

Instrucción `mret`: etapa EX



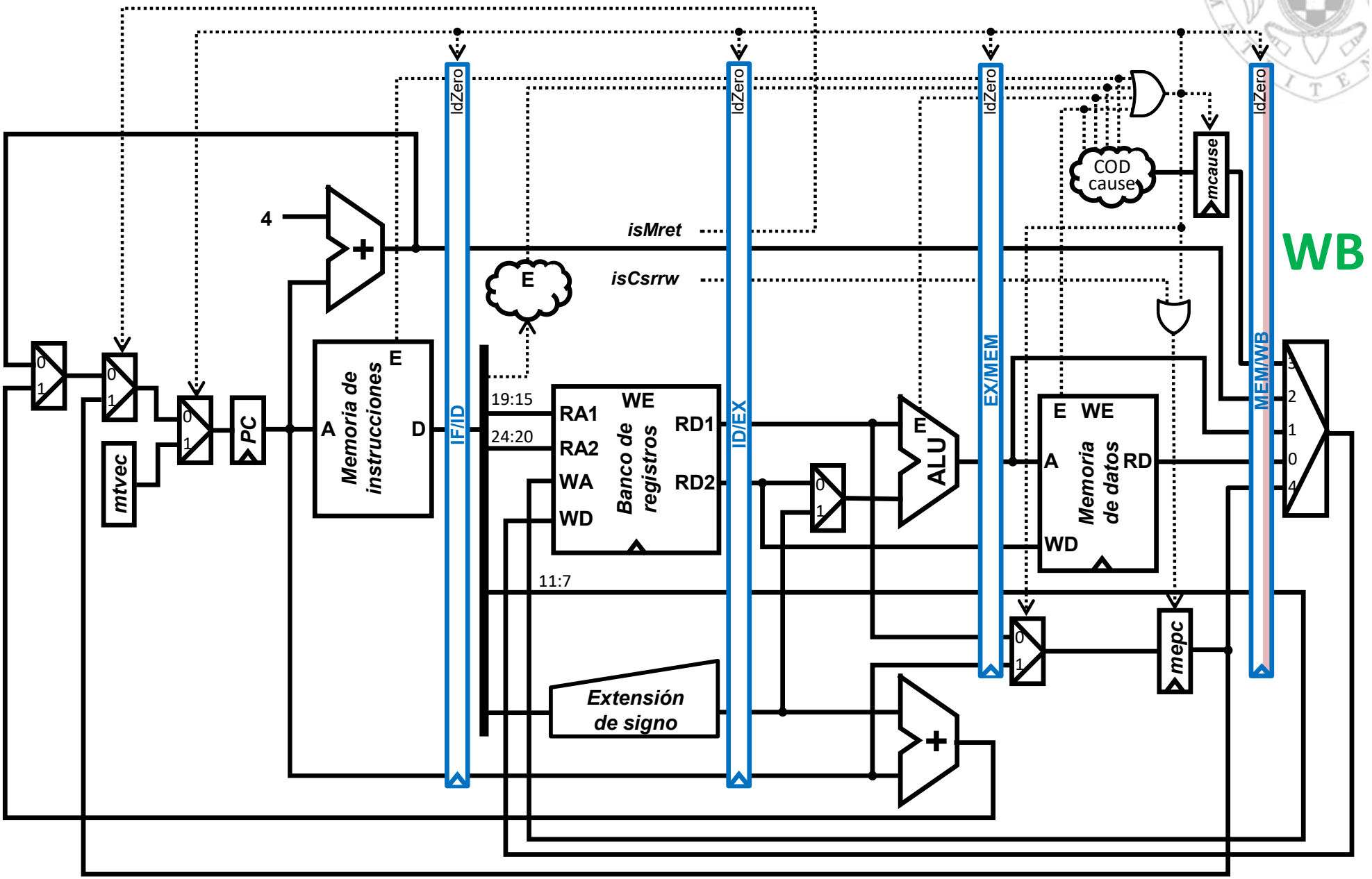
Procesador segmentado

Instrucción `mret`: etapa MEM



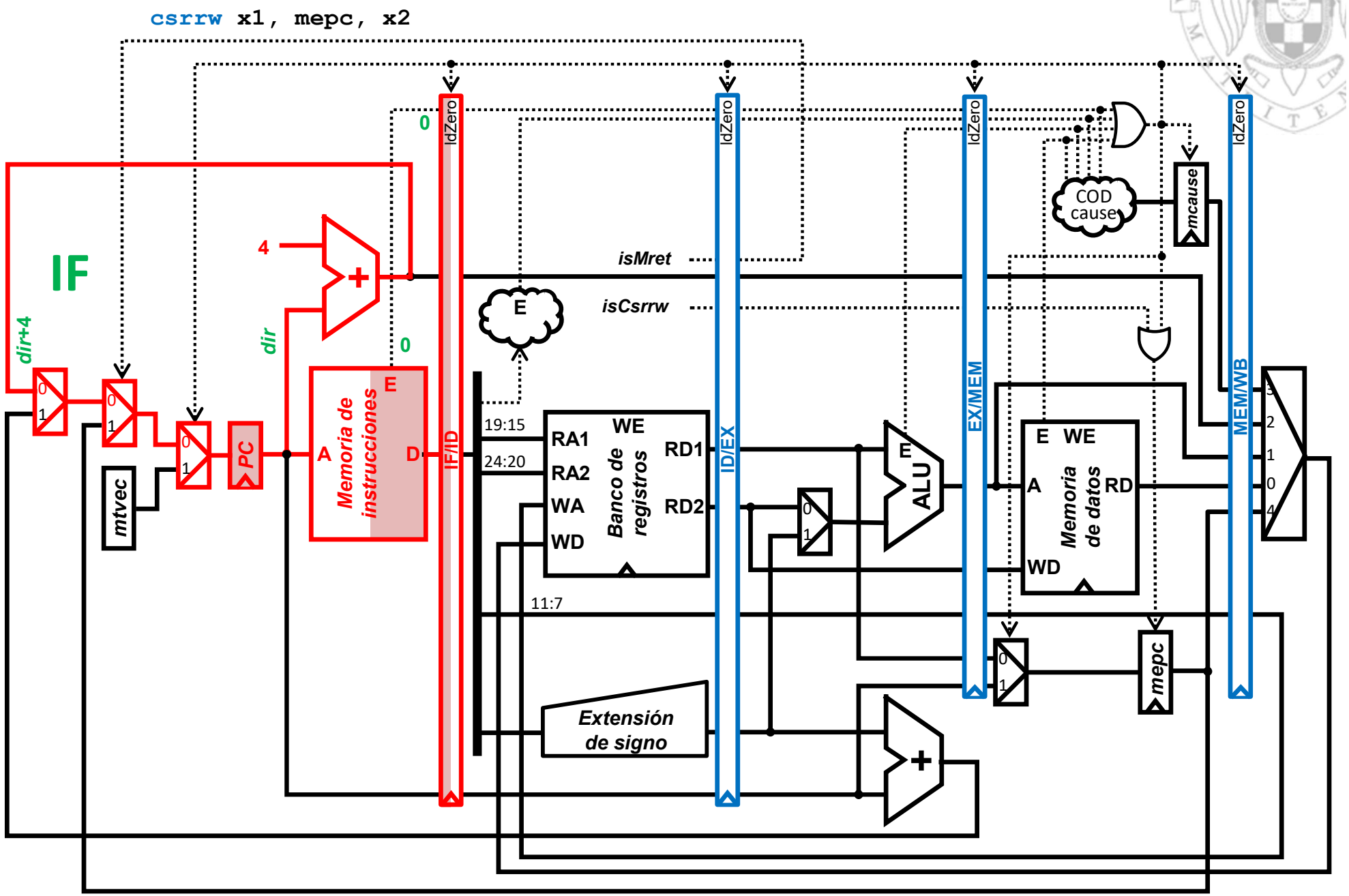
Procesador segmentado

Instrucción `mret`: etapa WB



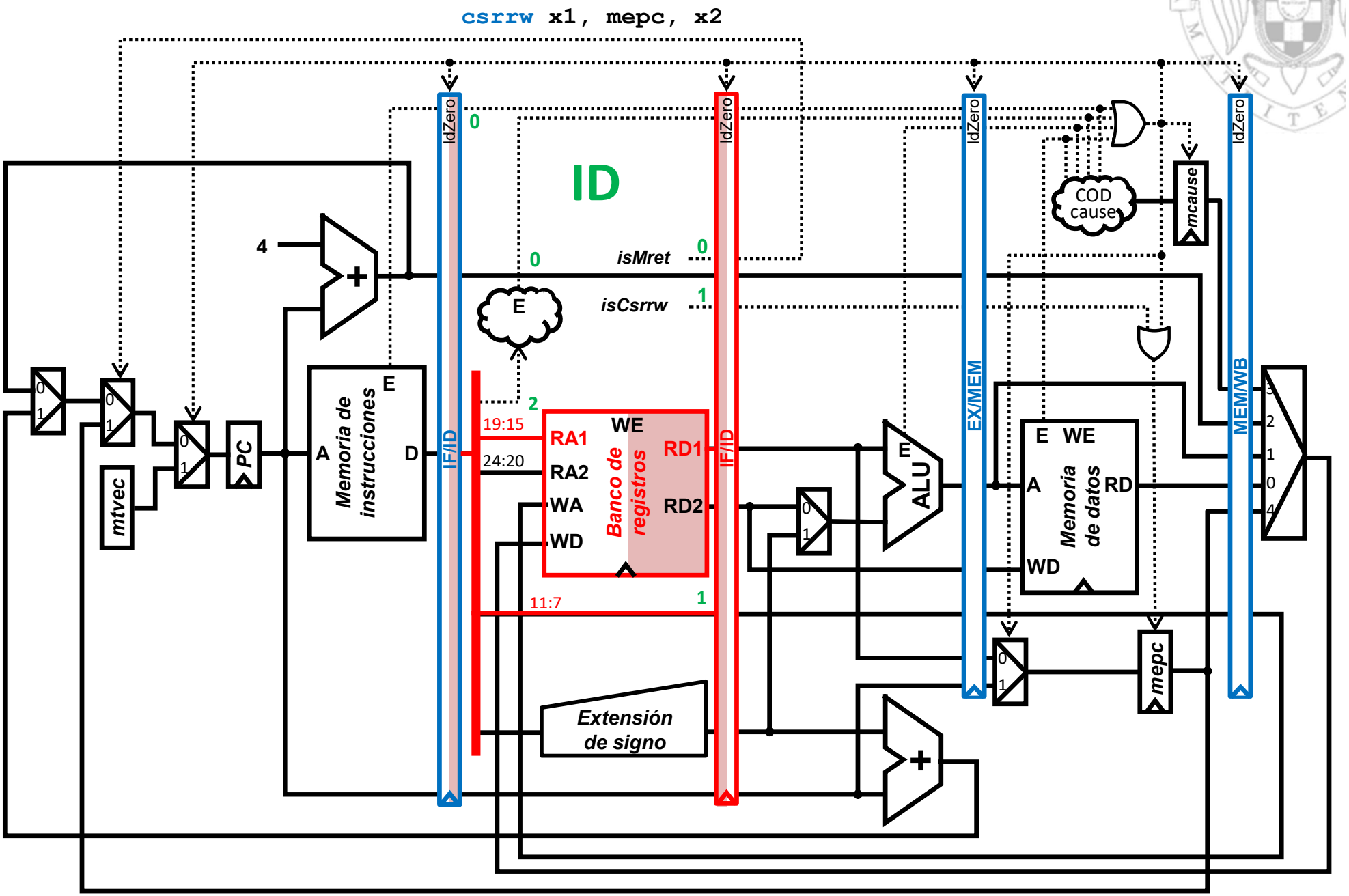
Procesador segmentado

Instrucción `csrrw`: etapa IF



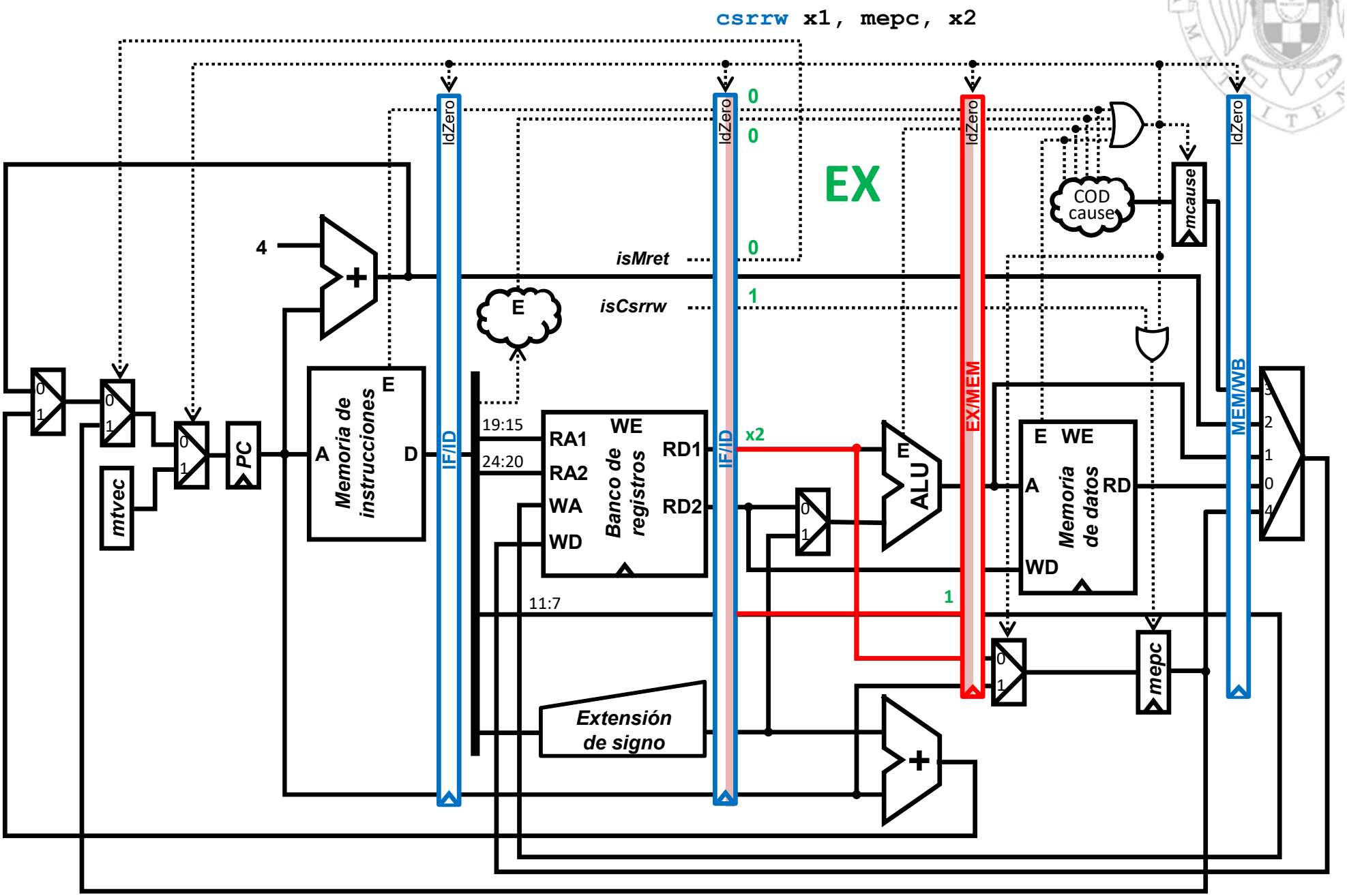
