

Arquitectura de Computadoras



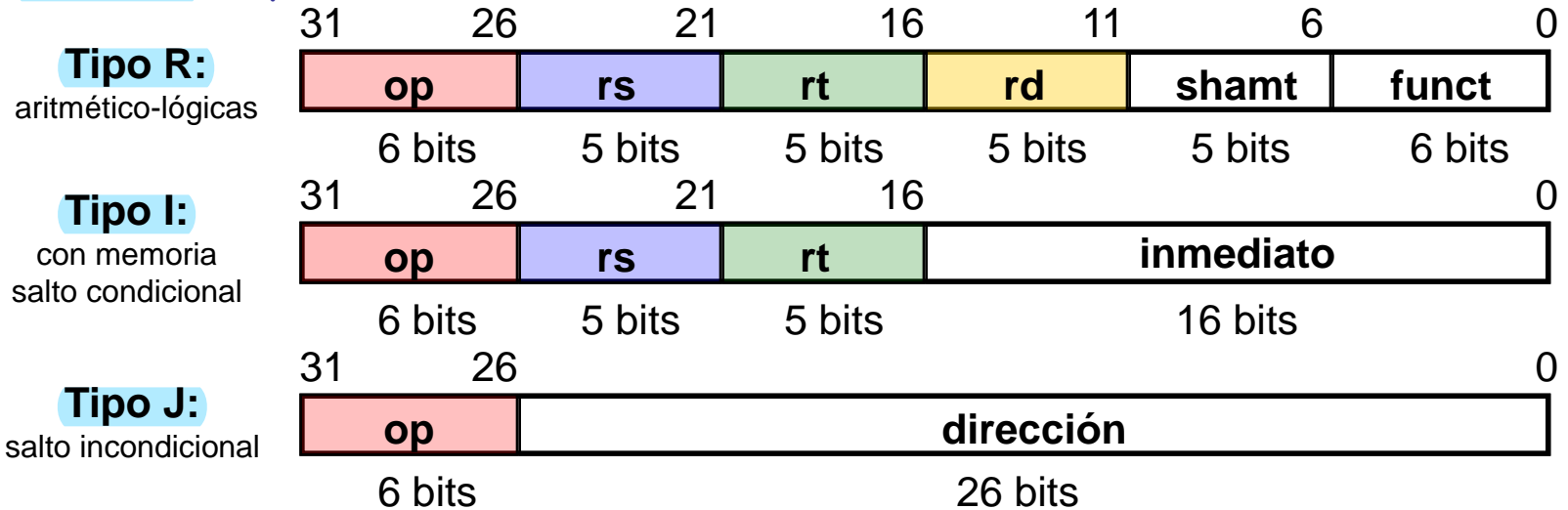
Tema 2 Procesadores Segmentados

2015-20

- ✓ Introducción: 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

❑ Arquitectura MIPS (DLX) (<https://brunonova.github.io/drmips/>)

- ✓ 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

4 Instrucciones

CISC processors also offer many different **addressing modes**.

`add r1, [r2+r3*4+60]` // i86 (not MIPS) example


El MIPS sólo permite un modo de direccionamiento bastante simple: para especificar una dirección, se puede especificar una constante y un registro. Así que, la instrucción Intel anterior podría ser traducida como

```
slli r3, r3, 2 // r3 := r3 << 2 (i.e. r3 := r3 * 4)
add r2, r2, r3 // r2 := r2 + r3
lw r4, 60(r2) // r4 := memory[60+r2]
add r1, r1, r4 // r1 := r1 + r4
```

SPEC pgm	x86	DLX DLX/x86
Gcc	3,771,327,742	3,892,063,460 1.03
espresso	2,216,423,413	2,801,294,286 1.26
spice	15,257,026,309	16,965,928,788 1.11
nasa7	15,603,040,963	6,118,740,321 0.39



Instr.	Description	Format	Opcode	Operation (C-style coding)
ADD	add	R	0x20	$Rd = Rs1 + Rs2$
ADDI	add immediate	I	0x08	$Rd = Rs1 + \text{extend}(\text{immediate})$
AND	and	R	0x24	$Rd = Rs1 \& Rs2$
ANDI	and immediate	I	0x0c	$Rd = Rs1 \& \text{immediate}$
BEQZ	branch if equal to zero	I	0x04	$PC += (Rs1 == 0 ? \text{extend}(\text{immediate}) : 0)$
BNEZ	branch if not equal to zero	I	0x05	$PC += (Rs1 != 0 ? \text{extend}(\text{immediate}) : 0)$
J	jump	J	0x02	$PC += \text{extend}(\text{value})$
JAL	jump and link	J	0x03	$R31 = PC + 4 ; PC += \text{extend}(\text{value})$
JALR	jump and link register	I	0x13	$R31 = PC + 4 ; PC = Rs1$
JR	jump register	I	0x12	$PC = Rs1$
LHI	load high bits	I	0x0f	$Rd = \text{immediate} \ll 16$
LW	load woRd	I	0x23	$Rd = \text{MEM}[Rs1 + \text{extend}(\text{immediate})]$
OR	or	R	0x25	$Rd = Rs1 Rs2$
ORI	or immediate	I	0x0d	$Rd = Rs1 \text{immediate}$
SEQ	set if equal	R	0x28	$Rd = (Rs1 == Rs2 ? 1 : 0)$
SEI	set if equal to immediate	I	0x18	$Rd = (Rs1 == \text{extend}(\text{immediate}) ? 1 : 0)$
SLE	set if less than or equal	R	0x2c	$Rd = (Rs1 \leq Rs2 ? 1 : 0)$
SLEI	set if less than or equal to immediate	I	0x1c	$Rd = (Rs1 \leq \text{extend}(\text{immediate}) ? 1 : 0)$
SLL	shift left logical	R	0x04	$Rd = Rs1 \ll (Rs2 \% 8)$
SLLI	shift left logical immediate	I	0x14	$Rd = Rs1 \ll (\text{immediate} \% 8)$
SLT	set if less than	R	0x2a	$Rd = (Rs1 < Rs2 ? 1 : 0)$
SLTI	set if less than immediate	I	0x1a	$Rd = (Rs1 < \text{extend}(\text{immediate}) ? 1 : 0)$
SNE	set if not equal	R	0x29	$Rd = (Rs1 != Rs2 ? 1 : 0)$
SNEI	set if not equal to immediate	I	0x19	$Rd = (Rs1 != \text{extend}(\text{immediate}) ? 1 : 0)$
SRA	shift right arithmetic	R	0x07	as SRL & see below
SRAI	shift right arithmetic immediate	I	0x17	as SRLI & see below
SRL	shift right logical	R	0x06	$Rd = Rs1 \gg (Rs2 \% 8)$
SRLI	shift right logical immediate	I	0x16	$Rd = Rs1 \gg (\text{immediate} \% 8)$
SUB	subtract	R	0x22	$Rd = Rs1 - Rs2$
SUBI	subtract immediate	I	0x0a	$Rd = Rs1 - \text{extend}(\text{immediate})$
SW	store woRd	I	0x2b	$\text{MEM}[Rs1 + \text{extend}(\text{immediate})] = Rd$
XOR	exclusive or	R	0x26	$Rd = Rs1 \wedge Rs2$
XORI	exclusive or immediate	I	0x0e	$Rd = Rs1 \wedge \text{immediate}$




openDLX

A DLX/MIPS processor simulator

Status: **Beta** Brought to you by: **smetzlaff**

[Add a Review](#)
[Downloads: 5 This Week](#)


[Download](#)

[Get Updates](#)
[Share This](#)

[Windows](#)
[Mac](#)
[Linux](#)

2. DESCRIPCIÓN DEL ENTORNO DE SIMULACIÓN

Una vez abierta la aplicación podrá observar que aparece una ventana principal que a su vez contiene 6 ventanas inicialmente minimizadas, mostrando cada una de ellas diferentes aspectos relacionados con el procesador que se está simulando. El nombre de cada una de estas seis ventanas es:

- Registros (*Register*).
- Código (*Code*)
- Pipeline.
- Diagrama de Ciclos de Reloj (*Clock Cycle Diagram*)
- Estadísticas (*Statistics*)
- Puntos de ruptura (*Breakpoints*)

openDLX 1.0-6

File Simulator Window Help

memory

start addr 0 rows 100

address	value
0x00000000	0x00000000
0x00000004	0x00000000
0x00000008	0x00000000
0x0000000c	0x00000000
0x00000010	0x00000000
0x00000014	0x00000000
0x00000018	0x00000000
0x0000001c	0x00000000
0x00000020	0x00000000
0x00000024	0x00000000
0x00000028	0x00000000
0x0000002c	0x00000000
0x00000030	0x00000000
0x00000034	0x00000000
0x00000038	0x00000000
0x0000003c	0x00000000
0x00000040	0x00000000
0x00000044	0x00000000
0x00000048	0x00000000
0x0000004c	0x00000000
0x00000050	0x00000000
0x00000054	0x00000000
0x00000058	0x00000000
0x0000005c	0x00000000

reload

register set

Register	Values
r0	0x00000000
r1	0x00000001
r2	0x00000008
r3	0x00000000
r4	0x00000000
r5	0x00000009
r6	0x00000007
r7	0x0000000f
r8	0x00000009
r9	0x00000001
r10	0x00000000
r11	0x00000000
r12	0x00000000
r13	0x00000000
r14	0x00000000
r15	0x00000000
r16	0x00000000
r17	0x00000000
r18	0x00000000
r19	0x00000000
r20	0x00000000
r21	0x00000000
r22	0x00000000
r23	0x00000000
r24	0x00000000
r25	0x00000000
r26	0x00000000
r27	0x00000000
r28	0x00000000
r29	0x00000000
r30	0x00000000
r31	0x00000000

code

address	code hex	code
0x00004000	0x20010001	addi r1,r
0x00004004	0x20020008	addi r2,r
0x00004008	0x00222820	add r5,r1
0x0000400c	0x00413022	sub r6,r2
0x00004010	0x00a63818	mult r7,r
0x00004014	0x00e6401a	div r8,r7
0x00004018	0x00a64824	and r9,r5
0x0000401c	0x00a65025	or r10,r5
0x00004020	0x00a65826	xor r11,r
0x00004024	0x00006080	slli r12,
0x00004028	0x0000000c	trap 0

coding name

```

12 ; addi Destination, Source, Value
13 ; adds the content of the source register with the immediate value and writ
14     addi    r1,r0,1    ; adds 1 to content of register 0 (=0) and w
15     addi    r2,r0,8    ; similar instruction, writing the 9 to R2
16                                     ; since r0 is always 0,
17
18 ; Instruction format of register-register-instructions
19 ; add Destination, Source1, Source2
20 ; adds the contents of the two source registers and writes result to destin
21     add     r5,r1,r2    ; r5 = 9
22     sub     r6,r2,r1    ; r6 = 7
23     mult    r7,r5,r6    ; r7 = 63
24     div     r8,r7,r6    ; r8 = 9
25     and     r9,r5,r6    ; r9 = 9 AND 7 = 1001 AND 0111 = 0001 = 1
26     or      r10,r5,r6   ; r10 = 9 OR 7 = 1001 OR 0111 = 1111 = 15
27     XOR     r11,r5,r6   ; exclusive or
28                                     ; r11 = 9 XOR 7 = 1001 X
29     SLI     r12,r6,2    ; r12 = 0111 << 2 = 011100 = 28
30                                     ; shift left, complies w
31
32 ; end of program
33 trap 0
  
```

assemble save as clear

statistics

----- SIMULATION STATISTI

Cycles: 11

Executed instructions: 7

Performed fetches: 11

Jumps: 0 (taken: 0, not take

Branch Target Buffer (1, S

Jumps correctly predicted: 0

Number of unique jumps: 0

log

DEBUG [EXECUTE/ALU]: 9(0x00000009) ..

DEBUG [EXECUTE]: PC: 0x00004020 ALU...

DEBUG [BP_MODULE]: instruction at: ...

DEBUG [MEMORY]: PC: 0x0000401c noth...

cycles and pipeline

Instructions/Cycles	3	4	5	6	7	8	9	10	11
addi r1,r0,1	MEM	WB							
addi r2,r0,8	EX	MEM	WB						
add r5,r1,r2	ID	EX	MEM	WB					
sub r6,r2,r1		IF	ID	EX	MEM	WB			
mult r7,r5,r6			IF	ID	EX	MEM	WB		
div r8,r7,r6				IF	ID	EX	MEM	WB	
and r9,r5,r6					IF	ID	EX	MEM	WB
or r10,r5,r6						IF	ID	EX	MEM
xor r11,r5,r6							IF	ID	EX
slli r12,r6,2								IF	ID

OpenDLX Simulator run

Run Pause Stop 1x 2x 4x 8x 16x

DrMIPS

Educational MIPS simulator

About DrMIPS

DrMIPS is a graphical simulator of the MIPS processor to support computer architecture teaching and learning. It is intuitive, versatile and configurable.

The simulator is available not only for personal computers but also for Android devices, especially tablets.

DrMIPS was created under the Master's dissertation entitled *Tool to Support Computer Architecture Teaching and Learning* at FEUP.

DrMIPS is open-source and licensed under the [GPLv3](#), so you are free to use, redistribute, modify and improve it (under certain conditions). Feel free to [contribute!](#)

```
Code Assembled Datapath Data memory
1 # el set de instrucciones http://www.cs.jhu.edu/~cs333/reference.html
2
3 .data
4 num: .word 5 # desired index of the Fibonacci sequence
5 res: .word 0 # the result
6
7 .text
8 # initialize
9 lw $a0, num # desired index
10 li $t0, 1 # current index (counter)
11 li $t1, 1 # n-1
12 li $t2, 1 # n
13
14 loop: # find the desired index
15 ble $a0, $t0, end # found? Salta a la instrucción situada en la dirección espec
16 addi $t0, $t0, 1 # increment counter
17 move $t3, $t2 # save $t3 temporarily
18 add $t2, $t2, $t1 # next n
19 move $t1, $t3 # next n-1
20 b loop # unconditional branch to program Label target
21
22 end:
23 sw $t2, res # store result, register_source, RAM_destination
24
```

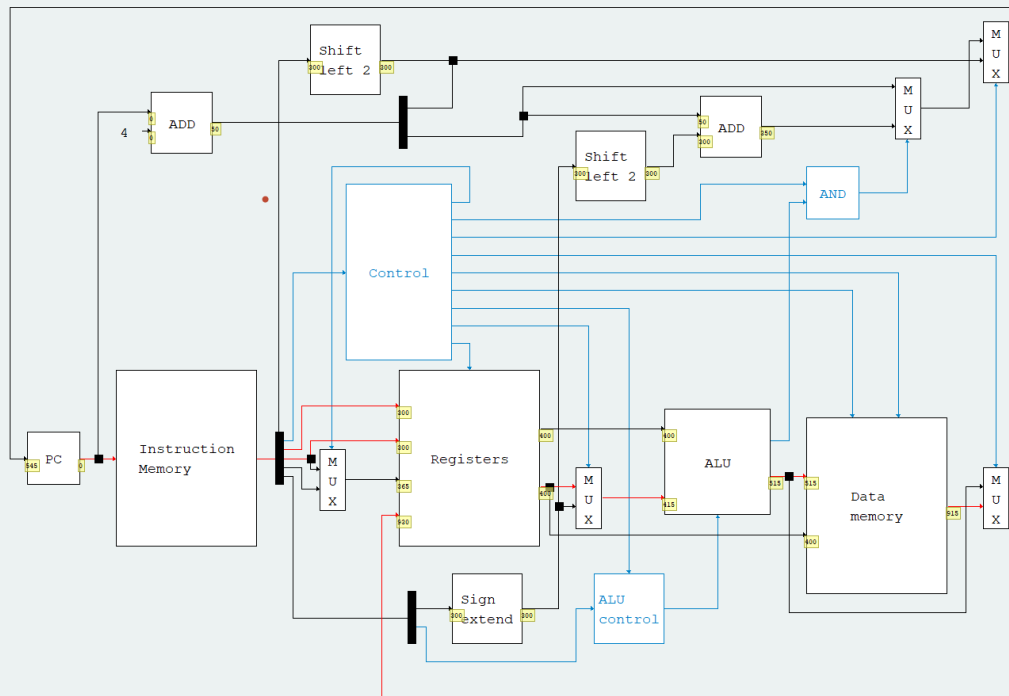
```
Code
9: lw $a0, num
10: addi $t0, $0, 1 # li $t0, 1
11: addi $t1, $0, 1 # li $t1, 1
12: addi $t2, $0, 1 # li $t2, 1
15: loop: slt $t1, $t0, $a0 # ble $a0, $t0, end
15: beq $t1, $0, end
16: addi $t0, $t0, 1
17: add $t3, $t2, $0 # move $t3, $t2
18: add $t2, $t2, $t1
19: add $t1, $t3, $0 # move $t1, $t3
20: beq $0, $0, loop # b loop
23: end: sw $t2, res
```

File View Edit Datapath Execute CPU Help

Zoom: 223%

Code Assembled Datapath Data memory

lw \$a0, num



Registers

Register	Value
0: \$zero	0
1: \$at	0
2: \$v0	0
3: \$v1	0
4: \$a0	0
5: \$a1	0
6: \$a2	0
7: \$a3	0
8: \$t0	0
9: \$t1	0
10: \$t2	0
11: \$t3	0
12: \$t4	0
13: \$t5	0
14: \$t6	0
15: \$t7	0
16: \$s0	0
17: \$s1	0
18: \$s2	0
19: \$s3	0
20: \$s4	0
21: \$s5	0
22: \$s6	0
23: \$s7	0
24: \$t8	0
25: \$t9	0
26: \$k0	0
27: \$k1	0
28: \$gp	0
29: \$sp	0
30: \$fp	0
31: \$ra	0
PC	0

❑ Ruta de datos (multiciclo)

