Arquitectura e Ingeniería de Computadores

Tema 2
Procesadores Segmentados

DEPARTAMENTO DE ARQUITECTURA DE COMPUTADORES Y AUTOMÁTICA

Curso 2009-2010

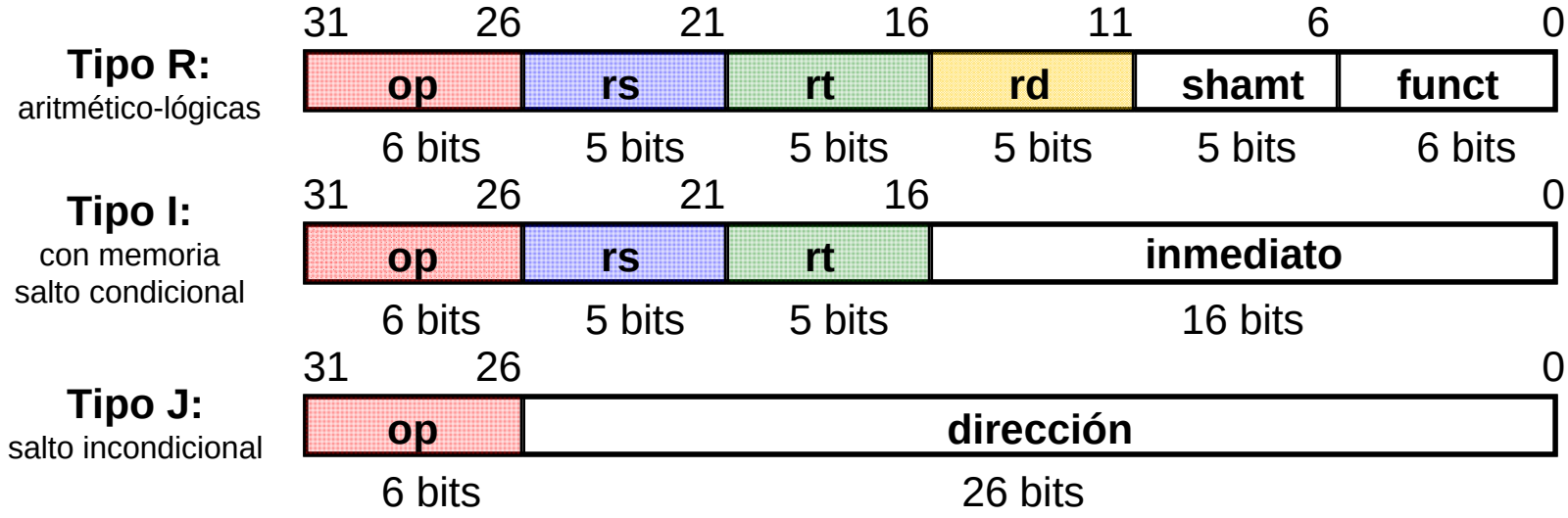
Contenidos

- ✓ Introducción: Recordatorio MPIS-DLX
- ✓ Excepciones y control
- ✓ Segmentación
- ✓ Riesgos: Estructurales, de datos y de control
- ✓ Segmentación del procesador. Diseño del control
- ✓ Diseño del control con riesgos
- ✓ Excepciones: una segunda mirada
- ✓ Operaciones multi-ciclo
- ✓ Un Ejemplo: MIPS R4000
- ✓ Bibliografía
  - o Apéndice A [HePa07]
  - o Capítulos 4 y 5 de [SiFK97]
  - o Simulador WinDLX

Recordatorio

❑ Arquitectura MIPS (DLX)

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

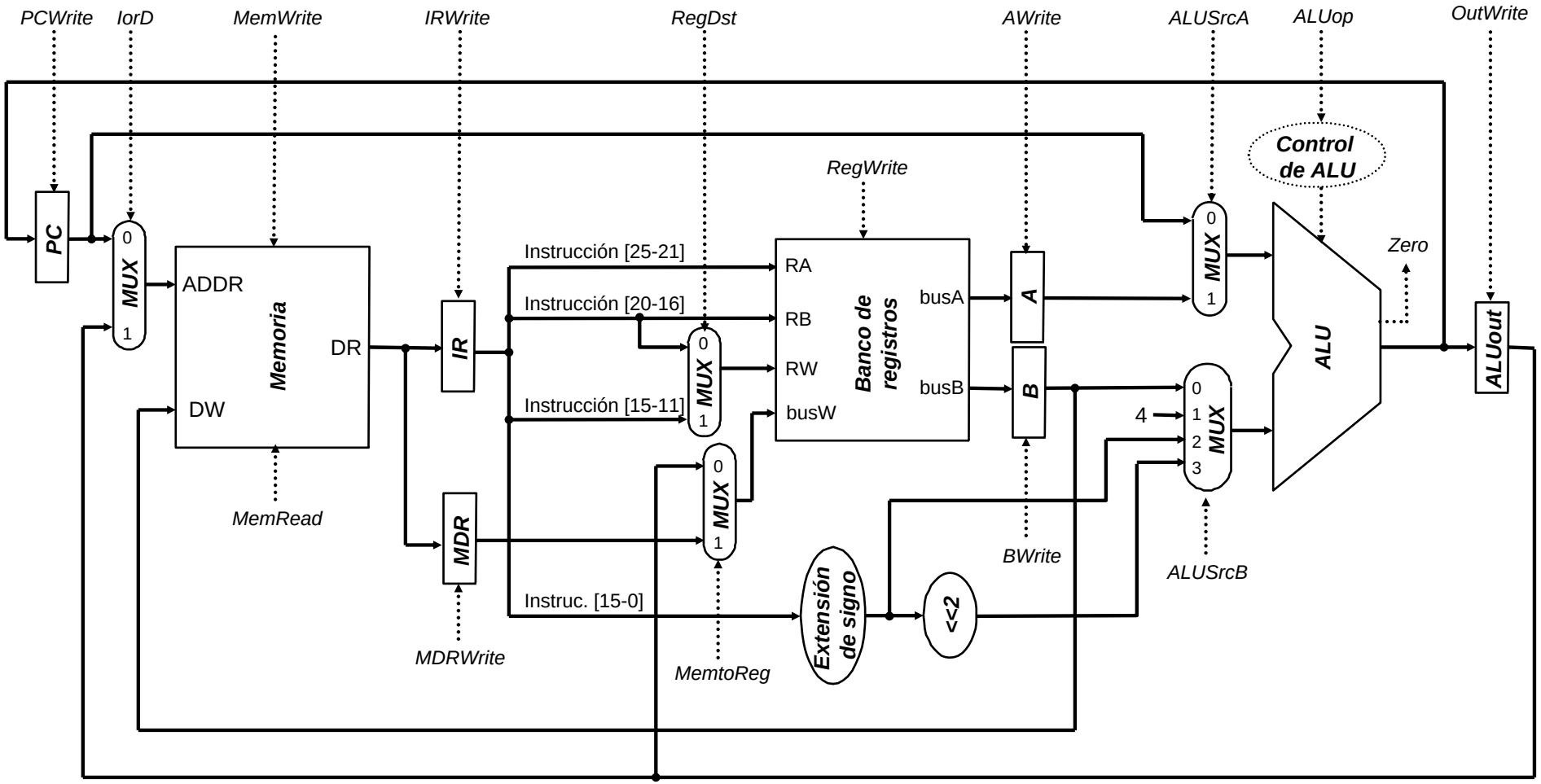


✓ El significado de los campos es:

- **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 a registro-base
- **dirección**: dirección destino del salto

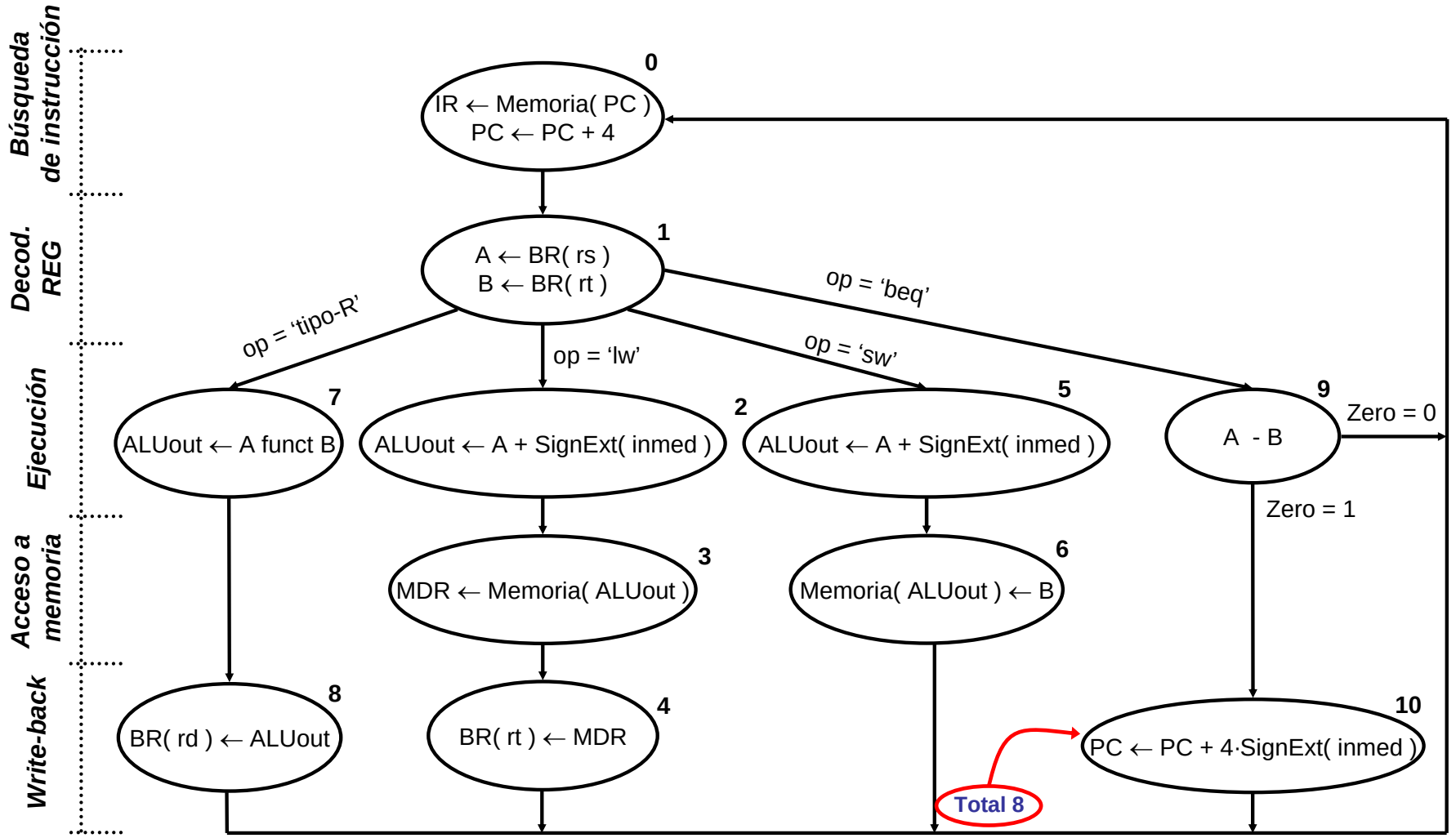
Recordatorio

❑ Ruta de datos (multiciclo)



Recordatorio

❑ Diagrama de estados del controlador multiciclo



Recordatorio

- Microprogramación
  - ✓ Implementa la instrucciones mediante un procesador simple que las *interpreta* (microprograma)
  - ✓ Es una buena alternativa para implementar repertorios de instrucciones complejos (CISC)
  - ✓ Penaliza el tiempo de ejecución. Hay que leer la memoria de control cada ciclo
- ¿ Existe otra alternativa ?
  - ✓ ¿Podrían los compiladores generar directamente microinstrucciones?
  - ✓ La mayoría de los programas utilizan instrucciones simples.
  - ✓ ¿Los microprogramas podían almacenarse en RAM ?
  - ✓ Esta memoria RAM podría ser una cache de microinstrucciones

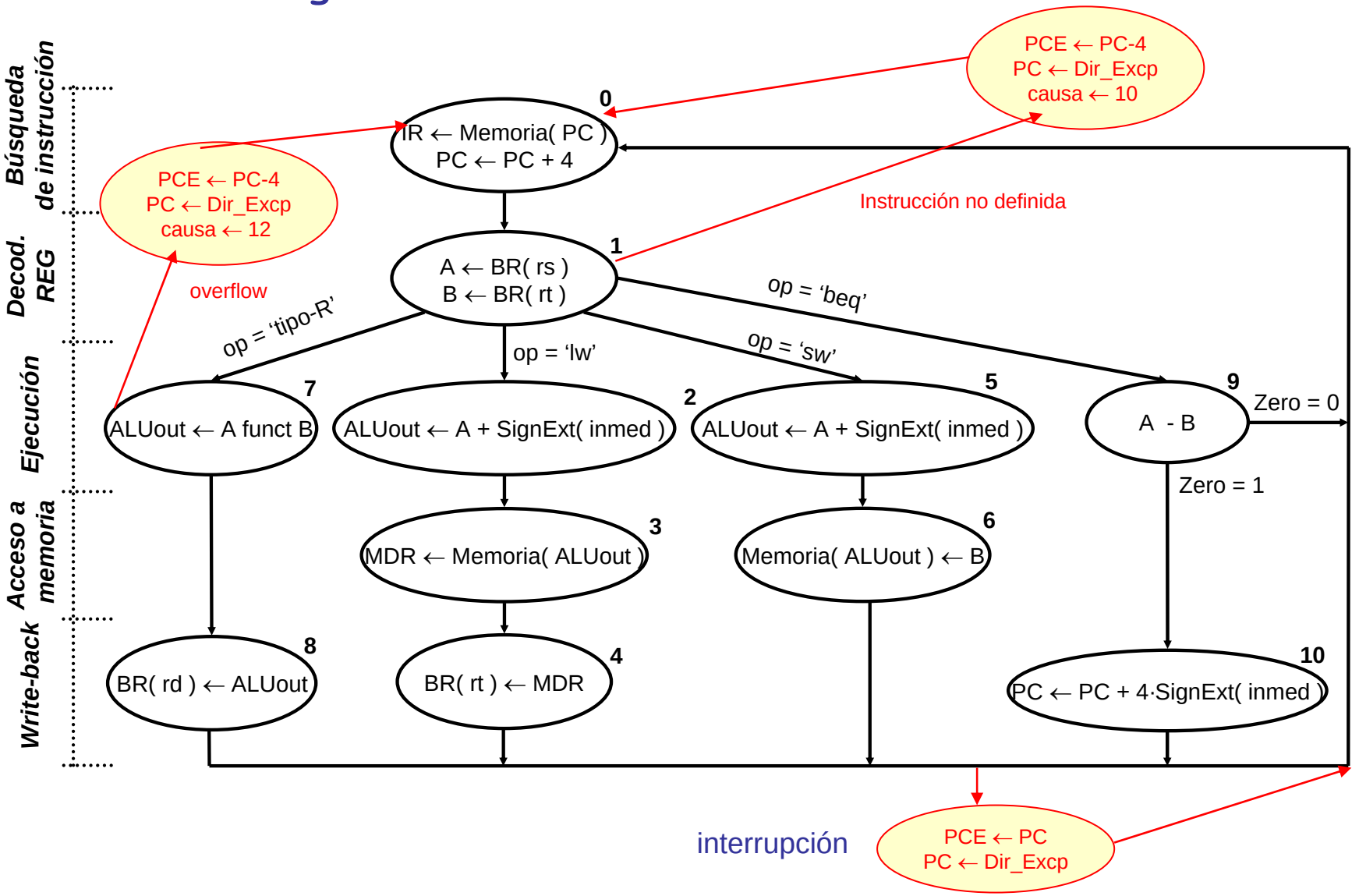
Conclusión: Eliminar la interpretación mediante un microprograma y compilar directamente a un lenguaje maquina equivalente a las microinstrucciones. Juego de instrucciones simple (RISC)

Excepciones y control

- 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
- Requerimientos
  - ✓ Tratamiento dentro del control
  - ✓ Salvar el estado (CP y la causa)
    - Stack 68k, x86
    - Registros específicos CPE, Causa

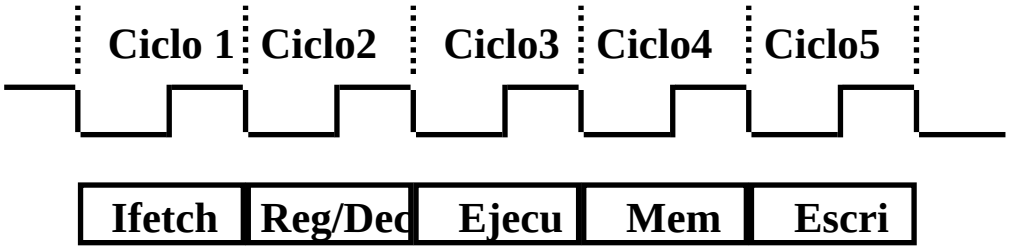
Excepciones y control

❑ Nuevo diagrama de estados



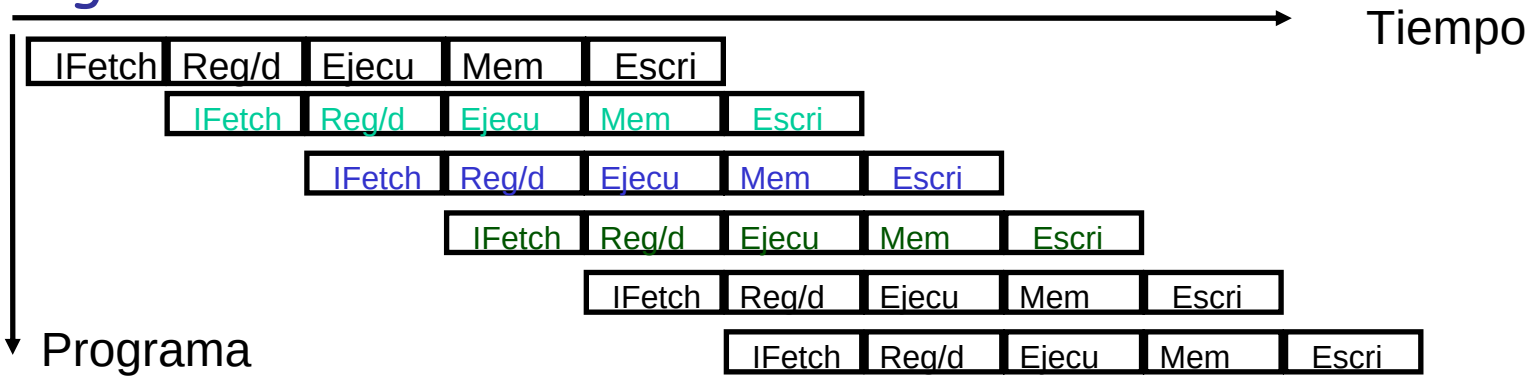
Segmentación

- Etapas de una instrucción (Load)



