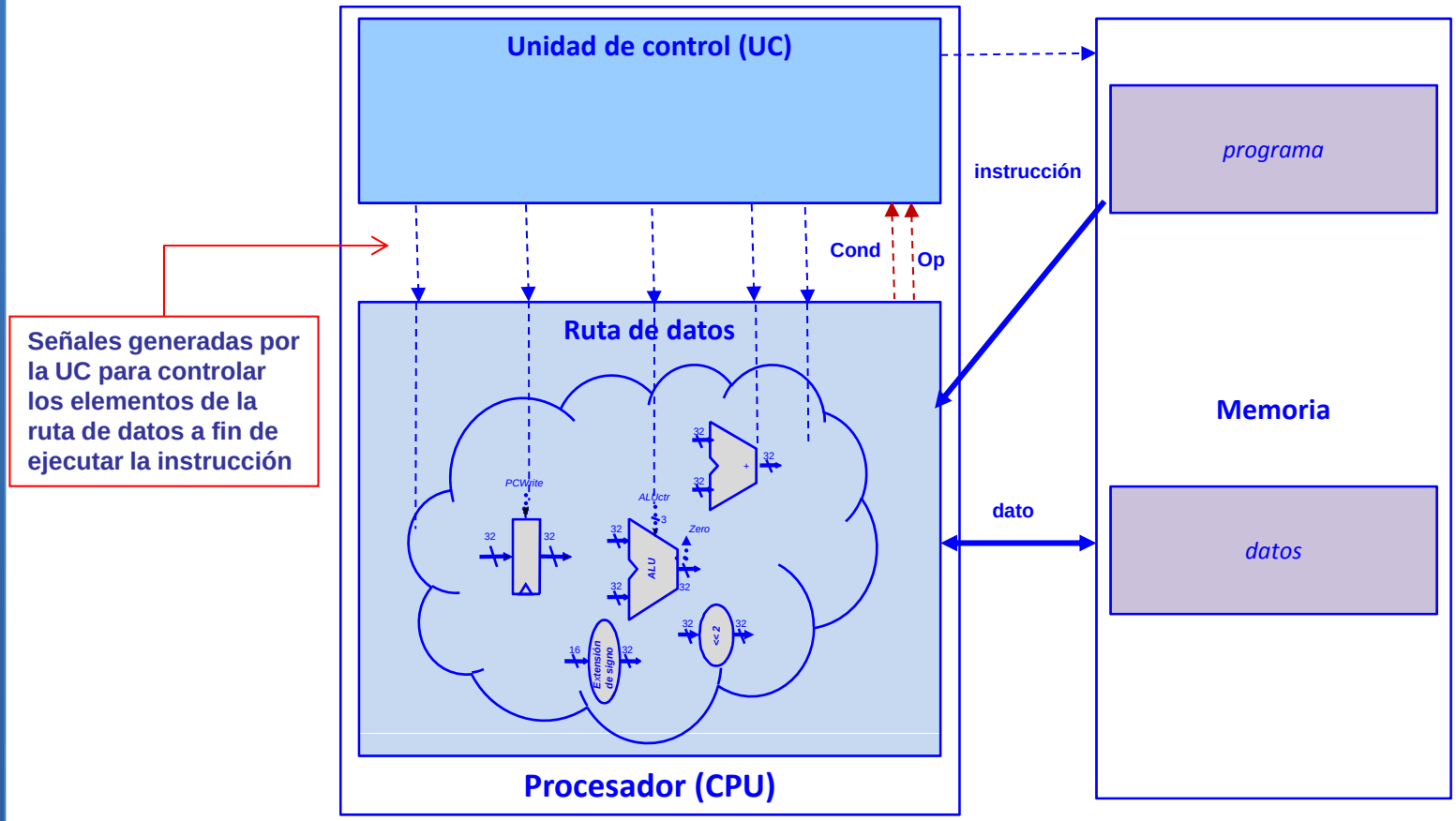
Tema 5: Segmentación

Índice

1. Introducción

Esquema general de la arquitectura de un computador: modelo Von Neuman

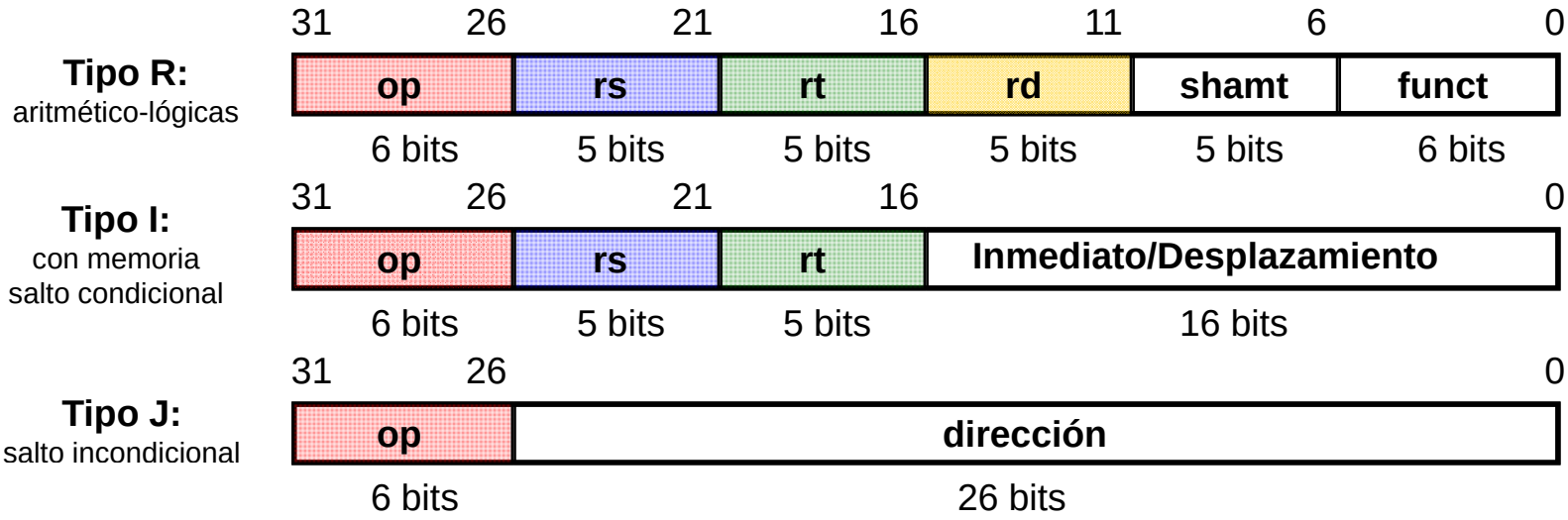
- ✓ Las instrucciones son ejecutadas sobre la ruta de datos
- ✓ La UC genera las señales que gobiernan los elementos de la ruta de datos



1. Introducción

Arquitectura MIPS (DLX)

- ✓ Instrucciones de longitud fija: 32 bits
- ✓ Tres formatos de instrucción diferentes:



- ✓ Significado de los campos:
  - **op:** identificador de instrucción
  - **rs, rt, rd:** identificadores de los registros fuentes y destino
  - **shamt:** cantidad a desplazar (en operaciones de desplazamiento)
  - **funct:** selecciona la operación aritmética a realizar
  - **inmediato:** operando inmediato o desplazamiento en direccionamiento indirecto
  - **dirección:** dirección destino del salto

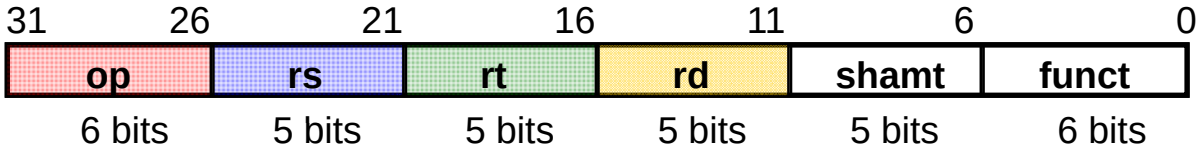
1. Introducción

Instrucciones aritmético-lógicas

– Con registros

add rd, rs, rt rd ← rs + rt, PC ← PC + 4

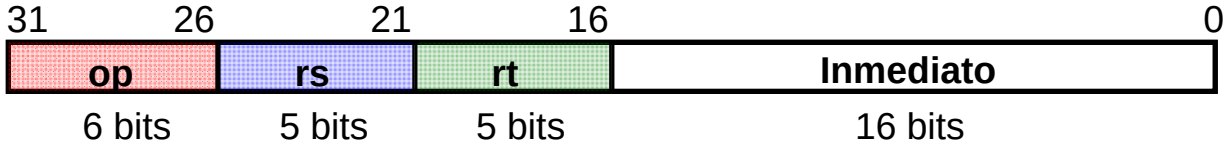
- Modos de direccionamiento:
  – **Directo a registro**



– Con operando inmediato

add rt, rs, #5 rt ← rs + 5, PC ← PC + 4

- Modos de direccionamiento:
  – **Directo a registro** para el 1er operando fuente y operando destino
  – **Inmediato** para el 2º operando fuente



1. Introducción

Instrucciones de acceso a memoria (load y store)

- Modos de direccionamiento:
  - **Directo a registro**
  - **Relativo a registro base**

lw rt, desplaz(rs) **rt ← Memoria(rs + SignExt(desplaz)) , PC ← PC + 4**

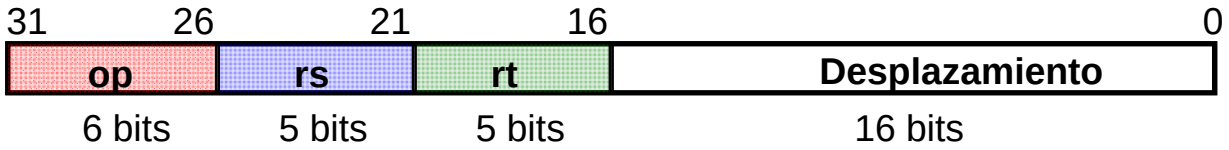
sw rt, desplaz(rs) **Memoria(rs + SignExt(desplaz)) ← rt , PC ← PC + 4**

Instrucciones de salto condicional

- Modos de direccionamiento:
  - **Directo a registro**
  - **Con desplazamiento relativo a contador de programa**

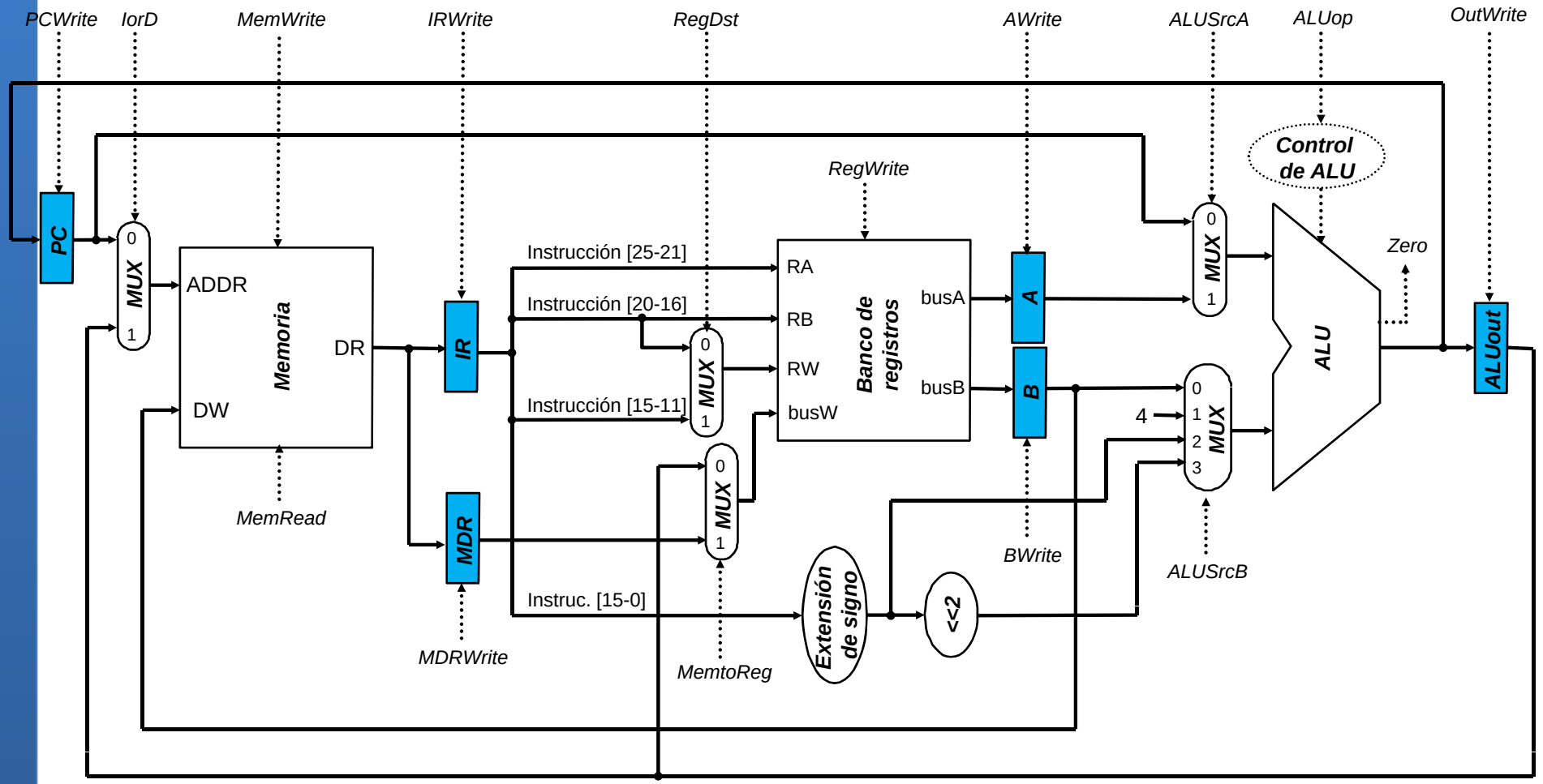
beq rs, rt, desplaz **si (rs = rt) entonces (PC ← PC + 4 + 4·SignExt(desplaz))**

