ESTRUCTURA DE COMPUTADORES

Tema 3. Segmentación

Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

ÍNDICE

1. MIPS
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Riesgos de control con riesgos LDE
7. Operaciones multiciclo
8. Excepciones
9. Rendimiento en los procesadores

Bibliografía

- Hennessy, J. L., Patterson, D. “Computer Architecture: A Quantitative Approach”, 5th ed., Morgan Kaufmann, 2012. ISBN 978-0-12-383872-8. Capitulo 1
- Paterson, D . Hennessy, J. L. “Computer Organization and Design” , 5th ed., Morgan Kaufmann, 2014, ISBN 0780124077263. Capítulo 1.

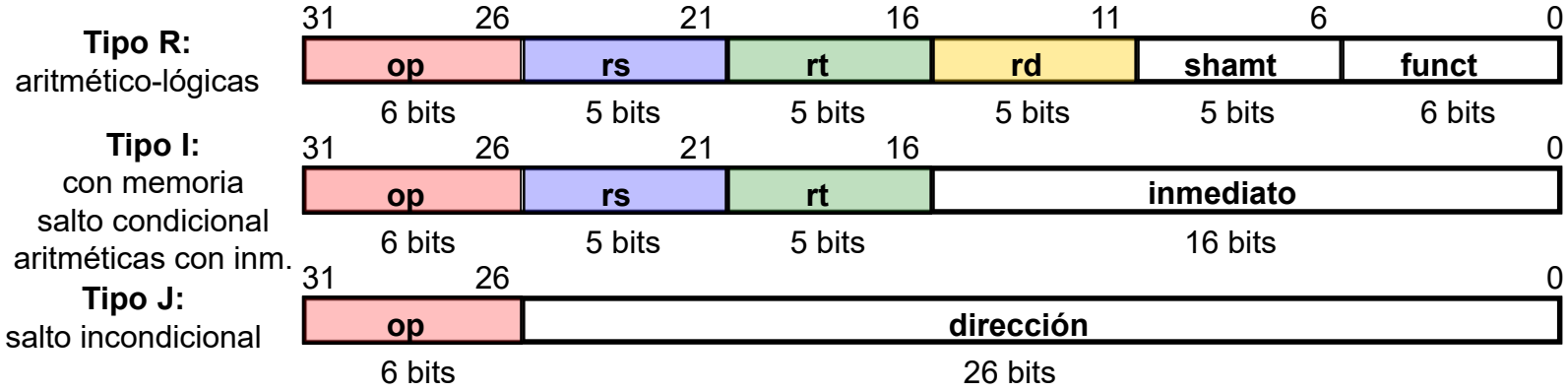
ÍNDICE

MIPS: REPERTORIO IMPLEMENTADO

- **Instrucciones aritmético-lógicas con operandos en registros**
  - add rd, rs, rt — rd ← rs + rt, PC ← PC + 4
  - sub rd, rs, rt — rd ← rs - rt , PC ← PC + 4
  - and rd, rs, rt — rd ← rs and rt , PC ← PC + 4
  - or rd, rs, rt — rd ← rs or rt , PC ← PC + 4
  - slt rd, rs, rt — (si (rs < rt) entonces (rd ← 1) en otro caso (rd ← 0)), PC ← PC+4
- **Instrucciones con referencia a memoria** (tipo I)
  - lw rt, inmed(rs) — rt ← Memoria(rs + SignExt(inmed)) , PC ← PC + 4
  - sw rt, inmed(rs) — Memoria(rs + SignExt(inmed)) ← rt , PC ← PC + 4
- **Instrucciones de salto condicional** (tipo I)
  - beq rs, rt, inmed — si (rs = rt) entonces (PC ← PC + 4 + 4·SignExt(inmed)) en otro caso PC ← PC + 4

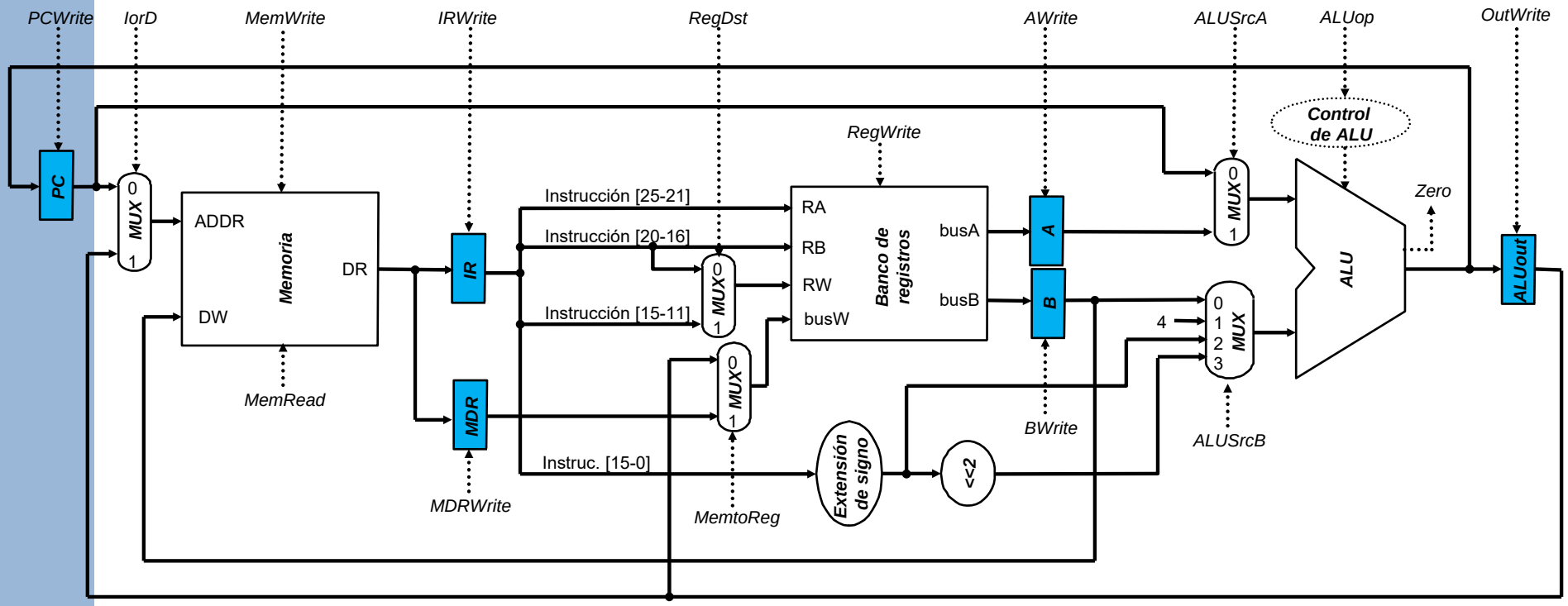
MIPS: FORMATO DE INSTRUCCIÓN

- Todas las instrucciones del repertorio del MIPS tienen 32 bits de anchura, repartidas en 3 formatos de instrucción diferentes:

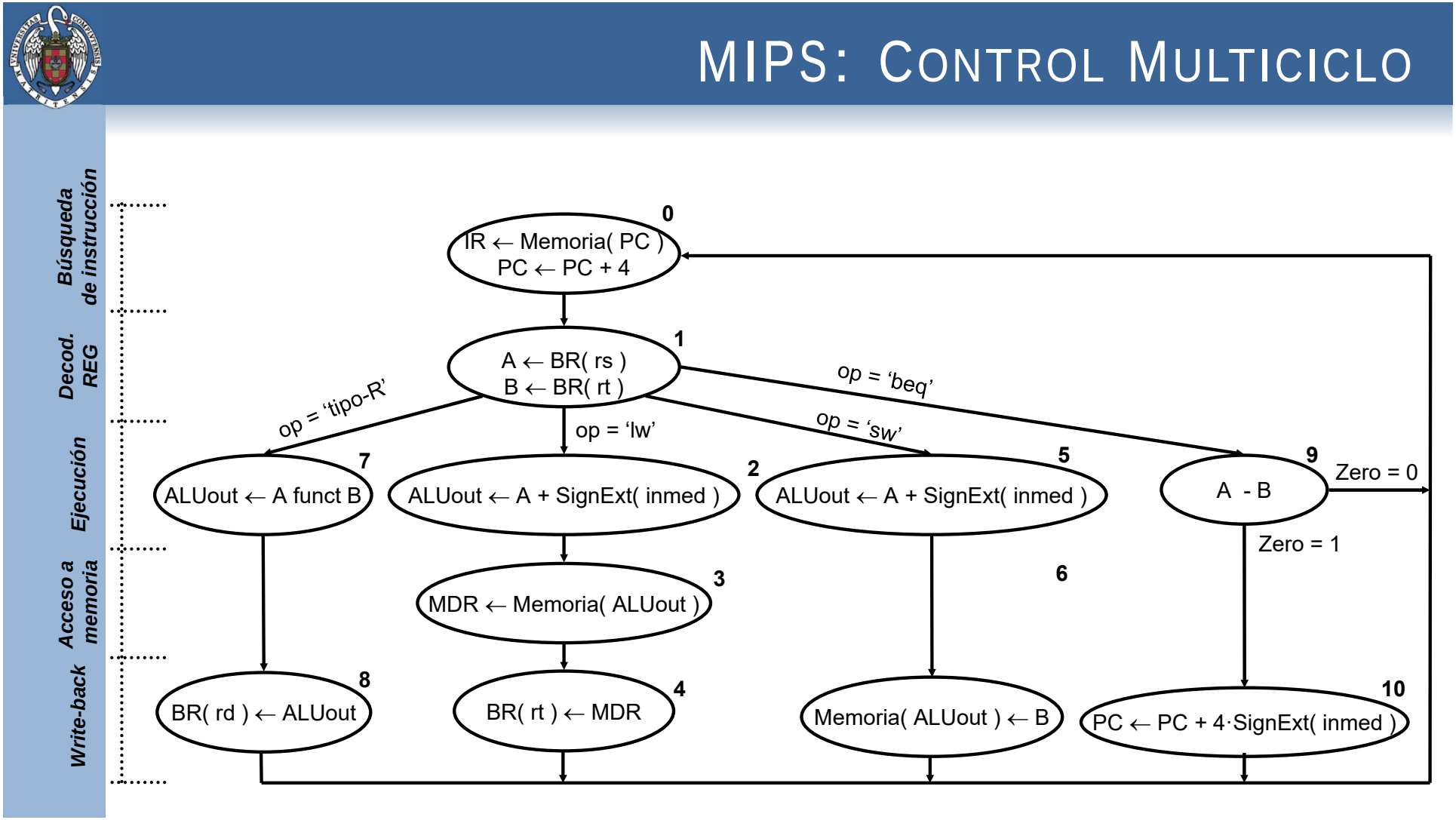


- El significado de los campos es:
  - **op**: identificador de instrucción
  - **rs**, **rt**, **rd**: identificadores de los registros fuentes y destino
  - **shamt**: cantidad a desplazar (en operaciones de desplazamiento)
  - **funct**: selecciona la operación aritmética a realizar
  - **inmediato**: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

MIPS: Ruta de Datos Multiciclo



MIPS: Control Multiciclo

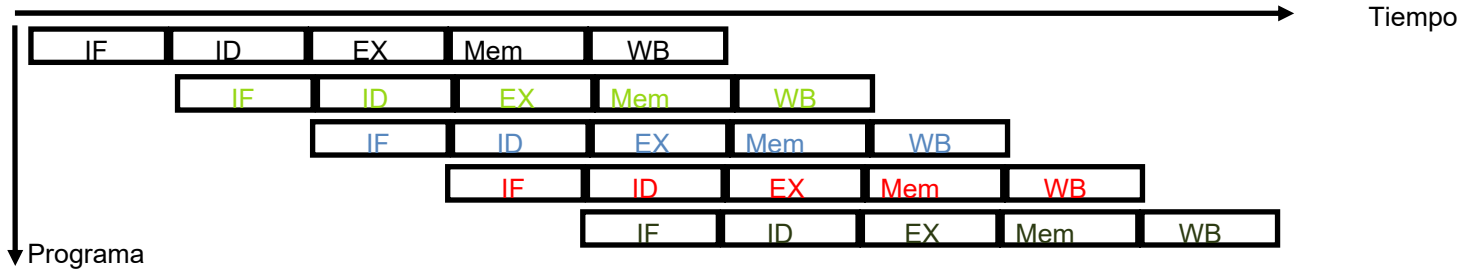


ÍNDICE

1. MIPS
2. **Segmentación**
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Riesgos de control con riesgos LDE
7. Operaciones multiciclo
8. Excepciones
9. Rendimiento en los procesadores

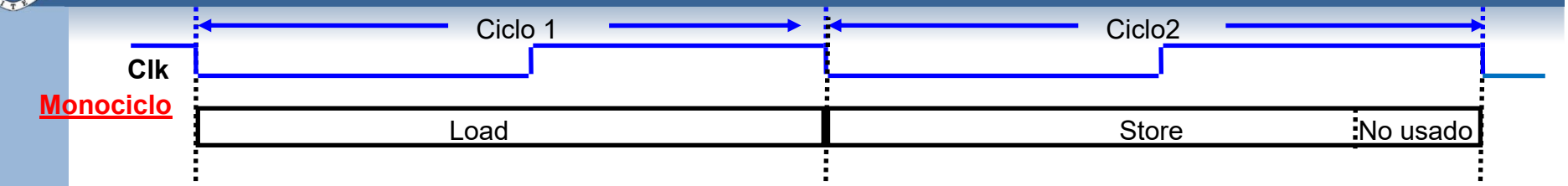
SEGMENTACIÓN

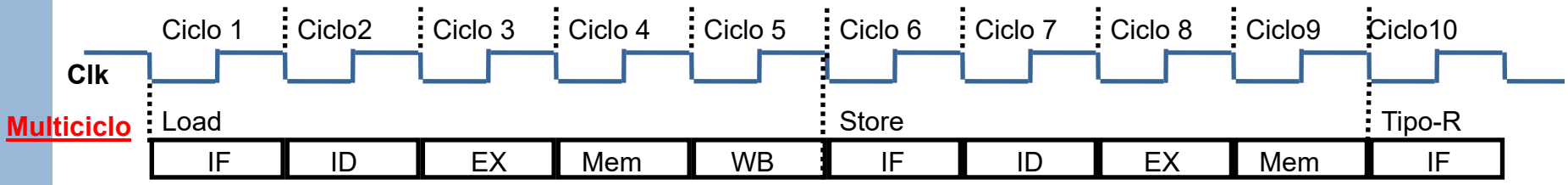
- Cada etapa opera en paralelo con otras etapas pero sobre instrucciones diferentes
- Una instrucción para ejecutarse tiene que atravesar todas y cada una de las etapas del pipeline
- El ciclo de reloj viene determinado por la etapa más lenta
- El orden de las etapas es el mismo para todas las instrucciones

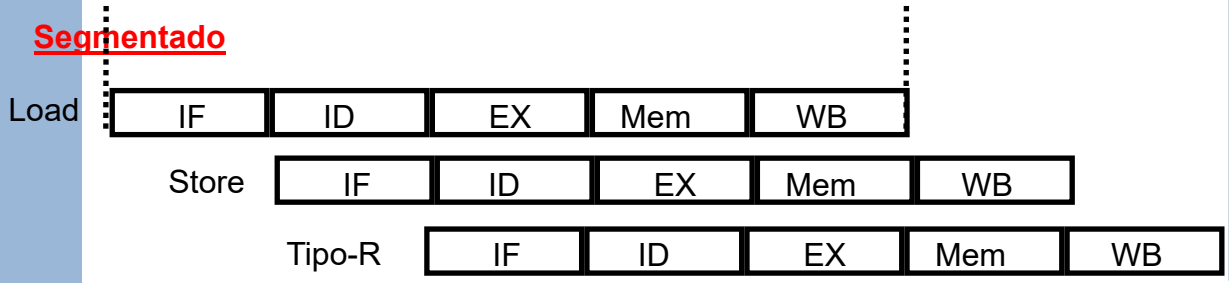


- A partir del ciclo 5
  - Sale una instrucción cada ciclo de reloj
  - CPI=1
- Los 4 primeros ciclos se llaman de llenado del pipeline.
- $CPI_{Ideal}=1$

SEGMENTACIÓN







100 instrucciones

- Monociclo
  45ns/ciclo x100 = 4500ns
- Multiciclo
  10ns/ciclo x 4.6 CPI x 100 =4600ns
- Segmentado
  10ns x (1CPIx100+4 llenado) =1040 ns

SEGMENTACIÓN

- ¿Qué facilita la segmentación?
  - Todas las instrucciones de igual anchura
  - Pocos formatos de instrucción
  - Búsqueda de operandos en memoria sólo en operaciones de carga y almacenamiento
- ¿Qué dificulta la segmentación?
  - Riesgos: Situaciones que impiden que en cada ciclo se inicie la ejecución de una nueva instrucción
    - **Estructurales**. Se producen cuando dos instrucciones tratan de utilizar el mismo recurso en el mismo ciclo.
    - **De datos**. Se producen al intentar utilizar un dato antes de que esté actualizado. Mantenimiento del orden estricto de lecturas y escrituras.
    - **De control**. Se producen al intentar tomar una decisión sobre una condición todavía no evaluada.

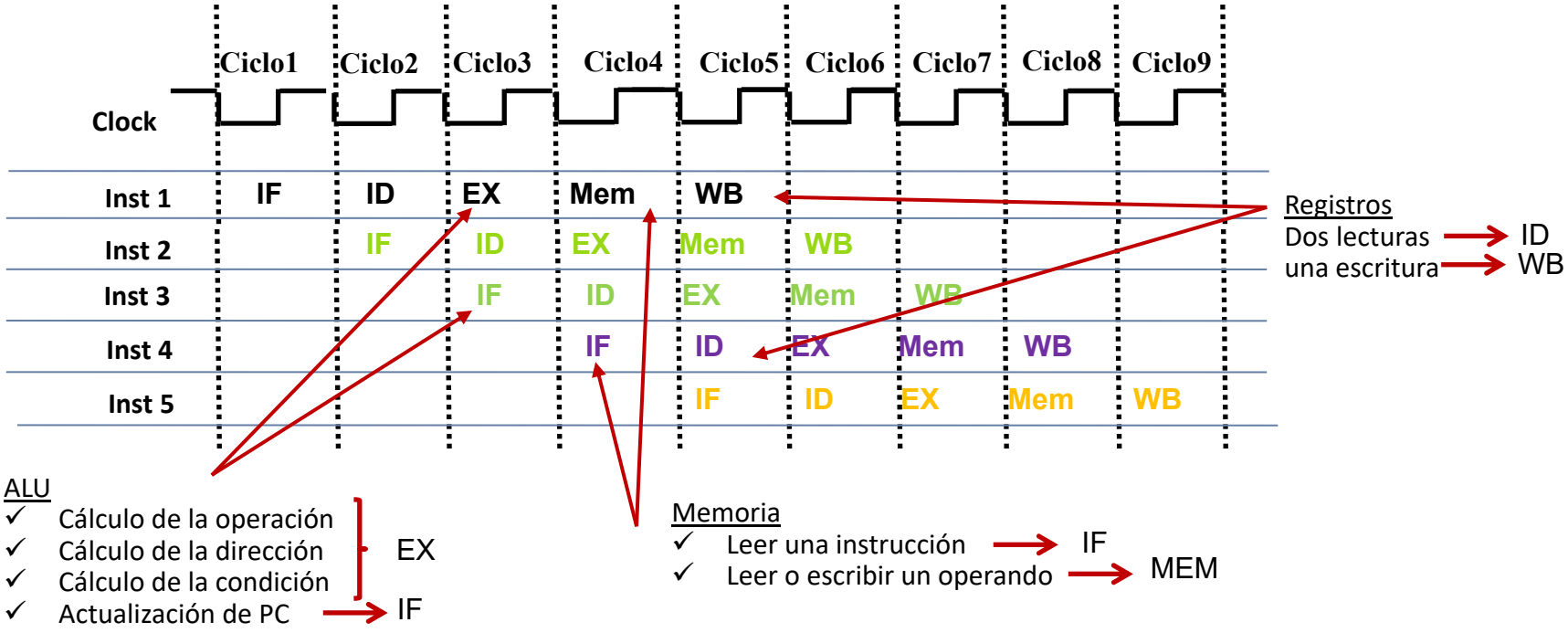
Los riesgos se deben detectar y resolver

- Gestión de interrupciones

ÍNDICE

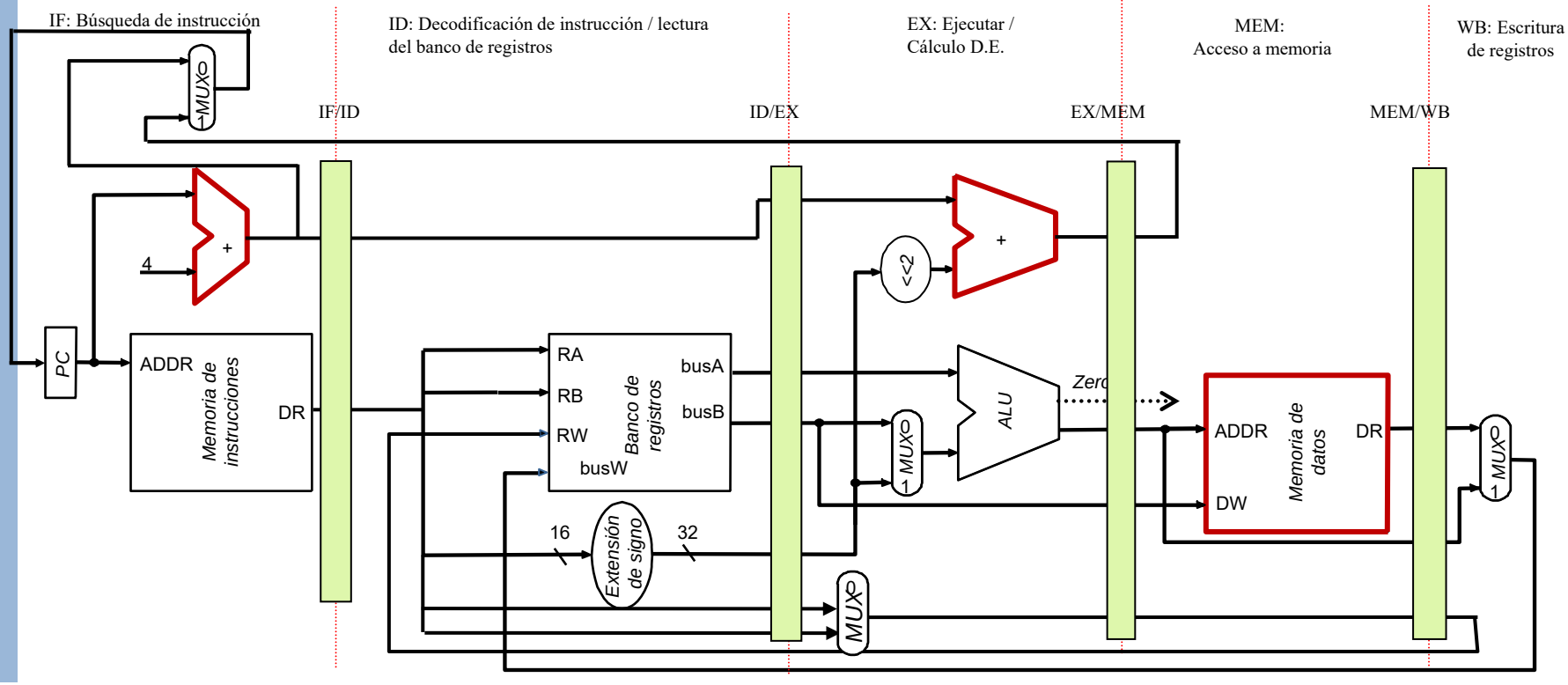
SEGMENTACIÓN: RIESGOS ESTRUCTURALES

- Se producen cuando dos instrucciones tratan de utilizar el mismo recurso en el mismo ciclo
  - ***Objetivo:*** **Ejecutar sin conflicto cualquier combinación de instrucciones**



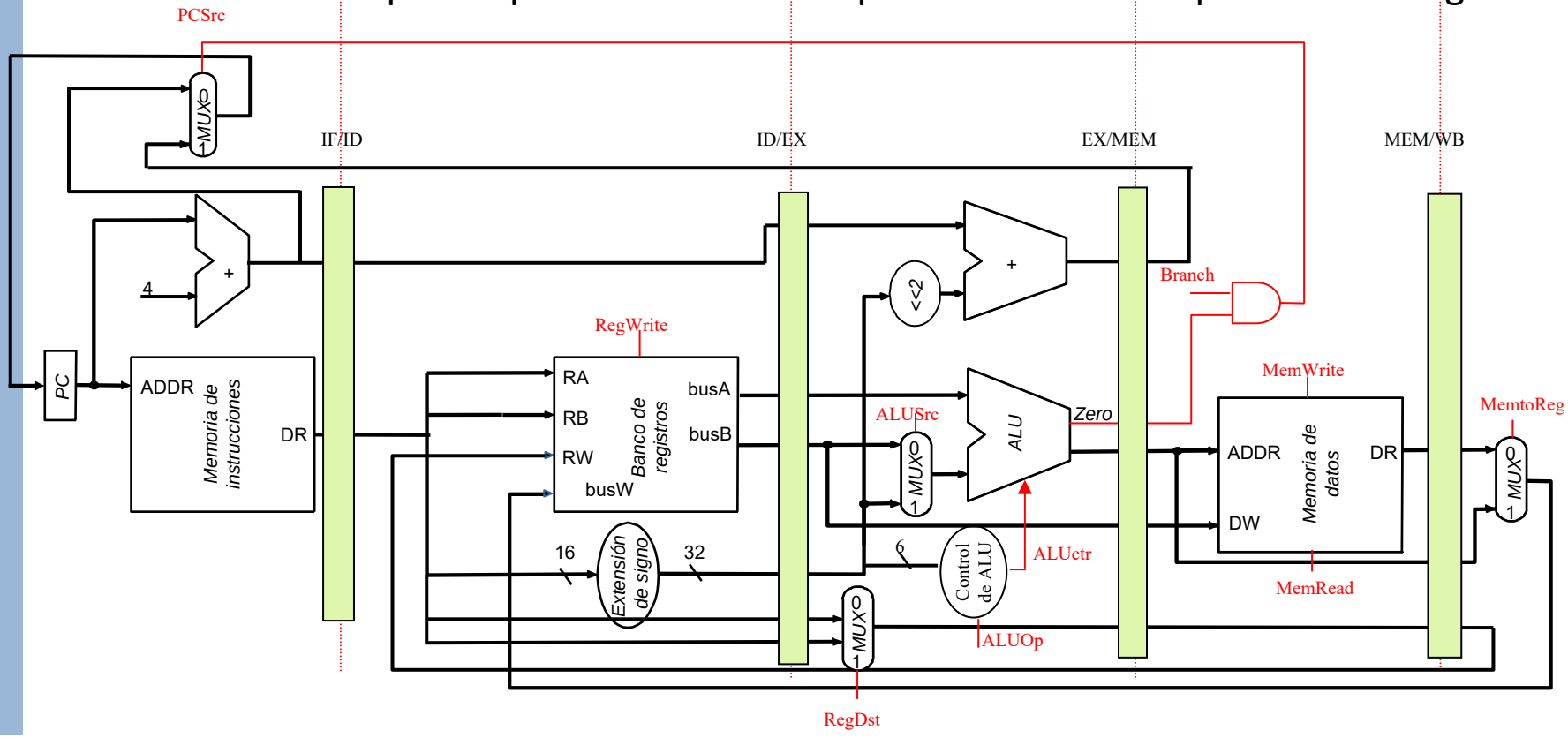
SEGMENTACIÓN: RIESGOS ESTRUCTURALES

- ¿Cómo resolver los riesgos estructurales?
  - Duplicar los recursos que se necesitan en el mismo ciclo: ALU y dos sumadores; memoria de instrucciones y de datos separadas.
  - El B.R. no es problema porque tiene puertos para leer de dos registros y escribir en un registro a la vez.