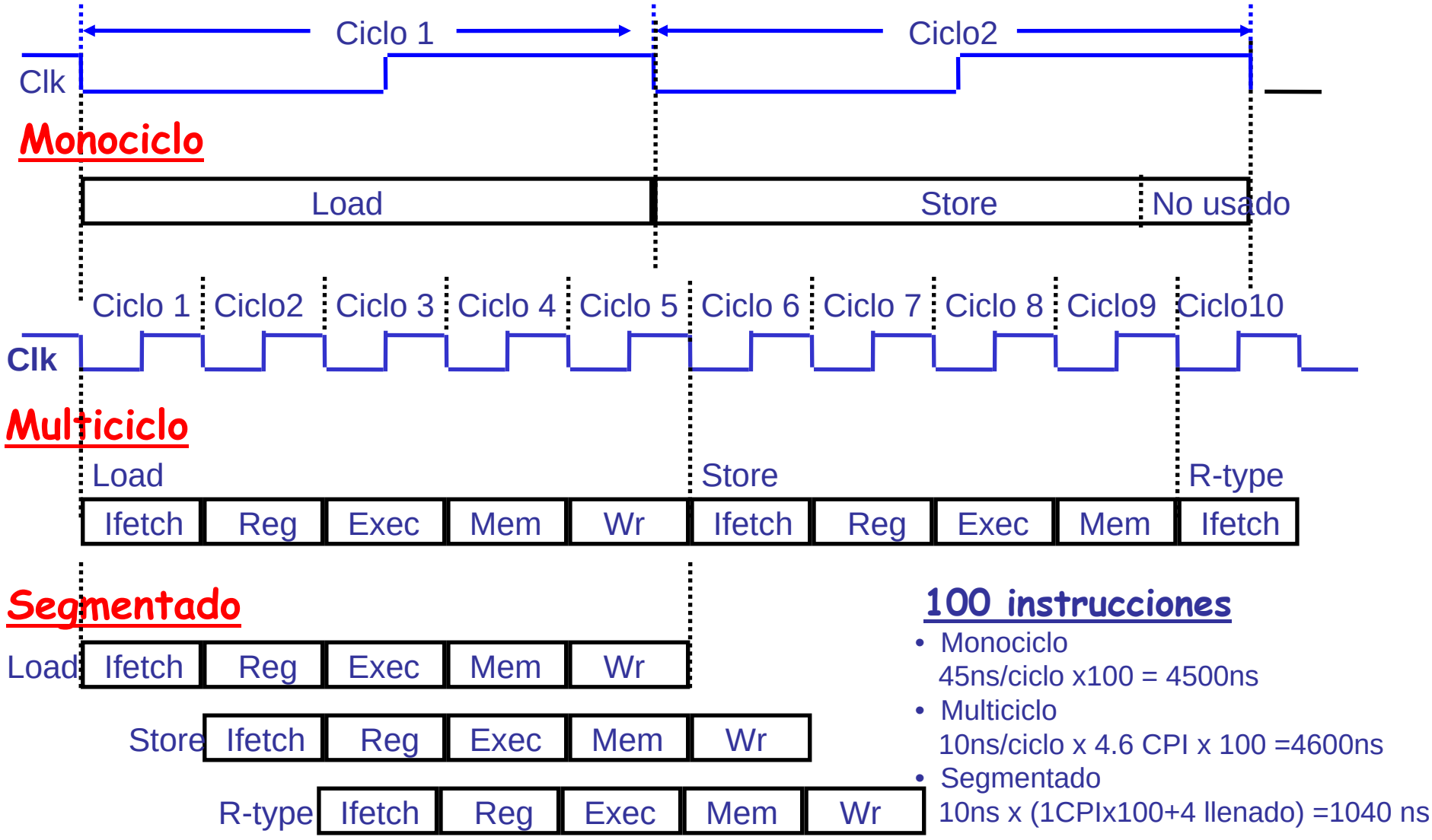
- Ifetch (búsqueda). Lee una instrucción desde la memoria
- Reg/dec. Decodifica la instrucción y lee registros
- Ejecuta. Calcula dirección de memoria
- Mem. Lee el dato de la memoria
- Escri. Escribe el dato en el registro

- Segmentación



Segmentación

- Monociclo, multiciclo, segmentado



100 instrucciones

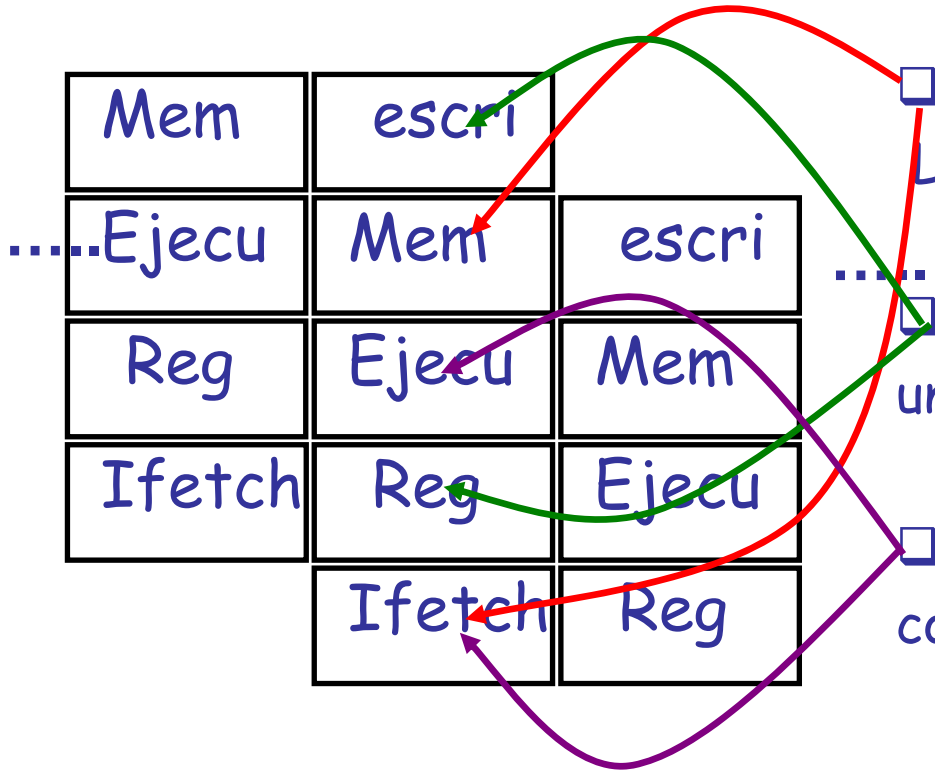
- Monociclo
  45ns/ciclo x100 = 4500ns
- Multiciclo
  10ns/ciclo x 4.6 CPI x 100 =4600ns
- Segmentado
  10ns x (1CPIx100+4 llenado) =1040 ns

Riesgos

- Situaciones que impiden que 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*

Riesgos estructurales.

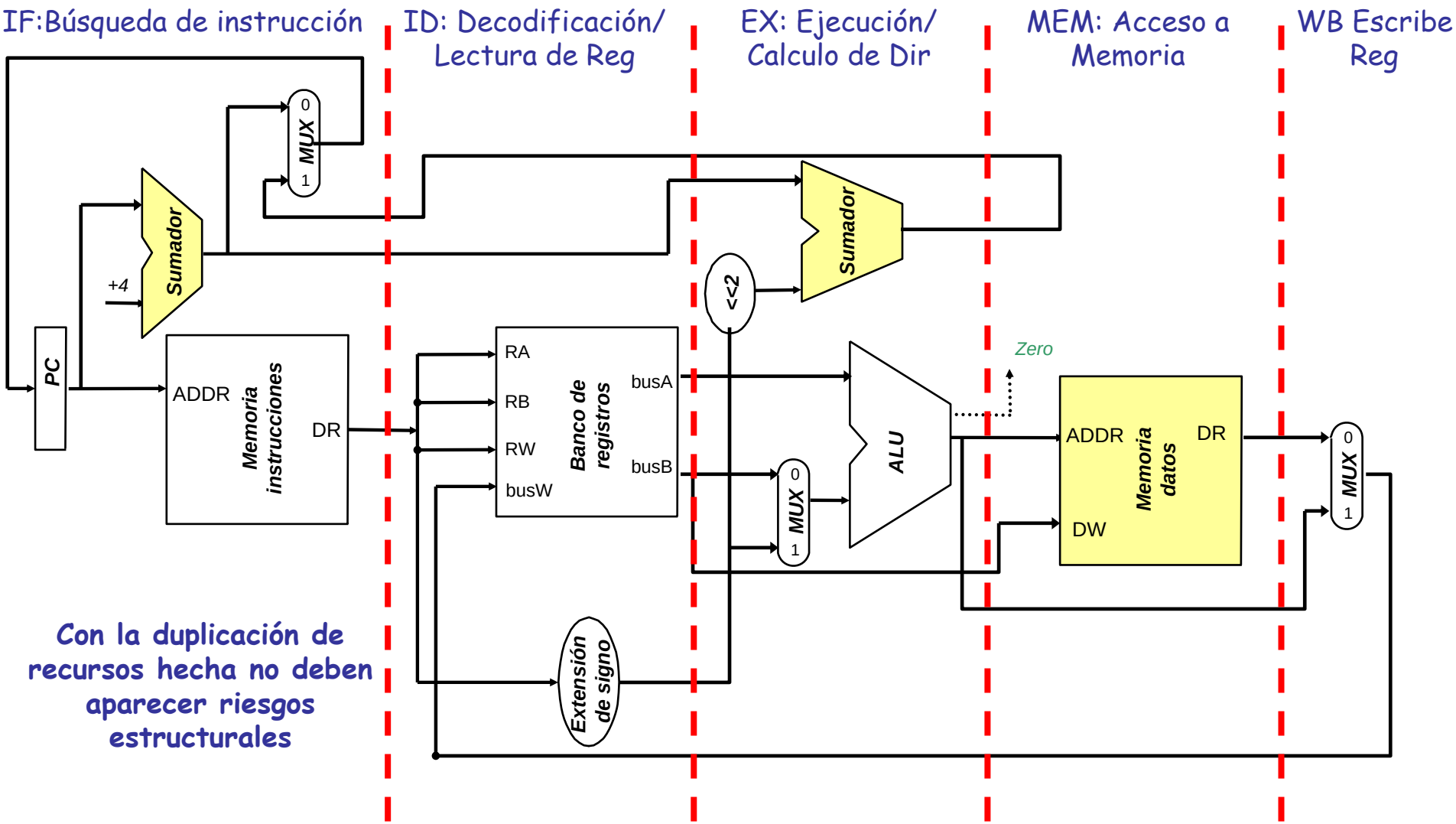
Objetivo: Ejecutar sin conflicto cualquier combinación de instrucciones



- Memoria. Leer una instrucción. Leer o escribir un operando
- Registros. Dos lecturas, una escritura
- ALU. Calculo de operando, dirección, condición, actualización de PC

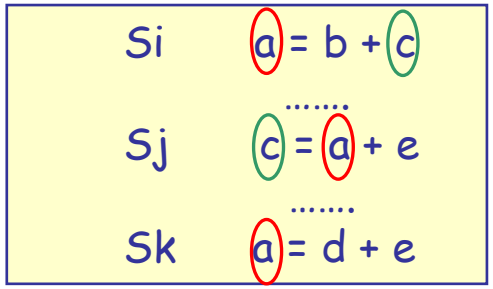
Segmentación: Riesgos estructurales

- Nueva Ruta de datos del DLX para ejecución segmentada (idea inicial)



Segmentación : 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.



Dominio: operandos de la instrucción
Rango: resultado de la instrucción

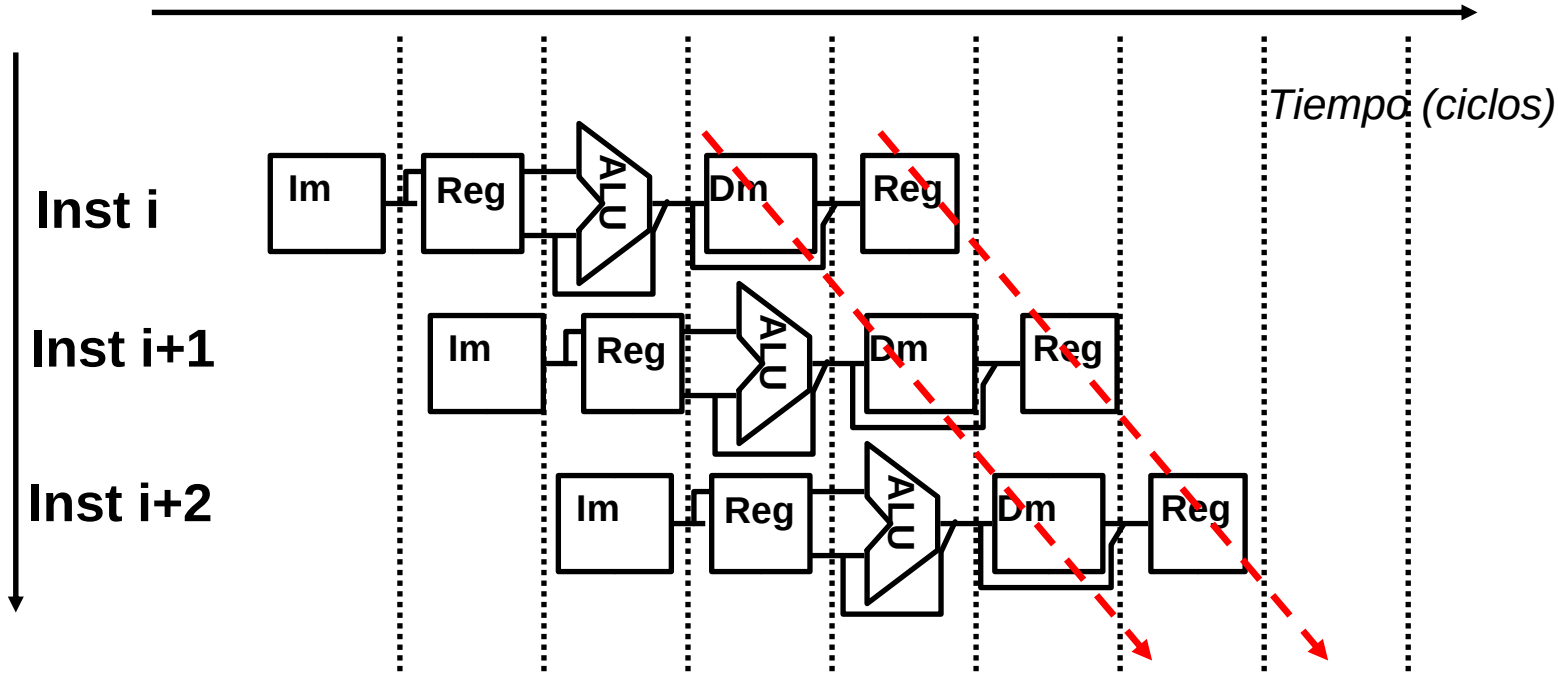
Situaciones: i precede a j

$D(i) \cap D(j) \neq \phi$ no RIESGO
$D(i) \cap R(j) \neq \phi$ riesgo EDL (WAR)
$R(i) \cap D(j) \neq \phi$ riesgo LDE (RAW)
$R(i) \cap R(j) \neq \phi$ riesgo EDE (WAW)

Riesgos de datos

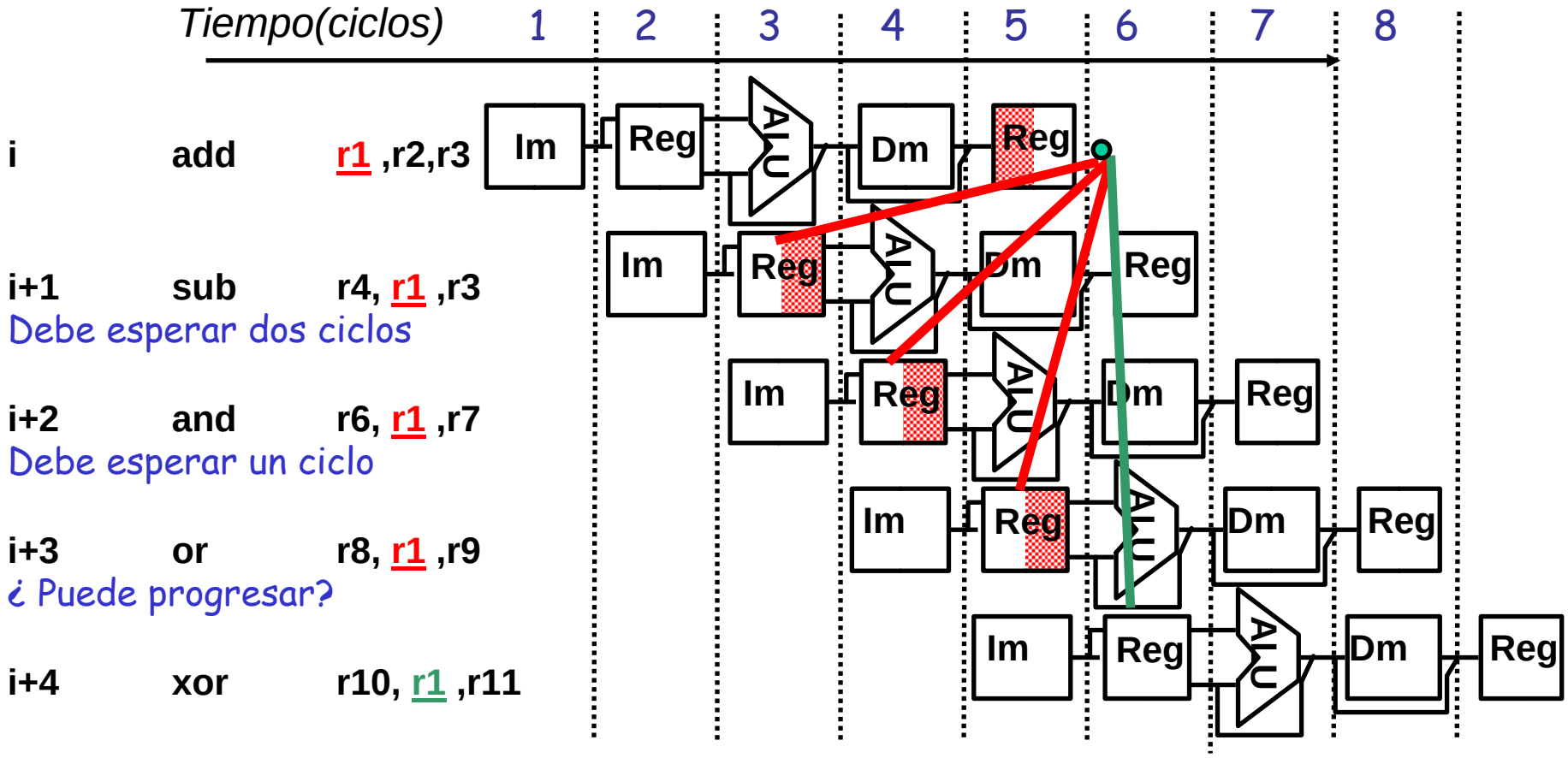
- Riesgos. $D(i) \cap R(i+?) \neq \phi$ EDL (WAR)
  $R(i) \cap R(i+?) \neq \phi$ EDE (WAW)