Procesador segmentado

Instrucción `csrrw`: etapa ID



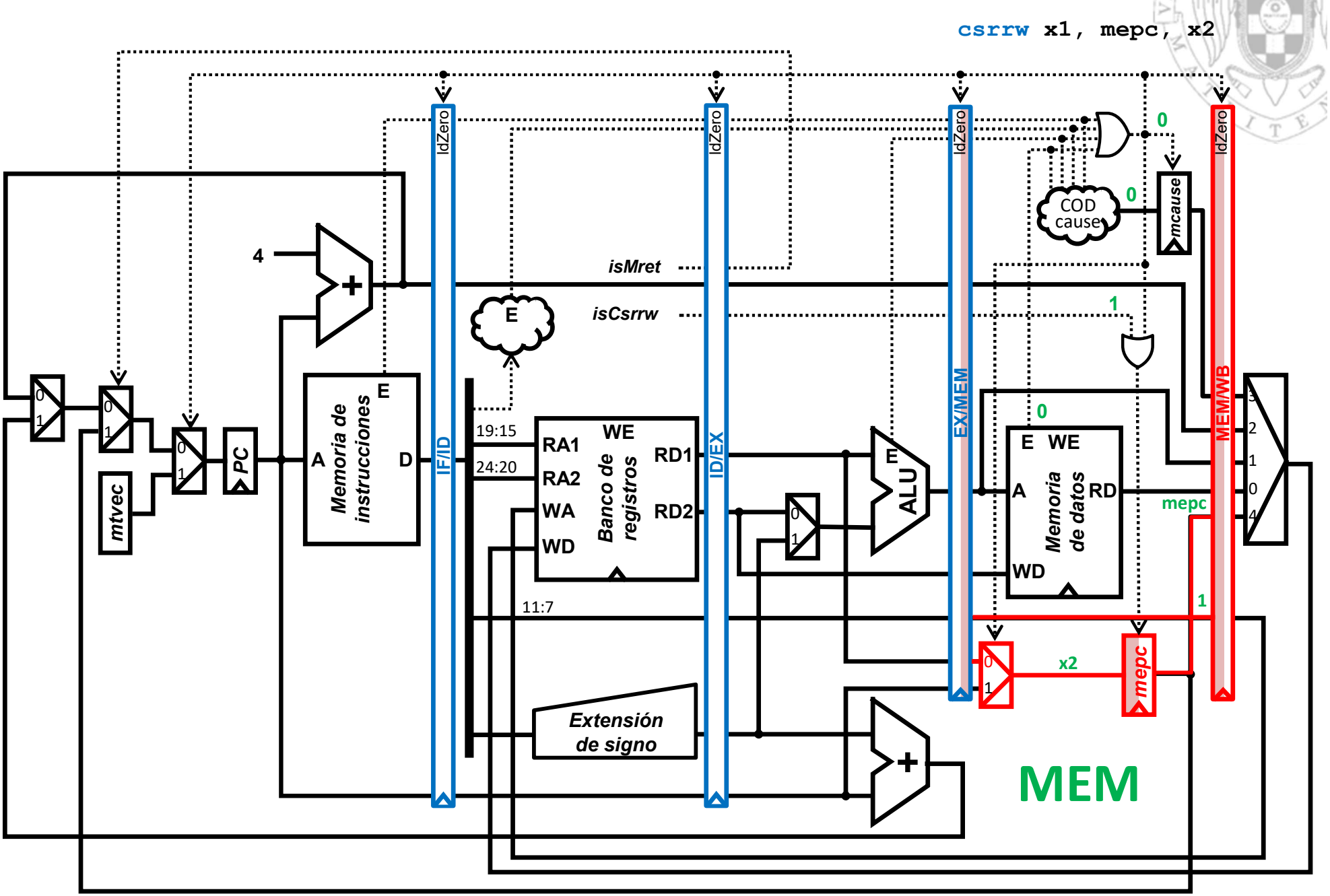
Procesador segmentado

Instrucción `csrrw`: etapa EX



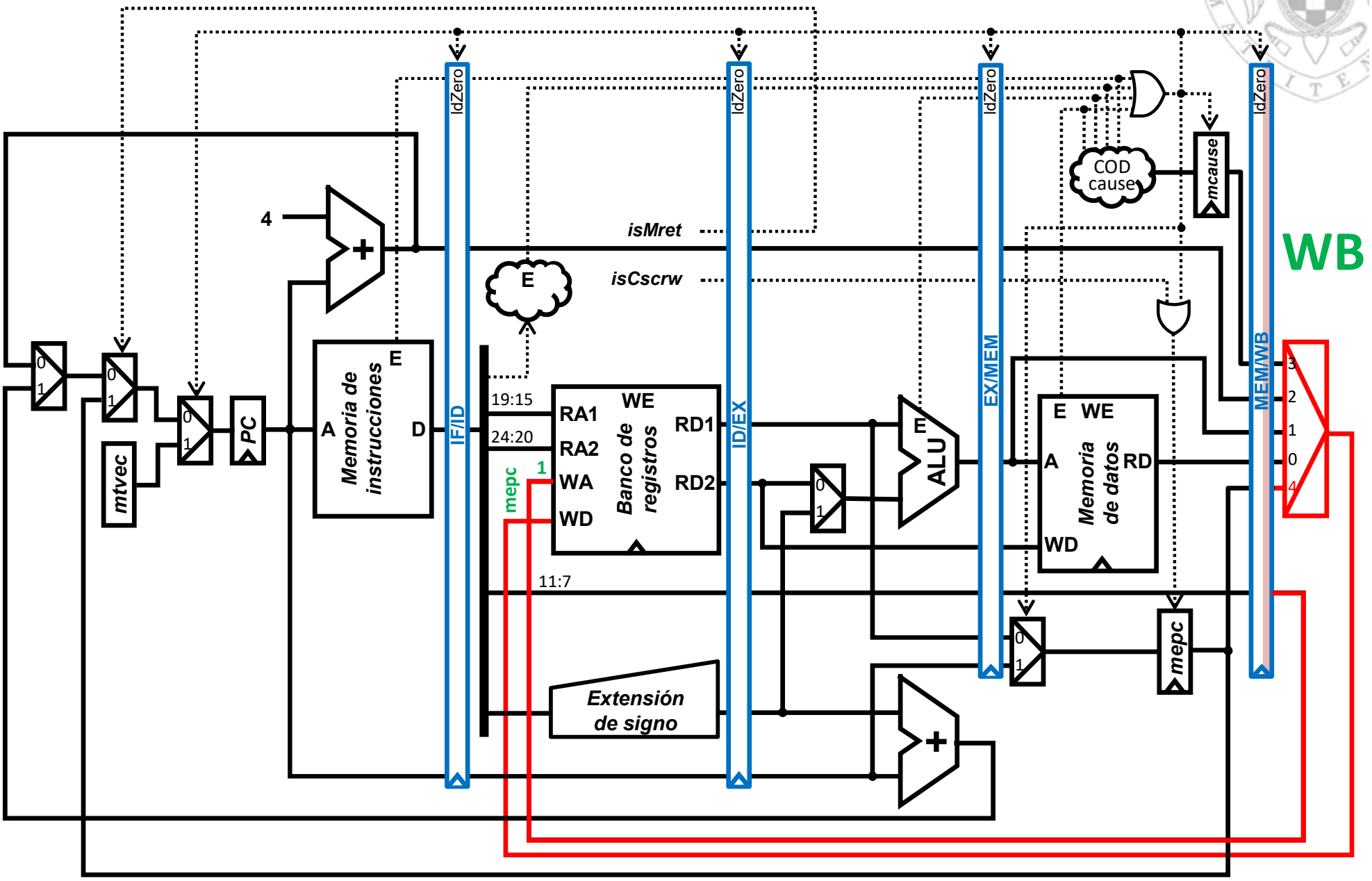
Procesador segmentado

Instrucción `csrrw`: etapa MEM



Procesador segmentado

Instrucción `csrrw`: etapa WB



Procesador segmentado

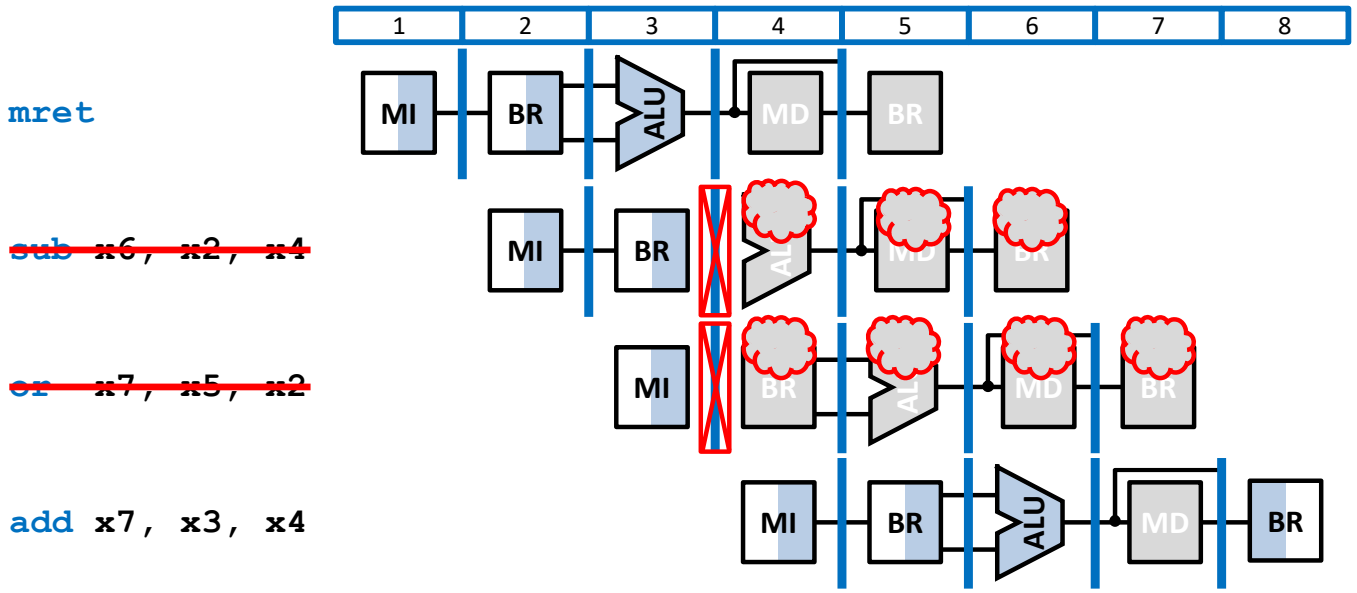
Conflictos (i)

- Para simplificar los esquemas hemos diseñado la gestión de excepciones para el procesador segmentado sin gestión de conflictos.
- Los conflictos entre las instrucciones ya existentes se resuelven añadiendo lógica idéntica diseñada en el tema anterior.
- Sin embargo, las nuevas instrucciones **`mret`** y **`csrrw`** generan nuevos conflictos que es necesario analizar.

