4.2 Segmentación del repertorio de instrucciones

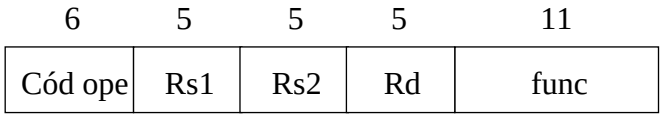
Tema 4. Segmentación

4.2. SEGMENTACIÓN DEL REPERTORIO DE INSTRUCCIONES

- 4.2.1. Segmentación básica en la arquitectura MIPS
- 4.2.2. Resolución de la problemática de segmentación para MIPS
- 4.2.3. Riesgos de la segmentación
- 4.2.4. Ruta de datos sementada
- 4.2.5. Ejemplos

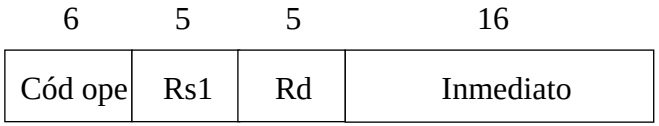
Formato de las instrucciones

Tipo R (Aritmético-lógicas)



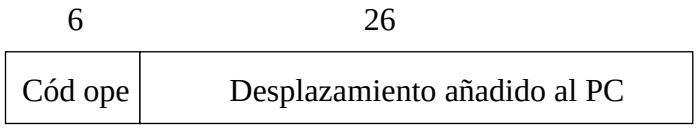
add R1,R2,R3

Tipo I (acceso a memoria, saltos condicionales, inmediatas)



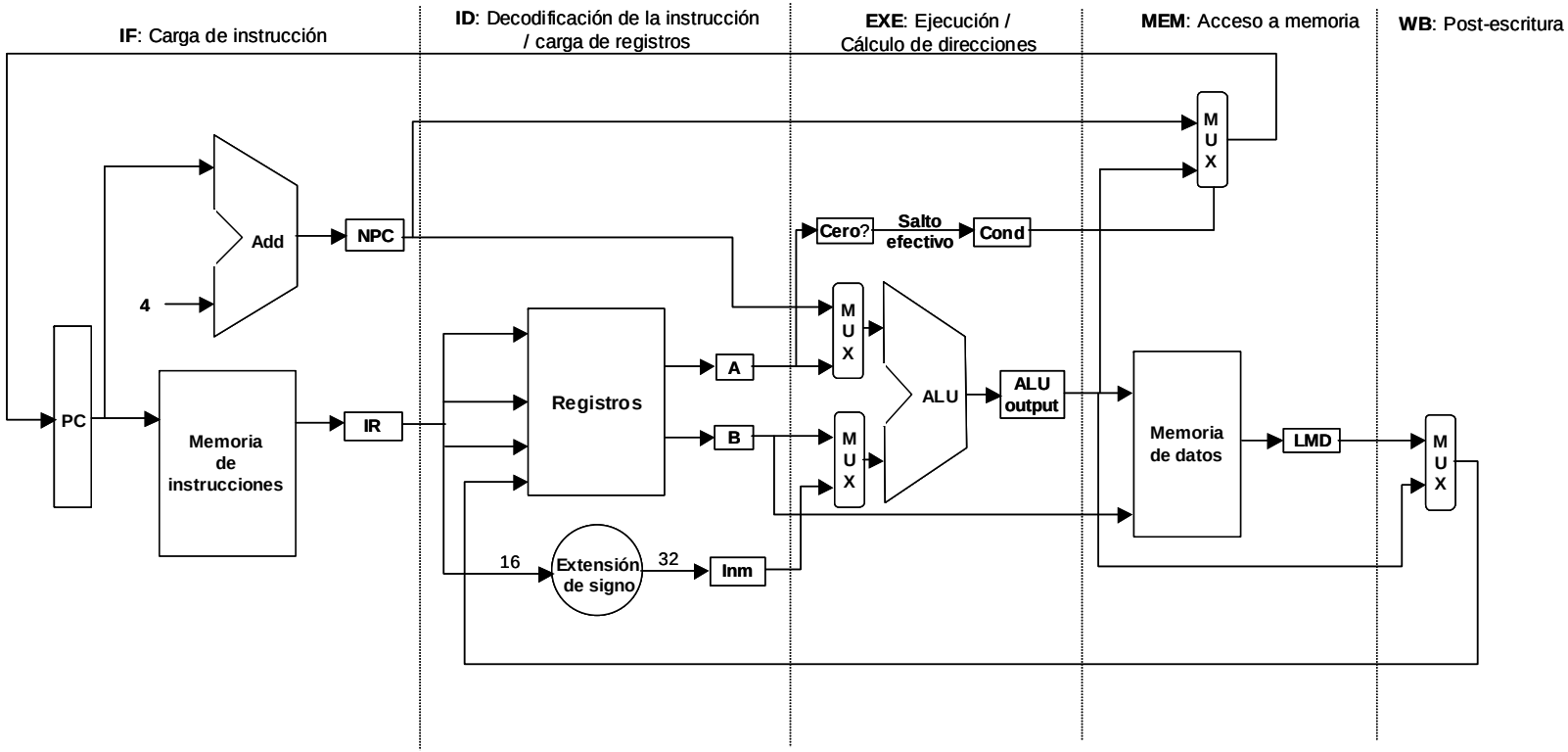
lw R1,100(R2)
sw R2, 50(R4)
add R1, R2,#100
beqz R1, ET

Tipo J (salto incondicional)



J Etiqueta

Ruta de datos no segmentada



Ejecución de las instrucciones

Etapa	Aritmético-lógicas	Carga/almacenamiento	Salto
IF	IR ← MEM [PC] NPC ← PC + 4		
ID	A ← Regs [$IR_{6..10}$] B ← Regs [$IR_{11..15}$] Inm ← ((IR_{16})^{16} ## $IR_{16..31}$)		
EX	ALUoutput ← A op B O bien ALUoutput ← A op Inm	ALUoutput ← A + Inm	ALUoutput ← NPC + Inm cond ← (A op 0)

MEM	PC ← NPC	PC ← NPC
		
		LMD ←MEM[ALUoutput
		O bien
		MEM [ALUoutput] ← B
			If (cond)
			PC ← ALUoutput
			else
			PC ← NPC

WB	Regs [$IR_{16..20}$] ← ALUoutput O bien Regs [$IR_{11..15}$] ← ALUoutput	Regs [$IR_{11..15}$] ← LMD	

Implementación simple del MIPS

- **1. Ciclo de búsqueda de instrucción**

$$IR \leftarrow MEM\ [PC]$$

$$NPC \leftarrow\ PC\ +\ 4$$

- **Operación:** Transfiere el PC y ubica la instrucción de memoria en el registro de instrucciones. Incrementa el PC en 4 para apuntar la siguiente instrucción de la secuencia.

Implementación simple del MIPS

1. Ciclo de búsqueda de instrucción

IR ← MEM [PC]
NPC ← PC + 4

Implementación simple del MIPS

- **2. Ciclo de decodificación de la instrucción / carga de registros**

$$A \leftarrow Regs\,[IR_{6..10}]$$

$$B \leftarrow Regs\,[IR_{11..15}]$$

$$Inm \leftarrow (\,(IR_{16})^{16}\ \#\#\ IR_{16..31})$$

- **Operación:** Decodifica la instrucción y accede al banco de registro para leer los registros. Las salidas de los dos registros de propósito general se cargan en dos registros temporales (A y B) para su uso posterior.
- Se extiende el signo a los 16 bits bajos del IR y se almacena el resultado en el registro temporal Inm

Implementación simple del MIPS

- **2. Ciclo de decodificación de la instrucción / carga de registros**

$$A \leftarrow Regs\,[IR_{6..10}]$$

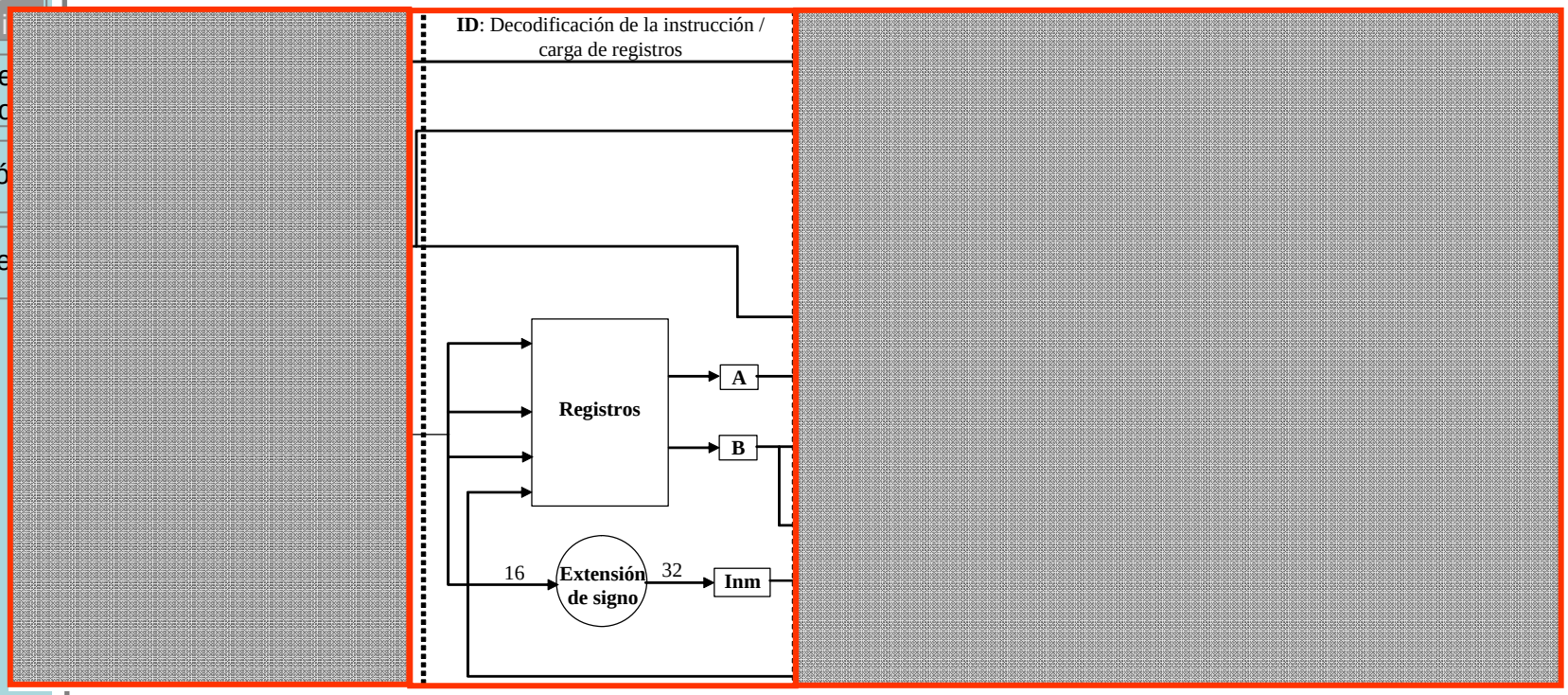
$$B \leftarrow Regs\,[IR_{11..15}]$$

$$Inm \leftarrow (\,(IR_{16})^{16}\ \#\#\ IR_{16..31})$$

- **Decodificación de campo fijo**: Decodificación en paralelo con la lectura de los registros posible porque los campos ocupan posiciones fijas en el formato de instrucción DLX.
- Inmediato situado en posiciones idénticas en todos los formatos de instrucción DLX, el inmediato de signo extendido se calcula también en este ciclo.

Implementación simple del MIPS

2. Ciclo de decodificación de la instrucción / carga de registros



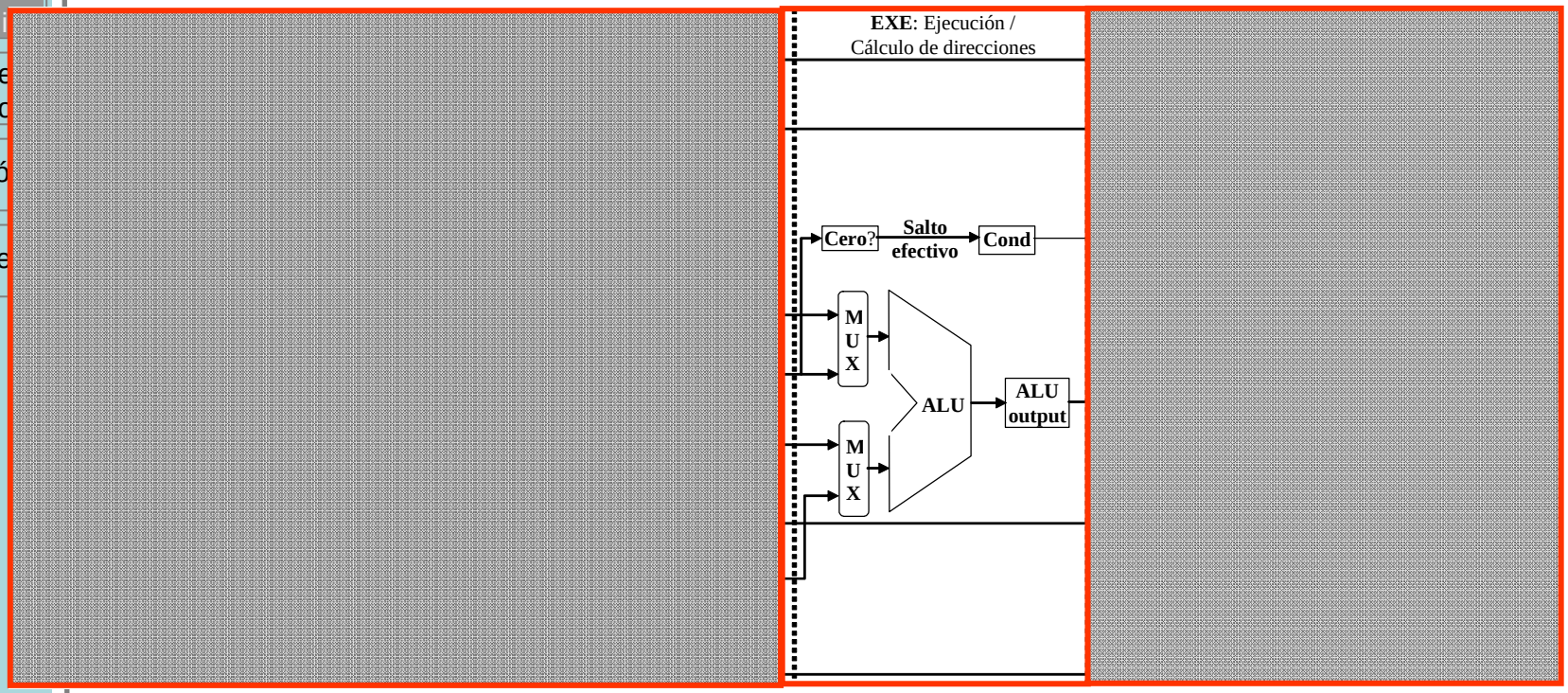
A ← Regs [IR6..10]
B ← Regs [IR11..15]
Inm ← ((IR16)16 ## IR16..31)

Implementación simple del MIPS

- **3. Ciclo de ejecución / dirección efectiva**
  - La ALU opera sobre los operandos preparados en el paso anterior, realizando una de cuatro funciones, dependiendo del tipo de instrucción DLX.
  - **Referencia a memoria**
    - **ALUoutput ← A + Inm**
    - **Operación:** La ALU suma los operandos para formar la dirección efectiva.