SEGMENTACIÓN: CONTROL

- El PC y los registros de desacoplo: IF/ID, ID/EX, EX/MEM,y MEM/WB se cargan en todos los ciclos por lo que no necesitan un punto de control específico de carga.



SEGMENTACIÓN: CONTROL

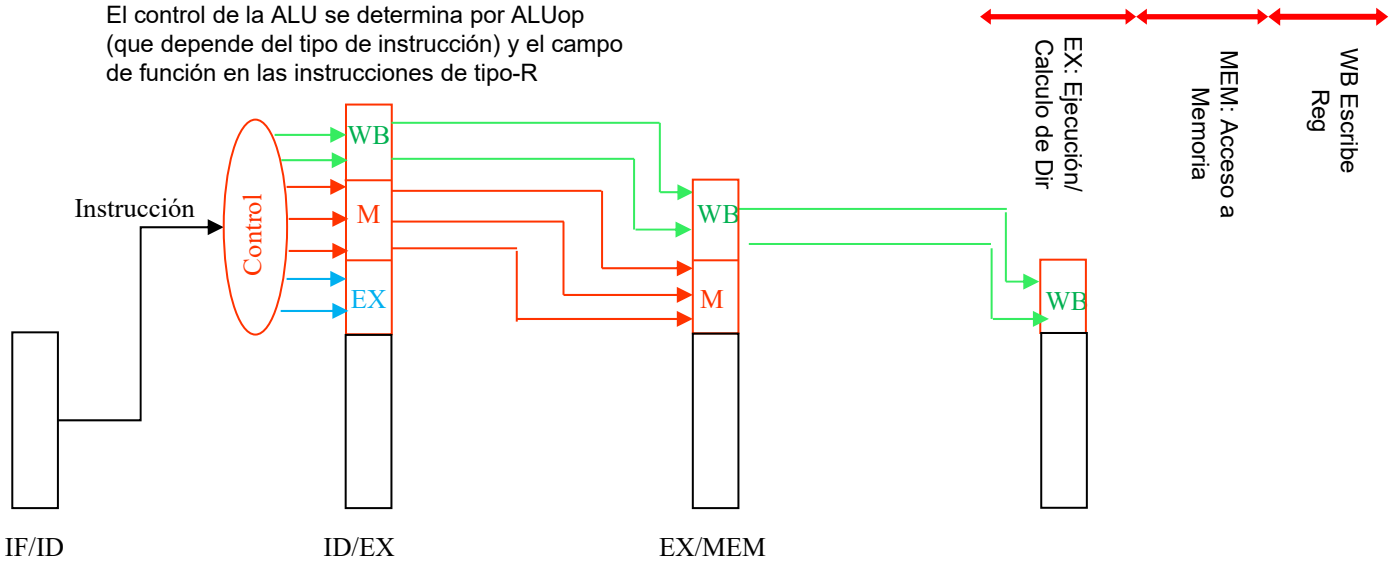
Control de la ALU

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

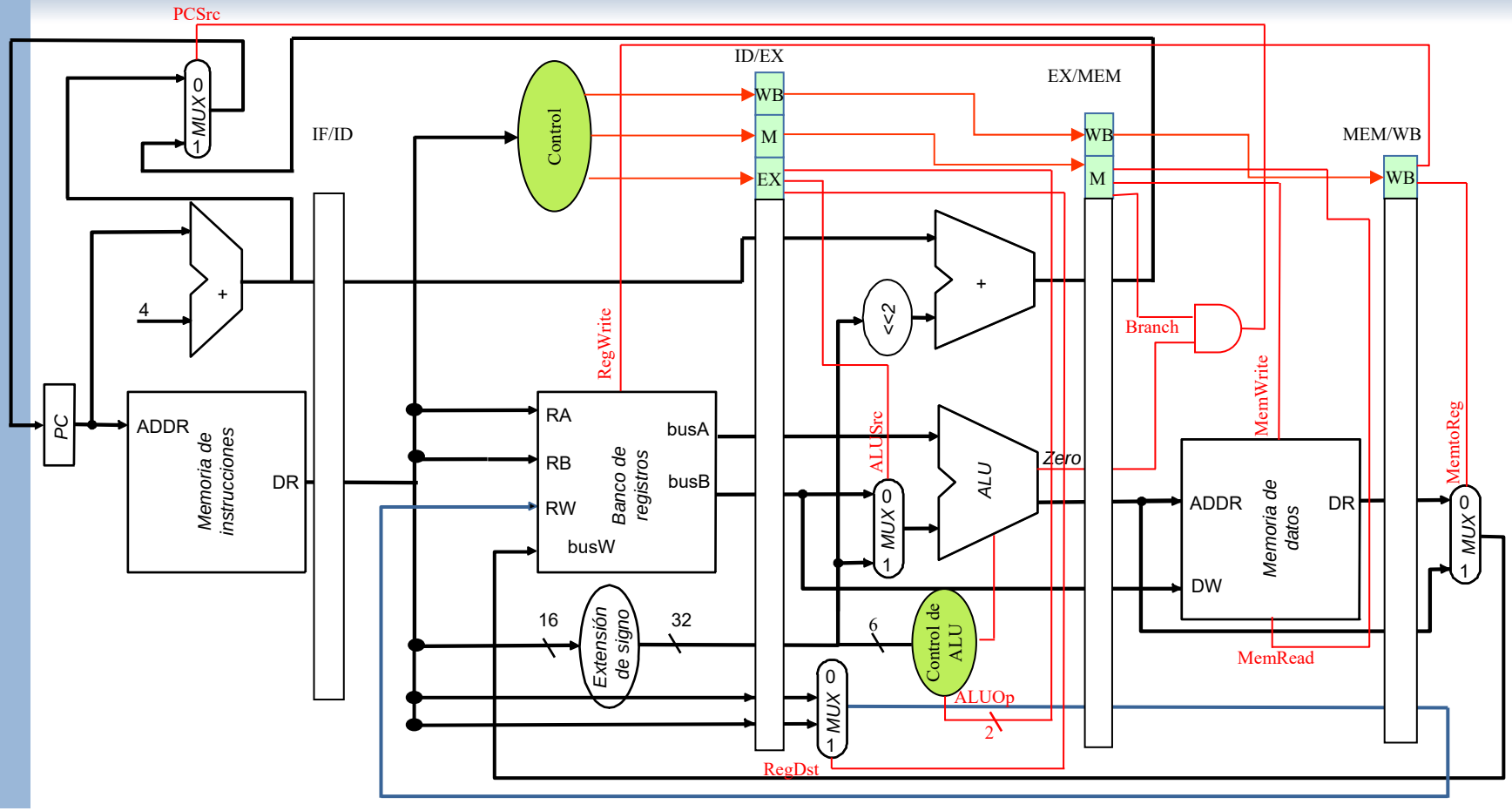
El control de la ALU se determina por ALUop (que depende del tipo de instrucción) y el campo de función en las instrucciones de tipo-R

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



SEGMENTACIÓN: CONTROL



ÍNDICE

1. MIPS
2. Segmentación
3. Riesgos estructurales
4. **Riesgos de datos**
5. Riesgos de control
6. Riesgos de control con riesgos LDE
7. Operaciones multiciclo
8. Excepciones
9. Rendimiento en los procesadores

SEGMENTACIÓN : RIESGOS DE DATOS

- **Riesgos de datos:**
  - Se produce un riesgo si existe dependencia entre instrucciones que se ejecutan concurrentemente
  - Este tipo de riesgos aumentan en operaciones multiciclo

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

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**,R5,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)

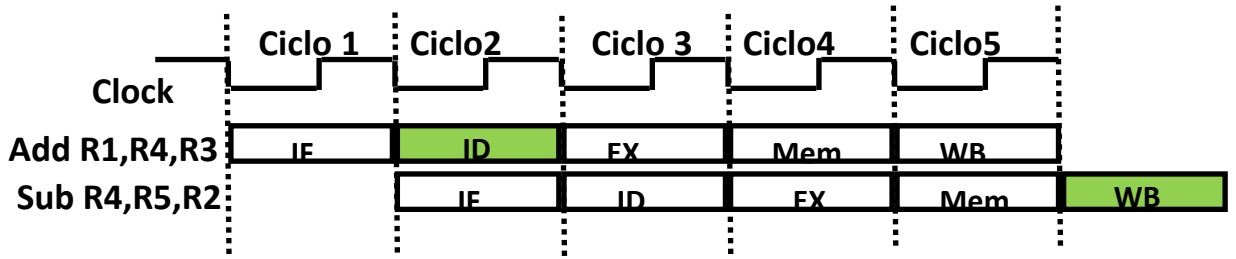
  *ADD **R4**,R2,R3 – escribe el registro R4*

  *Add **R4**,R1,R2—escribe el registro R4*

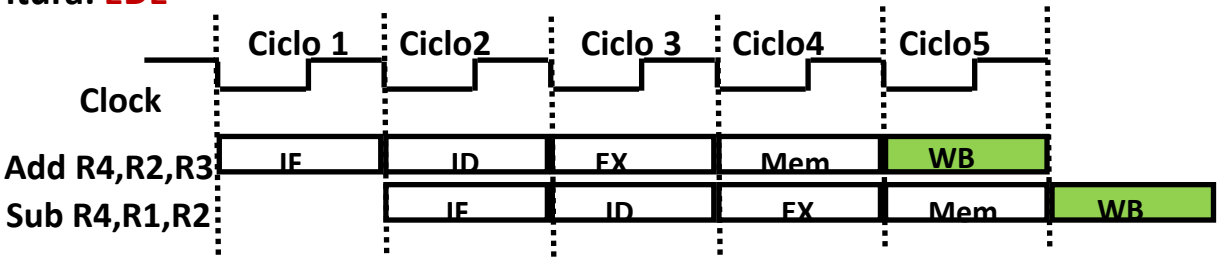
  - Se produce riesgo si r4 de la segunda instrucción se escribe antes de que lo escriba la primera instrucción

RIESGOS DE DATOS: EDL Y EDE

- **Escritura después de lectura: EDL**

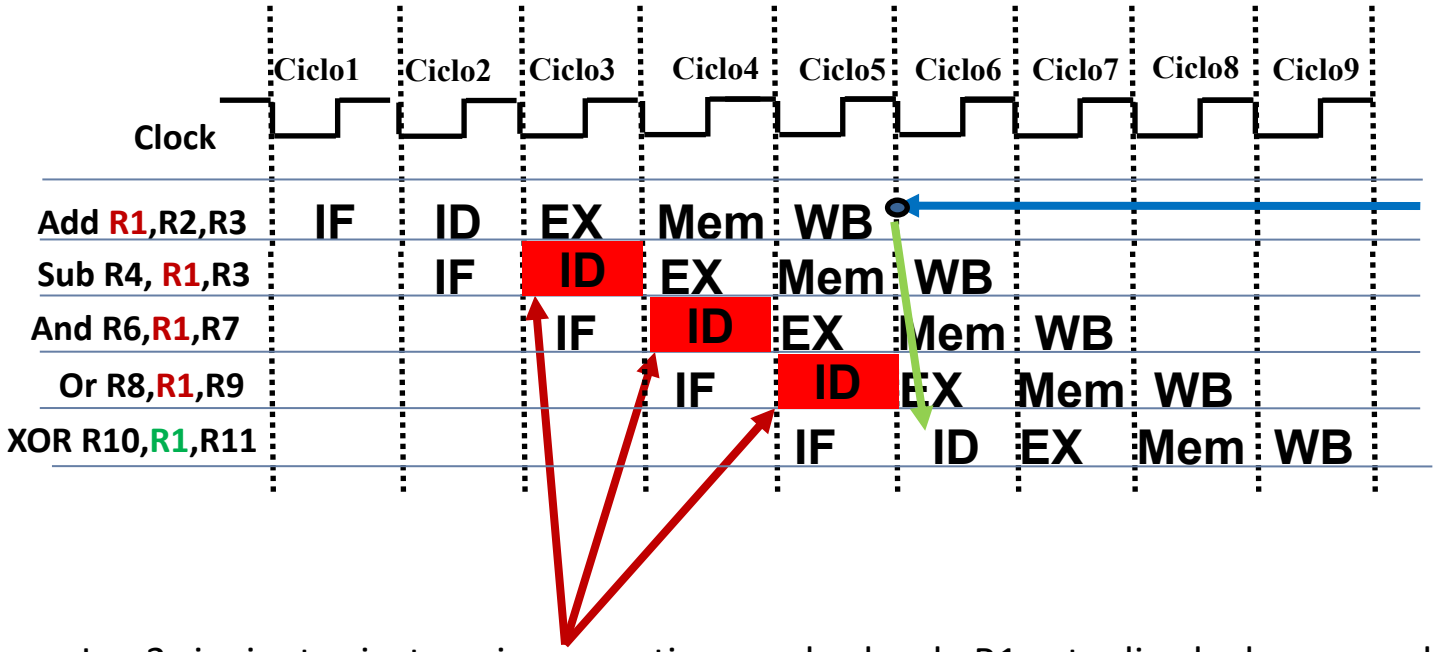


- **Escritura después de escritura: EDE**



- **Estos riesgos NO se dan en el pipeline con todas la instrucciones de igual duración**
  - Se leen los registros en el final de la segunda etapa
  - Las instrucciones escriben en el BR en la última etapa
  - Todas las instrucciones tienen igual duración

RIESGOS DE DATOS: LDE



R1 se escribe al empezar el ciclo6

Las 3 siguientes instrucciones no tienen el valor de R1 actualizado, leen un valor antiguo

RIESGOS DE DATOS: LDE

¿Cómo resolver los riesgos LDE?

Solución 1: Detener el pipeline

- ¿En qué etapa se realiza la parada?
  - Depende del diseño de la Ruta de datos
  - Nosotros vamos a suponer que **las paradas se realizan en decodificación**
  - Si se realiza en otra etapa lo especificará el enunciado del problema
- ¿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

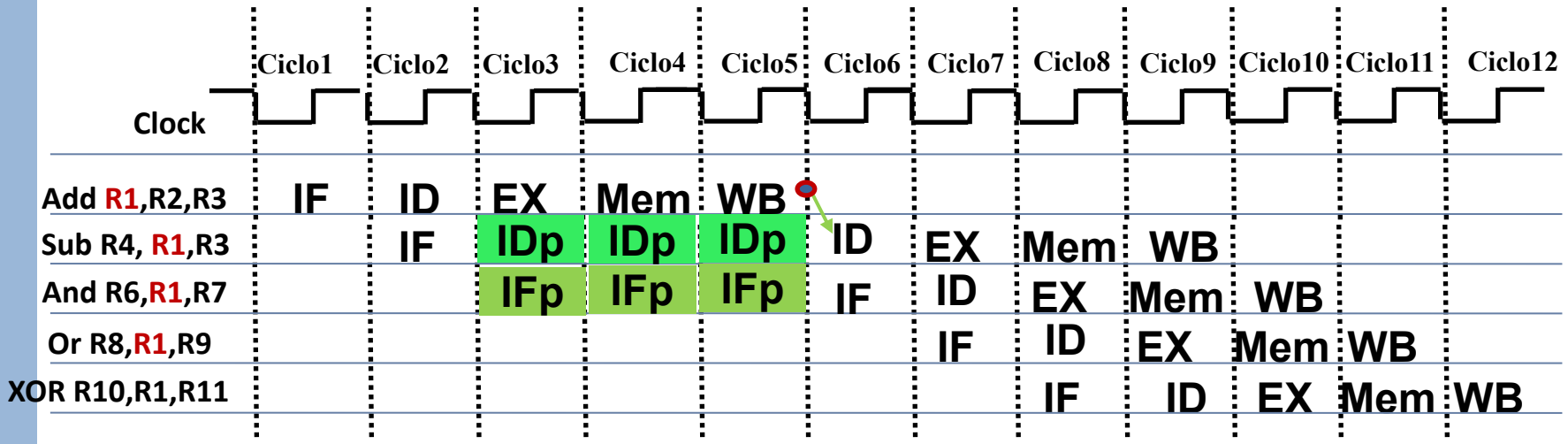
Solución 2: Reordenar código

- Si se puede, no siempre es posible
- Puede minimizar las paradas

RIESGOS DE DATOS: LDE

¿Cómo resolver los riesgos LDE?

- **Solución 1: Detener el pipeline** (en la etapa de decodificación)



3 ciclos de espera

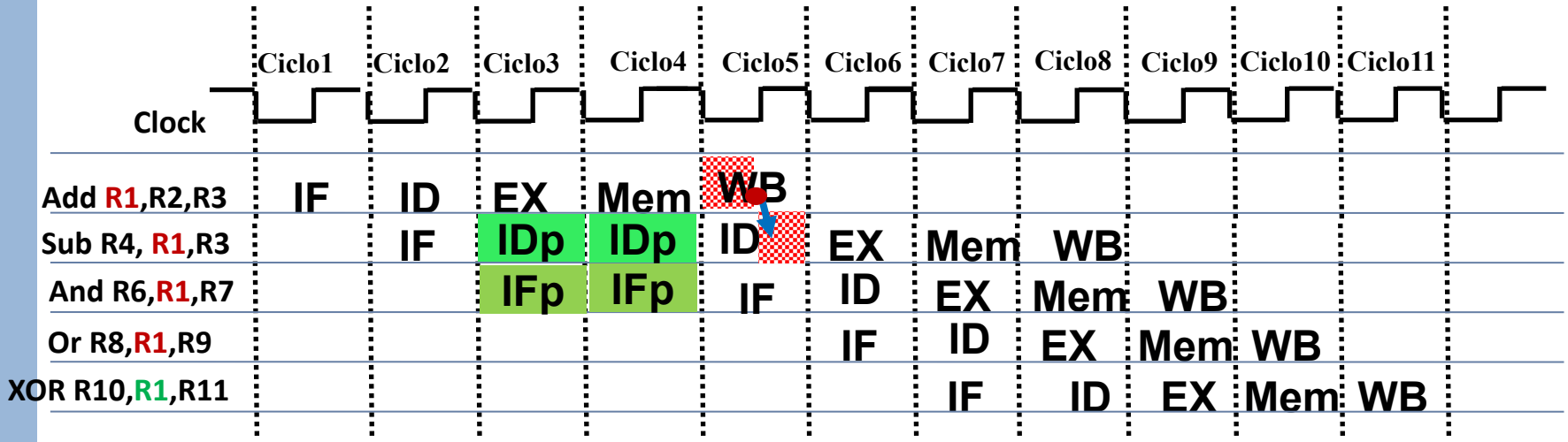
Sólo se está ejecutando la primera instrucción

IDp ✓ID_P **parada por riesgo LDE** entre instrucción 1 e instrucción 2

IFp ✓IF_P **parada por riesgo estructural**, la etapa está ocupada por la instrucción anterior

¿Cómo resolver los riesgos LDE?