Procesador segmentado

Conflictos (i)

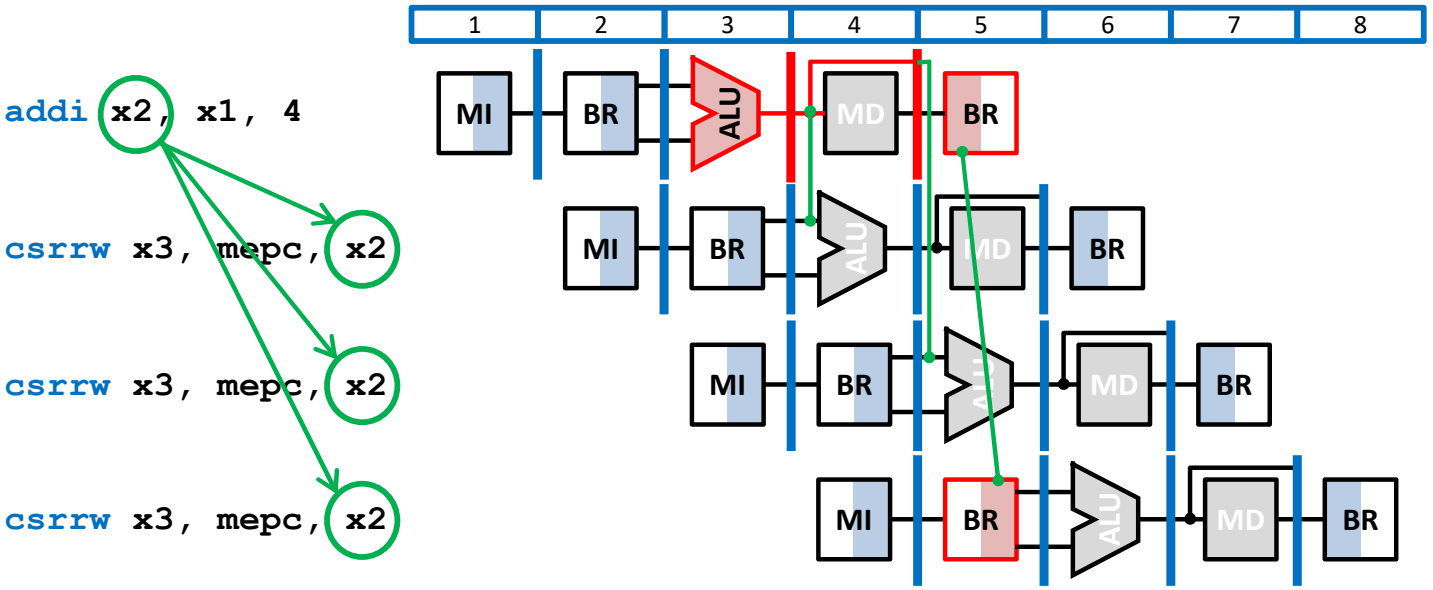
- Existe conflicto de control al ejecutar la instrucción **`mret`**:
  - El salto se toma en EX, pero ya se han lanzado las 2 instrucciones siguientes.
- Se resuelve como cualquier otra instrucción de salto:
  - Con predicción de salto no tomado y 2 ciclos de penalización.
  - La unidad de conflictos debe ampliarse para que borre los registros de segmentación IF/ID e ID/EX si detecta una instrucción **`mret`**.



Procesador segmentado

Conflictos (ii)

- Existe conflicto de datos al ejecutar una instrucción **`csrrw`** que lee el mismo registro que otra instrucción previa escribe.

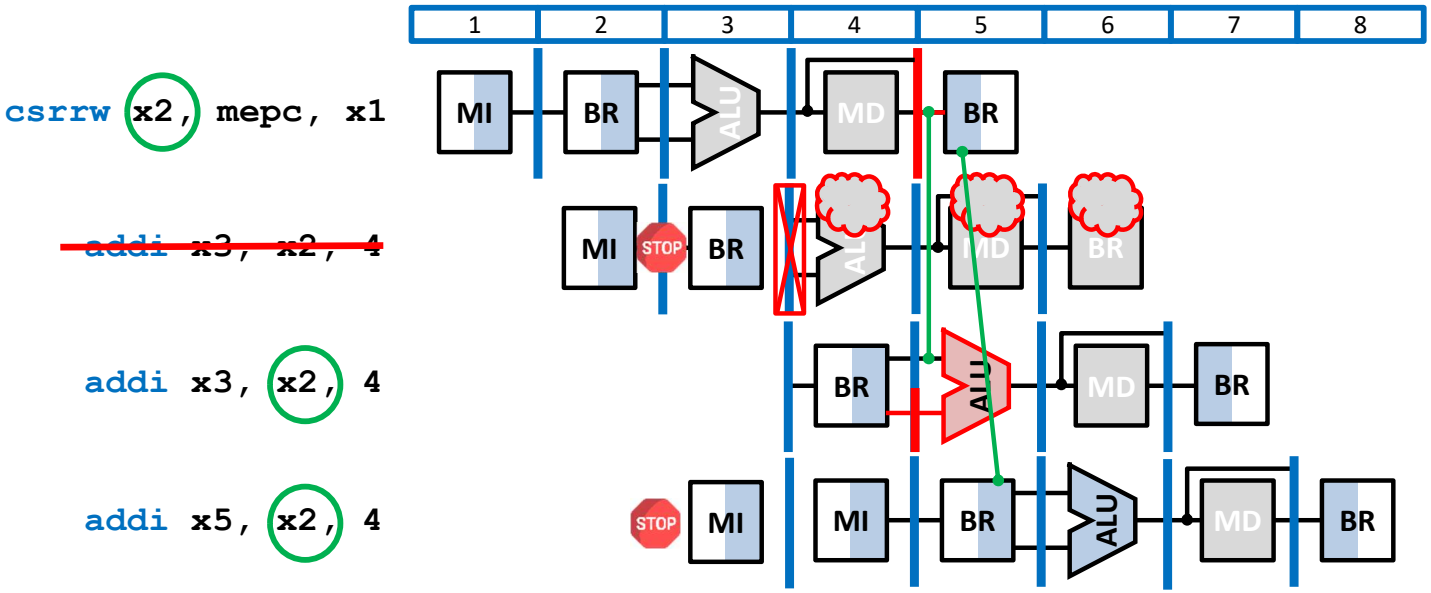


- Se resuelve como el resto de conflicto de datos:
  - Por anticipación o por escritura del BR en la primera mitad del ciclo.
  - Aplica también la parada de un ciclo si la instrucción previa fuera **`lw`**.
  - No requiere hacer cambios en la unidad de anticipación/conflictos.

Procesador segmentado

Conflictos (iii)

- Existe conflicto de datos al ejecutar una instrucción **`csrrw`** que escribe el mismo registro que otra instrucción posterior lee.

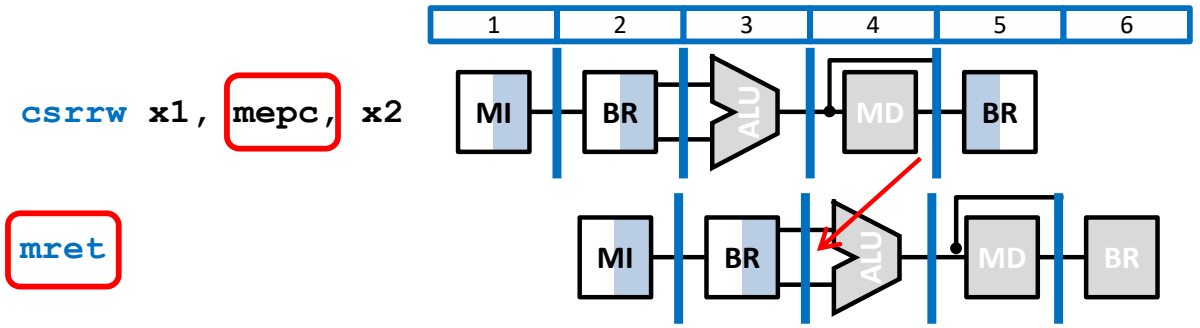


- Se resuelve de manera análoga al caso de instrucciones **`lw`**:
  - Parando un ciclo la instrucción que sigue a **`csrrw`** y el dato pueda anticiparse.
  - La unidad de conflictos debe ampliarse para que pare el pipeline si detecta una instrucción **`csrrw`** en EX con conflicto.