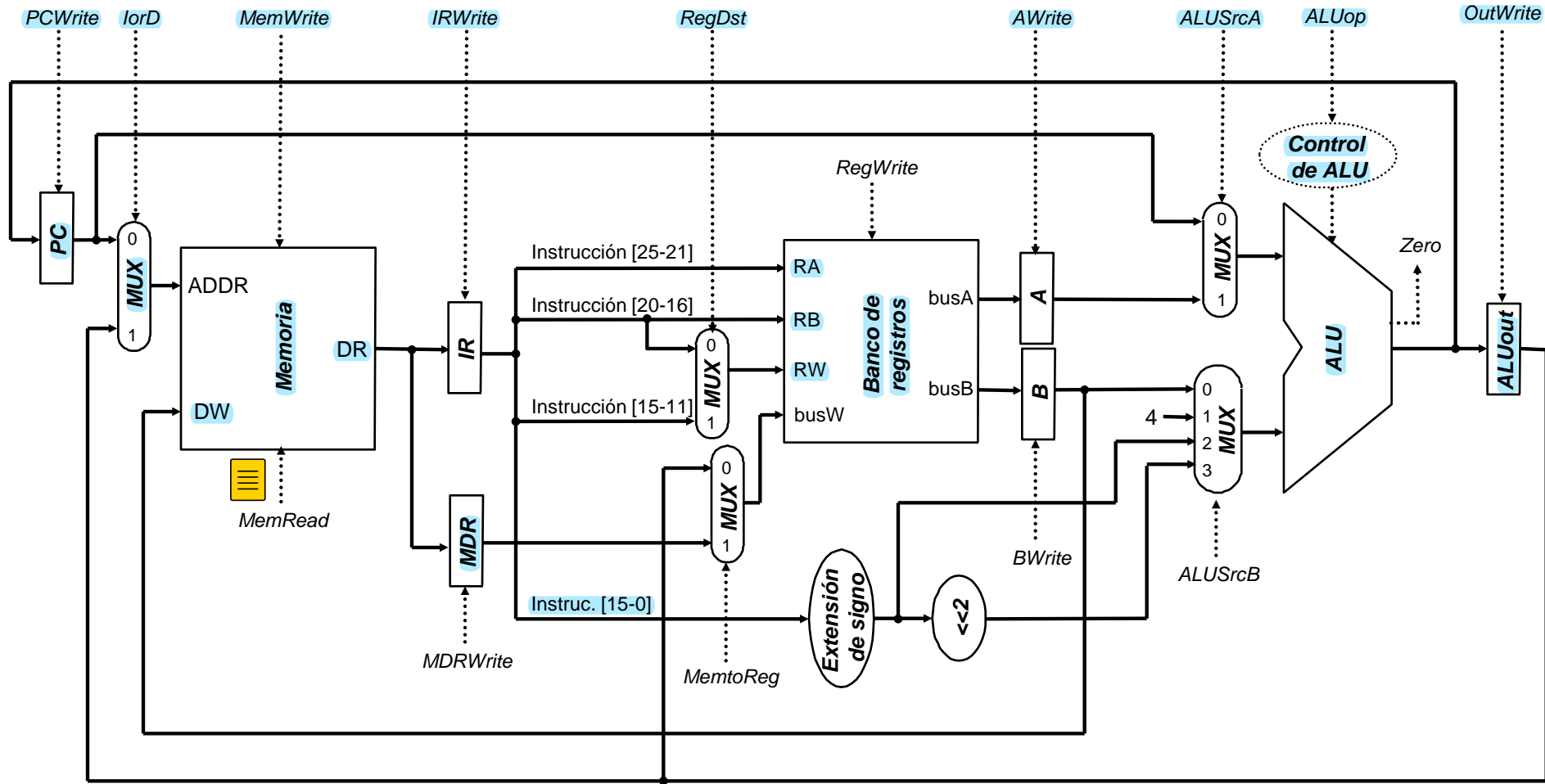
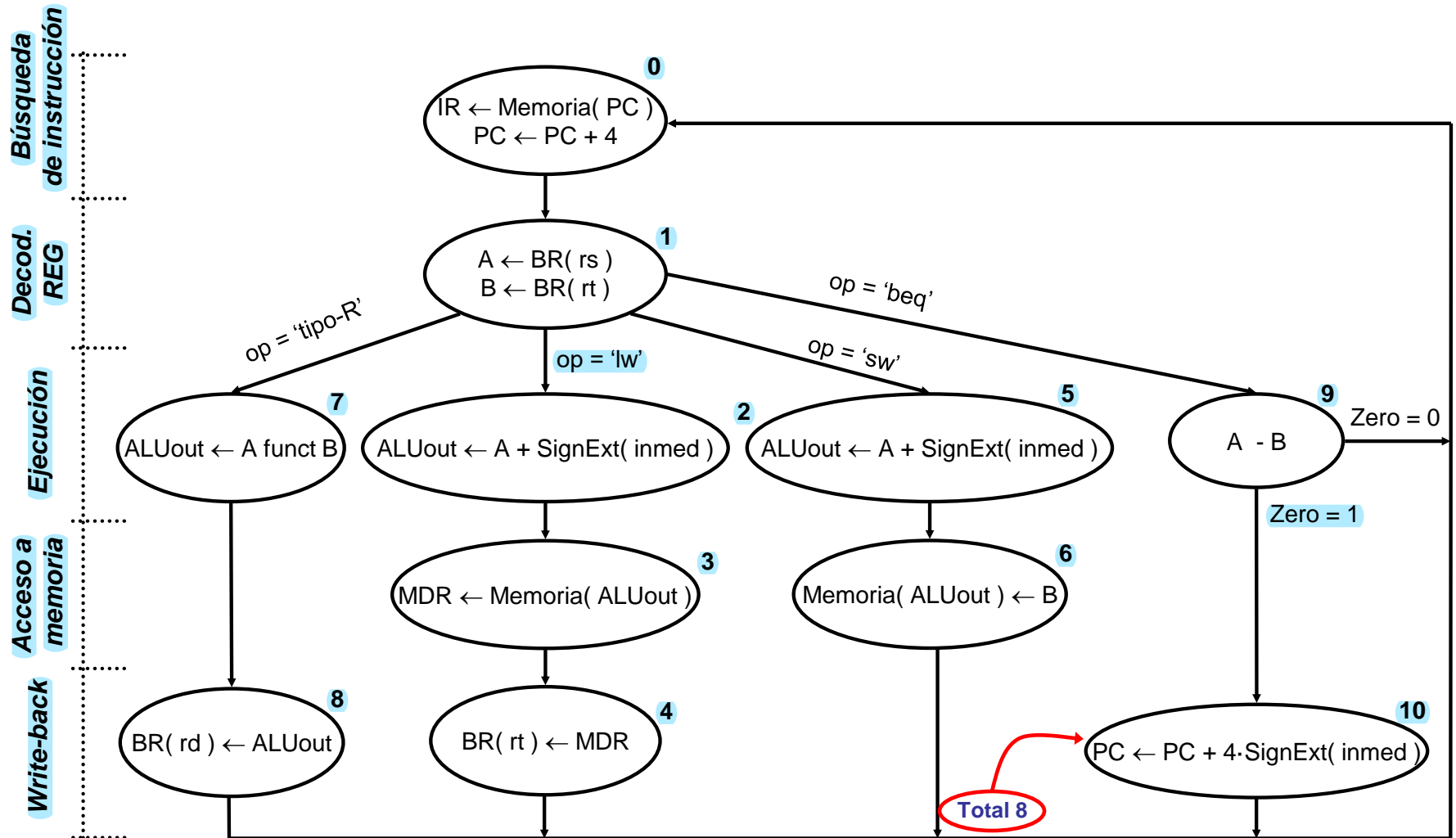


Diagrama de estados del controlador multiciclo



□ Dos tipos

✓ Interrupciones

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

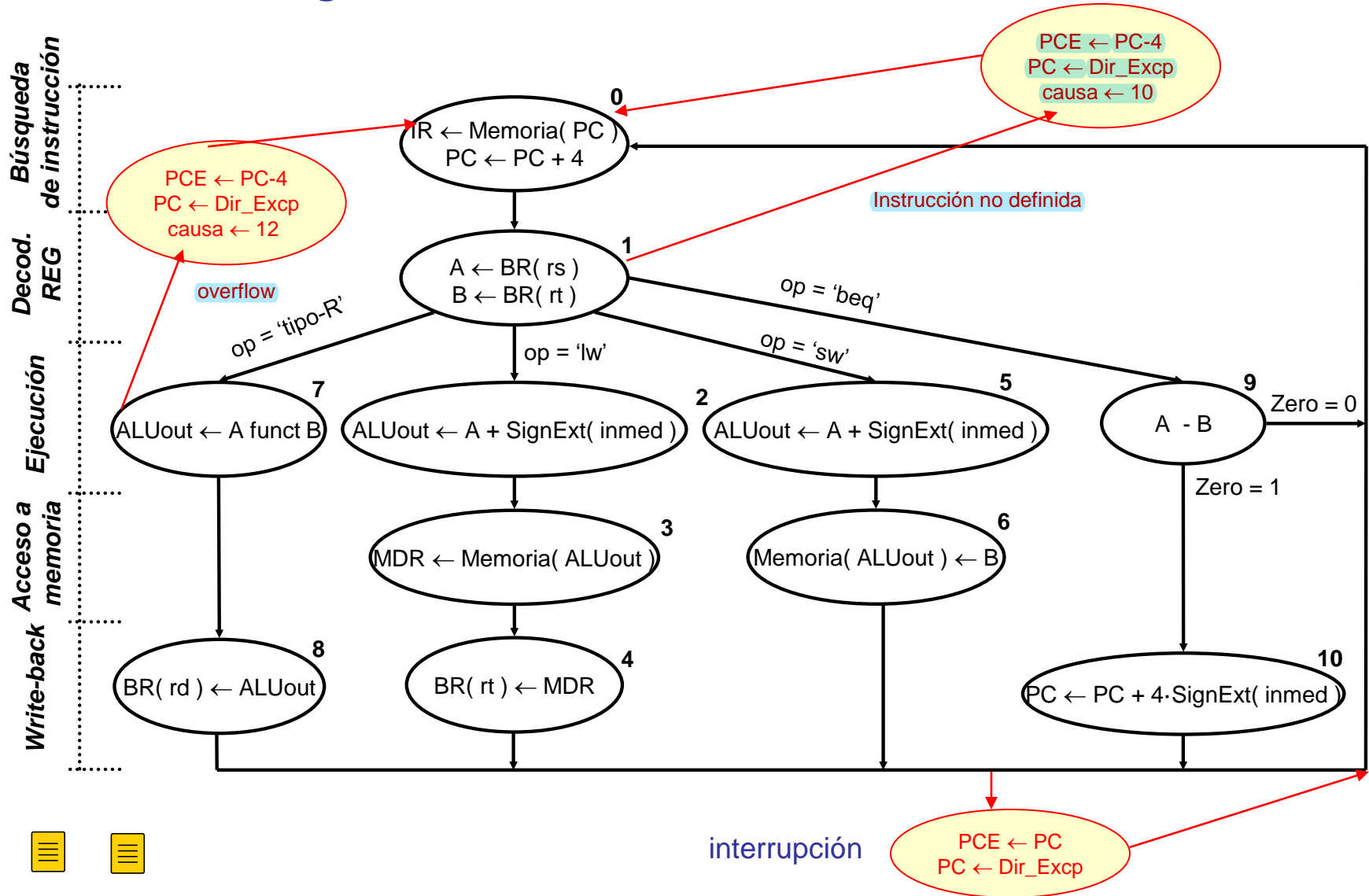
✓ Traps

- 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

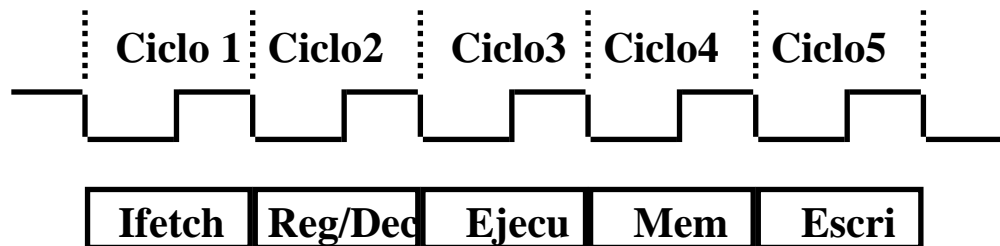
- ✓ Tratamiento dentro del control
- ✓ Salvar el estado (CP y la causa)
 - Stack 68k, x86
 - Registros específicos CPE, Causa

□ Nuevo diagrama de estados



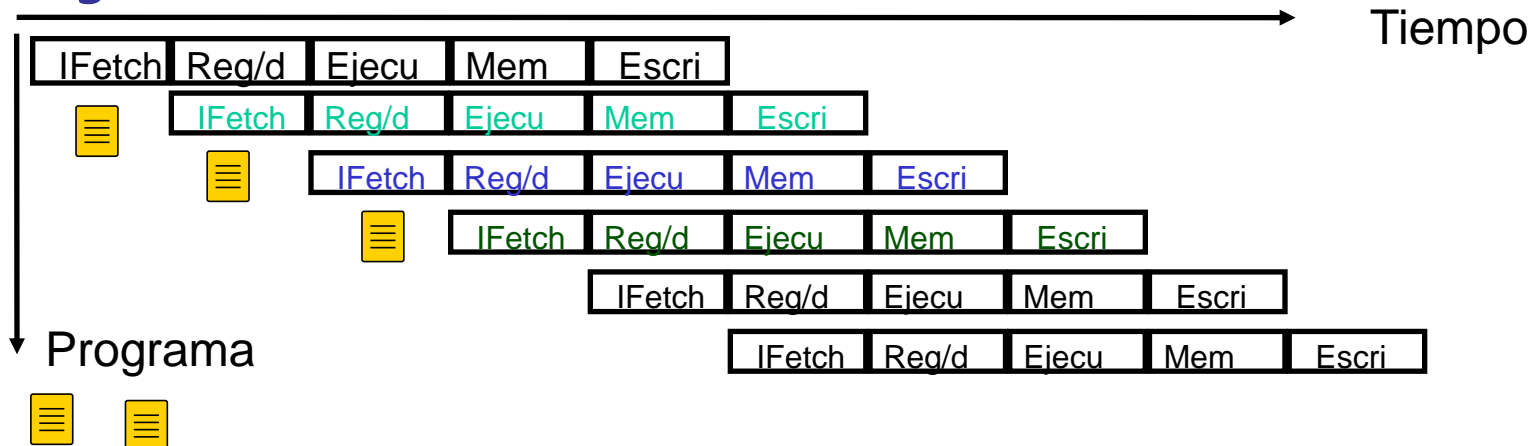


□ Etapas de una instrucción (Load)



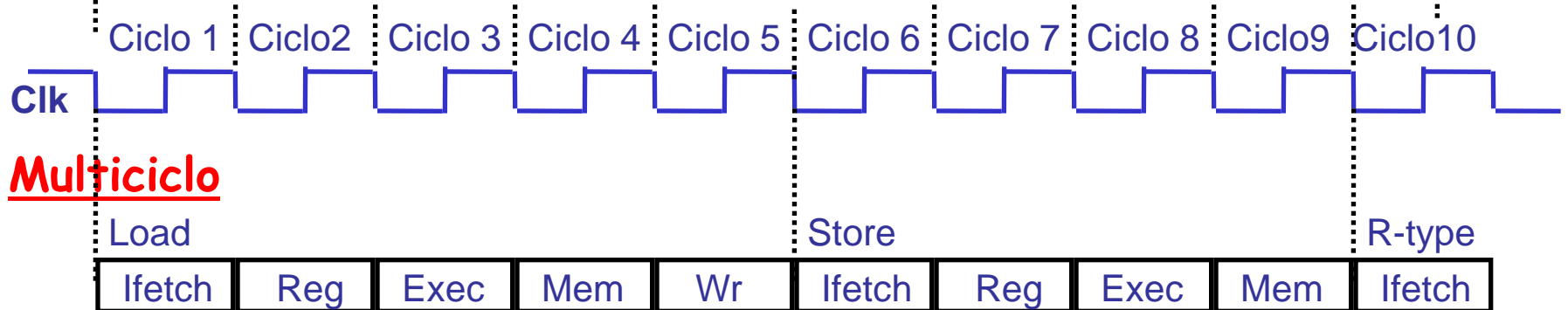
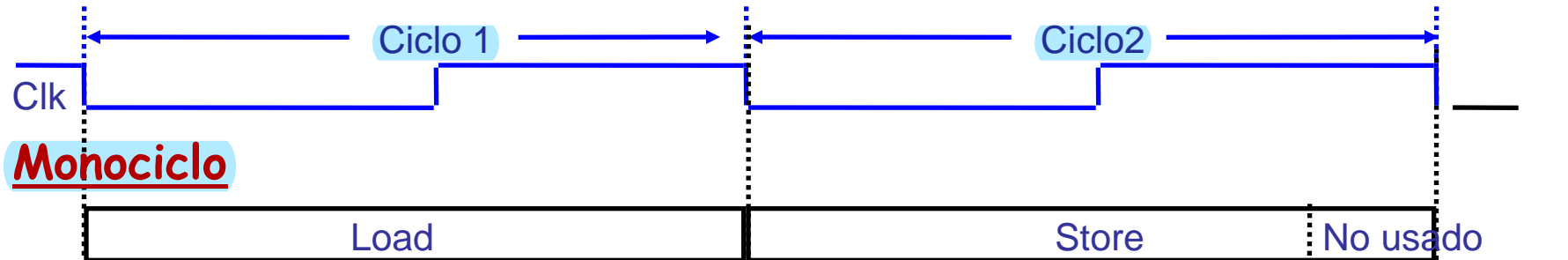
- Ifetch (búsqueda). Lee una instrucción desde la memoria
- Reg/dec. Decodifica la instrucción y lee registros
- Ejecuta. Calcula dirección de memoria
- Mem. Lee el dato de la memoria
- Escri. Escribe el dato en el registro

□ Segmentación

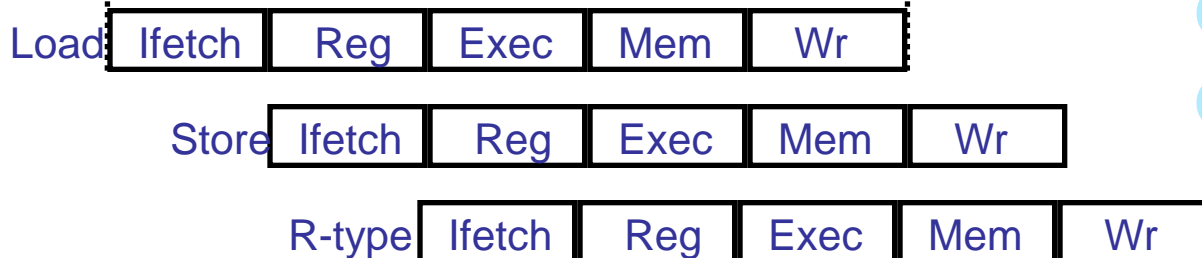


Segmentación

☐ Monociclo, multiciclo, segmentado



Segmentado



100 instrucciones

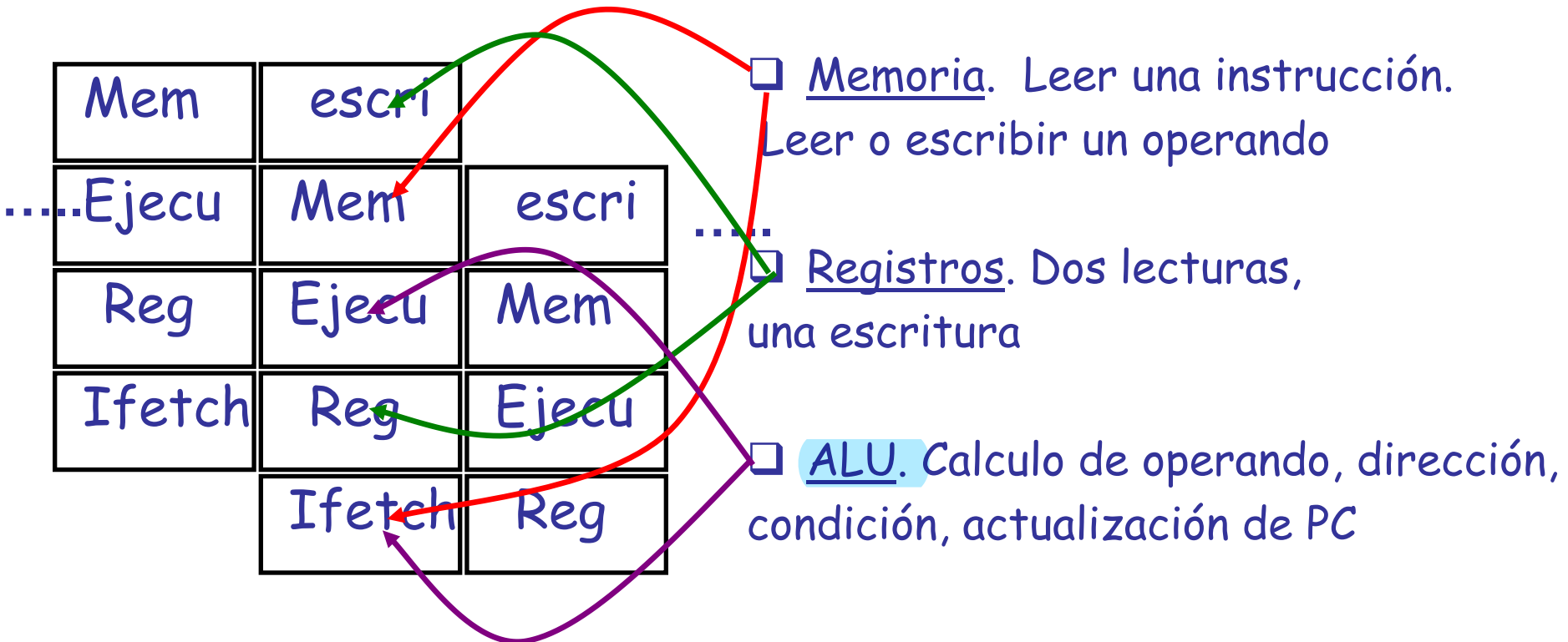
- Monociclo
 $45\text{ns/ciclo} \times 100 = 4500\text{ns}$
- Multiciclo
 $10\text{ns/ciclo} \times 4.6 \text{ CPI} \times 100 = 4600\text{ns}$
- Segmentado
 $10\text{ns} \times (1\text{CPI} \times 100 + 4 \text{ llenado}) = 1040 \text{ ns}$