3. Ciclo de ejecución / dirección efectiva

Referencia a memoria.



ALUoutput ← A + Inm

Implementación simple del MIPS

3. Ciclo de ejecución / dirección efectiva

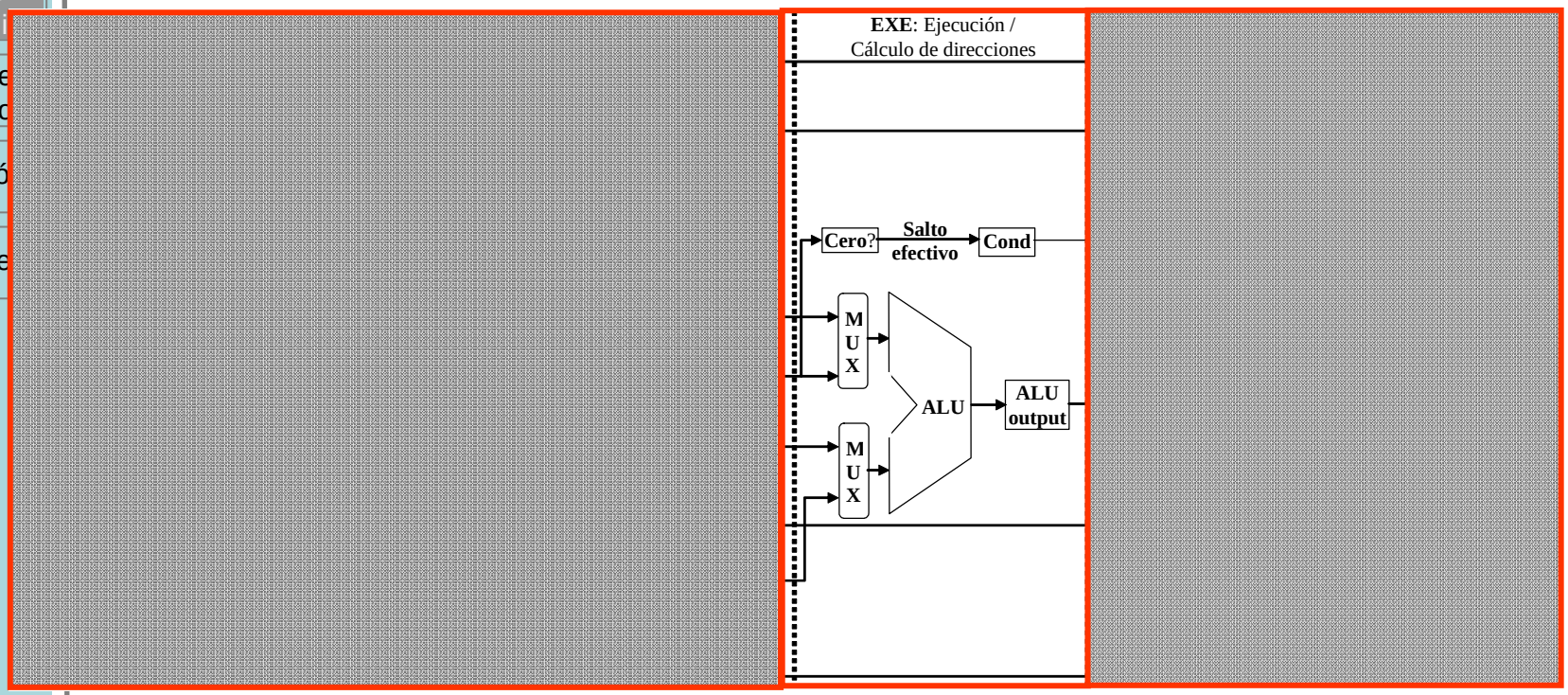
- La ALU opera sobre los operandos preparados en el paso anterior, realizando una de cuatro funciones, dependiendo del tipo de instrucción DLX.

Instrucción ALU registro-registro

- **ALUoutput ← A func B**
- **Operación**: La ALU realiza la operación especificada por el código de operación sobre el valor de A y sobre el valor de B. El resultado se coloca en el registro temporal ALUoutput.

3. Ciclo de ejecución / dirección efectiva

Instrucción ALU registro-registro



ALUoutput ← A func B

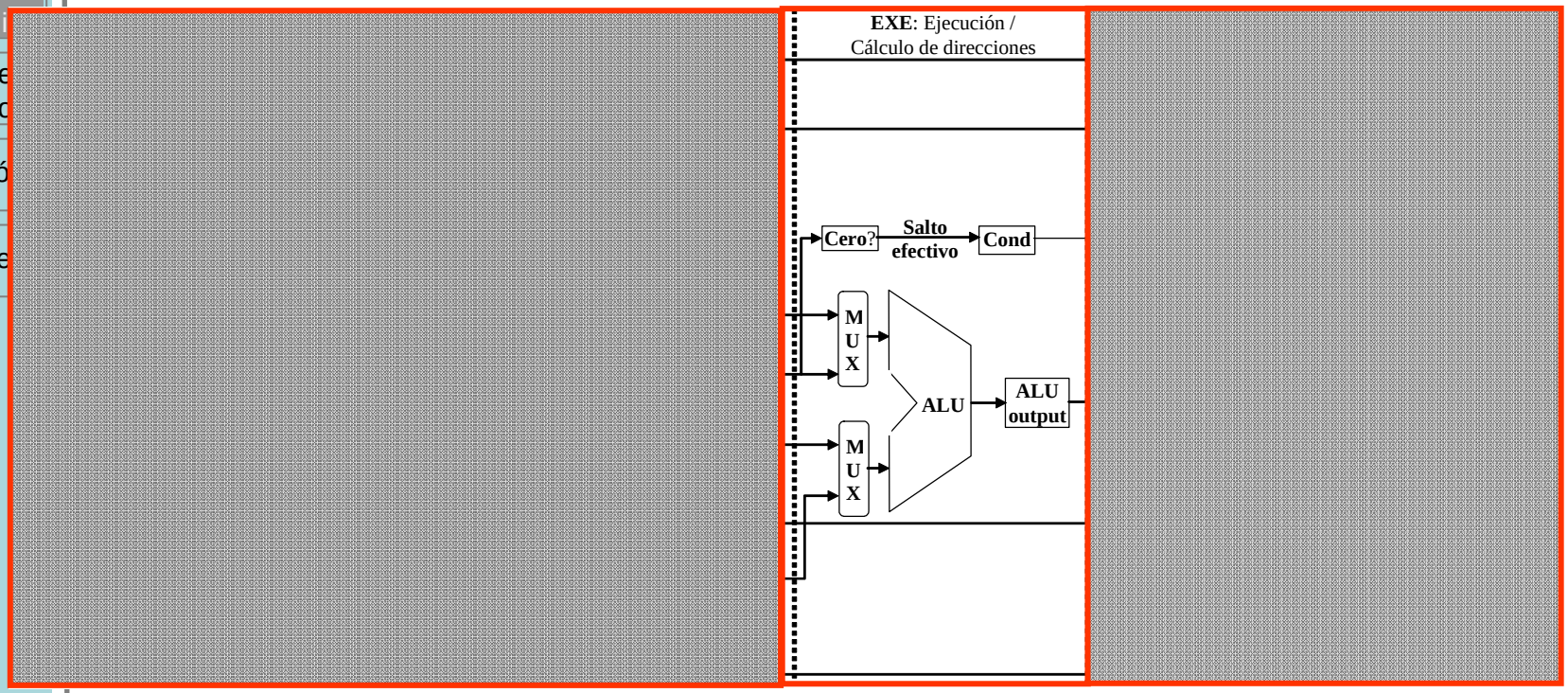
Implementación simple del MIPS

- **3. Ciclo de ejecución / dirección efectiva**
  - La ALU opera sobre los operandos preparados en el paso anterior, realizando una de cuatro funciones, dependiendo del tipo de instrucción DLX.
- **Instrucción ALU registro-inmediato**
  - **ALUoutput ← A op Inm**
  - **Operación**: La ALU realiza la operación especificada por el código de operación sobre el valor de A y sobre el valor del registro Inm. El resultado se coloca en el registro temporal ALUoutput.

Implementación simple del MIPS

3. Ciclo de ejecución / dirección efectiva

Instrucción ALU registro-inmediato



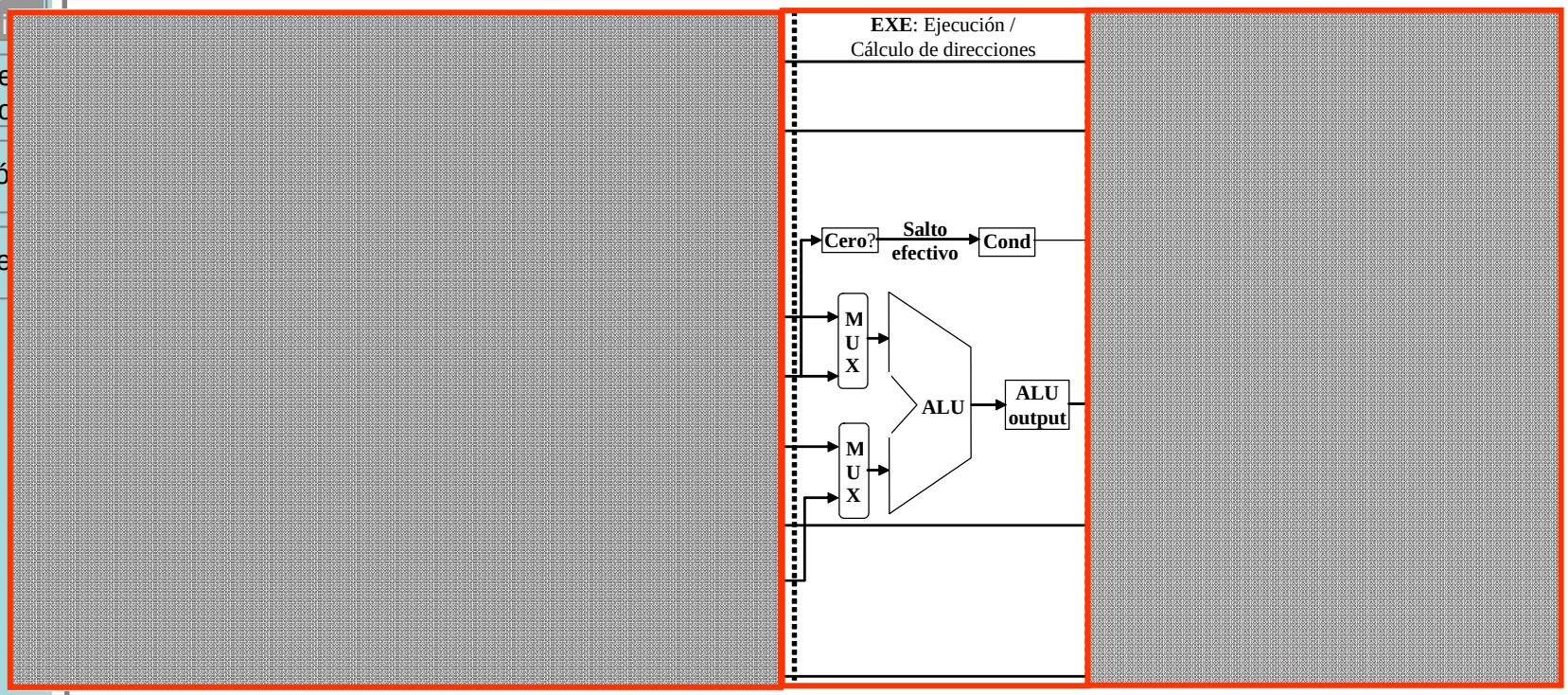
ALUoutput ← A op Inm

Implementación simple del MIPS

- **3. Ciclo de ejecución / dirección efectiva**
  - La ALU opera sobre los operandos preparados en el paso anterior, realizando una de cuatro funciones, dependiendo del tipo de instrucción DLX.
- **Salto / bifurcación**
  - **ALUoutput ← NPC + Inm**
  - **cond ← (A op 0)**
  - **Operación:**
    - La ALU suma el NPC al valor inmediato de signo extendido para calcular la dirección destino del salto.
    - El registro A se examina para decidir si se realiza el salto o no. Op es el operador relacional determinado por el código de operación, (Eje: op es = para la instrucción BEQZ)