en otro caso PC ← PC + 4



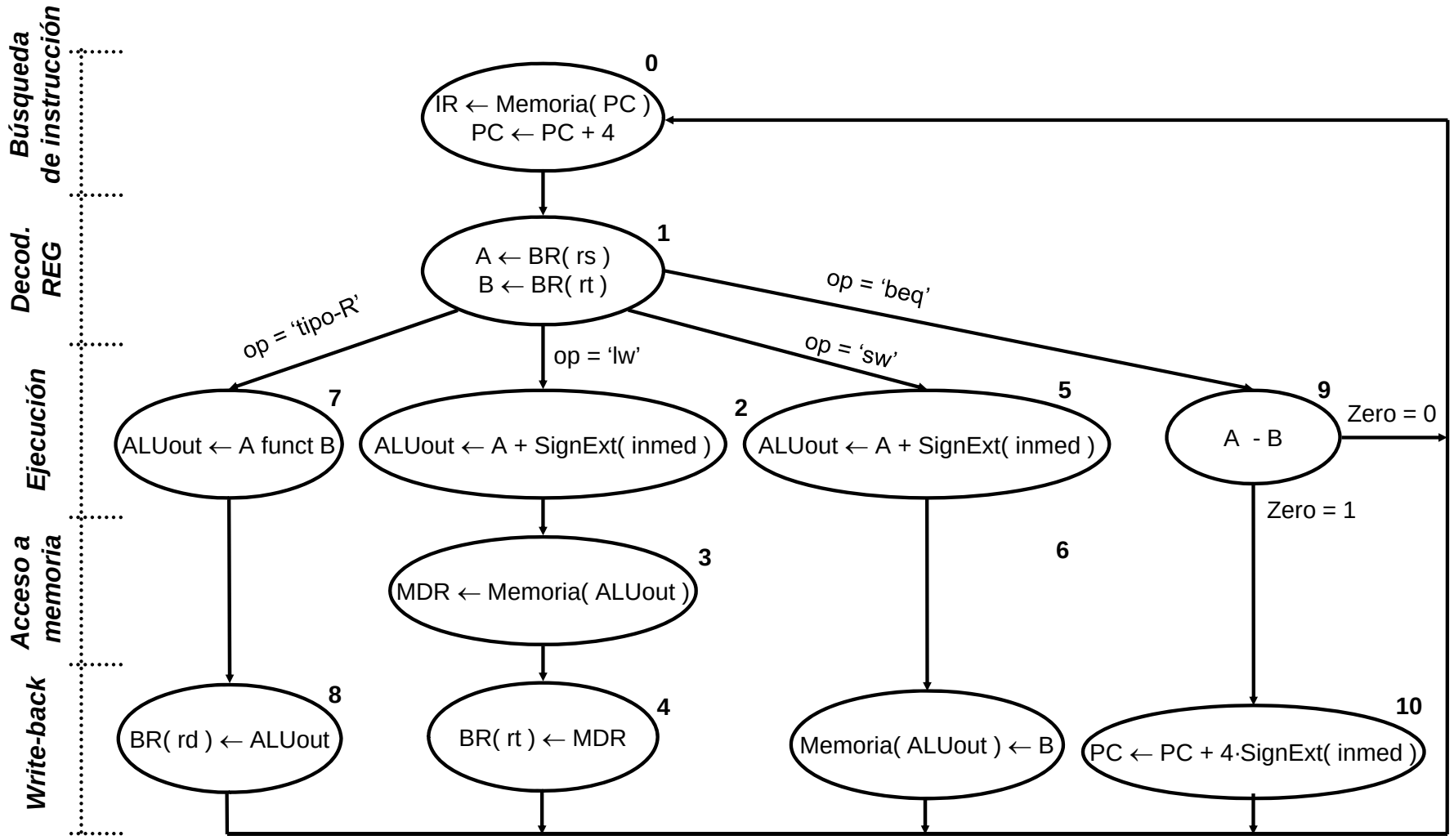
1. Introducción

Ruta de datos (Multiciclo)



1. Introducción

Diagrama de estados del control



2. MIPS: Excepciones y control

Excepciones: **hay dos tipos**

- **Interrupciones**
  - Se producen a causa de sucesos externos al procesador
  - Son asíncronas a la ejecución del programa
  - Se pueden tratar entre instrucciones

- **Traps**
  - Se producen por causas internas. Overflow, errores, fallos de pagina…
  - Son síncronas con la ejecución del programa
  - La condiciones deben ser almacenadas
  - El programa debe ser abortado o continuado desde esa instrucción

2. MIPS: Excepciones y control

Gestión de traps (Instrucción indefinida y overflow aritmético)

- Acciones básicas:

  - **Salvar el contador de programa** de la instrucción interrumpida en el *registro EPC* (exception program counter)
  - Transferir el **control al sistema operativo** en alguna dirección especificada
  - El S.O. realizará la acción apropiada, como ejecutar alguna tarea asociada al overflow o detener la ejecución del programa
  - **Volver a la ejecución normal** del programa en la dirección guardada en EPC

2. MIPS: Excepciones y control

Hardware añadido para gestionar las excepciones:

- **Registro EPC** (exception program counter)

- **Registro de estado**: **Cause register** (32 bits) con un campo que indica la causa de la excepción:

  Bit 0: Instrucción indefinida.

  Bit 1: Overflow aritmético.

- Se añaden las **señales de control**:

  EPCwrite. Escribe en EPC. (EPC <= PC – 4)

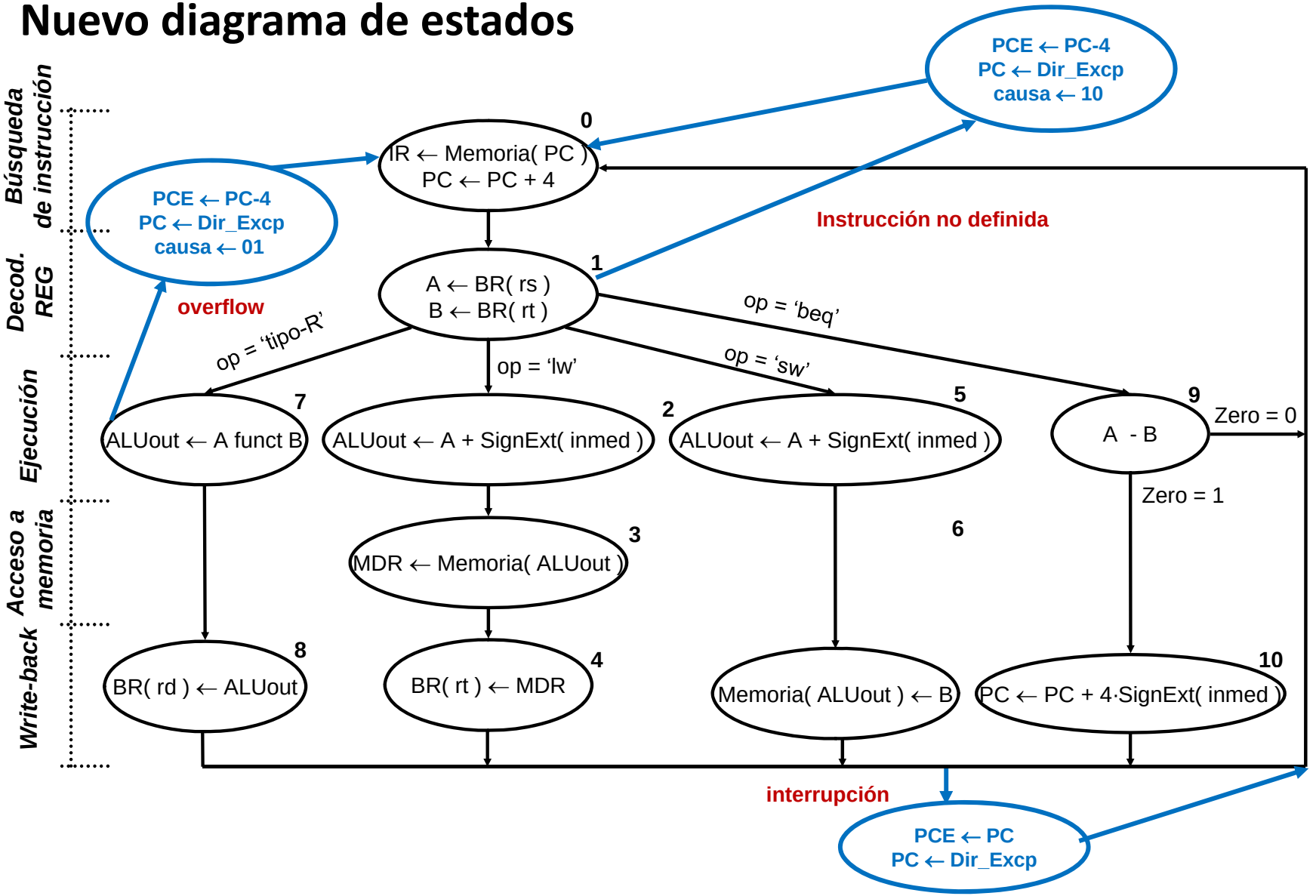
  CauseWrite. Escribe en Cause

  IntCause. Escribe un 1 sobre el bit apropiado de Cause.

- Para dar la dirección de la rutina de tratamiento de excepción, se añade una entrada al **multiplexor que controla la carga del PC**, con la dirección de esta rutina

2. MIPS: Excepciones y control

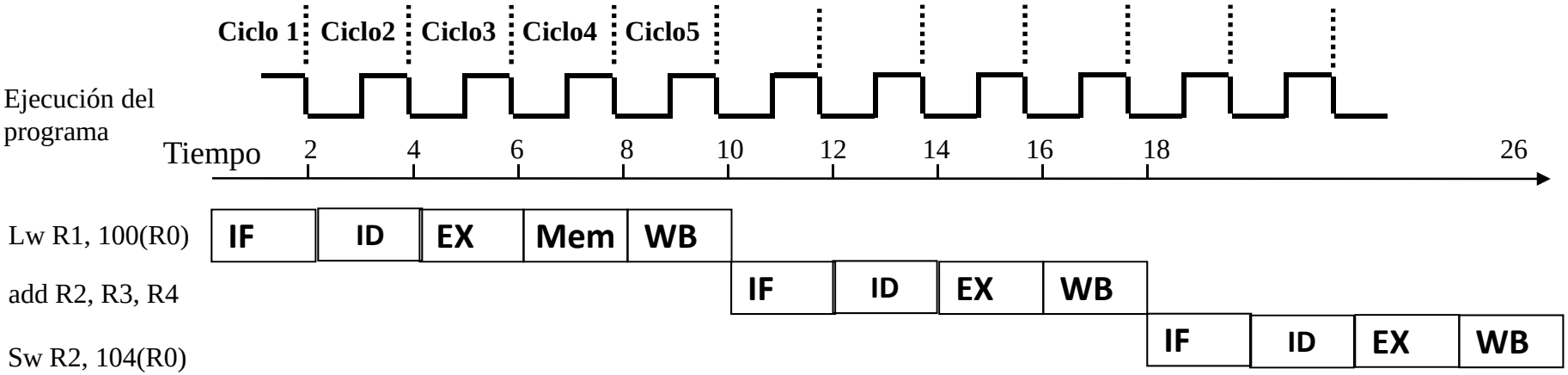
Nuevo diagrama de estados



3. Segmentación

Ejecución de instrucciones MIPS Multiciclo

- Instrucciones **Load tiene 5 fases** y cada una utiliza una de las etapas
- **El resto** de instrucciones **tienen 4 fases**, no usan la fase Mem



26ns

3. Segmentación

Cómo mejorar el rendimiento del procesador: Aplicar segmentación

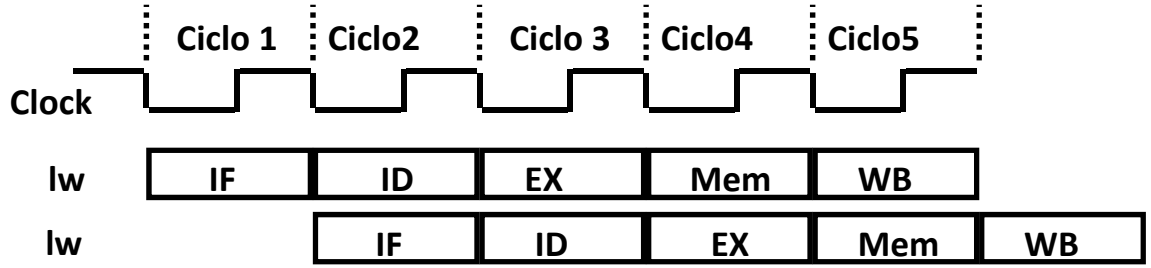
- Es una técnica de implementación en la que múltiples instrucciones se solapan en ejecución
  - **Cada etapa opera en paralelo con otras etapas pero sobre instrucciones diferentes**

- Se puede aplicar porque las fases por la que pasa una instrucción no usan todas las componentes de la ruta de datos
  - **Explota el paralelismos a nivel de instrucciones**