❑ 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

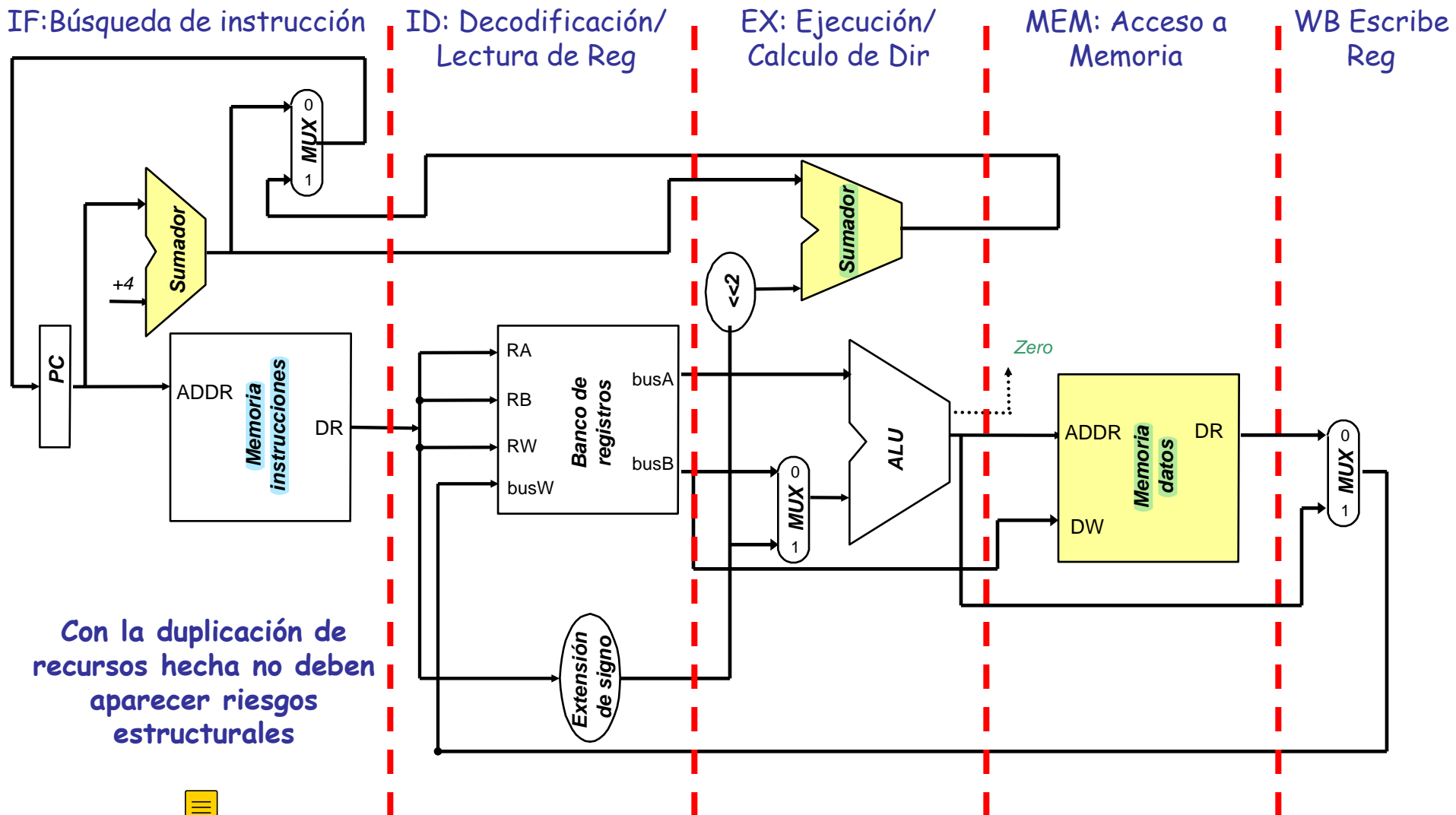
❑ Riesgos estructurales.

Objetivo: Ejecutar sin conflicto cualquier combinación de instrucciones



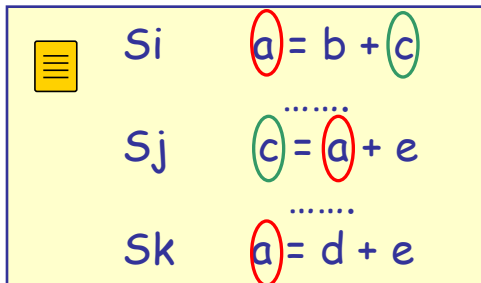
Segmentación: Riesgos estructurales

❑ 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*

$D(i) \cap D(j) \neq \emptyset$ no RIESGO

$D(i) \cap R(j) \neq \emptyset$ riesgo EDL (WAR)

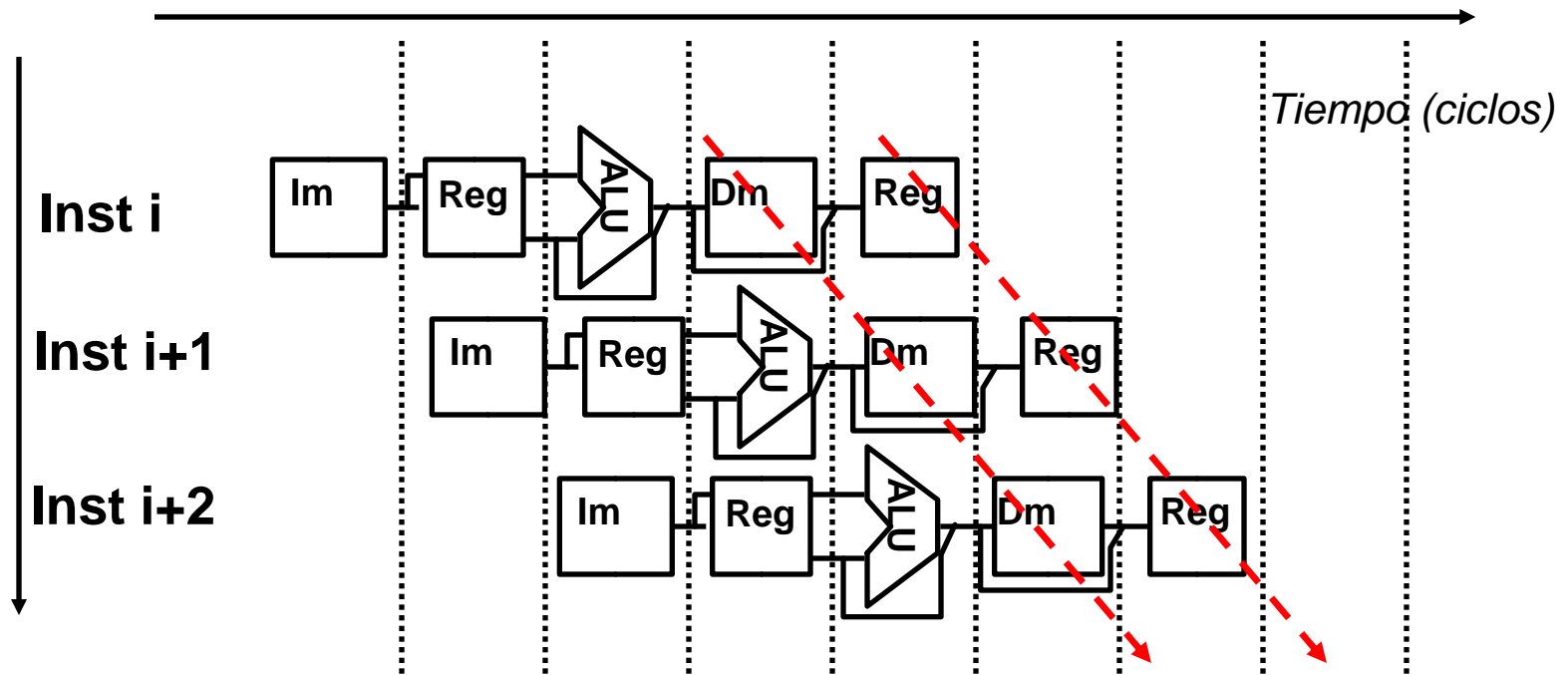
$R(i) \cap D(j) \neq \emptyset$ riesgo LDE (RAW)

$R(i) \cap R(j) \neq \emptyset$ riesgo EDE (WAW)

Riesgos de datos

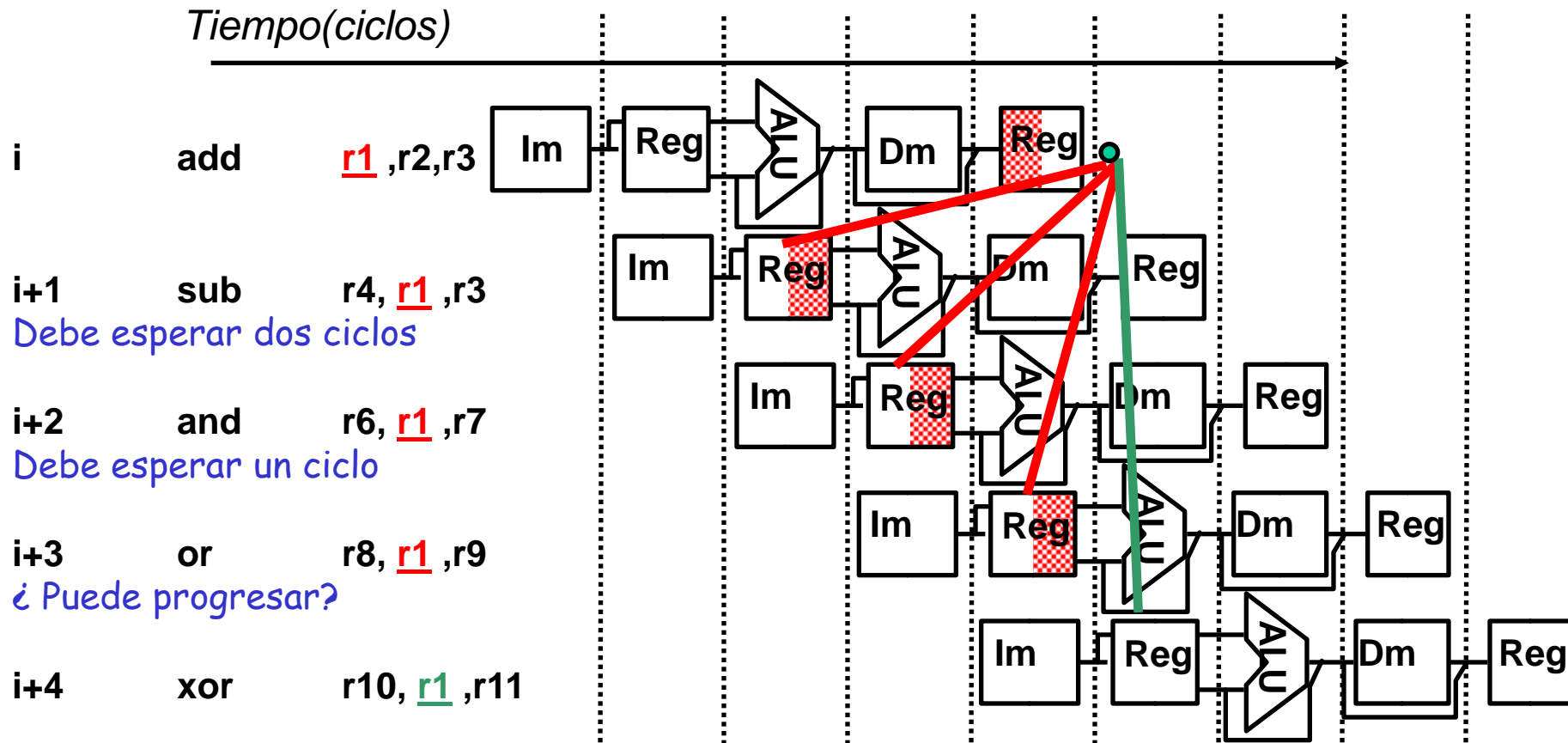
- ☐ Riesgos. $D(i) \cap R(i+?) \neq \emptyset$ EDL (WAR)
 $R(i) \cap R(i+?) \neq \emptyset$ EDE (WAW)

- o Se leen los registros en el final de la segunda etapa
- o Todas las instrucciones escriben en la ultima etapa
- o Todas las instrucciones tienen igual duración



Riesgos de datos

□ Riesgo $R(i) \cap D(i+?) \neq \emptyset$ LDE (RAW)

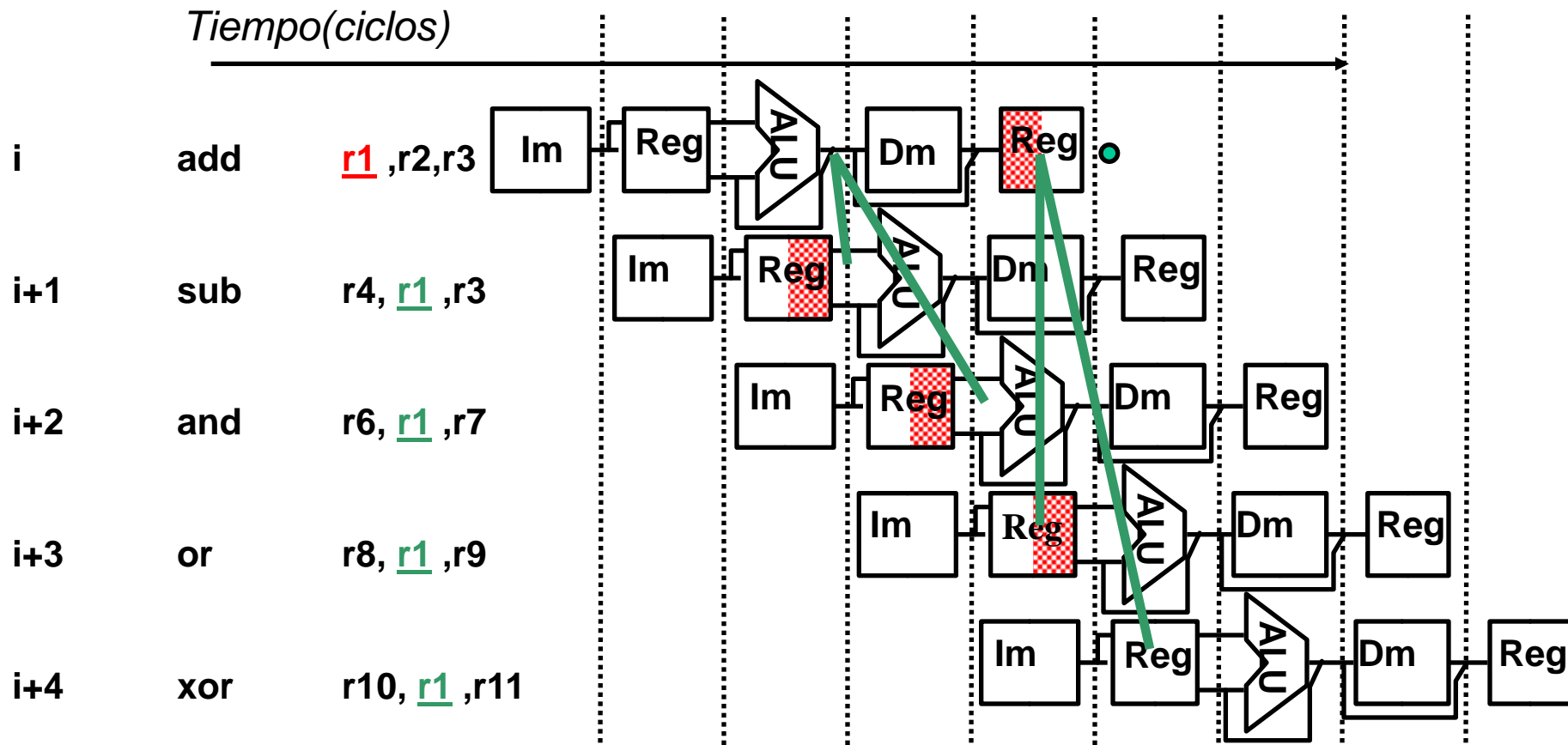


¿ Cuando esta listo el operando ?

Riesgos de datos

□ Riesgo $R(i) \cap D(i+?) \neq \emptyset$ LDE (RAW).

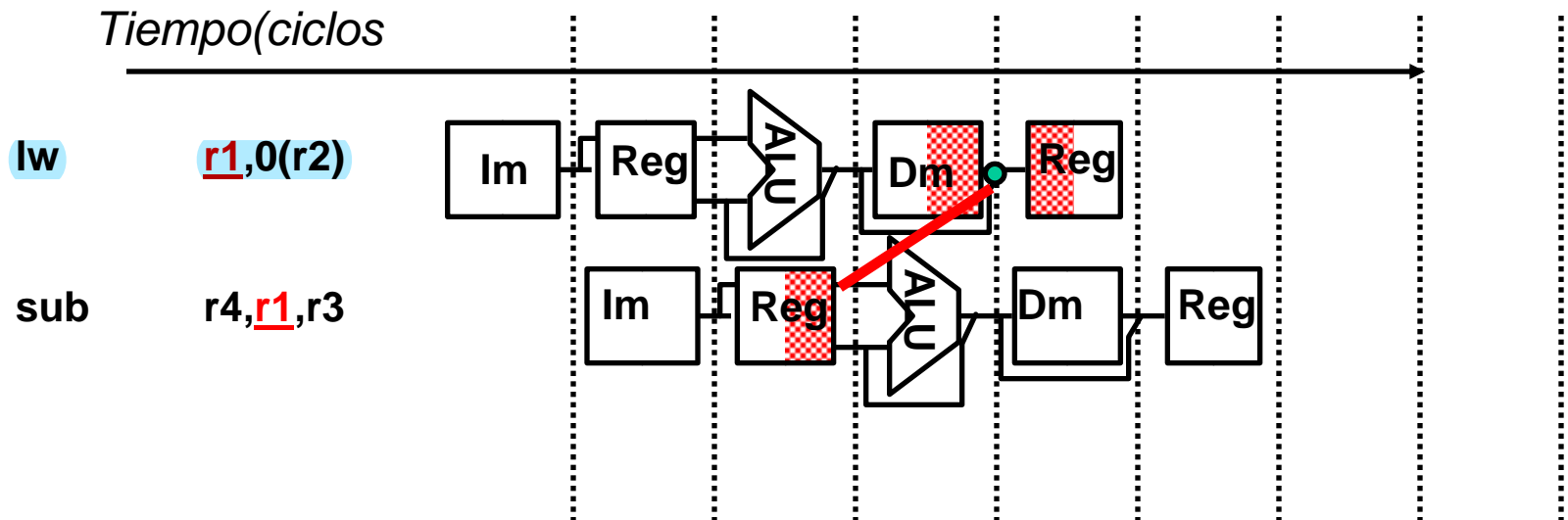
- ✓ Cortocircuito. Enviar el dato a las etapas que lo necesitan, cuanto esta listo



Se pueden ejecutar todas las instrucciones sin problemas

□ Riesgo $R(i) \cap D(i+?) \neq \emptyset$ LDE (RAW).