3. Ciclo de ejecución / dirección efectiva

Salto / bifurcación



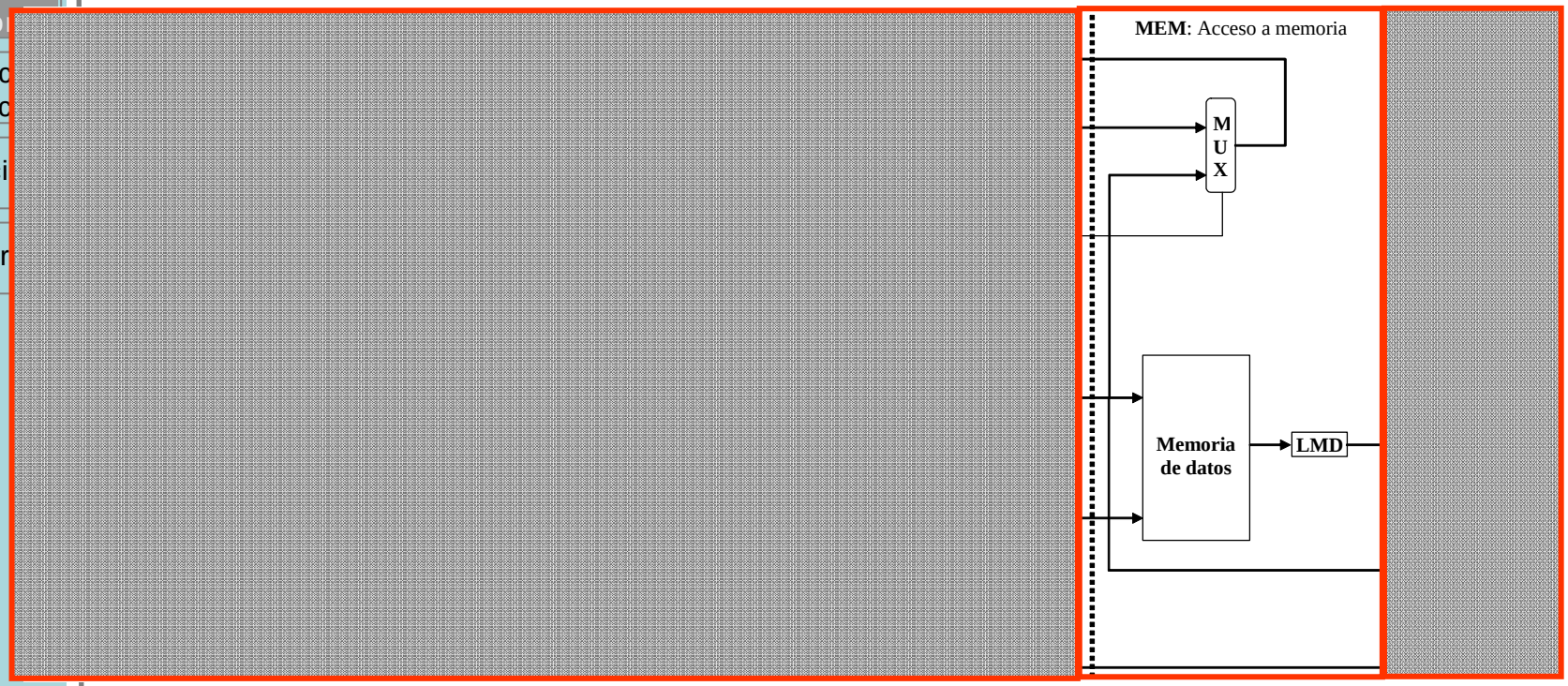
ALUoutput ← NPC + Inm
cond ← (A op 0)

Implementación simple de DLX

- **4. Paso de acceso a memoria / completar salto:**
  - Las únicas instrucciones DLX activas en este paso son cargas, almacenamientos y saltos.
- **Referencia a memoria**
  - **LMD ← MEM [ALUoutput] o**
  - **MEM [ALUoutput] ← B**
  - **Operación:** Accede a memoria si es necesario.
  - **Si la instrucción es una carga**, se lee el dato de memoria y se carga en LMD (load memory data);
  - **Si es un almacenamiento**, escribe el dato del registro B en memoria.
  - En cualquier caso la dirección utilizada es la calculada durante el paso anterior y almacenada en ALUoutput.

Implementación simple del MIPS

- **4. Paso de acceso a memoria / completar salto:**
  - **Referencia a memoria**



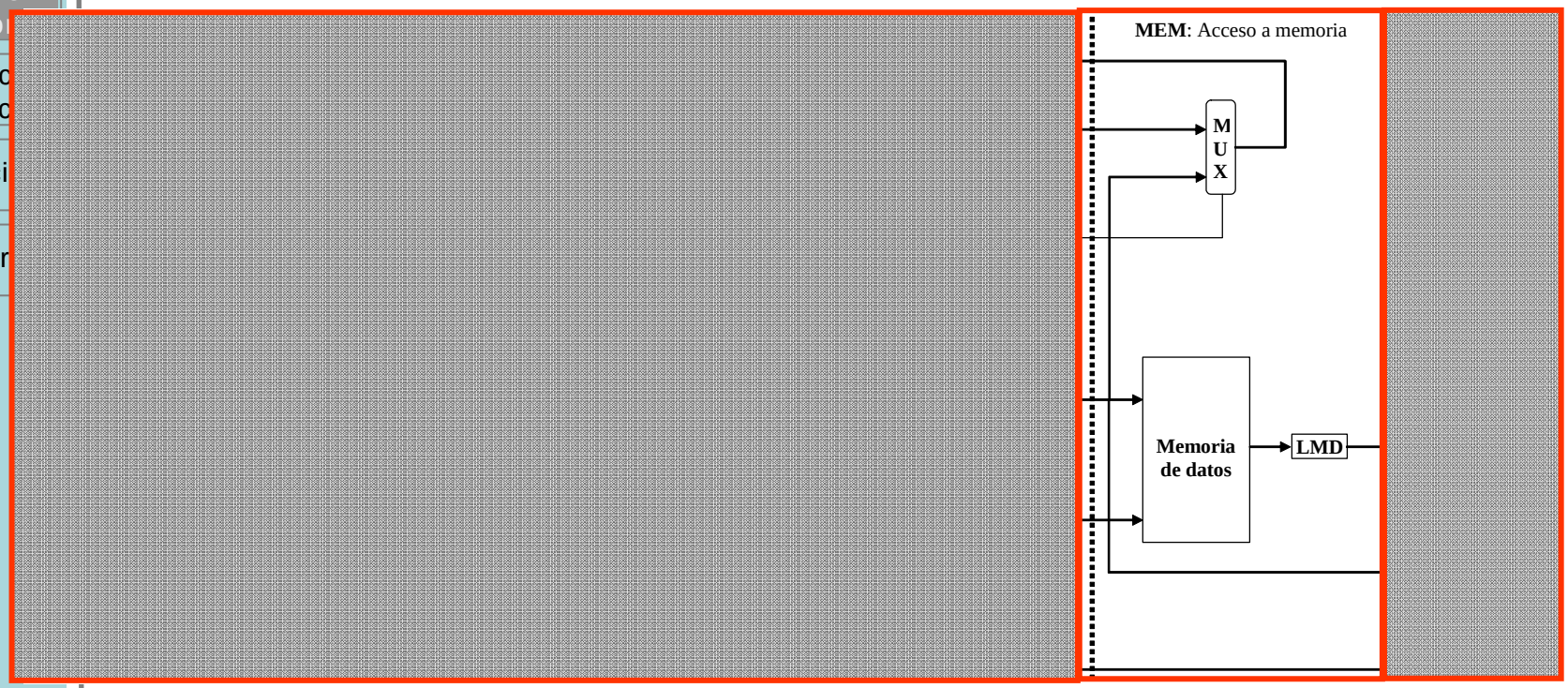
PC ← NPC
LMD ← MEM [ALUoutput] o
MEM [ALUoutput] ← B

Implementación simple del MIPS

- **4. Paso de acceso a memoria / completar salto:**
  - Las únicas instrucciones DLX activas en este paso son cargas, almacenamientos y saltos.
- **Salto**
  - **If (cond) then PC ← ALUoutput else PC ← NPC**
  - **Operación:** Si la instrucción salta, el PC es sustituido por la dirección destino del salto del registro ALUoutput, en caso contrario, se reemplaza con el contador de programa incrementado en el registro NPC.

Implementación simple del MIPS

- **4. Paso de acceso a memoria / completar salto:**
  - **Salto**



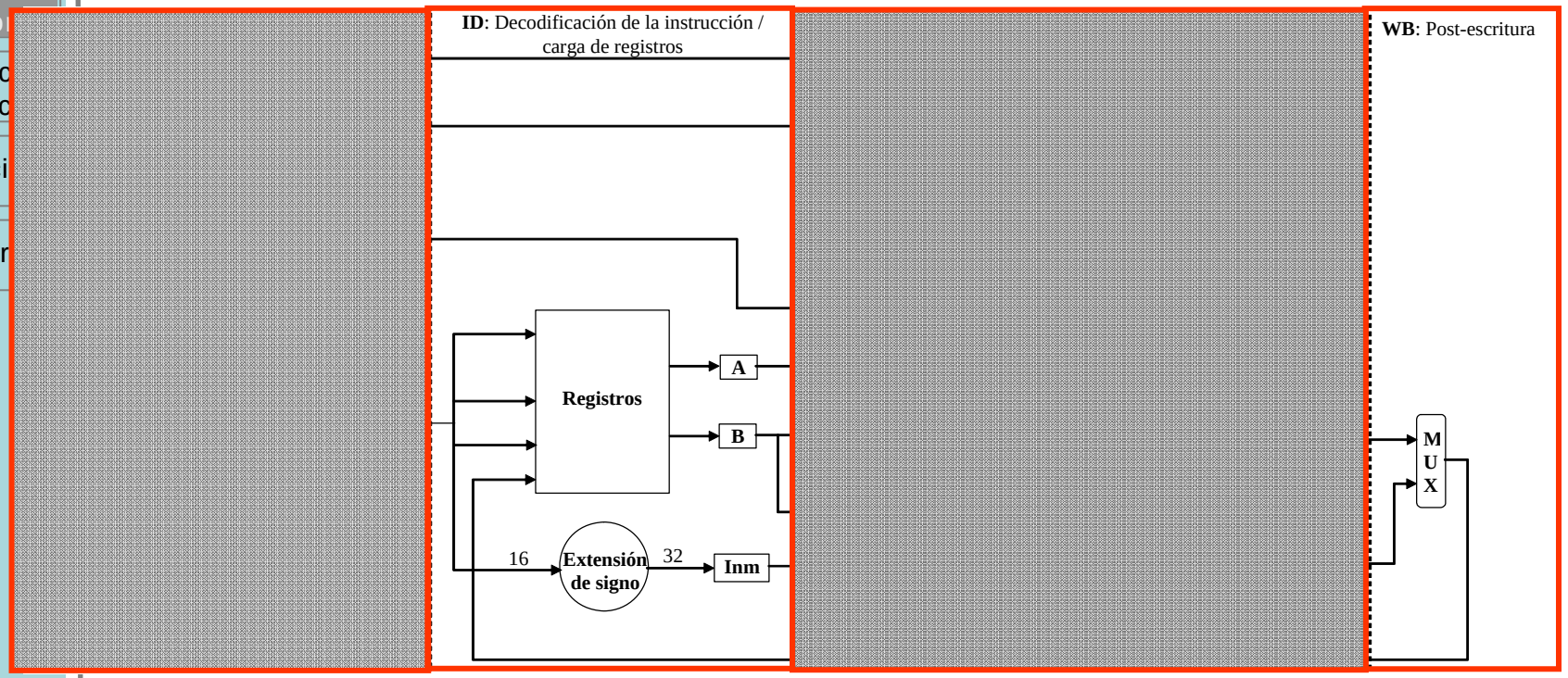
If (cond) PC ← ALUoutput
else PC ← NPC

Implementación simple del MIPS

5. Paso de postescritura (write-back)

- **Instrucciones ALU registro-registro:**
  - **Regs [$IR_{16..20}$] ← ALUoutput**
- **Instrucciones ALU registro-inmediato:**
  - **Regs [$IR_{11..15}$] ← ALUoutput**
- **Instrucciones load**
  - **Regs [$IR_{11..15}$] ← LMD**
- **Operación:** Escribir el resultado en el banco de registros tanto si proviene del sistema de memoria (en LMD) como de la ALU (en ALUoutput).
- El campo del registro destino puede estar en dos posiciones dependiendo del código de operación.

5. Paso de postescritura (write-back)



Instrucciones ALU R-R:

Regs [$IR_{16..20}$] ← ALUoutput

Instrucciones ALU R-I:

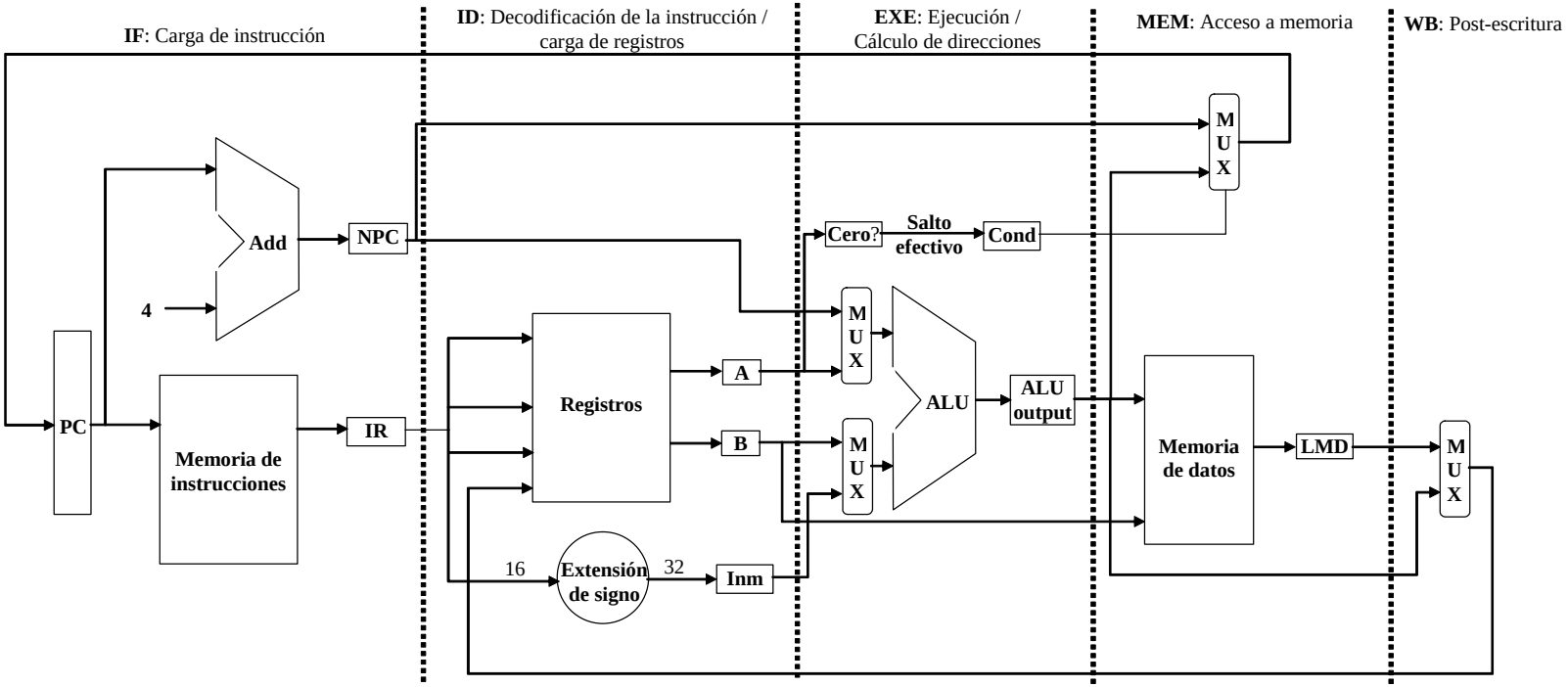
Regs [$IR_{11..15}$] ← ALUoutput

Instrucciones load

Regs [$IR_{11..15}$] ← LMD

Implementación simple del MIPS

Evolución de una instrucción en el camino de datos



- Observamos el PC y Registros en el ciclo de reloj en el que son leídos
- En ambos casos las escrituras en etapas posteriores se indican mediante multiplexores en las salidas (en acceso a memoria y post-escritura) que retornan resultados al PC o a los registros

Segmentación básica del MIPS

- Podemos segmentar el camino de datos sin más que iniciar una nueva instrucción en cada ciclo de reloj
- Cada ciclo de reloj se convierte en una etapa del cauce.