Procesador segmentado

Conflictos (iv)

- Existe un conflicto de datos implícito por **`mepc`**, cuando a una instrucción **`csrrw`** que actualiza **`mepc`** le sigue una instrucción **`mret`**:
  - **`csrrw`** escribe **`mepc`** en la fase MEM.
  - **`mret`** lee **`mepc`** en la fase EX para determinar la dirección a la que saltar.



- Se resuelve de manera análoga a otros conflictos de datos:
  - Escribiendo **`mepc`** al final de primera mitad del ciclo de reloj.
  - Requiere hacer cambios en la ruta de datos.

Procesador segmentado

Conflictos (v)

- En resumen, para que el procesador segmentado con gestión de excepciones pueda también resolver conflictos bastará con:
  - Añadir los multiplexores de anticipación en la ruta de datos.
  - Añadir la unidad anticipación.
  - Añadir la unidad de conflictos con las siguientes modificaciones:

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

El pipeline también se para si hay una instrucción **`csrrw`** *con conflicto en la etapa EX*

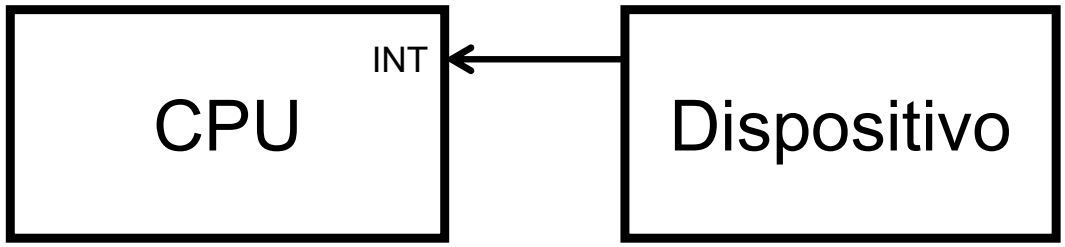
Los registros IF/ID e ID/EX también se borran si hay una instrucción **`mret`** *en la etapa EX*

  - Invertir la entrada de reloj del registro **`mepc`**:

Al invertir la entrada de reloj, **`mepc`** *carga a flanco de bajada (mitad del ciclo a flanco de subida)*

Interrupciones

- Las interrupciones son *traps* disparados por dispositivos externos.
  - Lo hacen activando una señal de entrada al procesador.

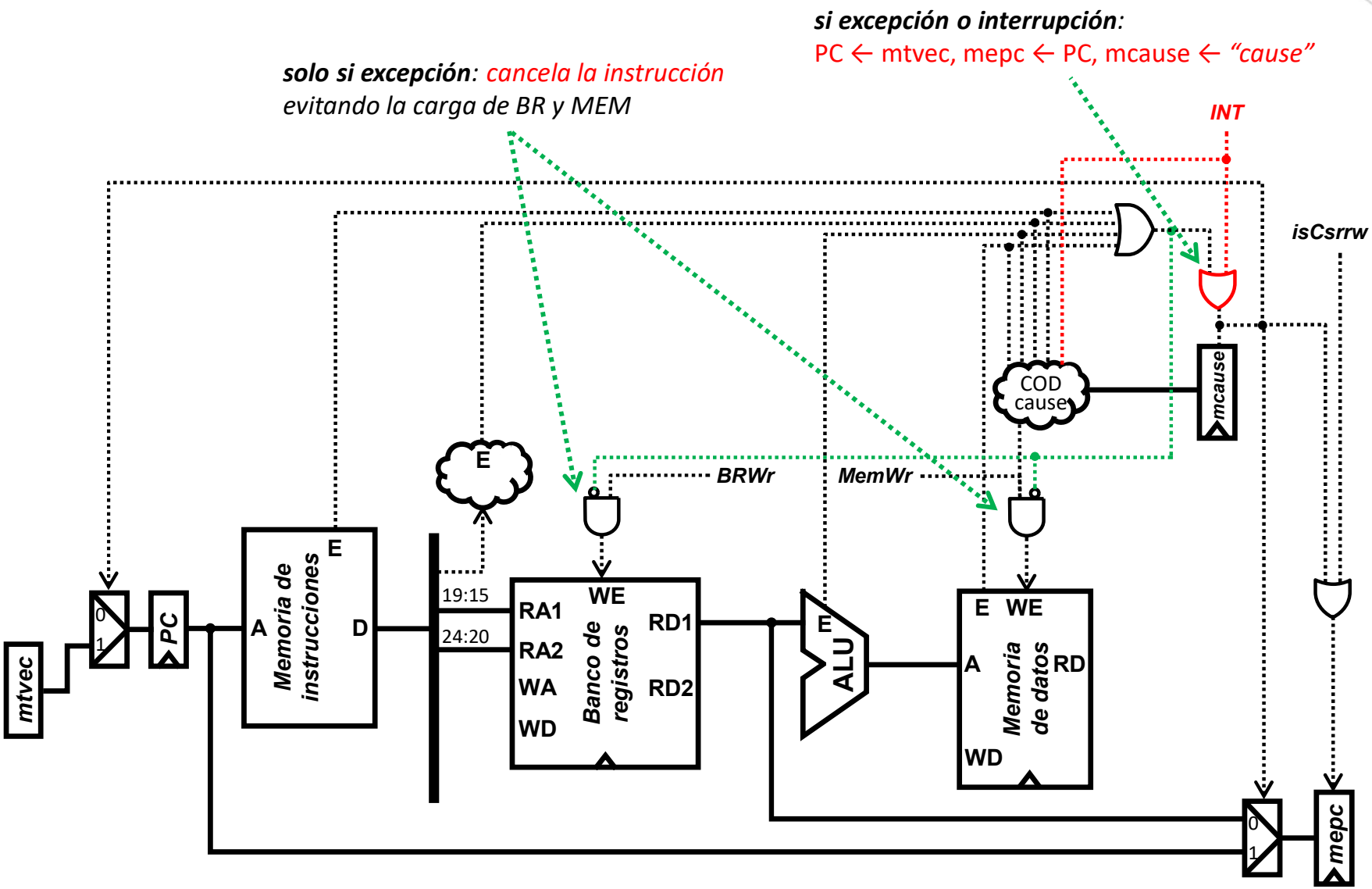


- El tratamiento de interrupciones desde el punto de vista del procesador es análogo al de excepciones con una diferencia:
  - En las excepciones, se cancela la instrucción en curso (causante de la excepción) antes de saltar a la rutina de tratamiento.
  - En las interrupciones, se finaliza la instrucción en curso (durante cuya ejecución se disparó la interrupción) y después se salta.

- El tratamiento de interrupciones en su conjunto es más complejo y se estudiará en asignaturas posteriores.