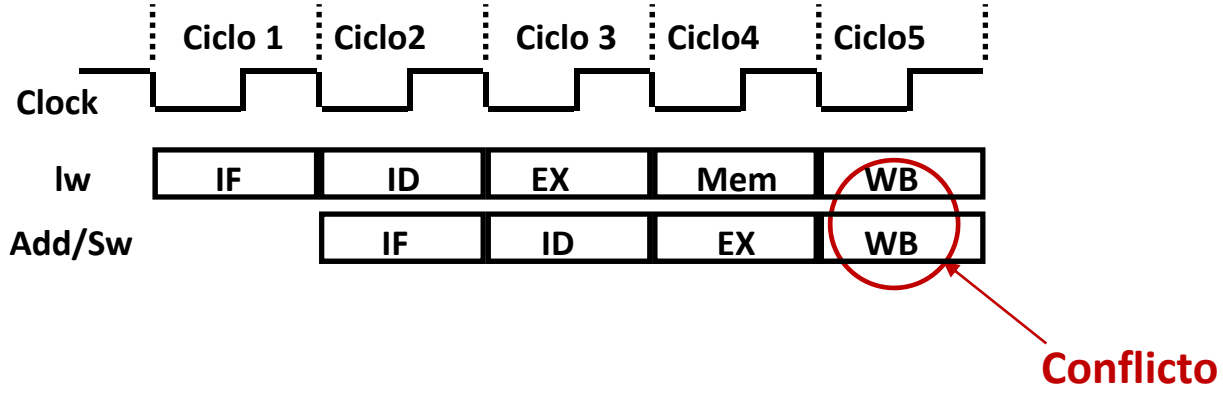
3. Segmentación

Ejecución Procesador Segmentado

– Ejemplo 1



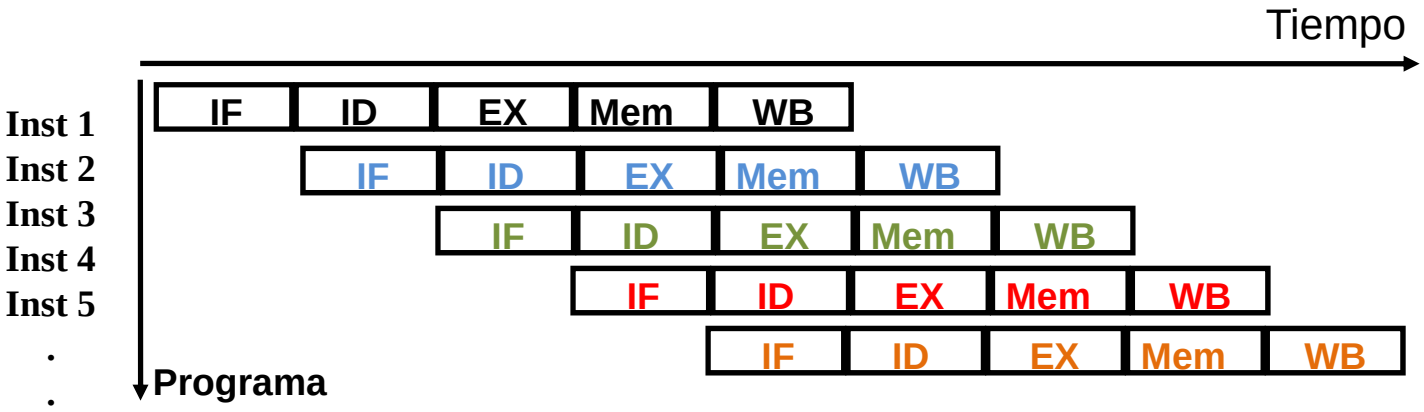
– Ejemplo 2



3. Segmentación

¿ Cómo evitar este conflicto?

- No puede haber más de una instrucción intentando acceder a una etapa
- **TODAS las instrucciones deben pasar por TODAS las etapas**
- El orden de etapas es el mismo para TODAS las instrucciones



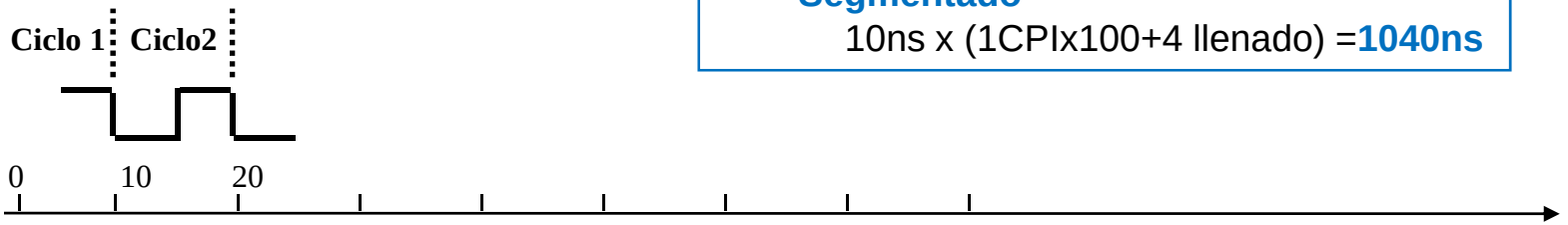
- A partir del ciclo 5
  - Sale una instrucción cada ciclo de reloj
  - **CPI=1**
- **Los 4 primeros ciclo se llaman de llenado del pipeline y se pueden despreciar.**
- $CPI_{Ideal}=1$

3. Segmentación

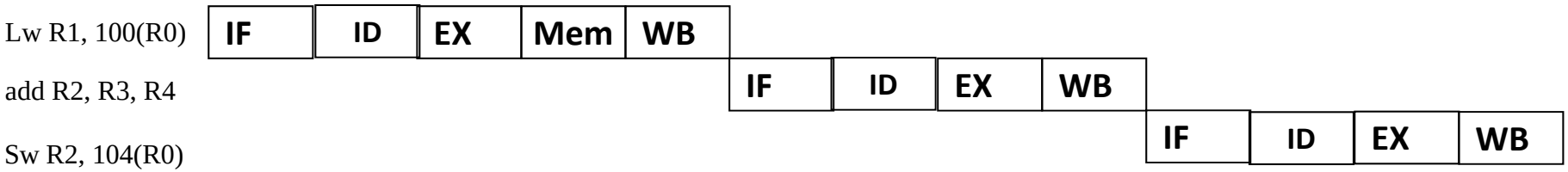
Segmentado frente a Multiciclo

Si se ejecutan **100 instrucciones**

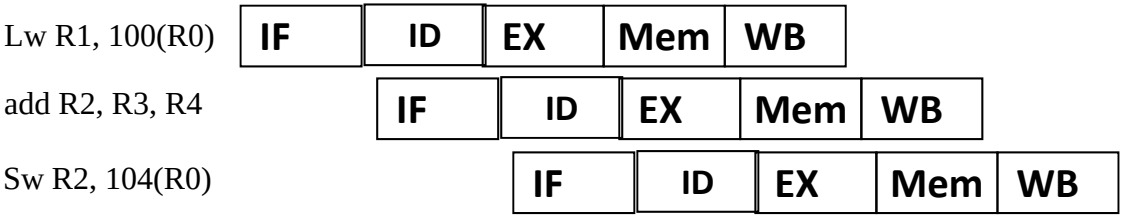
- **Multiciclo**
  10ns/ciclo x 4.6 CPI x 100 =**4600ns**
- **Segmentado**
  10ns x (1CPIx100+4 llenado) =**1040ns**



Multiciclo



Segmentado



3. Segmentación

¿Qué dificulta la segmentación?

- **Riesgos**
  - Aparecen situaciones que impiden que en cada ciclo se inicie la ejecución de una nueva instrucción
  - **Tipos:**
    - **Estructurales**
      - Se producen cuando dos instrucciones tratan de utilizar el mismo recurso en el mismo ciclo
    - **De datos**
      - Se intenta utilizar un dato antes de que este preparado. Mantenimiento del orden estricto de lecturas y escrituras
    - **De control**
      - Intentar tomar una decisión sobre una condición todavía no evaluada

  ✓ **Los riesgos se deben *detectar y resolver***

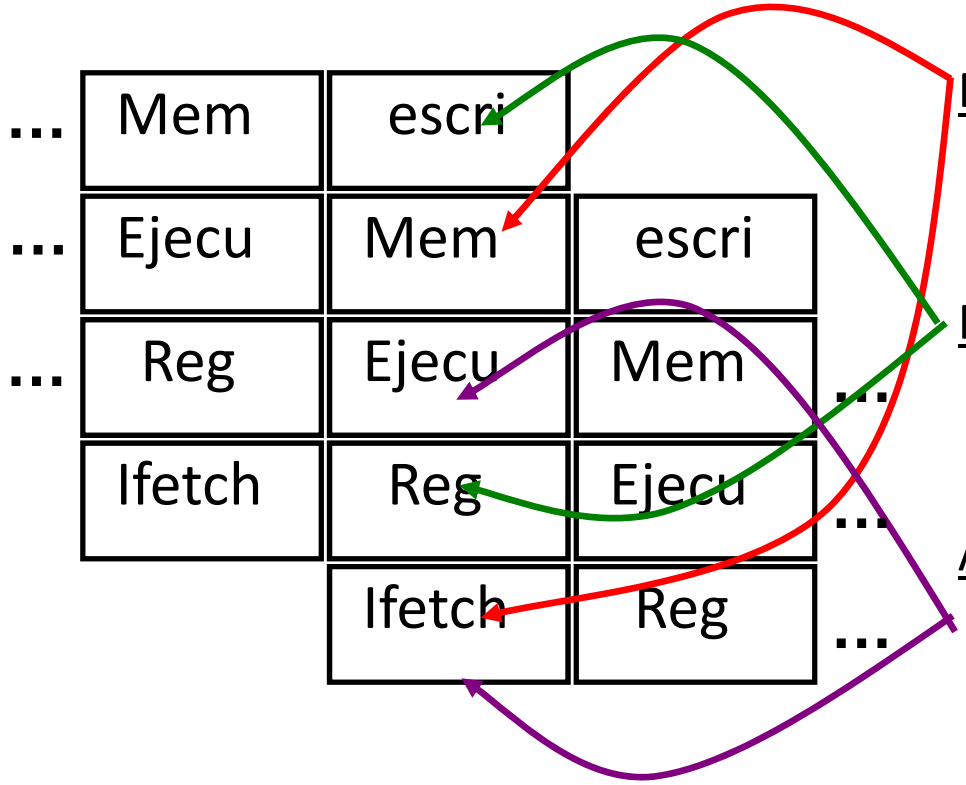
- **Gestión de interrupciones**

- **Ejecución fuera de orden**

4. Segmentación: Riesgos estructurales

Riesgos estructurales

- ✓ *Se producen cuando dos instrucciones tratan de utilizar el mismo recurso en el mismo ciclo*



Memoria
- ✓ Leer una instrucción
- ✓ Leer o escribir un operando

Registros
- ✓ Dos lecturas
- ✓ una escritura

ALU
- ✓ Cálculo de la operación
- ✓ Cálculo de la dirección
- ✓ Cálculo de la condición
- ✓ Actualización de PC

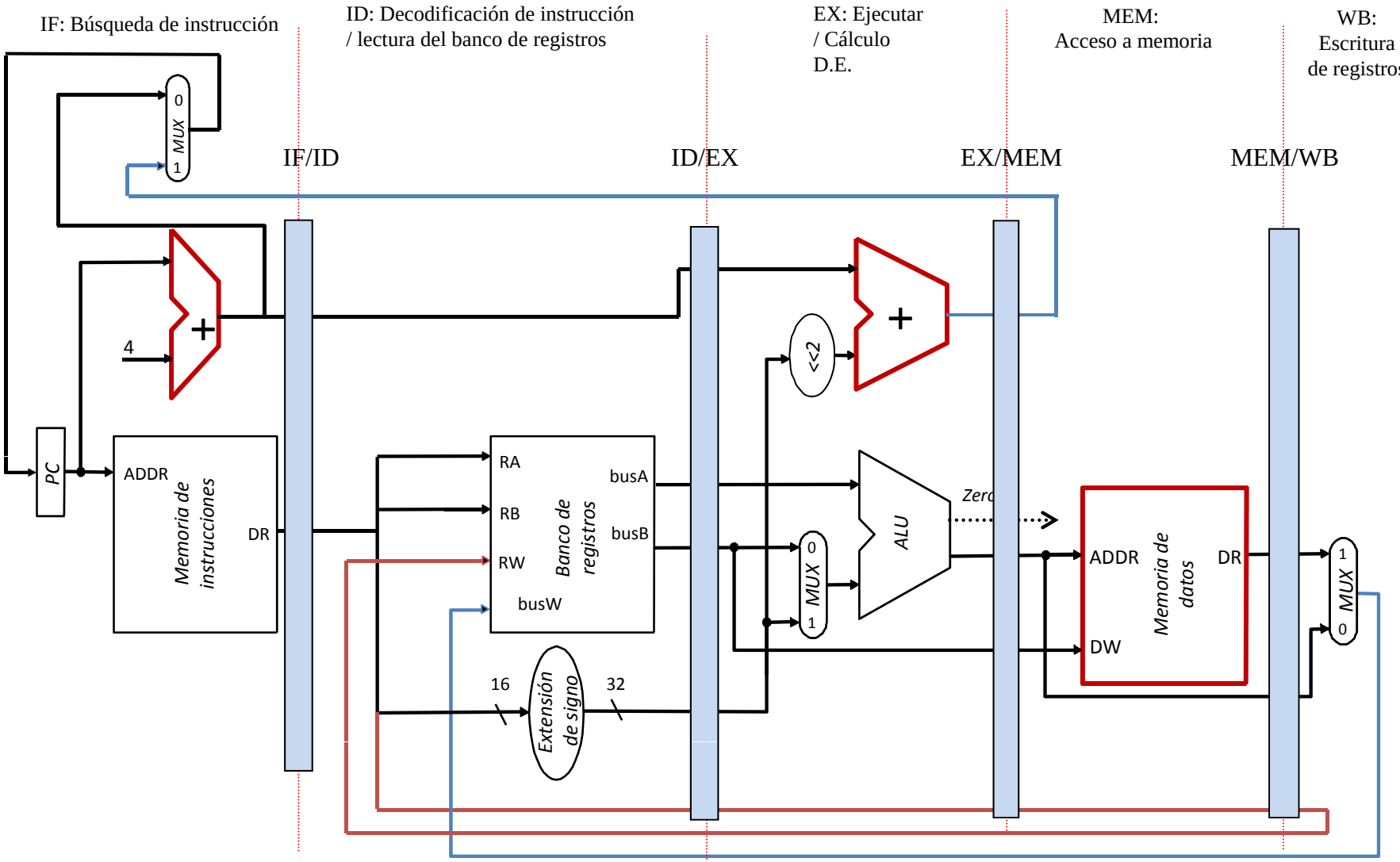
Objetivo:

- ✓ **Ejecutar sin conflicto cualquier combinación de instrucciones**

4. Segmentación: Riesgos estructurales

¿Cómo resolver los riesgos estructurales?