Número de instrucción	1	2	3	4	Ciclo 5	Reloj 6	7	8	9
Inst i	IF	ID	EX	MEM	WB				
Inst i+1		IF	ID	EX	MEM	WB			
Inst i+2			IF	ID	EX	MEM	WB		
Inst i+3				IF	ID	EX	MEM	WB	
Inst i+4					IF	ID	EX	MEM	WB

- Durante un ciclo de reloj el hardware ejecuta alguna parte de cinco instrucciones diferentes.
- Si comienza una instrucción nueva cada ciclo de reloj, el rendimiento mejora hasta cinco veces con respecto a una máquina no encauzada

Segmentación básica del MIPS

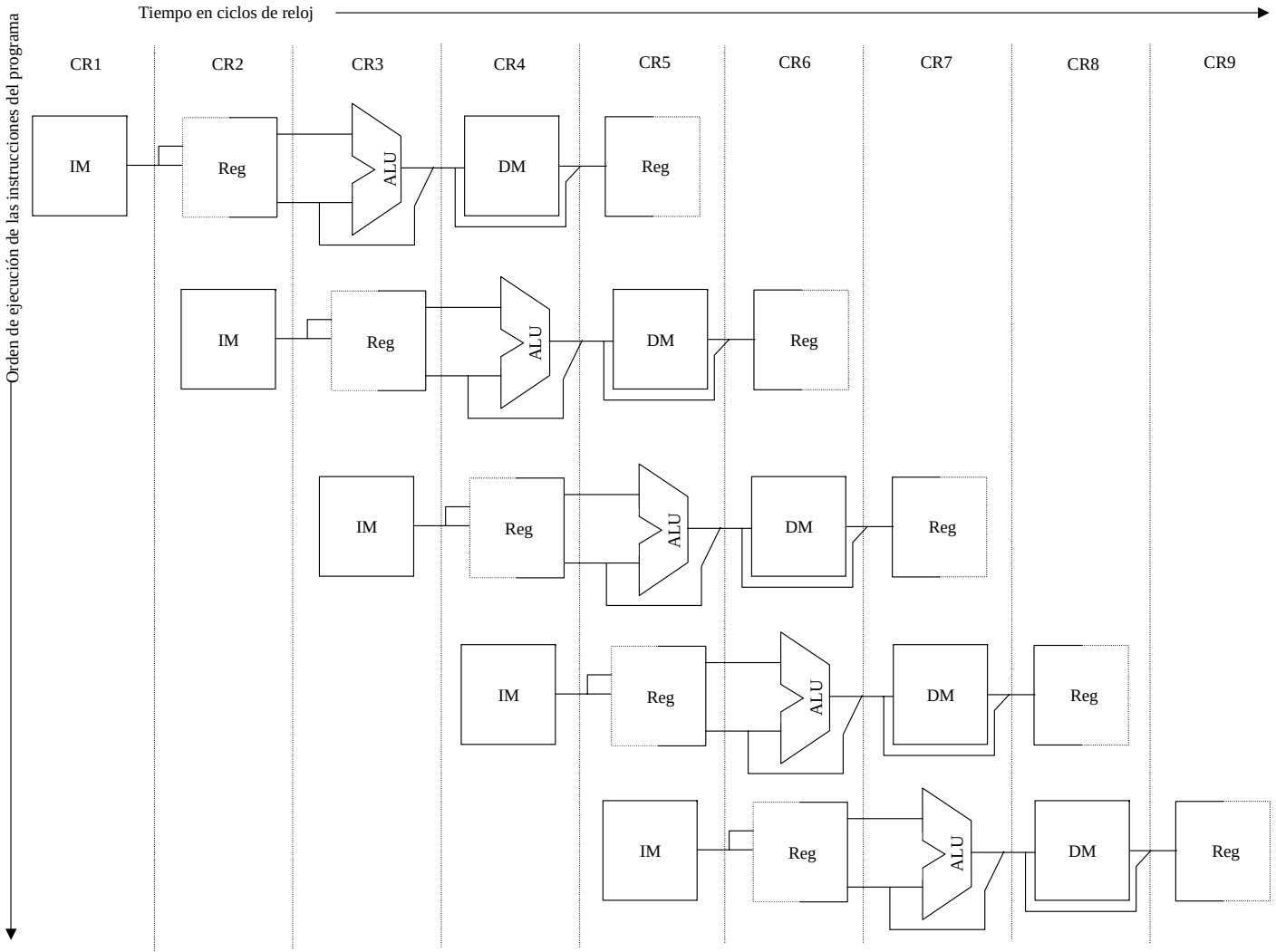
- Podemos segmentar el camino de datos sin más que iniciar una nueva instrucción en cada ciclo de reloj
- Cada ciclo de reloj se convierte en una etapa del cauce

Número de instrucción	1	2	3	4	Ciclo 5	Reloj 6	7	8	9
Inst i	IF	ID	EX	MEM	WB				
Inst i+1		IF	ID	EX	MEM	WB			
Inst i+2			IF	ID	EX	MEM	WB		
Inst i+3				IF	ID	EX	MEM	WB	
Inst i+4					IF	ID	EX	MEM	WB

- Iniciar una instrucción cada ciclo introduce **problemas**
- Tenemos que determinar que ocurre en cada ciclo de reloj de la máquina y asegurarnos que no estamos intentando realizar dos operaciones diferentes con el mismo recurso
- Ejemplo: una simple ALU calculando una dirección efectiva y una operación de resta al mismo tiempo

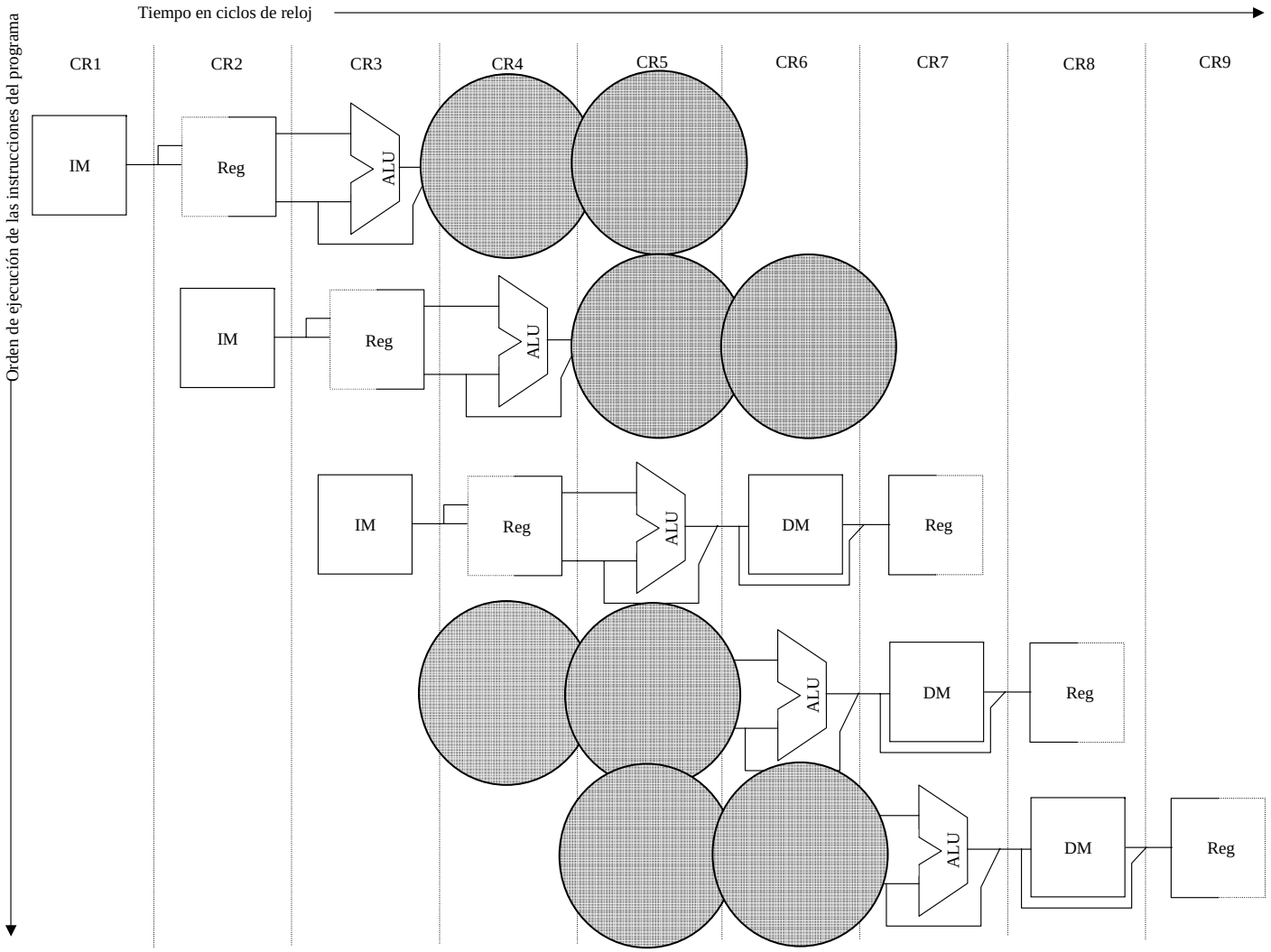
Segmentación básica del MIPS

Evaluación de conflictos entre recursos para el cauce DLX

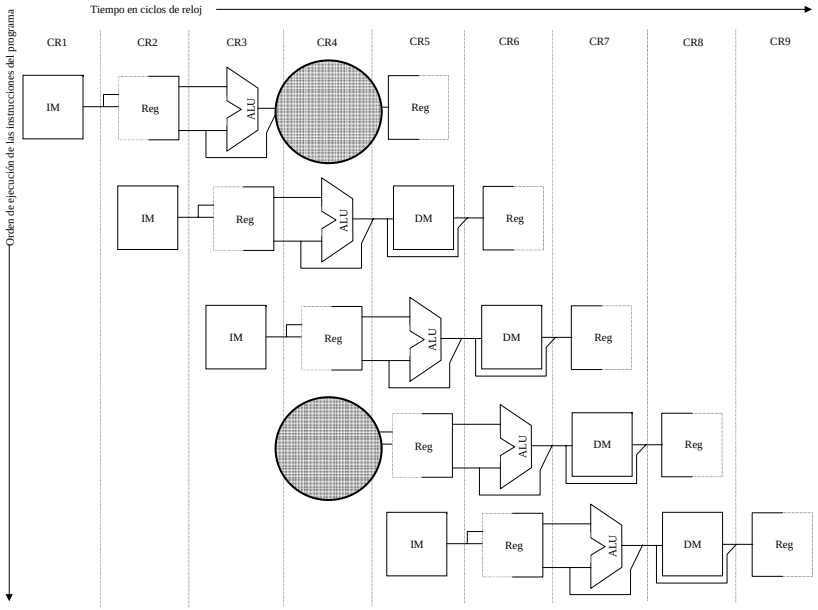


Segmentación básica del MIPS

Evaluación de conflictos entre recursos para el cauce DLX



Segmentación básica del MIPS



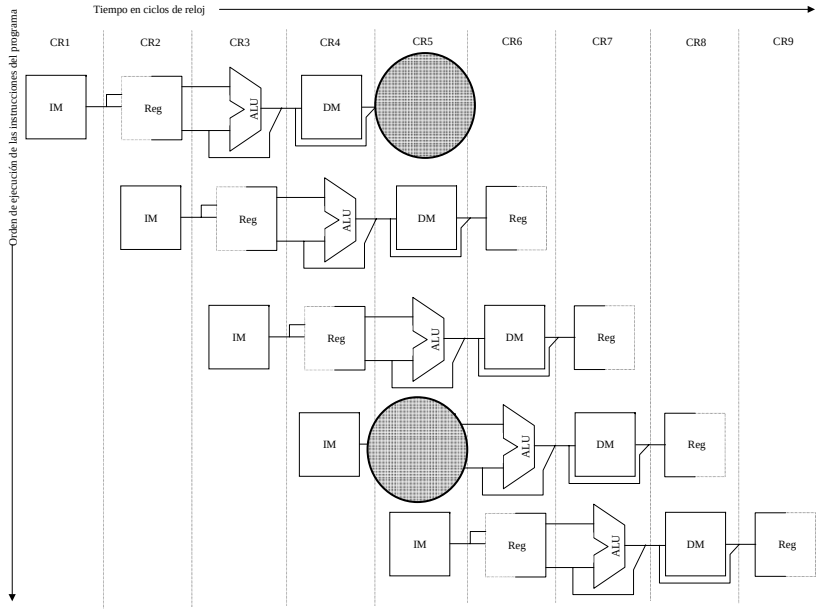
La memoria:

- Memorias de datos e instrucciones separadas con caches separadas de datos e instrucciones
- Elimina conflicto entre la búsqueda de una instrucción y acceso a los datos

La memoria

- Si la máquina encauzada tiene un ciclo de reloj como el de la máquina no encauzada, el sistema de memoria debe proporcionar un ancho de banda cinco veces mayor
- Máquina no encuazada (2 accesos cada cinco ciclos)
- Máquina encauzada (2 accesos cada ciclo)

Segmentación básica del MIPS



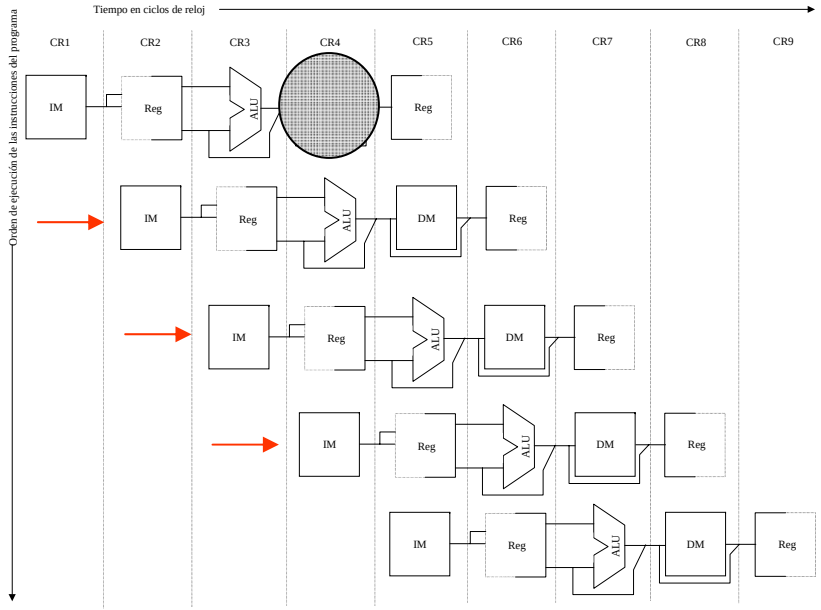
El banco de registros

- El banco de registros se usa en dos etapas
- Para leer en ID
- Para escribir en WB
- Dos lecturas y una escritura en cada ciclo

El banco de registros

- ¿Qué ocurre si una lectura y una escritura se realizan sobre el mismo registro?