  - 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



Riesgos de datos

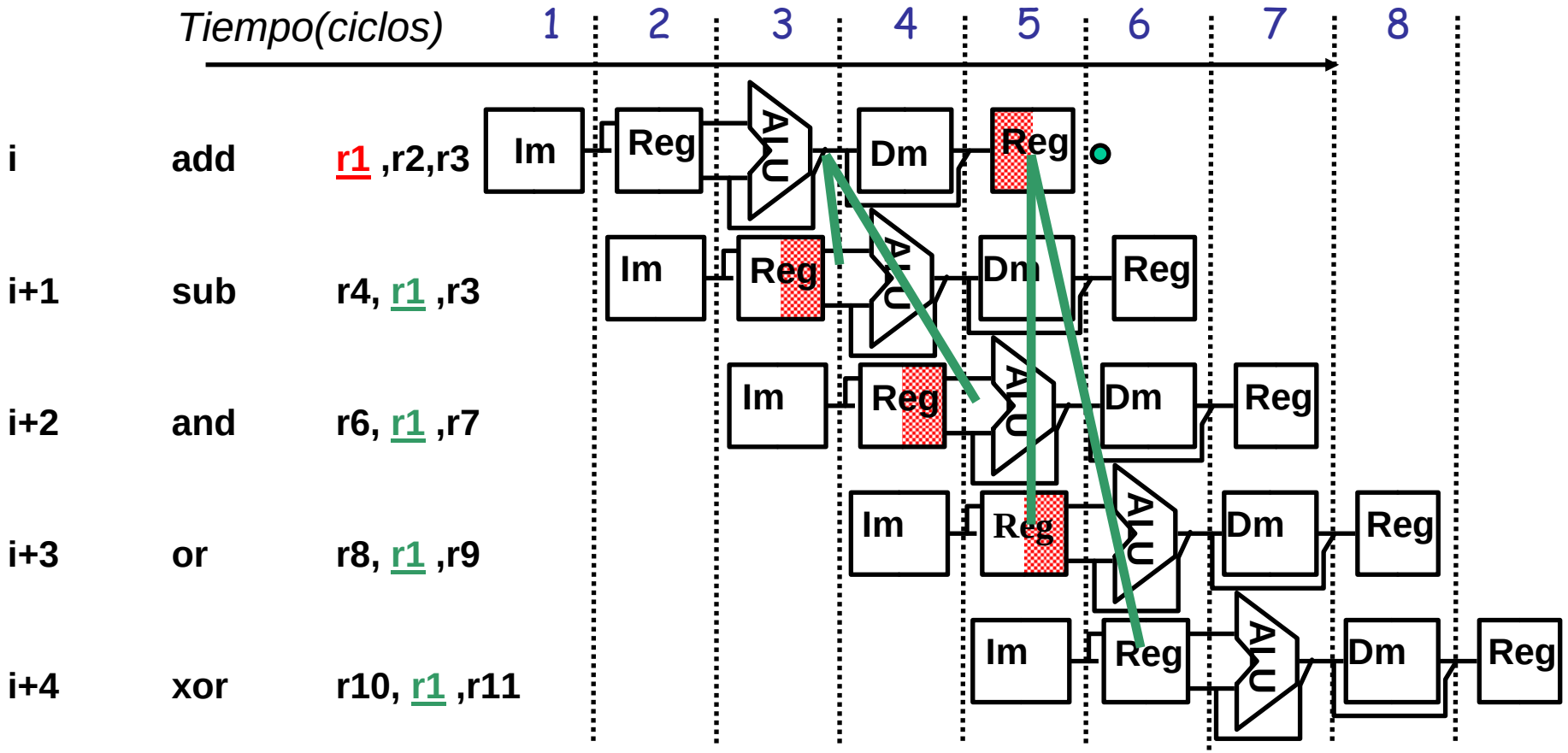
- Riesgo $R(i) \cap D(i+?) \neq \phi$ LDE (RAW)



¿ Cuando esta listo el operando ?

Riesgos de datos

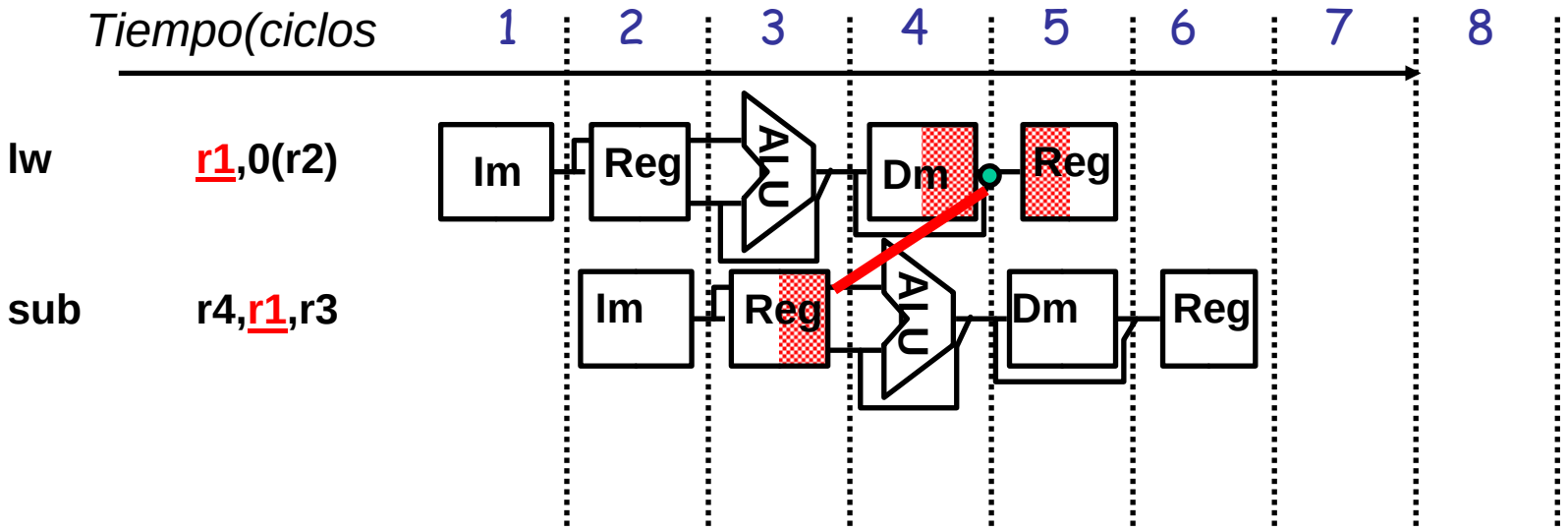
- Riesgo $R(i) \cap D(i+?) \neq \phi$ LDE (RAW).
  - ✓ Cortocircuito. Enviar el dato a las etapas que lo necesitan, cuanto esta listo



Se pueden ejecutar todas las instrucciones sin problemas

Riesgos de datos

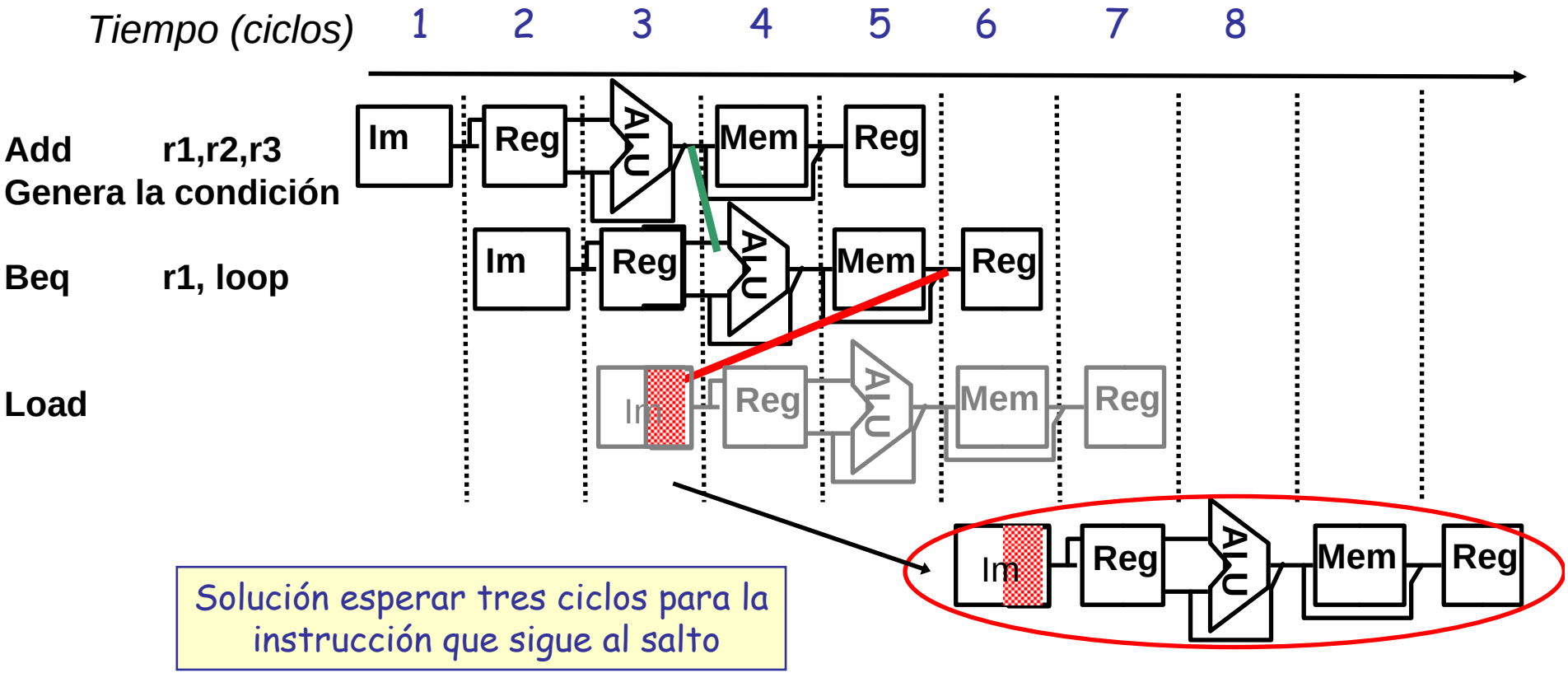
- Riesgo $R(i) \cap D(i+?) \neq \phi$ LDE (RAW).
  - ✓ Caso del Load



No se puede resolver con el cortocircuito
La instrucción dependiente del Load debe esperar un ciclo

Riesgos de control

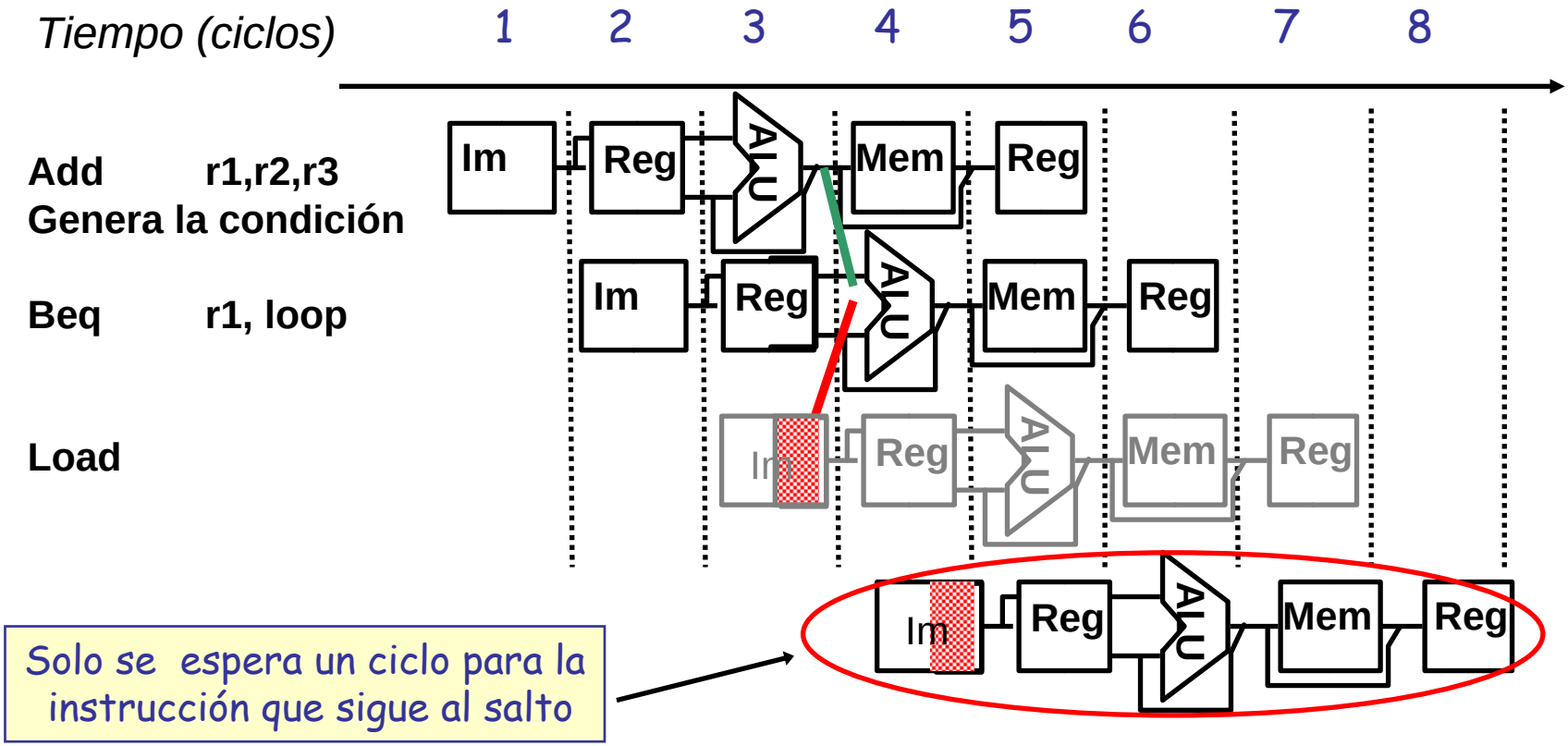
- Se debe evaluar la condición y obtener el nuevo valor del PC
  - ✓ La evaluación de la condición y el calculo del nuevo PC se realizan en la etapa de ejecución
  - ✓ Si el salto se toma, el destino del salto se carga en PC en la etapa Mem



Solución esperar tres ciclos para la instrucción que sigue al salto

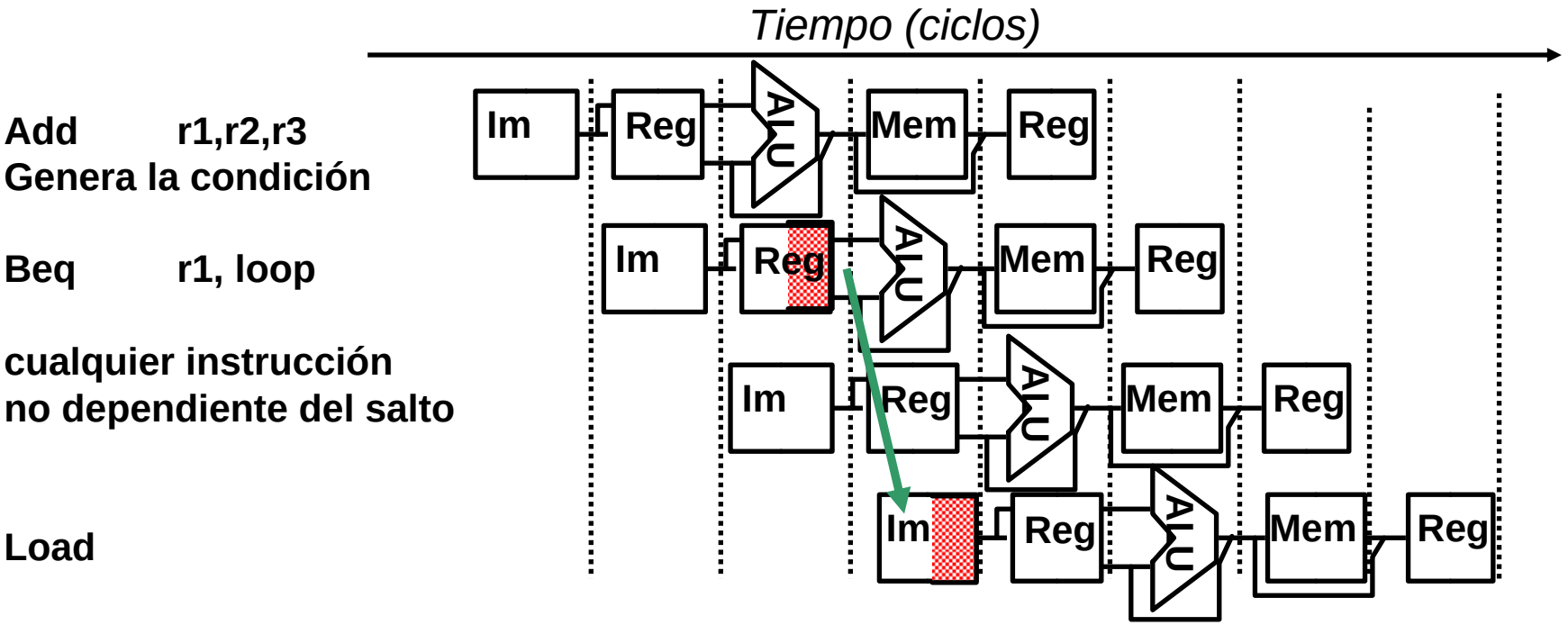
Riesgos de control

- Solución: Desplazar el calculo de la dirección y la evaluación de la condición a la etapa ID



