

## UT 2. Computadores segmentados

### Tema 2.1 Unidad de instrucción segmentada

J. Flich, P. López, V. Lorente,  
A. Pérez, S. Petit, J.C. Ruiz, S. Sáez, J. Sahuquillo

Departamento de Informática de Sistemas y Computadores  
Universitat Politècnica de València



**DOCENCIA VIRTUAL**

**Finalidad:**  
Prestación del servicio Público de educación superior (art. 1 LOU)

**Responsable:**  
Universitat Politècnica de València.

**Derechos de acceso, rectificación, supresión, portabilidad, limitación u oposición al tratamiento conforme a políticas de privacidad:**  
<http://www.upv.es/contenidos/DPD/>

**Propiedad intelectual:**  
Uso exclusivo en el entorno de aula virtual.  
Queda prohibida la difusión, distribución o divulgación de la grabación de las clases y particularmente su compartición en redes sociales o servicios dedicados a compartir apuntes.  
La infracción de esta prohibición puede generar responsabilidad disciplinaria, administrativa o civil

 UNIVERSITAT POLITÈCNICA DE VALÈNCIA





## Índice

- 1 Concepto de segmentación
- 2 El ciclo de instrucción
- 3 Segmentación del ciclo de instrucción
- 4 Riesgos
- 5 Riesgos de datos
- 6 Riesgos de control
- 7 Riesgos estructurales
- 8 Excepciones

### Bibliografía

 John L. Hennessy and David A. Patterson.

*Computer Architecture, Fifth Edition: A Quantitative Approach.*  
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5  
edition, 2012.

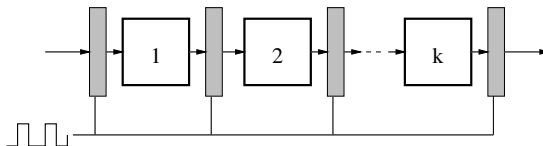
## Índice

- 1 Concepto de segmentación
- 2 El ciclo de instrucción
- 3 Segmentación del ciclo de instrucción
- 4 Riesgos
- 5 Riesgos de datos
- 6 Riesgos de control
- 7 Riesgos estructurales
- 8 Excepciones

# 1. Concepto de segmentación

## Segmentación

- Técnica que descompone un proceso en varios subprocesos
- Cada subproceso puede ejecutarse independientemente en un módulo autónomo
- Cada módulo opera concurrentemente con los demás.
- Un sistema se segmenta en  $k$  etapas:



→ Etapa: módulo que procesa un subproceso  
+  
registro de tipo *latch*

# 1. Concepto de segmentación

## Segmentación (cont.)

### ■ **Registros inter-etapa:**

Mantienen los datos estables durante el tiempo que necesita el módulo para hacer su función.

### ■ Un reloj sincroniza el avance de los datos por las etapas. El reloj marca:

- cuándo puede entrar un nuevo dato en la unidad segmentada.
- el tiempo que las etapas disponen para efectuar su función.

# 1. Concepto de segmentación

## Segmentación (cont.)

### ■ Periodo de reloj

- Caso ideal: mismo retardo en todos módulos.

$$\tau = \frac{D}{k}$$

- $D$ : Retardo del circuito original.
- $k$ : Número de etapas.
- Caso real: módulos con retardos distintos, registros inter-etapa y desfase del reloj.

$$\tau = \max_{i=1}^k(\tau_i) + T_R + T_S \geq \frac{D}{k}$$

- $\tau_i$ : Retardo del módulo  $i$ .
- $T_R$ : Retardo del registro inter-etapa.
- $T_S$ : Desfase del reloj.

# 1. Concepto de segmentación

## Mejora obtenida al segmentar

### ■ **Aceleración** (*Speedup*):

$$S = \frac{T_{ns}}{T_s} \approx \frac{nD}{n\tau} = \frac{D}{\tau}$$

- $T_{ns}$ : Tiempo para procesar  $n$  datos en unidad original.
- $T_s$ : Idem en la unidad segmentada.
- Caso ideal:  $\tau = \frac{D}{k}$ .

$$S = \frac{D}{\tau} = k$$

- Caso real:  $\tau \geq \frac{D}{k}$ .

$$S = \frac{D}{\tau} \leq k$$



# 1. Concepto de segmentación

## Mejora obtenida al segmentar (cont.)

### ■ Productividad

- Expresión general:

$$\chi = \frac{n}{T}, \text{ siendo:}$$

- $n$ : datos procesados.

- $T$ : Tiempo necesario para procesar  $n$  datos.

- Unidad no segmentada:

$$\chi_{ns} = \frac{n}{T_{ns}} = \frac{n}{nD} = \frac{1}{D} \text{ resultados/s}$$

- Unidad segmentada:

$$\chi_s = \frac{n}{T_s} \approx \frac{n}{n\tau} = \frac{1}{\tau} \text{ resultados/s}$$

→ 1 resultado cada  $\tau$  segundos = 1 resultado/ciclo de reloj.

## Índice

- 1 Concepto de segmentación
- 2 El ciclo de instrucción
- 3 Segmentación del ciclo de instrucción
- 4 Riesgos
- 5 Riesgos de datos
- 6 Riesgos de control
- 7 Riesgos estructurales
- 8 Excepciones

## 2. El ciclo de instrucción

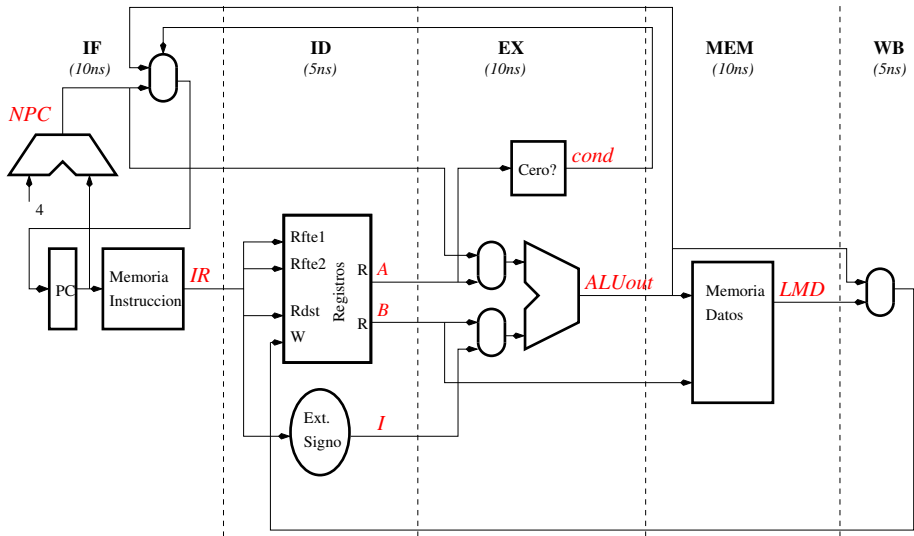
### Computador de ejemplo: MIPS reducido

- Instrucciones aritméticas (reg-reg y reg-imm). Ejemplos: `dadd Rdst, Rfte1, Rfte2` y `daddi Rdst, Rfte1, Imm`
- Cargas y almacenamientos. Ejemplos: `ld Rdst, Desp(Rfte1)` y `sd Rfte2, Desp(Rfte1)`
- Saltos condicionales: `beqz Rfte1, Desp` y `bnez Rfte1, Desp`

	0 5	6 10	11 15	16 20	21 31
R	Codop	R <sub>fte1</sub>	R <sub>fte2</sub>	R <sub>dst</sub>	Func (op)
	⇒ Instr. UAL reg-reg: R <sub>dst</sub> ← R <sub>fte1</sub> op R <sub>fte2</sub>				
	0 5	6 10	11 15	16 31	
I	Codop	R <sub>fte1</sub>	R <sub>fte2</sub> /dst	Imm/Desp	
	⇒ Instr. UAL reg-imm: R <sub>dst</sub> ← R <sub>fte1</sub> op Imm				
	⇒ Instr. Carga: R <sub>dst</sub> ← M[R <sub>fte1</sub> +Desp]				
	⇒ Instr. Almac.: M[R <sub>fte1</sub> +Desp] ← R <sub>fte2</sub>				
	⇒ Instr. Salto cond.: if (R <sub>fte1</sub> =, ≠ 0) then PC ← PC+4+Desp				

## 2. El ciclo de instrucción

## Ruta de datos del MIPS



## 2. El ciclo de instrucción

### Ciclo de instrucción del MIPS

Fase	Instrucciones ejemplo			
	dadd Rdst,Rftel,Rfte2 daddi Rdst,Rftel,Imm	ld Rdst,Desp(Rftel)	sd Rfte2,Desp(Rftel)	beqz Rftel,Desp
IF (10ns)	Leer instrucción en Memoria: $IR \leftarrow Mem[PC]$ ; Calcular NPC: $NPC \leftarrow PC+4$ ; Siguiente instrucción: if cond then $PC \leftarrow ALUout$ else $PC \leftarrow NPC$ ;			
ID (5ns)	Decodificar instrucción Leer operandos en Banco de Registros: $A \leftarrow Regs[Rfte1]$ ; $B \leftarrow Regs[Rfte2]$ ; Extender signo del campo inmediato/desplazamiento: $I \leftarrow Ext.Signo(Imm/Desp)$ ;			
EX (10ns)	Calcular resultado: $ALUout \leftarrow A+(B \text{ o } I)$ ;	Calcular dirección: $ALUout \leftarrow A+I$ ;		Calcular destino y condición: $ALUout \leftarrow NPC+I$ ; $cond \leftarrow A=0?$ ;
MEM (10ns)		Leer dato en Memoria: $LMD \leftarrow Mem[ALUout]$ ;	Escribir dato en Memoria: $Mem[ALUout] \leftarrow B$ ;	
WB (5ns)	Escribir resultado en Banco de Registros: $Regs[Rdst] \leftarrow ALUout \text{ o } LMD$			