Segmentación básica del MIPS



El PC

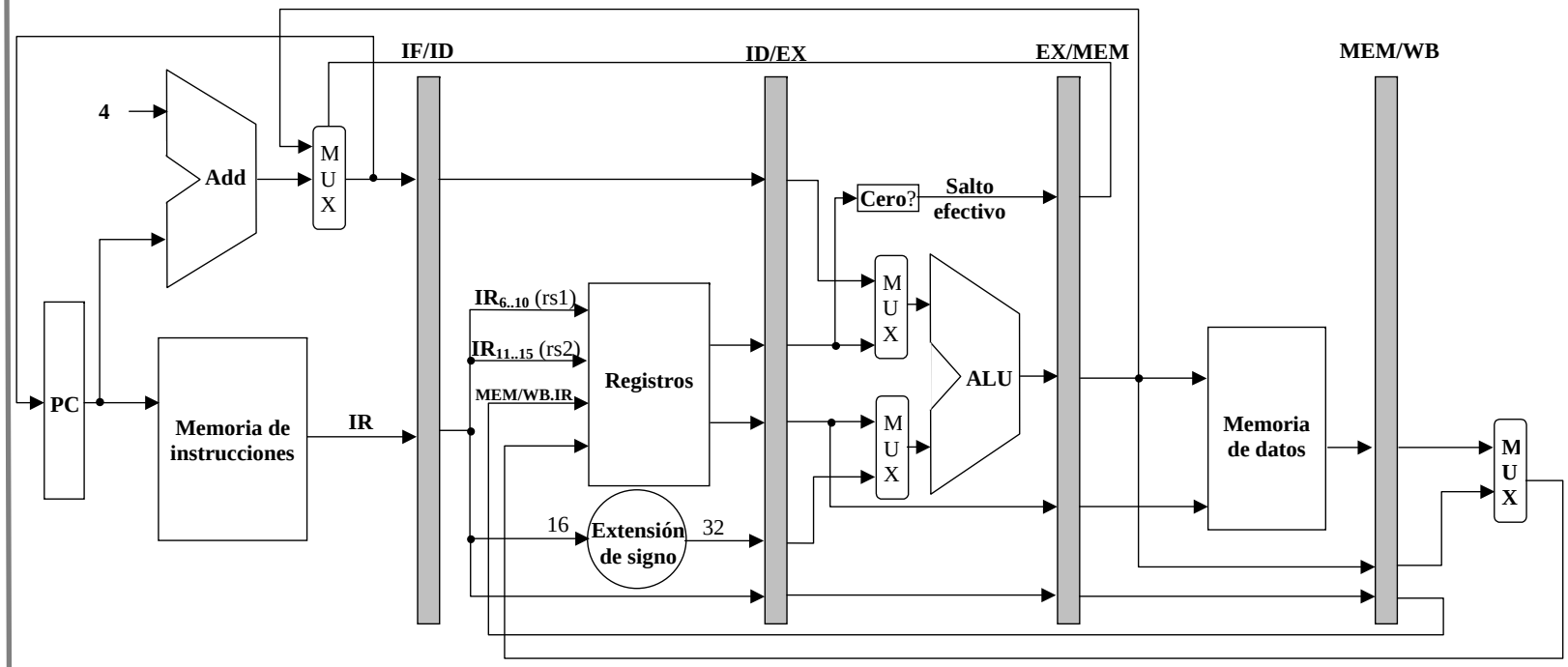
- Iniciar una nueva instrucción cada ciclo de reloj requiere incrementar y guardar el PC en cada ciclo, durante la etapa IF
- Problema con los saltos, que cambian el PC, pero no lo hacen hasta la etapa MEM

El PC

- En el caso encauzado, organizaremos el camino de datos para escribir el PC en IF bien con el PC incrementado en 4 o el valor del destino del salto de una instrucción de salto ejecutada previamente

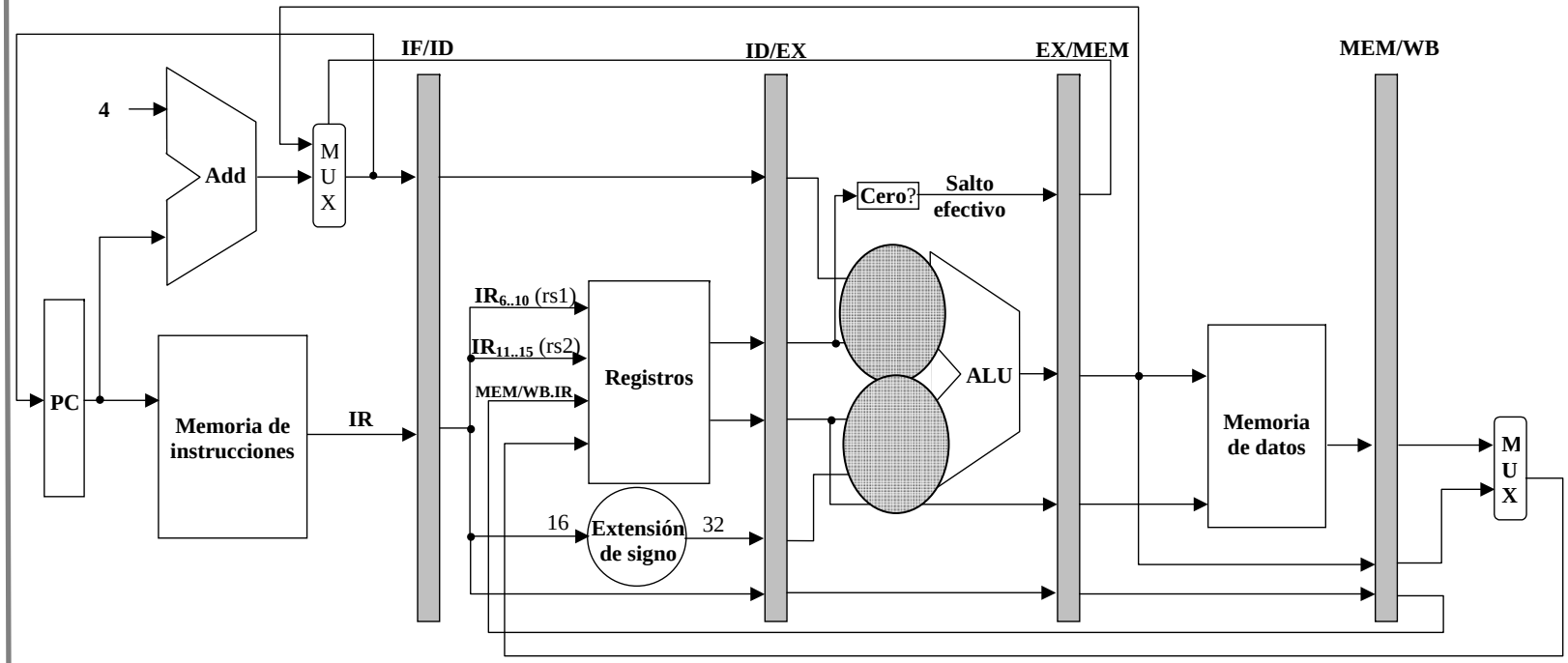
Segmentación básica del MIPS

Los registros intermedios



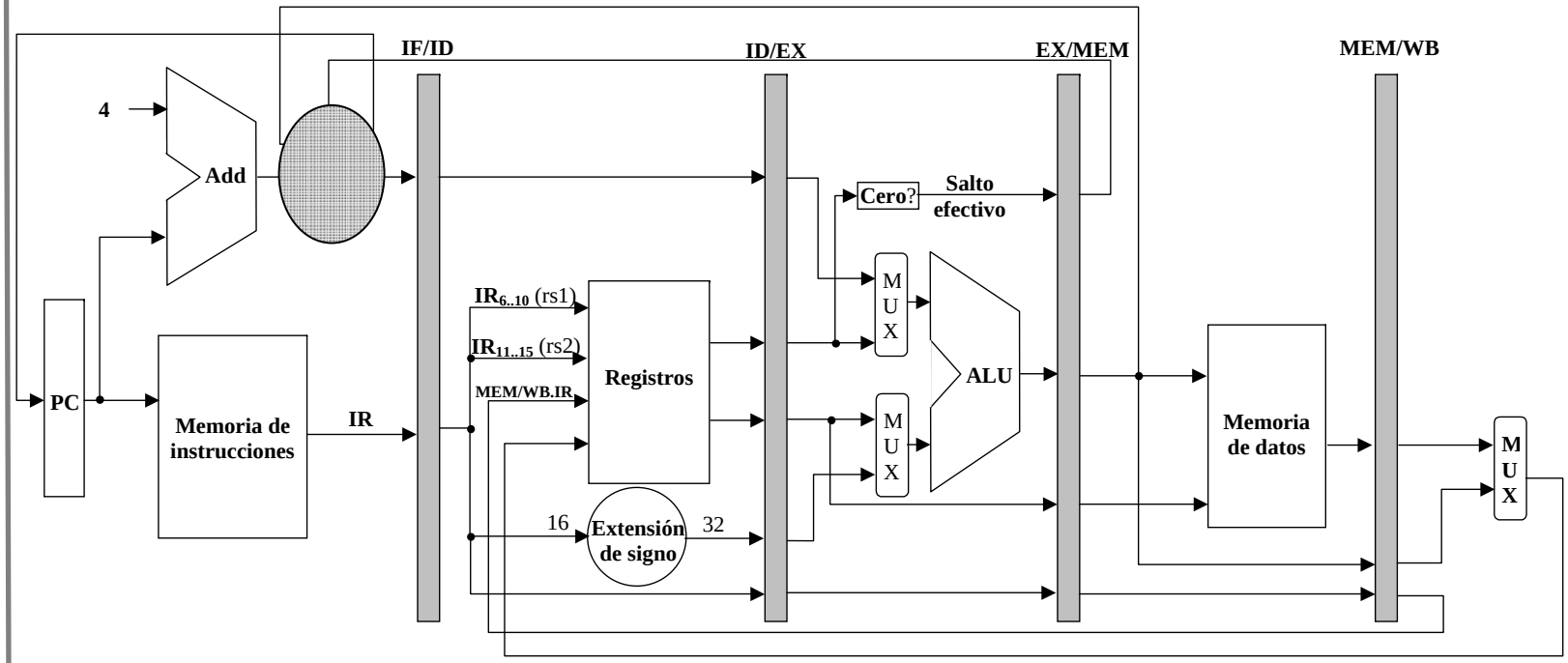
- Los registros intermedios sirven para transportar valores e información de control desde una etapa a la siguiente
- **Ejemplo**, campo de IR que apunta al operando registro destino de una operación load o ALU se proporciona desde el registro intermedio MEM/WB en lugar de obtenerlo de IF/ID.