✓ Caso del Load



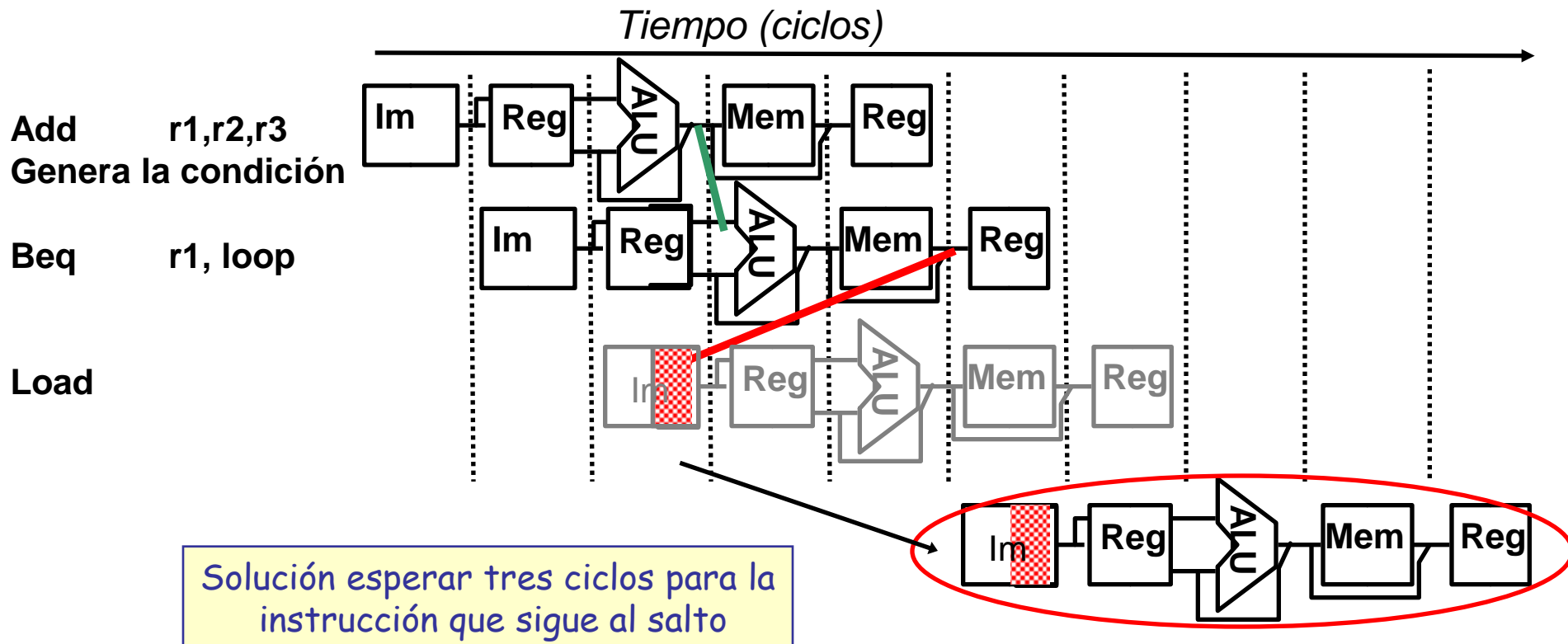
No se puede resolver con el cortocircuito

La instrucción dependiente del Load debe esperar un ciclo

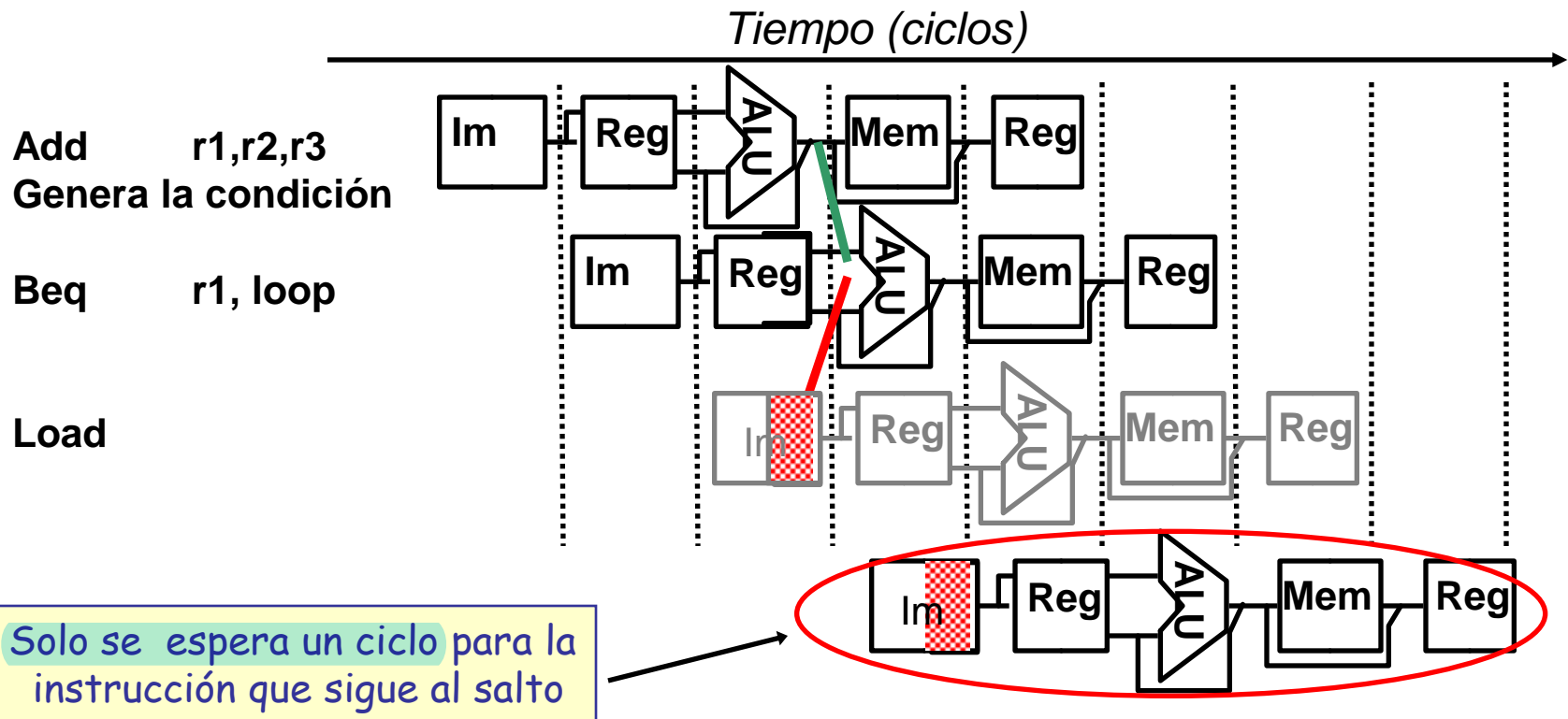
Riesgos de control

❑ 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

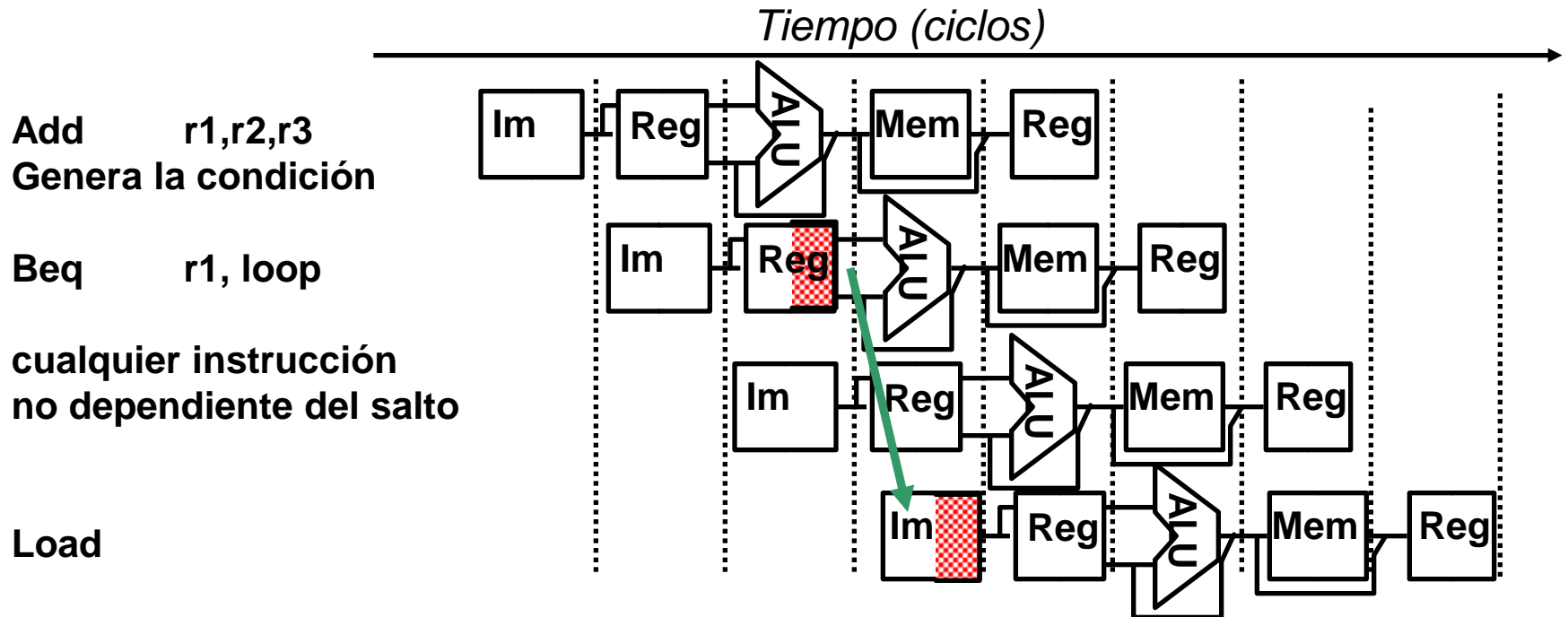


- ❑ **Solución:** Desplazar el calculo de la dirección y la evaluación de la condición a la etapa ID



❑ 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 \Rightarrow 0 ciclos de penalización

Segmentación del procesador

□ Nueva Ruta de datos del DLX para ejecución segmentada

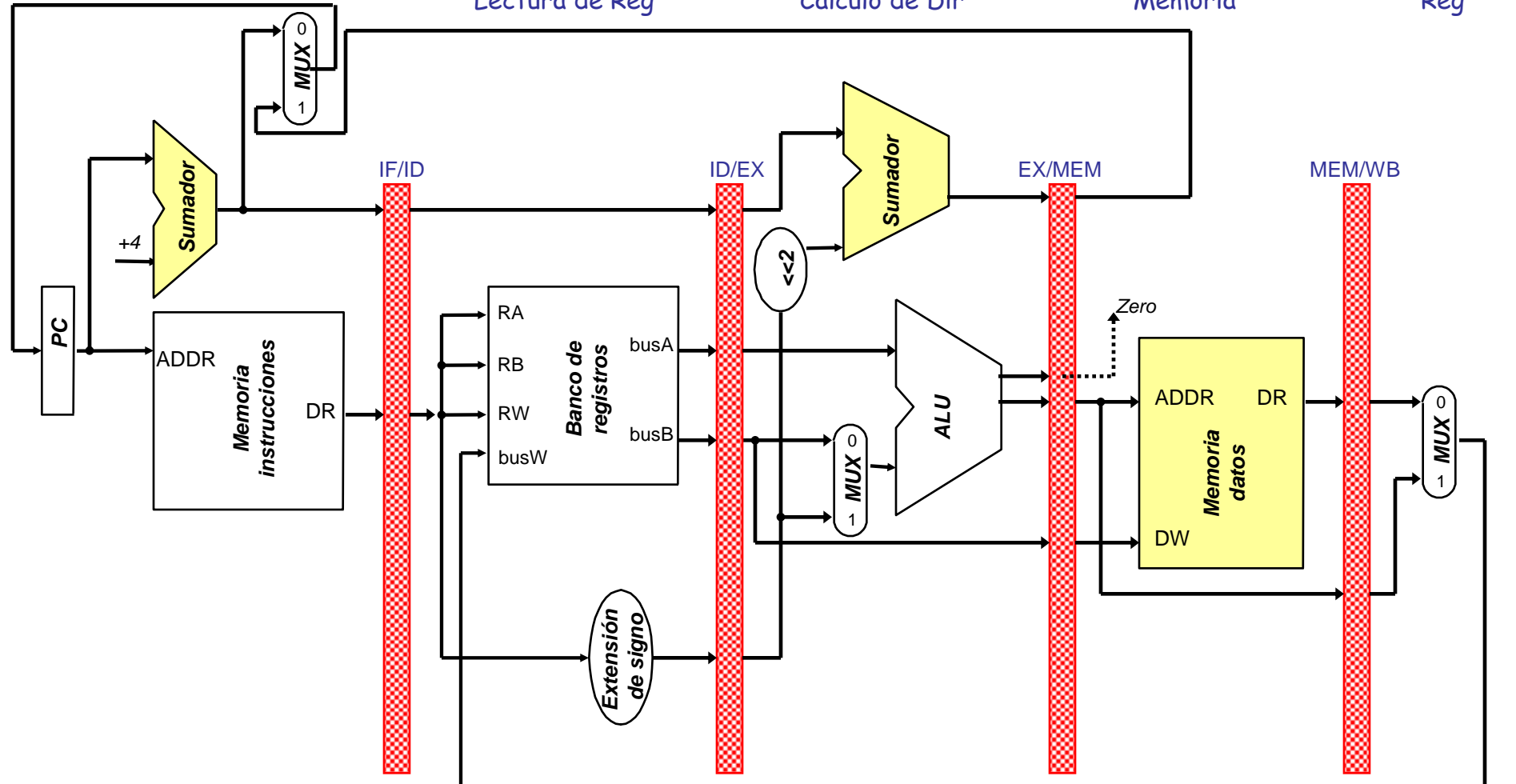
IF: Búsqueda de instrucción

ID: Decodificación/
Lectura de Reg

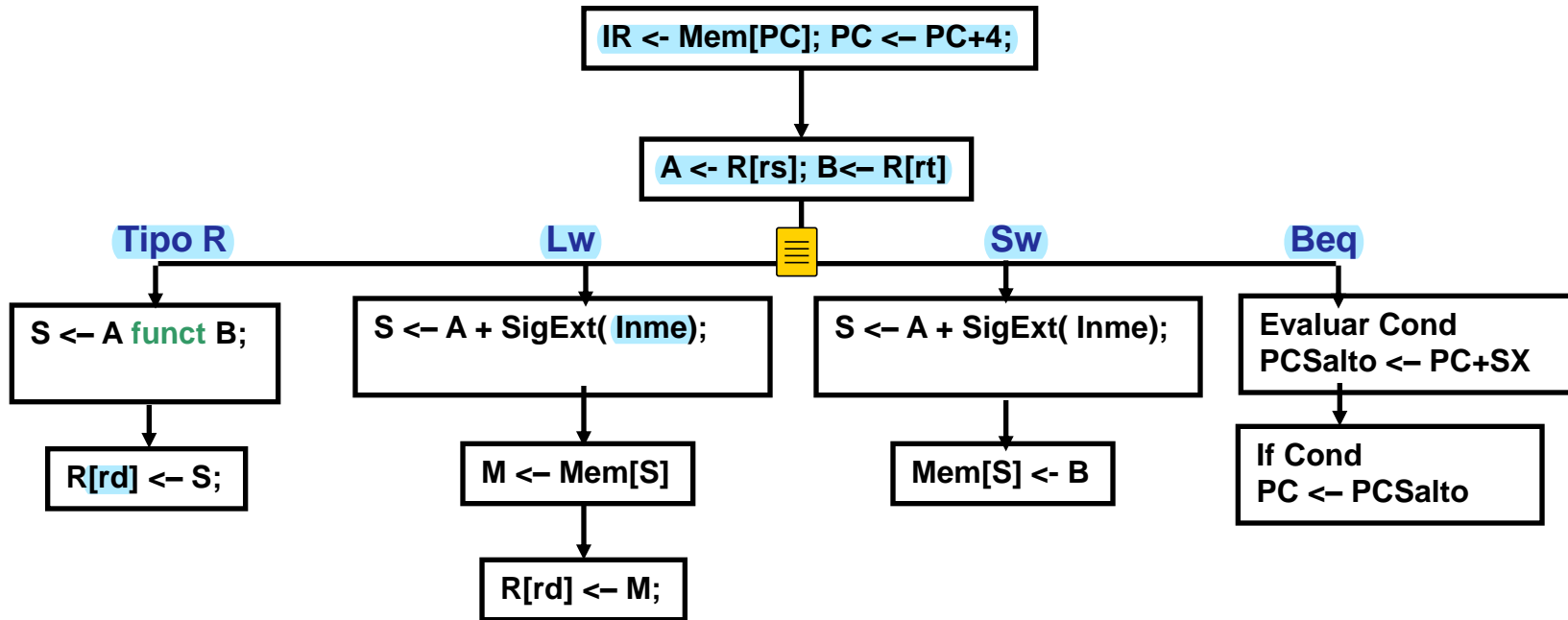
EX: Ejecución/
Cálculo de Dir

MEM: Acceso a
Memoria

WB: Escribe
Reg



□ Ejecución de instrucciones: diagrama RTL



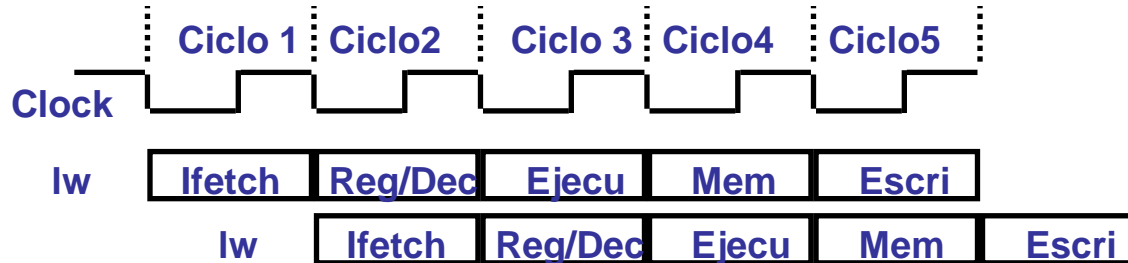
□ Recordatorio

- ✓ $rs = \text{Ins}[25-21]$
- ✓ $rt = \text{Ins}[20-16]$
- ✓ $rd = \text{Ins}[15-11]$
- ✓ $\text{Inme} = \text{Ins}[15-0]$

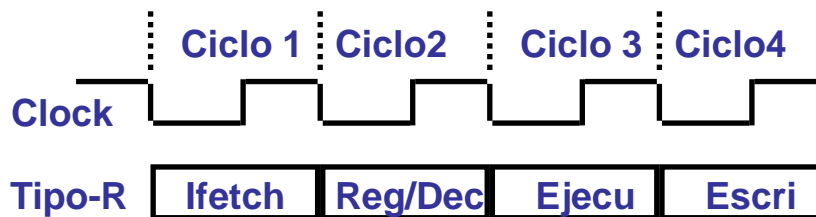
- *Register transfer language or Resistor-transistor logic*

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

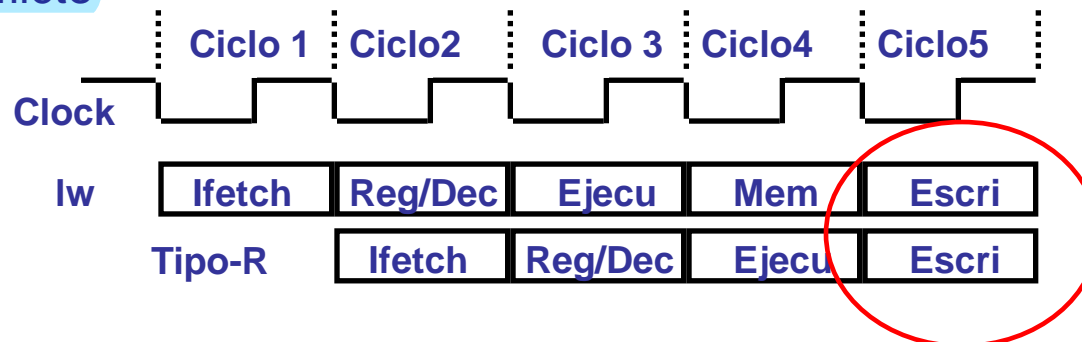
- ✓ Load. Tiene 5 fases y cada una utiliza una de las etapas



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



- ✓ Conflicto



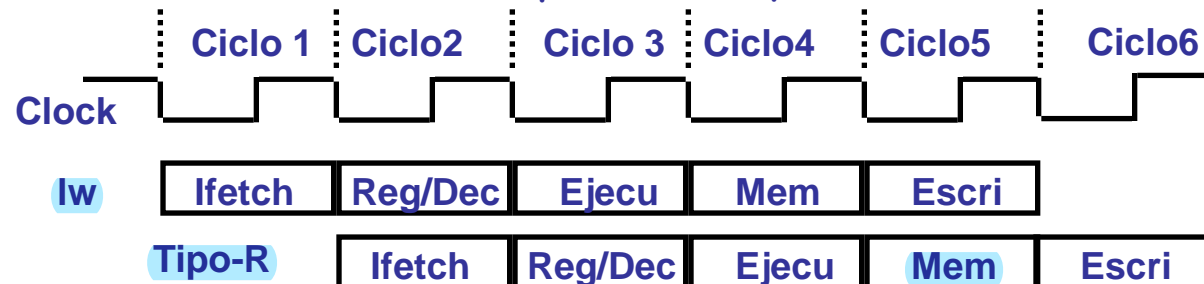
Solo una puerta en el bloque de registros

❑ Importante. ¿ como evitarlo ?