Tiempo de ciclo T:

- El tiempo de ciclo debe ser lo suficientemente largo para poder soportar la ejecución de la instrucción más larga.

$$T = \max \begin{pmatrix} \text{saltos} & \text{ual} & \text{almac.} & \text{carga} \\ 25, & 30, & 35, & 40 \end{pmatrix} = 40 \text{ ns}$$

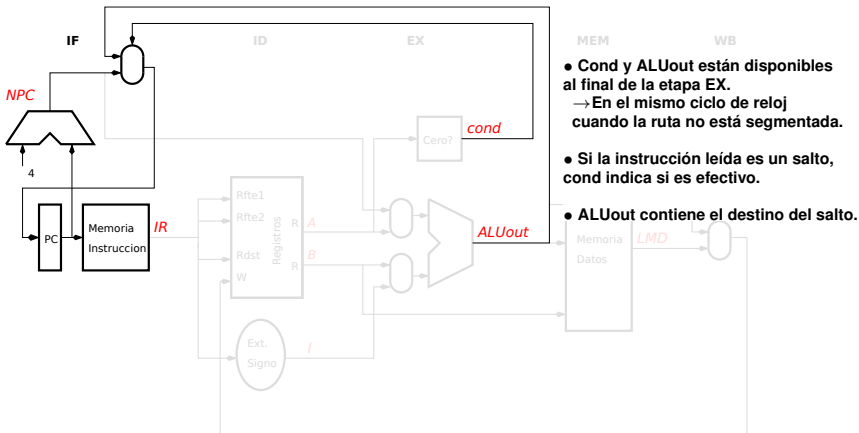
## Índice

- 1 Concepto de segmentación
- 2 El ciclo de instrucción
- 3 Segmentación del ciclo de instrucción**
- 4 Riesgos
- 5 Riesgos de datos
- 6 Riesgos de control
- 7 Riesgos estructurales
- 8 Excepciones

### 3. Segmentación del ciclo de instrucción

#### Segmentación del ciclo de instrucción

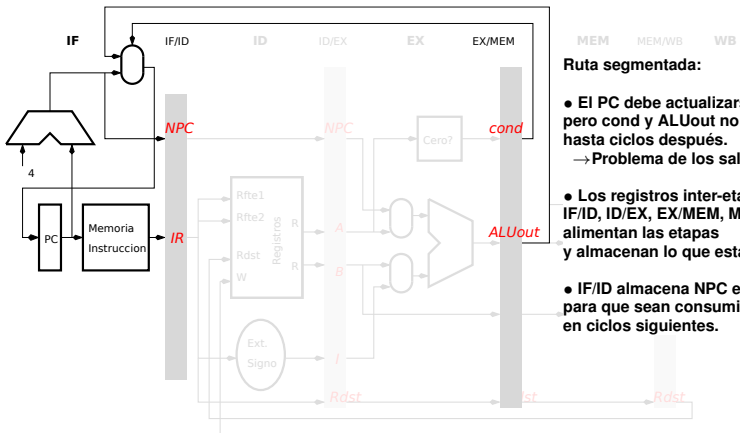
Etapa	Instrucciones ejemplo			
	dadd Rdst,Rfte1,Rfte2 daddi Rdst,Rfte1,Imm	ld Rdst,Desp(Rfte1)	sd Rfte2,Desp(Rfte1)	beqz Rfte1,Desp
IF	Leer instrucción en Memoria: $IR \leftarrow Mem[PC]$ ; Calcular NPC: $NPC \leftarrow PC+4$ ; Siguiente instrucción: if cond then $PC \leftarrow ALUout$ else $PC \leftarrow NPC$ ;			



### 3. Segmentación del ciclo de instrucción

#### Segmentación del ciclo de instrucción

Etapa	Instrucciones ejemplo			
	dadd Rdst,Rfte1,Rfte2 daddi Rdst,Rfte1,Imm	ld Rdst,Desp(Rfte1)	sd Rfte2,Desp(Rfte1)	beqz Rfte1,Desp
IF	Leer instrucción en Memoria: <b>IF/ID.NPC</b> ← Mem[PC]; Calcular NPC: <b>IF/ID.NPC</b> ← PC+4; Siguiente instrucción: if <b>EX/MEM.cond</b> then PC ← <b>EX/MEM.ALUout</b> else PC ← NPC;			



#### Ruta segmentada:

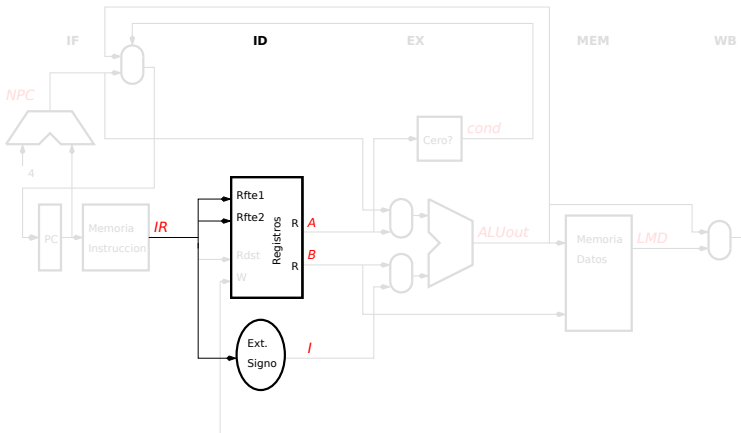
- El PC debe actualizarse cada ciclo, pero cond y ALUout no están disponibles hasta ciclos después.  
→ Problema de los saltos.
- Los registros inter-etapa IF/ID, ID/EX, EX/MEM, MEM/WB alimentan las etapas y almacenan lo que estas producen.
- IF/ID almacena NPC e IR para que sean consumidos en ciclos siguientes.



### 3. Segmentación del ciclo de instrucción

#### Segmentación del ciclo de instrucción

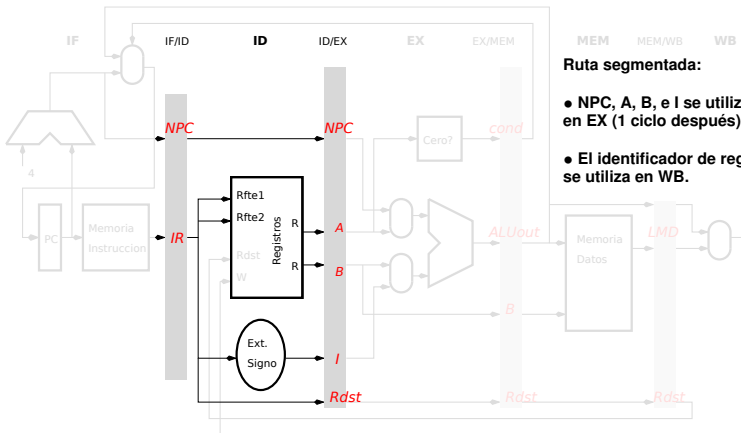
Etapa	Instrucciones ejemplo			
	dadd Rdst,Rfte1,Rfte2	daddi Rdst,Rfte1,Imm	ld Rdst,Desp(Rfte1)	sd Rfte2,Desp(Rfte1)
ID	Decodificar instrucción Leer operandos en Banco de Registros: <b>A</b> ← Regs[Rfte1]; <b>B</b> ← Regs[Rfte2]; Extender signo del campo inmediato/desplazamiento: <b>I</b> ← Ext.Signo(Imm/Desp);			



### 3. Segmentación del ciclo de instrucción

#### Segmentación del ciclo de instrucción

Etapa	Instrucciones ejemplo			
	dadd Rdst,Rfte1,Rfte2	daddi Rdst,Rfte1,Imm	ld Rdst,Desp(Rfte1)	sd Rfte2,Desp(Rfte1)    beqz Rfte1,Desp
ID	Decodificar instrucción Leer operandos en Banco de Registros: <b>ID/EX.A</b> ← Regs[ <b>ID/ID.IR.Rfte1</b> ]; <b>ID/EX.B</b> ← Regs[ <b>ID/ID.IR.Rfte2</b> ]; Extender signo del campo inmediato/desplazamiento: <b>ID/EX.I</b> ← Ext.Signo( <b>ID/ID.IR.Imm/Desp</b> );			



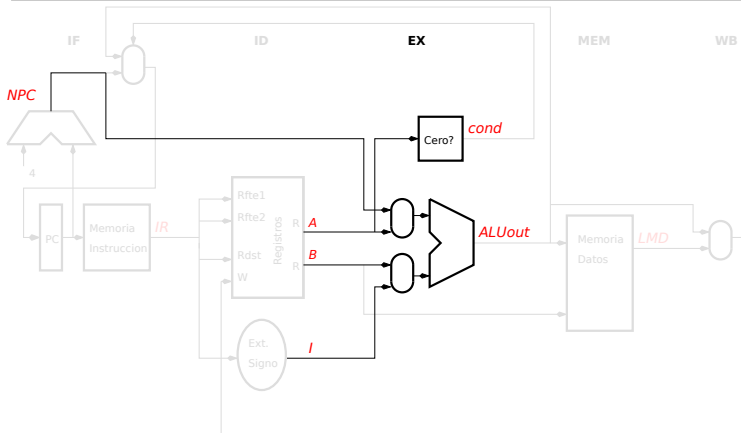
#### Ruta segmentada:

- NPC, A, B, e I se utilizan en EX (1 ciclo después).
- El identificador de registro destino (Rdst) se utiliza en WB.