Segmentación básica del MIPS



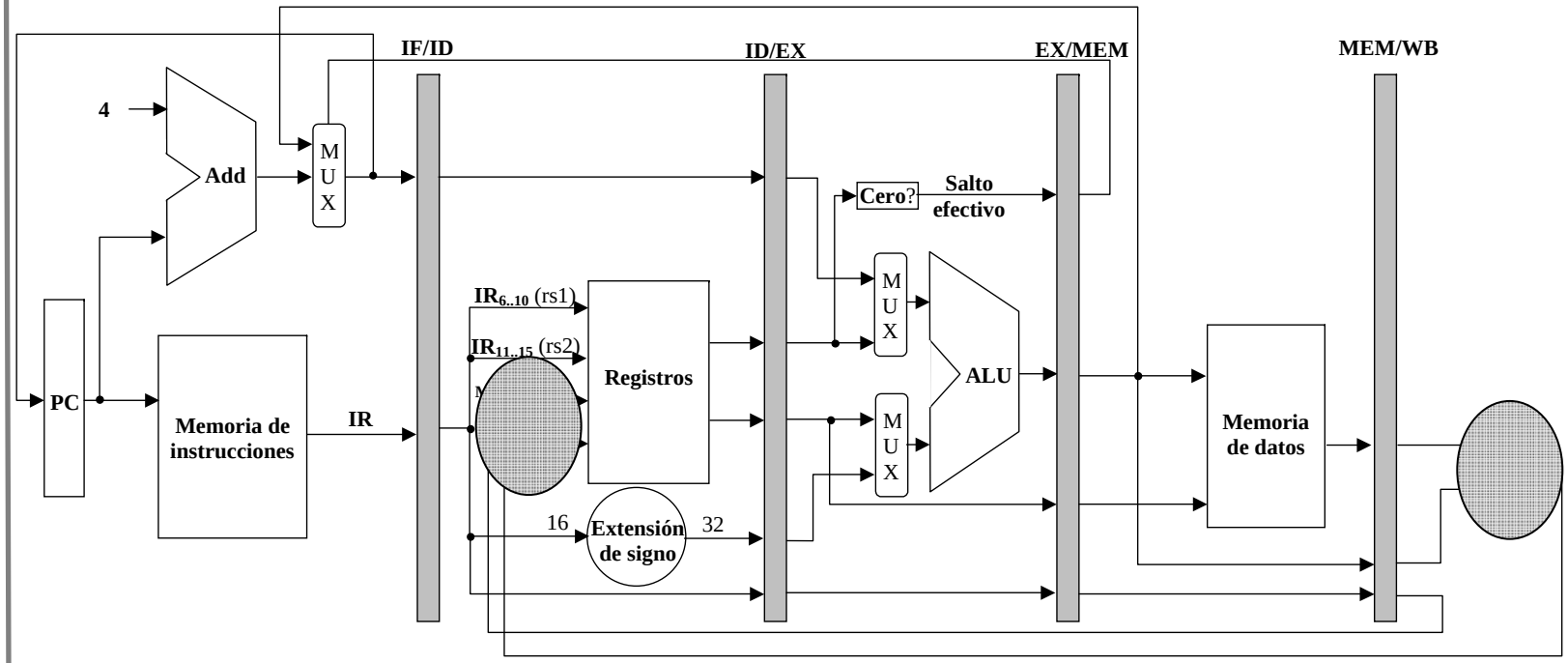
- **Los multiplexores**
  - **Los dos multiplexores de la ALU:**
    - Las entradas se establecen dependiendo del tipo de instrucción, que se encuentra en el campo IR del registro intermedio ID/EX
    - **Multiplexor alto**: se establece según la instrucción sea un salto o no
    - **Multiplexor bajo**: se establece según la instrucción sea ALU registro-registro o cualquier otro tipo de operación

Segmentación básica del MIPS



- **Los multiplexores**
  - **El multiplexor en la etapa IF**
  - El multiplexor que selecciona el valor que será escrito en el PC se ha movido a la etapa IF, de forma que el PC sólo puede ser escrito durante la etapa IF
  - Selecciona el uso del PC en curso o el valor de EX/MEM.ALUoutput como dirección de instrucción. Este multiplexor se controla mediante el campo EX/MEM.cond

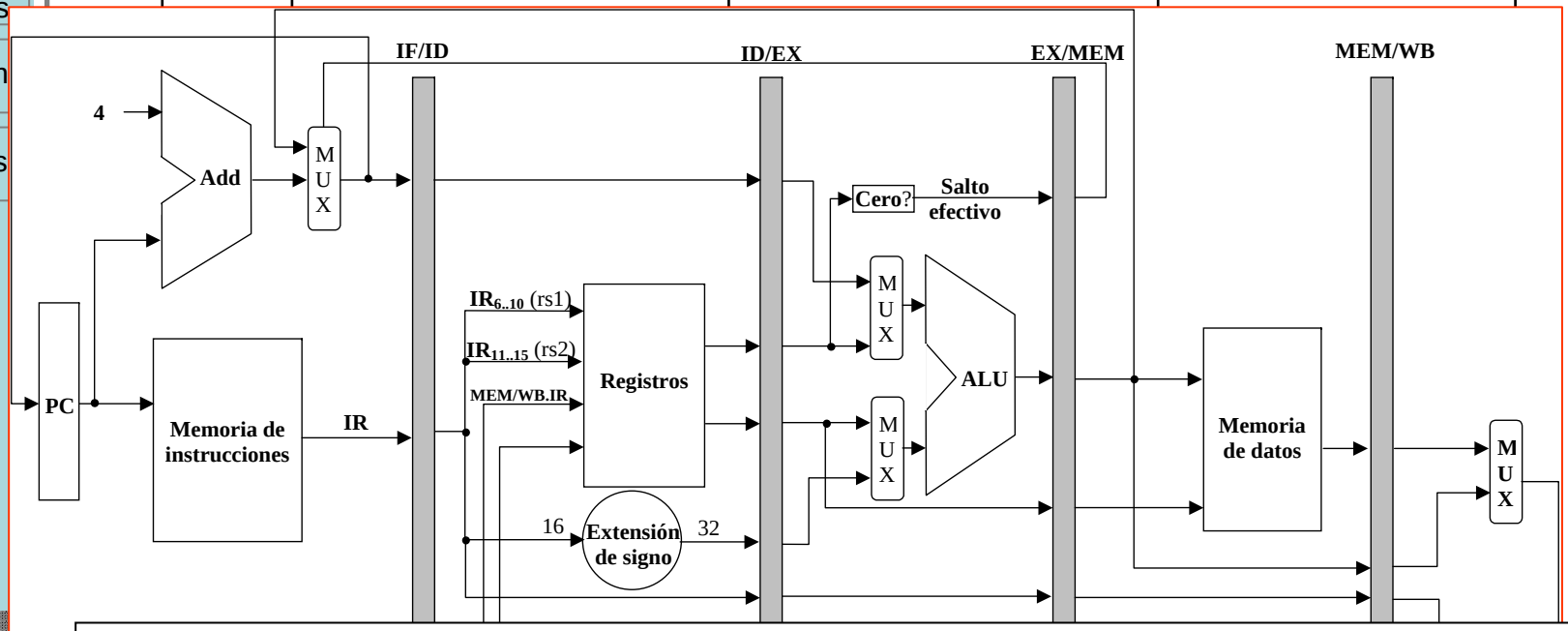
Segmentación básica del MIPS



- **Los multiplexores**
  - **El cuarto multiplexor**
  - Se controla según la instrucción en la etapa WB sea una operación ALU o una load.
  - Es necesario un multiplexor más para controlar si el campo de IR que expresa el registro destino está en una u otra posición según la instrucción sea ALU registro-registro frente a ALU inmediato o load.

Segmentación básica del MIPS

IF	**IF/ID.IR** ← Mem[PC]; **IF/ID.NPC, PC** ← (if EX/MEM.cond {EX/MEM.ALUoutput} else {PC+4})
ID	**ID/EX.A** ← Regs [IF/ID.IR$_{6..10}$]; ID/EX.B ← Regs [IF/ID.IR$_{11..15}$]; **ID/EX.NPC** ← IF/ID.NPC; **ID/EX.IR** ← IF/ID.IR **ID/EX.Inm** ← (IF/ID.IR$_{16}$)^{16} ## IF/ID.IR$_{16..31}$



lw R1,100(R2) (sw)
add R1,R2,#100
beqz R1,ET

Instrucción tipo I

6	5	5	16
Cód ope	Rs1	Rd	Inmediato

add R1,R2,R3

Instrucción tipo R

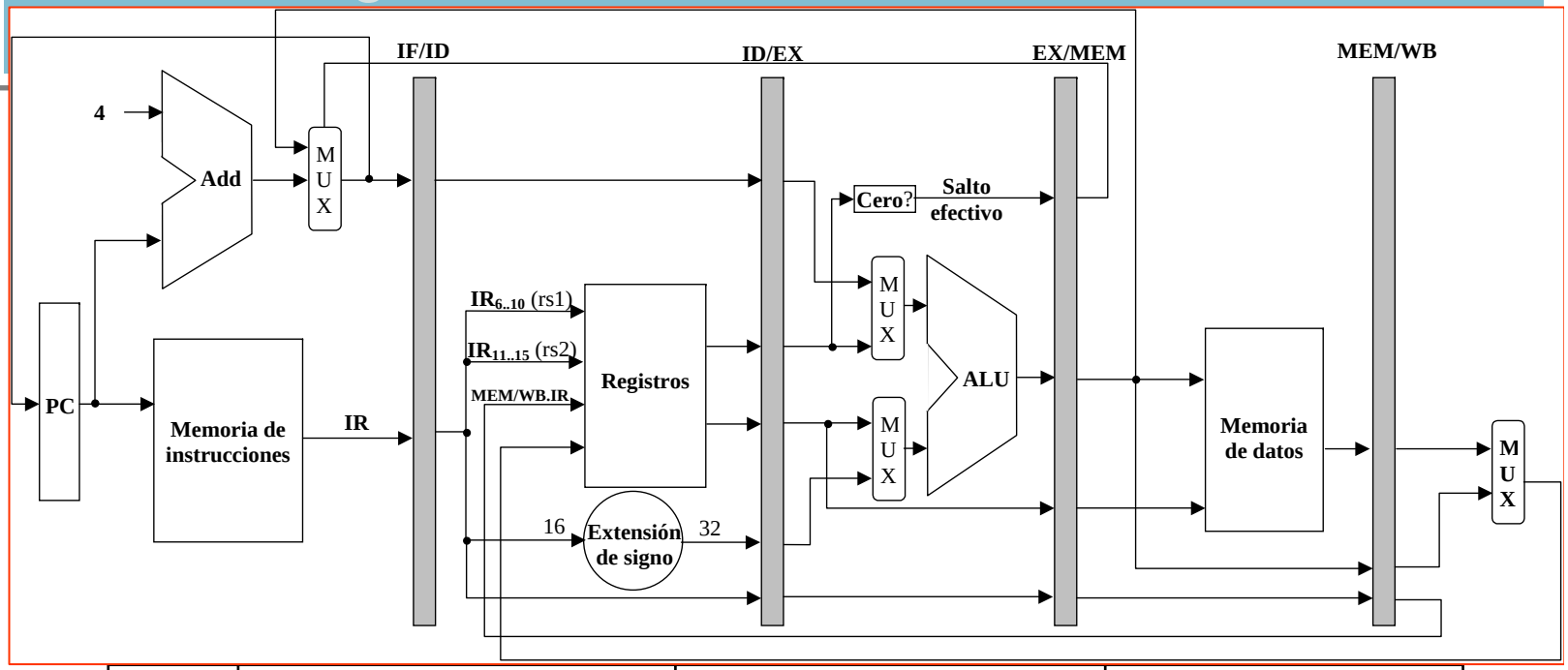
6	5	5	5	11
Cód ope	Rs1	Rs2	Rd	func

J ET

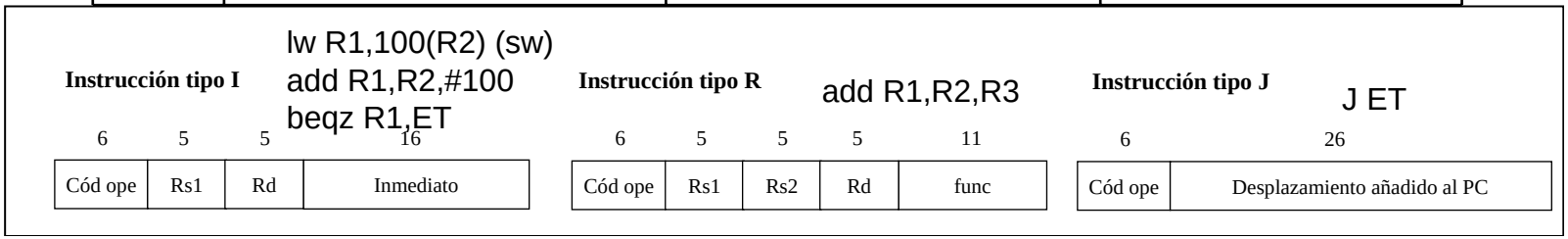
Instrucción tipo J

6	26
Cód ope	Desplazamiento añadido al PC

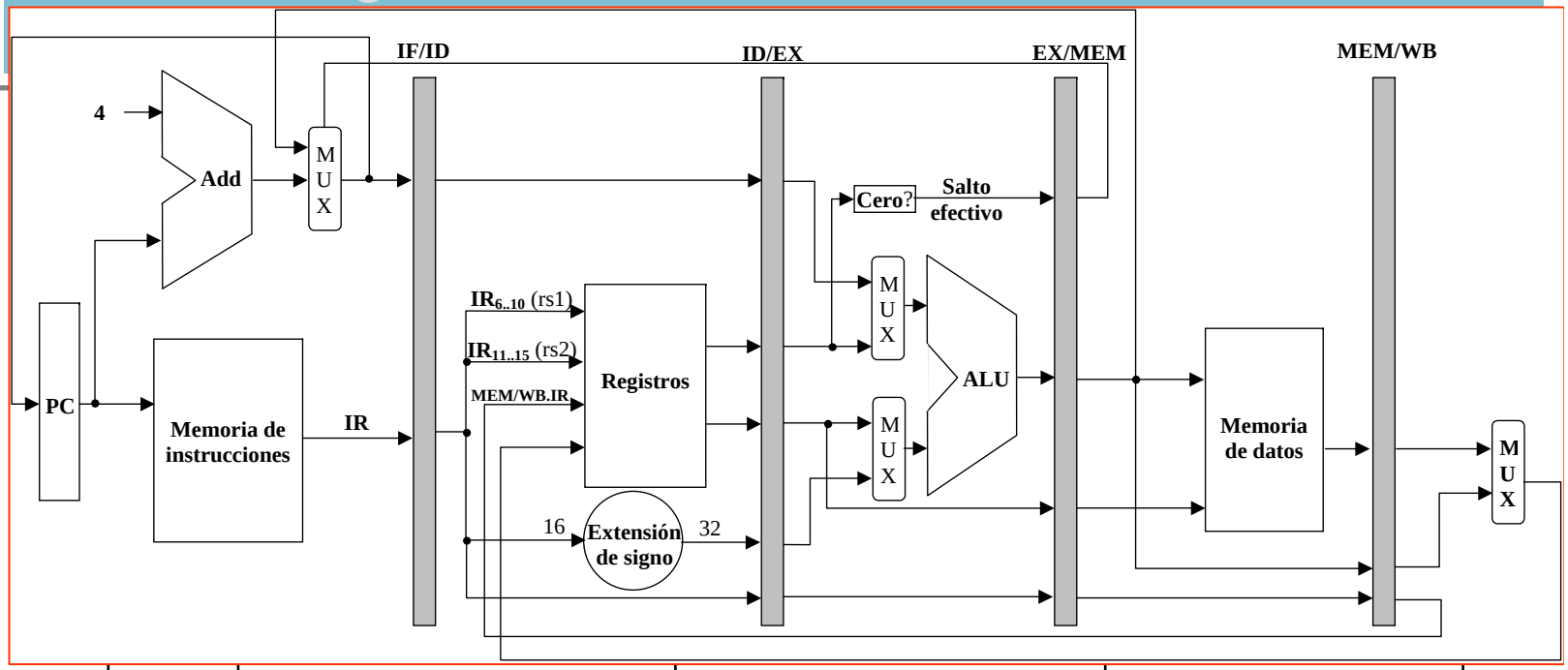
Segmentación básica del MIPS



	Instrucción ALU	Instrucción load o store	Instrucción de salto
EX	**EX/MEM.IR** ← ID/EX.IR; **EX/MEM.ALUoutput** ← ID/EX.A func ID/EX.B; O **EX/MEM.ALUoutput** ← ID/EX.A op ID/EX.Inm; **EX/MEM.Cond** ← 0;	**EX/MEM.IR** ← ID/EX.IR; **EX/MEM.ALUoutput** ← ID/EX.A + ID/EX.Inm; **EX/MEM.cond** ← 0; **EX/MEM.B** ← ID/EX.B;	**EX/MEM.ALUoutput** ← ID/EX.NPC + ID/EX.Inm; **EX/MEM.cond** ← (ID/EX.A op 0);