Riesgos de control

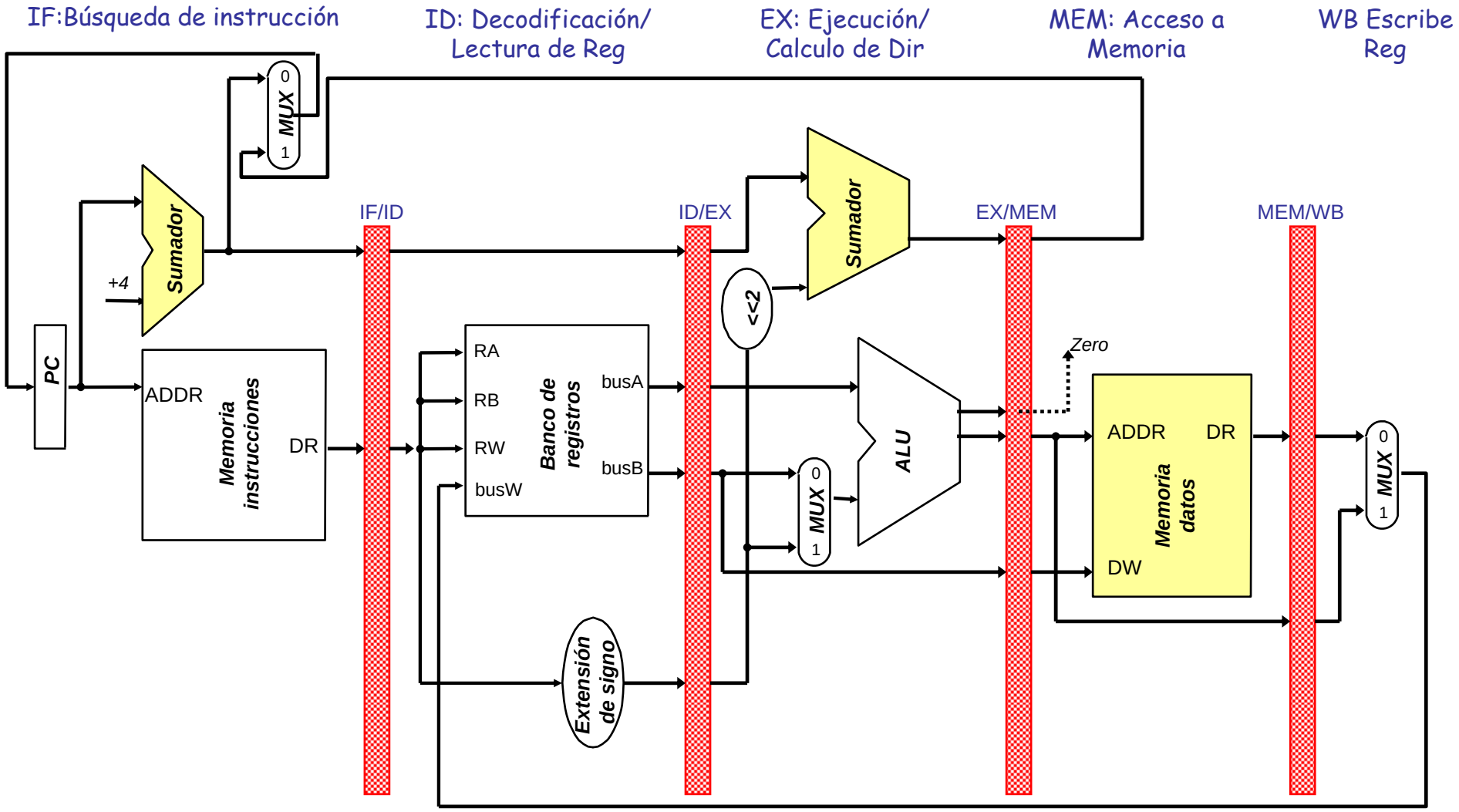
- Reinterpretar el comportamiento de los saltos: Salto retardado
  - ✓ Se efectúa después de la siguiente instrucción
  - ✓ La instrucción siguiente al salto se ejecuta siempre



Si es posible encontrar una instrucción ⇒ 0 ciclos de penalización

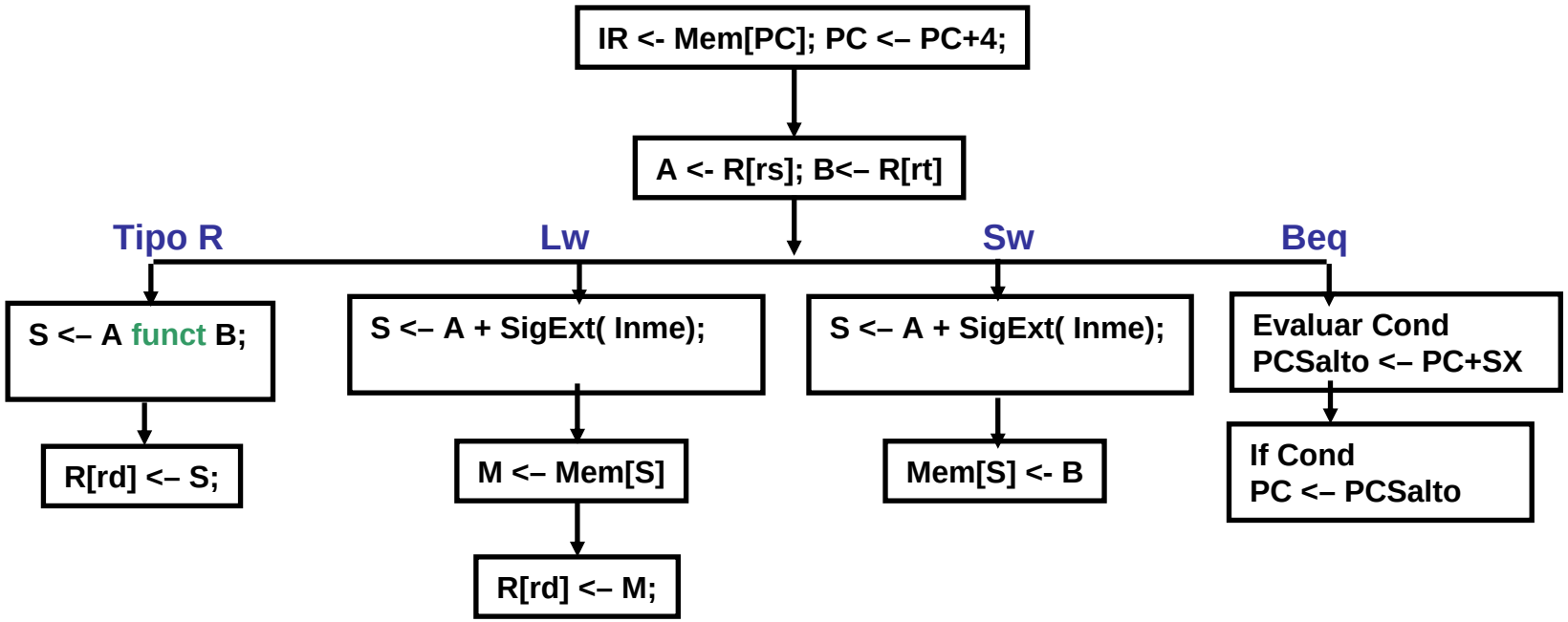
Segmentación del procesador

❑ Nueva Ruta de datos del DLX para ejecución segmentada



Diseño del control

❑ Ejecución de instrucciones: diagrama RT

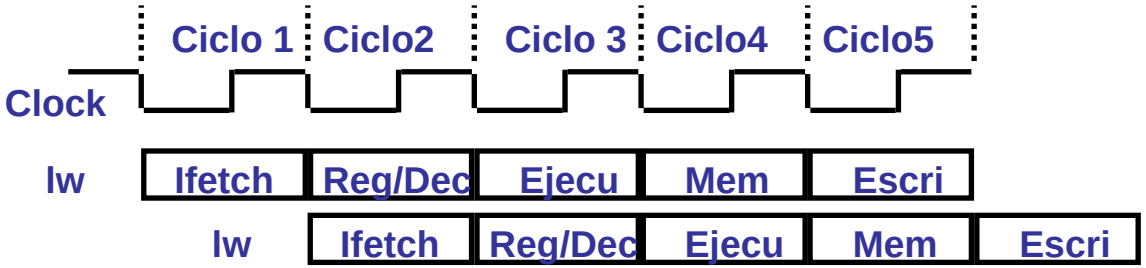


- ❑ Recordatorio
  - ✓ rs = Ins[25-21]
  - ✓ rt = Ins[20-16]
  - ✓ rd = Ins[15-11]
  - ✓ Inme = Ins[15-0]

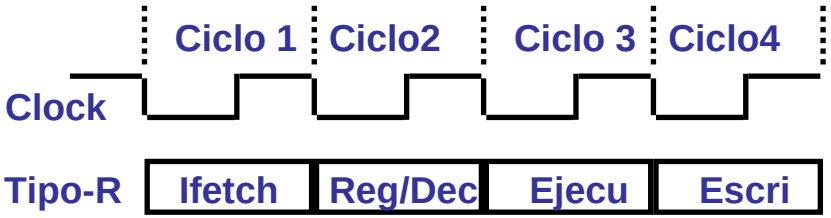
Diseño del control

❑ Las instruciones: Load (lw) y tipo-R

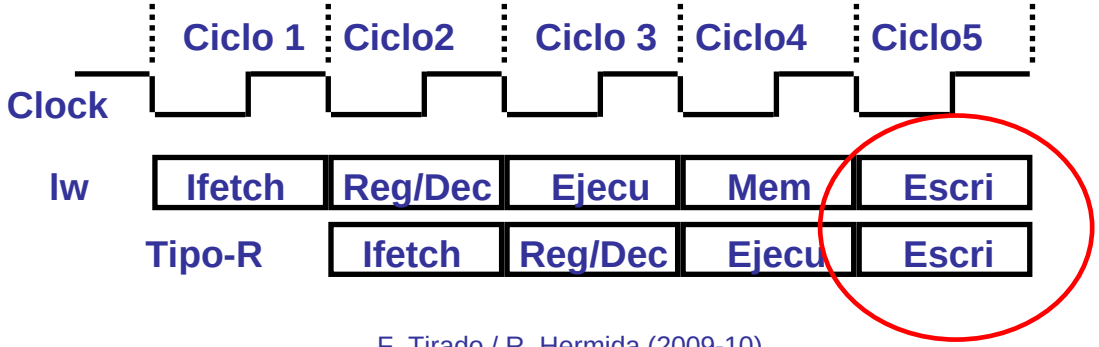
✓ Load. Tiene 5 fases y cada una utiliza una de las etapas



✓ Tipo-R. Tiene 4 fases y no usa la memoria de datos



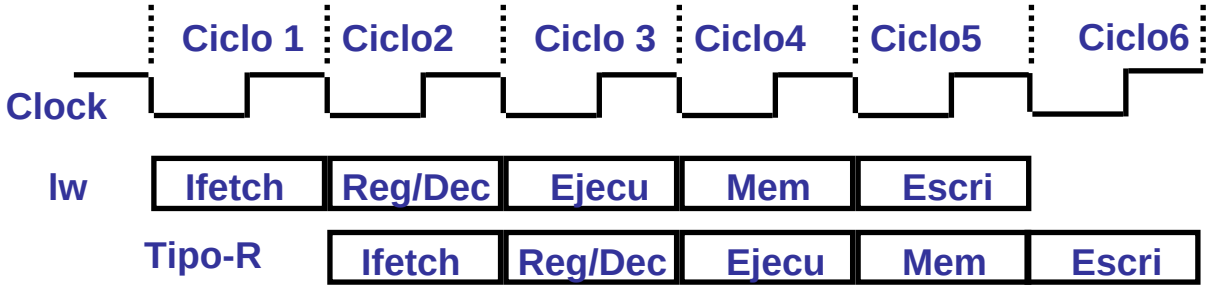
✓ Conflicto



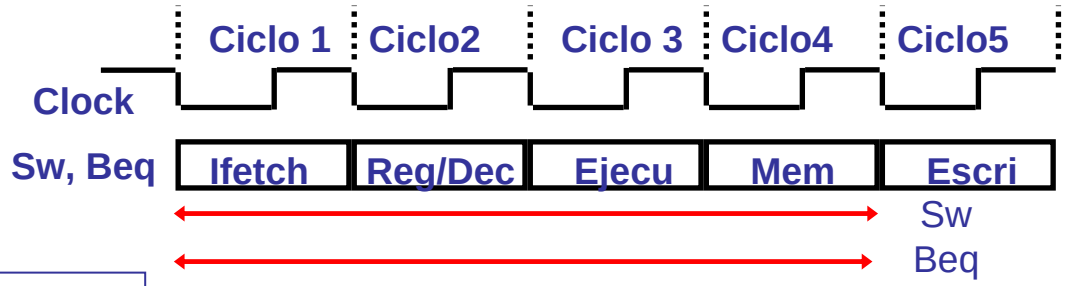
Solo una puerta en el bloque de registros

Diseño del control

- Importante. ¿ como evitarlo ?
  - ✓ Cada etapa del procesador puede usarse en cada ciclo por una sola instrucción.
  - ✓ Cada instrucción debe usar cada etapa del procesador en la misma fase de ejecución.
- Solución.
  - ✓ Introducir una etapa (Mem) que no hace nada.