### 3. Segmentación del ciclo de instrucción

#### Segmentación del ciclo de instrucción

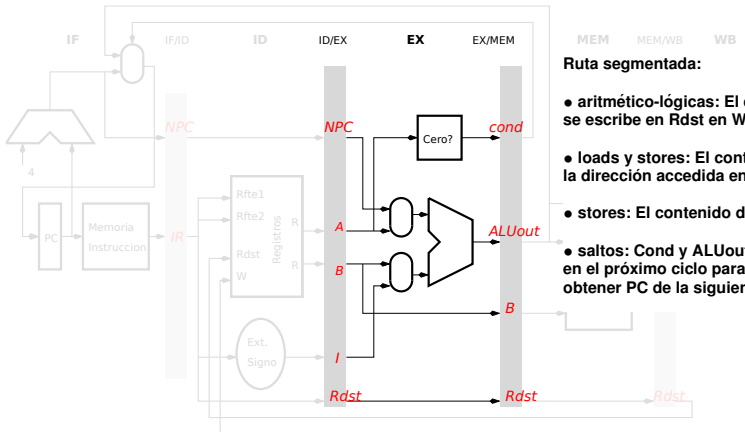
Etapa EX	Instrucciones ejemplo			
	dadd Rdst,Rfte1,Rfte2 daddi Rdst,Rfte1,Imm	ld Rdst,Desp(Rfte1)	sd Rfte2,Desp(Rfte1)	beqz Rfte1,Desp
	Calcular resultado: <b>ALUout</b> $\leftarrow A+(B \text{ o } I)$ ;	Calcular dirección: <b>ALUout</b> $\leftarrow A+I$ ;		Calcular destino y condición: <b>ALUout</b> $\leftarrow \text{NPC}+I$ ; <b>cond</b> $\leftarrow A=0?$ ;



### 3. Segmentación del ciclo de instrucción

#### Segmentación del ciclo de instrucción

Etapa	Instrucciones ejemplo			
	dadd Rdst,Rfte1,Rfte2 daddi Rdst,Rfte1,Imm	ld Rdst,Desp(Rfte1)	sd Rfte2,Desp(Rfte1)	beqz Rfte1,Desp
EX	Calcular resultado: <b>EX/MEM.ALUout</b> $\leftarrow \text{ID/EX.A} + \text{ID/EX.(B o I)};$	Calcular dirección: <b>EX/MEM.ALUout</b> $\leftarrow \text{ID/EX.A} + \text{ID/EX.I};$		Calcular destino y cond: <b>EX/MEM.ALUout</b> $\leftarrow \text{ID/EX.NPC} + \text{ID/EX.I};$ <b>EX/MEM.cond</b> $\leftarrow \text{IF/ID.A} = 0?;$



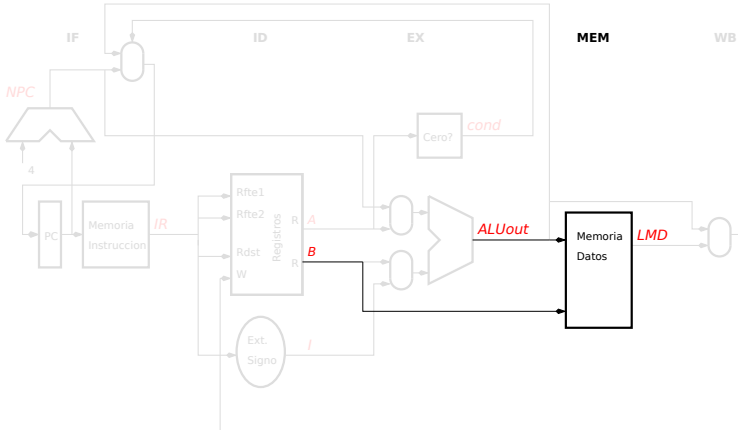
#### Ruta segmentada:

- aritmético-lógicas: El contenido de ALUout se escribe en Rdst en WB.
- loads y stores: El contenido de ALUout es la dirección accedida en MEM.
- stores: El contenido de B se escribe en MEM.
- saltos: Cond y ALUout alimentan IF en el próximo ciclo para obtener PC de la siguiente instrucción.

### 3. Segmentación del ciclo de instrucción

#### Segmentación del ciclo de instrucción

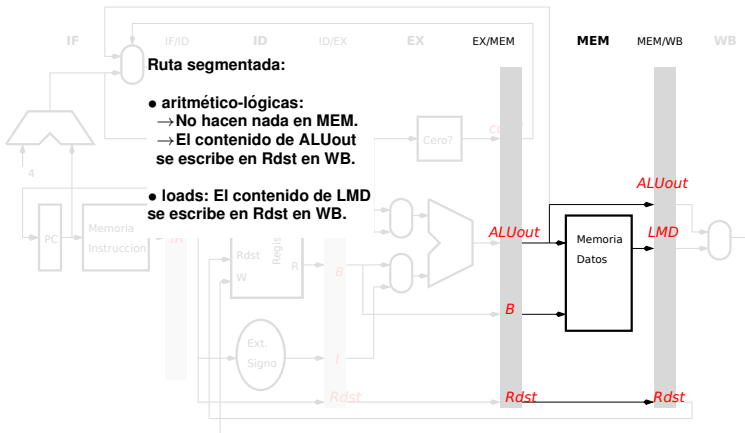
Fase MEM	Instrucciones ejemplo			
	dadd Rdst,Rfte1,Rfte2 daddi Rdst,Rfte1,Imm	ld Rdst,Desp(Rfte1)	sd Rfte2,Desp(Rfte1)	beqz Rfte1,Desp
		Leer dato en Memoria: <b>LMD</b> ← Mem[ALUout];	Escribir dato en Memoria: <b>Mem[ALUout]</b> ← B;	



### 3. Segmentación del ciclo de instrucción

#### Segmentación del ciclo de instrucción

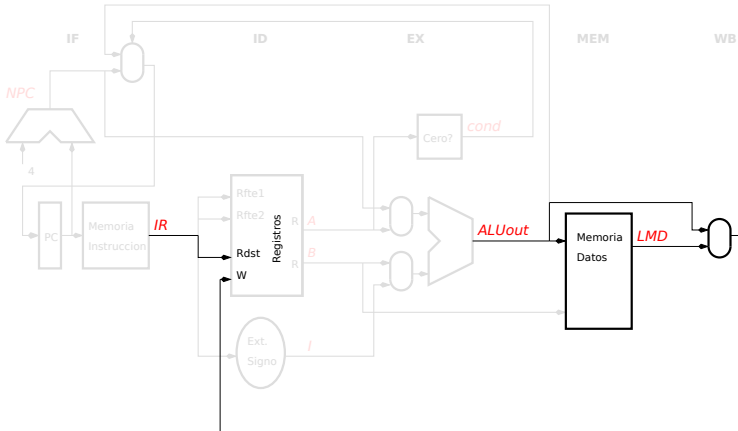
Etapa	Instrucciones ejemplo			
	dadd Rdst,Rfte1,Rfte2 daddi Rdst,Rfte1,Imm	ld Rdst,Desp(Rfte1)	sd Rfte2,Desp(Rfte1)	beqz Rfte1,Desp
MEM		Leer dato en Mem.: $MEM/WB.LMD \leftarrow Mem[EX/MEM.ALUout];$ $Mem[EX/MEM.ALUout];$	Escribir dato en Mem.: $Mem[EX/MEM.ALUout] \leftarrow EX/MEM.B;$	



### 3. Segmentación del ciclo de instrucción

#### Segmentación del ciclo de instrucción

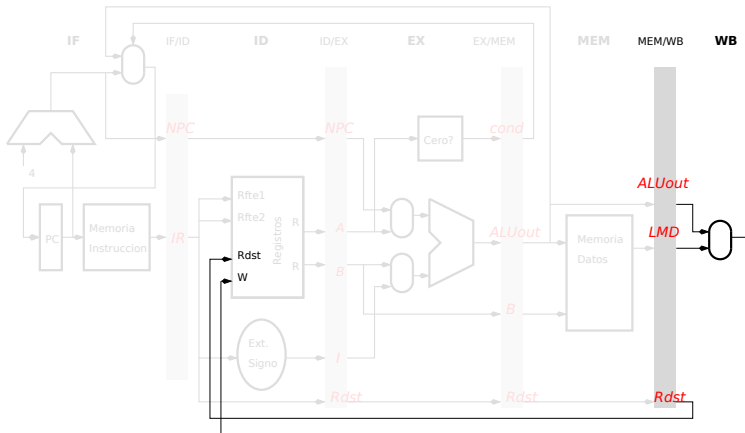
	Instrucciones ejemplo			
Fase	dadd Rdst,Rfte1,Rfte2	ld Rdst,Desp(Rfte1)	sd Rfte2,Desp(Rfte1)	beqz Rfte1,Desp
WB	Escribir resultado en Banco de Registros: <b>Regs[Rdst] ← ALUout o LMD</b>			



### 3. Segmentación del ciclo de instrucción

#### Segmentación del ciclo de instrucción

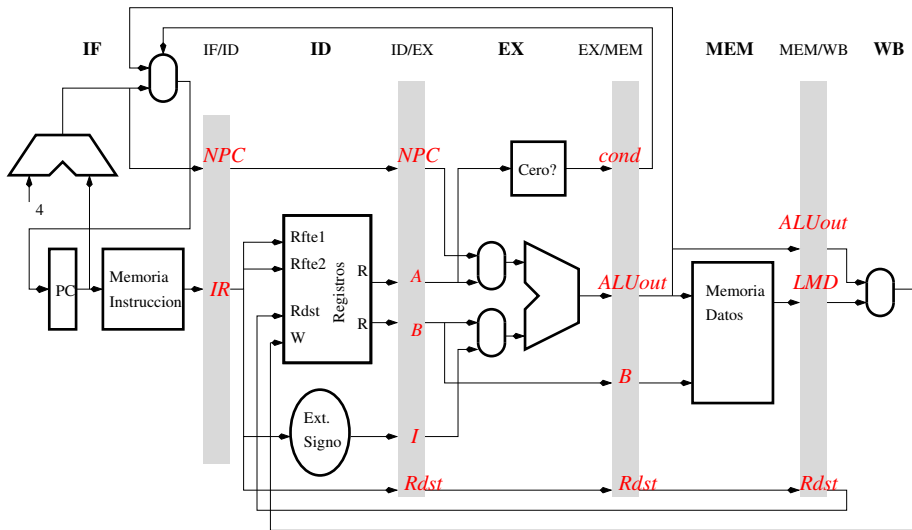
Fase	Instrucciones ejemplo			
	dadd Rdst,Rfte1,Rfte2	daddi Rdst,Rfte1,Imm	ld Rdst,Desp(Rfte1)	sd Rfte2,Desp(Rfte1)
WB	Escribir resultado en Banco de Registros: Regs[MEM/WB.Rdst] ← MEM/WB.(ALUout o LMD)			beqz Rfte1,Desp