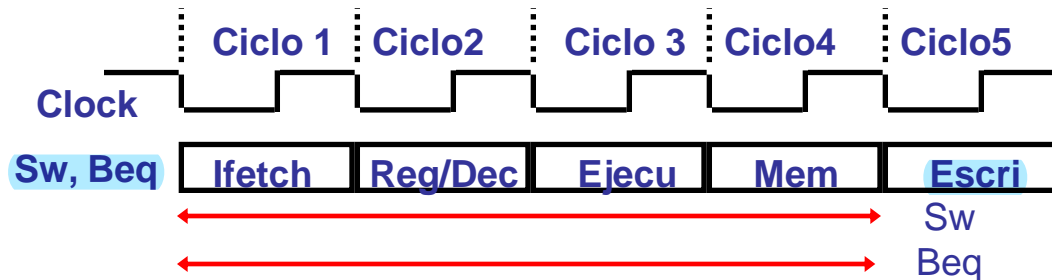
- ✓ Cada etapa del procesador puede usarse en cada ciclo por una sola instrucción.
- ✓ Cada instrucción debe usar cada etapa del procesador en la misma fase de ejecución.

❑ Solución.

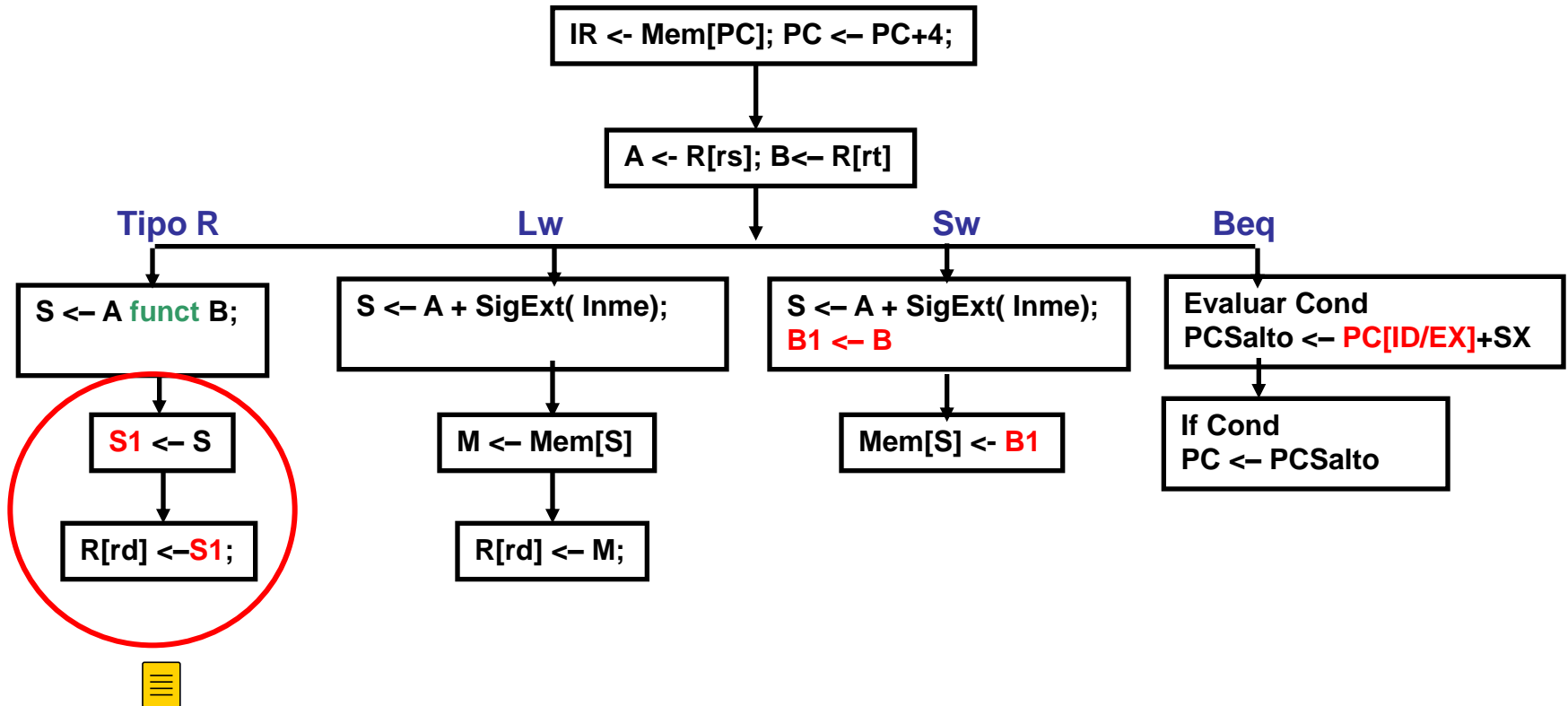
- ✓ Introducir una etapa (Mem) que no hace nada.



❑ Store 4 etapas con actividad. Branch cuatro etapas activas



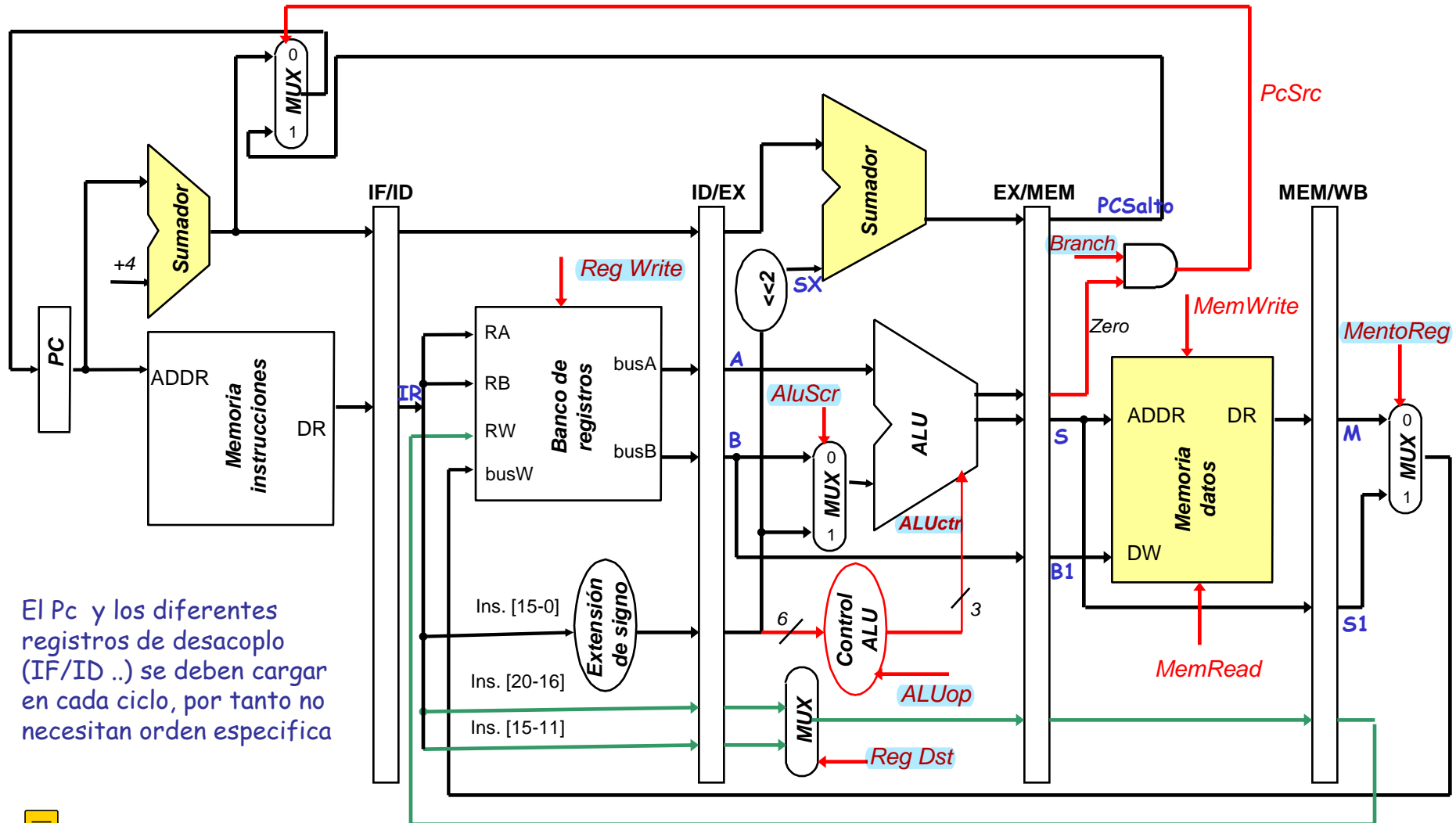
□ Ejecución de instrucciones: Diagrama RTL modificado





Diseño del control

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



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





Señales de control

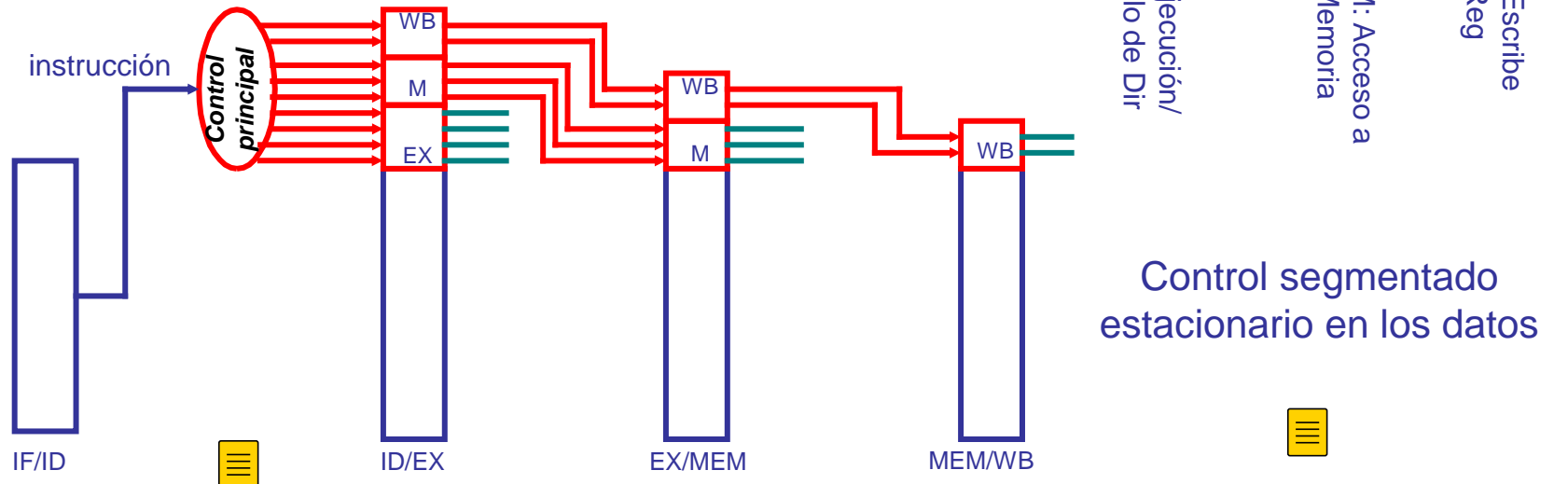
Control de la ALU

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

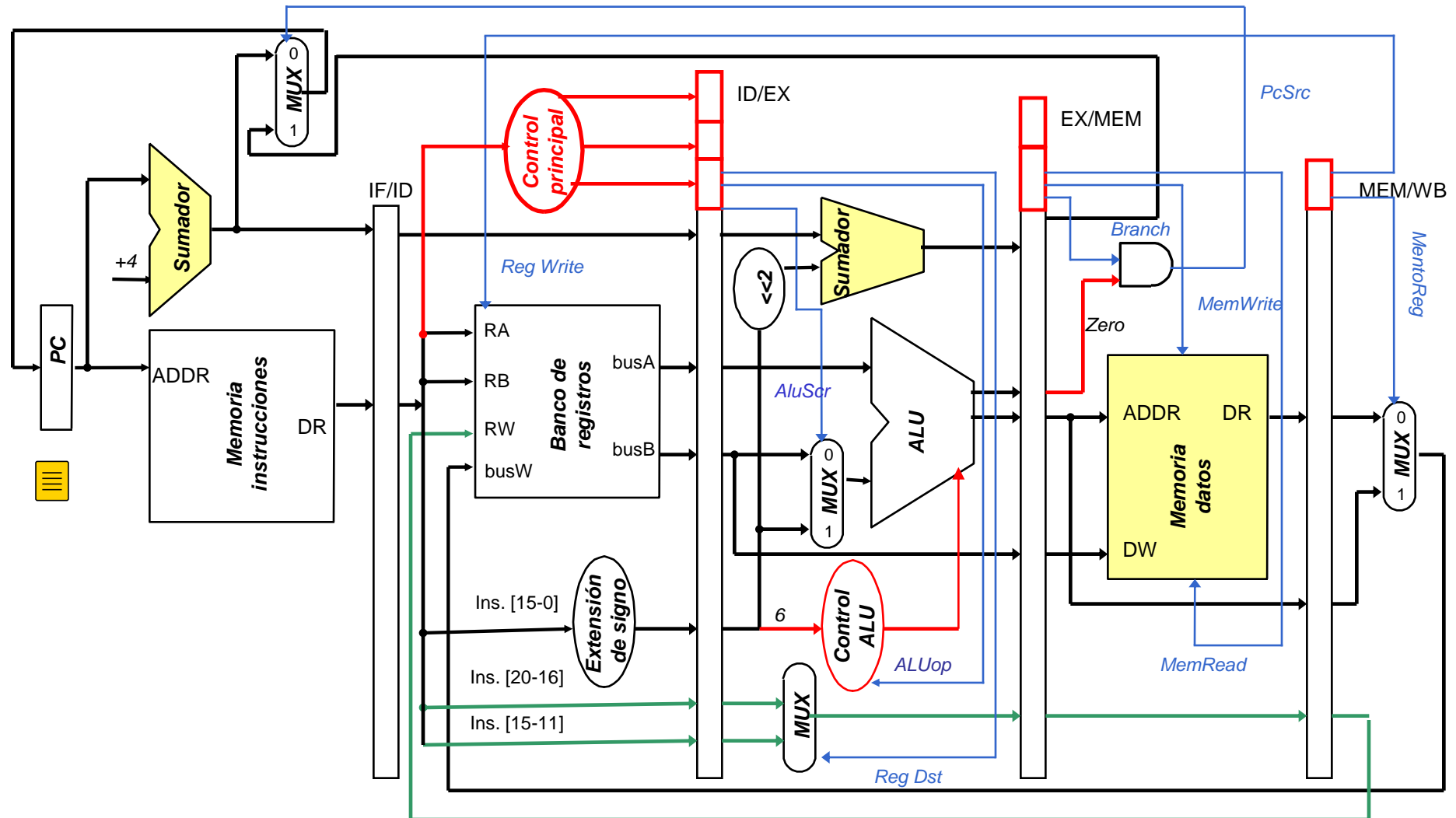
Control principal

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



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



❑ ¿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

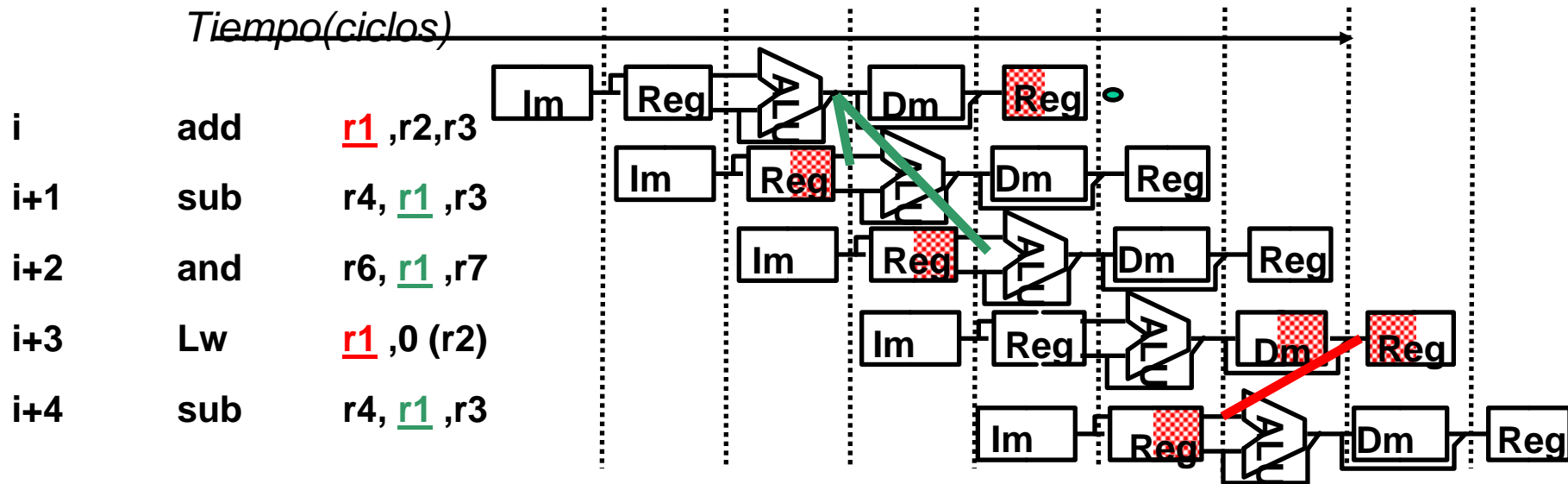
❑ El diseño anterior no tenia en cuenta los riesgos de datos y los riesgos de control los eliminaba con saltos retardados

- ✓ Implementación del cortocircuito
- ✓ Caso del load
- ✓ Mejora en el comportamiento de los saltos.