- **Mejora**: el **Banco de Registros escribe en la primera mitad del ciclo y lee en la segunda mitad del ciclo**



2 ciclos de espera

Sólo se está ejecutando la primera instrucción

IDp ID_P **parada por riesgo LDE** entre instrucción 1 e instrucción 2

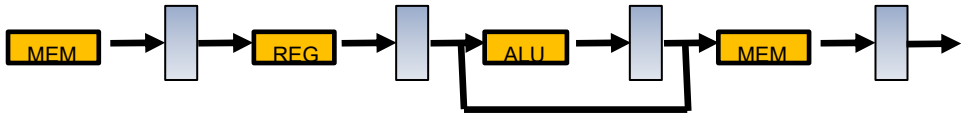
IFp IF_P **parada por riesgo estructural**, la etapa está ocupada por la instrucción anterior

RIESGOS DE DATOS: LDE

¿Cómo resolver los riesgos LDE? Solución 3: Cortocircuito (forwarding)

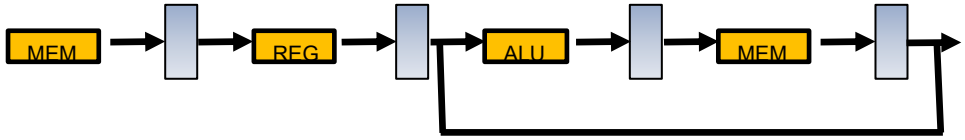
- *¿ Cuándo está listo el operando ?* Enviar el dato cuando esté calculado a las etapas que lo necesiten sin esperar a WB
  - Situación:

```
add r1,r2,r3
sub r4,r1,r3
and r6,r5,r7
```



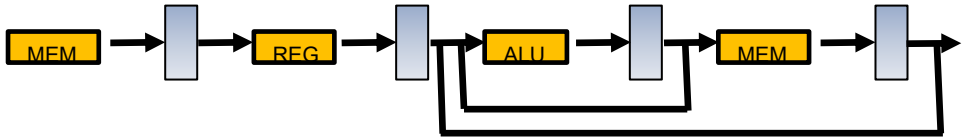
  - Situación:

```
add r1,r2,r3
sub r4,r5,r3
and r6,r1,r7
```



  - Situación:

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

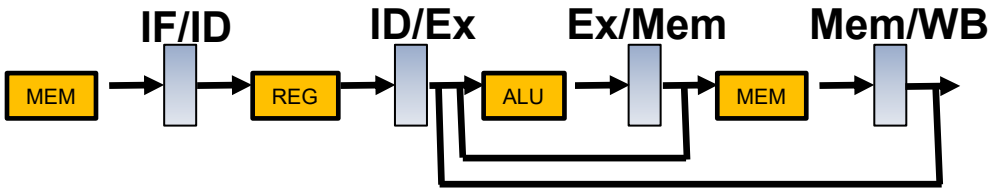


- **Para implementar el cortocircuito necesitamos dos caminos de datos:**
  - Desde el registro de pipeline EX/MEM (salida de la ALU) a entrada ALU
  - Desde el registro de pipeline MEM/WB (salida de la memoria) a entrada ALU

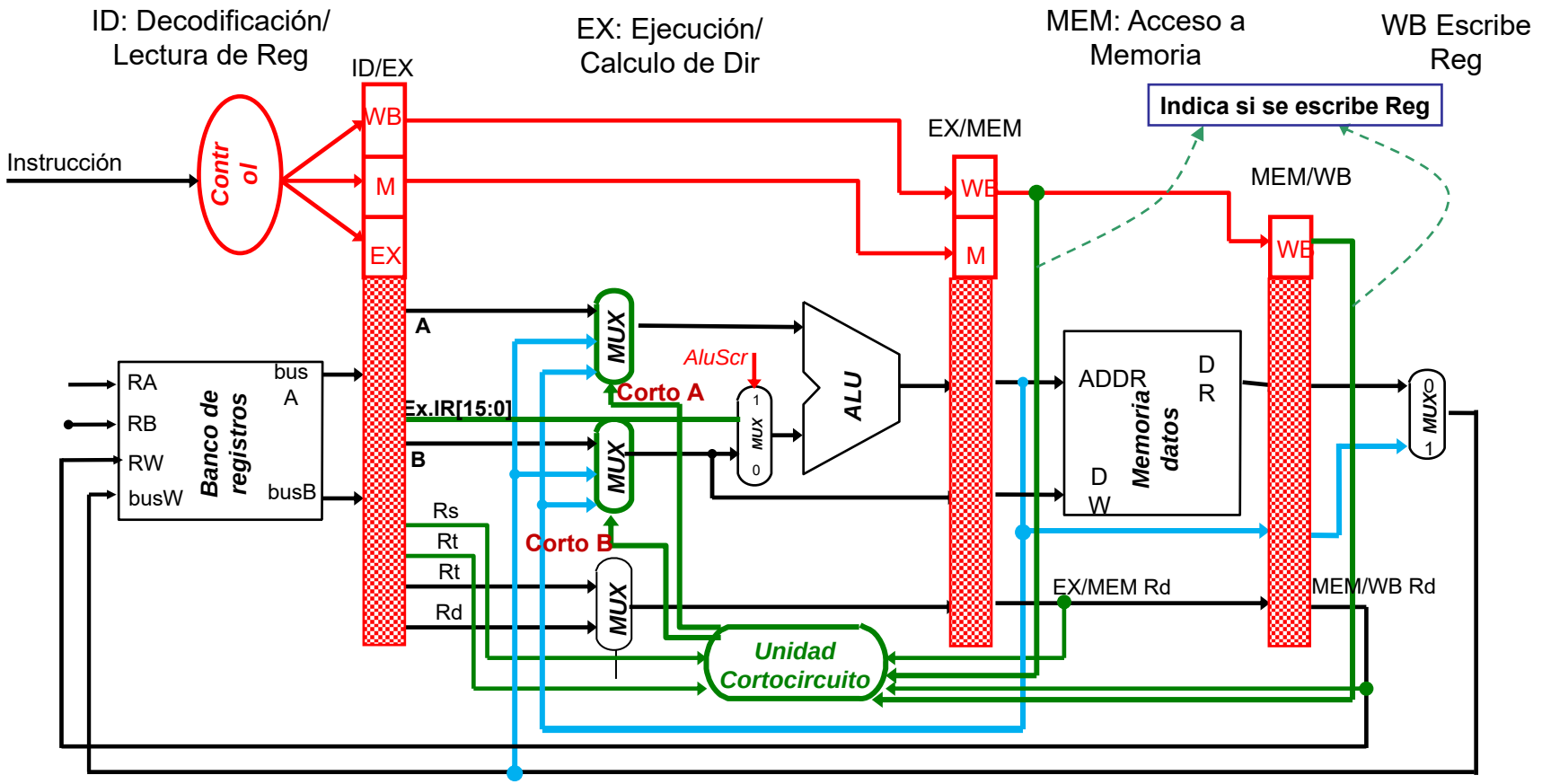
RIESGOS DE DATOS: LDE

¿Cómo resolver los riesgos LDE? Solución 3: Cortocircuito (forwarding)

- **Información necesaria para implementar el cortocircuito:**
  - 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

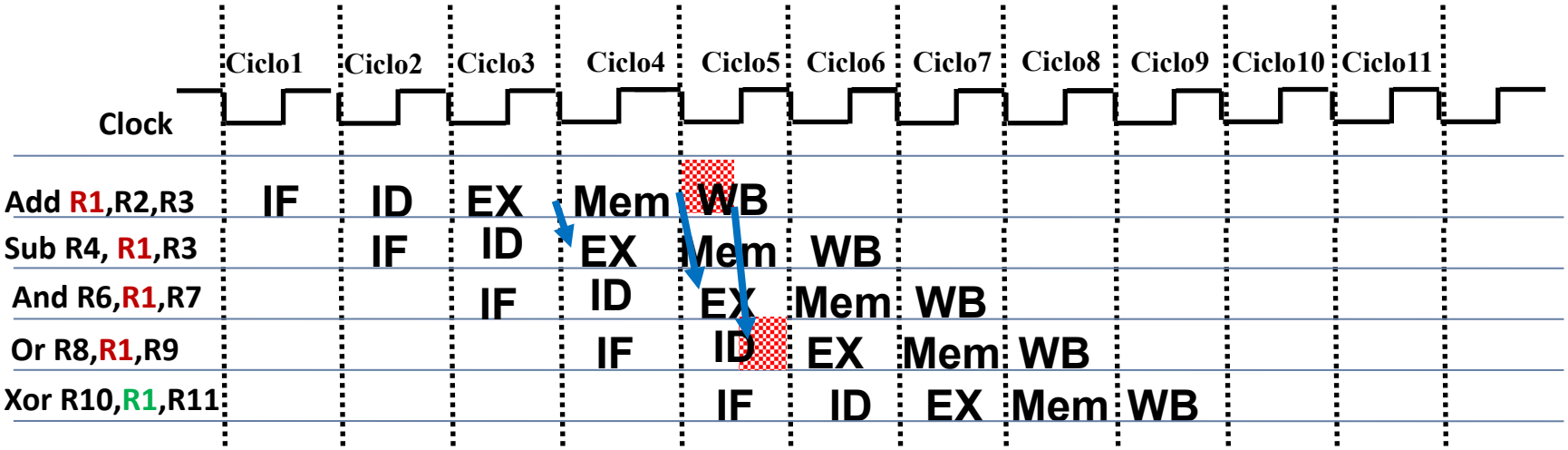


Riesgos LDE: Implementación del cortocircuito



¿Cómo resolver los riesgos LDE?

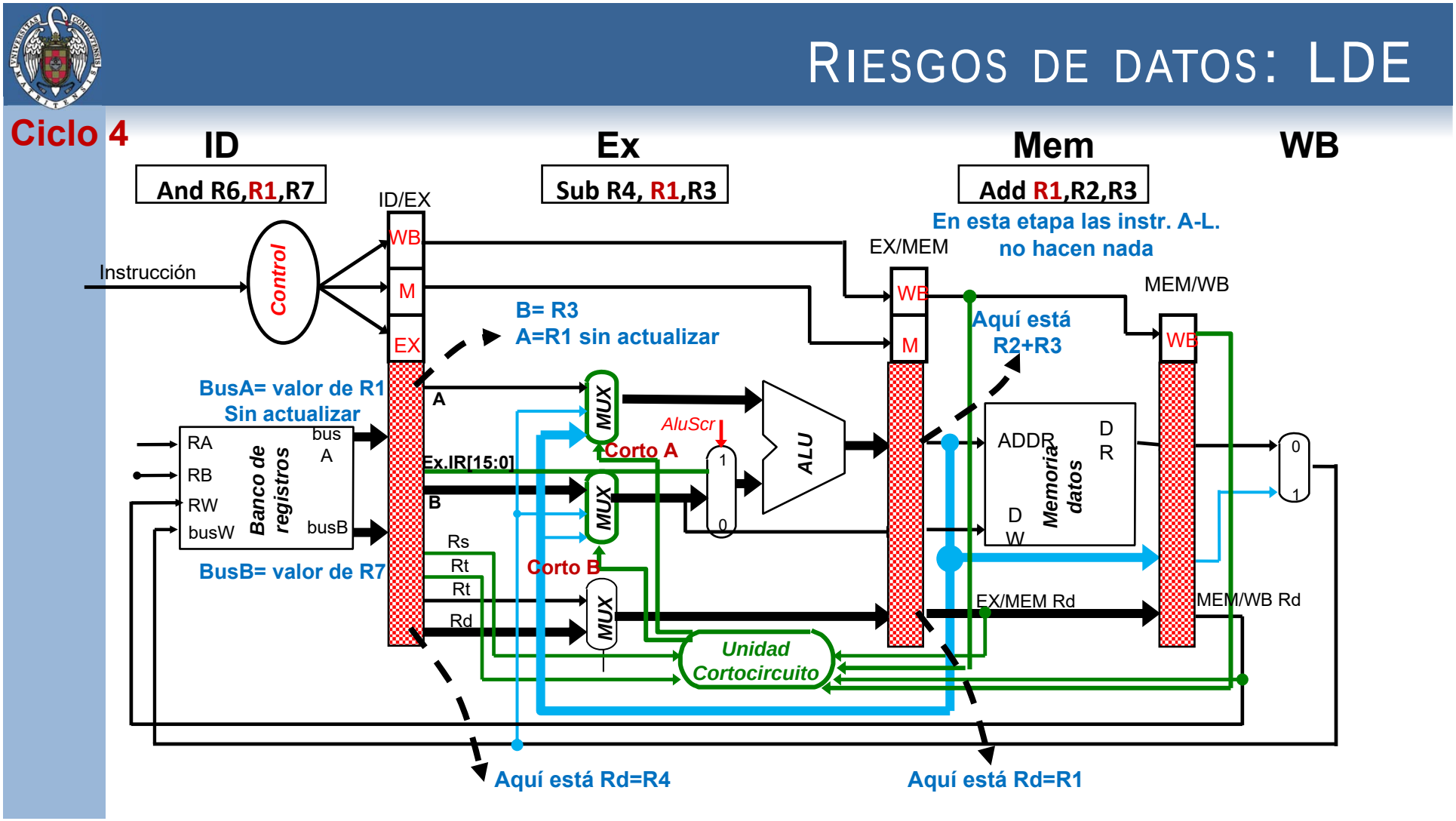
- **Solución 3:** Cortocircuito (forwarding)



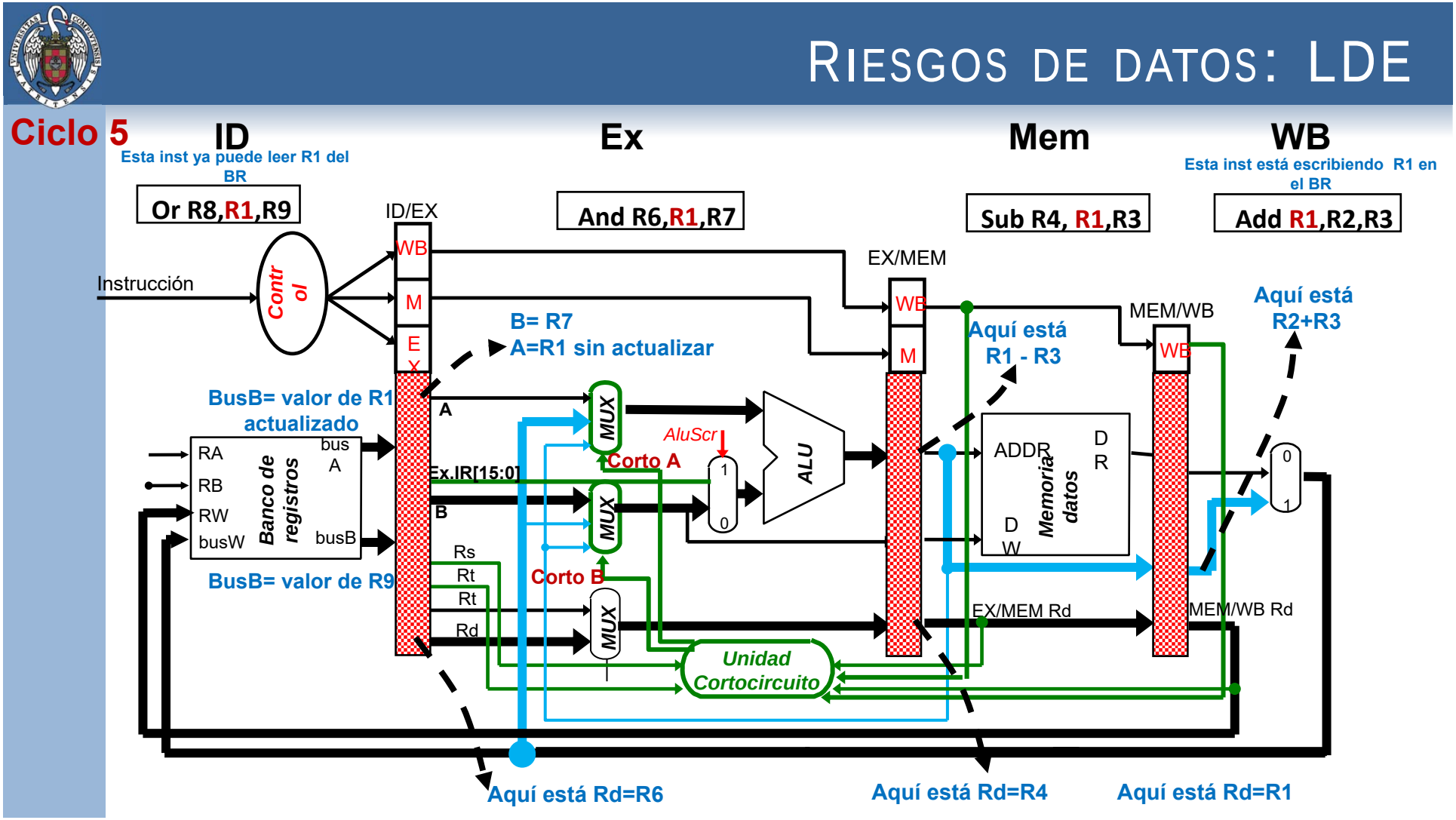
No hay ciclos de espera

Se pueden ejecutar todas las instrucciones sin problemas

RIESGOS DE DATOS: LDE

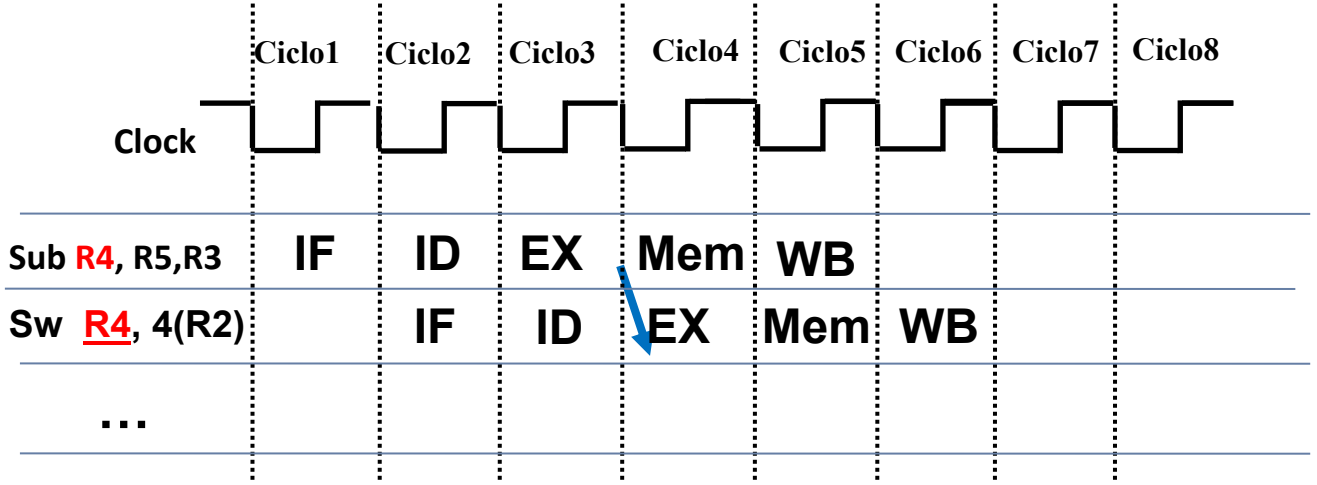


RIESGOS DE DATOS: LDE



Riesgo LDE: ejemplo 1 con instrucción store

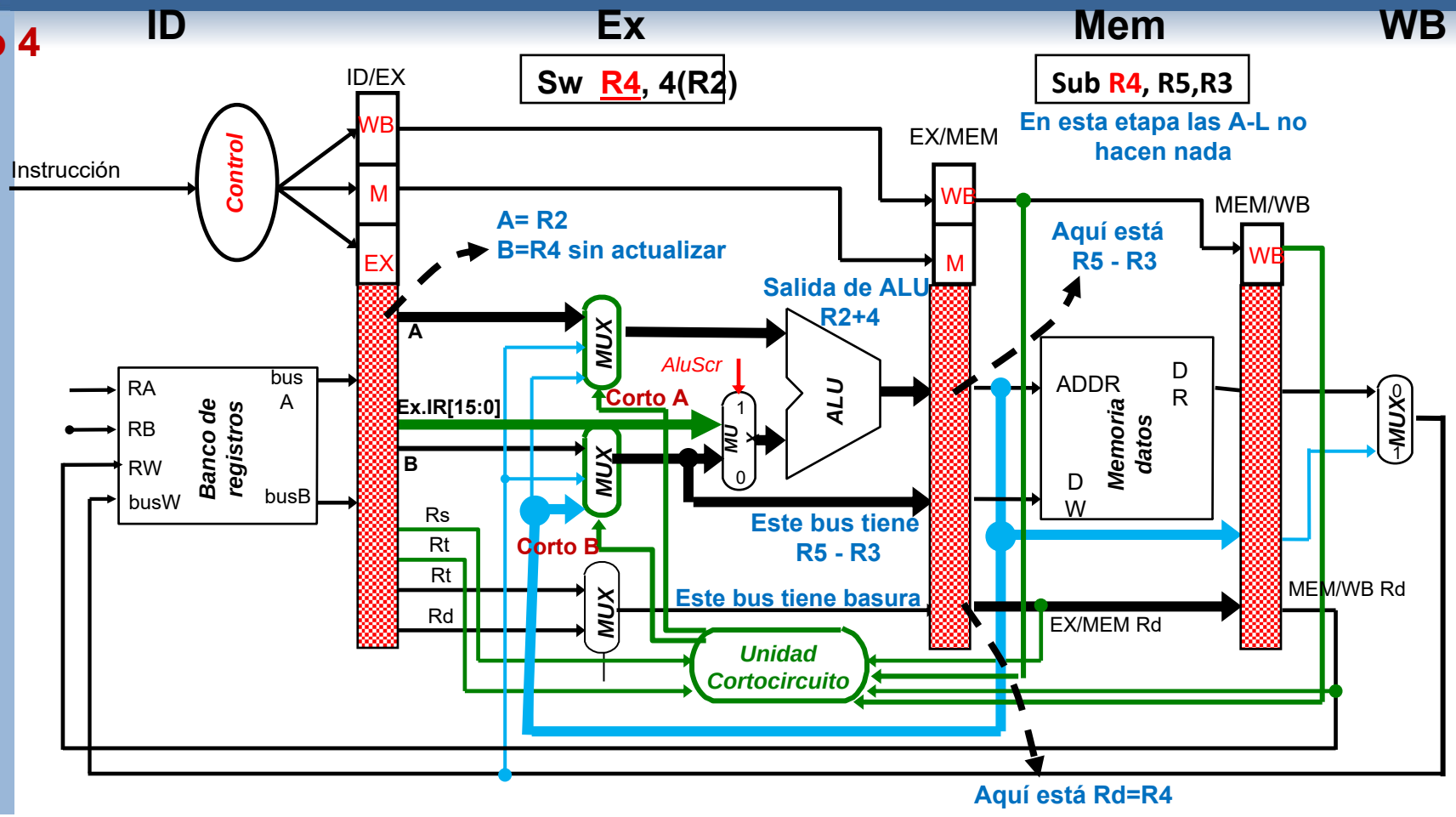
- El store escribe en memoria en la etapa Mem pero en nuestra ruta de datos **el dato que va a escribir** en memoria lo **tiene que tener en la etapa Ex**
  - Se puede resolver con cortocircuito