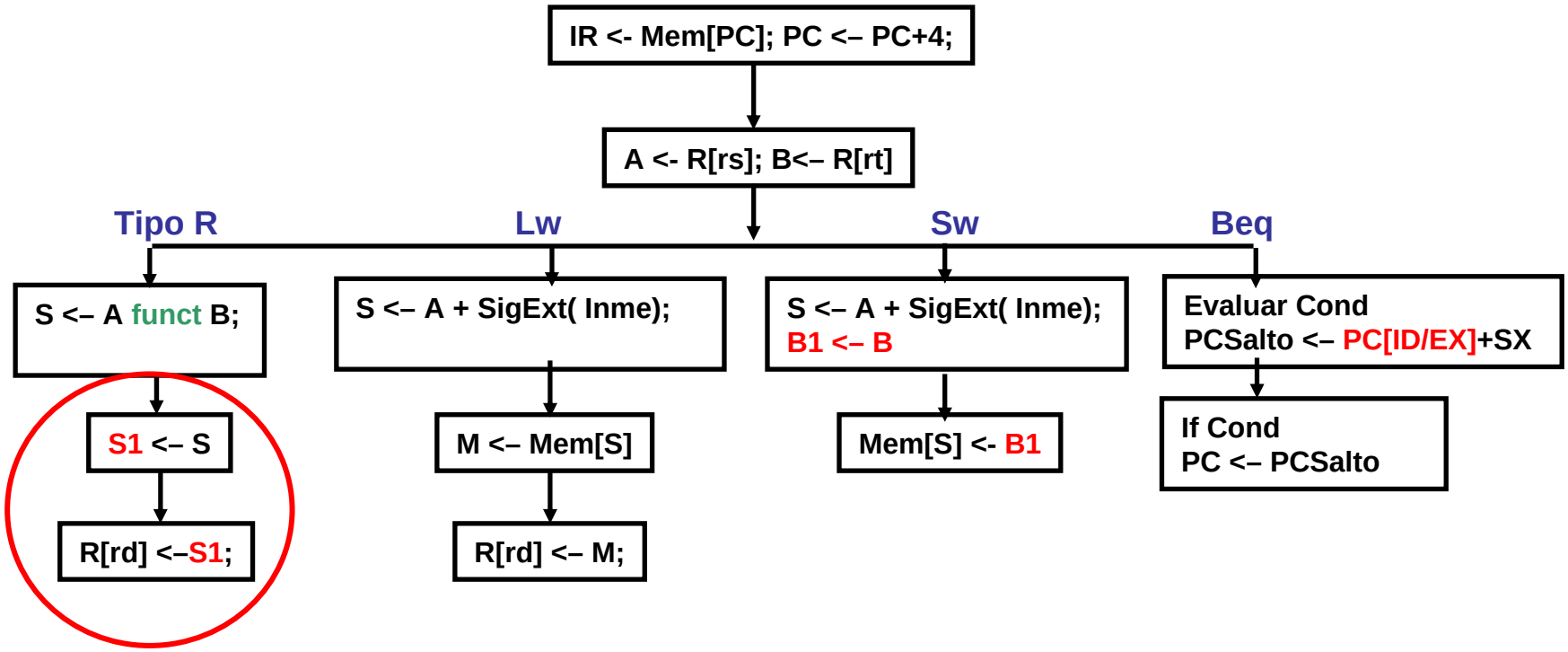


- Store 4 etapas con actividad. Branch cuatro etapas activas



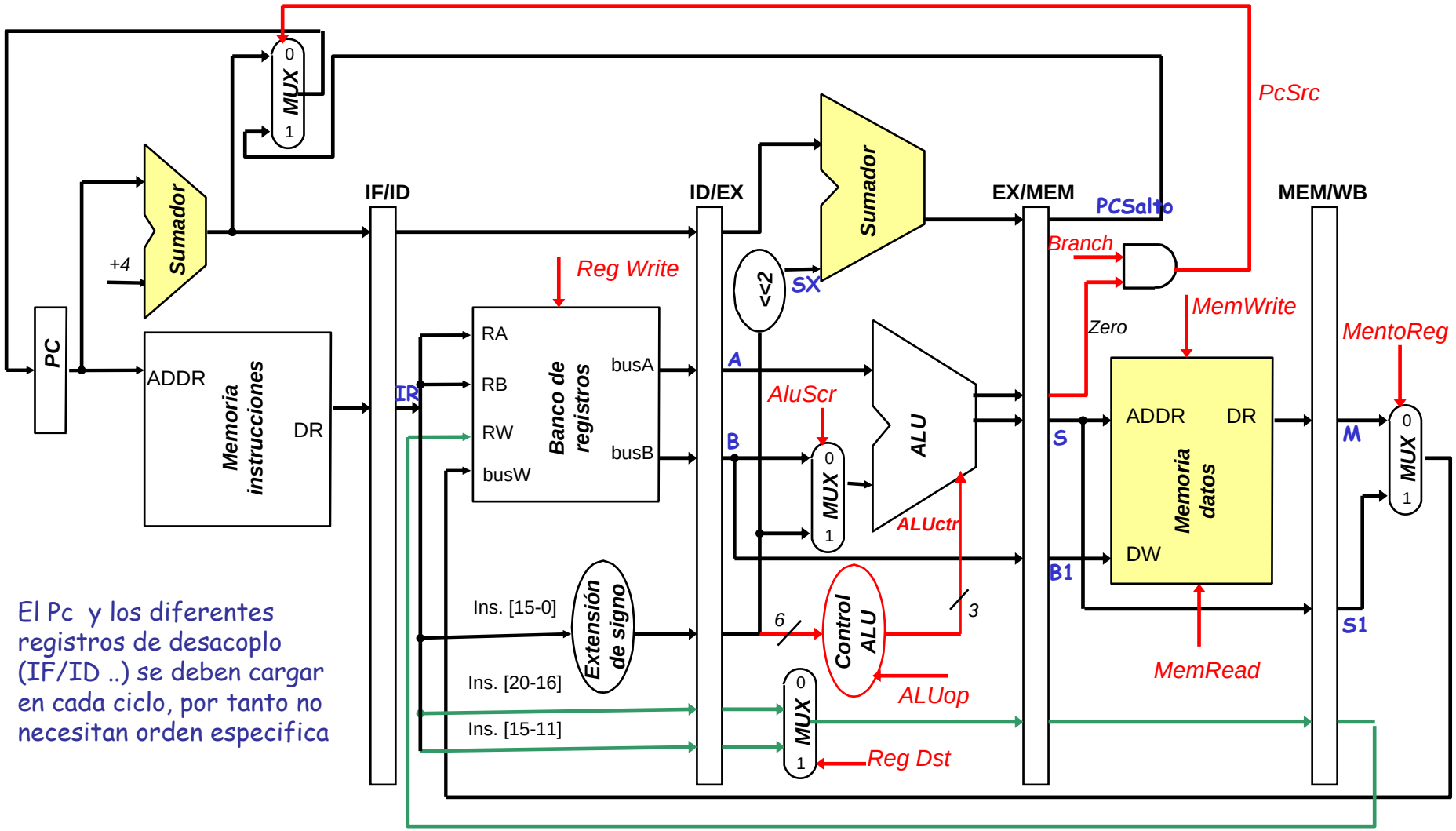
Diseño del control

❑ Ejecución de instrucciones: Diagrama RT modificado



Diseño del control

- Ruta de datos del DLX con las ordenes de control necesarias



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

Diseño del control

❑ Señales de control

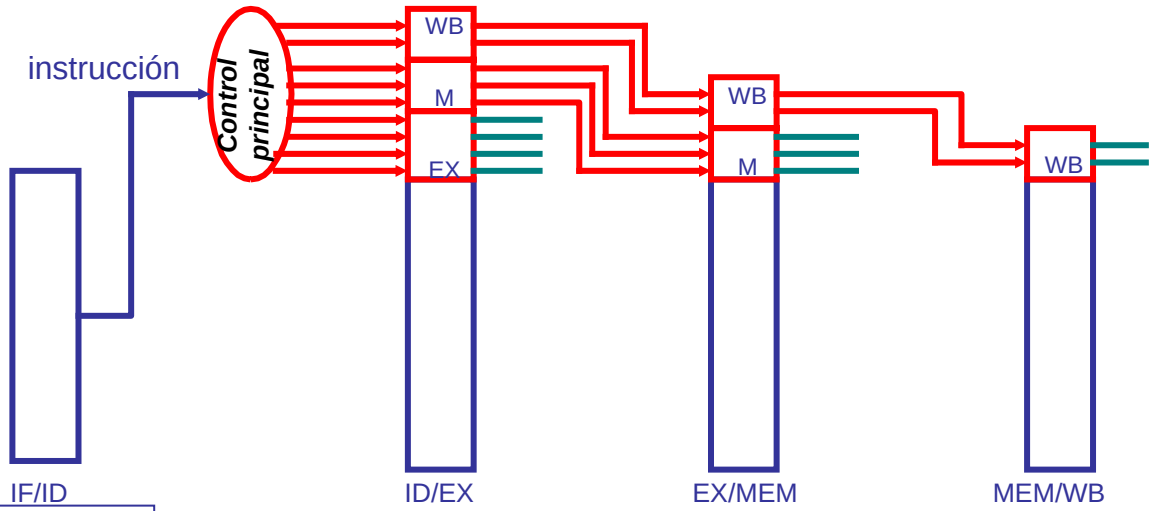
Control de la ALU

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

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

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



EX: Ejecución/ Calculo de Dir

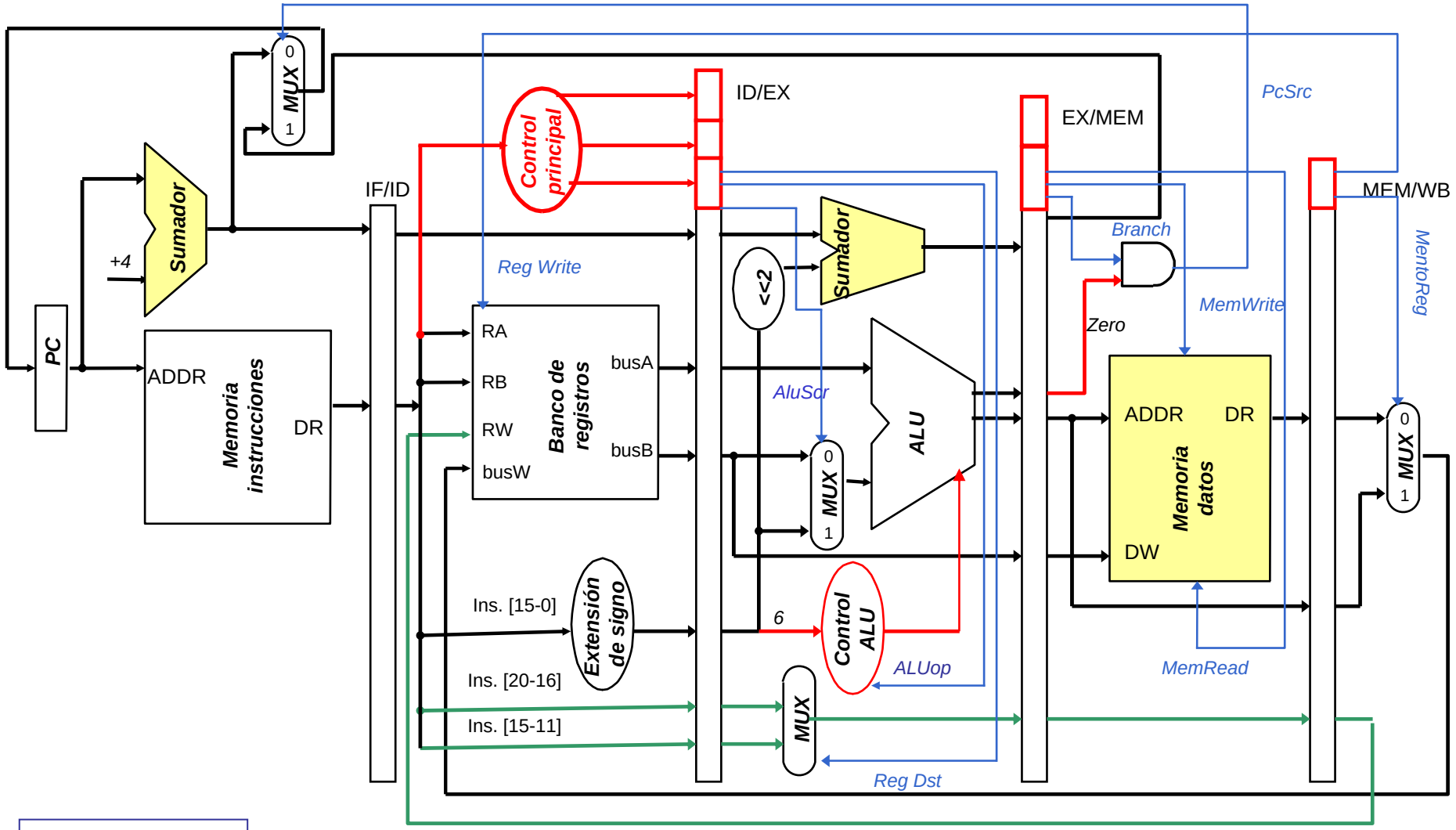
MEM: Acceso a Memoria

WB Escribe Reg

Control segmentado estacionario en los datos

Diseño del control

- Ruta de datos del DLX con las ordenes de control y control estacionario en los datos



Diseño del control

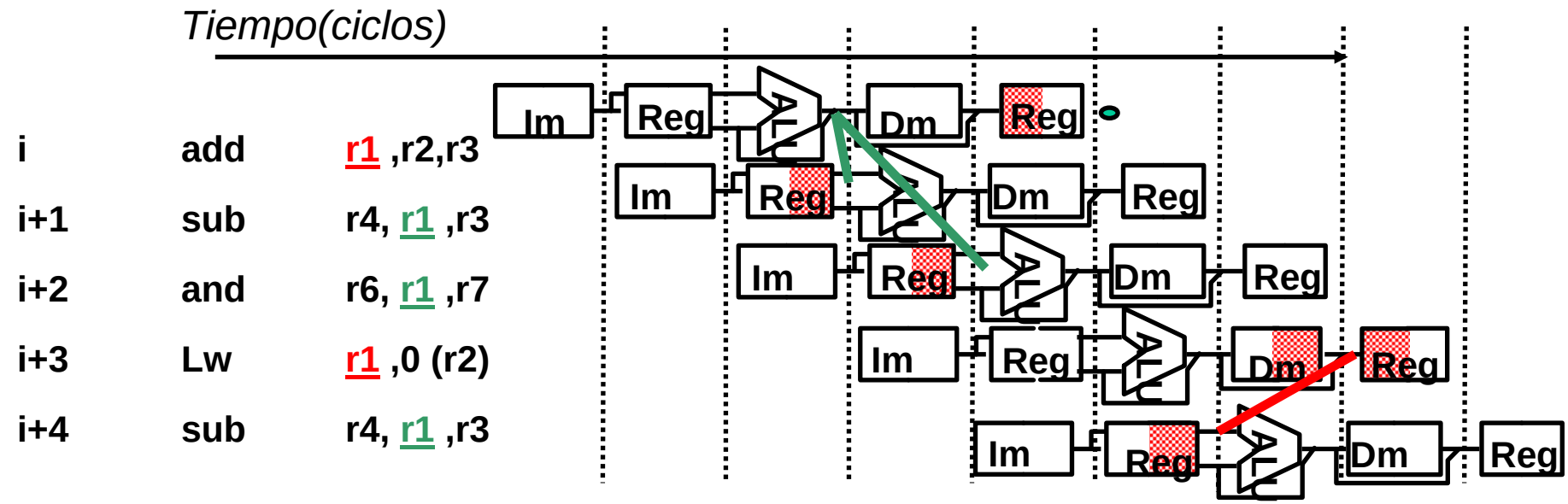
- ¿Qué facilita el control segmentado?
  - ✓ Todas las instrucciones con igual duración
  - ✓ Pocos formatos diferente de instrucción
  - ✓ Accesos a memoria solo en load y stores

- ¿Qué dificulta el control segmentado?
  - ✓ Riesgos estructurales. Conflictos de recursos
  - ✓ Riesgos de datos. Solo LDE
  - ✓ Riesgos de control

- El diseño anterior no tenia en cuenta los riesgos de datos y los riesgos de control los eliminaba con saltos retardados
  - ✓ Implementación del cortocircuito
  - ✓ Caso del load
  - ✓ Mejora en el comportamiento de los saltos.

Diseño del control con riesgos

- Riesgos de datos $R(i) \cap D(i+?) \neq \phi$ LDE (RAW).
  - ✓ Cortocircuito. Enviar el dato a las etapas que lo necesitan, cuanto esta listo

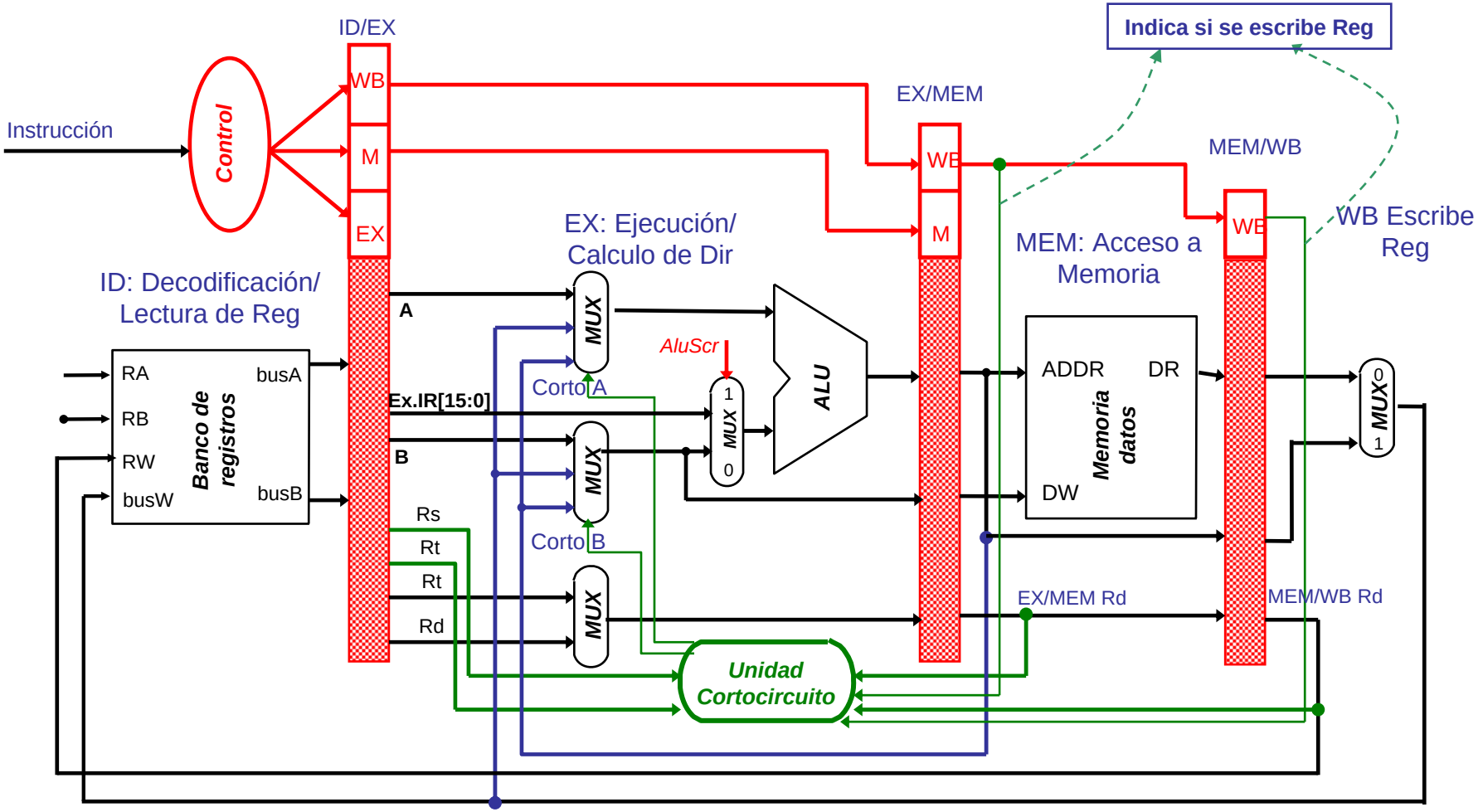


Dos caminos de datos: Desde salida ALU (EX/MEM) a entrada ALU
Desde la salida de la memoria (MEM/WB) a entrada ALU
Información necesaria: Registro a escribir en ultima etapa (Rd en Tipo-R y Rt en Lw)
Registros que se leen en segunda etapa (Rs y Rt)

Diseñando el control con riesgos

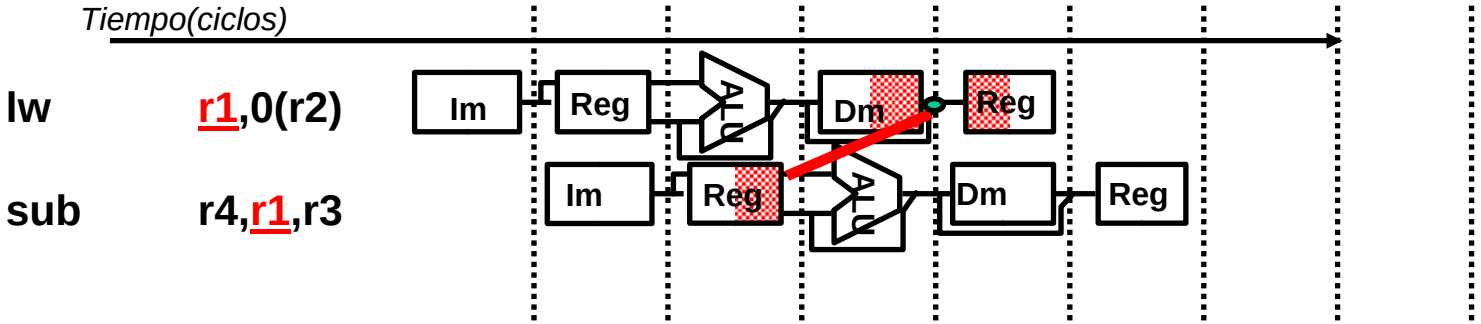
❑ Riesgos de datos LDE: Implementación del cortocircuito