Diseño del control con riesgos

❑ Riesgos de datos $R(i) \cap D(i+?) \neq \emptyset$ LDE (RAW).

✓ Cortocircuito. Enviar el dato a las etapas que lo necesitan, cuanto esta listo

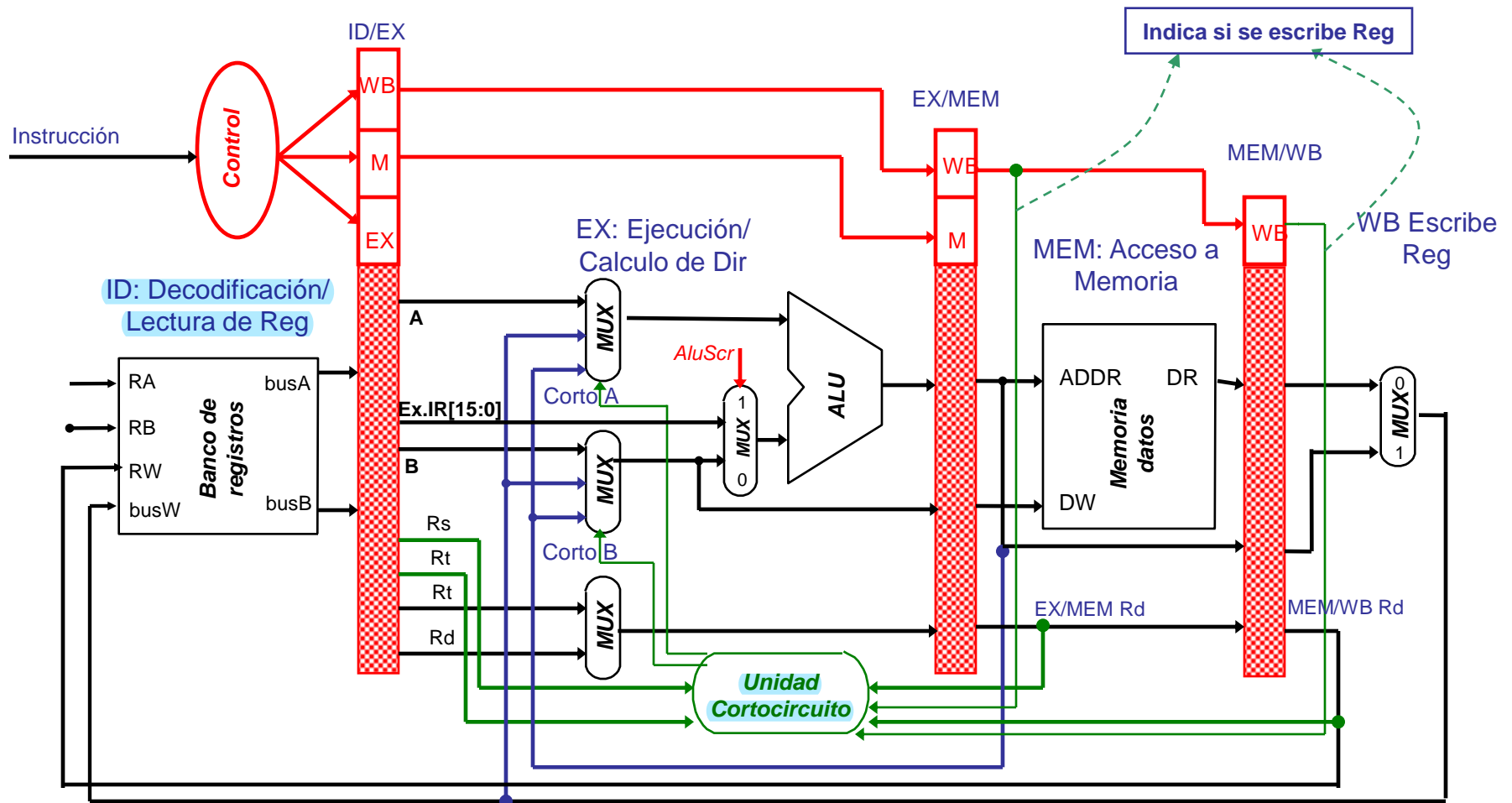


Dos caminos de datos: Desde salida ALU (EX/MEM) a entrada ALU
 Desde la salida de la memoria (MEM/WB) a entrada ALU
 Información necesaria: Registro a escribir en ultima etapa (Rd en Tipo-R y Rt en Lw)
 Registros que se leen en segunda etapa (Rs y Rt)

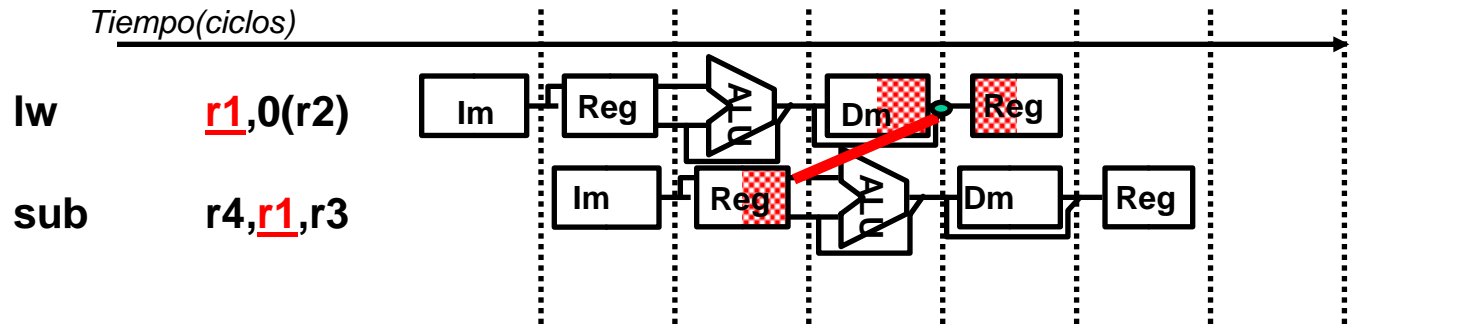
Diseñando el control con riesgos

❑ Riesgos de datos LDE: Implementación del cortocircuito

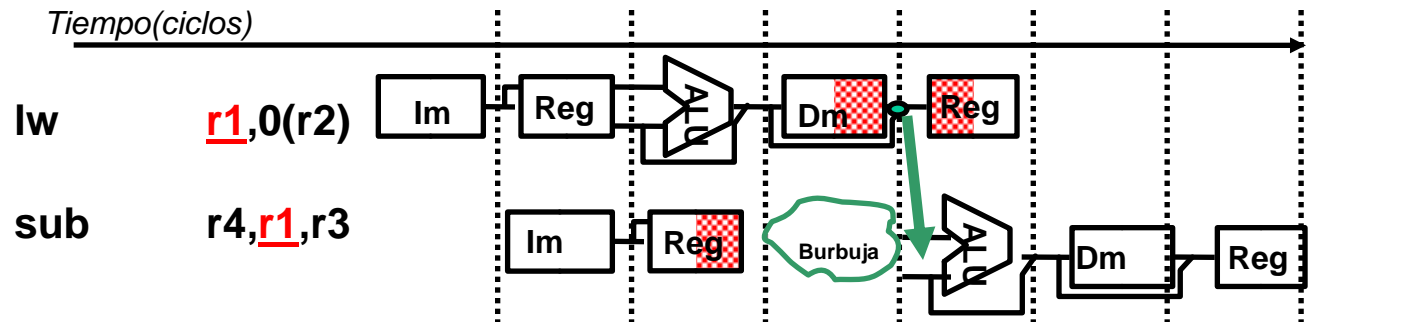
- ✓ Cortocircuitos de datos
- ✓ Información de control del cortocircuito



❑ Riesgos de datos LDE: Caso del load



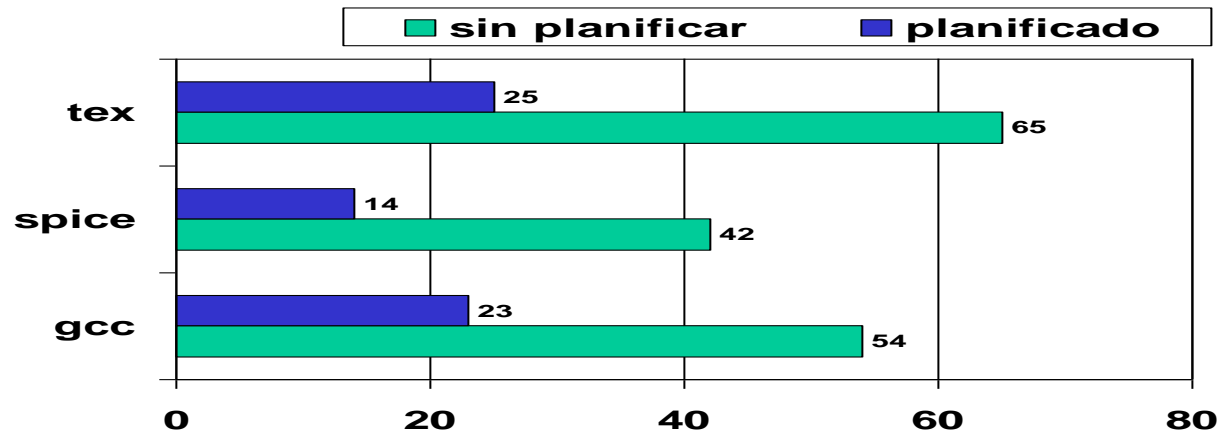
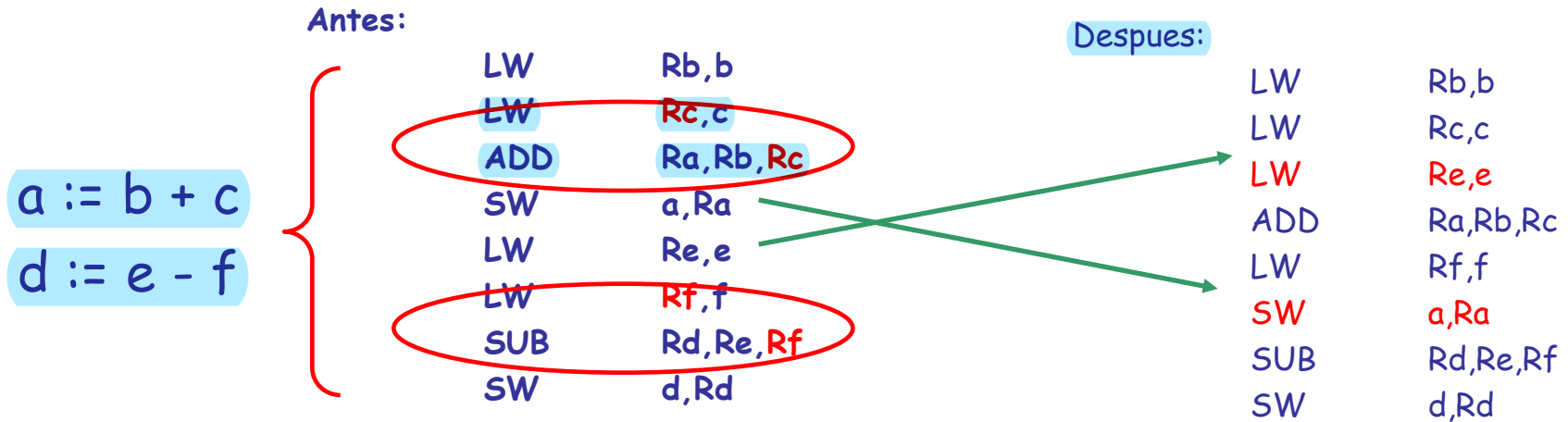
Se debe esperar un ciclo a pesar del cortocircuito