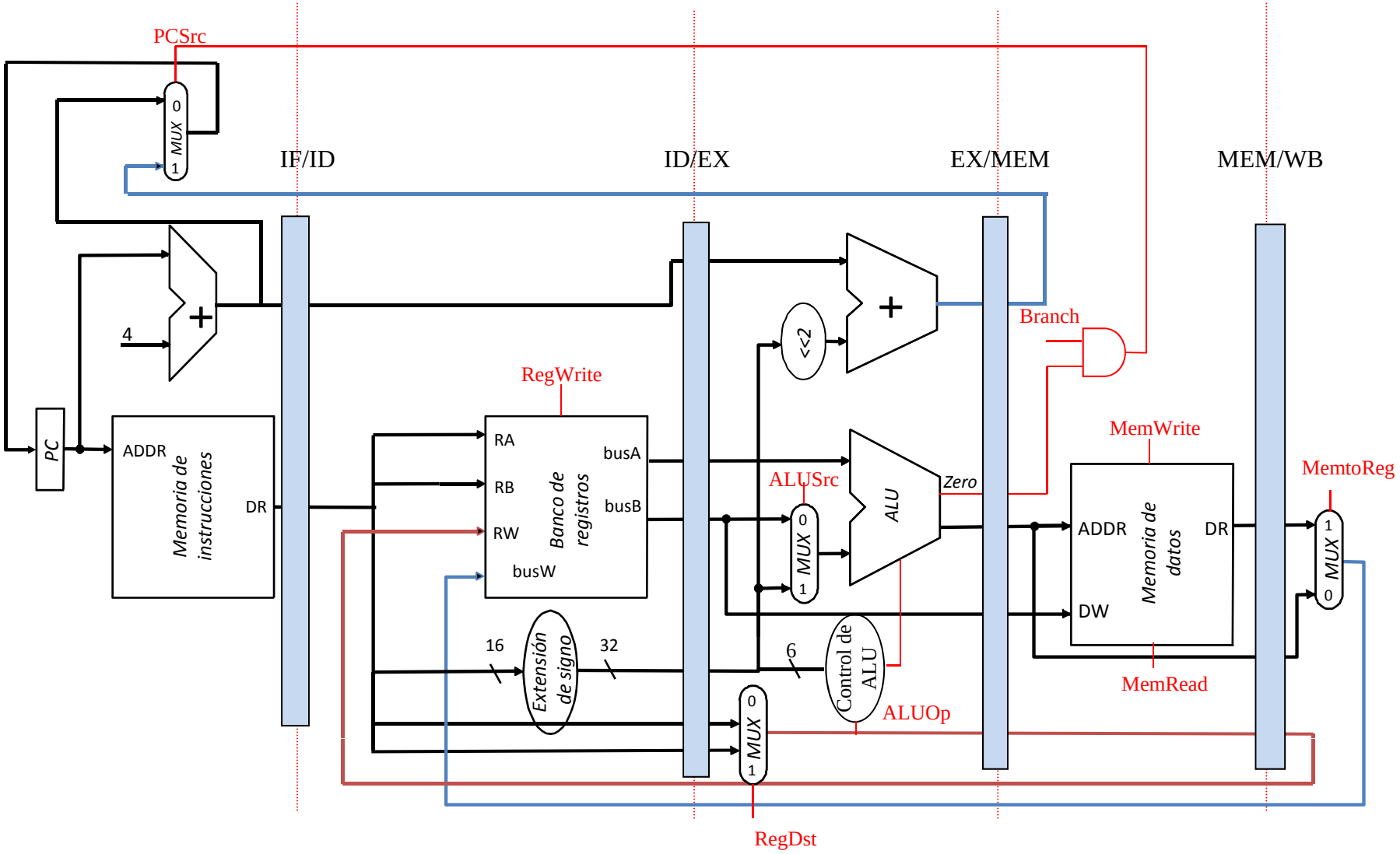
- ✓ **Duplicar los recursos que se necesitan en el mismo ciclo**



4. Segmentación: Riesgos estructurales

Diseño de control segmentado

✓ El PC y los diferentes registros de desacoplo (IF/ID ..) se deben cargar en cada ciclo, por tanto no necesitan orden específico de carga



4. Segmentación: Riesgos estructurales

Diseño de control segmentado

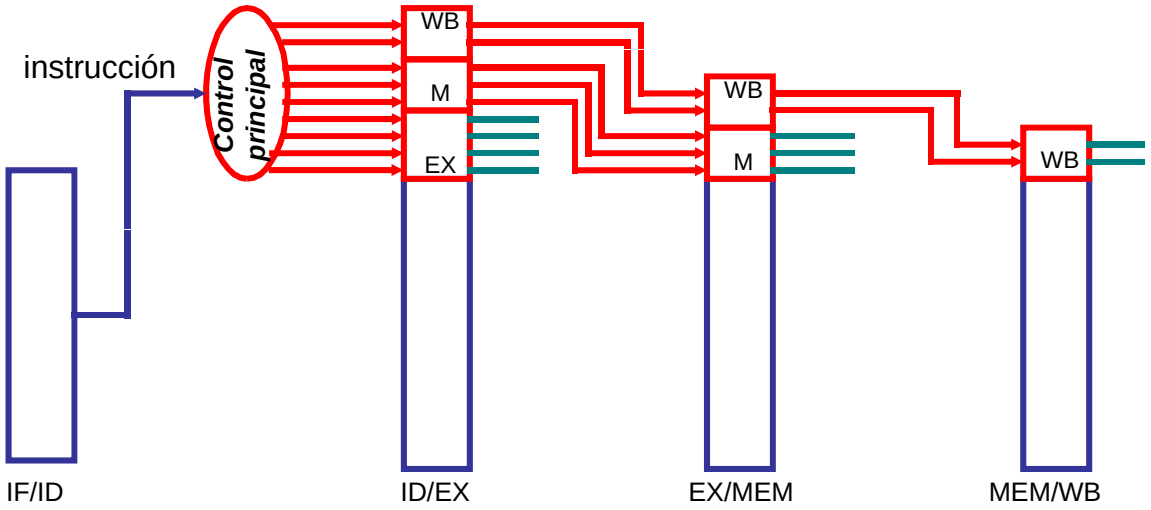
Control de la ALU

op	funct	ALUop	ALUctr
100011 (lw)	XXXXXX	00	010+
101011 (sw)		00	010 +
000100 (beq)		01	110-
000000 (tipo-R)	100000 (add)	10	010+
	100010 (sub)	10	110-
	100100 (and)	10	000
	100101 (or)	10	001
	101010 (slt)	10	111

El control de la alu se determina por ALUop (que depende del tipo de instrucción) y el campo de función en las instrucciones de tipo-R

Control principal

op	RegDst	ALUSrc	ALUop	MemRead	MemWrite	Branch	RegWrite	MemtoReg
100011 (lw)	0	1	00	1	0	0	1	0
101011 (sw)	X	1	00	0	1	0	0	X
000100 (beq)	X	0	01	0	0	1	0	X
000000 (tipo-R)	1	0	10	0	0	0	1	1



EX: Ejecución/ Calculo de Dir

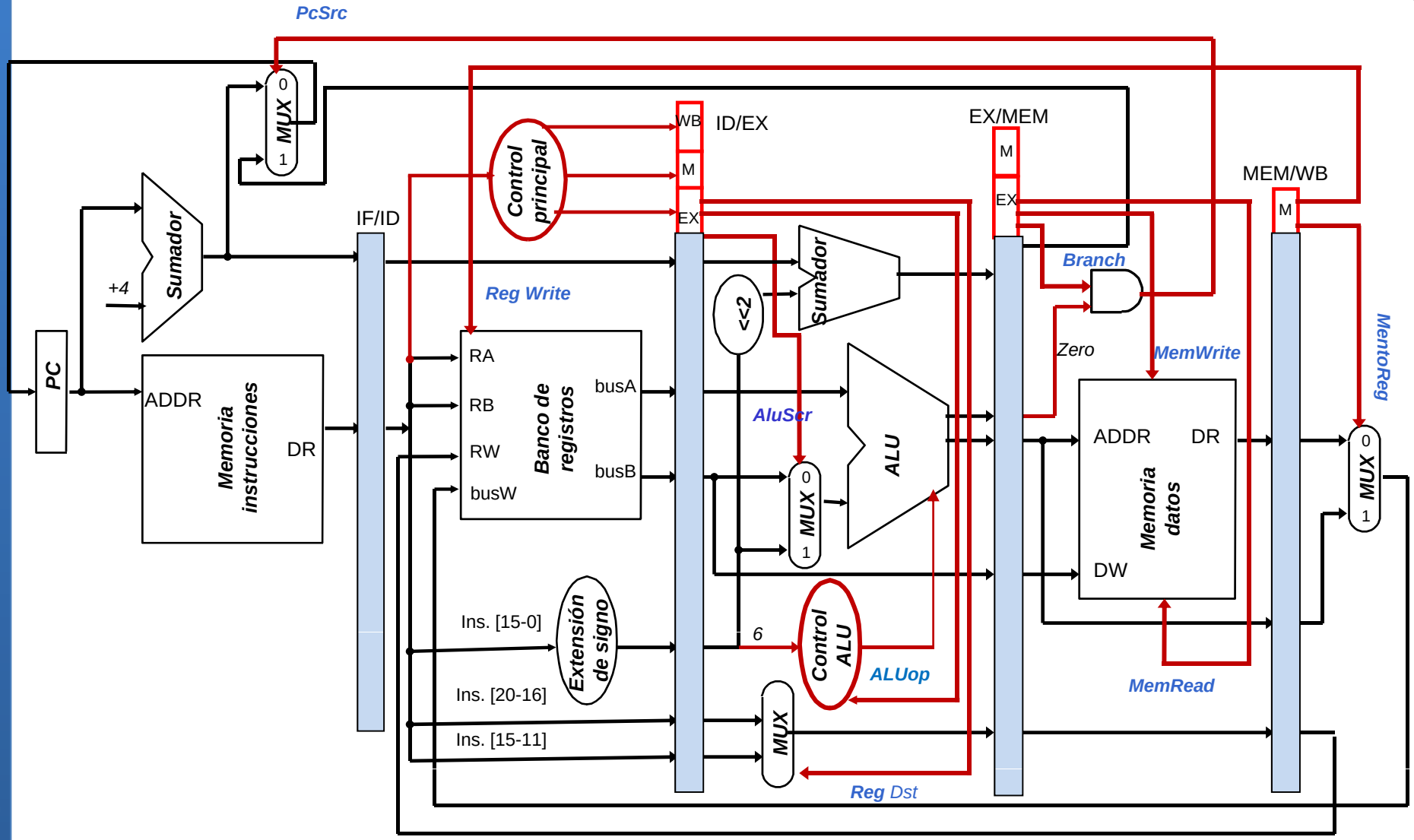
MEM: Acceso a Memoria

WB Escribe Reg

Control segmentado estacionario en los datos

4. Segmentación: Riesgos estructurales

Ruta de datos del procesador segmentado sin riesgos estructurales



5. Segmentación : Riesgos de datos

Riesgos de datos:

- Se produce cuando por la segmentación, el orden de LECTURA de los operandos y la ESCRITURA de resultados se modifica respecto al especificado por el programa
- Se produce un riesgo si existe dependencia entre instrucciones que se ejecutan concurrentemente
- Dependiendo del tipo de segmentación el riesgo se da o no.