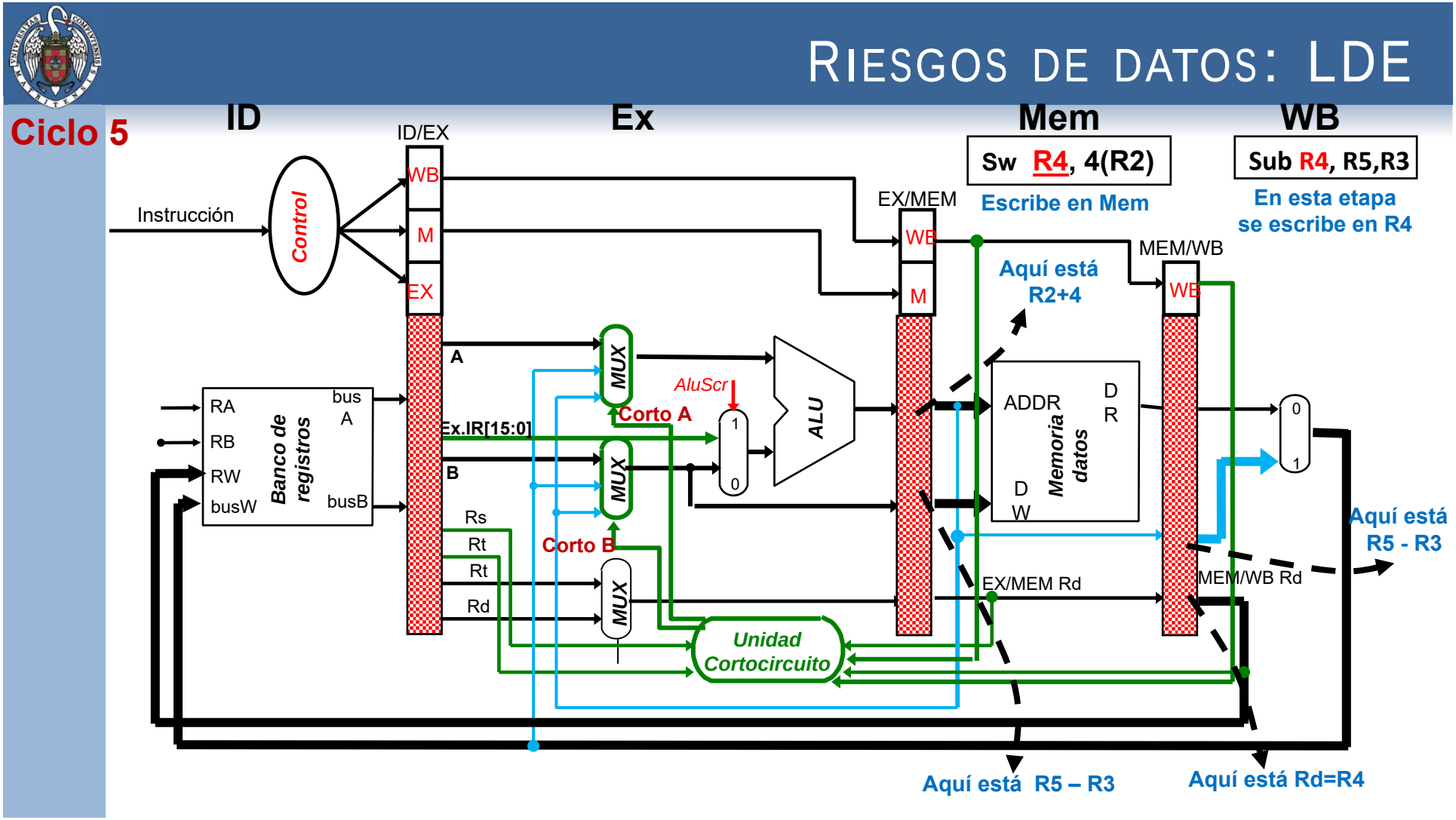
No hay ciclos de espera

RIESGOS DE DATOS: LDE

Ciclo 4

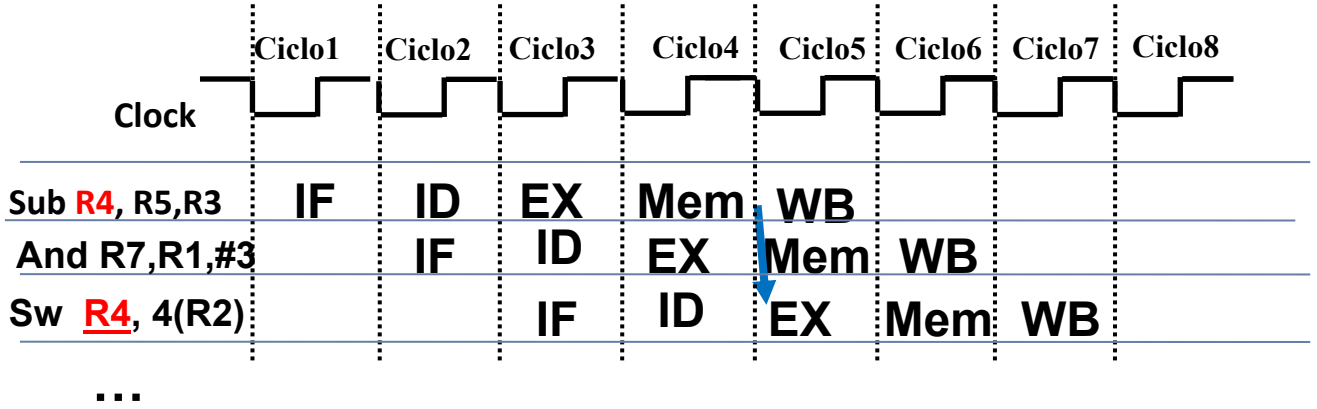


RIESGOS DE DATOS: LDE



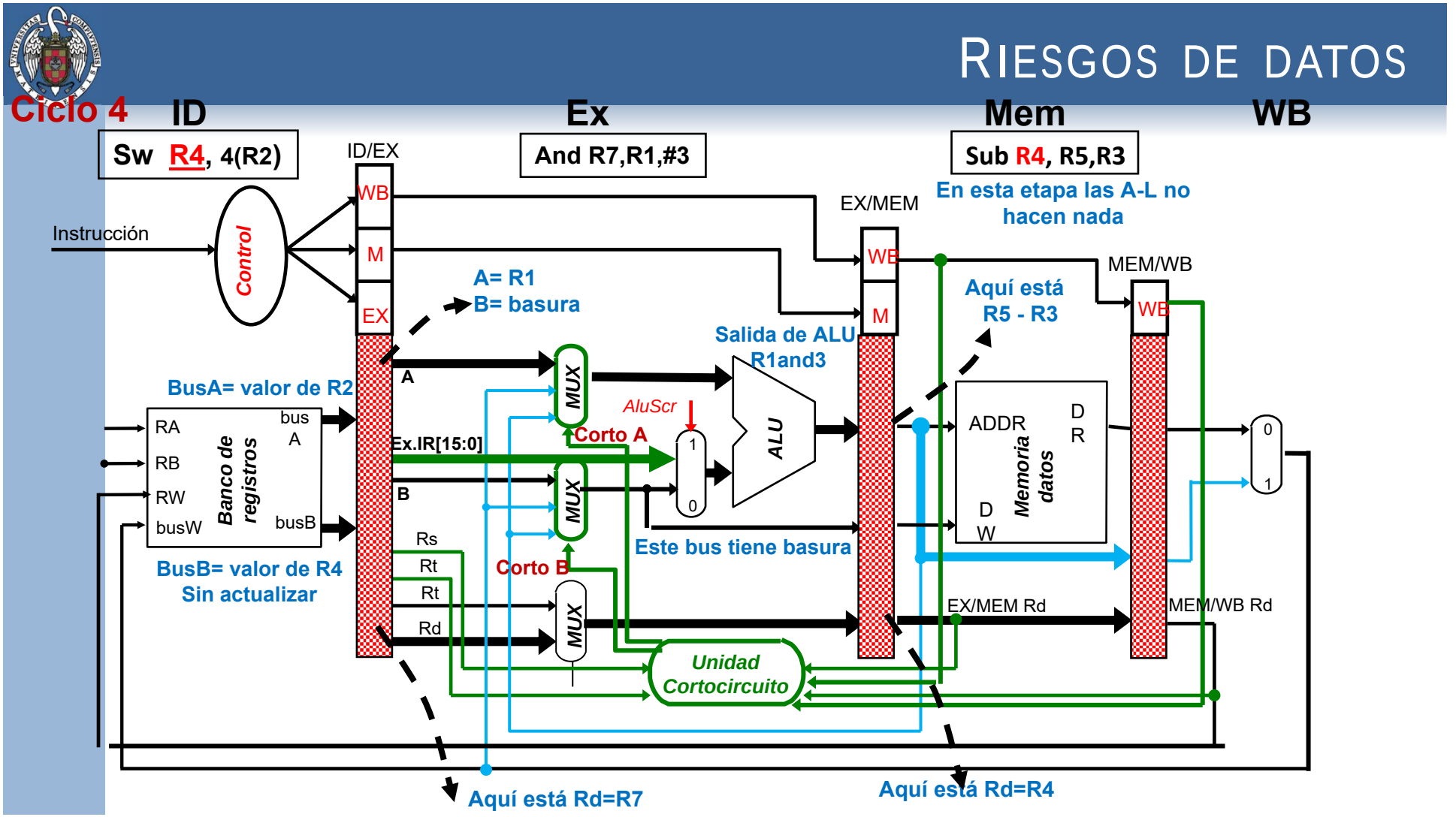
Riesgo LDE: ejemplo 2 con instrucción store

- El store escribe en memoria en la etapa Mem pero en nuestra ruta de datos **el dato que va a escribir** en memoria lo **tiene que tener en la etapa Ex**
  - **Se puede resolver con cortocircuito**

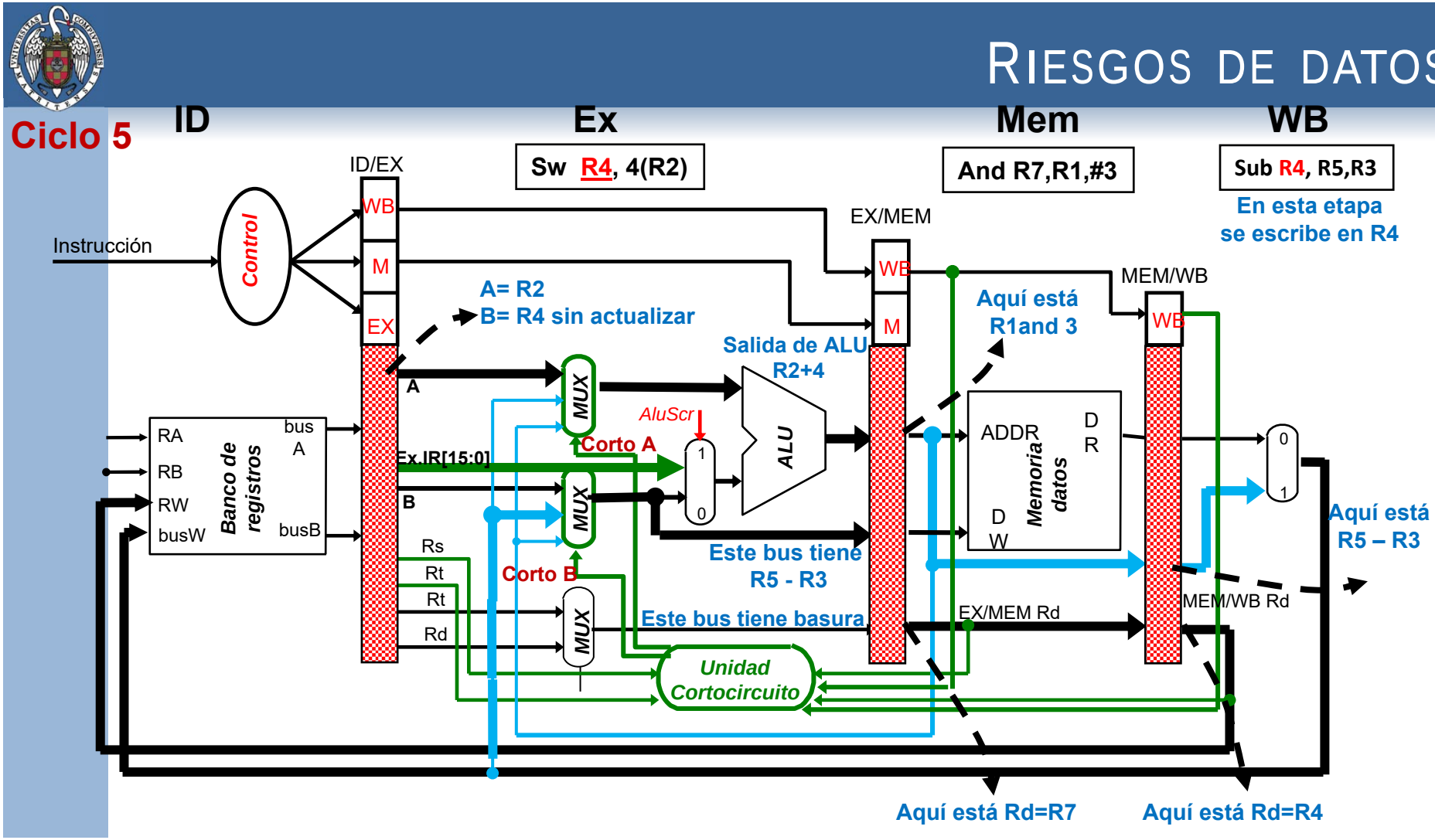


No hay ciclos de espera

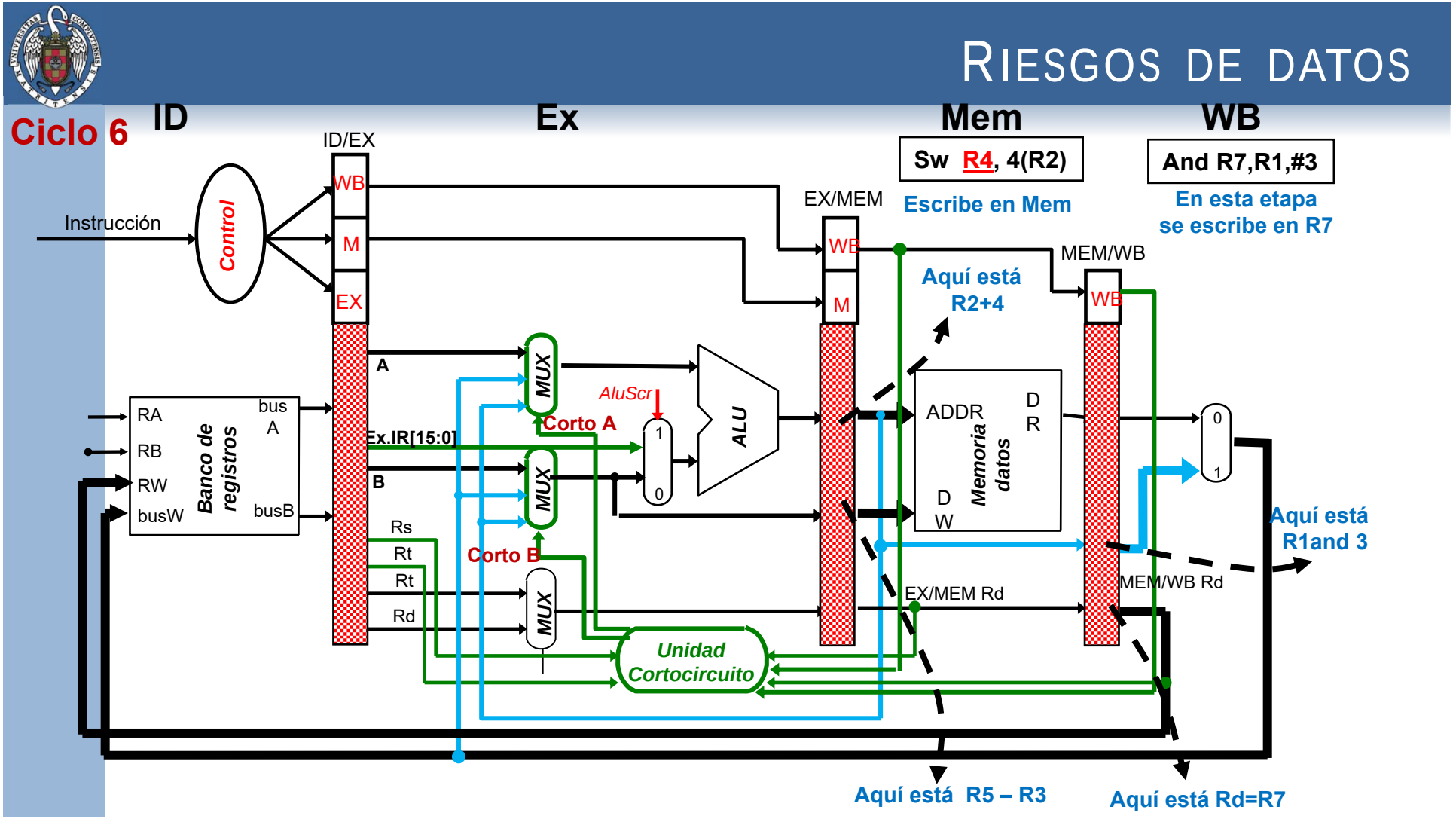
RIESGOS DE DATOS



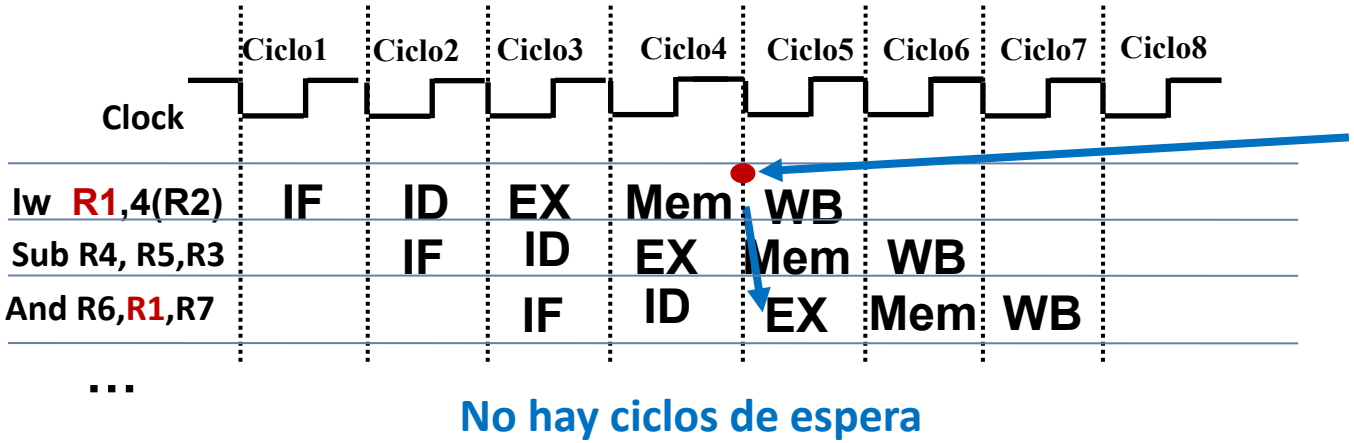
RIESGOS DE DATOS



RIESGOS DE DATOS



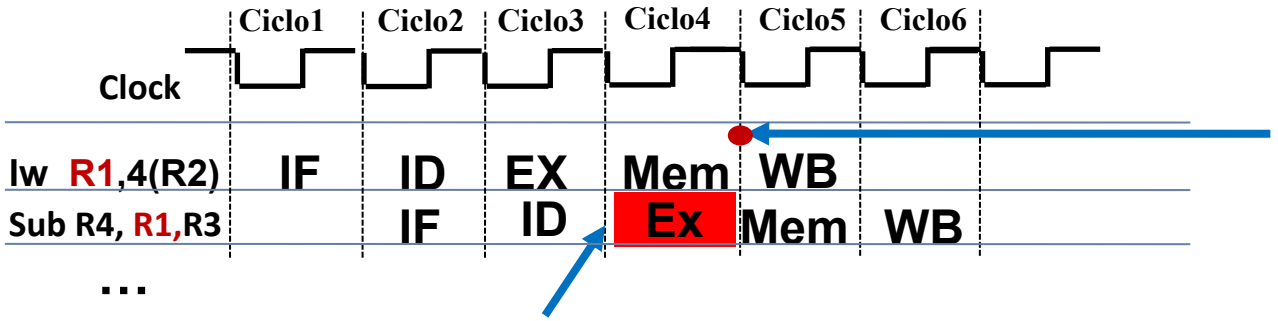
- **Riesgo LDE:** caso particular, **el dato lo proporciona un Load**



No hay ciclos de espera

- Se puede resolver sólo con cortocircuito si hay una instrucción intermedia

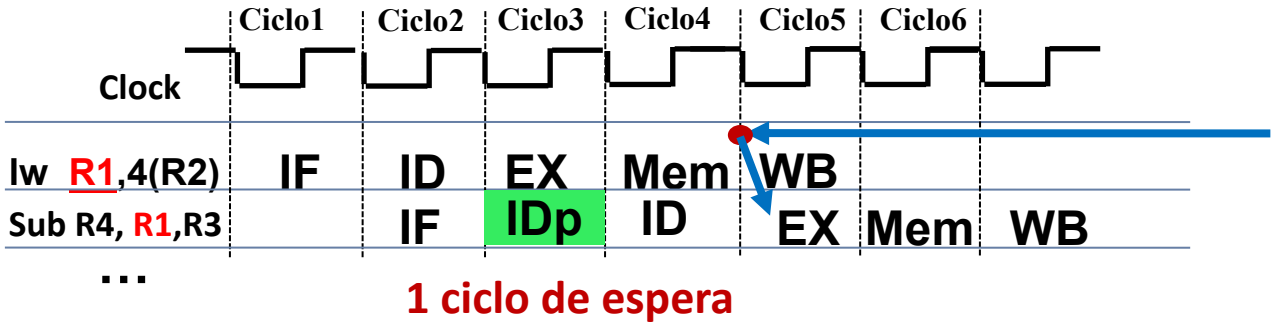
- **Riesgo LDE:** caso particular, **el dato lo proporciona un Load**



- No se puede resolver sólo con el cortocircuito

Riesgo LDE: caso particular, **el dato lo proporciona un Load**

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

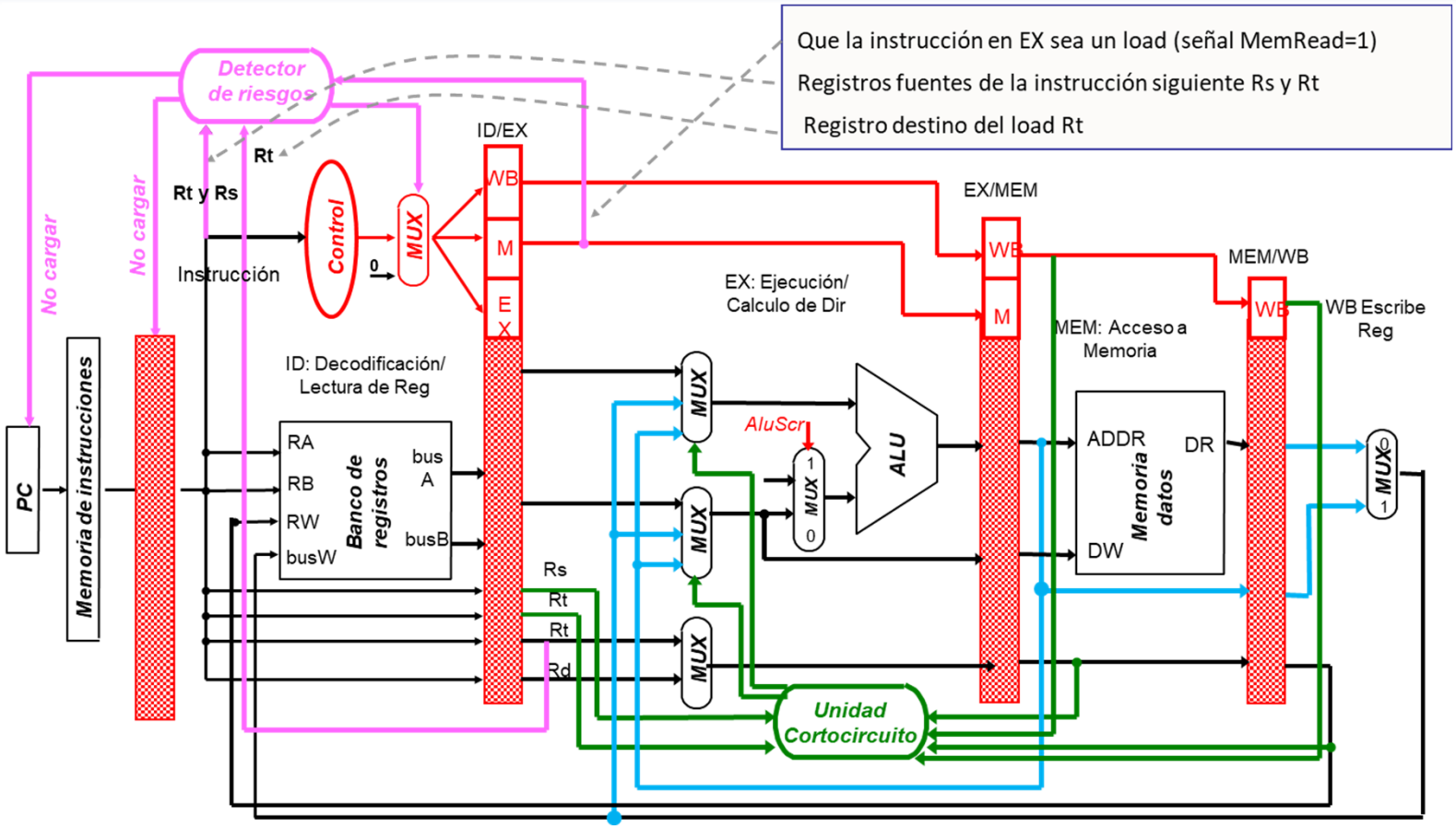


El valor que se quiere guardar en R1 está disponible

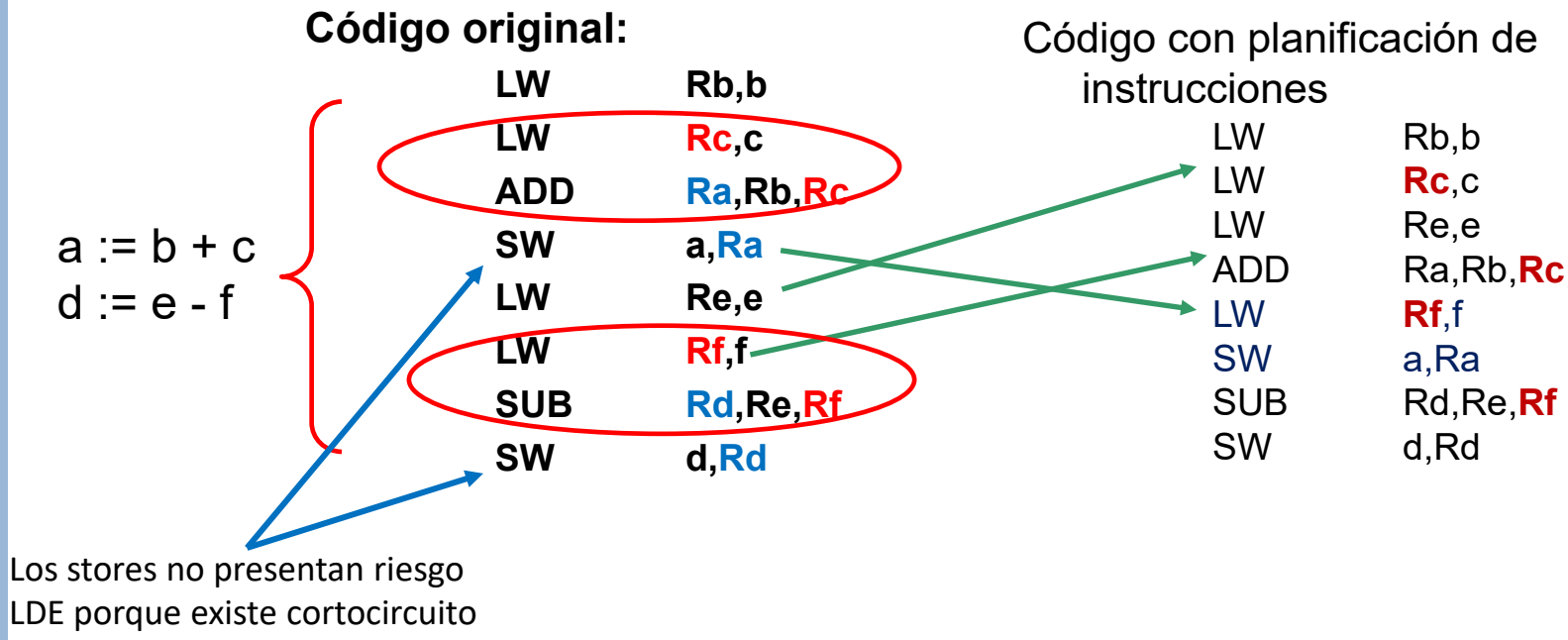
IDp ✓ID_P **parada por riesgo LDE** entre instrucción 1 e instrucción 2