### 3. Segmentación del ciclo de instrucción

#### Ruta de datos segmentada



### 3. Segmentación del ciclo de instrucción

#### Ciclo de instrucción segmentado

Fase	Instrucciones ejemplo			
	dadd Rdst,Rfte1,Rfte2 daddi Rdst,Rfte1,Imm	ld Rdst,Desp(Rfte1)	sd Rfte2,Desp(Rfte1)	beqz Rfte1,Desp
IF (10ns)	Leer instrucción en Memoria: <b>IF/ID.IR</b> ← Mem[PC]; Calcular NPC: <b>IF/ID.NPC</b> ← PC+4; Siguiente instrucción: if <b>EX/MEM.cond</b> then PC ← <b>EX/MEM.ALUout</b> else PC ← NPC;			
ID (5ns)	Decodificar instrucción Leer operandos en Banco de Registros: <b>ID/EX.A</b> ← Regs[ <b>IF/ID.IR.Rfte1</b> ]; <b>ID/EX.B</b> ← Regs[ <b>IF/ID.IR.Rfte2</b> ]; Extender signo del campo inmediato/desplazamiento: <b>ID/EX.I</b> ← Ext.Signo( <b>IF/ID.IR.Imm/Desp</b> );			
EX (10ns)	Calcular resultado: <b>EX/MEM.ALUout</b> ← <b>ID/EX.A + ID/EX.(B o I)</b> ;	Calcular dirección: <b>EX/MEM.ALUout</b> ← <b>ID/EX.A + ID/EX.I</b> ;		Calcular destino y cond: <b>EX/MEM.ALUout</b> ← <b>ID/EX.NPC + ID/EX.I</b> ; <b>EX/MEM.cond</b> ← <b>IF/ID.A = 0?</b> ;
MEM (10ns)		Leer dato en Mem.: <b>MEM/WB.LMD</b> ← Mem[ <b>EX/MEM.ALUout</b> ];	Escribir dato en Mem.: Mem[ <b>EX/MEM.ALUout</b> ] ← <b>EX/MEM.B</b> ;	
WB (5ns)	Escribir resultado en Banco de Registros: Regs[ <b>MEM/WB.Rdst</b> ] ← <b>MEM/WB.(ALUout o LMD)</b>			

Tiempo de ciclo T:

- El tiempo de ciclo debe ser lo suficientemente largo para poder soportar la ejecución de la **etapa** más larga.
- $T = \max(5, 10) = 10 \text{ ns}$

### 3. Segmentación del ciclo de instrucción

#### Ejecución de las instrucciones

$i$	IF	ID	EX	MEM	WB				
$i + 1$		IF	ID	EX	MEM	WB			
$i + 2$			IF	ID	EX	MEM	WB		
$i + 3$				IF	ID	EX	MEM	WB	
$i + 4$					IF	ID	EX	MEM	WB

Aceleración:

	No segmentado	Segmentado
$I$	$n$	$n$
$CPI$	1	1
$T$	$\text{máx}(\begin{matrix} \text{saltos} & \text{ual} & \text{almac.} & \text{carga} \\ 25, & 30, & 35, & 40 \end{matrix}) = 40 \text{ ns}$	$\text{máx}(5, 10) = 10 \text{ ns}$
$T_{ej} = I \cdot CPI \cdot T$	$n \cdot 1 \cdot 40 = 40n \text{ ns}$	$n \cdot 1 \cdot 10 = 10n \text{ ns}$

$$S = 4$$

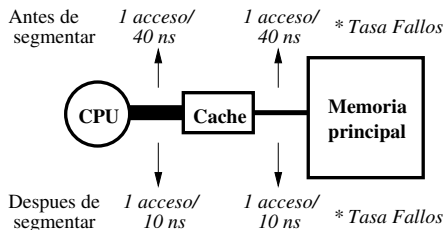
El número de etapas  $k = 5$  es la cota superior de la aceleración.



### 3. Segmentación del ciclo de instrucción

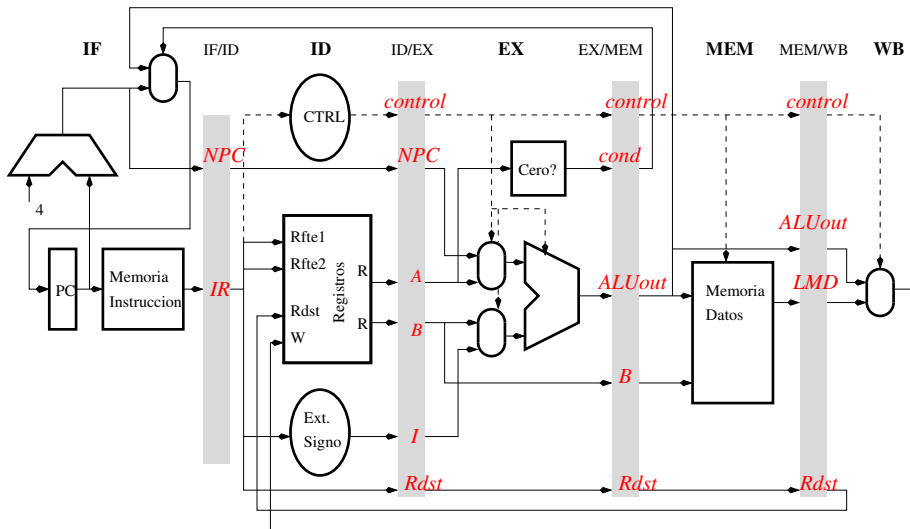
## Requisitos *hardware*: (cont.)

- Caché de instrucciones (IF) y de datos (MEM) separadas.
- Banco de registros con dos puertos de lectura (ID) y uno de escritura (WB) *simultáneos*.
- El tiempo de acceso a la memoria caché no varía, pero el ancho de banda requerido es 4 veces superior:



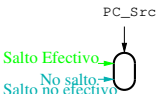
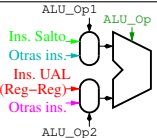
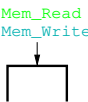
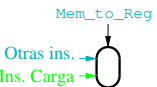
### 3. Segmentación del ciclo de instrucción

#### Señales de control



### 3. Segmentación del ciclo de instrucción

#### Señales de control

IF	ID	EX	MEM	WB
Mem_Read PC_Src	Reg_Read	ALU_Op ALU_Op1 ALU_Op2	Mem_Read Mem_Write	Reg_Write Mem_to_Reg
				

- Las señales de control necesarias en las etapas EX, MEM y WB se generan en la etapa ID.

## Índice

- 1 Concepto de segmentación
- 2 El ciclo de instrucción
- 3 Segmentación del ciclo de instrucción
- 4 Riesgos**
- 5 Riesgos de datos
- 6 Riesgos de control
- 7 Riesgos estructurales
- 8 Excepciones



## 4. Riesgos

### Concepto y clasificación

**Riesgo** (*hazard*): Situación que produce que la ejecución de algunas instrucciones genere resultados diferentes a los de la ruta de datos no segmentada.  $\Rightarrow$  Pérdida de compatibilidad binaria. Los riesgos se producen entre dos o más instrucciones que están presentes de forma simultánea en la ruta de datos segmentada.

**Tipos** de riesgos:

#### Datos

El resultado de una instrucción se utiliza como dato en la(s) instrucción(es) siguiente(s).

#### Control

En el flujo de instrucciones aparece una instrucción de salto.

#### Estructurales

Dos o más instrucciones pretenden utilizar el mismo recurso.

## 4. Riesgos

### Soluciones a los riesgos

**Inserción de ciclos de parada** Impedir el avance de la instrucción que origina el conflicto y de todas las que le siguen, pero no de las que le preceden (si no, el conflicto jamás desaparecería).

- En estos ciclos no se buscan nuevas instrucciones (ciclos de parada o *stalls*).
- Se origina una pérdida de prestaciones:

$$CPI_s = CPI_{s_{ideal}} + \frac{\text{Ciclos de parada}}{\text{instruccion}} = 1 + \frac{\text{stalls}}{\text{instruccion}} > 1$$

### Soluciones a los riesgos (cont.)

**Modificación de la ruta de datos** para detectarlos y resolverlos dinámicamente. La modificación puede:

- Reducir el número de ciclos de parada necesarios para resolverlo.
- Resolver el riesgo completamente (a veces no es posible).

**Modificación de la arquitectura del juego de instrucciones** Impedir que aparezca el problema prohibiendo la generación de ciertas secuencias de instrucciones. Para evitarlas, el compilador puede insertar instrucciones `NOP` o reordenar instrucciones independientes.

- Inconveniente: Se pierde la compatibilidad a nivel binario.