- ✓ Cortocircuitos de datos
- ✓ Información de control del cortocircuito

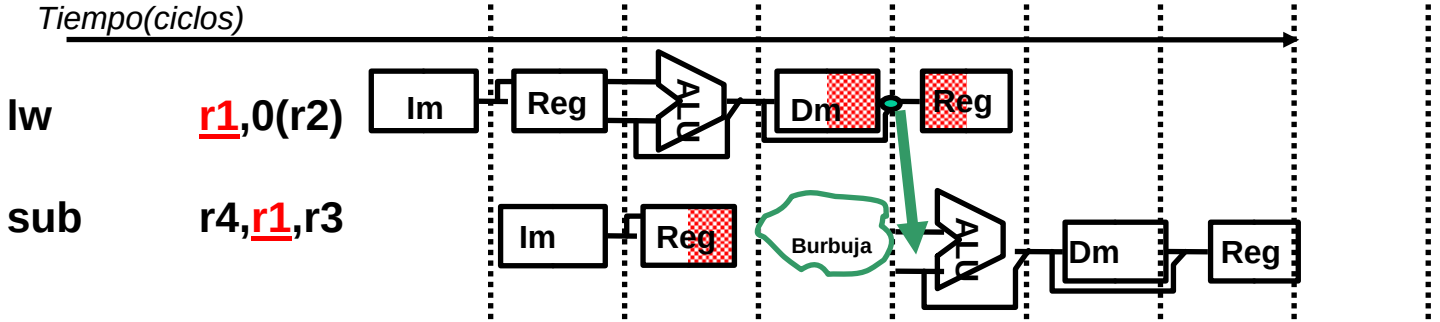


Diseño del control con riesgos

- Riesgos de datos LDE: Caso del load



Se debe esperar un ciclo a pesar del cortocircuito



Diseño del control con riesgos

❑ Riesgos de datos LDE: Caso del load

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

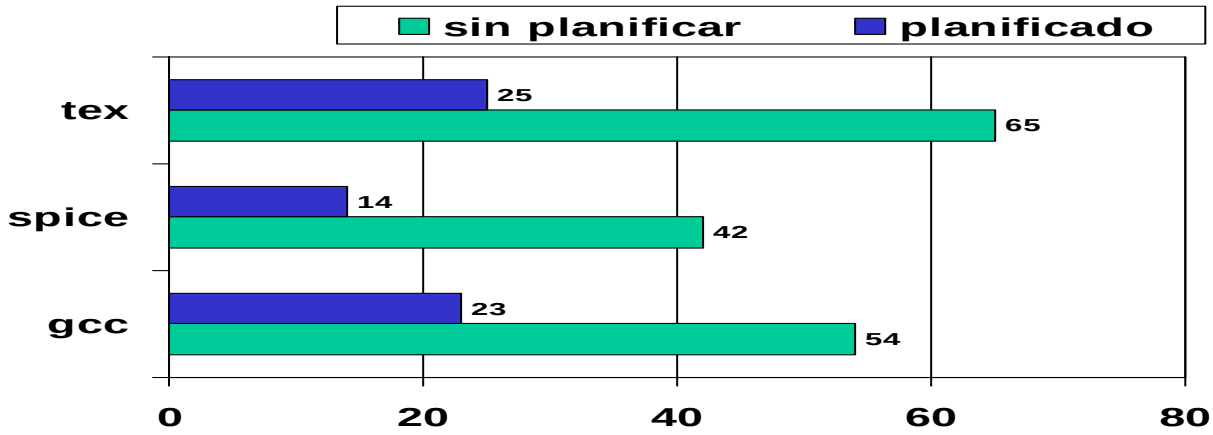
a := b + c
d := e - f

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
```



Diseño del control con riesgos

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

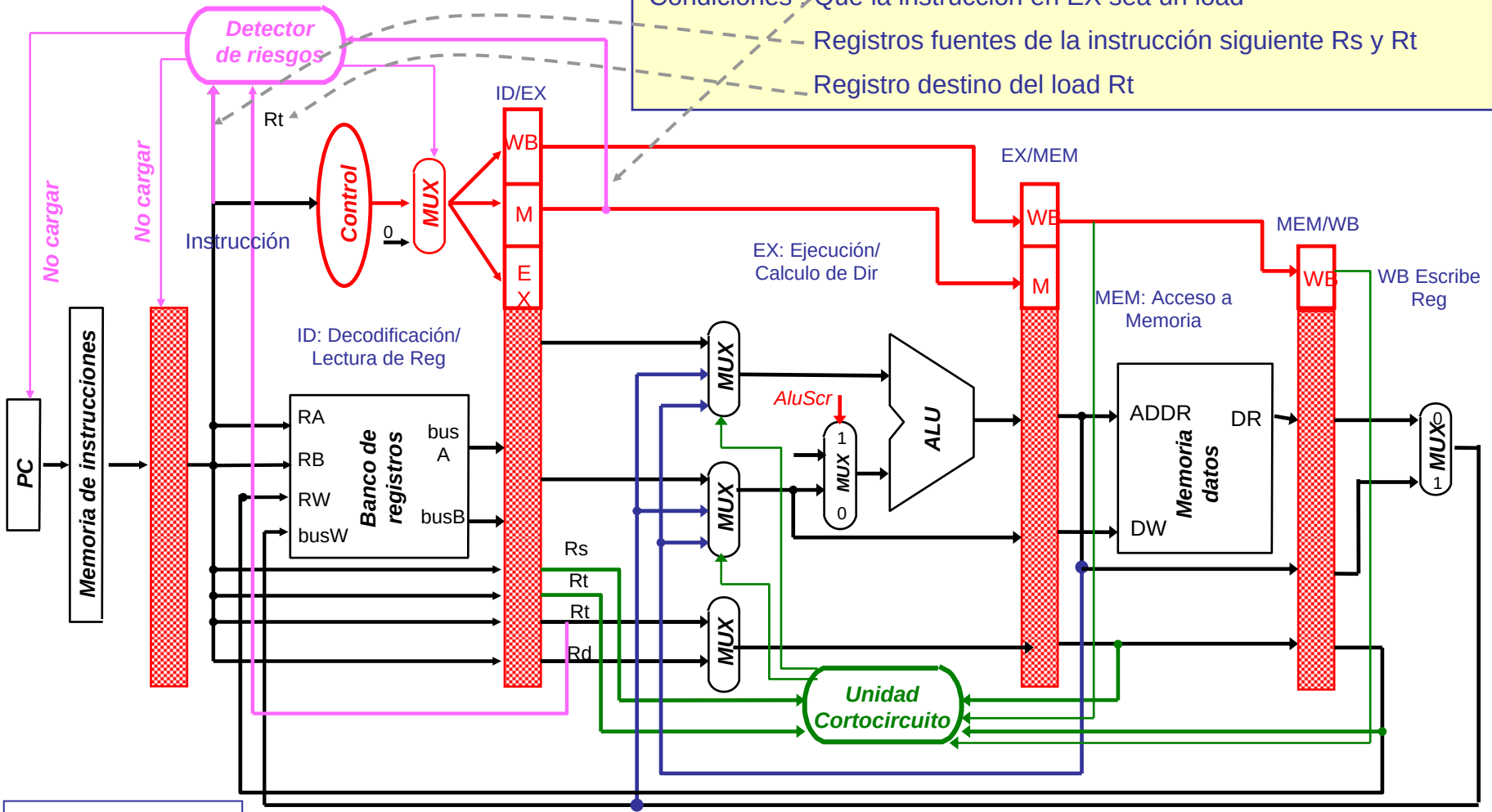
Detector de riesgos

Debe actuar en ID impidiendo la carga y lectura de una nueva instrucción

Condiciones : Que la instrucción en EX sea un load

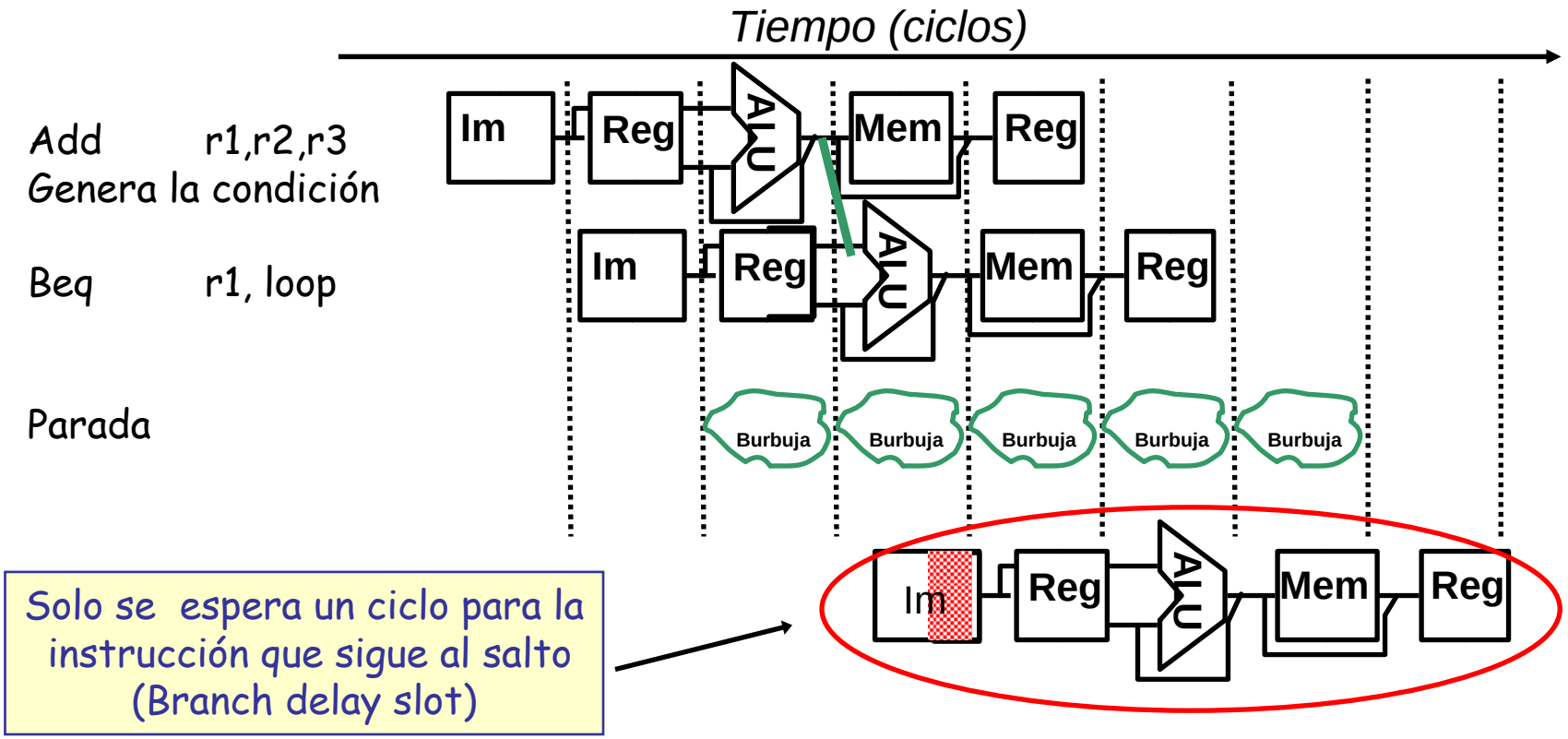
Registros fuentes de la instrucción siguiente Rs y Rt

Registro destino del load Rt



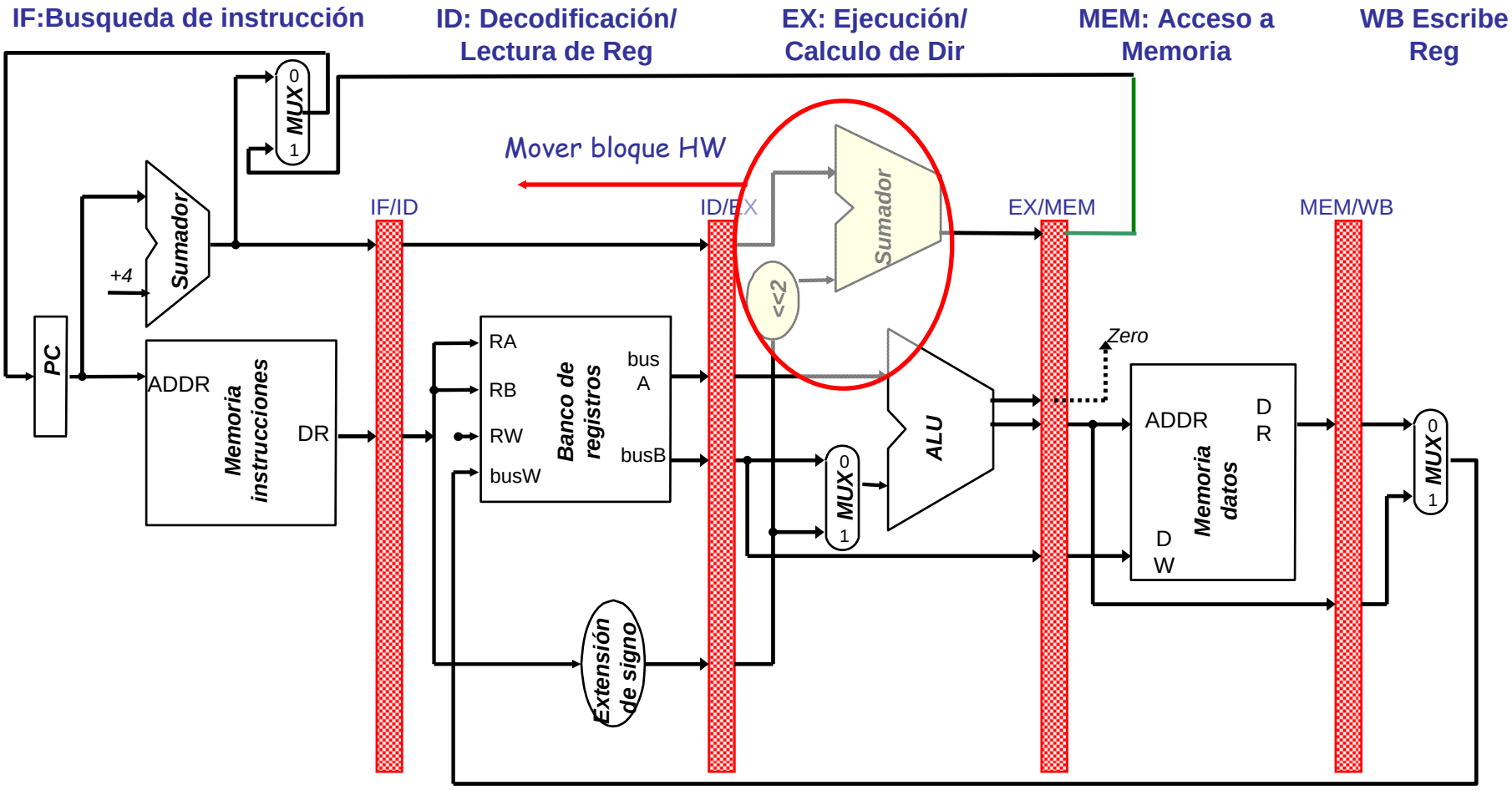
Diseño del control con riesgos

- Riesgos de control
  - ✓ Solución: Desplazar el calculo de la dirección y la evaluación de la condición a la etapa anterior



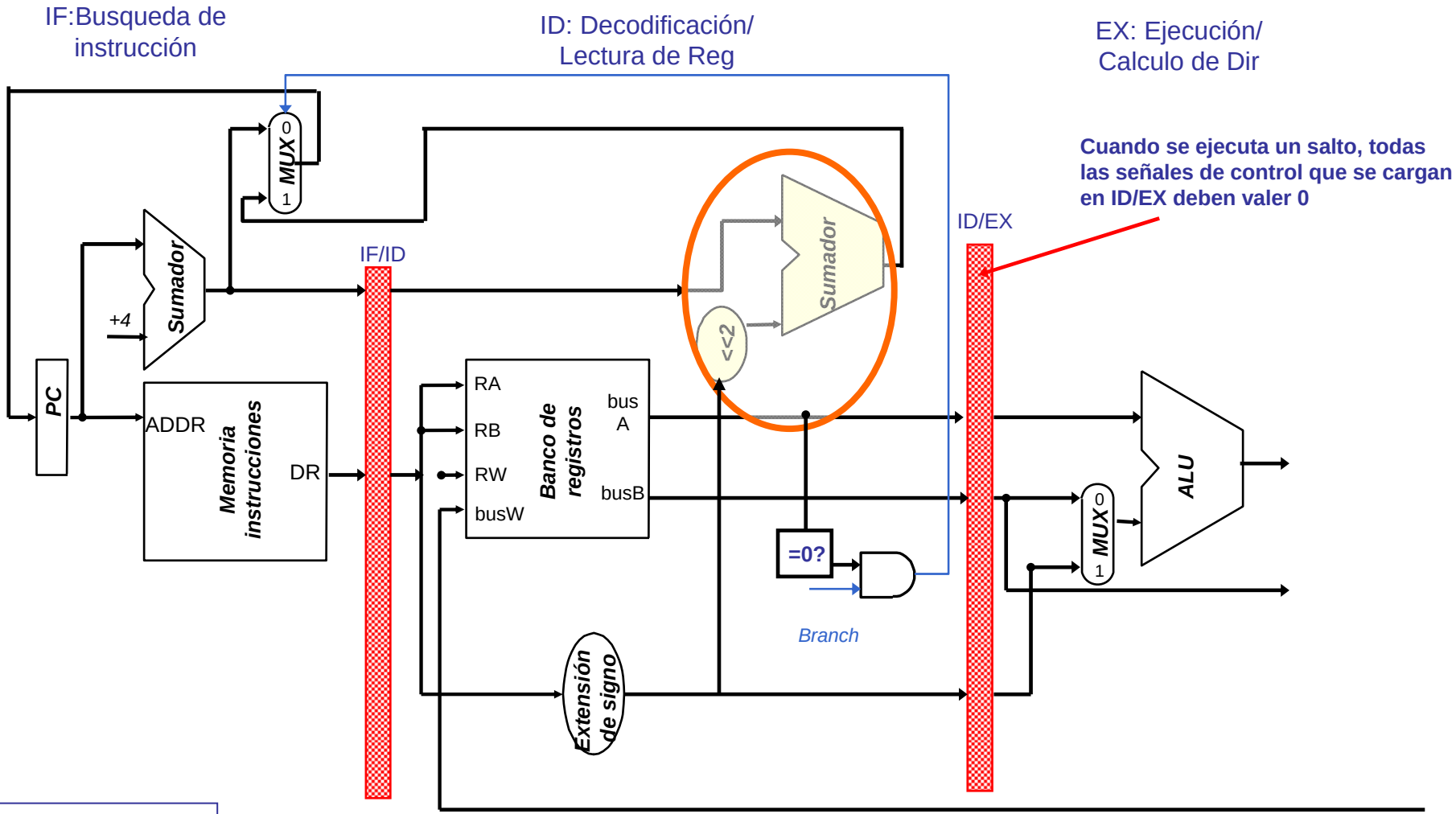
Diseño del control con riesgos

- Mejora del comportamiento de los saltos. Solo un ciclo de parada
  - ✓ Calculo de la dirección. Operandos disponibles (Pc y desplazamiento)
  - ✓ Calculo de la condición. Unidad de detección de cero



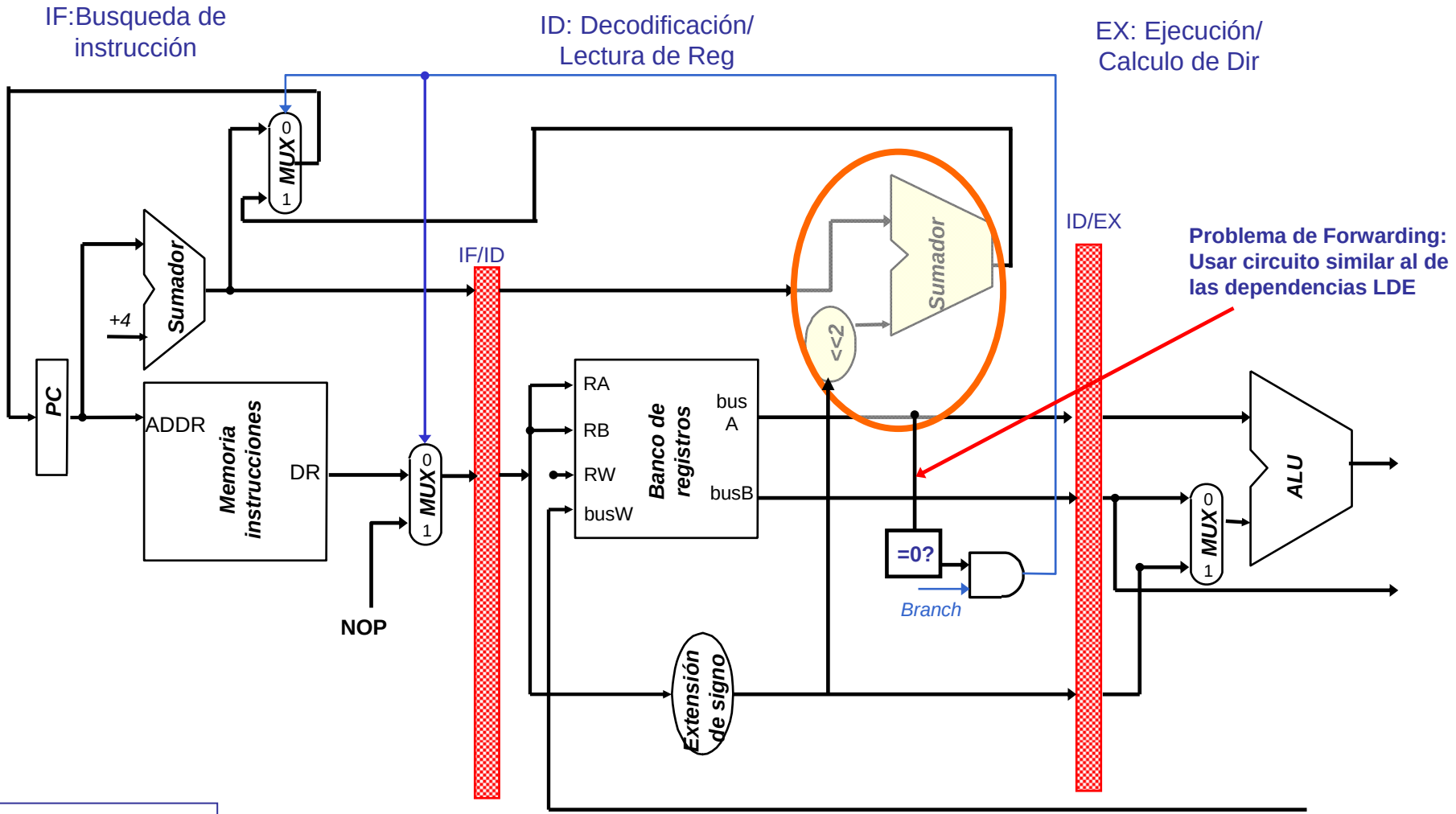
Saltos retardados

- Delay slot = 1 ciclo de reloj. La instrucción siguiente al salto se ejecuta siempre



Predicción estática

- Predecir que el salto no se toma: Si la predicción es falsa, la instrucción siguiente se aborta (se cambia por NOP)



Pérdida de rendimiento

- EL CPI ideal es 1
- Hay perdidas de rendimiento por las paradas del pipe

$$CPI_{real} = CPI_{ideal} + \text{Penaliz. media por instrucción} = 1 + \sum_{i=1}^{\#tipos\ de\ instr} Penaliz_i \times Frec_i$$

- Caso de los saltos. Un programa tipico 30% de saltos
  - CPI = 1 + (1 x 0.3) = 1.3

$$\textbf{Speedup} = \frac{\text{Nº Instrucciones x nº de etapas}}{\text{Nº instrucciones x CPI}} = \frac{5}{1.3} = 3.84$$

  - Se pierde un 24 % respecto al caso ideal:
    $Speedup_{real} / Speedup_{ideal} = 3.84 / 5 = 0.76$

Excepciones

- Interrupciones, Excepciones, Fallos
  - **Síncronas – asincronas**
  - **Solicitadas por el programa - generadas por el programa**
  - **Dentro de instrucciones - entre instrucciones**
  - **Continuar – terminar**

- Problema: El solapamiento en la ejecución de las instrucciones dificulta el saber si una instrucción puede cambiar el estado de la maquina sin peligro
- Cualquier instrucción en el pipeline puede provocar una excepción
- El sistema debe resolver la excepción y recomenzar la ejecución. El sistema debe recuperar el estado previo a la excepción

- Excepciones problemáticas: (un ejemplo fallo de página)
  - **Ocurren en el medio de una instrucción**
  - **Deben ser recomenzables**

- Interrupciones externas (I/O)
  - **Vaciar el pipeline y entrar no operaciones (NOPs)**
  - **Almacenar el PC con la dirección de interrupción**

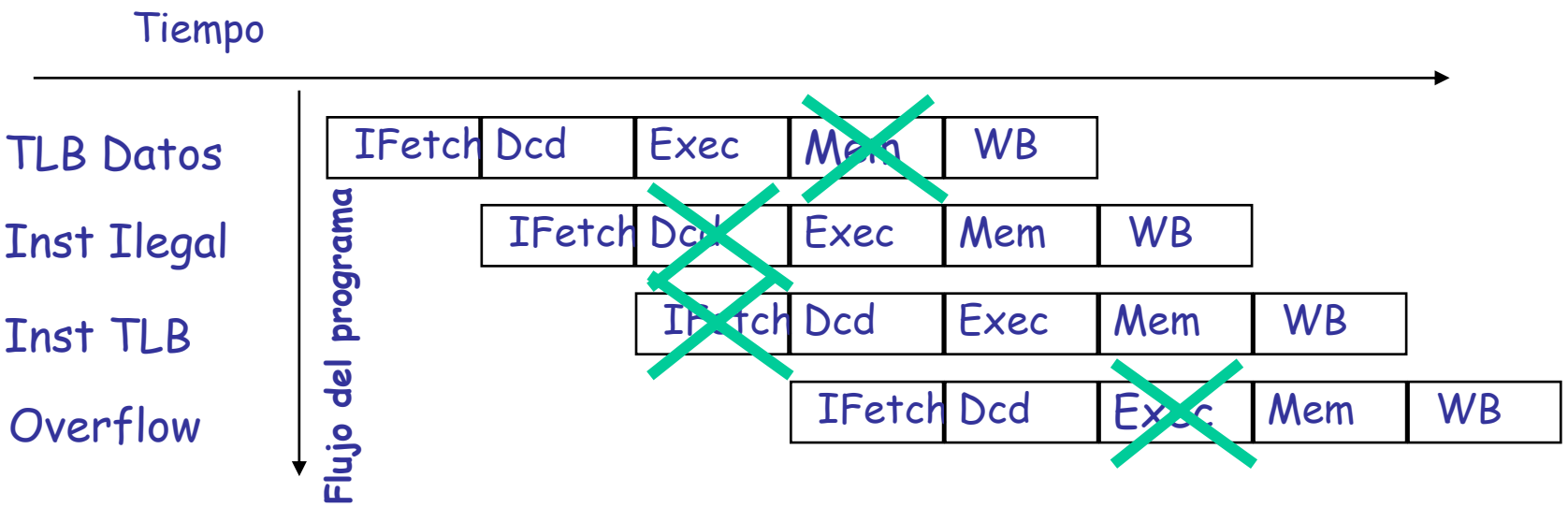
Excepciones

- Excepciones en el DLX

IF	Fallo de pagina de instrucción; Acceso no alineado; Violación de protección
ID	Instrucción ilegal
Ex	Excepción aritmética
MEM	Fallo de pagina de datos; Acceso no alineado; Violación de protección
WB	Ninguna

- Excepciones (fallo de pagina). Reinicio
  - Introducir una instrucción de trap en IF
  - Anular las escrituras de la instrucción que produce la excepción y las posteriores
  - Salvar el PC(se debe conservar hasta MEM) de la instrucción que produjo la excepción y pasar a la rutina de gestión
- Excepciones precisas:
  - Todas las instrucciones anteriores completas
  - La que produce la interrupción y las siguientes como si no hubieran empezado