RIESGOS DE DATOS

- Detección de riesgos y parada del procesador un ciclo



- **Riesgo LDE:** caso particular, **el dato lo proporciona un Load**
  - **Solución SW**: Anticipar el Load en la planificación de instrucciones que hace el compilador

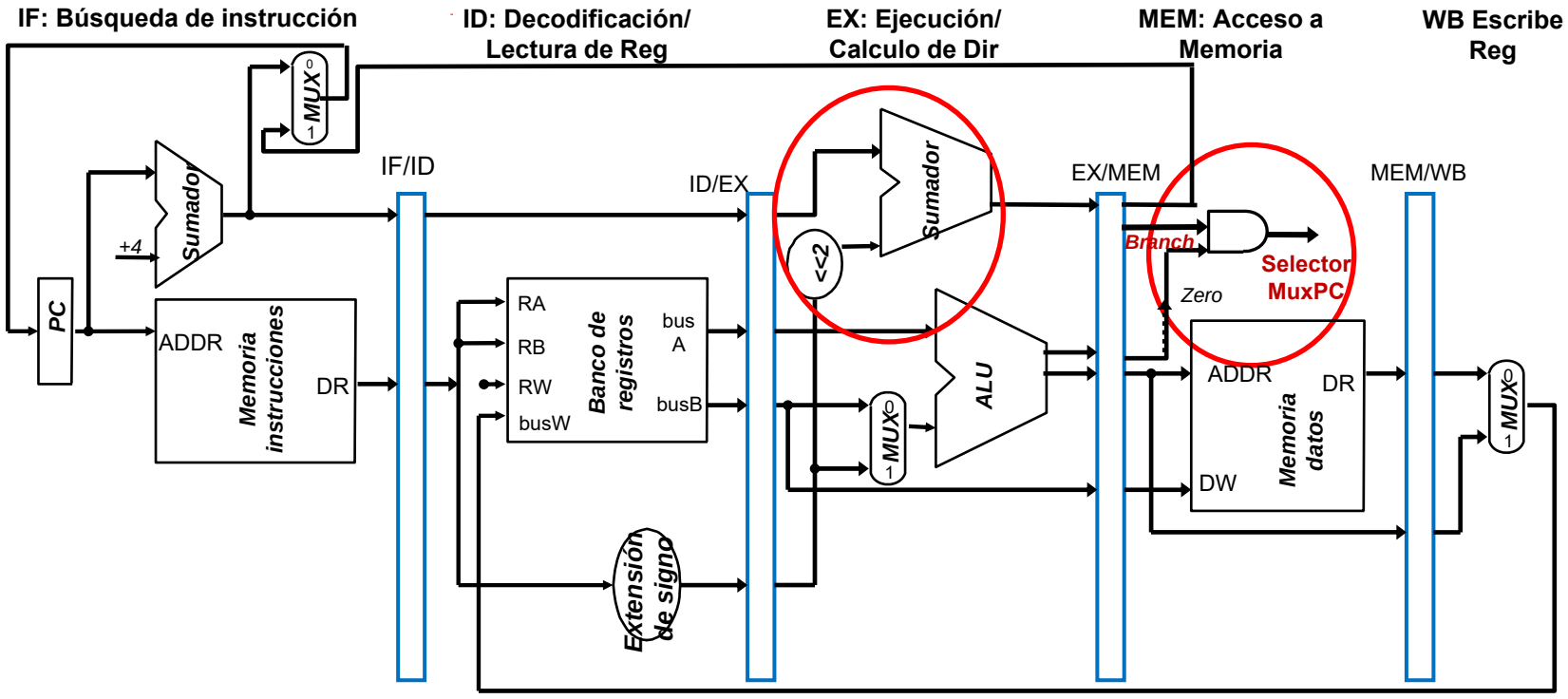


ÍNDICE

RIESGOS DE CONTROL

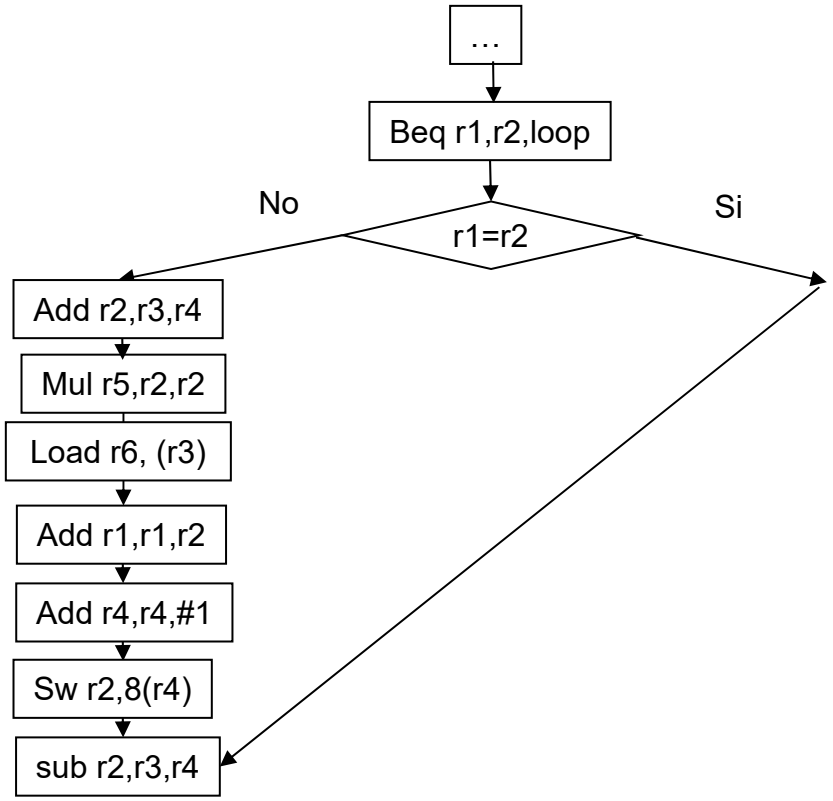
- Riesgos de control: ¿Por qué aparecen?
  - Para saltar se necesita**:**
    - Haber calculado la **dirección de salto**
    - Saber **si se cumple la condición** de salto
  - Aparece el riesgo porque
    - En la ruta de datos que hemos estudiado **ambos datos se calculan en Ex**
    - **Si el salto se toma, la dirección del salto se carga en PC al final de Mem,** cuando llega el flanco de reloj

RIESGOS DE CONTROL



- **Ejemplo:**

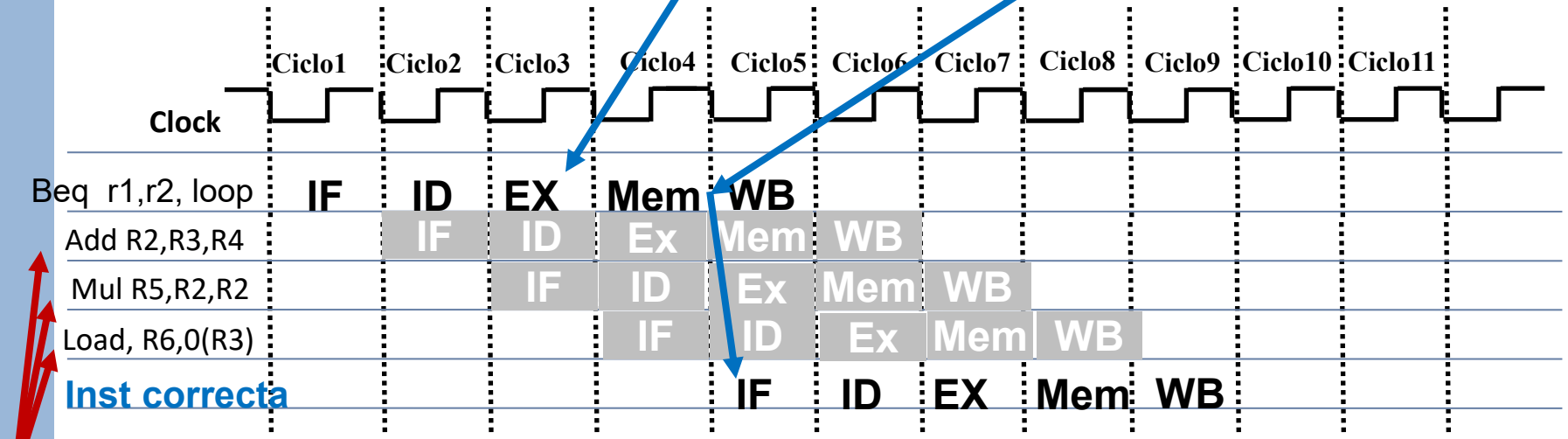
```
        …
        Beq r1,r2,loop
        Add r2,r3,r4
        Mul r5,r2,r2
        load r6, (r3)
        Add r1,r1,r2
        Add r4,r4,#1
        Sw   r2,8(r4)
Loop    sub r2,r3,r4
```



RIESGOS DE CONTROL

El cálculo 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 entrar en WB**



Todavía no sabe si salta o no en el pipeline entra la siguiente instrucción

Al empezar el ciclo 5 es cuando entra en el pipeline la instrucción correcta

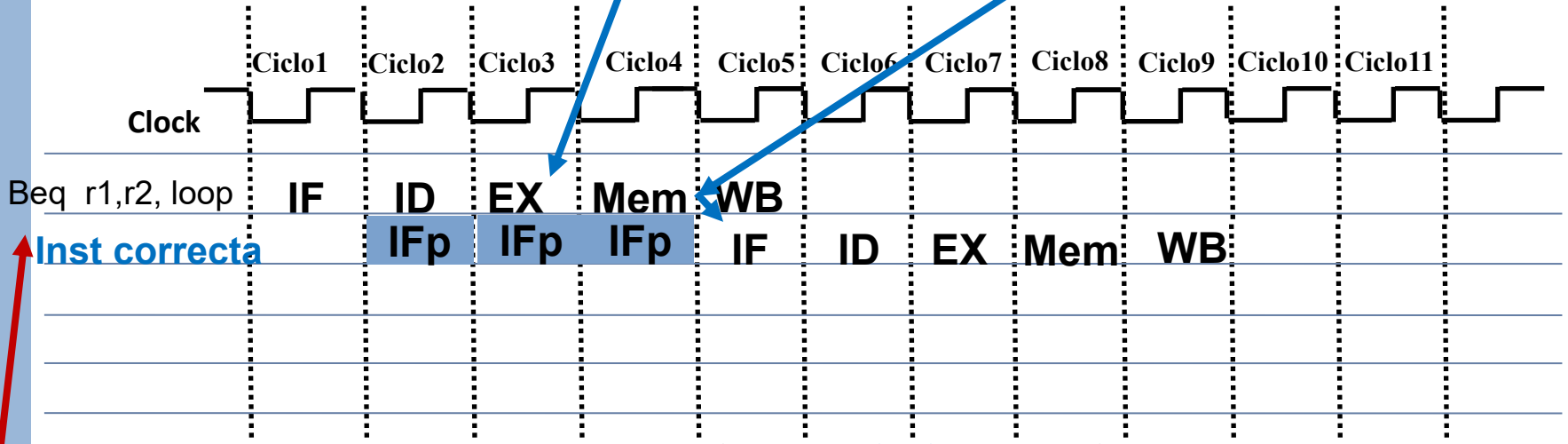
Se han ejecutado instrucciones que no debían ejecutarse si el salto se toma

RIESGOS DE CONTROL: SOLUCIÓN

- **Solución 1**: Hay que **esperar tres ciclos** para saber cuál es la instrucción siguiente al salto: **Parando el pipeline**

El cálculo 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 entrar en WB**



Hasta que no se sepa si salta o no el pipeline está parado

Al empezar el ciclo 5 es cuando entra en el pipeline la instrucción correcta

3 ciclos de penalización

RIESGOS DE CONTROL: SOLUCIÓN

- **Solución 2**:
  - Desplazar el **cálculo de la dirección** y la evaluación de **la condición** a la etapa **ID**
  - Sólo hay que **esperar un ciclo** para saber la instrucción que sigue a la de salto. Este ciclo de espera se puede implementar:
    - HW: Se introduce la siguiente instrucción
      - Si el salto se realiza: se elimina la instrucción introducida
        - Se ponen "0" en las etapas del pipeline para que no se realice ninguna acción
        - *1 ciclo de penalización*
      - Si el salto NO se realiza: se introduce en el pipeline la instrucción correcta
        - *Ningún ciclo de penalización*

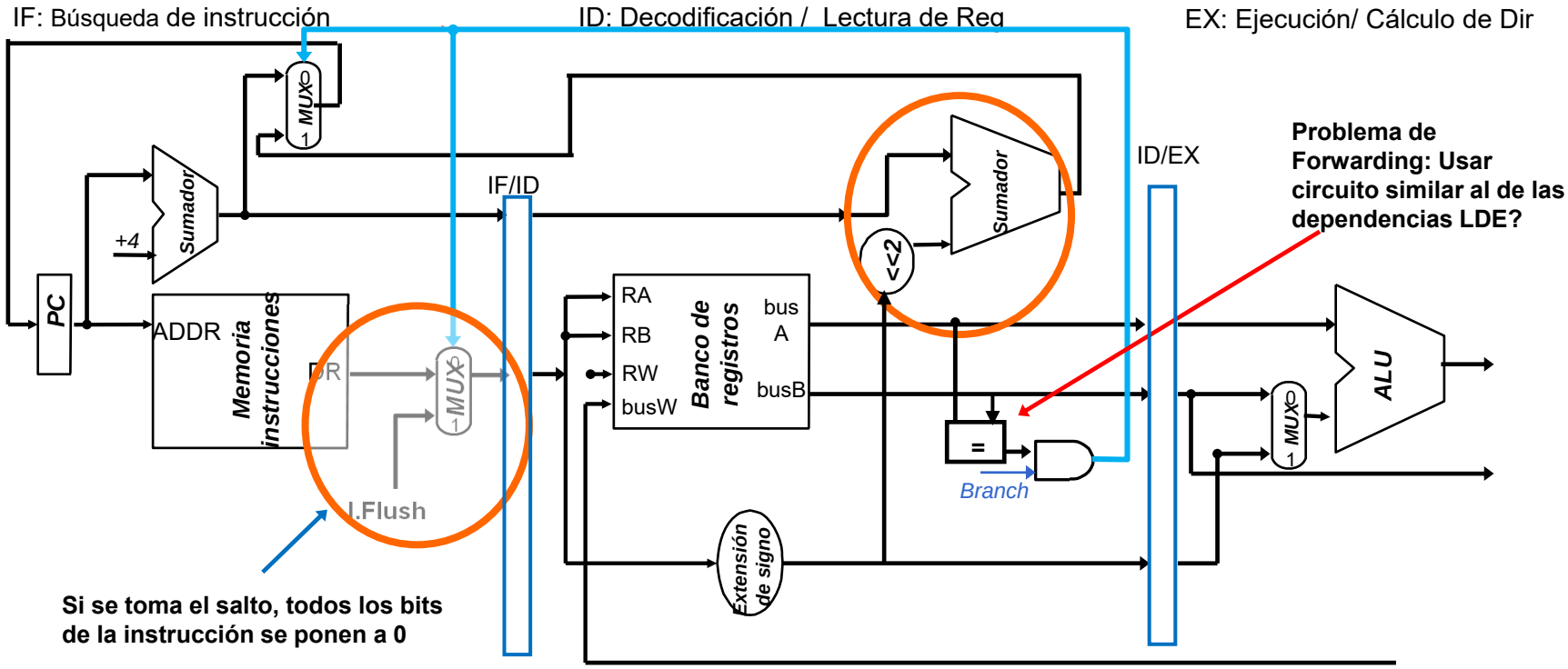
RIESGOS DE CONTROL: SOLUCIÓN

- **Solución 2**:
  - Este ciclo de espera se puede implementar:
    - **SW: Saltos retardados**
      - Ejecutar instrucciones ***independientes del salto*** durante el ciclo de retardo
        - **Se ejecutarán siempre**, tanto si el salto se realiza como si no
      - ***Si es posible encontrar instrucciones que se tengan que ejecutar siempre***
        - ***Ningún ciclo de penalización***

RIESGOS DE CONTROL: SOLUCIÓN

Ruta de datos con la implementación HW de la solución 2 (sin saltos retardados):

- Calcular la **dirección de salto** y la evaluación de la **condición** en la **etapa ID**
- Eliminar la instrucción siguiente si el salto no se realiza

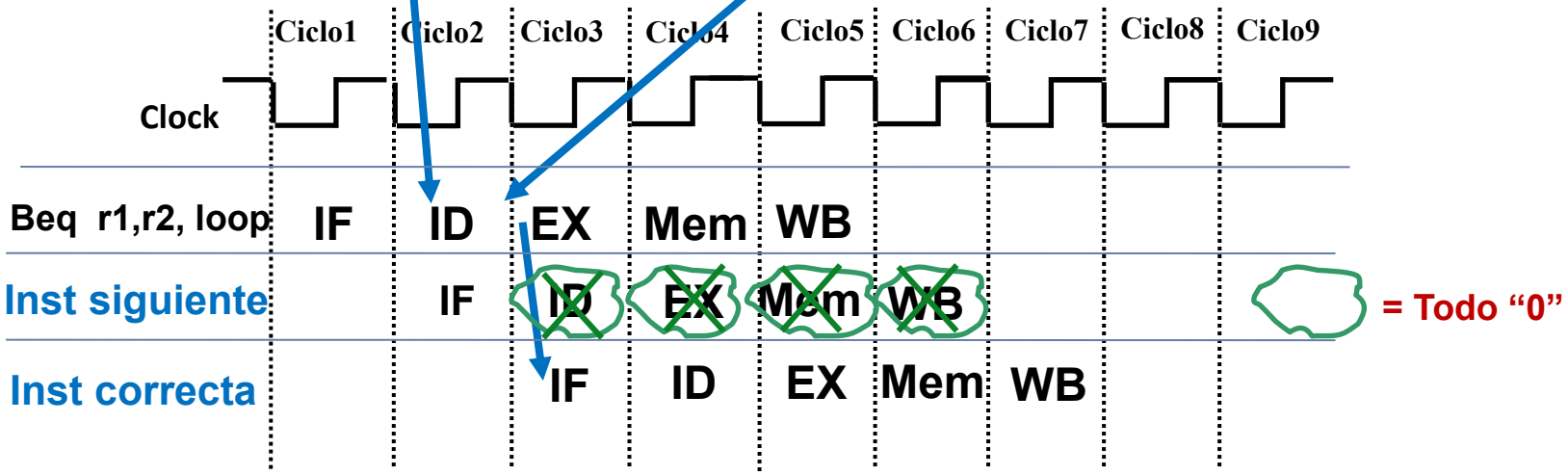


RIESGOS DE CONTROL: SOLUCIÓN

- Ejemplo: Aplicando la Solución 2 con implementación HW
  - Si el salto se realiza

El cálculo de la **dirección de salto** y la evaluación de **la condición** se realizan en la **etapa ID**

La nueva dirección se carga en PC cuando llega el flanco de reloj **al entrar en Ex**



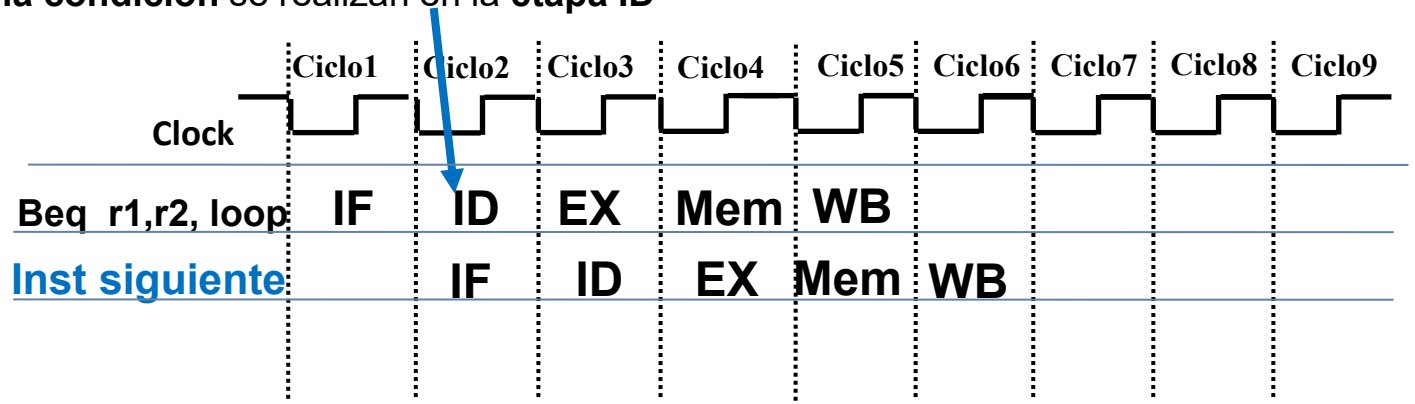
Al empezar el ciclo 3 es cuando entra en el pipeline la instrucción correcta

1 ciclos de penalización

RIESGOS DE CONTROL: SOLUCIÓN

- Ejemplo: Aplicando la Solución 2 con implementación HW
  - Si el salto no se realiza

El cálculo de la **dirección de salto** y la evaluación de **la condición** se realizan en la **etapa ID**



La Instrucción siguiente es la instrucción correcta

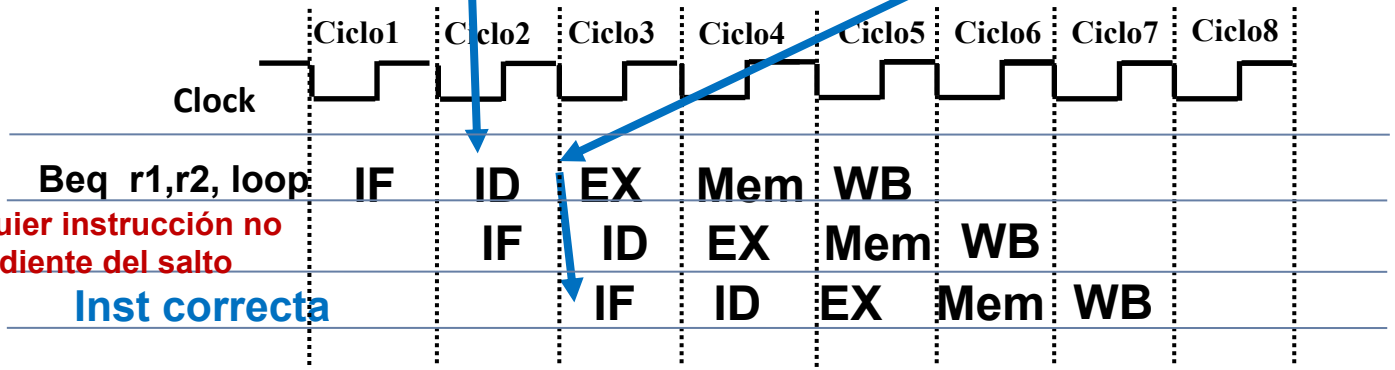
0 ciclos de penalización

RIESGOS DE CONTROL: SOLUCIÓN

- Ejemplo: Aplicando la Solución 2 con implementación SW: Saltos retardados

El cálculo de la **dirección de salto** y la evaluación de **la condición** se realizan en la **etapa ID**