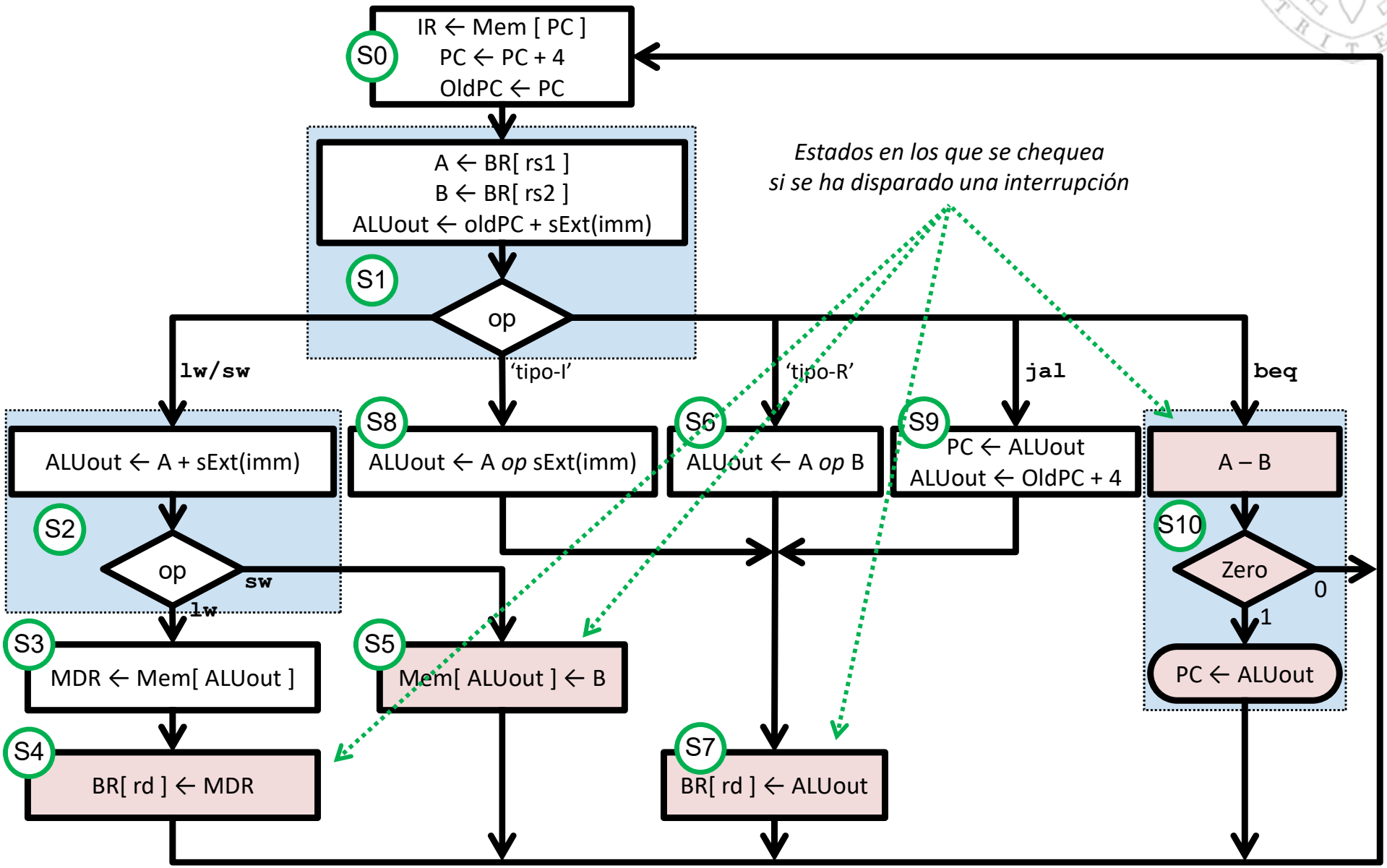
Procesador monociclo

Gestión de interrupciones



Procesador multiciclo

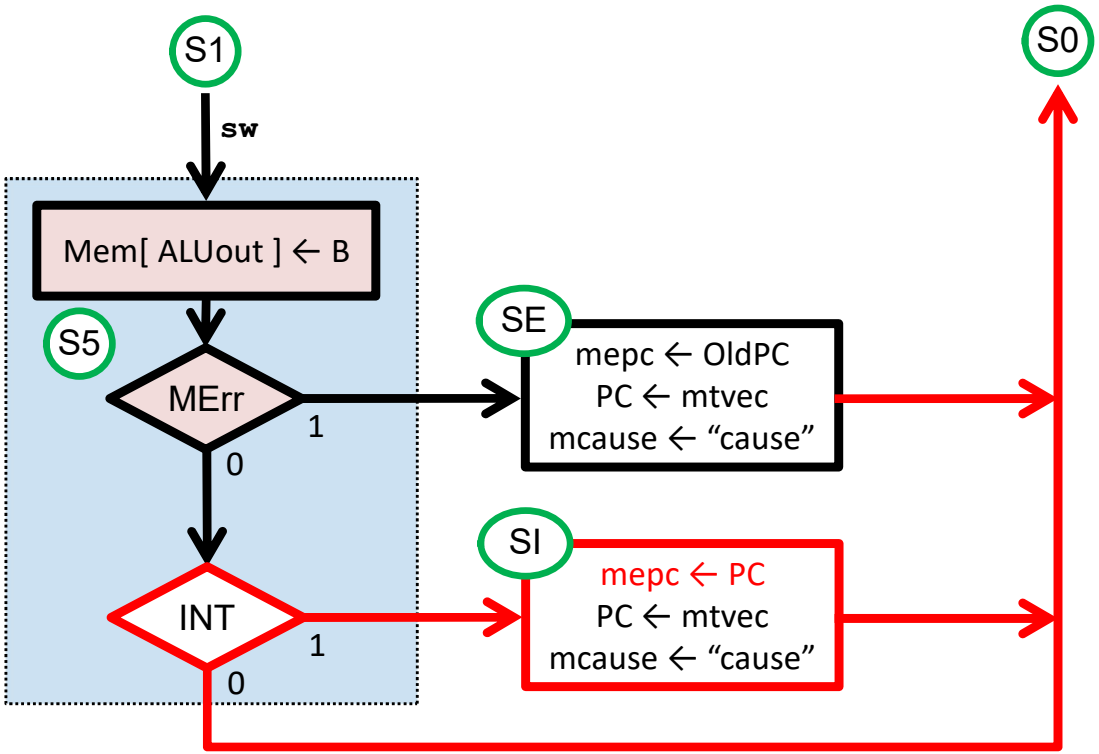
Gestión de interrupciones (i)



Procesador multiciclo

Gestión de interrupciones (ii)

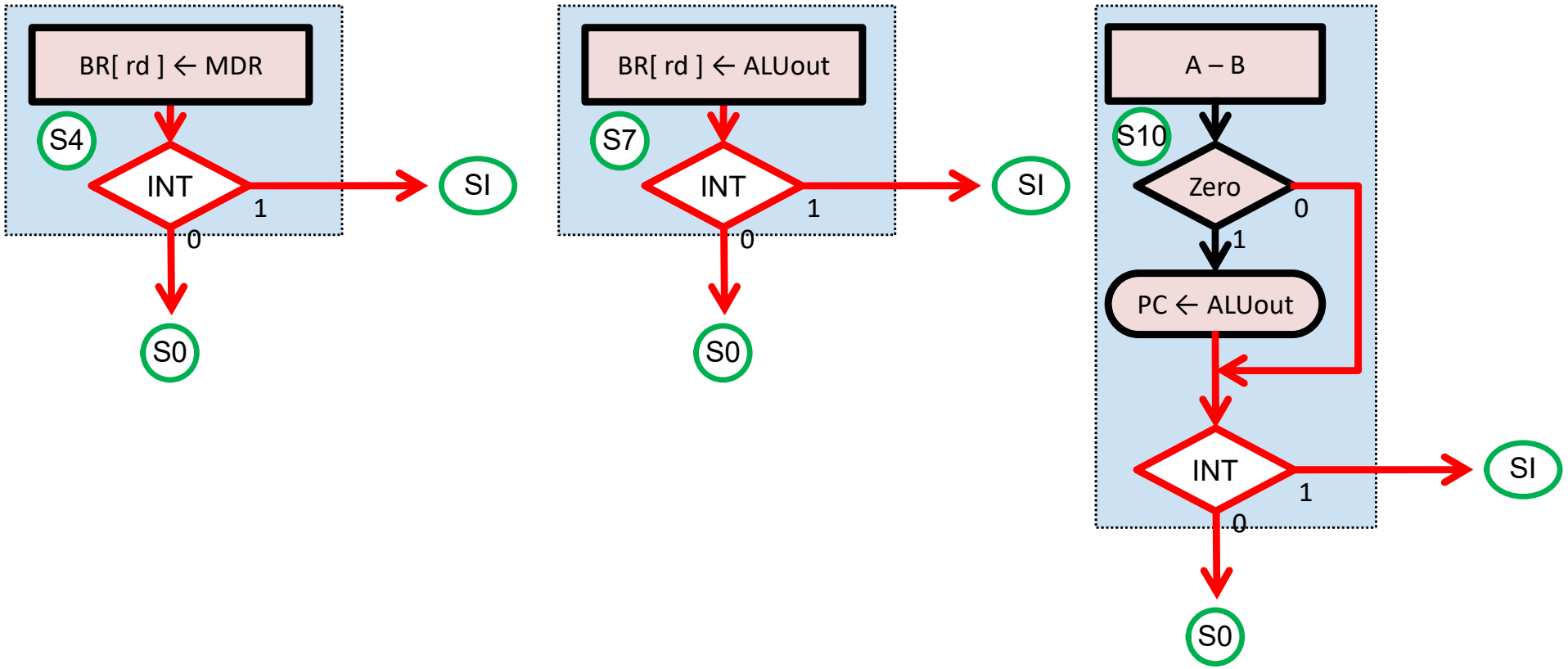
- Se añade estado SI al que se salta en caso de interrupción.
  - SI actualiza CSR y dado que la instrucción ha finalizado, salva en **mepc** la dirección de la instrucción siguiente (almacenada en PC).
  - SI salta a S0 para iniciar la ejecución de la rutina de tratamiento.