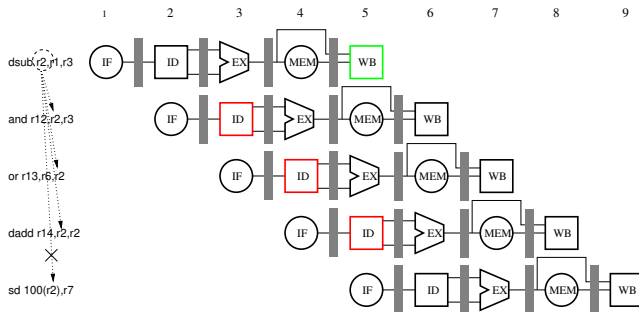
## Índice

- 1 Concepto de segmentación
- 2 El ciclo de instrucción
- 3 Segmentación del ciclo de instrucción
- 4 Riesgos
- 5 Riesgos de datos**
- 6 Riesgos de control
- 7 Riesgos estructurales
- 8 Excepciones

## 5. Riesgos de datos

### Causa

Una instrucción depende de los resultados producidos por otra anterior → al segmentar se puede invertir el orden de acceso a los operandos.

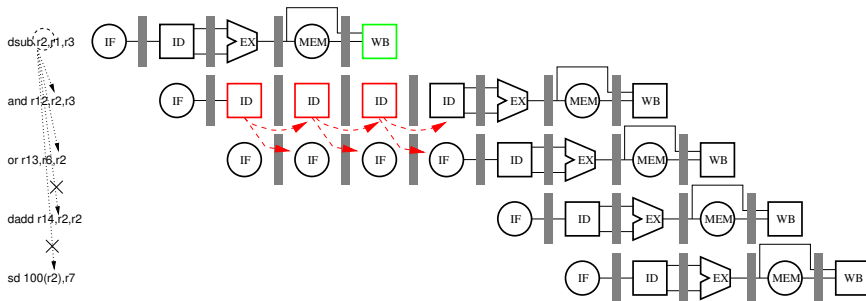


Se puede dar hasta entre 4 instrucciones consecutivas.

## 5. Riesgos de datos

### Inserción de ciclos de parada (*stalls*)

Retrasar las operaciones que causan el conflicto.



- 3 *stalls* por cada dos instrucciones dependientes consecutivas.
- 2 *stalls* por cada dos instr. dependientes separadas por 1 instr.
- 1 *stall* por cada dos instr. dependientes separadas por 2 instr.
- ⇒ Notable pérdida de prestaciones.

## 5. Riesgos de datos

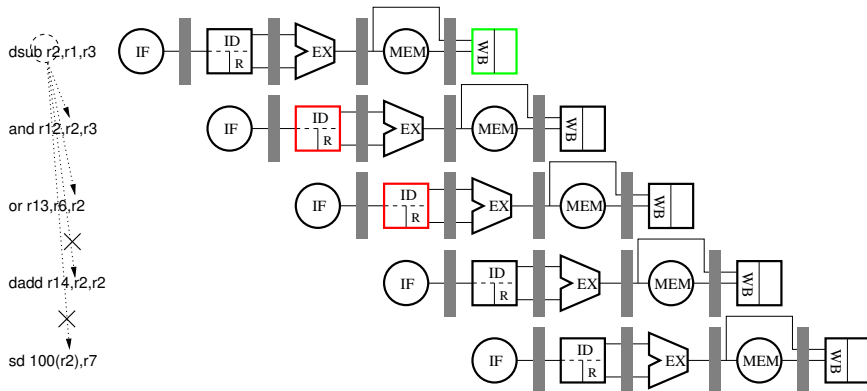
### Reducción del número de instrucciones involucradas

#### ⇒ Modificación de la ruta de datos

- Banco de registros multipuerto soportando lectura y escritura simultáneas.
- Modificación del acceso al banco de registros:
  - Lectura de registros → Segundo semiciclo.
  - Escritura en registros → Primer semiciclo.
- No hay problema si el tiempo de acceso al banco de registros es  $\leq$  a la mitad del periodo de reloj.
- Simplifica el diseño del banco de registros: No es necesario soportar lectura y escritura simultáneas.

## 5. Riesgos de datos

### Reducción del número de instrucciones involucradas (cont.)



⇒ el número de instrucciones involucradas es 3: **dsub**, **and** y **or**.

→ Penalización por instrucciones dependientes consecutivas:  
*2 stalls*



## 5. Riesgos de datos

### Inserción de 2 ciclos de parada

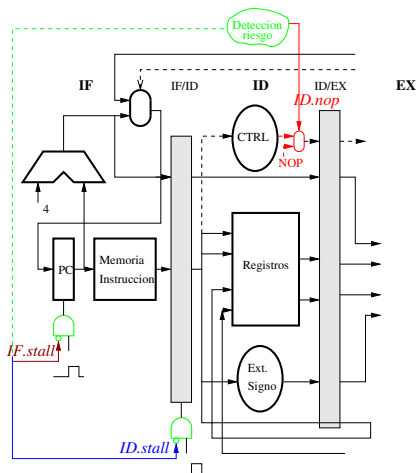
dsub <b>r2</b> ,r1,r3	IF	ID	<b>EX</b>	<b>ME</b>	WB				
and r12, <b>r2</b> ,r3	IF	<b>id</b>	<b>id</b>	ID	EX	ME	WB		
or r13,r6,r2		if	if	IF	ID	EX	ME	WB	
dadd r14,r2,r2					IF	ID	EX	ME	WB
sd 100(r2),r7						IF	ID	EX	ME WB

- Detectar las situaciones (en negrita) donde debe insertarse un ciclo de parada.
- Para insertar el ciclo de parada:
  - Poner las señales de control que circulan hacia la etapa EX como si de una instrucción `NOP` se tratara.
  - Conservar las instrucciones en IF y ID.

## 5. Riesgos de datos

### Inserción de 2 ciclos de parada (cont.)

#### Lógica de control



### Inserción de 2 ciclos de parada (cont.)

#### ■ Primer ciclo:

```
if (ID/EX.IR.CODOP = "ALU") and
    (IF/ID.IR.CODOP = "ALU") and
    ((ID/EX.IR.Rdst = IF/ID.IR.Rfte1) or
     (ID/EX.IR.Rdst = IF/ID.IR.Rfte2))
then
    IF.stall, ID.stall, ID.nop
```

#### ■ Segundo ciclo:

```
if (EX/ME.IR.CODOP = "ALU") and
    (IF/ID.IR.CODOP = "ALU") and
    ((EX/MEM.IR.Rdst = IF/ID.IR.Rfte1) or
     (EX/MEM.IR.Rdst = IF/ID.IR.Rfte2))
then
    IF.stall, ID.stall, ID.nop
```

## 5. Riesgos de datos

### Cortocircuitos

Problema: 2 *stalls* por riesgo de datos entre instrucciones consecutivas

```
dsub r2, r1, r3    IF ID EX ME WB
and  r12, r2, r3    IF id id ID EX ME WB
or   r13, r6, r2      if if IF ID EX ME WB
```

Consideremos el riesgo entre las dos primeras instrucciones si no insertáramos ciclos de parada:

```
          1  2  3  4  5
dsub r2, r1, r3    IF ID EX ME WB
and  r12, r2, r3    IF ID EX ME WB
```

- `and` necesita el dato al principio del ciclo 4 (su etapa EX)
- `dsub` tiene el resultado al principio del ciclo 4 (su etapa MEM)

## 5. Riesgos de datos

### Cortocircuitos (cont.)

⇒ **Modificación de la ruta de datos**

Añadir un bus desde la salida de la UAL (etapa MEM) hacia su entrada (etapa EX) + lógica de control ⇒ “Cortocircuito” de MEM hacia EX.

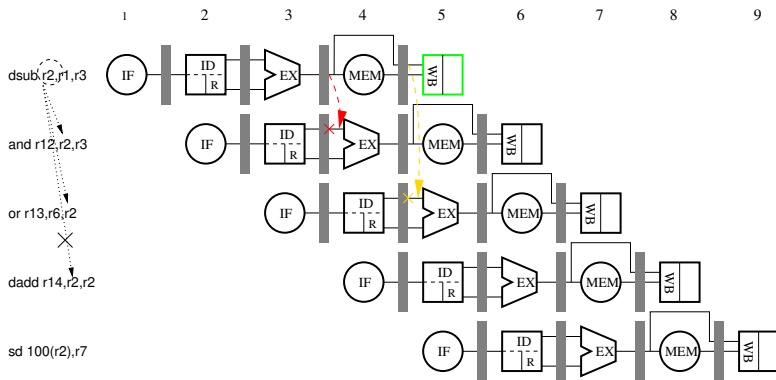
Si consideramos la tercera instrucción, el riesgo se puede resolver de forma similar con un cortocircuito de WB hacia EX.

dsub <b>r2</b> , r1, r3	IF	ID	EX	<b>ME</b>	<b>WB</b>
and r12, <b>r2</b> , r3	IF	ID	<b>EX</b>	ME	WB
or r13, r6, <b>r2</b>		IF	ID	<b>EX</b>	ME WB

## 5. Riesgos de datos

### Cortocircuitos (cont.)

Instrucciones consumidoras posteriores ya no requieren cortocircuito.

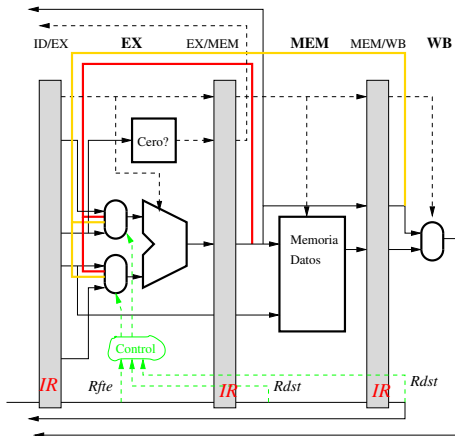


## 5. Riesgos de datos

### Cortocircuitos (cont.)

#### Implementación

- Multiplexores a la entrada de la UAL.