La nueva dirección se carga en PC cuando llega el flanco de reloj **al entrar en Ex**



No hay ciclos de penalización

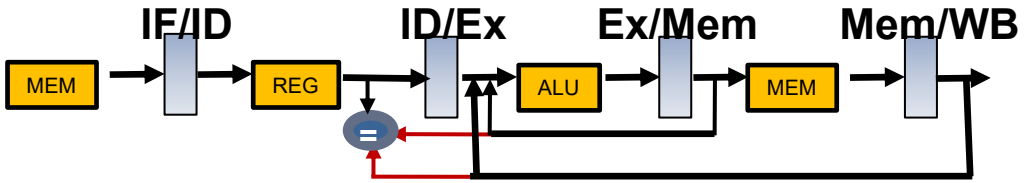
- **Lo ideal es elegir una instrucción que se tenga que ejecutar siempre**
- **Si no es posible, elegir una instrucción cuya ejecución no afecte al resultado**

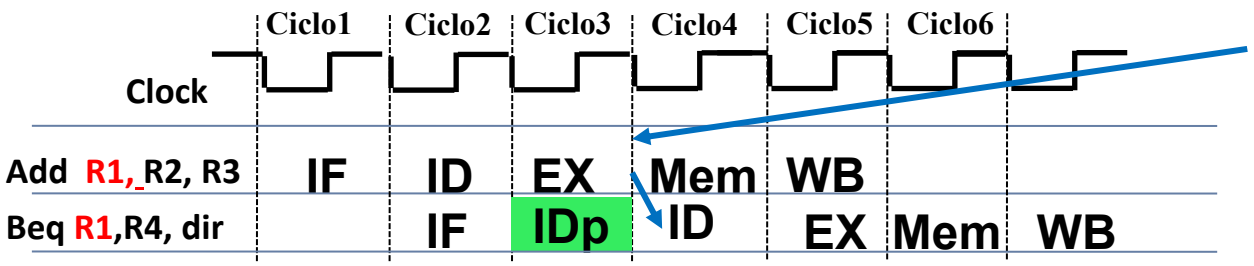
ÍNDICE

1. MIPS
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. **Riesgos de control con riesgos LDE**
7. Operaciones multiciclo
8. Excepciones
9. Rendimiento en los procesadores

RIESGO DE CONTROL + RIESGOS DE LDE

- La instrucción de salto necesita un dato que proporciona la instrucción anterior
  - Problema:
    - El salto evalúa la condición en la etapa ID
    - El cortocircuito tal y como está implementado no se lo proporciona
  - Solución: Ampliar el cortocircuito a la etapa ID





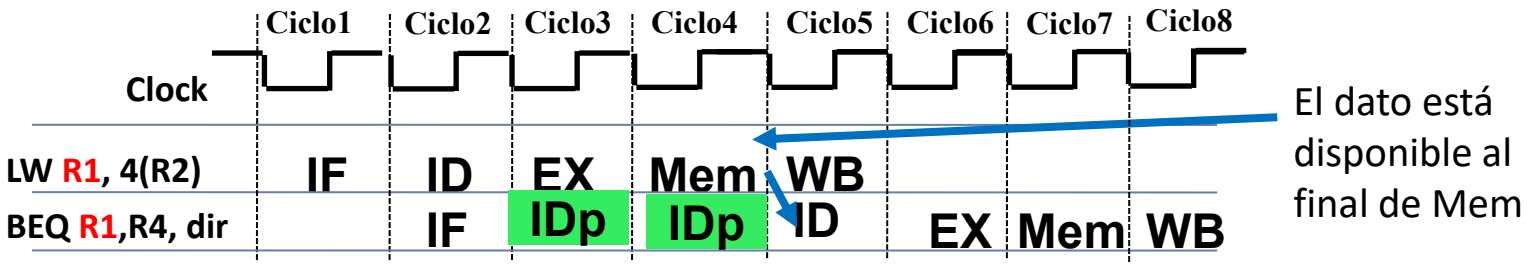
El dato está disponible al final de Ex

1 ciclo de espera

IDp ✓ID_P **parada por riesgo LDE** entre instrucción 1 e instrucción 2

RIESGO DE CONTROL + RIESGOS DE LDE

- La instrucción de salto necesita un dato que proporciona la instrucción anterior
  - Si la instrucción que proporciona el dato es un Load

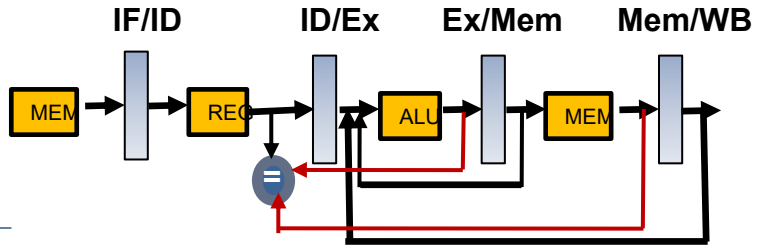


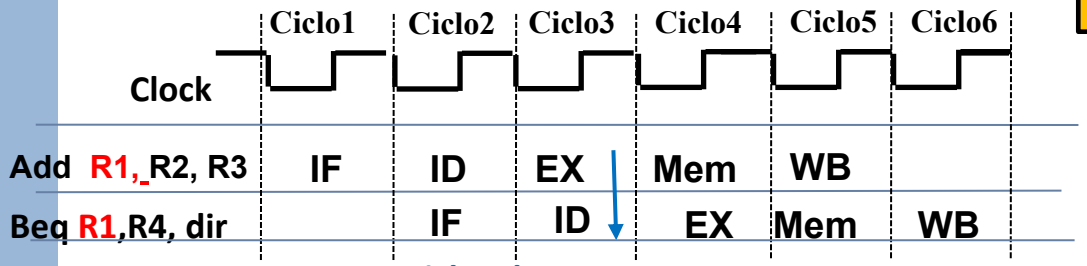
2 ciclos de espera

IDp ✓ID_P **parada por riesgo LDE** entre instrucción 1 e instrucción 2

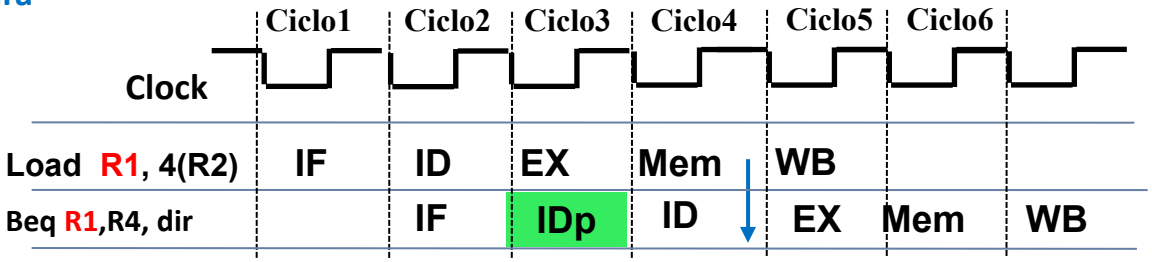
RIESGO DE CONTROL + RIESGOS DE LDE

- La instrucción de salto necesita un dato que proporciona la instrucción anterior
  - Si el forwarding es combinacional
  - El dato no se anticipa desde los registros del pipeline sino desde la salida de la ALU en el caso de una Aritmético-Lógica o desde la salida de DM en el caso del load





0 ciclos de espera

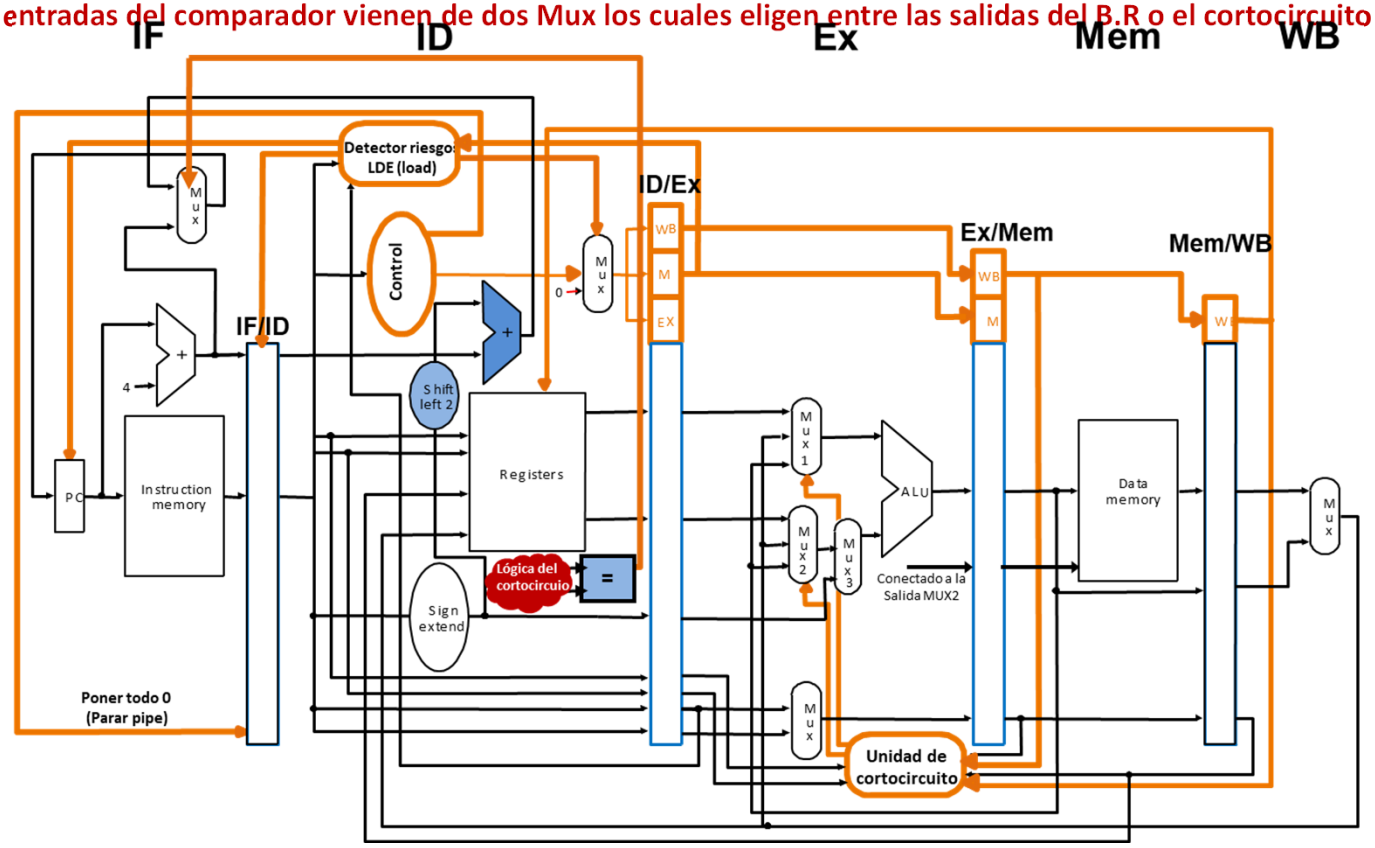


1 ciclo de espera

RESUMEN

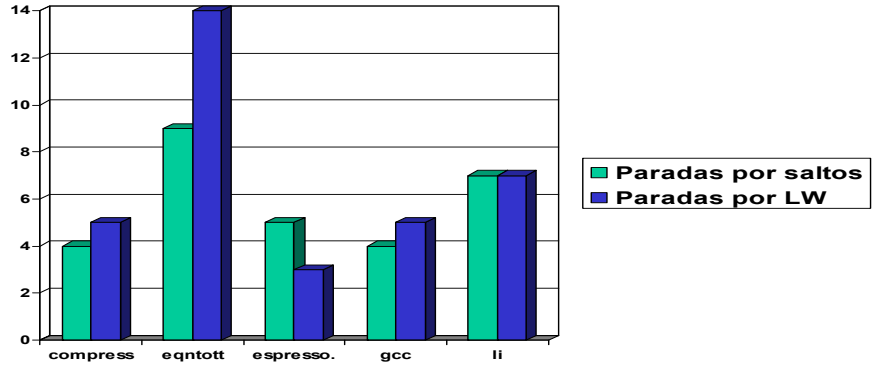
Ruta datos completa del procesador segmentado

- Tiene implementadas las soluciones óptimas para solucionar los riesgos vistos hasta ahora
- La entradas del comparador vienen de dos Mux los cuales eligen entre las salidas del B.R o el cortocircuito



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 se solucionan:
  - HW: Si el salto se realiza se introduce en el pipeline una instrucción NOP (= "0")
  - SW: con Saltos retardados



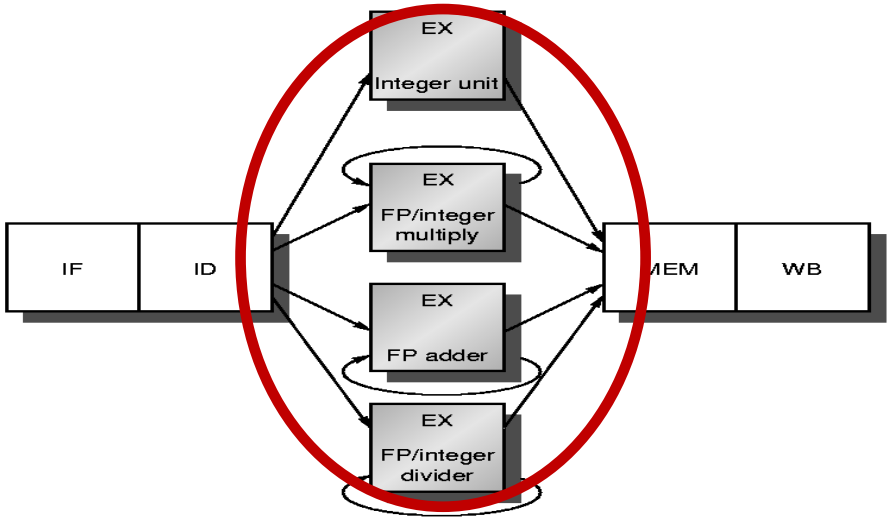
ÍNDICE

1. MIPS
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Riesgos de control con riesgos LDE
7. **Operaciones multiciclo**
8. Excepciones
9. Rendimiento en los procesadores

OPERACIONES MULTICICLO

- ¿ Qué ocurre si las instrucciones tienen diferente duración?
  - Esto ocurre cuando la operaciones requieren más de un ciclo de ejecución
    - Las operaciones en punto flotante
    - La multiplicación y división de enteros
  - **Latencia de las UF** : Nº de ciclos de duración de una instrucción en una UF

Unidad funcional	Latencia
ALU entera	1
Suma PF	4
Multiplicación PF	7
División PF	24

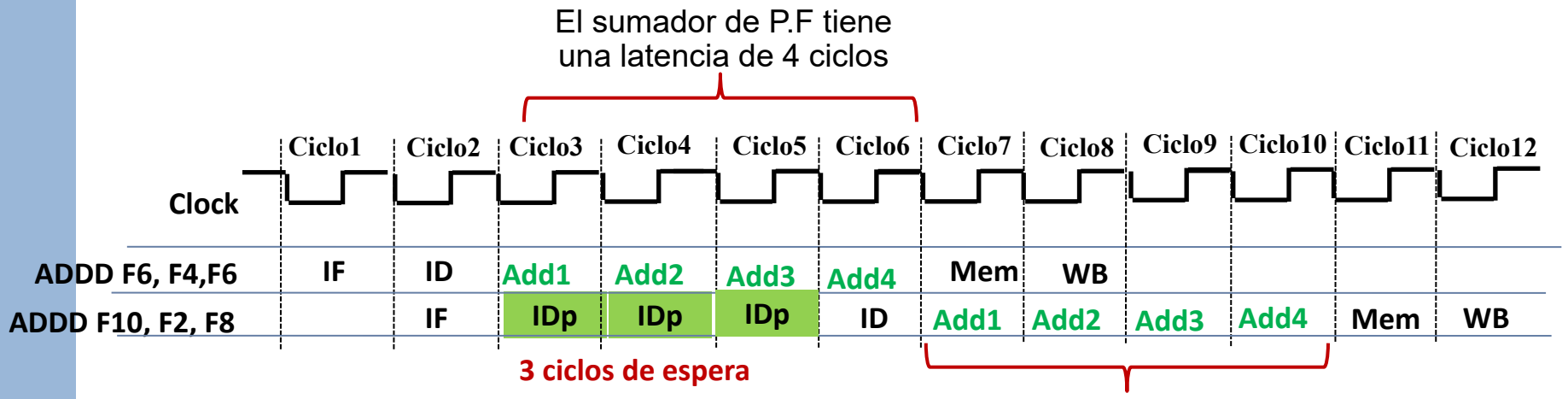


OPERACIONES MULTICICLO

- **Problemas**
  - Riesgos estructurales
  - **Mayor penalización** de los riesgos LDE
  - Aparecen riesgos EDE y EDL
  - Problemas con la **finalización fuera de orden**

OPERACIONES MULTICICLO

- Riesgo estructural: dos instrucciones necesitan la misma UF
  - Hay que esperar que la UF haya acabado la operación de la primera instrucción



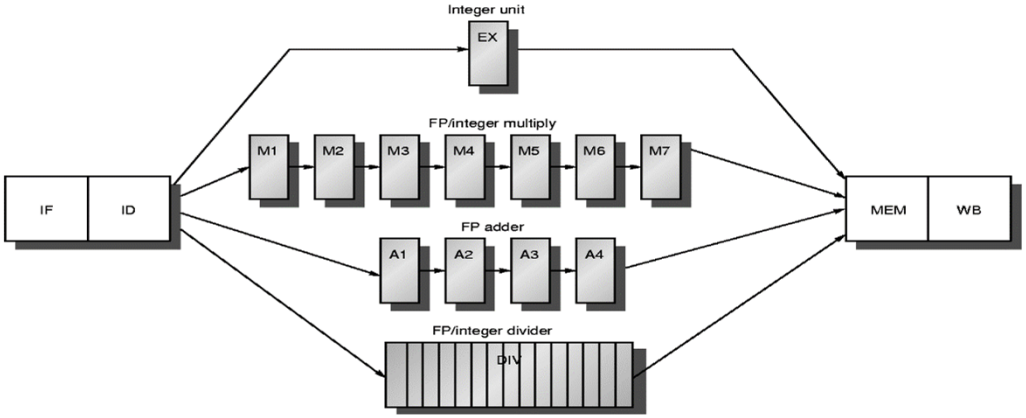
IDp ✓ID_P **parada por riesgo estructural, se necesita el sumador y no está disponible**

OPERACIONES MULTICICLO

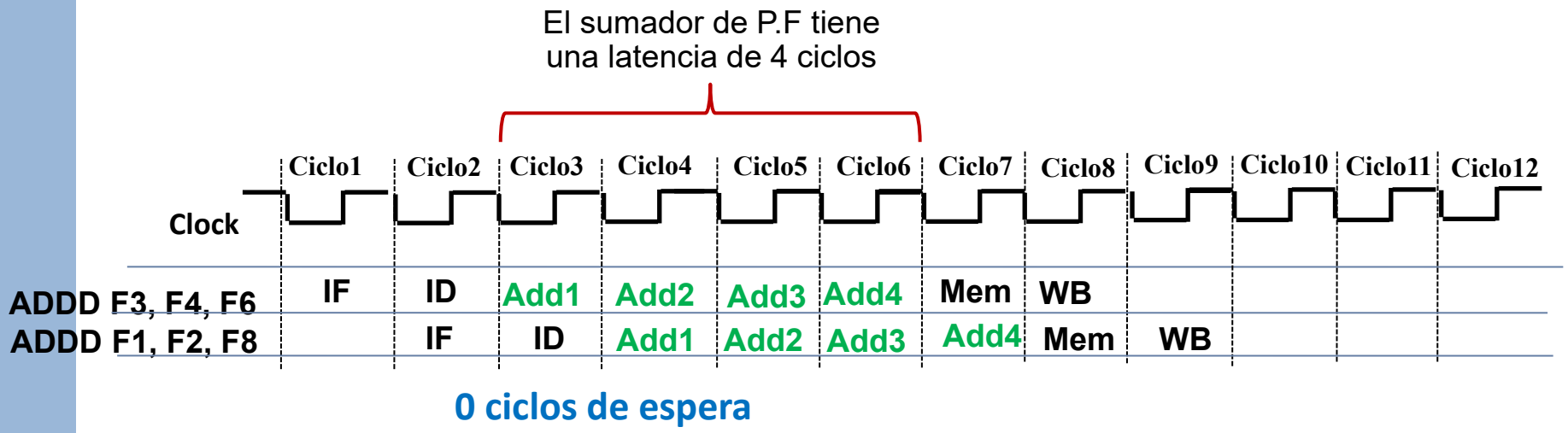
Riesgos estructurales

- Solución: Segmentar las UF con latencia > 1
- **Intervalo de iniciación:** Nº de ciclos que tiene que esperar una instrucción para poder utilizar una UF que está utilizando otra
- La división no suele estar segmentada: se tiene que detectar el riesgo y realizar paradas

Unidad funcional	Latencia	Intervalo de iniciación
ALU entera	1	1
Suma PF	4	1
Multiplicación PF	7	1
División PF	24	24



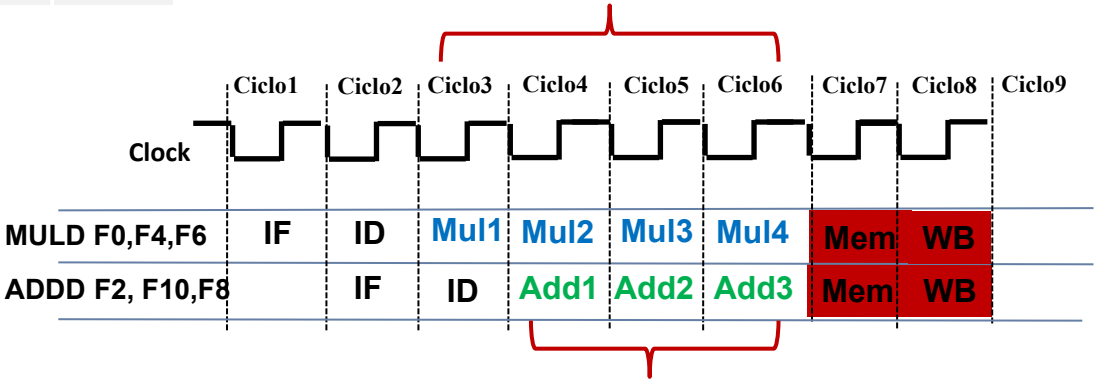
- Riesgos estructurales
  - Solución: Segmentar las UF con latencia > 1
  - Sólo hay que esperar que la UF haya acabado la operación asociada al primer ciclo de ejecución de la primera instrucción



Riesgo estructural

- Dos instrucciones no pueden acceder a la vez a la etapa de Mem
- Dos instrucciones no pueden acceder a la vez a la etapa WB

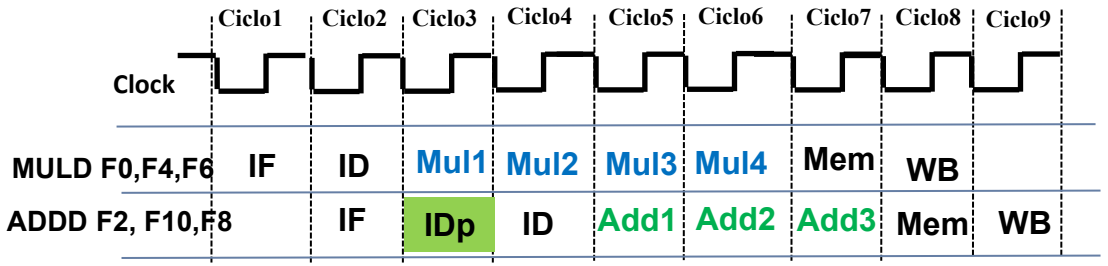
Unidad Funcional	Latencia
FP add	3
FP multiplicador	4