Procesador multiciclo

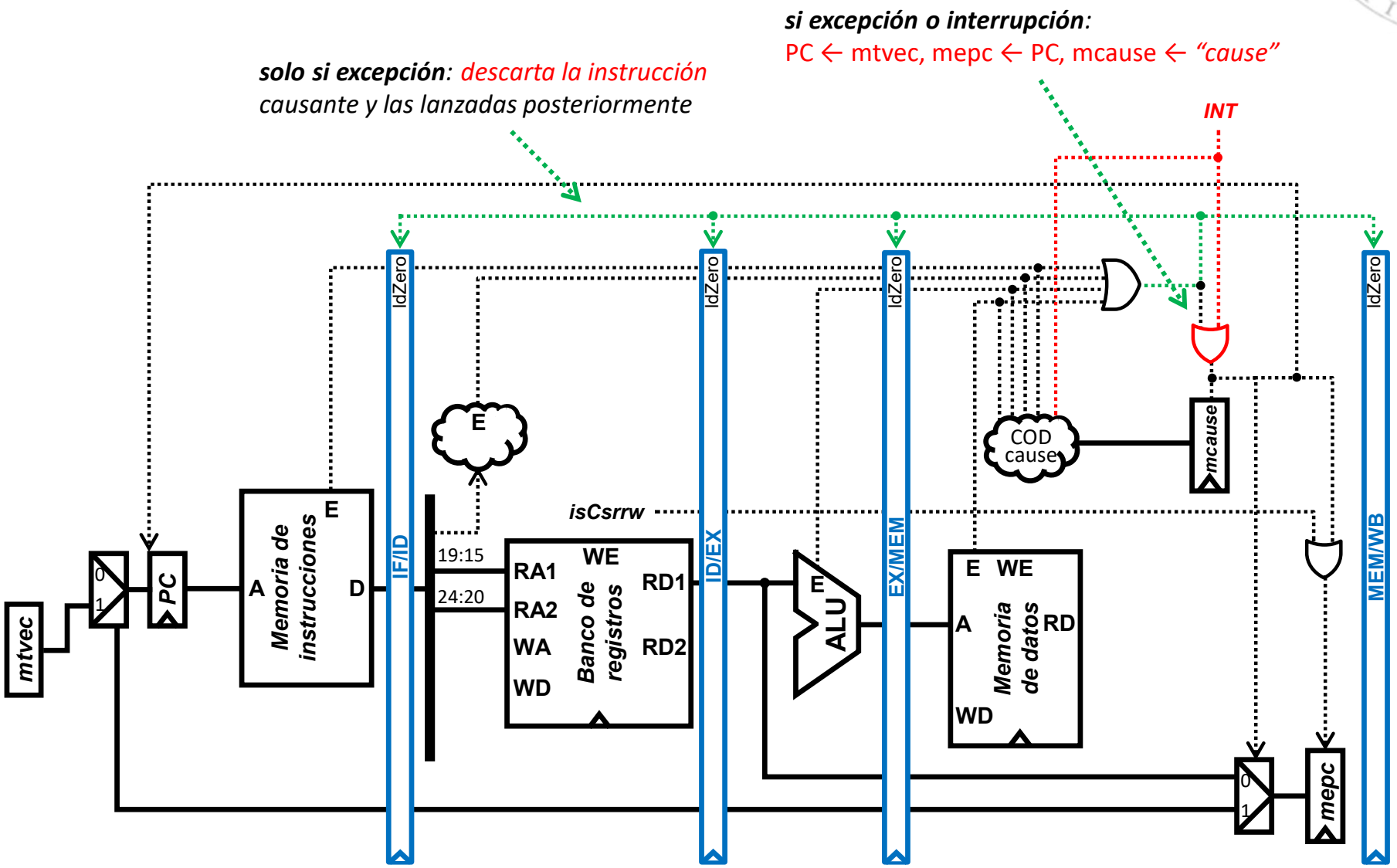
Gestión de interrupciones (iii)

- Todos los estados finales de cada instrucción (cuando esta ha finalizado) chequean la señal de interrupción para saltar o no a SI.



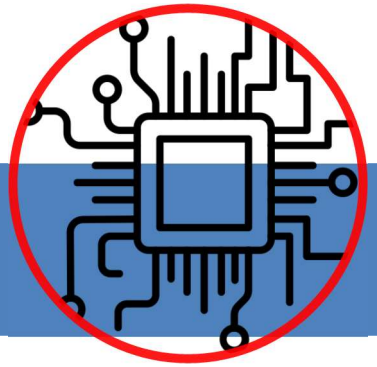
Procesador segmentado

Gestión de interrupciones



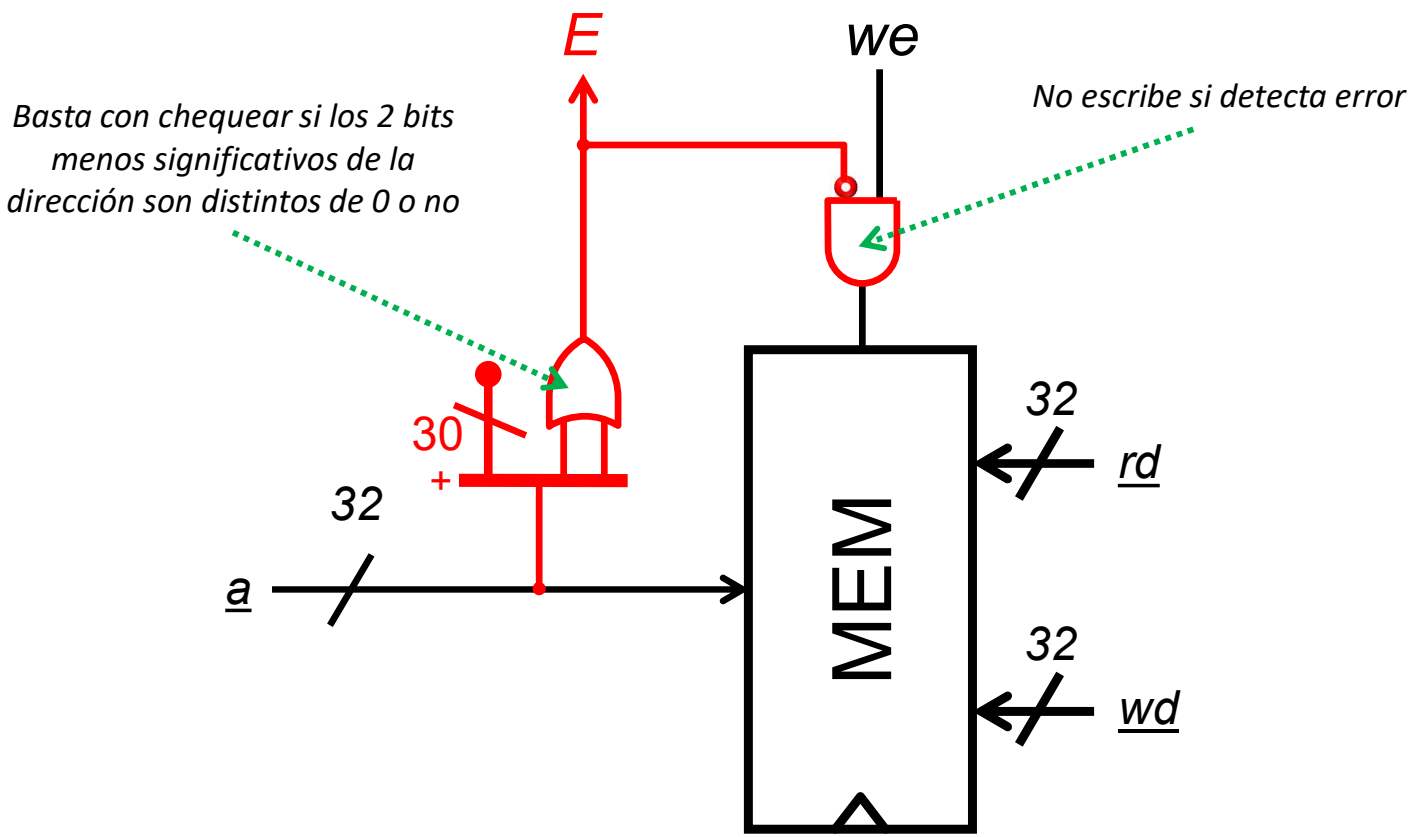
- Rediseño de la Memoria.
- Rediseño de la ALU.
- Diseño del detector de instrucción ilegal.
- Rediseño del controlador monociclo.
- Rediseño del controlador multiciclo.

Apéndice tecnológico



Rediseño de la Memoria

- Añadimos lógica para que pueda indicar error en caso de acceso no alineado a palabra (direcciones no múltiplos de 4)
  - Aparte de indicar el error, en caso de escritura no deberá escribir.