- **Tres tipos** diferentes:
  - ✓ **Lectura después de escritura** **(LDE)**
  - ✓ **Escritura después de lectura** **(EDL)**
  - ✓ **Escritura después de escritura** **(EDE)**

5. Segmentación : Riesgos de datos

- **Lectura después de escritura** (LDE)

  *Add **r1**,r2,r3 – escribe el registro r1*

  *Add r4,**r1**,r2—lee el registro r1*

  - **Se produce riesgo si r1 se lee antes de que lo escriba la primera instrucción**

- **Escritura después de lectura** (EDL)

  *Add r1,**r4**,r3 – lee el registro r4*

  *Add **r4**,r1,r2—escribe el registro r4*

  - **Se produce riesgo si r4 se escribe antes de que lo lea la primera instrucción**

- **Escritura después de escritura** (EDE)

  *Add **r4**,r2,r3 – escribe el registro r4*

  *Add **r4**,r1,r2—escribe el registro r4*

  - **Se produce riesgo si r4 de la segunda instrucción se escribe antes de que lo escriba la primera instrucción**

5. Segmentación : Riesgos de datos

Los riesgos EDL y EDE

- **No se dan en el pipeline lineal**
  - Se leen los registros en el final de la segunda etapa
  - Todos las instrucciones escriben en la ultima etapa
  - Todas las instrucciones tienen igual duración

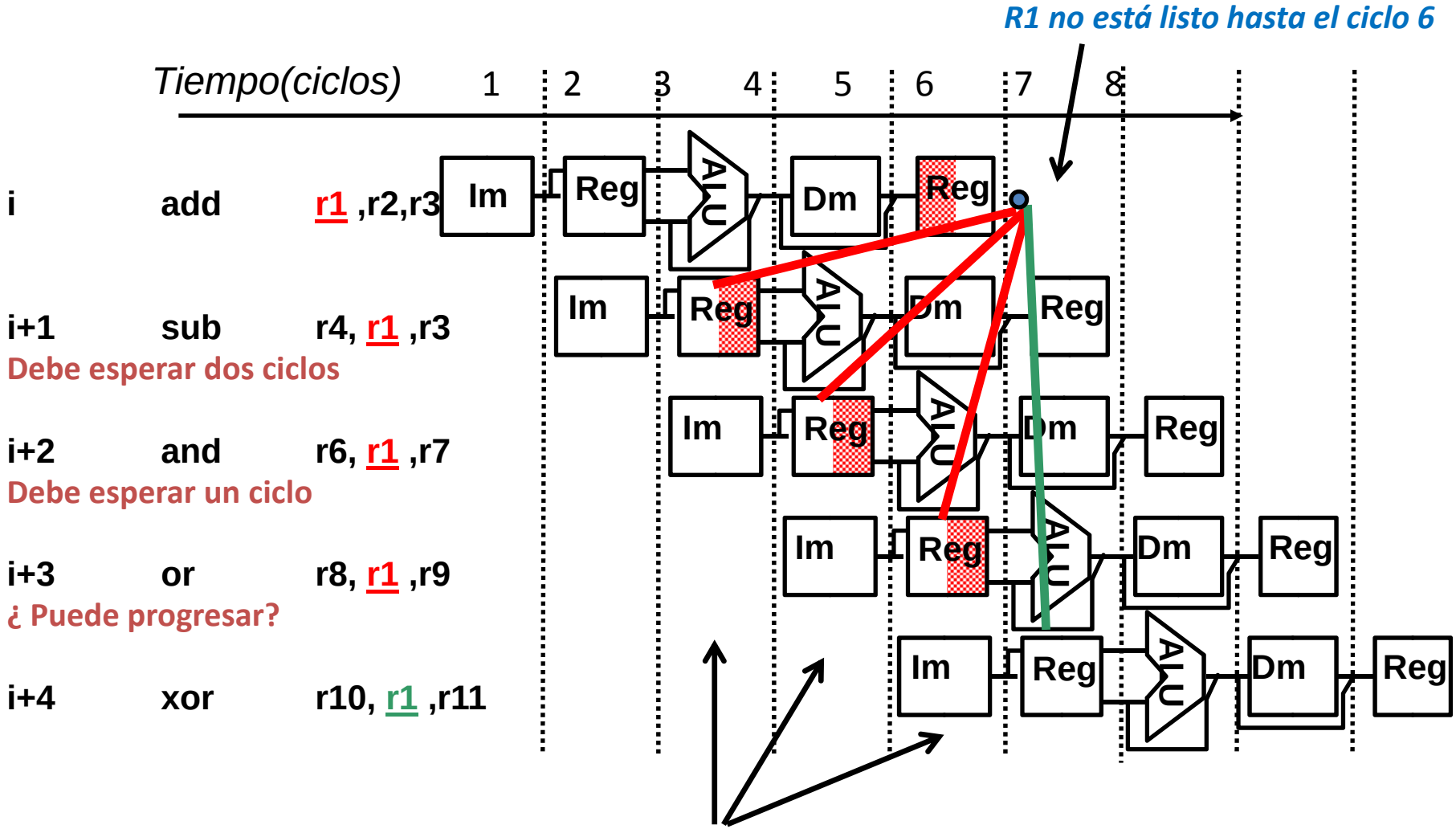
EDL	c1	c2	c3	c4	c5	c6
Add r1,r4,r3	búsqueda	**Dec/reg**	ejecución	memoria	escritura	
Sub r4,r1,r8		búsqueda	Dec/reg	ejecu	memoria	**escritura**

EDE	c1	c2	c3	c4	c5	c6
Add r4,r1,r3	búsqueda	Dec/reg	ejecución	memoria	**escritura**	
Sub r4,r1,r8		búsqueda	Dec/reg	ejecu	memoria	**escritura**

5. Segmentación : Riesgos de datos

Riesgos LDE

– Se dan en la siguiente situación



Las 3 siguientes instrucciones no tienen el valor de r1 disponible

5. Segmentación : Riesgos de datos

¿Cómo resolver los riesgos LDE?

- **Solución 1: Detener el pipeline**
  - **¿Cómo afectan las paradas a la ejecución de un programa?**
    - Las instrucciones que están en etapas anteriores a la etapa de parada también se paran
    - Las instrucciones que están en etapas posteriores a la etapa de parada siguen ejecutándose
    - En general supondremos que las paradas se realizan en decodificación

	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10
Add r1,r2,r3	IF	ID	EX	MEM	WB					
Sub r4,r1,r8		IF	ID_P	ID_P	ID	EX	MEM	WB		
Add r5,r6,r9			IF_P	IF_P	IF	ID	EX	MEM	WB	
Add r8,r7,r0						IF	ID	EX	MEM	WB

2 ciclos de espera

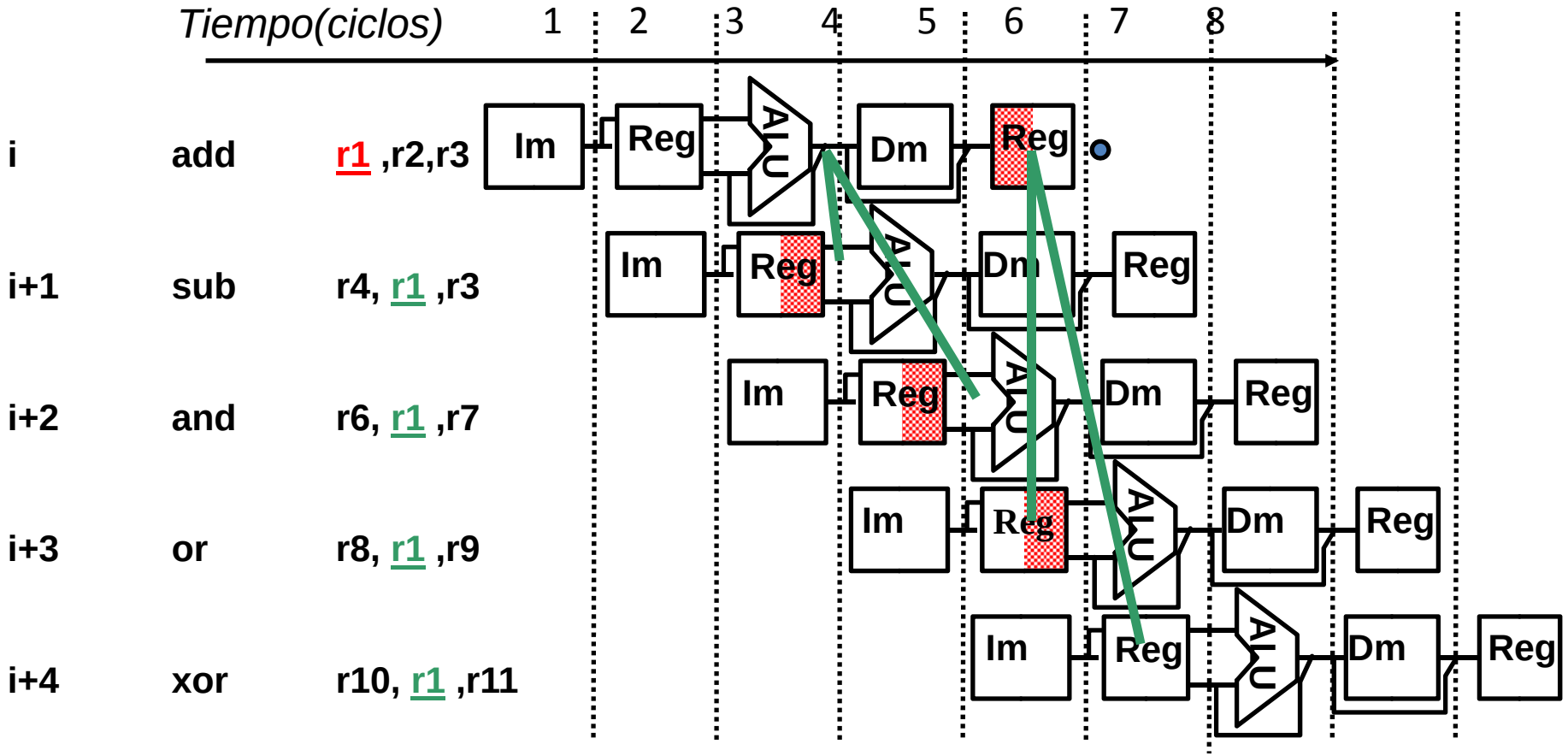
Sólo se está ejecutando la primera instrucción

- **Solución 2: Reordenar código**
  - Puede minimizar las paradas

5. Segmentación : Riesgos de datos

¿Cómo resolver los riesgos LDE?

- **Solución 3: Cortocircuito (forwarding)**
  - Enviar el dato a las etapas que lo necesiten cuando esté calculado sin esperar a WB
  - **El cortocircuito se realiza desde la salida del registro que contiene el resultado de la ALU**



Se pueden ejecutar todas las instrucciones sin problemas

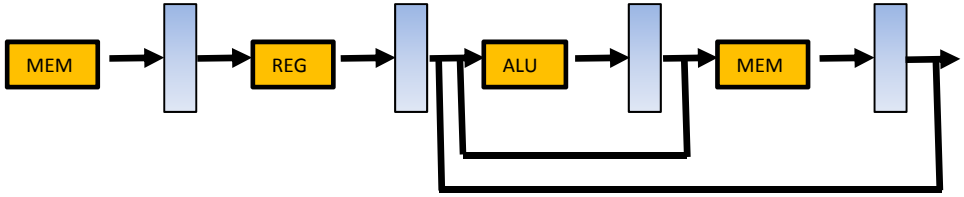
5. Segmentación : Riesgos de datos

Riesgos LDE: Instrucciones Aritméticas y Store

- Situación:

```
add r1,r2,r3
sub r4,r1,r3
and r6,r1,r7
```

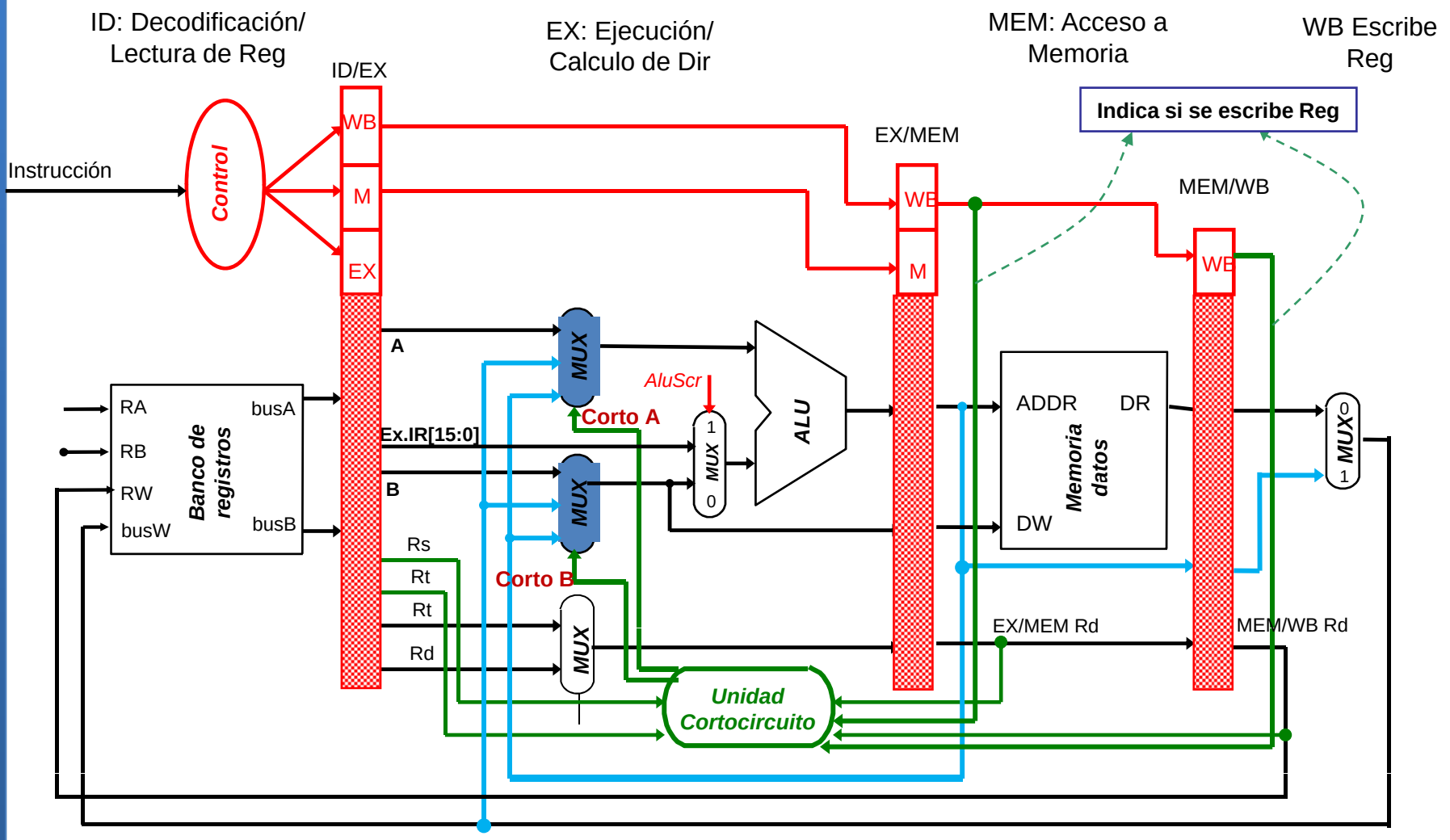
- **Dos caminos de datos:**
  - Desde el registro de pipeline EX/MEM (salida de la ALU) a entrada ALU
  - Desde el registro de pipeline MEM/WB (salida de la memoria) a entrada ALU