Solución 1:

- **Detener** la segunda instrucción en la **etapa de decodificación**

Unidad Funcional	Latencia
FP add	3
FP multiplicador	4

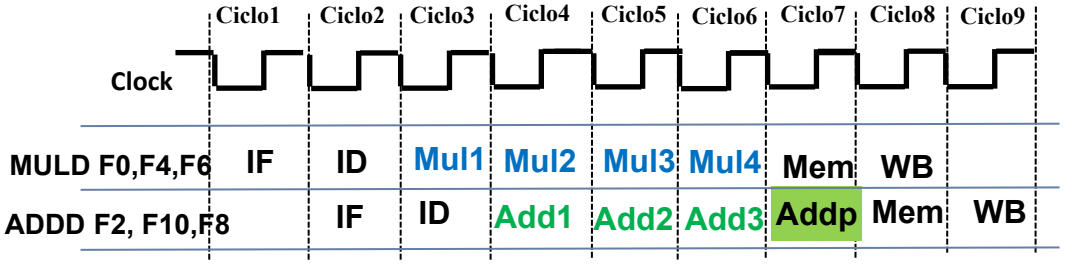


IDp ✓ID_P parada por **riesgo estructural,** dos instrucciones van a acceder a la vez a la memoria de datos

Solución 2:

- **Detener** las instrucciones conflictivas **al final de ejecución**
  - Necesidad de establecer prioridades de acceso
    - Dar mayor prioridad a la unidad de mayor latencia
  - Lógica de detección y generación de paradas en dos puntos diferentes

Unidad Funcional	Latencia
FP add	3
FP multiplicador	4



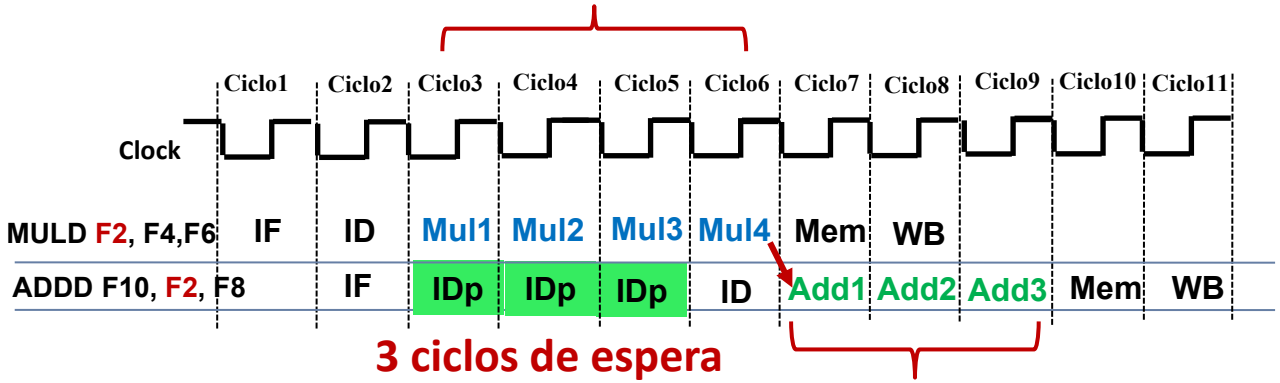
Addp ✓ID_P **parada por riesgo estructural, dos instrucciones van a acceder a la vez a la memoria de datos**

OPERACIONES MULTICICLO: RIESGOS LDE

Qué pasa cuando hay Riesgo LDE y el dato lo proporciona una instrucción Aritmético-Lógica

- El **cortocircuito NO elimina todas las paradas**

Unidad Funcional	Latencia
FP add	3
FP multiplicador	4



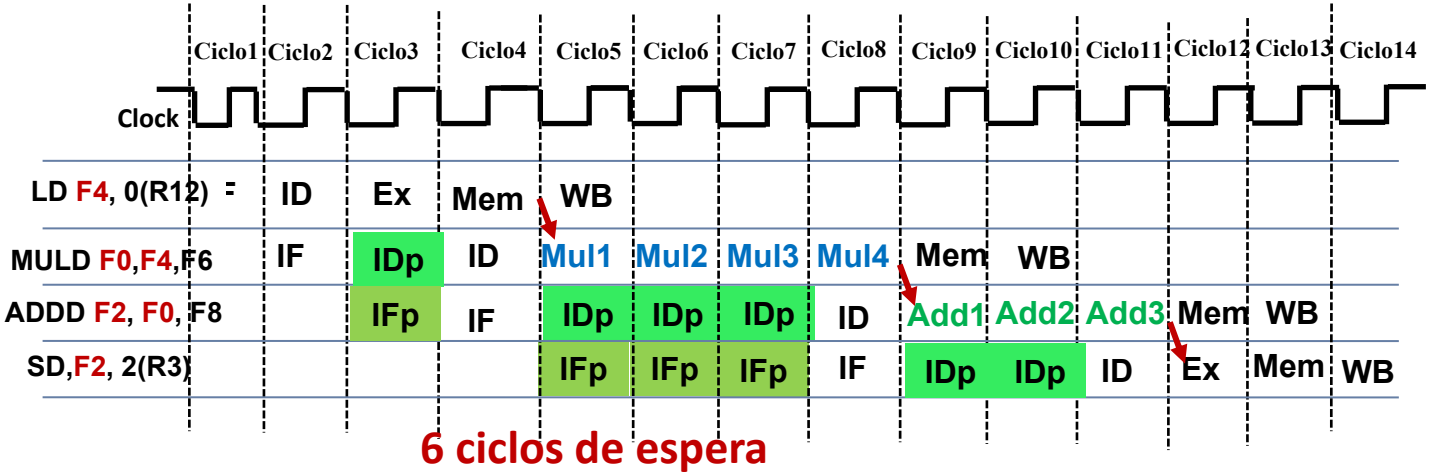
IDp D_P **parada por riesgo LDE** entre instrucción 1 e instrucción 2

OPERACIONES MULTICICLO: RIESGOS LDE

Qué pasa cuando hay Riesgo LDE y el dato lo proporciona un load

- La instrucción que depende del load además tiene una parada
  - El **cortocircuito NO elimina todas las paradas**

Unidad Funcional	Latencia
FP add	3
FP multiplicador	4



6 ciclos de espera

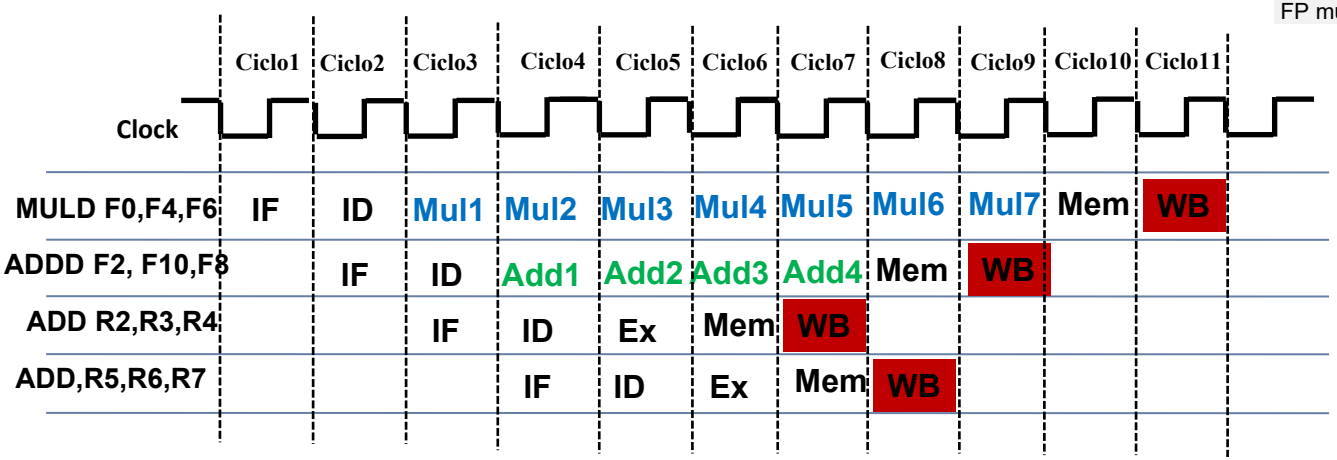
IDp **parada por riesgo LDE** entre instrucción 1 e instrucción 2

IFp **parada por fallo estructural**, la etapa está ocupada por la instrucción anterior

Finalización fuera de orden

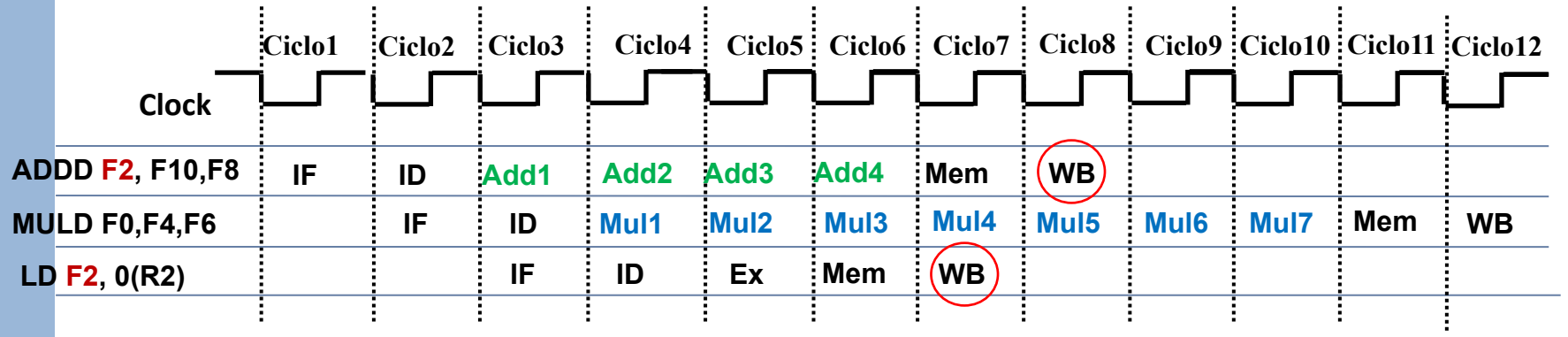
- Las instrucciones pueden acabar en un orden diferente al de lanzamiento
- **Problemas**:
  - Conflictos por escritura simultánea en el banco de registros (riesgo estructural)
  - Aparecen riesgos de escritura después de escritura (EDE)

Unidad Funcional	Latencia
FP add	4
FP multiplicador	7



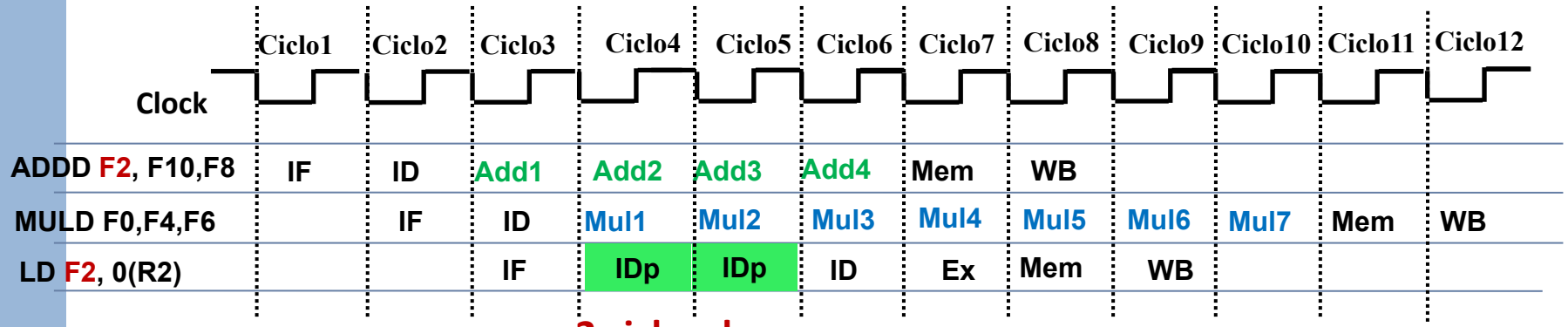
OPERACIONES MULTICICLO: RIESGOS EDE

- Problema: Escritura después de escritura



OPERACIONES MULTICICLO: RIESGOS EDE

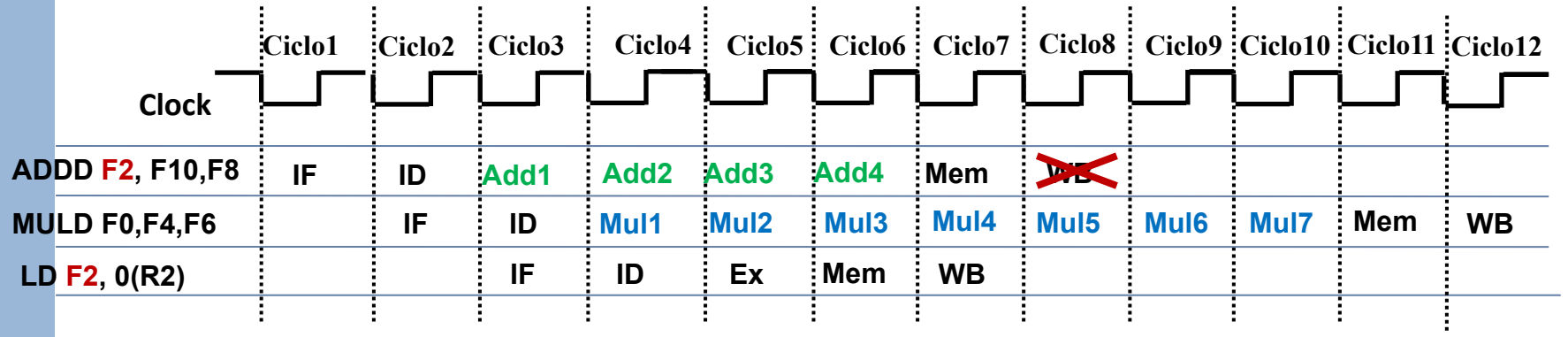
- Solución 1
  - **Detener la instrucción que provoca el riesgo** (la segunda)
  - El número de paradas depende de la longitud de la primera instrucción y de la distancia entre ambas



IDp ✓ID_P **parada por riesgo EDE** entre instrucción 1 e instrucción 2

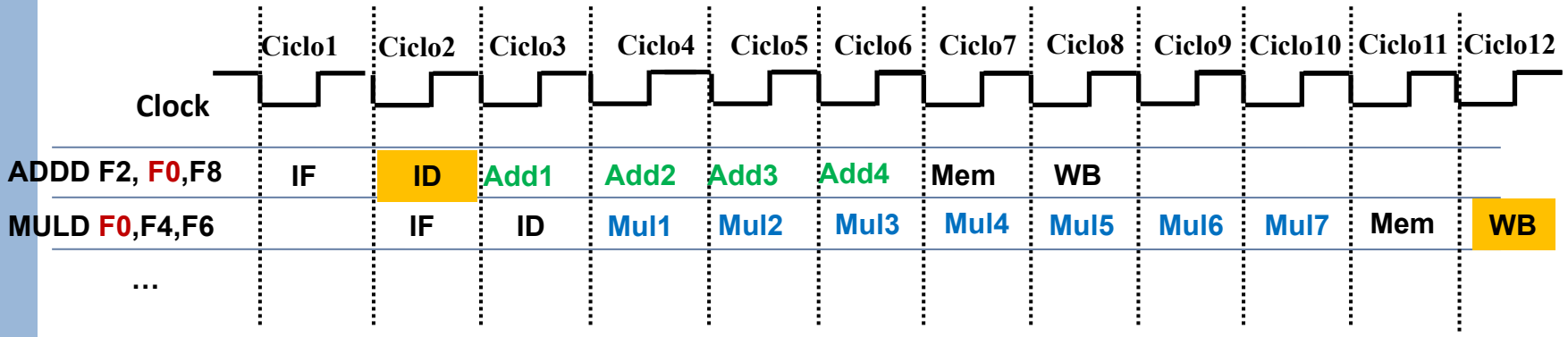
OPERACIONES MULTICICLO: RIESGOS EDE

- Solución 2
  - **Inhibir la escritura de la primera instrucción**



OPERACIONES MULTICICLO: RIESGOS EDL

- Estos riesgos **NO se producen** por construcción del pipeline
  - La lectura siempre se realiza en la etapa de decodificación
  - La escritura siempre se realiza en la última etapa



ÍNDICE

1. MIPS
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Riesgos de control con riesgos LDE
7. Operaciones multiciclo
8. **Excepciones**
9. Rendimiento en los procesadores

TIPOS DE EXCEPCIONES

- **Dos tipos de excepciones**
  - 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 página…
    - Son síncronas con la ejecución del programa
    - La condiciones deben ser almacenadas
    - El programa debe ser abortado o continuado desde esa instrucción

GESTIÓN EXCEPCIONES MIPS MULTICICLO

- Nos centramos, como ejemplo, en dos traps : Instrucción indefinida y Overflow aritmético

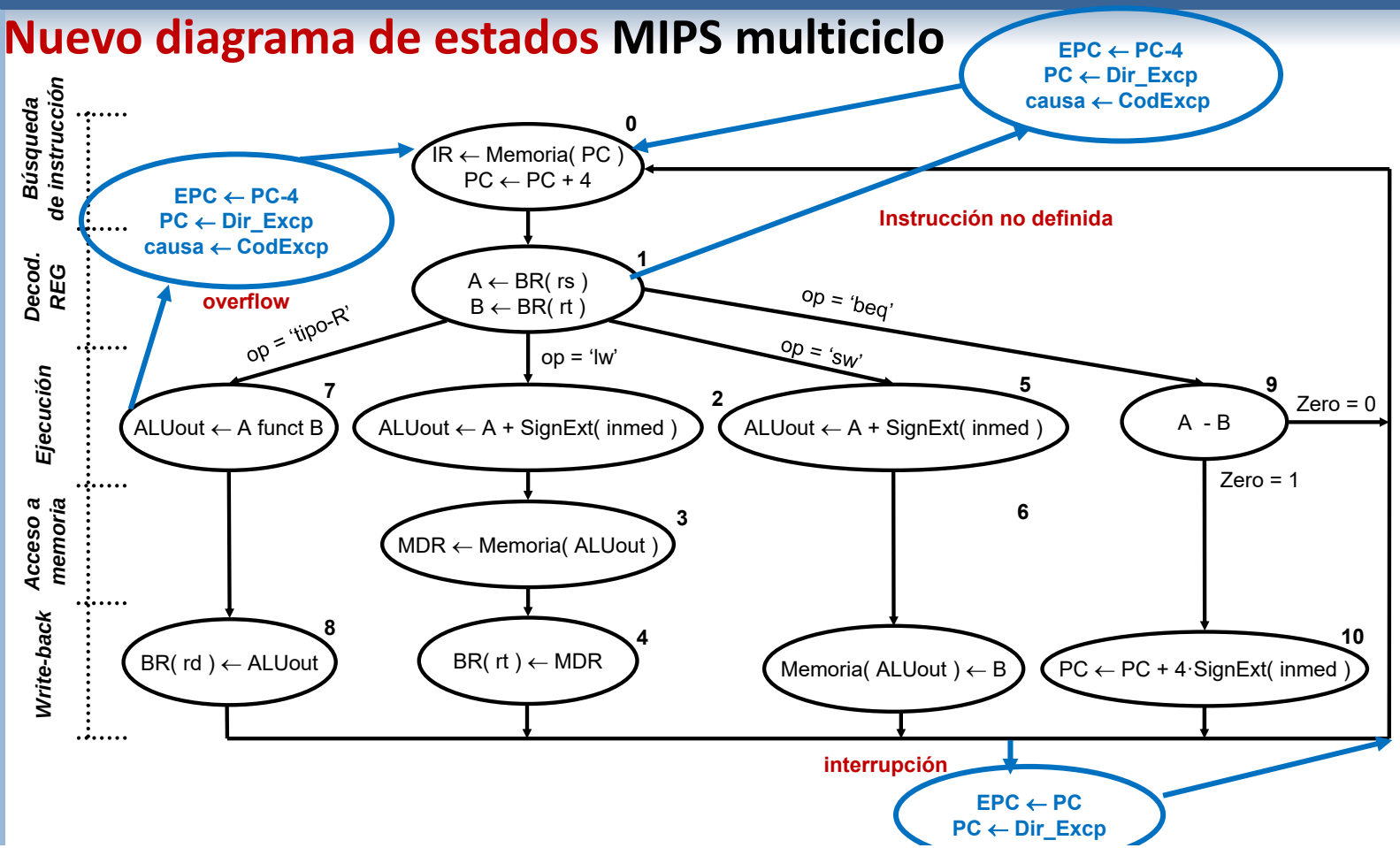
- Acciones básicas:
  - Salvar el contador de programa de la instrucción interrumpida en el *registro EPC* (exception program counter)
  - Transferir el control al sistema operativo en alguna dirección especificada
  - El S.O. realizará la acción apropiada, como ejecutar alguna tarea asociada al overflow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

GESTIÓN EXCEPCIONES MIPS MULTICICLO

- 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.
  - Se añade una entrada al multiplexor que controla la carga del PC, para poder seleccionar la dirección de la rutina de tratamiento de excepción

GESTIÓN EXCEPCIONES MIPS MULTICICLO

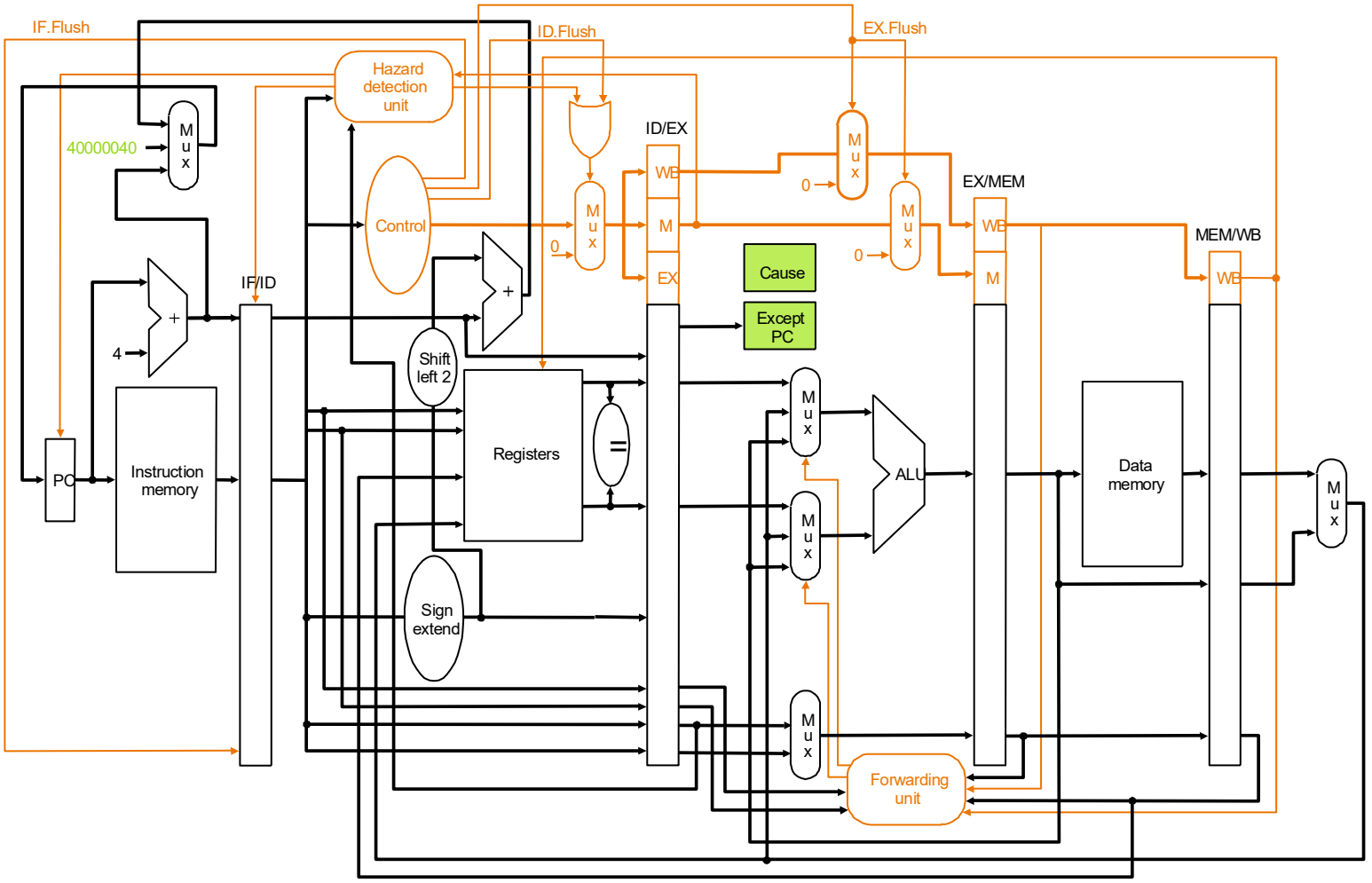
Nuevo diagrama de estados MIPS multiciclo