Segmentación básica del MIPS



	Instrucción ALU	Instrucción load o store	Instrucción de salto
MEM	**MEM/WB.IR**←EX/MEM.IR; **MEM/WB.ALUoutput** ← EX/MEM.ALUoutput;	**MEM/WB.IR**←EX/MEM.IR; **MEM/WB.LMD** ← Mem [EX/MEM.ALUoutput]; O **Mem[EX/MEM.ALUoutput]** ← EX/MEM.B,	

Instrucción tipo I — lw R1,100(R2) (sw), add R1,R2,#100, beqz R1,ET

6	5	5	16
Cód ope	Rs1	Rd	Inmediato

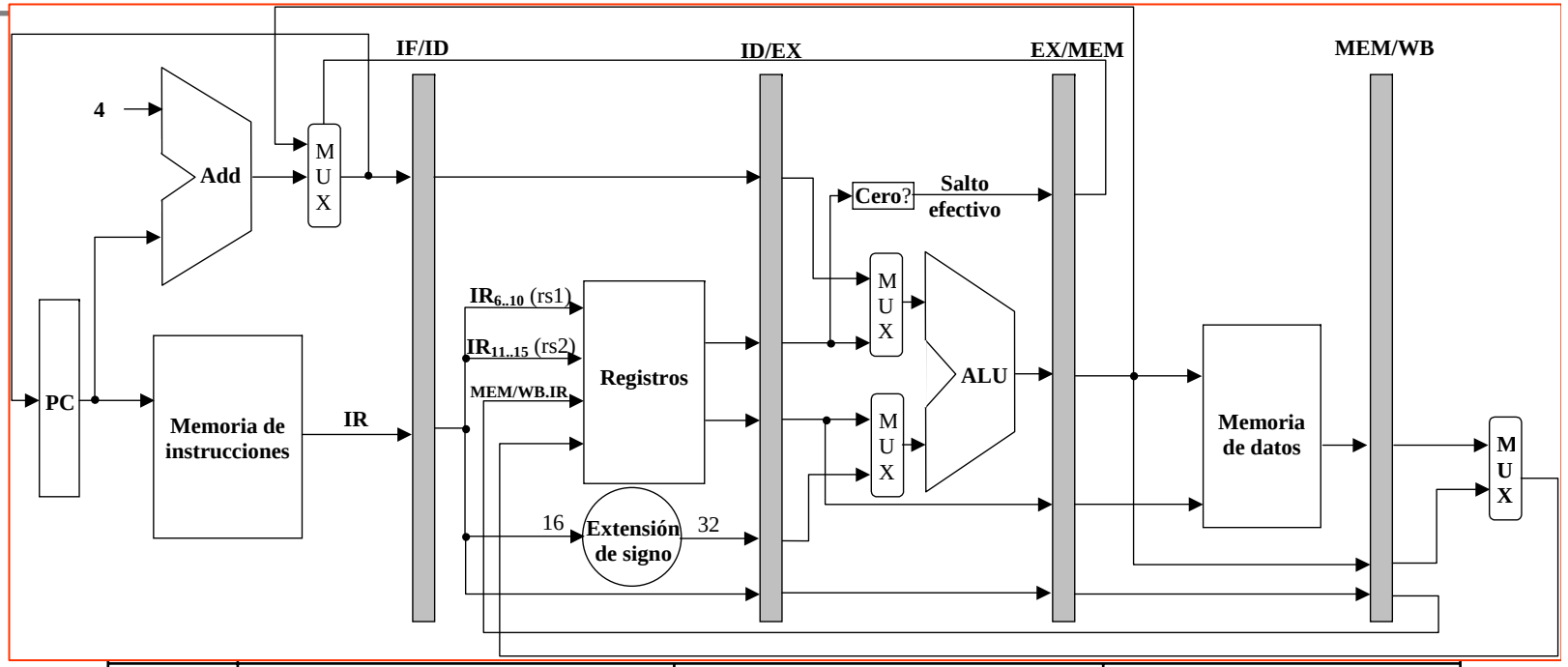
Instrucción tipo R — add R1,R2,R3

6	5	5	5	11
Cód ope	Rs1	Rs2	Rd	func

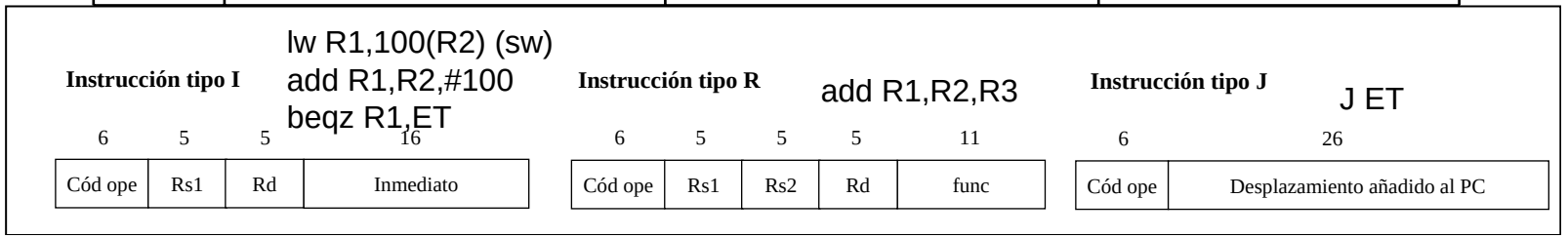
Instrucción tipo J — J ET

6	26
Cód ope	Desplazamiento añadido al PC

Segmentación básica del MIPS



	Instrucción ALU	Instrucción load o store	Instrucción de salto
WB	**Regs[MEM/WB.IR$_{16..20}$]** ← MEM/WB.ALUoutput; O **Regs[MEM/WB.IR$_{11..15}$]** ← MEM/WB.ALUoutput;	**Regs[MEM/WB.IR$_{11..15}$]** ← MEM/WB.LMD;	



Segmentación básica del MIPS

- **Rendimiento segmentado**
- La segmentación **incrementa la productividad** de las instrucciones de la CPU (número de instrucciones completas por unidad de tiempo) pero **no reduce** el tiempo de **ejecución de una instrucción individual**
- Normalmente se ve incrementado el tiempo de ejecución de instrucciones individuales debido a la sobrecarga en el control de la segmentación
- El incremento en la productividad de instrucciones significa que los **programas se ejecutan más rápido** y tienen **menor tiempo total de ejecución**

Segmentación básica del MIPS

- **Ejemplo.** Considerando la máquina no segmentada de la sección previa. Supongamos que tiene un ciclo de reloj de 10ns y que usa cuatro ciclos para las operaciones ALU y saltos y cinco ciclos para las operaciones de memoria. Supongamos que las frecuencias relativas para estas operaciones son 40%,20% y 40% respectivamente. Suponemos que segmentar la máquina añade 1ns de recarga al ciclo de reloj. Ignorando cualquier impacto sobre la latencia, ¿Qué ganancia en el índice de ejecución de instrucciones se obtiene de la segmentación?

Segmentación básica del MIPS

- **Máquina sin segmentación**

Tiempo de ejecución medio por instrucción = T_{emi}= clk x CPI

T_{emi}= 10 ns x ((0,4+0,2)x4 + 0,4x5) = 10 x 4,4 = 44 ns

- **Implementación segmentada**

El ciclo de reloj debe ir a la velocidad de la etapa más lenta más sobrecargas, que puede ser 10+1 = 11 ns; este es el tiempo de ejecución medio por instrucción. Por eso, la ganancia para la segmentación es:

$$G_s = \frac{tiempo\ medio\ instrucción\ \text{sin}\ segmentación}{tiempo\ medio\ instrucción\ con\ segmentación} = \frac{44ns}{11ns} = 4$$

Segmentación básica del MIPS

- **Ejemplo**. Supongamos que los tiempos requeridos para las cinco unidades funcionales, que operan en cada uno de los cinco ciclos son: 10ns, 8ns, 10ns, 10ns y 7ns. Suponemos que la segmentación añade 1ns de sobrecarga. Buscar la ganancia frente a la versión monociclo.

Monociclo

- Máquina no segmentada ejecuta todas las instrucciones en un único ciclo de reloj
- Tiempo medio por instrucción es el tiempo del ciclo de reloj que es la suma de los tiempos para cada paso de la ejecución
- Tiempo de ejecución medio por instrucción

$$T_{emi} = 10 + 8 + 10 + 10 + 7 = 45 \text{ ns}$$

Segmentación básica del MIPS

- **Ejemplo**. Supongamos que los tiempos requeridos para las cinco unidades funcionales, que operan en cada uno de los cinco ciclos son: 10ns, 8ns, 10ns, 10ns y 7ns. Suponemos que la segmentación añade 1ns de sobrecarga. Buscar la ganancia frente a la versión monociclo.

Segmentada

- El tiempo del ciclo de reloj en la máquina segmentada debe ser el tiempo más largo de cada una de las etapas (10ns) más la sobrecarga de 1 ns, sumando un total de 11ns.
- CPI = 1,
- Tiempo de ejecución medio por instrucción de 11ns.

$$G_s = \frac{tiempo\ medio\ instrucción\ \text{sin}\ segmentación}{tiempo\ medio\ instrucción\ con\ segmentación} = \frac{45ns}{11ns} = 4.1$$

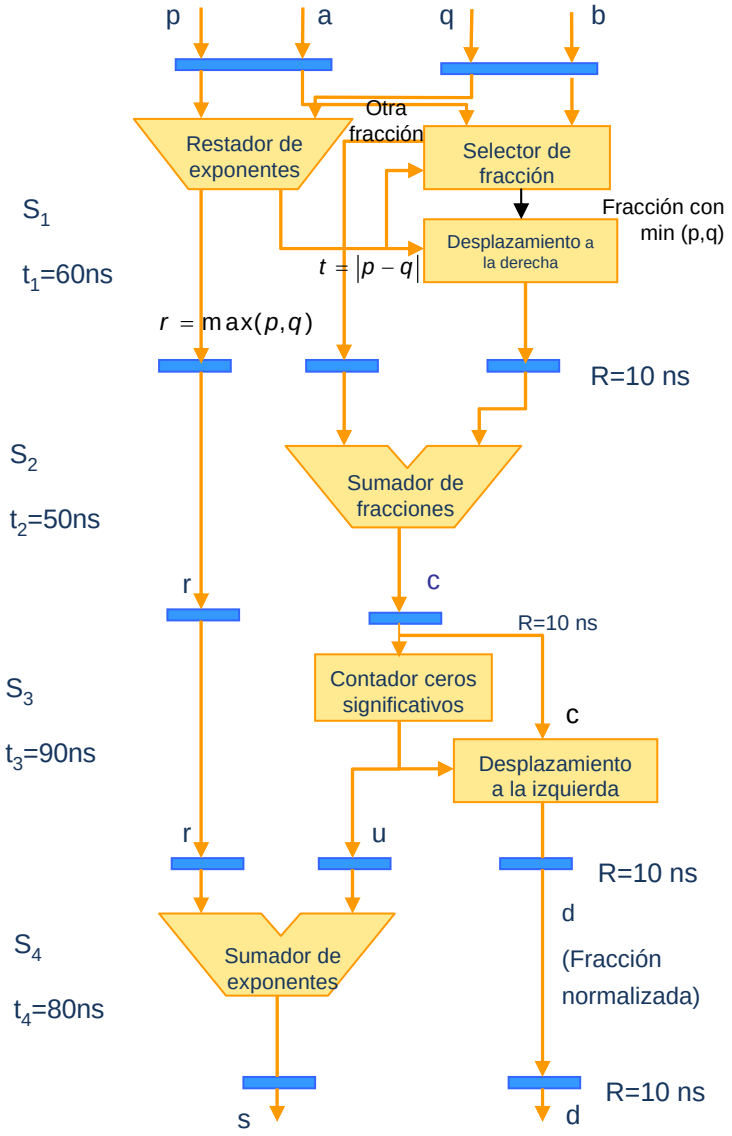
Segmentación básica del MIPS

- **Ejemplo**. Supongamos que los tiempos requeridos para las cinco unidades funcionales, que operan en cada uno de los cinco ciclos son: 10ns, 8ns, 10ns, 10ns y 7ns. Suponemos que la segmentación añade 1ns de sobrecarga. Buscar la ganancia frente a la versión monociclo.

La segmentación puede entenderse como una mejora del CPI, que es lo que típicamente entendemos o como una reducción del ciclo de reloj

Segmentación básica del MIPS

- **Ejemplo**. A partir del sumador en coma flotante encauzado de la figura, suponiendo que van a ser procesadas 106 sumas y que el retardo incorporado por los registros intermedios es 10ns, calcula:
  - La frecuencia de reloj del cauce
  - La ganancia de la segmentación
  - La eficiencia de la segmentación
  - La productividad de la segmentación



Segmentación básica del MIPS

- **Ejemplo**. Para la implementación de un sumador de b bits con propagación de acarreo se van a utilizar como bloques constructivos sumadores completos, donde el Ts=2Tc (Ts tiempo de suma, Tc tiempo de acarreo). Se han realizado tres diseños para el sumador, dos de los diseños operan de forma secuencial y uno de forma segmentada. En el primer diseño secuencial cada sumador completo empieza a operar una vez ha finalizado el cálculo de la suma y el acarreo del sumador inmediatamente anterior. En el segundo diseño secuencial cada sumador completo empieza a operar una vez ha finalizado el cálculo del acarreo del sumador inmediatamente anterior. En el diseño segmentado cada sumador completo empieza a operar una vez ha finalizado el cálculo de la suma y el acarreo del sumador inmediatamente anterior, pero siguiendo una dinámica segmentada que permite el solapamiento de diferentes bits de distintos números. Asumiendo que en el caso segmentado el tiempo de los registros intermedios es despreciable.