- **Información necesaria:**
  - Registro a escribir en última etapa (Rd en Tipo-R y Rt en Lw)
    - EX/MEM.Rd
    - MEM/WB.Rd
  - Registros que se leen en segunda etapa (Rs y Rt)
    - ID/EX.Rt
    - ID/EX.Rs
  - Información sobre si se escribe en el banco de registros
    - EX/MEM.RegWrite
    - MEM/WB.RegWrite

5. Segmentación : Riesgos de datos

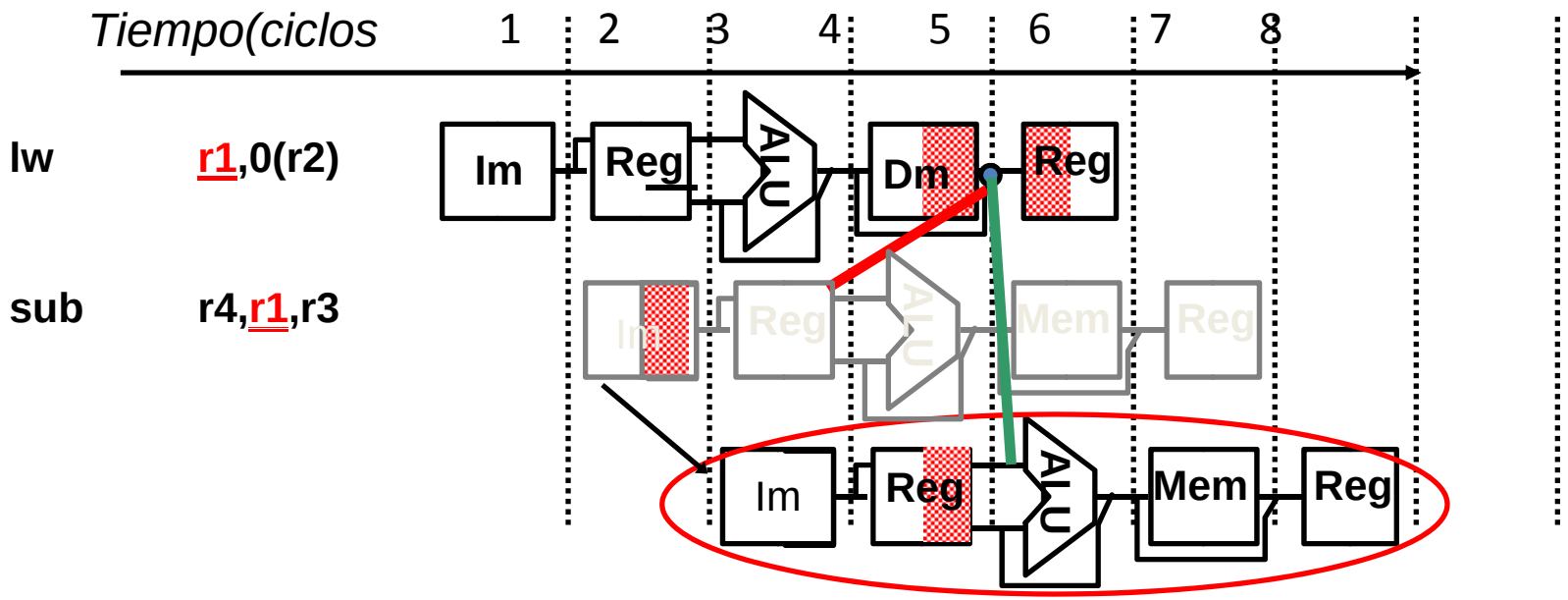
Riesgos LDE: Implementación del cortocircuito



5. Segmentación : Riesgos de datos

Riesgo LDE: Caso del Load

- No se puede resolver sólo con el cortocircuito si las instrucciones están consecutivas
  - La instrucción dependiente del Load debe esperar un ciclo
- Sí se puede resolver con cortocircuito si hay una instrucción intermedia



En este caso, aunque esté implementado el cortocircuito
Se tiene que esperar un ciclo

5. Segmentación : Riesgos de datos

Detector de riesgos LDE para el caso del Load

–Debe actuar en la etapa D impidiendo la carga y lectura de una nueva instrucción

- **Provoca una parada en la búsqueda y decodificación de instrucciones**

– **Información necesaria**:

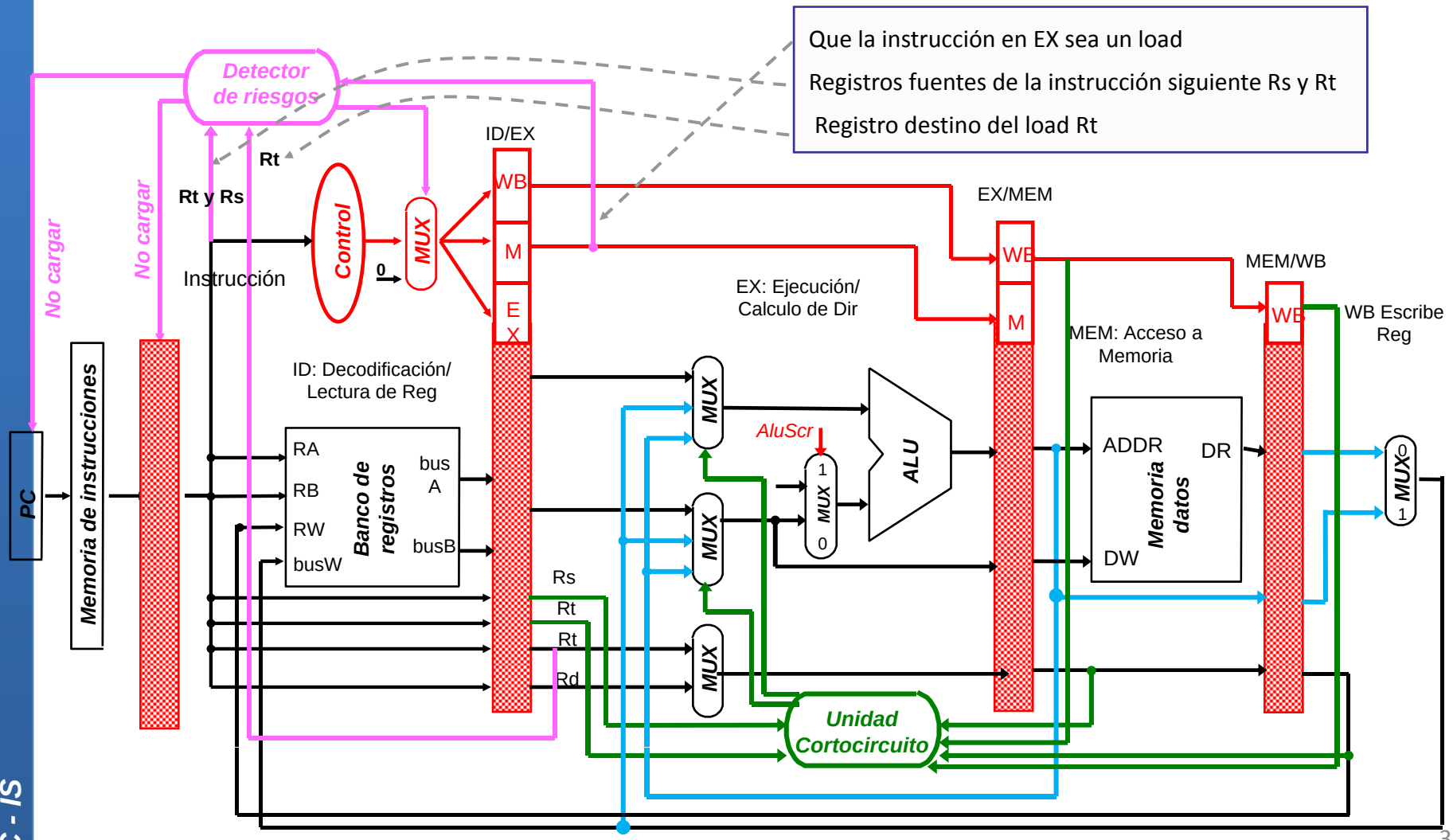
- Qué instrucción se está ejecutando en EX
- Que los registros fuentes de la instrucción en decodificación (Rs y Rt) tienen que coincidir con el registro destino del load que está en ejecución (Rt)

	c1	c2	c3	c4	c5	c6	c7
Load **r1**, 0(r2)	IF	ID	EX	MEM	WB		
Sub r4,**r1**,r3		IF	ID_P	ID	EX	MEM	WB

5. Segmentación : Riesgos de datos

Detector de riesgos LDE para el caso del Load

- **Solución HW:** Detección de riesgos y parada del procesador un ciclo



5. Segmentación : Riesgos de datos

Detector de riesgos LDE para el caso del Load

– **Solución SW**: Anticipar el Load en la planificación de instrucciones que hace el compilador

a := b + c
d := e - f

Antes:

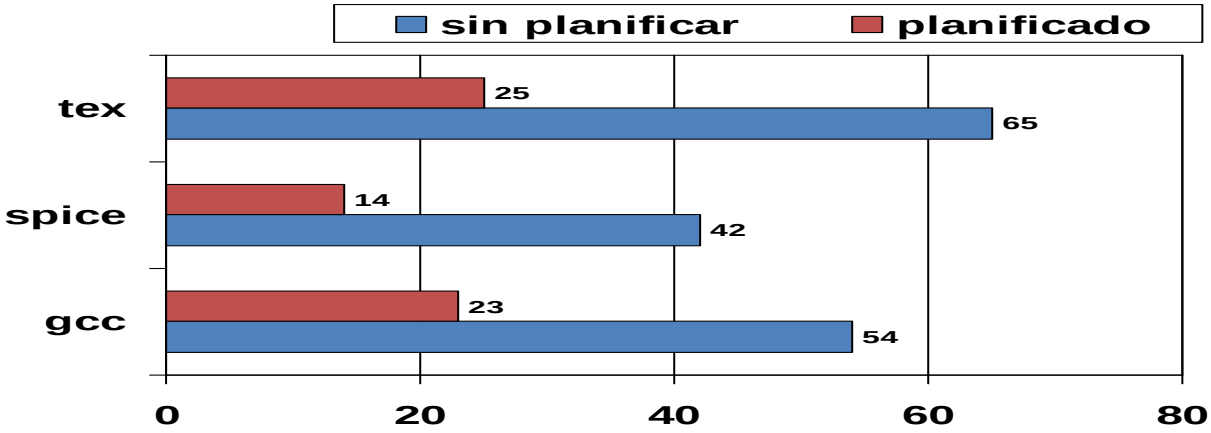
```
LW    Rb,b
LW    Rc,c
ADD   Ra,Rb,Rc
SW    a,Ra
LW    Re,e
LW    Rf,f
SUB   Rd,Re,Rf
SW    d,Rd
```

Despues:

```
LW    Rb,b
LW    Rc,c
LW    Re,e
ADD   Ra,Rb,Rc
LW    Rf,f
SW    a,Ra
SUB   Rd,Re,Rf
SW    d,Rd
```

Los stores no presentan riesgo LDE porque existe cortocircuito

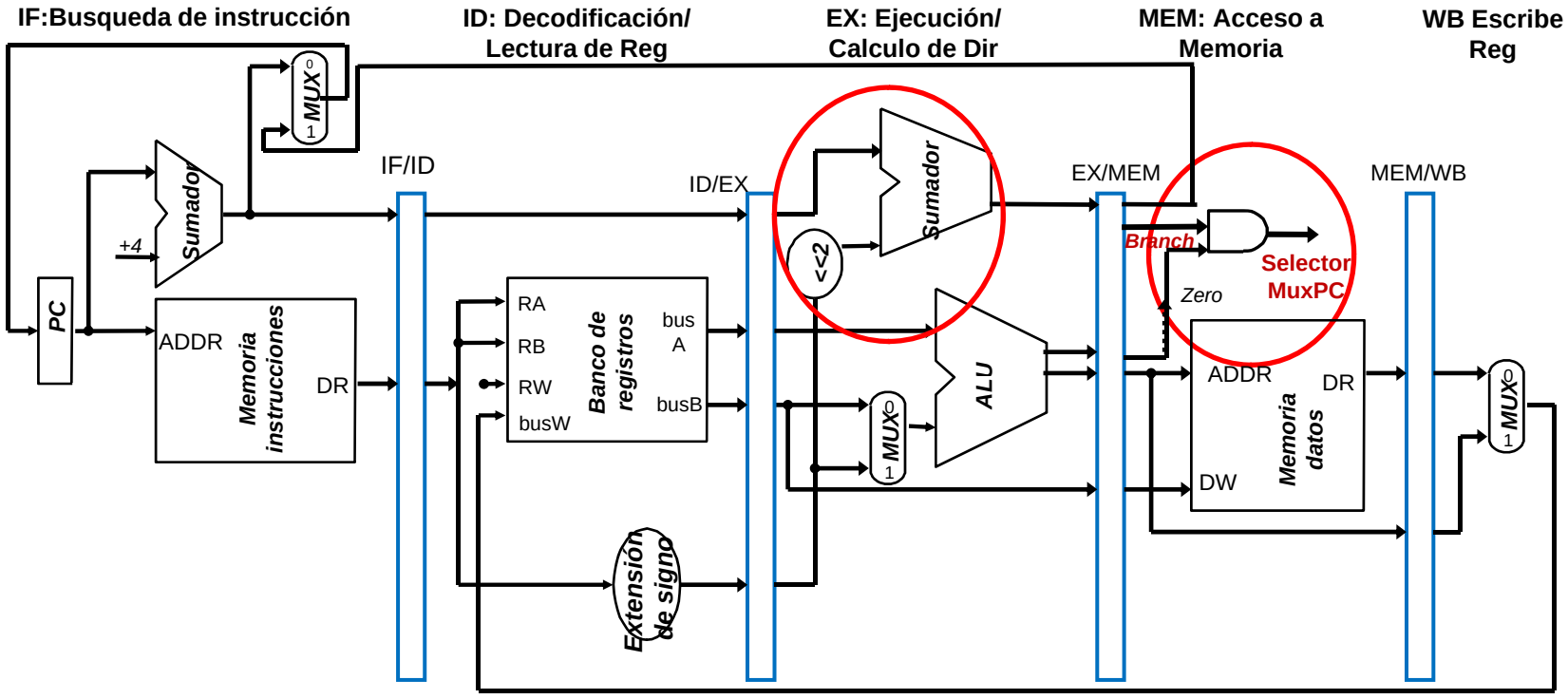
% de load que provocan un ciclo de parada