### Cortocircuitos (cont.)

#### Lógica de control

##### ■ Cortocircuito de MEM a EX:

###### ■ Rfte1:

```
if (EX/MEM.IR.CODOP = "ALU") and  
   (ID/EX.IR.CODOP = "ALU") and  
   (EX/MEM.IR.Rdst = ID/EX.IR.Rfte1)  
then  
    Cortocircuito MEM-a-EX (entrada sup. UAL)
```

###### ■ Rfte2:

```
if (EX/MEM.IR.CODOP = "ALU") and  
   (ID/EX.IR.CODOP = "ALU") and  
   (EX/MEM.IR.Rdst = ID/EX.IR.Rfte2)  
then  
    Cortocircuito MEM-a-EX (entrada inf. UAL)
```



### Cortocircuitos (cont.)

#### ■ Cortocircuito de WB a EX:

##### ■ Rfte1:

```
if (MEM/WB.IR.CODOP = "ALU") and  
   (ID/EX.IR.CODOP = "ALU") and  
   (MEM/WB.IR.Rdst = ID/EX.IR.Rfte1)  
then  
    Cortocircuito WB-a-EX (entrada sup. UAL)
```

##### ■ Rfte2:

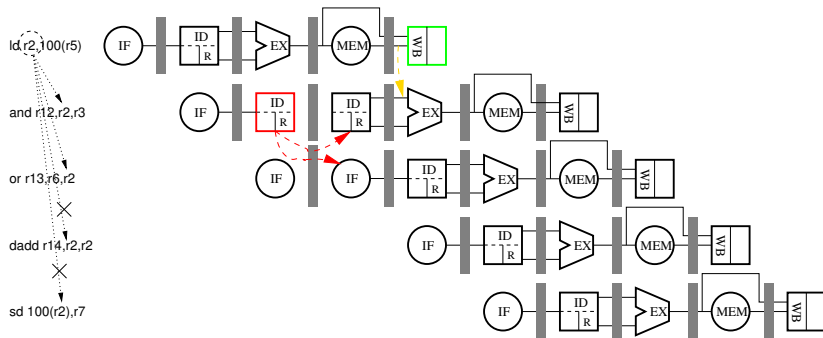
```
if (MEM/WB.IR.CODOP = "ALU") and  
   (ID/EX.IR.CODOP = "ALU") and  
   (MEM/WB.IR.Rdst = ID/EX.IR.Rfte2)  
then  
    Cortocircuito WB-a-EX (entrada inf. UAL)
```



## 5. Riesgos de datos

### Instrucción consecutiva y dependiente de una *load* (cont.)

Utilizando el cortocircuito de WB a EX hace falta insertar 1 ciclo de parada.



## 5. Riesgos de datos

### Instrucción consecutiva y dependiente de una *load* (cont.)

#### Lógica de control para insertar el ciclo de parada

```
if (ID/EX.IR.CODOP = "LOAD") and  
    (IF/ID.IR.CODOP = "ALU") and  
    ((ID/EX.IR.Rdst = IF/ID.IR.Rfte1) or  
     (ID/EX.IR.Rdst = IF/ID.IR.Rfte2))  
then  
    IF.stall, ID.stall, ID.nop
```

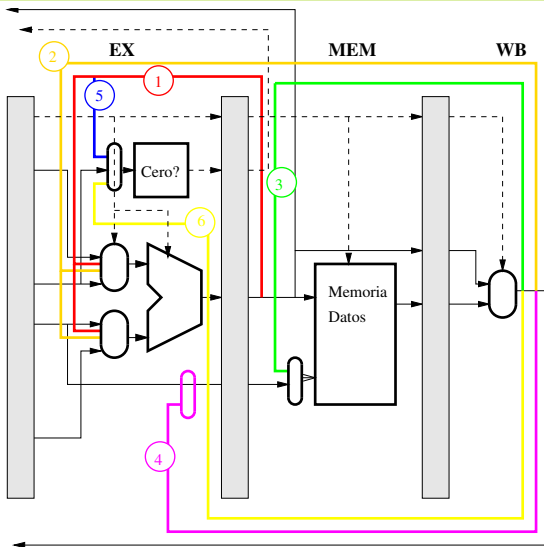
## 5. Riesgos de datos

### Resumen de cortocircuitos y ciclos de parada

Tipos de instrucciones	Ejemplo	Cortocircuito	Ciclos de parada	Fig.
UAL - UAL	DADD R1,R2,R3	MEM a EX WB a EX	0	1
	DSUB R4,R1,R5 AND R7,R1,R6		0	2
Carga - UAL	LD R1,20(R2) DADD R3,R1,R4	WB a EX	1	2
UAL - Carga/Almac.	DADD R1,R2,R3	MEM a EX WB a EX	0	1
	LD R2,20(R1) LD R3,40(R1)		0	2
UAL - Almac.	DADD R1,R2,R3	WB a MEM WB a EX	0	3
	SD R1,20(R2) SD R1,40(R2)		0	4
Carga - Almac.	LD R1,20(R3)	WB a MEM WB a EX	0	3
	SD R1,20(R2) SD R1,40(R2)		0	4
Carga - Carga/Almac.	LD R1,30(R3) LD R2,20(R1)	WB a EX	1	2
UAL - Salto	DSLT R1,R2,R3 BEQZ R1,loop	MEM a EX	0	5
UAL - Salto	DSLT R1,R2,R3 ...	WB a EX	0	6
	BEQZ R1,loop			
Carga - Salto:	LD R1,20(R2) BEQZ R1,loop	WB a EX	1	6

## 5. Riesgos de datos

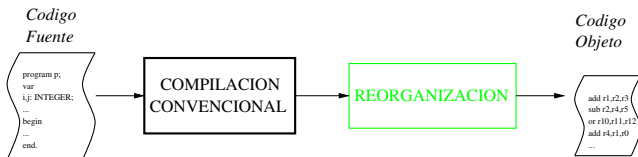
### Resumen de cortocircuitos y ciclos de parada (cont.)



## 5. Riesgos de datos

### Reorganización del código

⇒ El compilador tiene una fase más: reorganización del código:



⇒ El objetivo es evitar ciclos de parada. Ejemplo:

Código convencional	Código reorganizado
LD Rb,b	LD Rb,b
LD Rc,c	LD Rc,c
DADD Ra,Rb,Rc	LD Re,e
LD Re,e	DADD Ra,Rb,Rc
LD Rf,f	LD Rf,f
DSUB Rd,Re,Rf	SD Ra,a
SD Ra,a	DSUB Rd,Re,Rf
SD Rd,d	SD Rd,d
8 ins + 2 stalls = 10 ciclos	8 ins = 8 ciclos

## Índice

- 1 Concepto de segmentación
- 2 El ciclo de instrucción
- 3 Segmentación del ciclo de instrucción
- 4 Riesgos
- 5 Riesgos de datos
- 6 Riesgos de control**
- 7 Riesgos estructurales
- 8 Excepciones

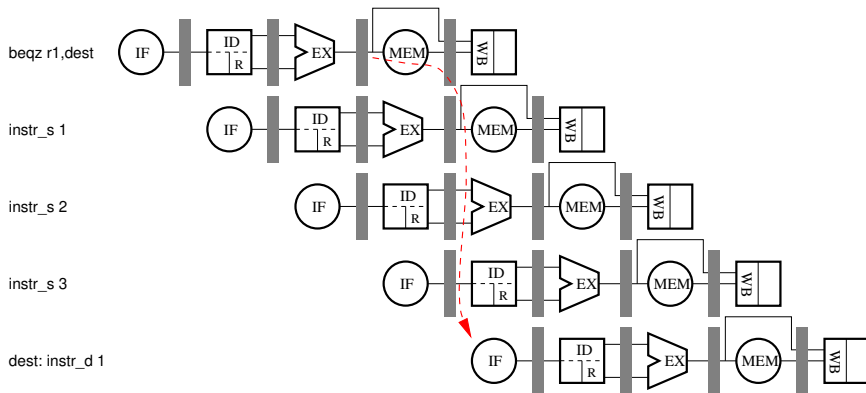




## 6. Riesgos de control

### Causa (cont.)

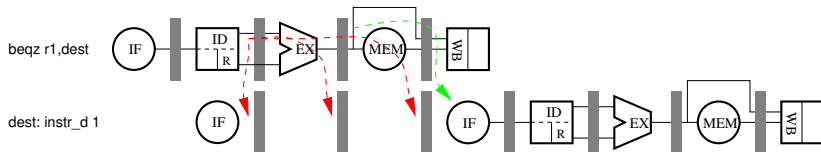
En ese ciclo se han lanzado ya 3 instrucciones:



## 6. Riesgos de control

### Inserción de ciclos de parada

Insertar ciclos de parada *siempre* que se decodifica una instrucción de salto:



3 ciclos de parada → Pérdida de prestaciones.

## 6. Riesgos de control

### Inserción de ciclos de parada (cont.)

#### Lógica de control para la inserción de 3 ciclos de parada

- Inhibir la etapa IF durante tres ciclos de reloj, pasando instrucciones NOP a la etapa ID:

```
beqz r1,dest    IF ID EX ME WB  
<dest>         if if if IF ID EX ME WB
```

```
if (IF/ID.IR.CODOP = "Salto") or  
   (ID/EX.IR.CODOP = "Salto") or  
   (EX/MEM.IR.CODOP = "Salto")  
then  
    IF.stall, IF.nop
```

## 6. Riesgos de control

### Predicción (fija)

⇒ Modificación de la ruta de datos

**Predict-not taken:** Suponer el salto no efectivo → las instrucciones buscadas a continuación de la de salto son válidas.

Si, finalmente, el salto es efectivo, se abortan las tres instrucciones en curso.