Excepciones: aparición fuera de orden



- Muchas excepciones simultaneas
- Fallo de pagina en MEM y otras excepciones en el resto. Se atiende el fallo de pagina y se recomienza la siguiente instrucción.
- Problema:
  Fallo de pagina en MEM , instrucción ilegal en reg./De y fallo de pagina en IF.
  ¡La excepción de la segunda y tercera aparece antes !
- Solución:
  Vector de estado, un bit por cada etapa donde es posible excepción
  Cada excepción se marca en un vector de estado asociado con la instrucción, y se impide la escritura. El vector se chequea al final de MEM inicio de WB)
- Excepciones precisas (se atienden por orden)

Saltos retardados: Compilador

- Se puede mejorar el rendimiento introduciendo una instrucción no dependiente del salto en la burbuja

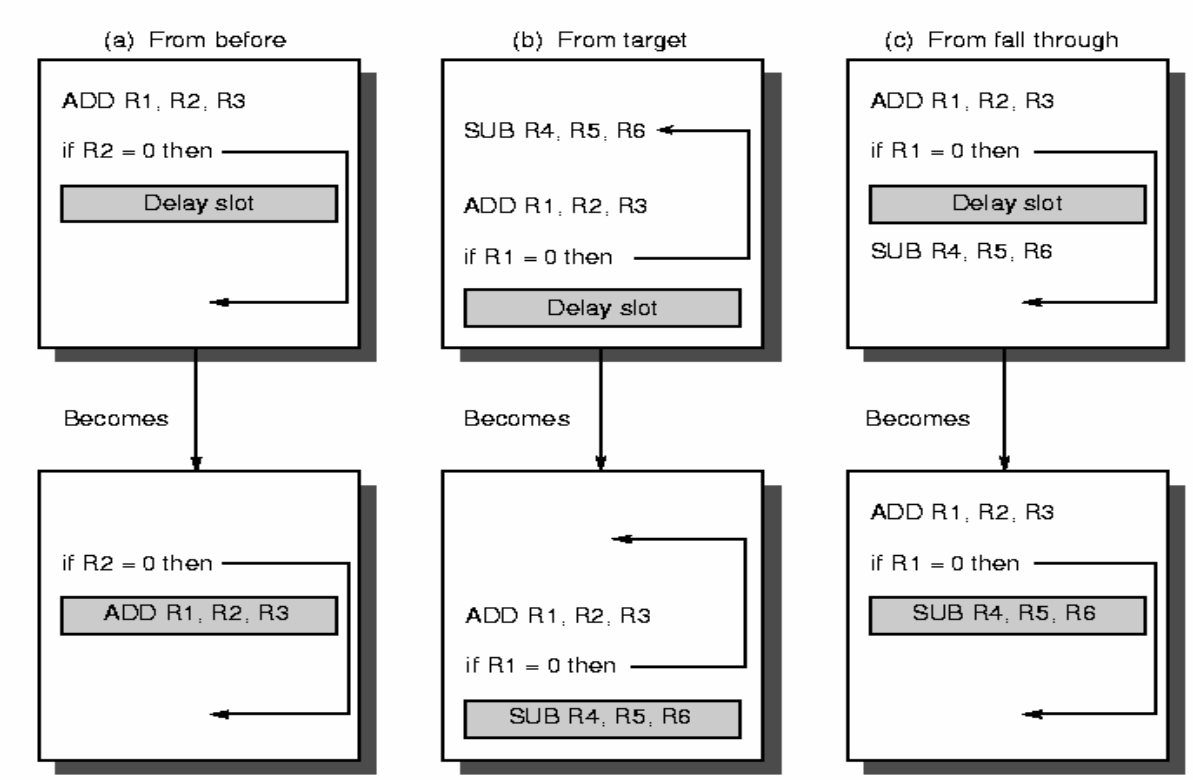
Fundamento de los saltos retardados: Ejecutar instrucciones ***independientes del salto*** durante los ciclos de retardo

Estas instrucciones se ejecutarán ***siempre*** (tanto si el salto es tomado como si no)

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

(a) mejor solución
siempre trabajo útil

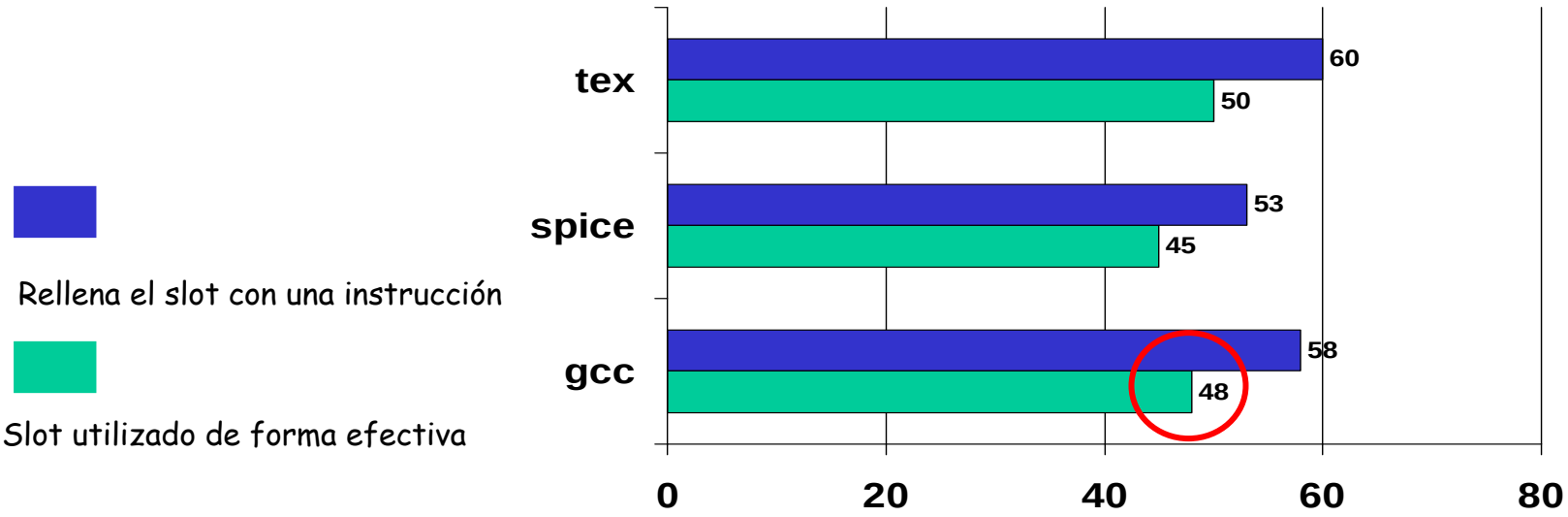
(b) y (c) la instrucción
elegida no debe modificar
la semántica (aunque se ejecute indebidamente).



Saltos retardados: benchmarks

- Tex, Spice, Gcc: En media el 80 % de las posiciones rellenas son útiles (barra verde / barra azul).
  - ✓ El resto son NOP
  - ✓ En Gcc el 22% de las instrucciones son saltos

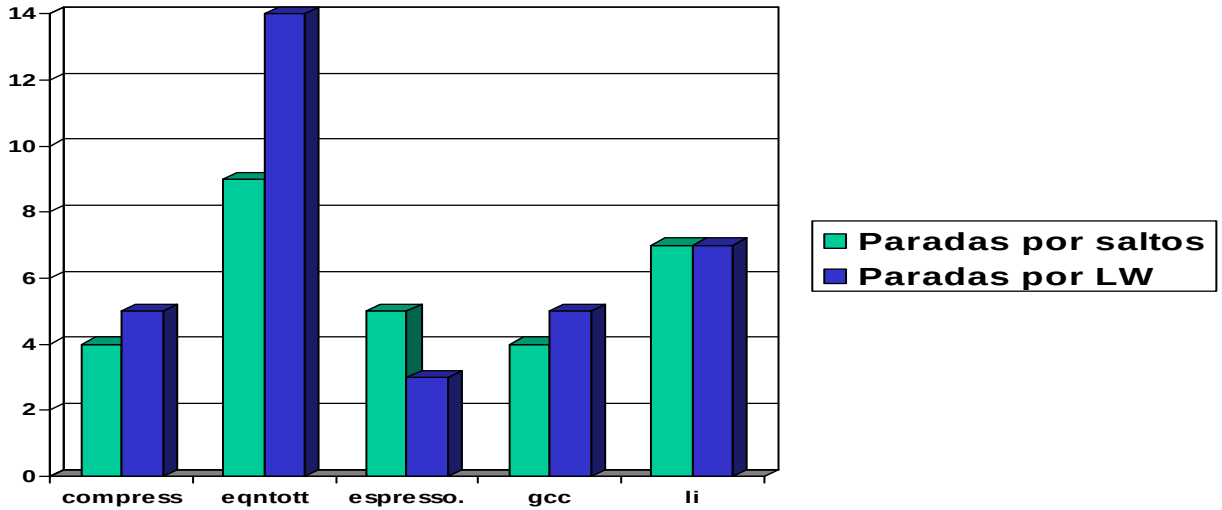
    CPI = 1+ (1-0.48) x 0.22 = 1.11



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
  - ✓ Riesgso 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.

Muy importante:
Las instrucciones empiezan y terminan en orden

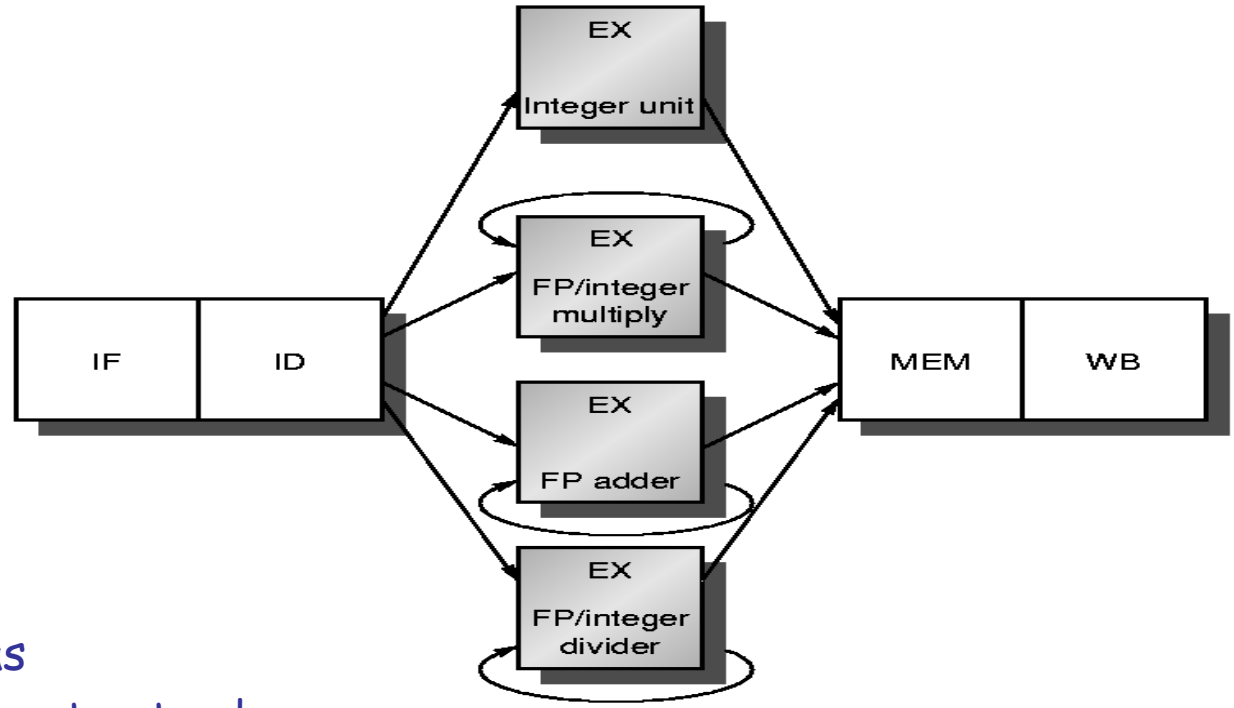


¿ Que ocurre si las instrucciones tienen diferentes duración?
Instrucciones de aritmética en punto flotante

Operaciones multiciclo

- Es el caso típico de las operaciones en punto flotante

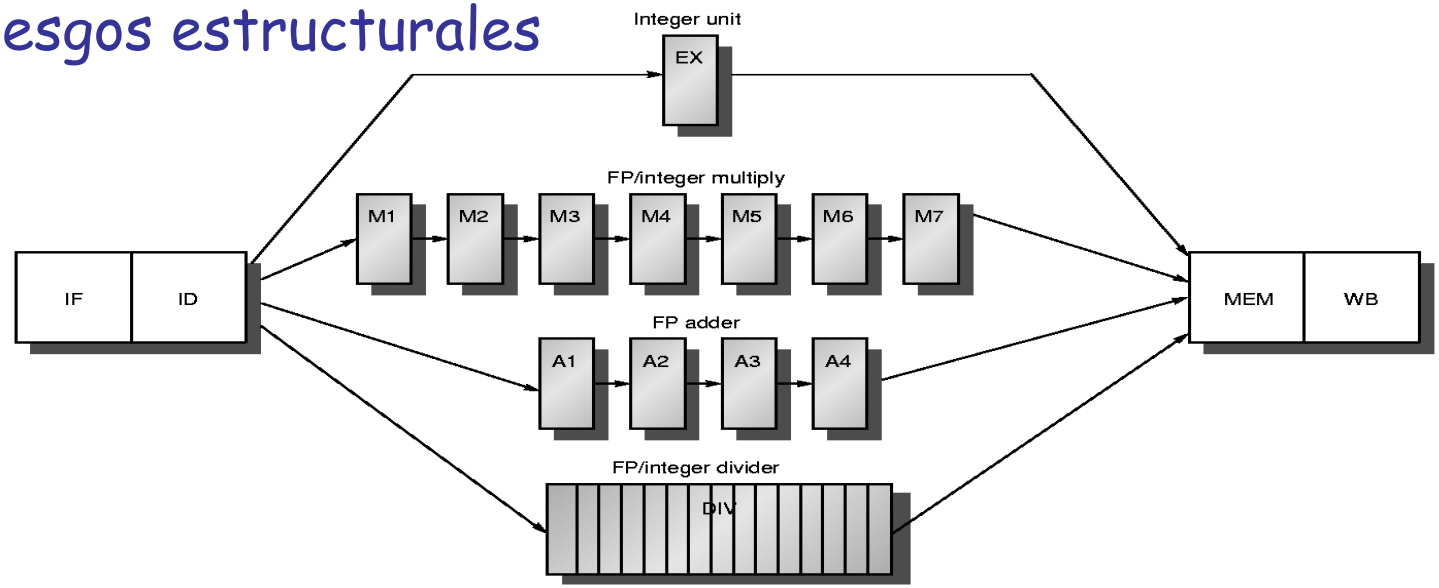
  Estructura de etapas de procesador: Es necesario HW adicional para la fase de ejecución



- Problemas
  - Riesgos estructurales
  - Mayor penalización de los riesgos LDE
  - Problemas con la finalización fuera de orden
- Solapamiento operaciones enteras y PF
  - No hay problemas operandos independientes

Operaciones multiciclo

- Riesgos estructurales

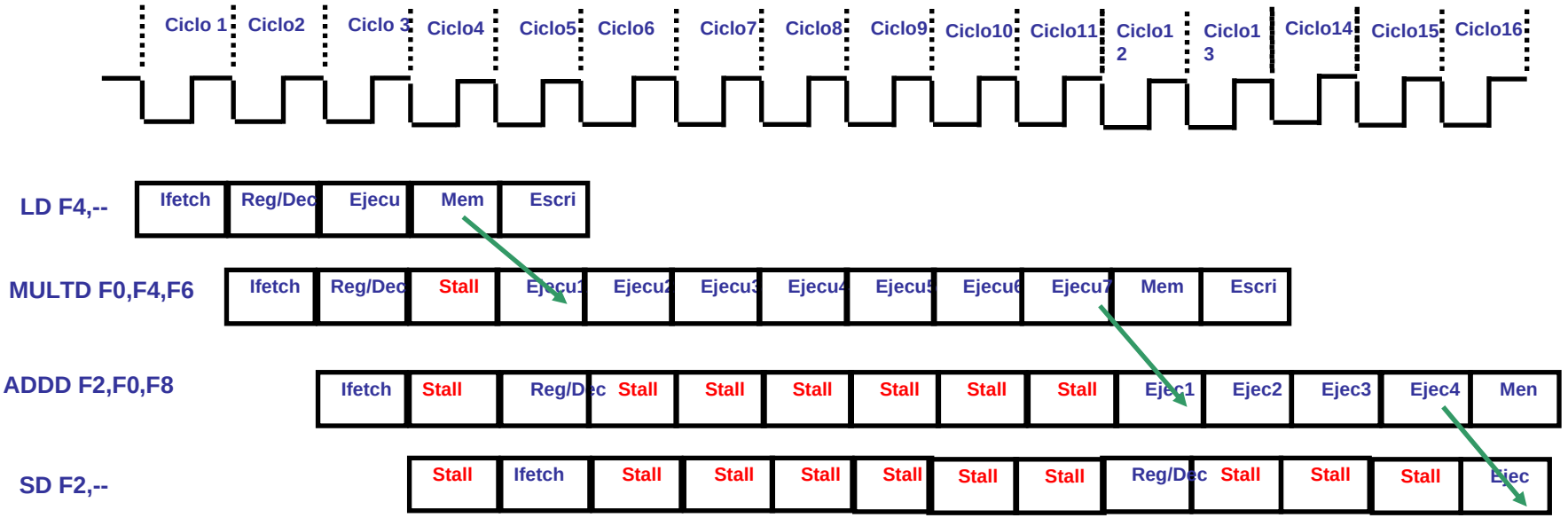


Unidad Funcional	Latencia (de uso)	Intervalo de Iniciación
ALU entera	0	1
FP add	3	1
FP multiplica	6	1
FP división	24	24

- Replicación o segmentación de las unidades de PF
- La división no suele estar segmentada. Detección del riesgo y parada de procesador

Operaciones multiciclo

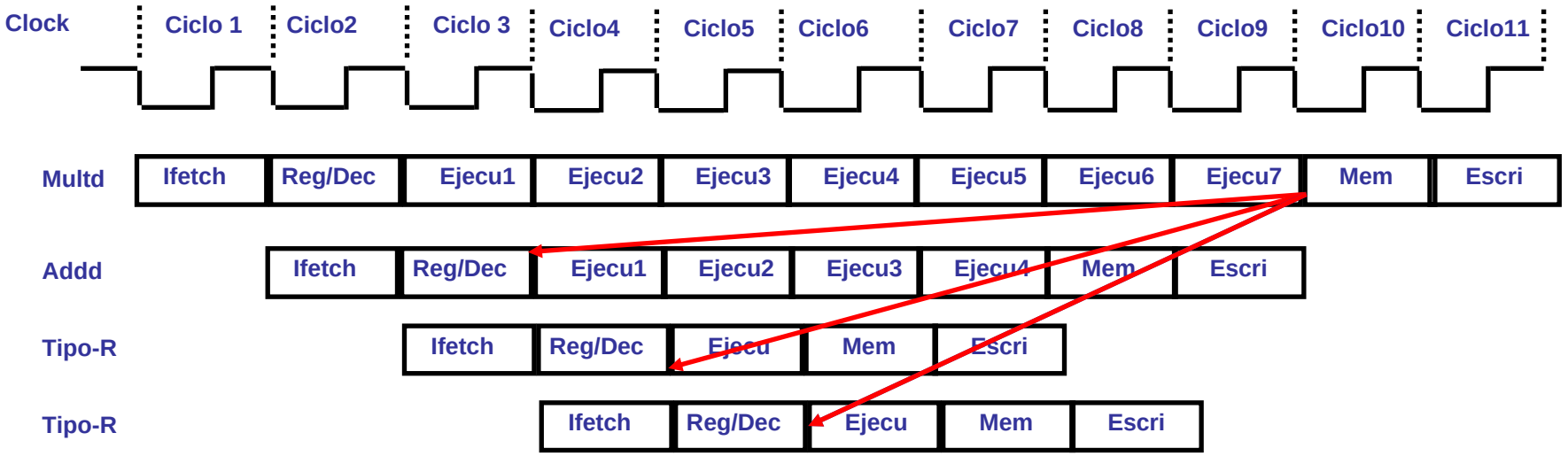
- Riesgos LDE (RAW)



- Mayor impacto en el rendimiento
  - La gran duración de las instrucciones implica más ciclos de detención
  - Es necesaria una planificación más cuidadosa de las instrucciones
- Lógica de cortocircuito más compleja
  - Muchas más comparaciones
  - Bloqueo cuando el cortocircuito no vale para resolver el riesgo LDE

Operaciones multiciclo

- Finalización fuera de orden



- Las instrucciones acaban en orden distinto al lanzamiento
- Problemas
  - Conflictos por escritura simultanea en registros (riesgo estructural)
  - Aparición de riesgos EDE
  - Problema con las excepciones

Muy importante Las instrucciones empiezan en orden y terminan fuera de orden

Operaciones multiciclo

❑ Conflictos por escritura simultánea en registros

- ❑ Problemas si el bloque de registros tiene un único puerto de escritura→ riesgo

- ❑ Solución:
  - o Detener, en la etapa Reg/Dec, las instrucciones que produzcan conflicto:

    Si la instrucción en Reg/Dec necesita escribir en registros en el mismo ciclo que una instrucción ya emitida, la primera se detiene un ciclo.