Diseño del control con riesgos

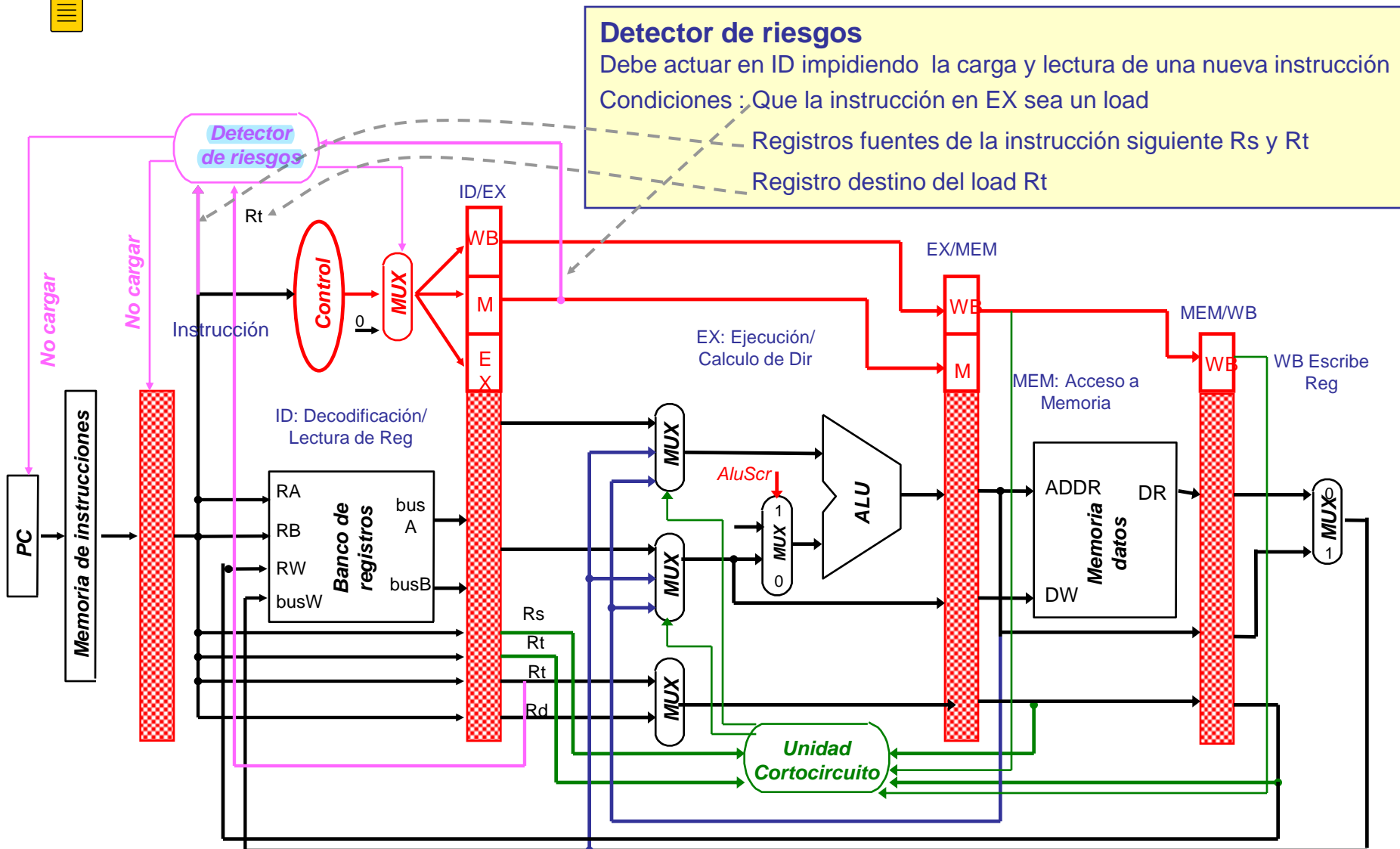
❑ Riesgos de datos LDE: Caso del load

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



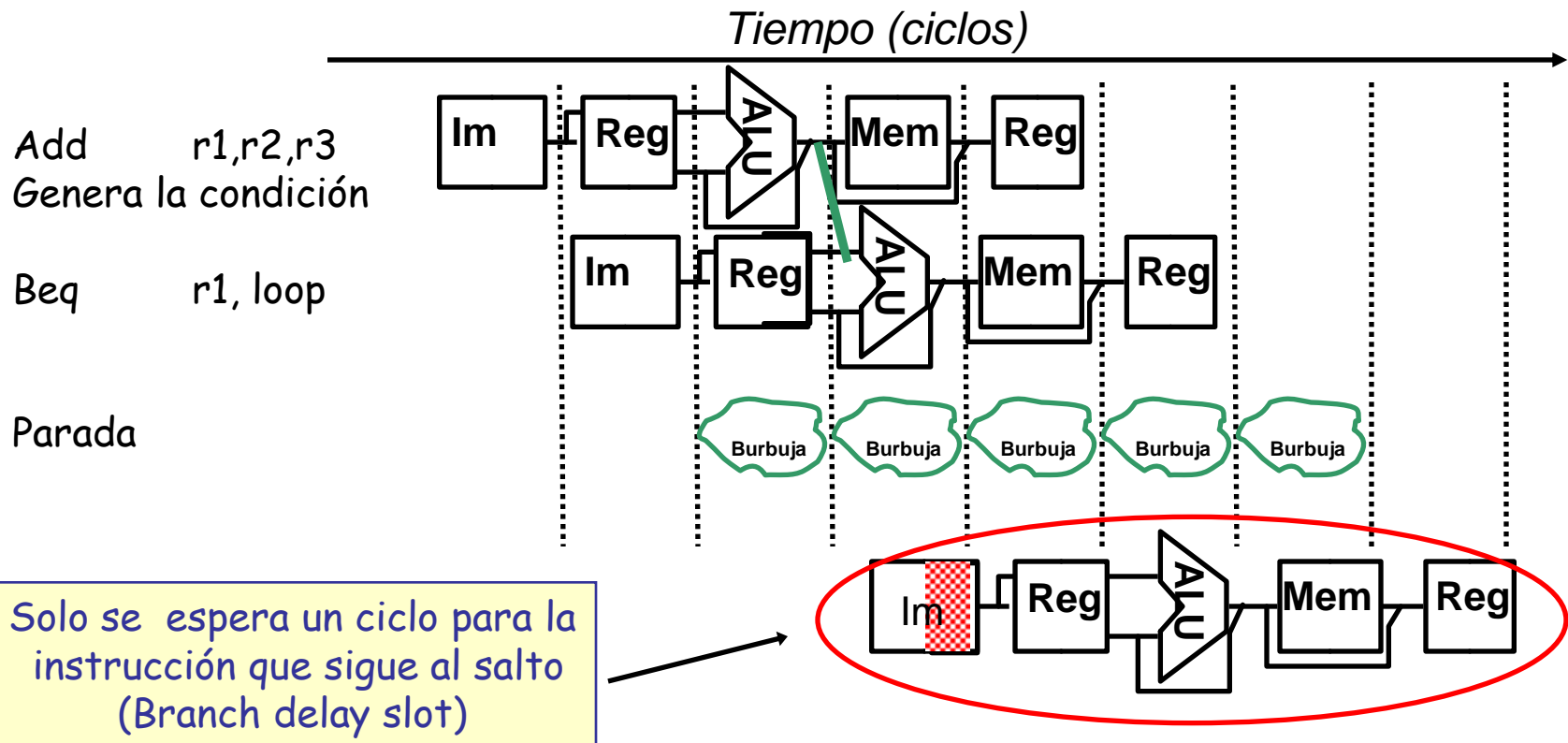
Diseño del control con riesgos

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



❑ Riesgos de control

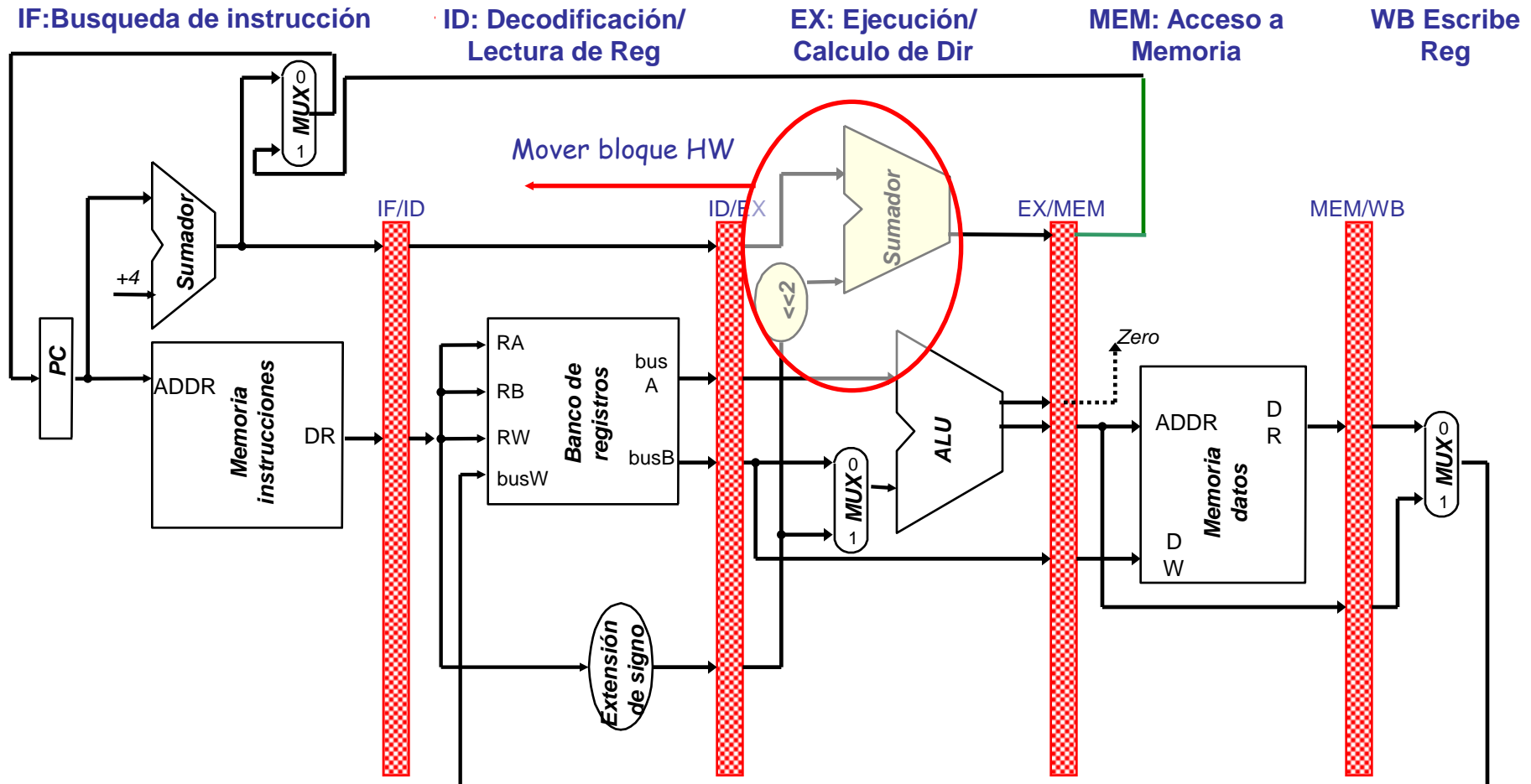
- ✓ Solución: Desplazar el calculo de la dirección y la evaluación de la condición a la etapa anterior



Diseño del control con riesgos

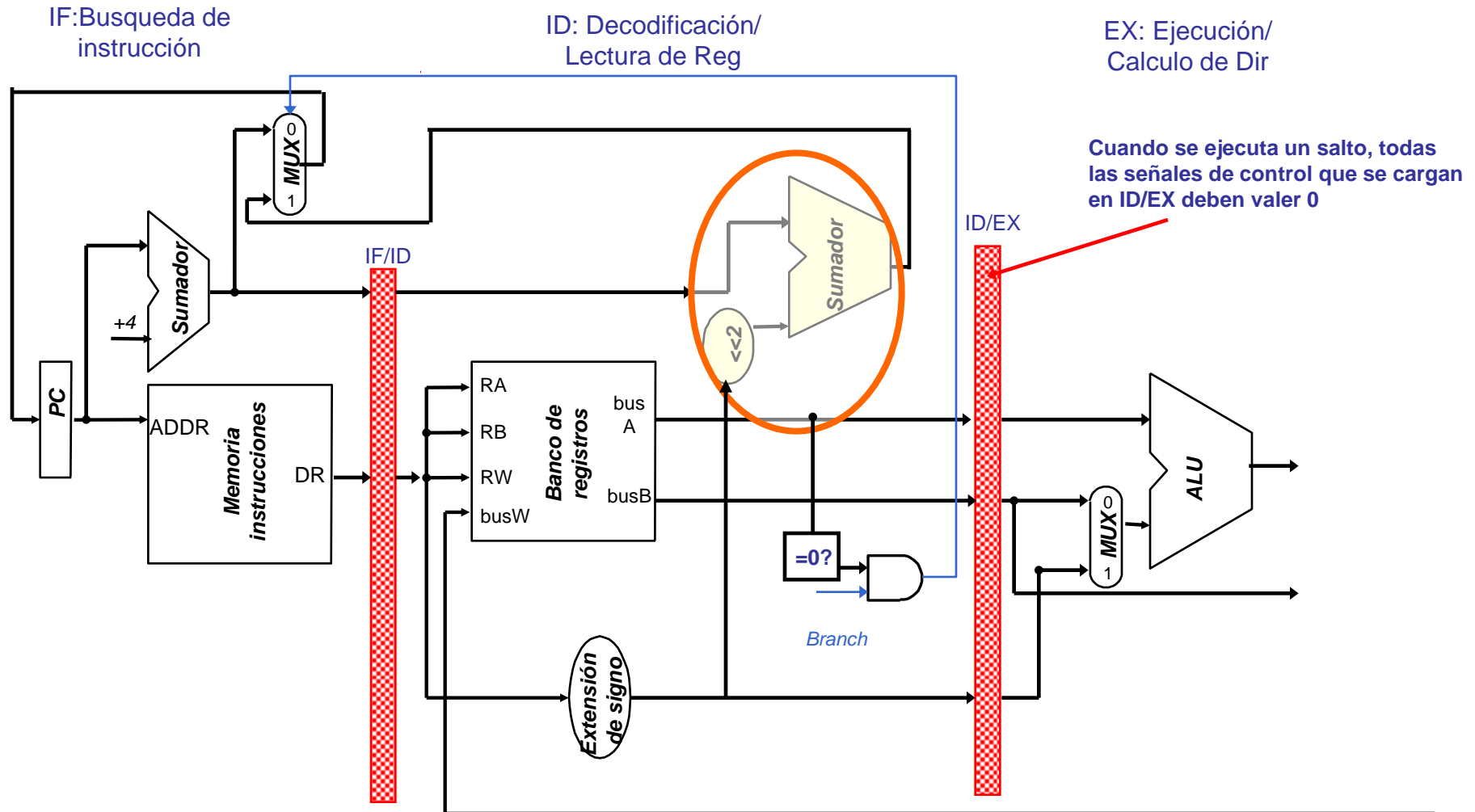
Mejora del comportamiento de los saltos. Solo un ciclo de parada

- ✓ Cálculo de la dirección. Operandos disponibles (Pc y desplazamiento)
- ✓ Cálculo de la condición. Unidad de detección de cero



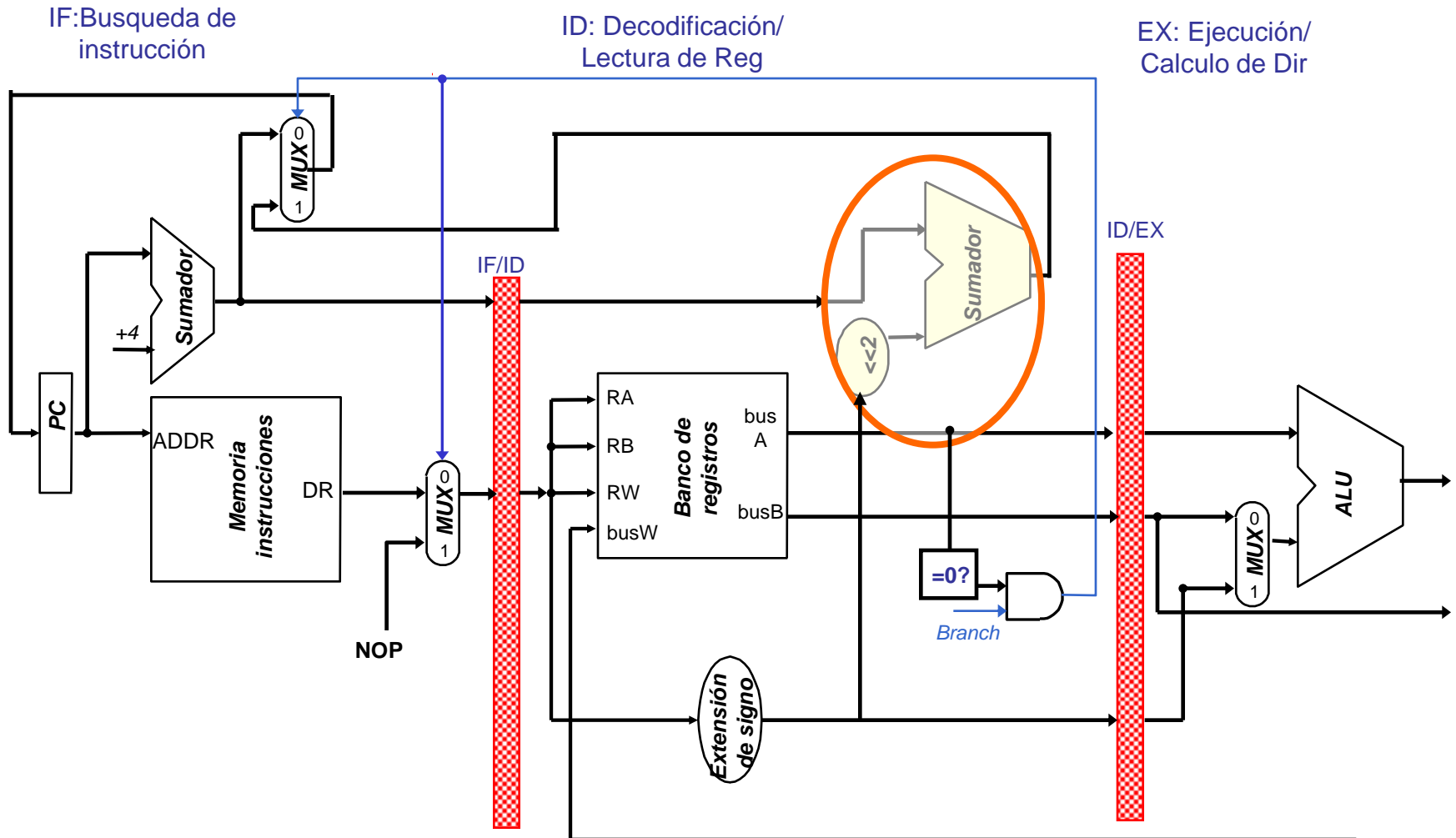
Saltos retardados

- 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)



□ EL CPI ideal es 1

□ Hay perdidas de rendimiento por las paradas del pipe

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

□ Caso de los saltos. Un programa tipico 30% de saltos

o $CPI = 1 + (1 \times 0.3) = 1.3$

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

o Se pierde un 24 % respecto al caso ideal:

$$\text{Speedup}_{\text{real}} / \text{Speedup}_{\text{ideal}} = 3.84 / 5 = 0.76$$

- ❑ Interrupciones, Excepciones, Fallos
 - o Síncronas - asíncronas
 - o Solicitadas por el programa - generadas por el programa
 - o Dentro de instrucciones - entre instrucciones
 - o 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)
 - o Ocurren en el medio de una instrucción
 - o Deben ser recomenzables

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

❑ 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

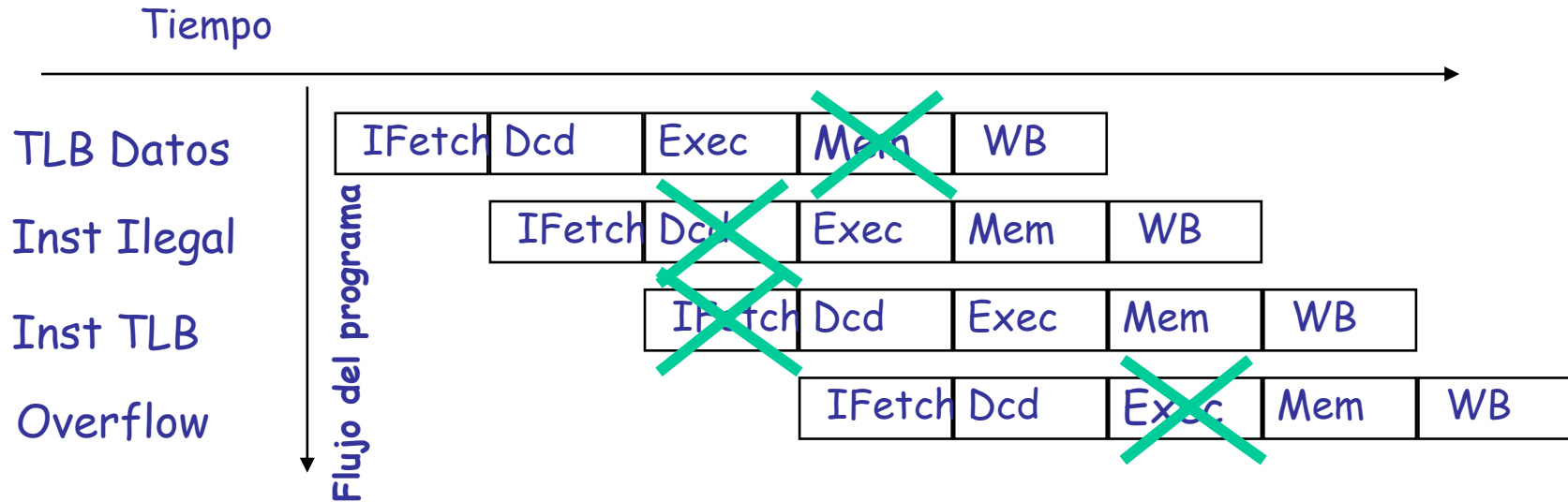
❑ Excepciones (fallo de pagina). Reinicio

- o Introducir una instrucción de trap en IF
- o Anular las escrituras de la instrucción que produce la excepción y las posteriores
- o Salvar el PC(se debe conservar hasta MEM) de la instrucción que produjo la excepción y pasar a la rutina de gestión

❑ Excepciones precisas:

- o Todas las instrucciones anteriores **completas**
- o La que produce la interrupción y las siguientes **como si no hubieran empezado**

Excepciones



- ❑ 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

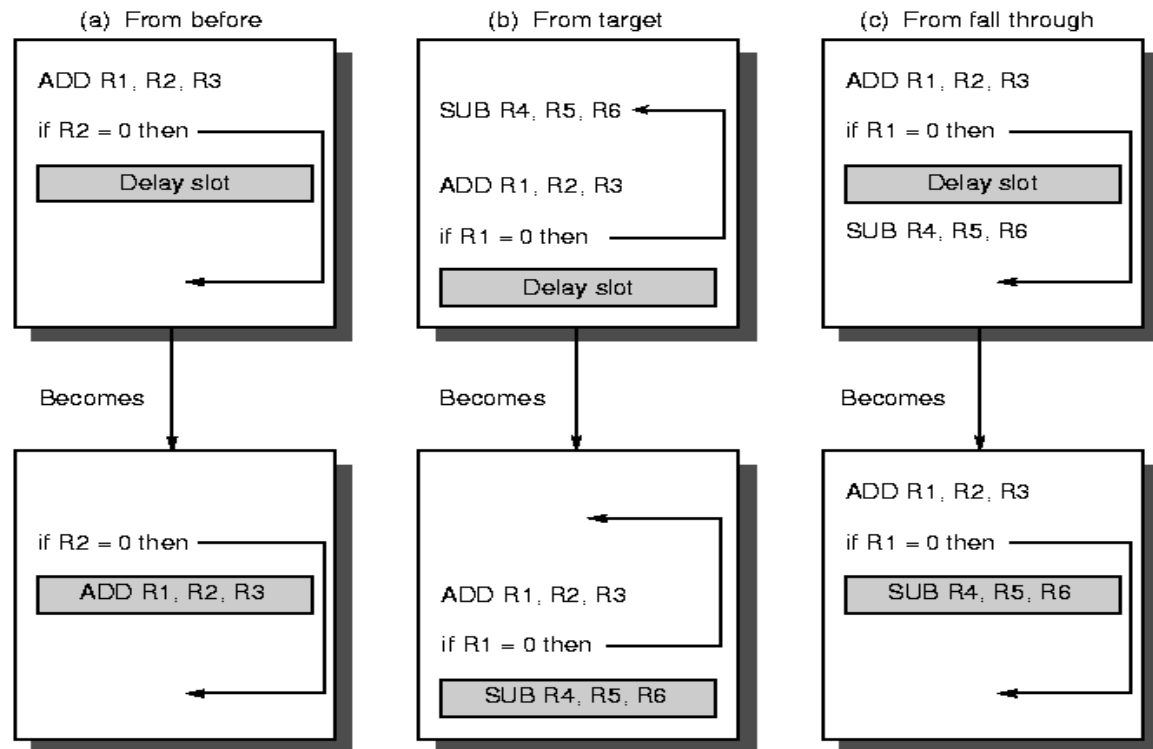
Fundamento de los saltos retardados: Ejecutar instrucciones *independientes del salto* durante los ciclos de retardo

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).