6. Segmentación : Riesgos de control

Riesgos de control: **¿Por qué aparecen?**

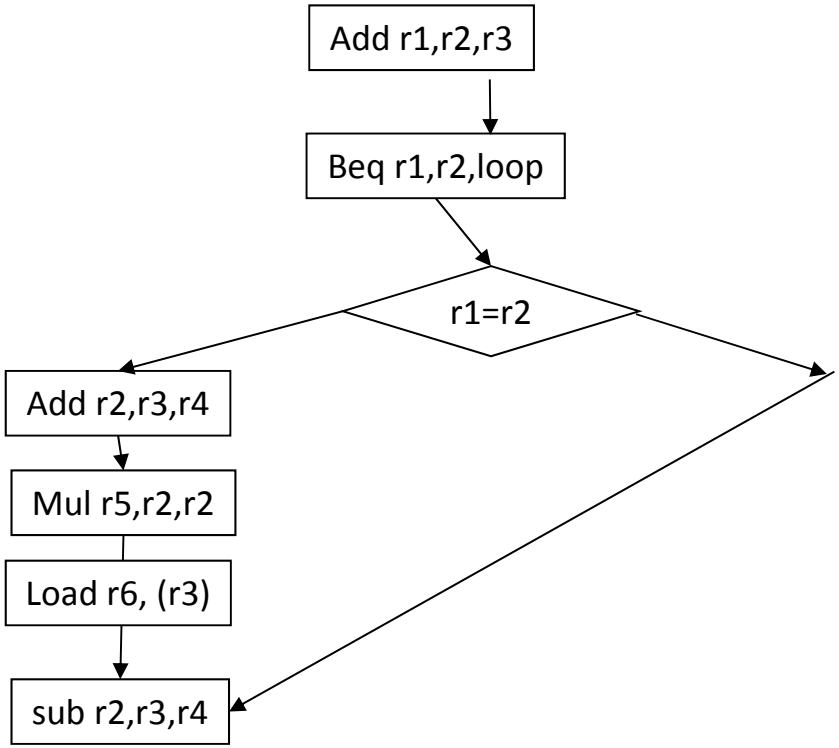
- Para saltar se necesita:
  - Haber calculado la **dirección de salto**
  - Saber **si se cumple la condición** de salto
- **Aparece el riesgo porque**
  - En la ruta de datos que hemos estudiado **ambos datos se calculan en ejecución**
  - **Si el salto se toma, el destino del salto se carga en PC en la etapa Mem**



6. Segmentación : Riesgos de control

Ejemplo:

```
        Add r1,r2,r3
        Beq r1,r2,loop
        Add r2,r3,r4
        Mul r5,r2,r2
        load r6, (r3)
Loop    sub r2,r3,r4
```

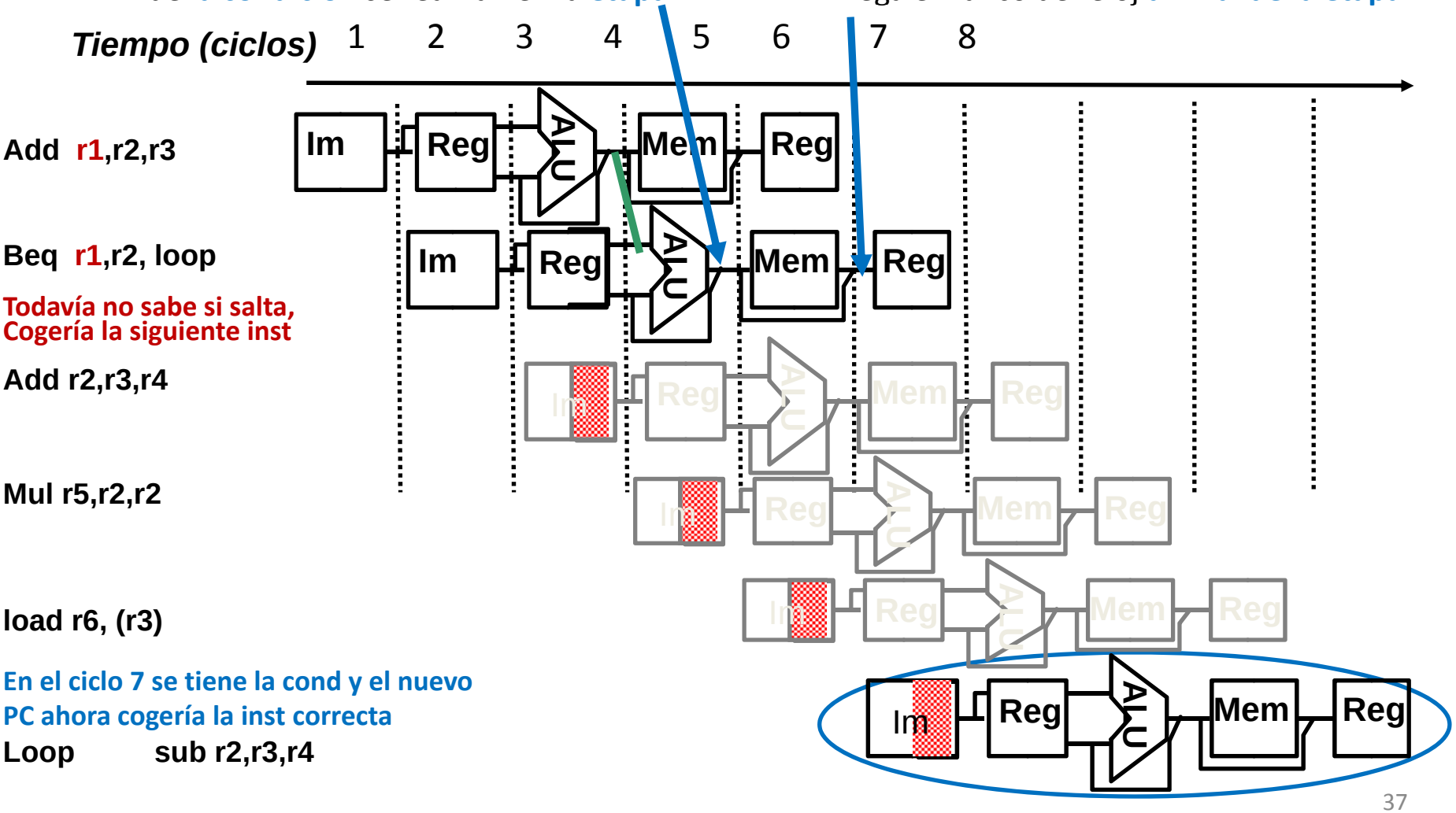


6. Segmentación : Riesgos de control

Ejemplo:

El calculo de la **dirección de salto** y la evaluación de **la condición** se realizan en la **etapa EX**

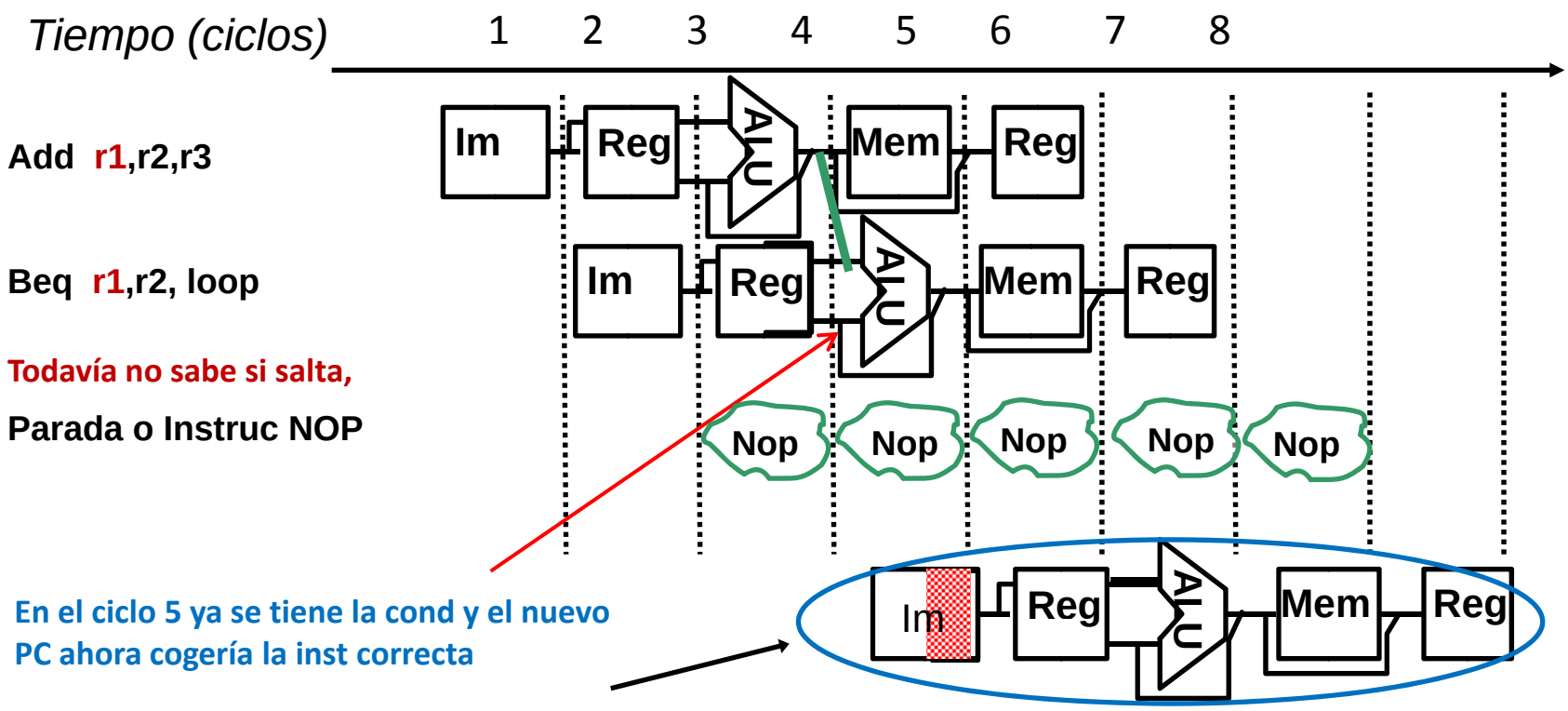
La nueva dirección se carga en PC cuando llega el flanco de reloj **al final de la etapa M**



6. Segmentación : Riesgos de control

¿Cómo solucionar los riesgos de control?

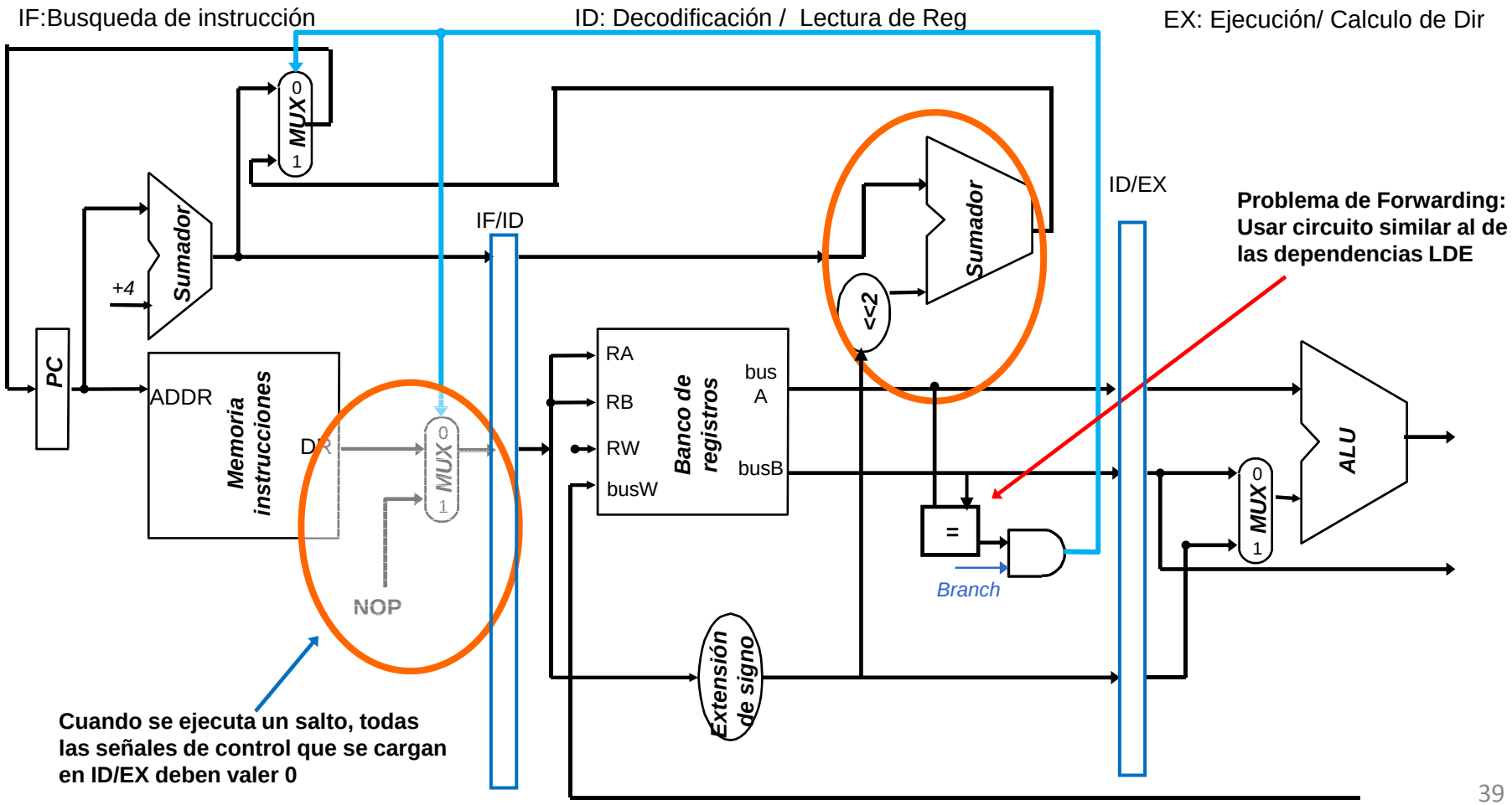
- **Solución 1**: Hay que **esperar tres ciclos** para saber la instrucción que sigue a la de salto
- **Solución 2**: **Branch Delay slot**
  - Desplazar el **calculo de la dirección** y la evaluación de **la condición** a la etapa **Dec/Reg**
  - **Sólo hay que esperar un ciclo para saber la instrucción que sigue a la de salto**



6. Segmentación : Riesgos de control

Implementar la solución 2

- Calcular la dirección de salto y la evaluación de la condición en la etapa ID
- Permitir **introducir instrucciones NOP** en el pipeline



6. Segmentación : Riesgos de control

¿Cómo mejorar el rendimiento en los saltos?