Hardware añadido para gestionar las excepciones

– **Registros “Cause” y EPC**; dirección constante de salto a tratamiento de excepción **“4000040”**

– Generación de las señales de control **IF.Flush, ID.Flush, EX.Flush** para **vaciar el *pipeline***



GESTIÓN EXCEPCIONES MIPS SEGMENTADO

- Cualquier instrucción en el pipeline puede provocar una excepción
- Problema: El solapamiento en la ejecución de las instrucciones dificulta el saber si una instrucción puede cambiar el estado de la máquina sin peligro

Etapa	Excepción
IF	Fallo de página de instrucción; Acceso no alineado; Violación de protección
ID	Instrucción ilegal
EX	Excepción aritmética
Mem	Fallo de página de datos; Acceso no alineado; Violación de protección
WB	Ninguna

GESTIÓN EXCEPCIONES MIPS SEGMENTADO

- Cuando se produce una excepción el sistema debe resolverla: **Excepciones precisas.**
  - Se tratan en el mismo orden que se tratarían en un procesador multiciclo
  - Completar todas las instrucciones anteriores a la excepción
  - Tratar la instrucción que produce la interrupción y las siguientes como si no hubieran empezado
- El sistema debe recuperar el estado previo a la excepción y recomenzar la ejecución
- Excepciones problemáticas: (un ejemplo: fallo de página)
  - Ocurren en el medio de una instrucción
  - Deben ser recomenzables
- **Interrupciones externas** (I/O)
  - Vaciar el pipeline e introducir no operaciones (NOPs)
  - Almacenar el PC con la dirección de la subrutina de tratamiento de la interrupción

GESTIÓN EXCEPCIONES MIPS SEGMENTADO

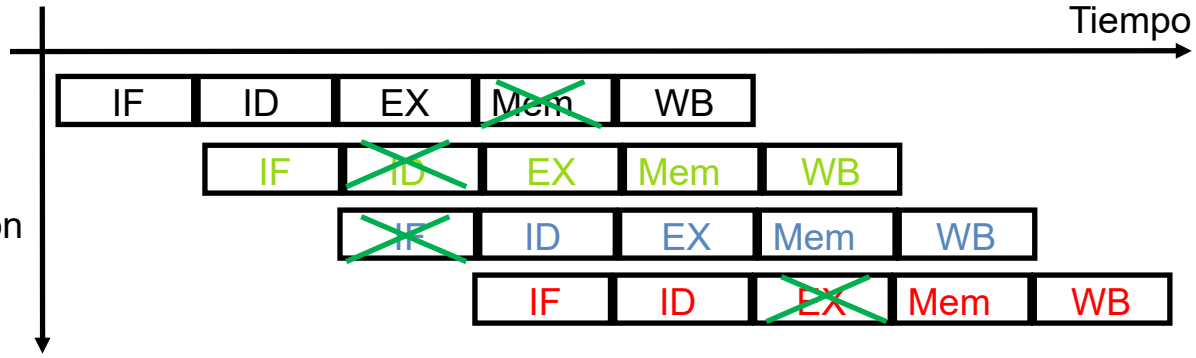
Excepciones

Inst 1: **Fallo** de página de datos

Inst 2: **Instrucción ilegal**

Inst 3: **Fallo** de página de instrucción

Inst 4: **Overflow**



- Las excepciones deben tratarse, primero la de la instrucción mas antigua
  - En este caso se debe atender el Fallo de pagina de datos y recomenzar en la siguiente instrucción
- Problema: las excepciones no aparecen en el orden de entrada de las instrucciones en el pipeline

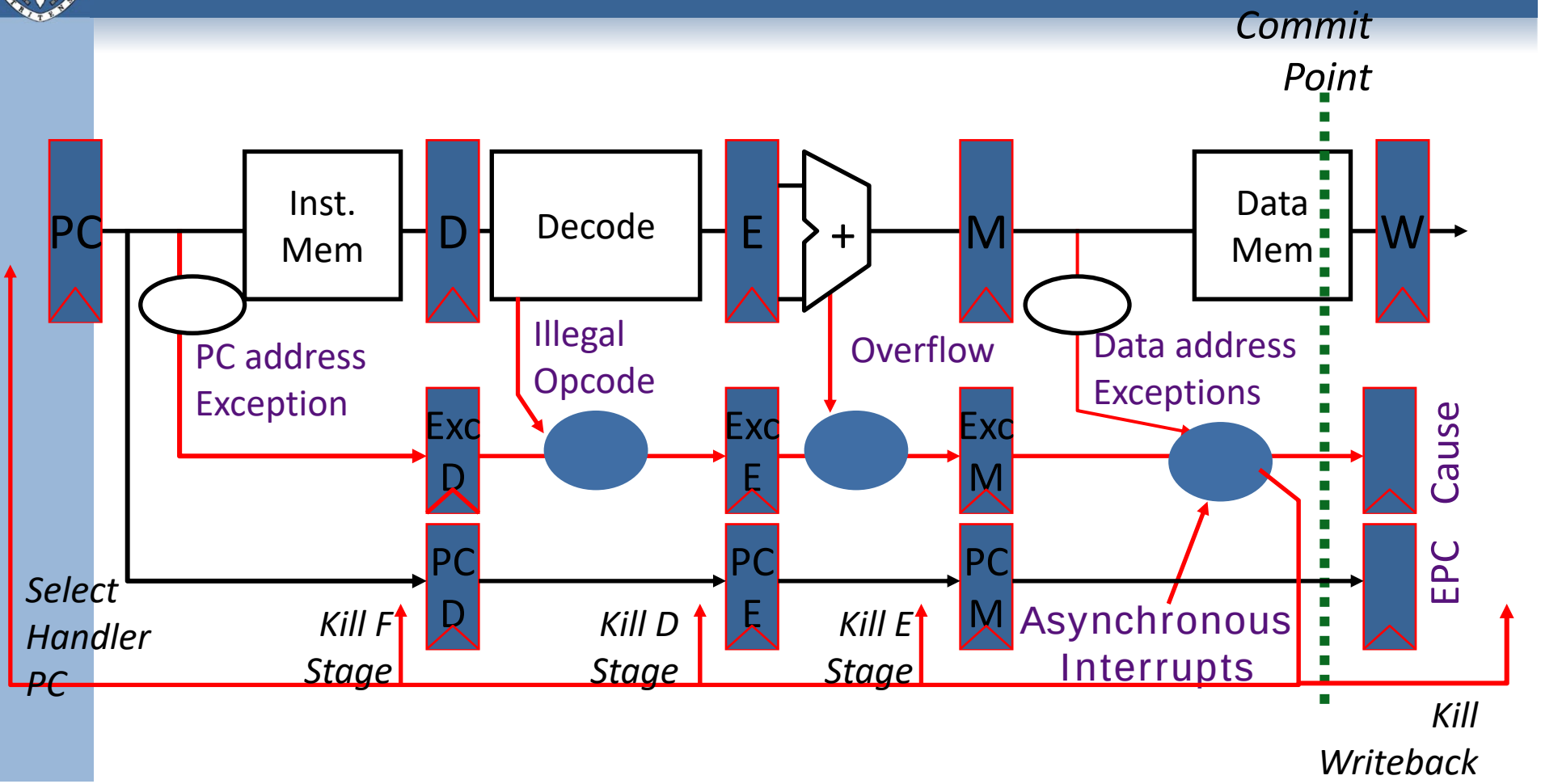
¡Las excepciones de la segunda y tercera instrucción aparecen antes que la de la primera !

GESTIÓN EXCEPCIONES MIPS SEGMENTADO

- **Solución**

  - Un Registro de excepciones para cada instrucción dentro del procesador (5 en el caso de estudio)
  - El registro tiene un bit por cada etapa donde es posible la excepción
  - En cuanto se activa un bit se impide la escritura en el procesador para evitar que cambie el estado
  - El registro se chequea al final de MEM inicio de WB
  - Con este registro de excepciones se implementa un mecanismo para excepciones precisas (se atienden por orden)

GESTIÓN EXCEPCIONES MIPS SEGMENTADO



ÍNDICE

1. MIPS
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Riesgos de control con riesgos LDE
7. Operaciones multiciclo
8. Excepciones
9. **Rendimiento de los procesadores**

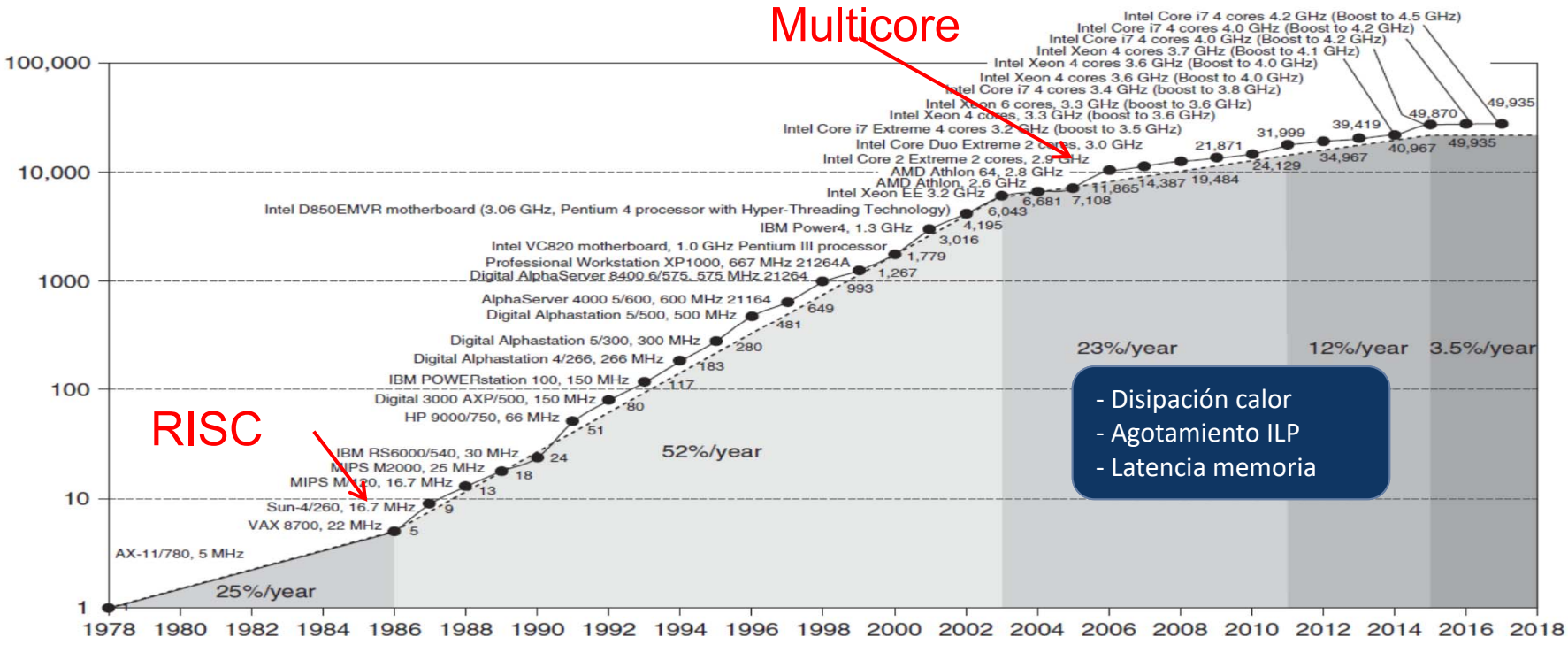
RENDIMIENTO

"Los buenos programadores se han preocupado siempre por el rendimiento de sus programas porque la rápida obtención de resultados es crucial para crear programas de éxito"

D. A. Patterson y J. L. Hennessy

CRECIMIENTO DEL RENDIMIENTO DE LOS PROCESADORES

Medida de rendimiento utilizada:
número de veces más rápido qué el VAX-11/780



RENDIMIENTO DE LOS PROCESADORES

- ¿Cuántos ciclos tarda en ejecutarse este programa?
  - Depende del procesador: por ejemplo en el MIPS multiciclo

lw r1, 0(r0) → 5

lw r2, 4(r0) → 5

add r3, r1, r2 → 4

beq r3, r5, 1 → 4

sub r3, r3, r5 → 4

sw r3, 8(r0) → 4

- ¿Y cuánto Tiempo?
  - Depende de la frecuencia del procesador

MEDIDAS DEL RENDIMIENTO

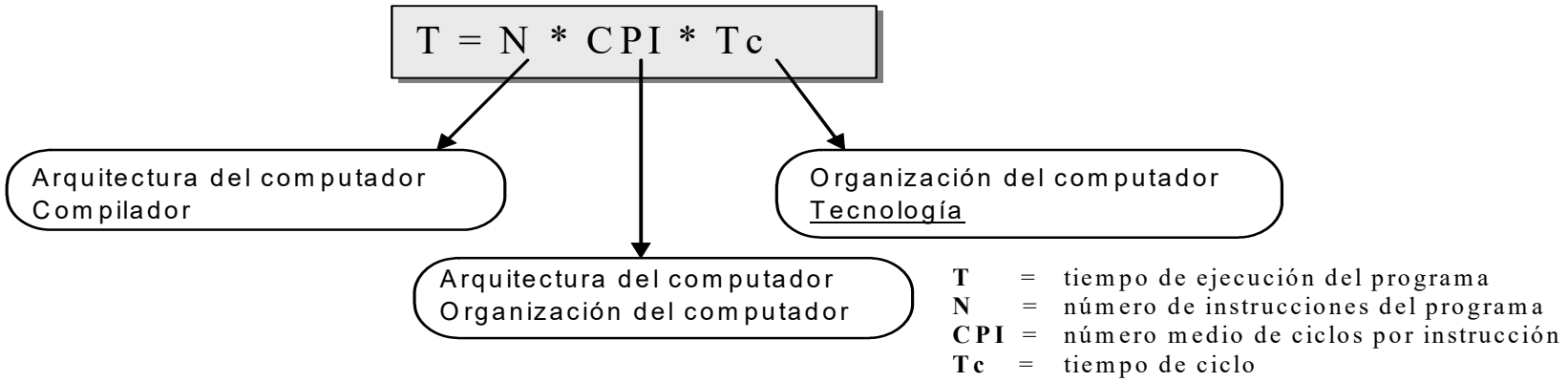
- Para poder comparar diferentes procesadores hace falta establecer una medida del rendimiento que permita cuantificar los resultados de la comparación
  - **Métrica**: establece la unidad de medida, que casi siempre es el tiempo, aunque hay que considerar dos aspectos diferentes del tiempo:
    - **Tiempo de ejecución:** tiempo que tarda en realizarse una tarea determinada
    - **Productividad** (throughput): tareas realizadas por unidad de tiempo
  - **Patrón de medida**: establece los programas que se utilizan para realizar la medida (benchmarks). Existen muchos posibles benchmarks aunque los más utilizados son:
    - Núcleos de programas reales: SPEC (www.spec.org)
    - Programas sintéticos

MEDIDAS DEL RENDIMIENTO: TIEMPO DE EJECUCIÓN

- ***Tiempo de respuesta*:** tiempo para completar una tarea (que percibe el usuario).
- ***Tiempo de CPU:*** tiempo que tarda en ejecutarse un programa, sin contar el tiempo de E/S o el tiempo utilizado para ejecutar otros programas. Se divide en:
  - ***Tiempo de CPU utilizado por el usuario:*** tiempo que la CPU utiliza para ejecutar el programa del usuario sin tener en cuenta el tiempo de espera debido a la E/S
  - ***Tiempo de CPU utilizado por el S.O:*** tiempo que el S.O. emplea para realizar su gestión interna.
- La función ***time*** de Unix produce: 90.7u 12.9s 2:39 65%, donde:
  - Tiempo de CPU del usuario = 90.7 segundos
  - Tiempo de CPU utilizado por el sistema = 12.9 segundos
  - Tiempo de CPU= 90.7 seg.+ 12.9seg = 103.6
  - Tiempo de respuesta = 2 minutos 39 segundos =159 segundos
  - Tiempo de CPU = 65% del tiempo de respuesta = 159 segundos*0.65 = 103.6
  - Tiempo esperando operaciones de E/S y/o el tiempo ejecutando otras tareas 35% del tiempo de respuesta = 159 segundos*0.35 = 55.6 segundos

MEDIDAS DEL RENDIMIENTO: TIEMPO DE EJECUCIÓN

- **Tiempo de ejecución de un programa**



- **CPI =** Ciclos medios por instrucción
  - Una instrucción necesita varios ciclos de reloj para su ejecución
  - Diferentes instrucciones tardan diferentes cantidades de tiempo
  - CPI = Es una suma ponderada del número de ciclos que tarda por separado cada tipo de instrucción

MEDIDAS DEL RENDIMIENTO: TIEMPO DE EJECUCIÓN

Cálculo del CPI

- El número total de ciclos de reloj de la CPU se calcula como:

$$CPI = \frac{\left(\sum_{i=1}^{n} CPI_i \bullet NI_i \right)}{NI} = \sum_{i=1}^{n} (CPI_i \bullet \frac{NI_i}{NI})$$

  - NI_i = número de veces que el grupo/tipo de instrucciones *i* es ejecutado en un programa
  - CPI_i = número medio de ciclos para el conjunto de instrucciones del grupo/tipo *i*
- Podemos calcular el *CPI* multiplicando cada CPI_i individual por la fracción de ocurrencias de las instrucciones *i* en el programa.
- CPI_i debe ser medido, y no calculado a partir de la tabla del manual de referencia

PÉRDIDA DE RENDIMIENTO

- El CPI ideal es 1
- Hay pérdidas de rendimiento por las paradas del pipe

CPI $_{real}$ = CPI $_{ideal}$ + Penalización media por instrucción =

$$= 1 + \sum_{i=1}^{\#tipos\ de\ instr} Penalizacion_i * Frec_i$$

- Caso de los saltos. Un programa típico 30% de saltos

  CPI = 1 + (1 x 0.3) = 1.3

$$Speep\_up = \frac{Nºinstrucciones * Nº\ etapas}{Nºinstrucciones * CPI} = \frac{5}{1{,}3} = 3{,}84$$

- Eficiencia:
  - Speedup $_{real}$ / Speedup $_{ideal}$ =3.84 / 5 = 0.76

  Se pierde un 24 % respecto al caso ideal

MEDIDAS DEL RENDIMIENTO: MIPS

- **MIPS** (Millones de Instrucciones Por Segundo)

$$MIPS = \frac{NI}{Tiempo\ de\ ejecucion * 10^6} = \frac{1}{CPI * 10^6 * Tc}$$

- Dependen del repertorio de instrucciones, por lo que resulta un parámetro difícil de utilizar para comparar máquinas con diferente repertorio de instrucciones
- Varían entre programas ejecutados en el mismo computador
- Pueden variar inversamente al rendimiento, como ocurre en máquinas con hardware especial para punto flotante

MEDIDAS DEL RENDIMIENTO: MFLOPS

- **MFLOPS** (Millones de Operaciones en punto FLotante Por Segundo)

$$MFLOPS = \frac{Numero\ de\ operaciones\ en\ punto\ flotante\ de\ un\ programa}{Tiempo\ de\ ejecucion\ *\ 10^6}$$

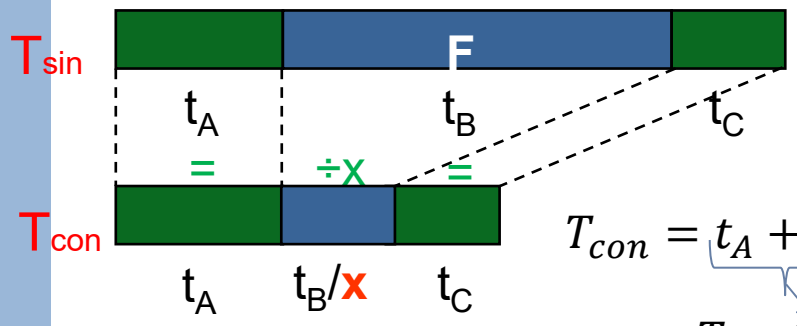
  - Existen operaciones en coma flotante rápidas (como la suma) o lentas (como la división), por lo que puede resultar una medida poco significativa.
  - Se han utilizado los MFLOPS normalizados, que dan distinto peso a las diferentes operaciones.
  - Por ejemplo: suma, resta, comparación y multiplicación se les da peso 1; división y raíz cuadrada peso 4; y exponenciación, trigonométricas, etc. peso 8:

MEDIDAS DEL RENDIMIENTO. LEY DE AMDAHL

- **Ganancia de velocidad (*speedup*): Ley de Amdahl**
  - ***La mejora obtenida en el rendimiento global de un computador al utilizar un modo de ejecución más rápido está limitada por la fracción de tiempo que se puede utilizar dicho modo***
- Un principio básico: Hacer rápidas las funciones frecuentes.

$$\text{Speedup} = \frac{Tiempo\ de\ ejecucion\ sin\ mejora\ (Tsin)}{Tiempo\ de\ ejecucion\ con\ mejora\ (Tcon)}$$

- Qué speedup se obtendrá al aplicar una cierta mejora, M, que permite ejecutar una fracción, F, del código x veces más rápido.



$$T_{sin} = t_A + t_B + t_C \qquad F = \frac{t_B}{T_{sin}}$$

$$T_{con} = t_A + t_C + \frac{t_B}{x} = \quad \{\text{sustituyendo}\}$$

$$= T_{sin}\,(1 - F) + T_{sin}\frac{F}{x} = T_{sin}\left[(1 - F) + \frac{F}{x}\right]$$

MEDIDAS DE RENDIMIENTO: LEY DE AMDAHL

$$T_{con} = T_{sin}\left[(1-F) + \frac{F}{x}\right]$$

$$Speedup = \frac{T_{sin}}{T_{con}} = \frac{1}{(1-F) + \frac{F}{x}}$$

Ejemplo 1:. El 10% del tiempo de ejecución de mi programa es consumido por operaciones en PF. Se mejora la implementación de la operaciones PF reduciendo su tiempo a la mitad

$T_{con} = T_{sin}$ x (0.9 + 0.1 / **2**) = 0.95 x T_{sin} Speedup = $\frac{1}{0.95}$ = 1.053

Mejora de sólo un 5.3%

Ejemplo 2: Para mejorar la velocidad de una aplicación, se ejecuta una parte que consumía el 90% del tiempo sobre 100 procesadores en paralelo. El 10% restante no admite la ejecución en paralelo.

$T_{con} = T_{sin}$ x (0.1 + 0.9 / **100**) = 0.109 x T_{sin} Speedup = $\frac{1}{0.109}$ = 9.17

El uso de 100 procesadores sólo multiplica la velocidad por 9.17

Medidas de rendimiento: Ley de Amdahl

- Concepto de eficiencia (E)

$$\mathbf{E} = \frac{Speedup}{x} = \frac{\frac{1}{(1-F)+\frac{F}{x}}}{x} = \frac{1}{x(1-F)+F} = \frac{1}{x+F(1-x)}$$

El valor máximo posible de E es 1 (para lo que se necesitaría que F=1)

- Ampliación del Ejemplo 2:

Procesadores (x)	F	Speedup	Eficiencia
10	0.9	5.26	0,526 (52.6%)
100	0.9	9.17	0,0917 (9.17%)
1000	0.9	9.91	0.00991 (0.99%)

- Observaciones:

1. La fracción no paralelizable de un cálculo, (1-F), limita seriamente el Speedup, incluso cuando esta fracción es pequeña.
2. A partir de cierto punto, aumentar mucho el nº de procesadores apenas mejora el Speedup, por lo que se degradada mucho la Eficiencia.

MEDIDAS DE RENDIMIENTO: LEY DE AMDAHL

- ***Everyone knows Amdahl's Law but quickly forgets it!***

  Thomas Puzak (IBM's T. J. Watson Research Center)