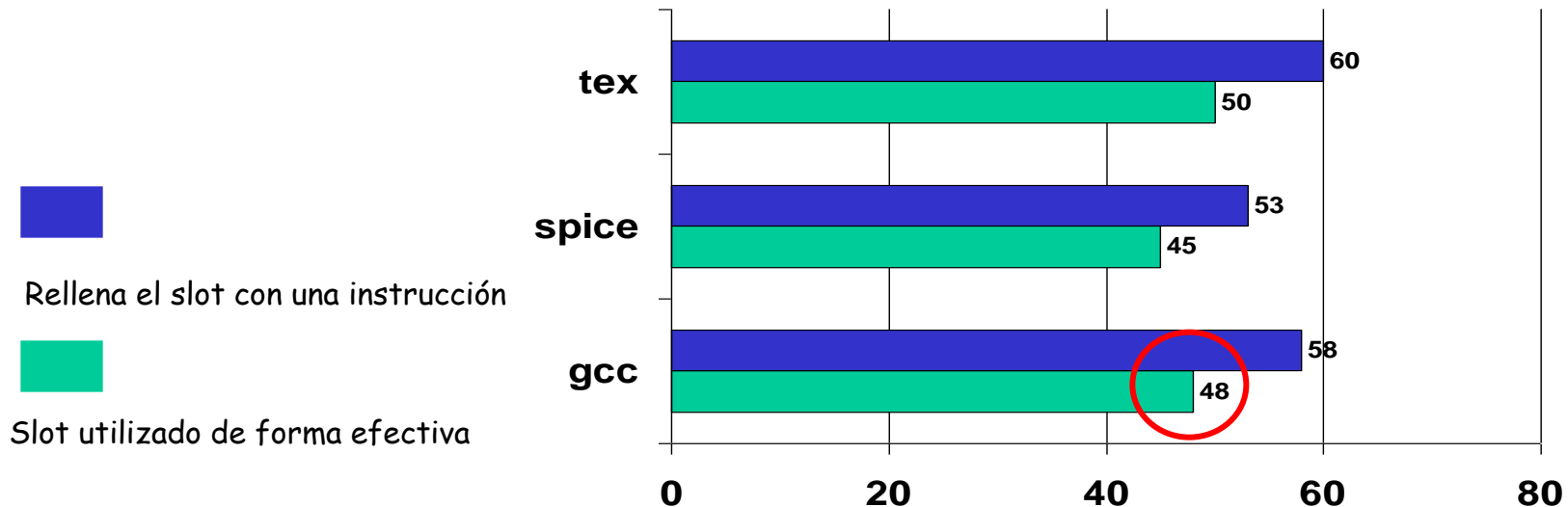
Saltos retardados: benchmarks

□ 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

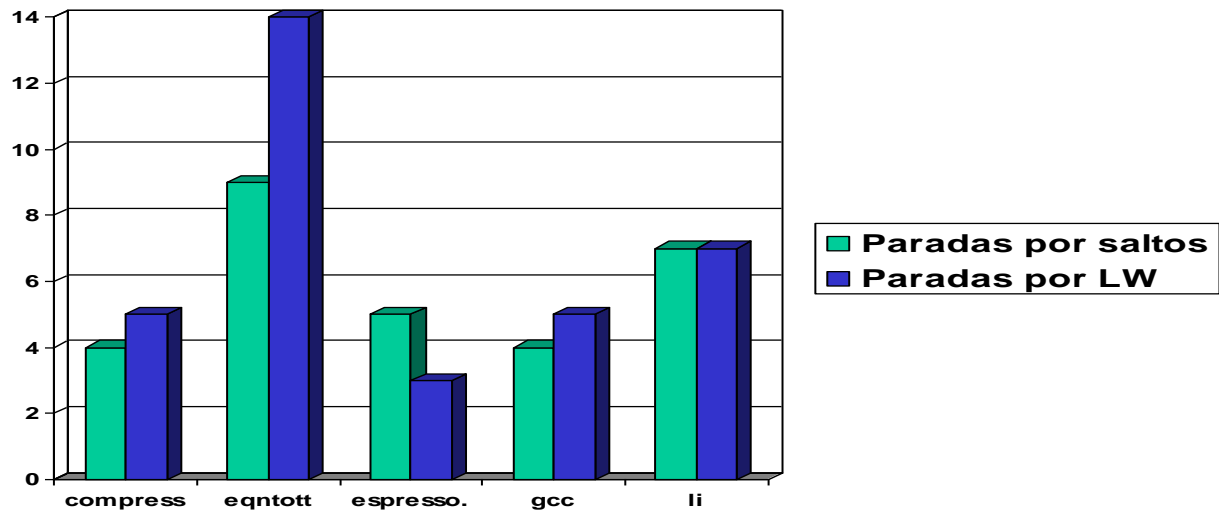
$$\text{CPI} = 1 + (1 - 0.48) \times 0.22 = 1.11$$



❑ 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
- ✓ Riesgo 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

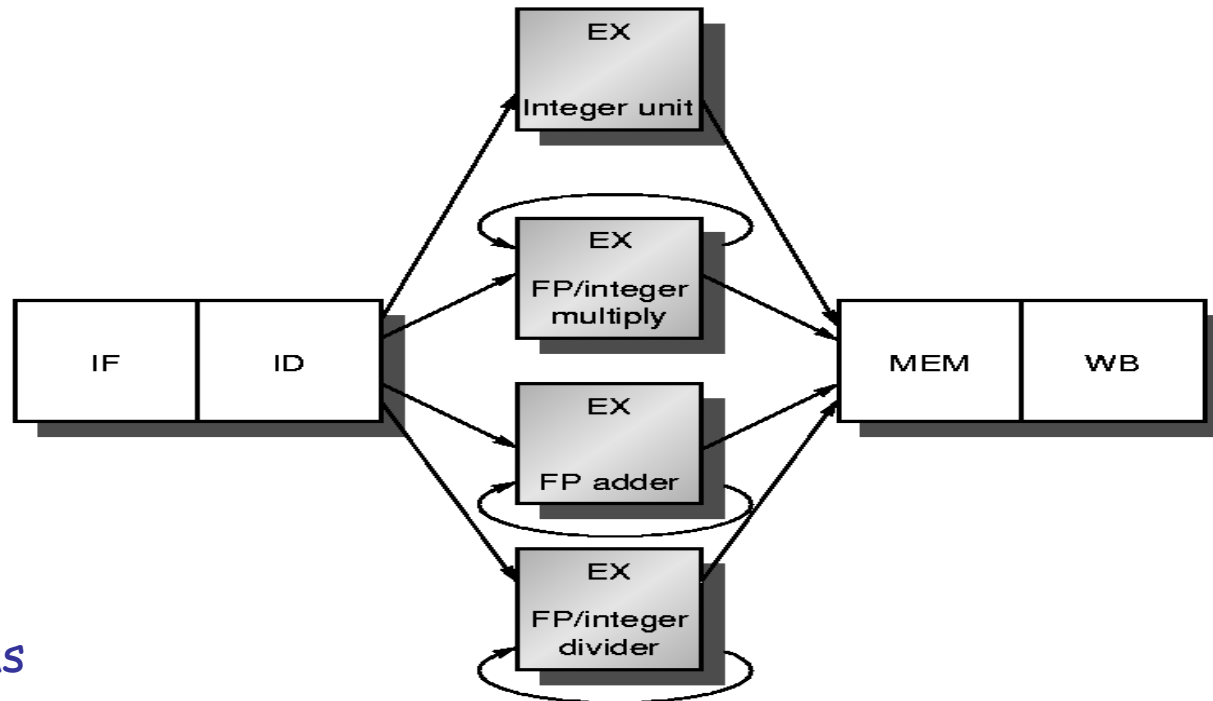


¿ Que ocurre si las instrucciones tienen diferentes duración?
Instrucciones de aritmética en punto flotante

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



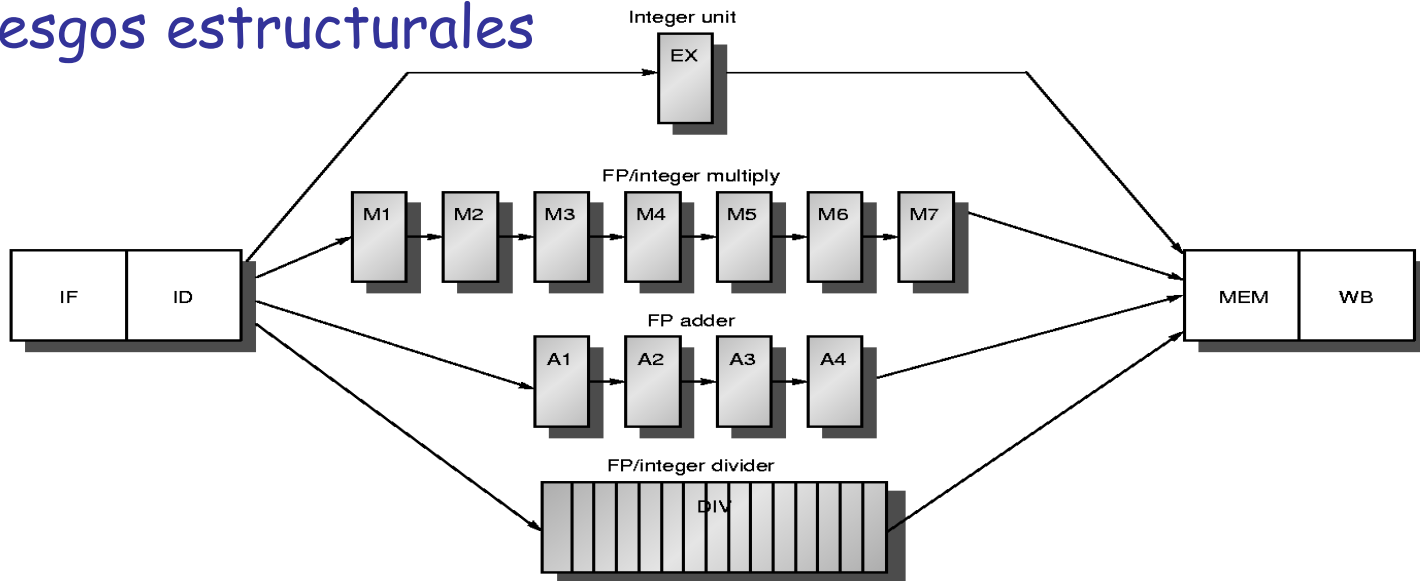
❑ Problemas

- o Riesgos estructurales
- o Mayor penalización de los riesgos LDE
- o Problemas con la finalización fuera de orden

❑ Solapamiento operaciones enteras y PF

- o No hay problemas operandos independientes

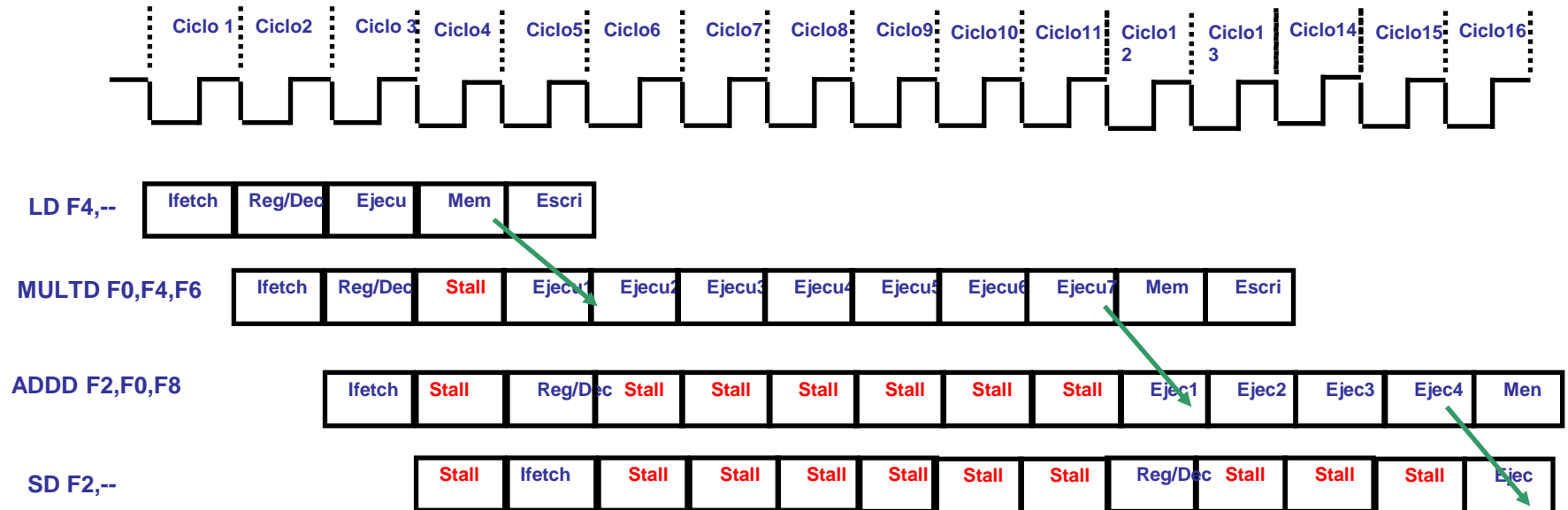
❑ Riesgos estructurales



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

❑ Riesgos LDE (RAW)

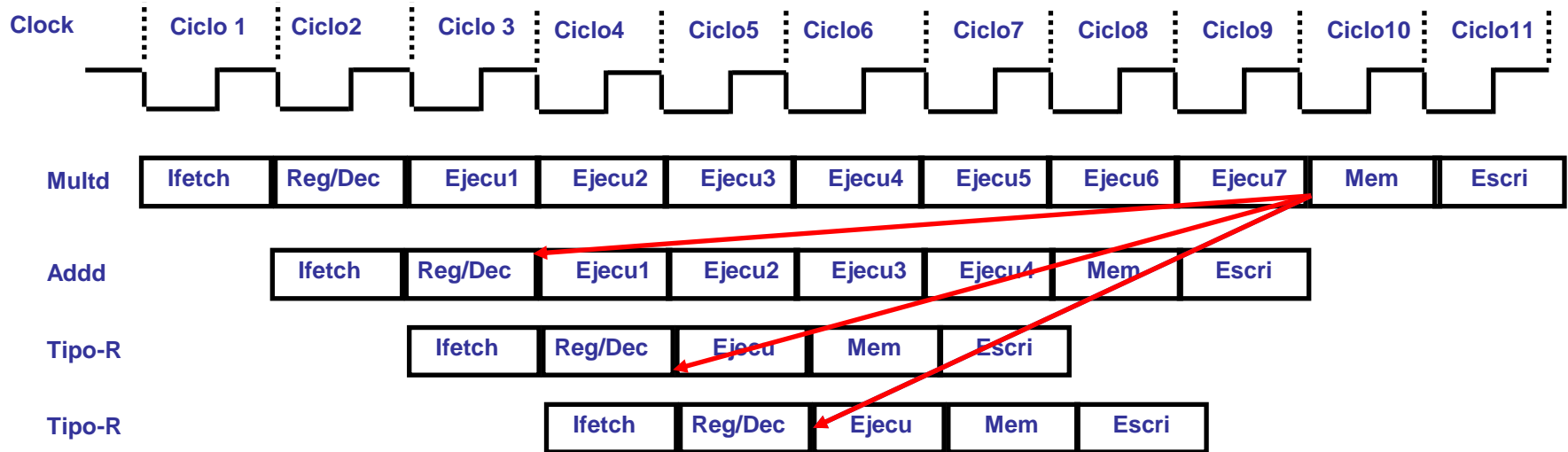


❑ Mayor impacto en el rendimiento

- o La gran duración de las instrucciones implica más ciclos de detención
- o Es necesaria una planificación más cuidadosa de las instrucciones

Operaciones multiciclo

❑ Finalización fuera de orden



❑ Las instrucciones acaban en orden distinto al lanzamiento

❑ Problemas

- o Conflictos por escritura simultanea en registros
- o Aparición de riesgos EDE
- o Problema con las excepciones

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

❑ 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.

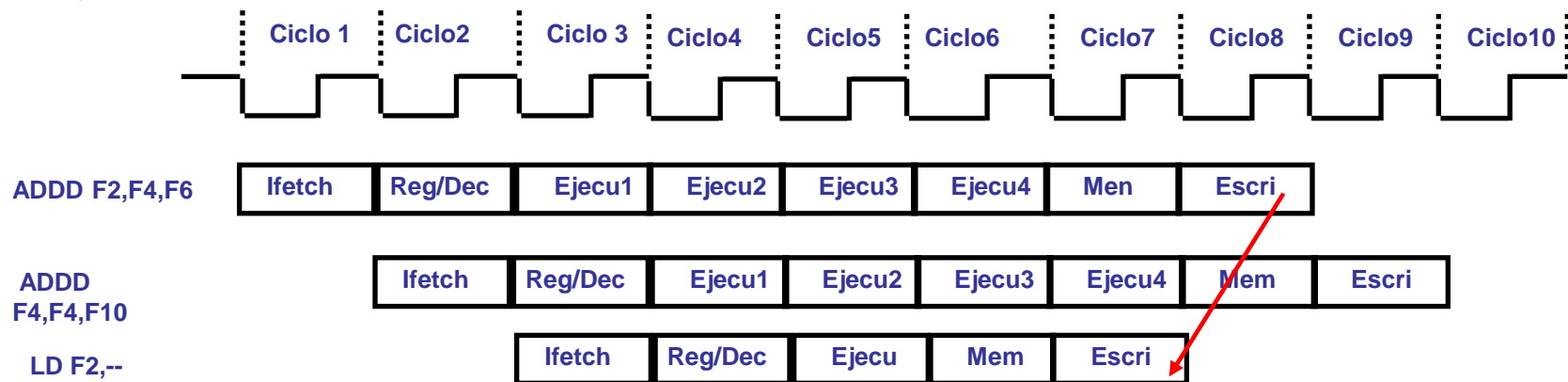
Se puede usar un registro de desplazamiento para indicar cuándo usarán el bloque de registros las instrucciones ya lanzadas.

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

□ Aparición de riesgos EDE



□ LD F2,0(R2) escribe en F2 antes que ADDD F2,F4,F6 → error

- o Situación poco común: Instrucción `ADDD F2,F4,F6` inútil. Puede ocurrir con instrucciones que ocupen un hueco de retardo
- o Si la segunda `ADDD` lee F2 → riesgo LDE que elimina el EDE
- o Riesgos EDL(WAR) No aparecen porque las lecturas de registros son en orden etapa Reg/Dec

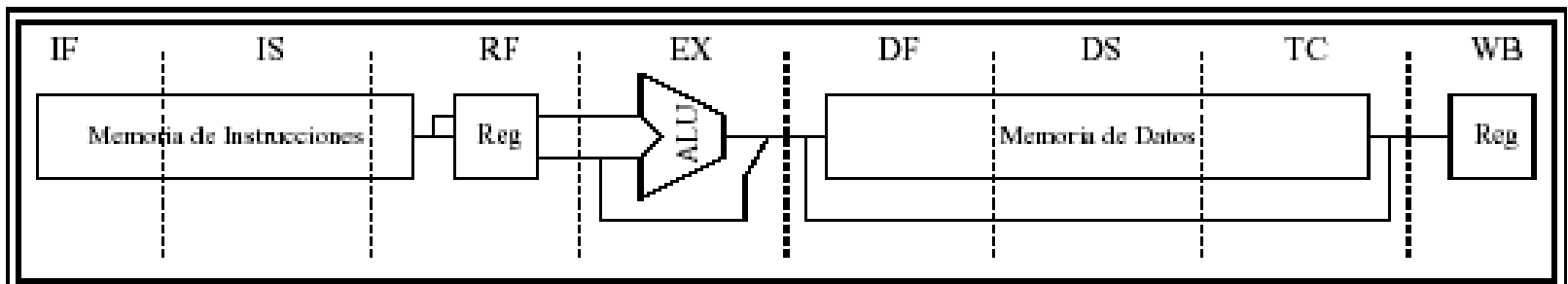
□ Tratamiento de riesgos EDE WAW:

- o Detener la instrucción que produce el riesgo (LD en el ejemplo)
 - o Eliminar la escritura de la primera instrucción (`ADDD` en el ejemplo)
- Situaciones poco comunes. No implica demasiado problema cualquiera de las aproximaciones

❑ Más segmentación 8 etapas

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

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



❑ Más penalización en cargas y saltos

Dos ciclos de latencia en los Load con cortocircuito

IF	IS	RF	EX	DF	DS	TC	WB
	IF	IS	RF	EX	DF	DS	TC
		IF	IS	RF	EX	DF	DS
			IF	IS	RF	EX	DF
				IF	IS	RF	EX
					IF	IS	RF
						IF	IS
							IF

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

IF	IS	RF	EX	DF	DS	TC	WB	
	IF	IS	RF	EX	DF	DS	TC	(*)
		IF	IS	RF	EX	DF	DS	(**)
			IF	IS	RF	EX	DF	(**)
			IF	IS	RF	EX	DF	
				IF	IS	RF	EX	
					IF	IS	RF	
						IF	IS	
							IF	

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

(*) Continúa la ejecución en el caso

(**) Se abortan en caso de que el salto se tome

❑ Operaciones en PF

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

<i>Etapas</i>	<i>Unidad</i>	<i>Descripción</i>
A	FP adder	Etapas de suma de mantisas
D	FP divider	Etapas de división
E	FP multiplier	Etapas de test de excepciones
M	FP multiplier	1ª etapa del multiplicador
N	FP multiplier	2ª etapa del multiplicador
R	FP adder	Etapas de redondeo
S	FP adder	Etapas de desplazamiento
U		Etapas de desempaqueado

❑ Operaciones en PF: Implementación

<i>Instr PF</i>	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

□ Rendimiento

□ No se alcanza el rendimiento ideal CPI= 1:

- o Load stalls (1 o 2 ciclos)
- o Saltos (2 ciclos + slots no rellenos)
- o FP stalls (resultados): riesgo de LDE (latencia)
- o FP stalls (estructurales): Hw FP muy escaso (paralelismo)