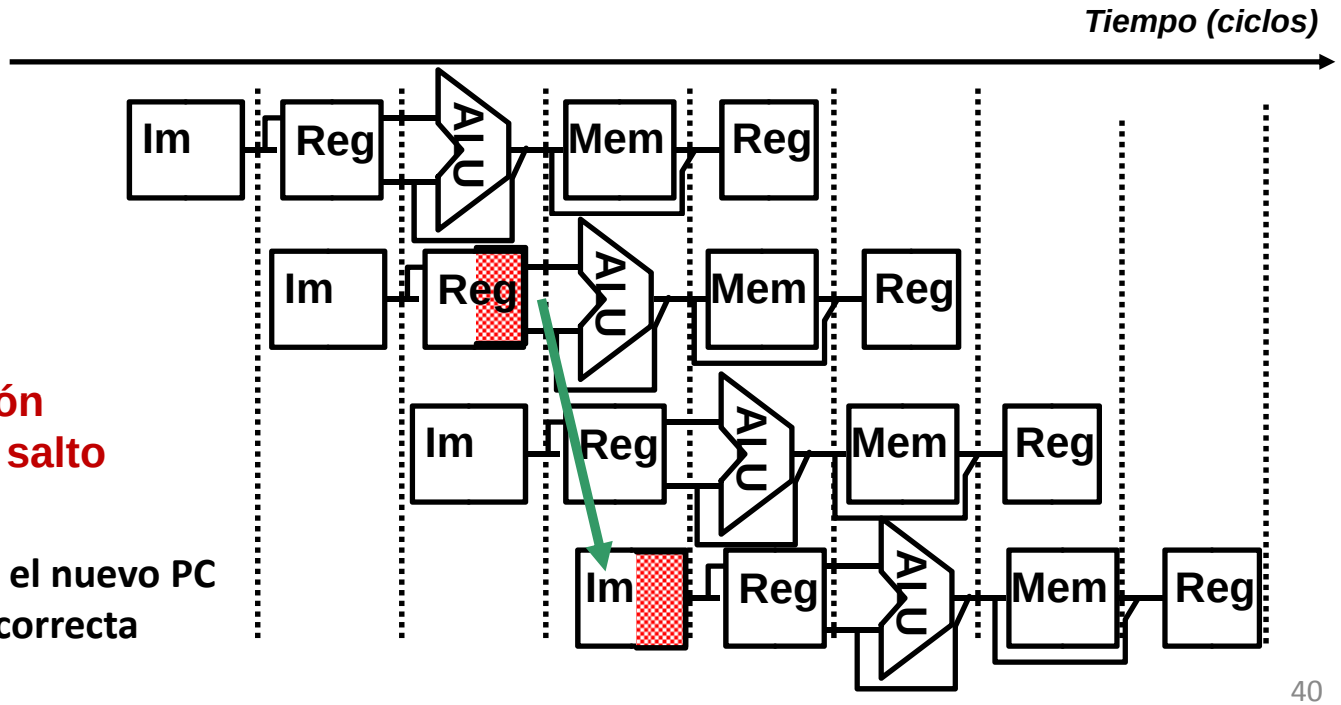
- **Solución SW:** Saltos retardados
  - Ejecutar instrucciones ***independientes del salto*, *en vez de NOP,*** durante los ciclos de retardo
    - Estas instrucciones se ejecutarán siempre, tanto si el salto es tomado como si no
    - ***Si es posible encontrar una instrucción*** ⇒ ***0 ciclos de penalización***

Add r1,r2,r3

Beq r1,r2,loop

Cualquier instrucción no dependiente del salto

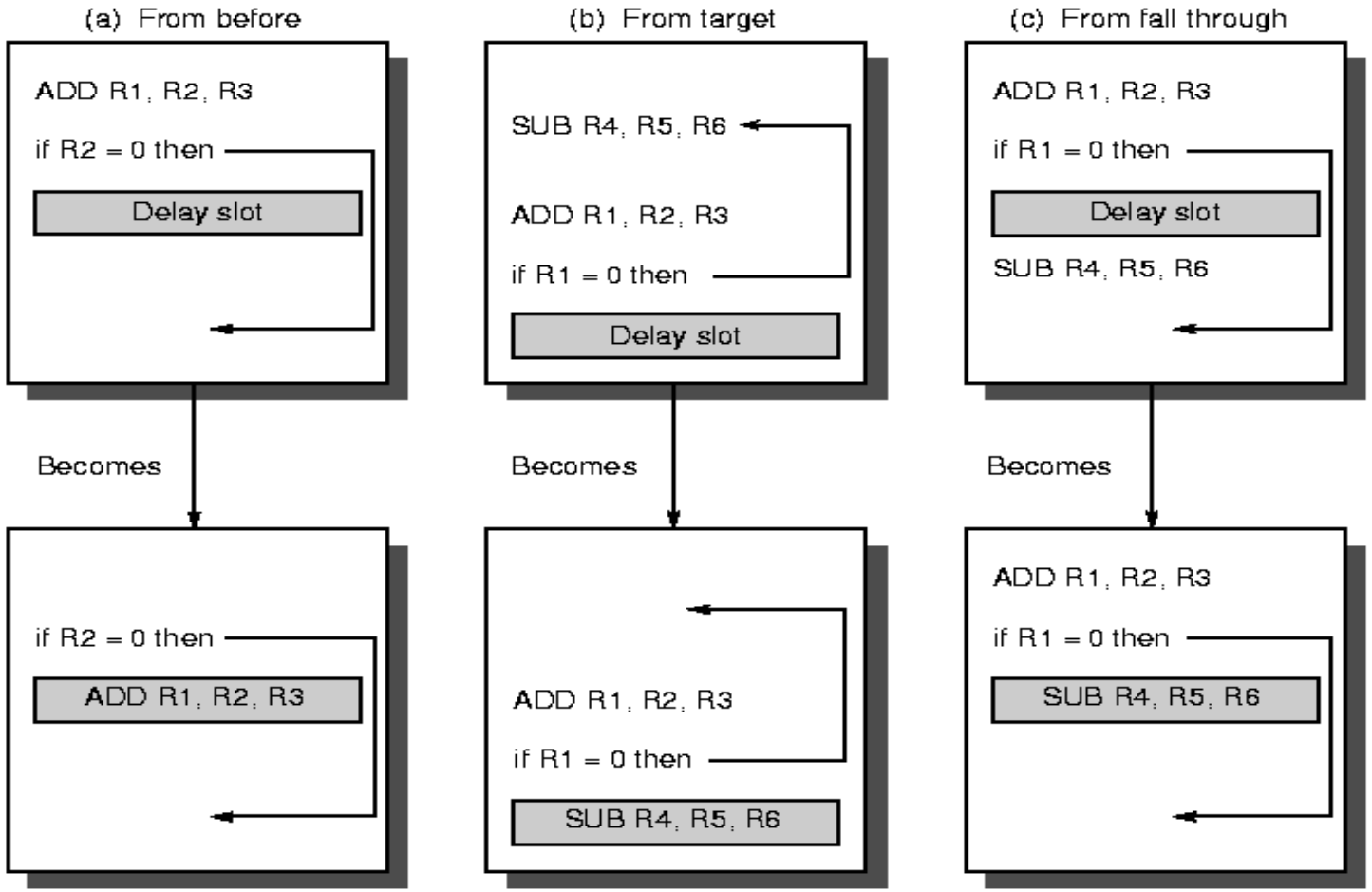
Aquí ya se tiene la cond y el nuevo PC ahora cogería la inst correcta



6. Segmentación : Riesgos de control

Saltos retardados

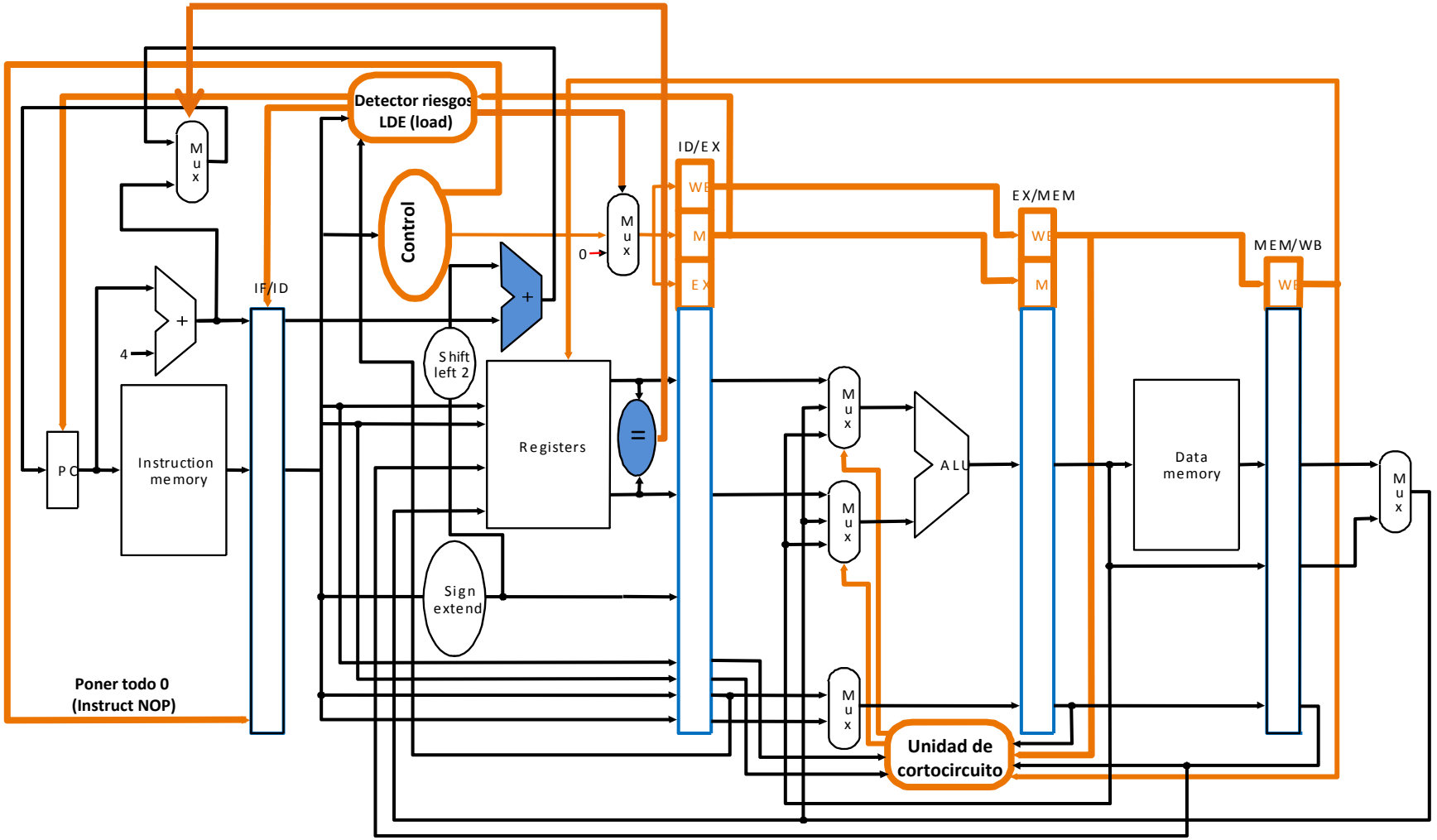
– El compilador debe encargarse de elegir adecuadamente las instrucciones que se planifican en los ciclos de retardo



7. Resumen

Ruta datos completa del procesador segmentado

- Tiene en cuenta todos los riesgos vistos en este tema



7. Resumen

Procesador segmentado

- ✓ Todas las instrucciones tienen igual duración
- ✓ Rendimiento ideal, una instrucción por ciclo CPI=1
- ✓ Riesgos estructurales y de datos EDE y EDL se resuelven por construcción
- ✓ Riesgos LDE en instrucciones tipo-R se solucionan con el cortocircuito
- ✓ Riesgos LDE en instrucciones de *load* implican paradas del procesador. Ayuda del compilador planificando las instrucciones
- ✓ Riesgos de control. Paradas y saltos retardados con ayuda del compilador
- ✓ **Las instrucciones empiezan y terminan en orden**