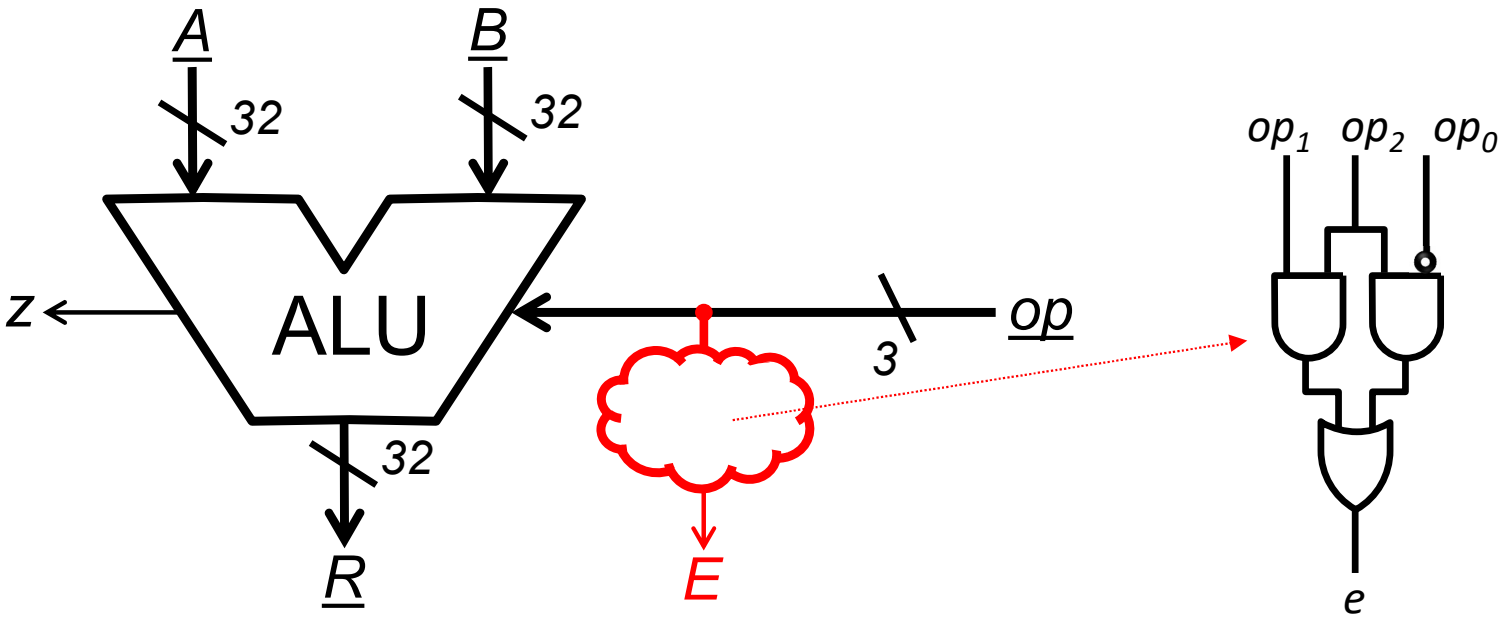


Rediseño de la ALU

- Añadimos lógica para que pueda indicar error en caso de operación aritmético/lógica no implementada:

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

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



Diseño del detector de instrucción ilegal

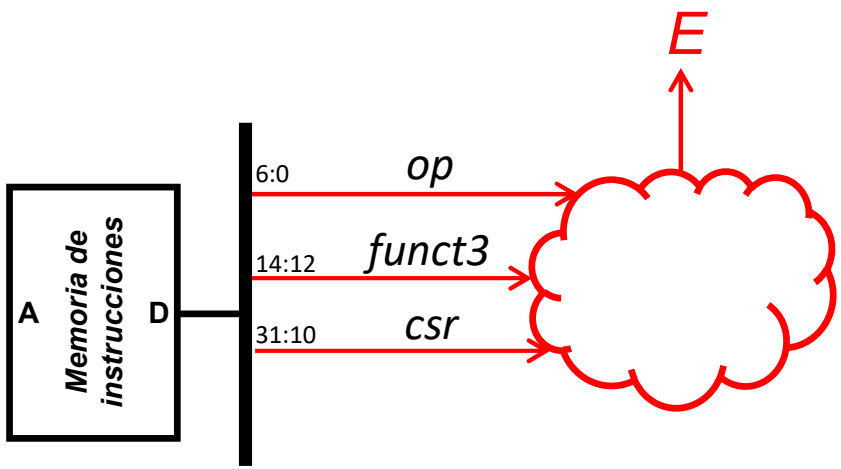


Tabla de verdad

op	funct3	csr	E
0000011 (`lw`)	X	X	0
0100011 (`sw`)	X	X	0
0010011 (tipo-I)	X	X	0
0110011 (tipo-R)	X	X	0
1100011 (`beq`)	X	X	0
1101111 (`jal`)	X	X	0
1110011 (mret)	000	X	0
1110011 (csrrw)	001	0x342 (mcause)	0
1110011 (csrrw)	001	0x341 (mepc)	0
resto			1

Procesador monociclo

Rediseño del DEC principal

Tabla de verdad

op	funct3	csr	Branch	Jump	BRwr	ALUsrc	ALUop	MemWr	ResSrc	isMret	isCsrrw
0000011 (lw)	X	X	0	0	1	1	00(sumar)	0	000	0	0
0100011 (sw)	X	X	0	0	0	1	00(sumar)	1	–	0	0
0010011 (tipo-I)	X	X	0	0	1	1	10(operar)	0	001	0	0
0110011 (tipo-R)	X	X	0	0	1	0	10(operar)	0	001	0	0
1100011 (beq)	X	X	1	0	0	0	01(restar)	0	–	0	0
1101111 (jal)	X	X	0	1	1	–	–	0	010	0	0
1110011 (mret)	000	0x302	0	0	0	–	–	0	–	1	0
1110011 (csrrw)	001	0x342 (mcause)	0	0	1	–	–	0	011	0	1
1110011 (csrrw)	001	0x341 (mepc)	0	0	1	–	–	0	100	0	1

Procesador multiciclo

Rediseño de la FSM principal: función de transición (i)

Tabla de verdad

estado	MErr	OpErr	ALUErr	INT	op	funct3	csr	estado'
S0	0	X	X	X	X	X	X	S1
S0	1	X	X	X	X	X	X	SE
S1	X	0	X	X	0X00011 (lw/sw)	X	X	S2
S1	X	0	X	X	0010011 (tipo-I)	X	X	S8
S1	X	0	X	X	0110011 (tipo-R)	X	X	S6
S1	X	0	X	X	1101111 (jal)	X	X	S9
S1	X	0	X	X	1100011 (beq)	X	X	S10
S1	X	0	X	X	1110011 (mret)	000	0x302	S11
S1	X	0	X	X	1110011 (csrrw)	001	0x342 (mcause)	S12
S1	X	0	X	X	1110011 (csrrw)	001	0x341 (mepc)	S13
S1	X	1	X	X	X	X	X	SE
S2	X	0	X	X	0000011 (lw)	X	X	S3
S2	X	0	X	X	0100011 (sw)	X	X	S5
S3	0	X	X	X	X	X	X	S4
S3	1	X	X	X	X	X	X	SE
S4	X	X	X	X	X	X	X	S0
S4	X	X	X	1	X	X	X	SI

Procesador multiciclo

Rediseño de la FSM principal: función de transición (ii)

Tabla de verdad (cont.)

estado	MErr	OpErr	ALUErr	INT	op	funct3	csr	estado'
S5	0	X	X	X	X	X	X	S0
S5	1	X	X	X	X	X	X	SE
S5	0	X	X	1	X	X	X	SI
S6	X	X	0	X	X	X	X	S7
S6	X	X	1	X	X	X	X	SE
S7	X	X	X	X	X	X	X	S0
S7	0	X	X	1	X	X	X	SI
S8	X	X	0	X	X	X	X	S7
S8	X	X	1	X	X	X	X	SE
S9	X	X	X	X	X	X	X	S7
S10	X	X	X	X	X	X	X	S0
S10	0	X	X	1	X	X	X	SI
S11	X	X	X	X	X	X	X	S0
S12	X	X	X	X	X	X	X	S0
S13	X	X	X	X	X	X	X	S0
SE	X	X	X	X	X	X	X	S0
SI	X	X	X	X	X	X	X	S0

Procesador multiciclo

Rediseño de la FSM principal: función de salida

Función de salida

estado	Branch	PCupdate	AddrSrc	MemWr	IRwr	BRwr	ALUsrcA	ALUsrcB	ALUop	ResSrc	CauseWr	EPCWr
S0	0	1	0	0	1	0	00	10	00	010	0	0
S1	0	0	–	0	0	0	01	01	00	–	0	0
S2	0	0	–	0	0	0	10	01	00	–	0	0
S3	0	0	1	0	0	0	–	–	–	000	0	0
S4	0	0	–	0	0	1	–	–	–	001	0	0
S5	0	0	1	1	0	0	–	–	–	000	0	0
S6	0	0	–	0	0	0	10	00	10	–	0	0
S7	0	0	–	0	0	1	–	–	–	000	0	0
S8	0	0	–	0	0	0	10	01	10	–	0	0
S9	0	1	–	0	0	0	01	10	00	000	0	0
S10	1	0	–	0	0	0	10	00	01	000	0	0
S11	0	1	–	0	0	0	–	–	–	100	0	0
S12	0	0	–	0	0	1	–	–	–	101	0	0
S13	0	0	–	0	0	1	10	–	–	100	0	1
SE	0	1	–	0	0	0	01	–	–	011	1	1
SI	0	1	–	0	0	0	00	–	–	011	1	1

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/