IMPORTANTE: Estas instrucciones no deben haber modificado el estado.

**Predict-taken:** Suponer el salto efectivo → en cuanto se conoce la dirección destino se buscan las nuevas instrucciones.

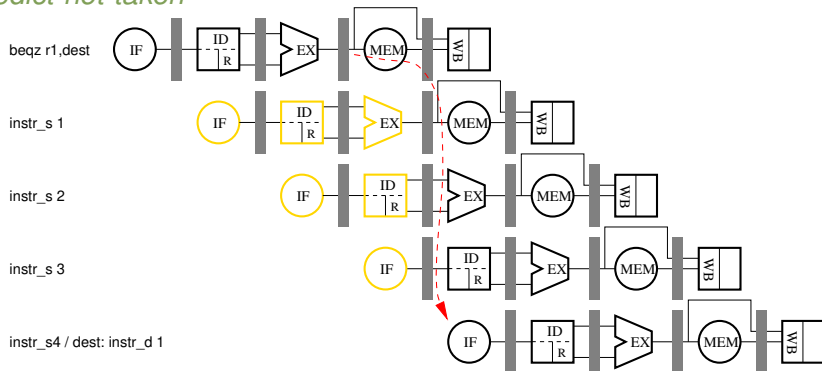
Si, finalmente, el salto es no efectivo, se abortan estas instrucciones.

Solo es útil si se conoce la dirección destino *antes que* la condición de salto → inútil en MIPS.

## 6. Riesgos de control

### Predicción (fija) (cont.)

#### *Predict-not-taken*



#### Lógica de control

```
if EX/MEM.cond then  
    IF.nop, ID.nop, EX.nop
```

### Reducción de la latencia de salto

#### ⇒ Modificación de la ruta de datos

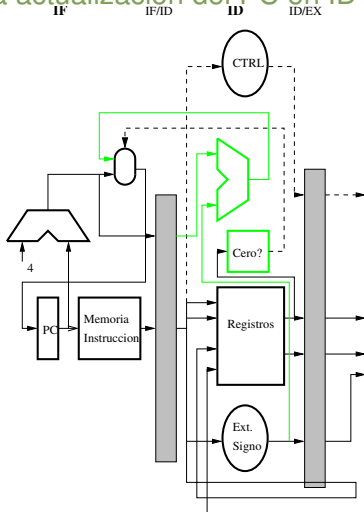
Reducir el número de ciclos comprendidos entre la búsqueda de la instrucción de salto y la correspondiente al destino del salto.

- Actualizar el PC en la etapa EX  $\rightarrow$  N° de ciclos = 2.  
⇒ En la etapa EX del salto se conoce el nuevo PC para el ciclo siguiente.
- Actualizar el PC en la etapa ID  $\rightarrow$  N° de ciclos = 1.  
⇒ En la etapa ID del salto se conoce el nuevo PC para el ciclo siguiente. Requiere:
  - Trasladar el cálculo de la dirección efectiva de EX a ID  $\rightarrow$  hace falta un sumador adicional
  - Trasladar la evaluación de la condición de EX a ID.

## 6. Riesgos de control

### Reducción de la latencia de salto (cont.)

#### Implementación de la actualización del PC en ID





## 6. Riesgos de control

### Reducción de la latencia de salto (cont.)

#### Lógica de control para actualizar el PC en ID

- Asumiendo *predict-not-taken*, si el salto no es efectivo, se busca la siguiente instrucción.
- Pero si el salto es efectivo, la instrucción buscada en el mismo ciclo debe cancelarse. → Se entrega una NOP a la etapa ID.

```
beqz r1,dest    IF ID EX ME WB
<sgte>          IF X
<dest>          IF ID EX ME WB
```

```
if (IF/ID.IR.CODOP = "Salto") and
  (Regs[IF/ID.IR.Rftel] op 0)
then
  IF.nop
  PC <- IF/ID.NPC + Ext.Signo(IF/ID.IR.Desp)
else
  PC <- PC + 4
```

## 6. Riesgos de control

### Reducción de la latencia de salto (cont.)

Impacto en el periodo de reloj

#### Etapa ID

tiempo de acceso a los registros +  
retardo de la evaluación de la condición +  
selección +  
escritura sobre el PC

→ ID puede convertirse en la etapa más lenta  
⇒ las condiciones de salto deben ser sencillas ( $=$  y  $\neq$ ).

Esta modificación puede ser incompatible con la lectura de registros en el segundo semiciclo.

## 6. Riesgos de control

### Reducción de la latencia de salto (cont.)

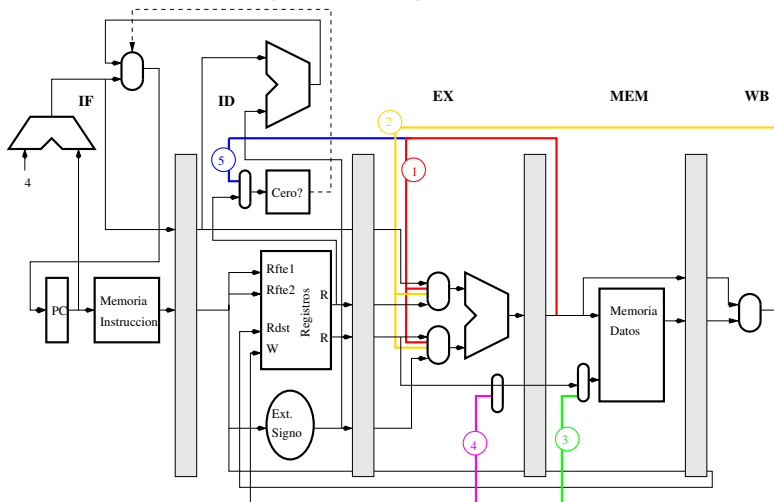
#### Impacto en cortocircuitos y ciclos de parada

Instrucciones	Ejemplo	Cortocircuito	stalls	Fig.
UAL - UAL	DADD R1,R2,R3	MEM a EX	0	1
	DSUB R4,R1,R5 AND R7,R1,R6	WB a EX	0	2
Carga - UAL	LD R1,20(R2) DADD R3,R1,R4	WB a EX	1	2
UAL - Carga/Almac.	DADD R1,R2,R3	MEM a EX	0	1
	LD R2,20(R1) LD R3,40(R1)	WB a EX	0	2
UAL - Almac.	DADD R1,R2,R3	WB a MEM	0	3
	SD R1,20(R2) SD R1,40(R2)	WB a EX	0	4
Carga - Almac.	LD R1,20(R3)	WB a MEM	0	3
	SD R1,20(R2) SD R1,40(R2)	WB a EX	0	4
Carga - Carga/Almac.	LD R1,30(R3) LD R2,20(R1)	WB a EX	1	2
UAL - Salto	DSLT R1,R2,R3	MEM a EX	0	5
	BEQZ R1,loop	MEM a ID	1	
UAL - Salto	DSLTLT R1,R2,R3	WB a EX	0	5
	BEQZ R1,loop	MEM a ID		
Carga - Salto	LD R1,20(R2)	WB a EX	1	-
	BEQZ R1,loop	Por BR	2	
Carga - Salto	LD R1,20(R2)	Por BR	0	-
	BEQZ R1,loop		1	

## 6. Riesgos de control

### Reducción de la latencia de salto (cont.)

#### Impacto en cortocircuitos y ciclos de parada



### Salto retardado (*delayed branch*)

⇒ **Modificación de la arquitectura del juego de instrucciones**

El compilador *debe* colocar tras los saltos instrucciones que se ejecutarán tanto si se salta como si no se salta.

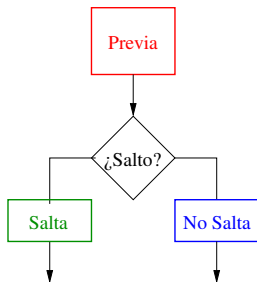
- Como estas instrucciones siempre se ejecutan, no es necesario introducir ciclos de parada o cancelarlas.
- En general, se eligen instrucciones *anteriores* al salto que no dependan de la condición del salto.

### *Branch delay slot*

- Número de instrucciones tras el salto que se ejecutarán siempre independientemente de si es efectivo o no.
  - Coincide con la latencia del salto.
  - 1 ciclo si se actualiza el PC en ID.

## 6. Riesgos de control

### Salto retardado (*delayed branch*) (cont.)



Código convencional:



¿Salto?

No Salta

.....

Salta

Código con salto retardado:



¿Salto?



No Salta

.....

Salta

Ejemplo:

Convencional

DADD Rc, Ra, Rb

BEQZ Ra, dst

INSTR1

dst: INSTR2

Salto retardado

BEQZ Ra, dst

DADD Rc, Ra, Rb

INSTR1

dst: INSTR2

### Salto retardado (*delayed branch*) (cont.)

#### Salto retardado en los juegos de instrucciones actuales

El salto retardado deja de utilizarse aproximadamente a partir de los 90. Algunas razones:

- Si el *branch delay slot* es elevado (esto es, mayor que una instrucción), el compilador tiene más dificultad para encontrar instrucciones útiles para colocar tras los saltos. Esto ocurre si:
  - La latencia de salto es elevada (p.e. la ruta de datos tiene muchas etapas).
  - Se lanzan múltiples instrucciones por ciclo de reloj (Tema 2.5).
- Condiciona el juego de instrucciones a una implementación de la ruta de datos, lo cual es inflexible. Por ejemplo, un diseño de ruta de datos diferente podría variar la latencia de salto, perdiéndose la compatibilidad.
- Hay otras soluciones mucho mejores basadas en realizar una predicción dinámica de los saltos (Tema 2.3).