    Se puede usar un registro de desplazamiento para indicar cuándo usarán el bloque de registros las instrucciones ya lanzadas.

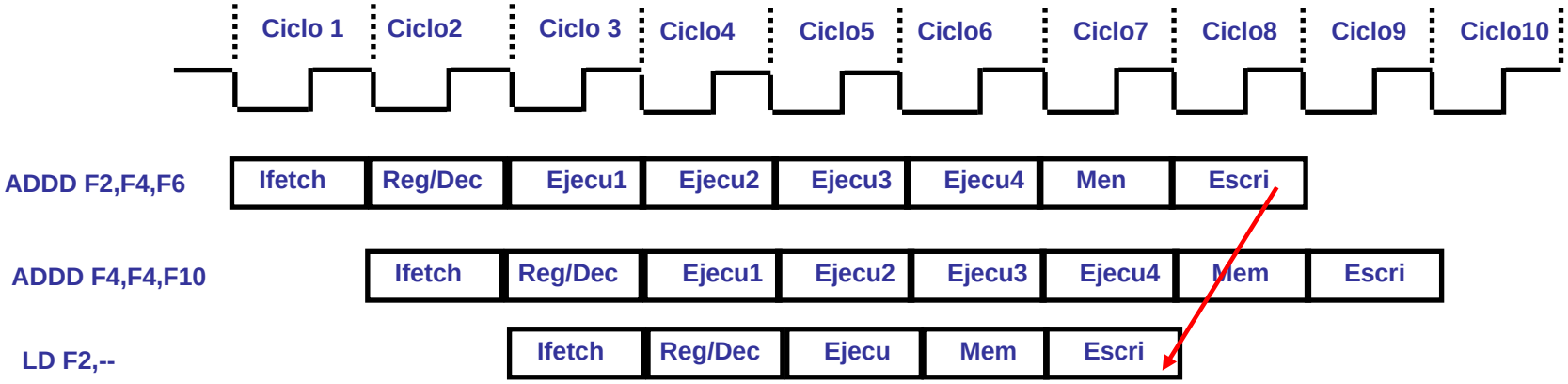
  - o Detener las instrucciones conflictivas al final de Ejecu

    Necesidad de establecer prioridades de acceso: dar mayor prioridad a la unidad con mayor latencia

    Lógica de chequeo de detenciones en dos puntos

Operaciones multiciclo

- Aparición de riesgos EDE



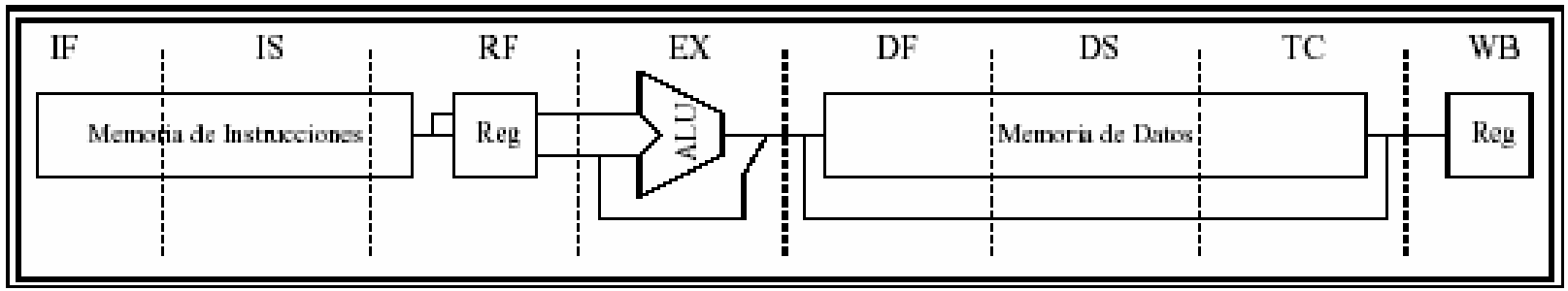
- LD F2,0(R2) escribe en F2 antes que ADDD F2,F4,F6 → error
  - Situación poco común: Instrucción ADDD F2,F4,F6 inútil. Puede ocurrir con instrucciones que ocupen un hueco de retardo
  - Si la segunda ADDD lee F2 → riesgo LDE que elimina el EDE
  - Riesgos EDL(WAR) No aparecen porque **las lecturas de registros son en orden** etapa Reg/Dec)
- Tratamiento de riesgos EDE WAW:
  - Detener en Reg/Dec la instrucción que produce el riesgo (LD en el ejemplo)
  - Eliminar la escritura de la primera instrucción (ADDD en el ejemplo)
- Situaciones poco comunes. No implica demasiado problema cualquiera de las aproximaciones

Un ejemplo: MIPS R4000

Más segmentación 8 etapas

- IF: Primera mitad de búsqueda de instrucción. Inicia el acceso a cache de instrucciones. Actualización del PC.
- IS:{ Segunda mitad de búsqueda de instrucción. Completa el acceso a cache.
- Reg/Dec: Comprobación de acierto en cache. Decodificación y búsqueda de registros. Chequeo de riesgos.
- EX: Ejecución; operación de la ALU, calculo de dirección efectiva, evaluación de condición de salto y cálculo de destino (puede durar varios ciclos)
- DF: Inicio de la búsqueda de datos
- DS: Segunda mitad de la búsqueda de datos
- TC: Chequeo de etiquetas, determinación de acierto en la cache
- WB: Postescritura

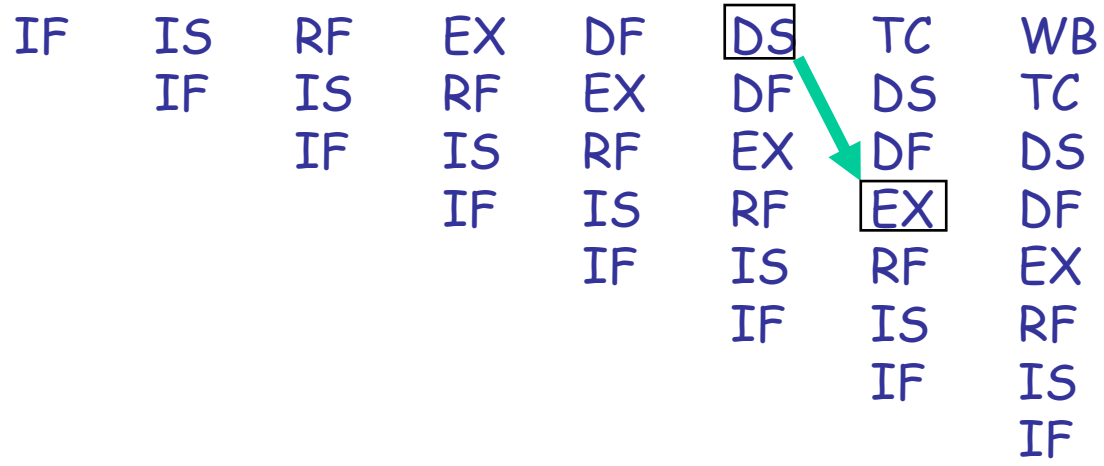
Los datos se encuentran disponibles al final de DS, aunque no se ha comprobado el acierto en la cache



Un ejemplo: MIPS R4000

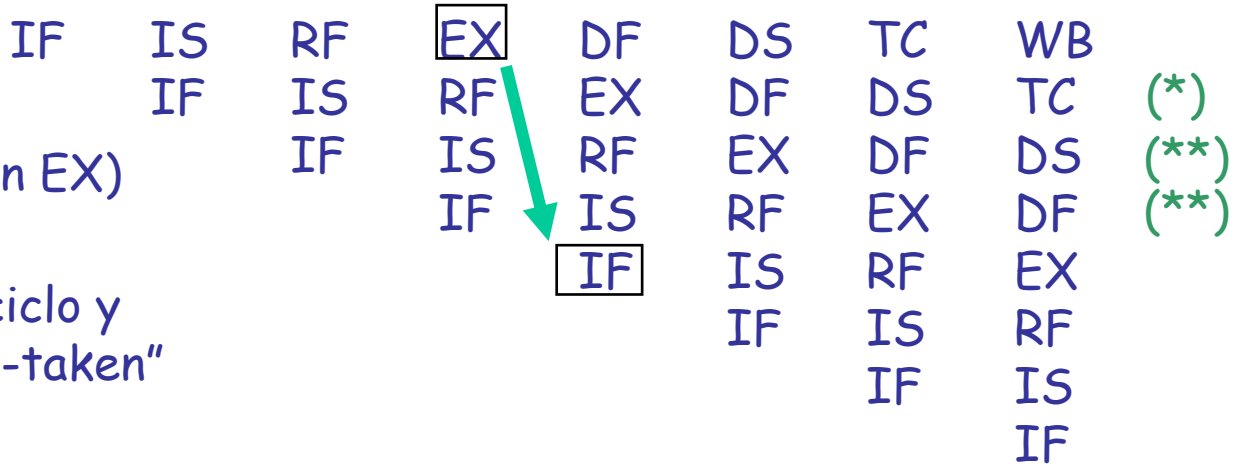
- Más penalización en cargas y saltos

Dos ciclos de latencia en los Load con cortocircuito



Tres ciclos de latencia en saltos
(la condición se evalúa en EX)

Salto retardado de un ciclo y 2 ciclos de "predict-not-taken"



(*)
(**)
(**)

(*) Continúa la ejecución en cq caso
(**) Se abortan en caso de que el salto se tome

Un ejemplo: MIPS R4000

- Operaciones en PF
  - Tres unidades funcionales: divisor, multiplicador y sumador en punto flotante
  - Diferentes unidades son usadas para completar una operación
  - Operaciones con duración entre 2 ciclos (para una negación) y 112 ciclos (para una raíz cuadrada)
  - Segmentadas con un total de 8 estados
  - Descripción de las etapas de las unidades en PF

Etapas	*Unidad*	*Descripción*
A	FP adder	Etapa de suma de mantisas
D	FP divider	Etapa de división
E	FP multiplier	Etapa de test de excepciones
M	FP multiplier	1ª etapa del multiplicador
N	FP multiplier	2ª etapa del multiplicador
R	FP adder	Etapa de redondeo
S	FP adder	Etapa de desplazamiento
U		Etapa de desempaquetado

Un ejemplo: MIPS R4000

❑ Operaciones en PF: Implementación

Instr PF	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*…*
Add, Subtract	U	S+A	A+R	R+S					
Multiply	U	E+M	M	M	M	N	N+A	R	
Divide	U	A	R	D^{28}	…	D+A	D+R,	D+R,	D+A, D+R, A, R
Square root	U	E	$(A+R)^{108}$		…	A	R		
Negate	U	S							
Absolute value	U	S							
FP compare	U	A	R						

Comportamiento de las unidades de PF

	Latencia	Intervalo de inicialización
Add, Subtract	4	3
Multiply	8	4
Divide	36	35
Square root	112	111
Negate	2	1
Absolute value	2	1
FP compare	3	2

Un ejemplo: MIPS R4000

- Rendimiento
  - No se alcanza el rendimiento ideal CPI= 1:
    - Load stalls (1 o 2 ciclos)
    - Saltos (2 ciclos + slots no rellenos)
    - FP stalls (resultados): riesgo de LDE (latencia)
    - FP stalls (estructurales): Hw FP muy escaso (paralelismo)