## Índice

- 1 Concepto de segmentación
- 2 El ciclo de instrucción
- 3 Segmentación del ciclo de instrucción
- 4 Riesgos
- 5 Riesgos de datos
- 6 Riesgos de control
- 7 Riesgos estructurales**
- 8 Excepciones



## 7. Riesgos estructurales

### Causas

- El hardware no permite todas las combinaciones posibles de las instrucciones presentes en la unidad.  
→ Un recurso no ha sido replicado suficientemente.
- **Ejemplo:** Procesador con caché única de instrucciones y datos.  
→ La etapa MEM de las instrucciones de Carga/Almac. colisiona con la etapa IF de la instrucción que ocupa el 3<sup>er</sup> puesto tras estas.

## 7. Riesgos estructurales

### Soluciones

#### Modificaciones de la ruta de datos

Replicar el recurso para que sea posible esa combinación.

→ Ejemplo: Arquitectura *Harvard*: utiliza caché de instrucciones y datos separadas.

→ Aumento del coste.

→ No siempre es posible o tiene sentido replicar el recurso.

#### Insertión de ciclos de parada

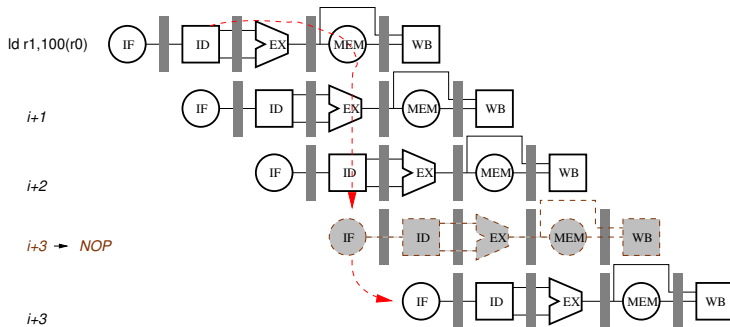
Retrasar una de las operaciones que causa el conflicto.

→ *stalls* → Pérdida de prestaciones.

→ Las prestaciones dependen de la frecuencia de aparición de las combinaciones que originan los riesgos estructurales.

## 7. Riesgos estructurales

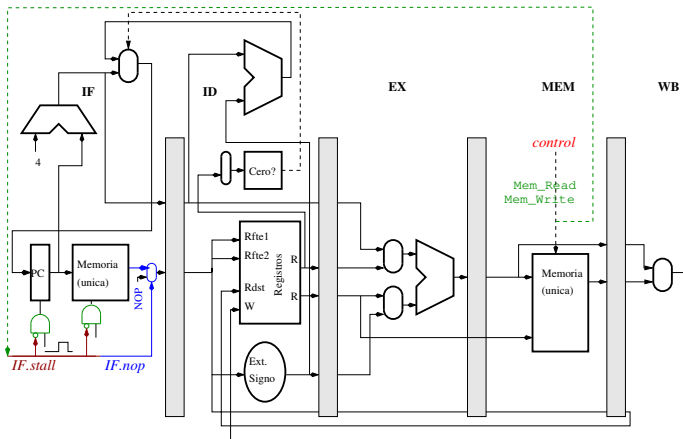
### Inserción de ciclos de parada



## 7. Riesgos estructurales

### Inserción de ciclos de parada (cont.)

#### Implementación



## 7. Riesgos estructurales

### Inserción de ciclos de parada (cont.)

#### Lógica de control

Cuando una instrucción de Carga/Almac. está en la etapa MEM,

- No se accede a la memoria de instrucciones.
- Conservando la instrucción de la etapa IF.
- Entregando a la etapa ID una instrucción NOP.

```
if EX/MEM.Mem_Read or EX/MEM.Mem_Write then  
    IF.stall, IF.nop
```

## Índice

- 1 Concepto de segmentación
- 2 El ciclo de instrucción
- 3 Segmentación del ciclo de instrucción
- 4 Riesgos
- 5 Riesgos de datos
- 6 Riesgos de control
- 7 Riesgos estructurales
- 8 Excepciones**

## 8. Excepciones

### Concepto y clasificación

**Denominaciones:** Interrupción, **excepción** o falta.

**Tipos:**

- Sincrona vs. asíncrona. Es síncrona si el evento ocurre en el mismo lugar cada vez que el programa se ejecuta.
- Solicitada por el usuario vs. lanzada hacia el usuario.
- Enmascarables por el usuario vs. no enmascarables.
- En medio de una instrucción vs. entre instrucciones.
- Continuar vs. terminar el programa.

Una excepción es un salto condicional implícito → Si se produce la excepción, hay que saltar a la rutina de servicio.

## 8. Excepciones

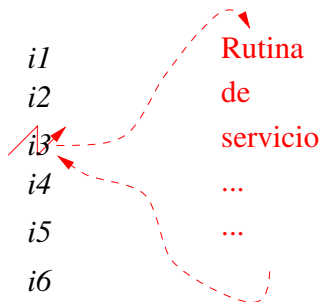
### Excepciones posibles en el MIPS

<i>Etapas</i>	<i>Excepciones</i>
IF	Fallo de página de instrucción, Acceso desalineado Violación de protección, Petición E/S
ID	Instrucción ilegal, Petición E/S
EX	Excepción aritmética, Petición E/S
MEM	Fallo de página de datos, Acceso desalineado Violación de protección, Petición E/S
WB	Petición E/S



## 8. Excepciones

### Excepciones en computadores convencionales



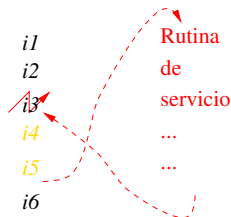
Secuencia correcta: ... *i1*, *i2*, *i3* - Rutina de servicio - *i3*, *i4*, *i5*, *i6* ...



## 8. Excepciones

### Excepciones en unidades de instrucción segmentadas (cont.)

El funcionamiento es incorrecto:



Secuencia: ...  $i1$ ,  $i2$ ,  $i3$ ,  $i4$ ,  $i5$  - Rutina de servicio -  $i3$ ,  $i4$ ,  $i5$ ,  $i6$  ...

- El PC de las instrucciones solo se usa en IF.

Las primeras máquinas segmentadas finalizaban el programa, imprimiendo el PC de la instrucción que se encontraba en IF

⇒ indicaban *aproximadamente* la instrucción que había originado la excepción → excepciones *imprecisas*.

## 8. Excepciones

### Excepciones *precisas* en unidades de instrucción segmentadas

Un computador soporta un comportamiento *preciso* frente a las excepciones si:

- Las instrucciones anteriores a la que origina la excepción terminan correctamente.
- La instrucción que origina la excepción y todas las posteriores son abortadas.
- Tras completar la rutina de servicio se puede relanzar el programa comenzando por la instrucción que originó la excepción.

⇒ Se puede identificar la instrucción causante de la excepción.

⇒ El comportamiento es idéntico al que tendría el mismo computador no segmentado.

## 8. Excepciones

### Implementación de excepciones precisas en el MIPS

#### Requisitos:

- Se pueden producir no una excepción, sino hasta 5 excepciones.
- En el mismo o en distintos ciclos de reloj.
- Teniendo en cuenta cómo funciona el salto retardado.

Idea: Asegurar que el orden de atención de las excepciones se realiza en el orden natural.

⇒ La llegada de las instrucciones a la última etapa se realiza en orden

- 1 Cada instrucción que entra en la unidad tiene asociado un registro con tantos bits como etapas en las que se pueden originar excepciones, el cual acompaña a la instrucción durante su recorrido por la misma.

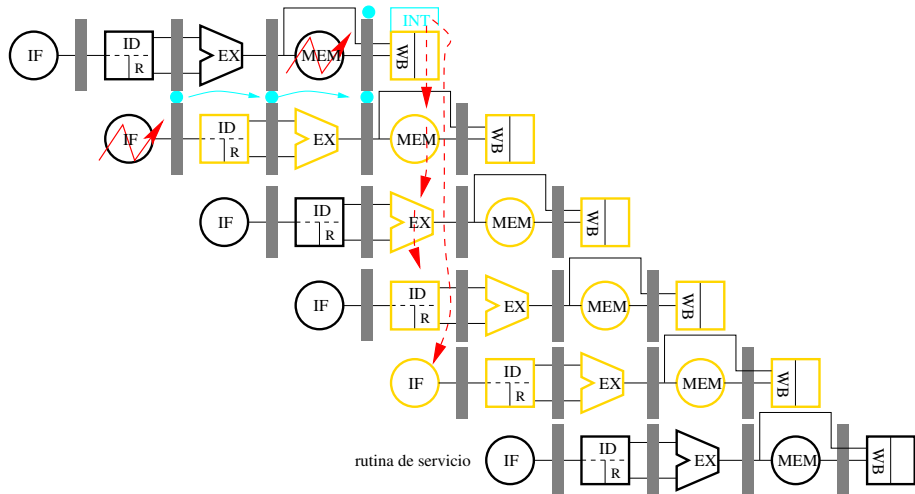
## 8. Excepciones

### Implementación de excepciones precisas en el MIPS (cont.)

- 2 Si se produce una excepción, se pone a “1” el bit de la etapa correspondiente, al tiempo que se convierte en `NOP` la instrucción implicada.
- 3 En la última etapa del ciclo de instrucción se verifica si alguno de los bits está activado.  
En caso afirmativo, convierte en `NOP` las instrucciones posteriores y escribe la dirección de la rutina de servicio en el PC.
- 4 Se guarda la dirección de la instrucción que origina la excepción.
- 5 La rutina de servicio toma el control.
- 6 Cuando finaliza la rutina de servicio, se restaura el PC con la dirección de la instrucción que originó la excepción, continuando la ejecución desde este punto.

## 8. Excepciones

### Implementación de excepciones precisas en el MIPS (cont.)



## 8. Excepciones

### Implementación de excepciones precisas en el MIPS (cont.)