Segmentación básica del MIPS

- **Ejemplo**.
  - a) Calcula la ganancia de la segmentación frente al primer caso secuencial, para sumar n números de b bits. ¿Cuánto vale la ganancia?
  - b) Calcula la ganancia de la segmentación frente al segundo caso secuencial, para sumar n números de b bits. ¿Cuánto vale la ganancia?
  - c) Calcula la ganancia del mejor caso secuencial frente al peor, para sumar números de b bits.
  - d) Calcula los resultados de los apartados anteriores para b=16 bits y n=10.000 pares de números en la entrada.

Riesgos de la segmentación

- Hay situaciones, llamadas riesgos, que impiden que se ejecute la siguiente instrucción del flujo de instrucciones. Estos riesgos reducen la ganancia.
  - **Tres clases de riesgos:**
    - **Riesgos estructurales**
      - Surgen de conflictos de los recursos cuando el hardware no puede soportar todas las combinaciones de instrucciones en ejecución.
    - **Riesgos por dependencias de datos**
      - Surgen cuando una instrucción depende de los resultados de una instrucción anterior.
    - **Riesgos de control**
      - Surgen de la segmentación de los saltos y otras instrucciones que cambian el PC.

Riesgos de la segmentación

- Pueden hacer necesario detener la segmentación.
- Cuando una instrucción se detiene, las instrucciones posteriores también lo hacen y no se buscan instrucciones nuevas. Las anteriores pueden continuar
- Una detención disminuye la ganancia con respecto a la ideal

Riesgos de la segmentación

- **Rendimiento de la segmentación con detenciones**

$$G_s = \frac{tiempo\ medio\ instrucción\ sin\ segmentación}{tiempo\ medio\ instrucción\ con\ segmentación} =$$

$$= \frac{CPI\ sin\ segmentación \cdot Ciclo\ de\ reloj\ sin\ segmentación}{CPI\ con\ segmentación \cdot Ciclo\ de\ reloj\ con\ segmentación} =$$

$$= \frac{Ciclo\ de\ reloj\ \text{sin}\ segmentación}{Ciclo\ de\ reloj\ con\ segmentación} \times \frac{CPI\ \text{sin}\ segmentación}{CPI\ con\ segmentación}$$

- La segmentación puede entenderse como una mejora del CPI o como una reducción del ciclo de reloj.

Riesgos de la segmentación

Rendimiento de la segmentación con detenciones

Segmentación como una mejora del CPI

Suponed que el CPI ideal de un cauce es 1, el CPI de una cauce se calcular como:

$$CPI \ \ con \ \ segmentación = CPI_{ideal} + Ciclos\ reloj\ \det ención\ de\ segmentación\ \ por \ \ instrución$$

$$= 1 + Ciclos\ de\ reloj\ de\ \det ención\ de\ la\ segmentación\ \ por\ instrución$$

Si ignoramos el incremento potencial en el ciclo de reloj debido a la segmentación, y asumimos que las etapas están equilibradas, podemos igualar el ciclo de reloj de las dos máquinas:

$$G_s = \frac{Ciclo\ de\ reloj\ \text{sin}\ segmentación}{Ciclo\ de\ reloj\ con\ segmentación} \times \frac{CPI\ \text{sin}\ segmentación}{CPI\ con\ segmentación}$$

$$G_s = \frac{CPI\ \text{sin}\ segmentación}{1 + ciclos\ de\ reloj\ de\ \det ención\ de\ la\ segmentación\ por\ instrucción}$$

Riesgos de la segmentación

Rendimiento de la segmentación con detenciones

- Un caso simple e importante: todas las instrucciones tardan el mismo número de ciclos que además es igual al número de etapas del cauce (profundidad de la segmentación). CPI sin segmentación es igual a la profundidad del cauce y por tanto:

$$G_s = \frac{\text{Profundidad de la segmentación}}{1 + \text{ciclos de reloj de detención de la segmentación por instrucción}}$$

- Si no hubiese detenciones esto llevaría al resultado intuitivo de que la segmentación puede mejorar el rendimiento en la magnitud de la profundidad de la segmentación.

Riesgos de la segmentación

Rendimiento de la segmentación con detenciones

- **Segmentación como una reducción del ciclo de reloj.**
- Alternativamente si entendemos la segmentación como una mejora del ciclo de reloj, entonces podemos suponer que el CPI de la máquina no segmentada así como el de la segmentada vale 1. Esto nos lleva a:

$$G_s = \frac{CPI \text{ sin } segmentación}{CPI \text{ con } segmentación} \times \frac{Ciclo\ de\ reloj \text{ sin } segmentación}{Ciclo\ de\ reloj \text{ con } segmentación}$$

$$= \frac{1}{1 + ciclos\ de \text{det} ención\ \ por\ instrucción} \times \frac{Ciclo\ de\ reloj \text{ sin } segmentación}{Ciclo\ de\ reloj \text{ con } segmentación}$$

Riesgos de la segmentación

Rendimiento de la segmentación con detenciones

- En el caso de que las etapas del cauce estén equilibradas y no exista sobrecarga, el ciclo de reloj en la máquina segmentada es menor que el ciclo de reloj de la máquina no segmentada en un factor igual a la profundidad del cauce.

$$Ciclo\ de\ reloj\ con\ segmentación = \frac{Ciclo\ de\ reloj\ \text{sin}\ segmentación}{profundidad\ de\ la\ segmentación}$$

$$profundidad\ de\ la\ segmentación = \frac{Ciclo\ de\ reloj\ \text{sin}\ segmentación}{Ciclo\ de\ reloj\ con\ segmentación}$$

Riesgos de la segmentación

Rendimiento de la segmentación con detenciones

- Esto nos lleva a

$$G_s = \frac{1}{1 + ciclos\ de\ \text{det}ención\ \ por\ instrucción} \times \frac{Ciclo\ de\ reloj\ \text{sin}\ segmentación}{Ciclo\ de\ reloj\ con\ segmentación}$$

$$= \frac{1}{1 + ciclos\ de\ \text{det}ención\ \ por\ instrucción} \times profundidad\ de\ la\ segmentación$$

- De nuevo nos encontramos con el resultado intuitivo de que la segmentación puede mejorar el rendimiento en la magnitud de la profundidad de la segmentación, si no hubiese detenciones

Riesgos estructurales

- Si alguna combinación de instrucciones no puede darse en el cauce por conflicto de recursos, se dice que la máquina tiene un riesgo estructural.

- Cuando una máquina se segmenta, la ejecución solapada de instrucciones necesita la segmentación de unidades funcionales y la duplicación de recursos para permitir todas las posibles combinaciones de instrucciones en el cauce.

Riesgos estructurales

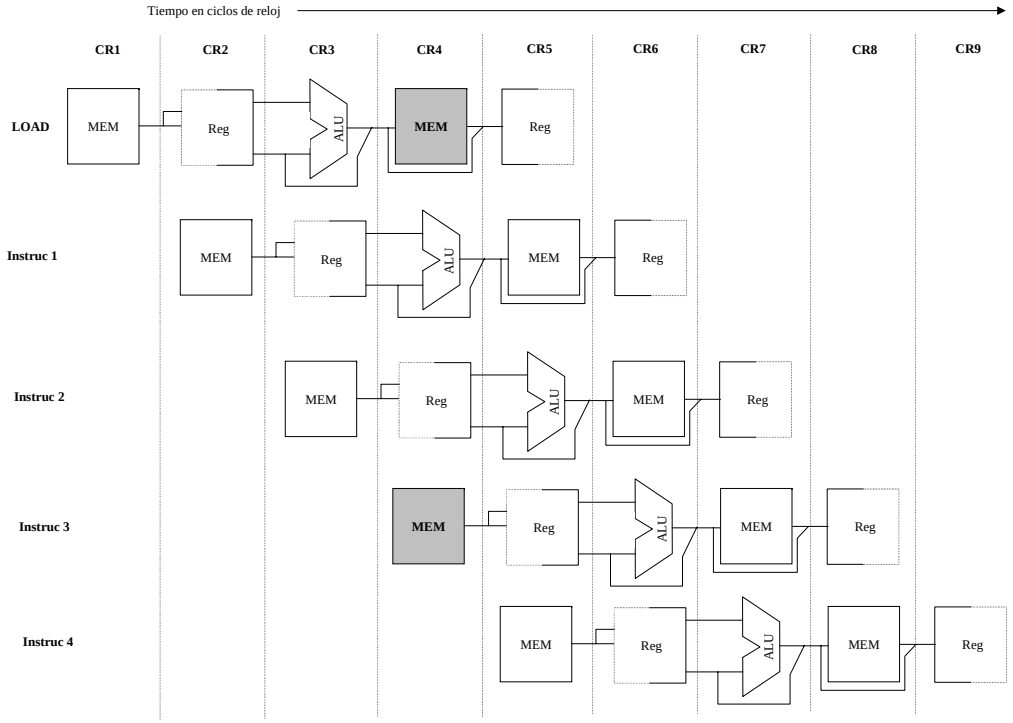
- **Dos formas de encontrar riesgos estructurales**
  - **Unidades funcionales que no están completamente segmentada**
    - Los casos más comunes de riesgos estructurales surgen cuando alguna unidad funcional no esta completamente segmentada. En ese caso una secuencia de instrucciones usando esta unidad no segmentada no puede proceder a razón de una instrucción por ciclo.
    - **Ejemplo:** Unidad aritmética en punto flotante que no permite segmentación.

Riesgos estructurales

- **Dos formas de encontrar riesgos estructurales**
  - **Recurso que no se ha duplicado lo suficiente**
    - Otra forma habitual de encontrar riesgos estructurales es cuando un recurso no se ha duplicado lo suficiente para soportar todas las combinaciones.
    - **Ejemplo**: una máquina puede tener un solo puerto de escritura en el banco de registros. La segmentación puede necesitar realizar dos escrituras en un único ciclo
  - Cuando una secuencia de instrucciones encuentre este riesgo, el cauce detendrá una de las instrucciones hasta que la unidad requerida este libre. Estas detenciones incrementarán el CPI ideal de valor 1.

Riesgos estructurales

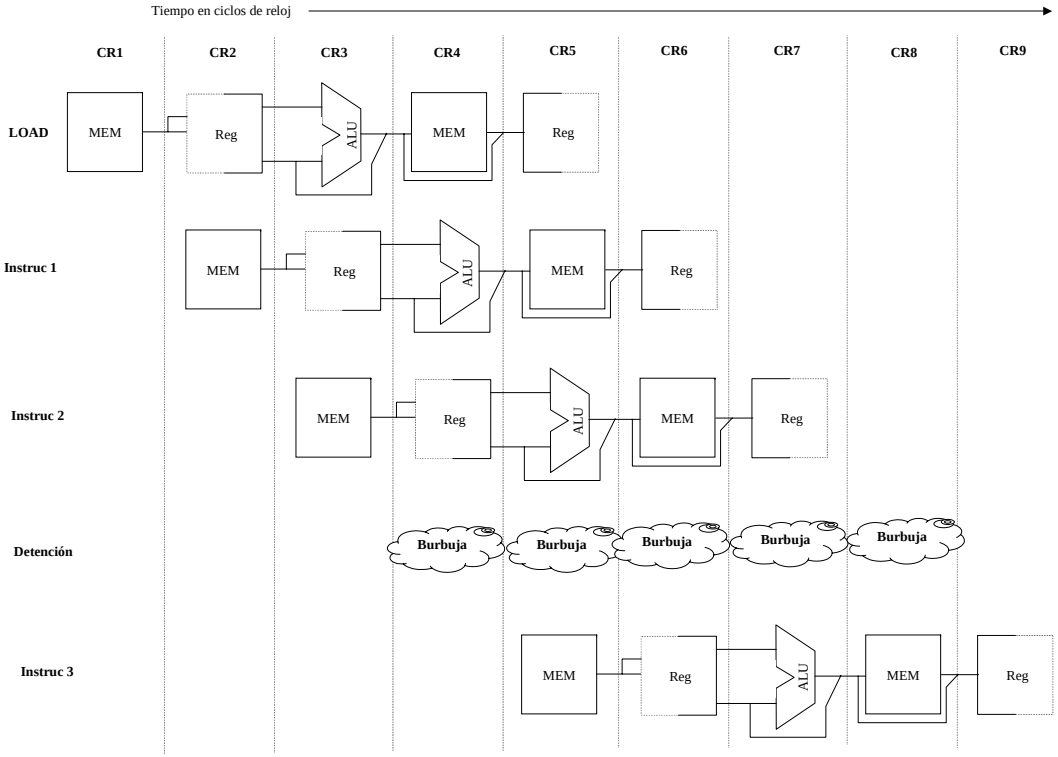
- **Ejemplo**: Algunas máquinas segmentadas comparten una única memoria para datos e instrucciones.
  - Cuando una instrucción contiene una referencia a un dato de memoria, entrará en conflicto con la referencia a instrucción de una instrucción posterior.



Riesgos estructurales

- Detenemos la segmentación durante un ciclo de reloj cuando ocurre el acceso a los datos de memoria.
  - A las detenciones se les llama burbujas, ya que flotan en el cauce ocupando espacio pero sin realizar trabajo.

- Ninguna instrucción terminará durante el ciclo de reloj 8,
- Asumimos que la instrucción 1 no es load ni store, en ese caso la instrucción 3 no puede comenzar su ejecución.



Riesgos estructurales

- En lugar de dibujar el camino de datos segmentado cada vez, los diseñadores a menudo indican el comportamiento de las detenciones usando diagramas simples con los nombres de las etapas.
- La figura muestra la detención indicando el ciclo donde no ocurren acciones y desplazando la instrucción 3 a la derecha

Nº ciclo de reloj										
Instrucción	1	2	3	4	5	6	7	8	9	10
Load	IF	ID	EX	MEM	WB					
Instrucción i+1		IF	ID	EX	MEM	WB				
Instrucción i+2			IF	ID	EX	MEM	WB			
Instrucción i+3				Deten	IF	ID	EX	MEM	WB	
Instrucción i+4						IF	ID	EX	MEM	WB
Instrucción i+5							IF	ID	EX	MEM
Instrucción i+6								IF	ID	EX

Riesgos estructurales

- El efecto de la burbuja es ocupar los recursos para el hueco de una instrucción como si esta cruzase todo el cauce.
- El impacto sobre el rendimiento es que la inst. 3 no se completa hasta el ciclo 9 y ninguna inst. se completa durante el 8.

Nº ciclo de reloj										
Instrucción	1	2	3	4	5	6	7	8	9	10
Load	IF	ID	EX	MEM	WB					
Instrucción i+1		IF	ID	EX	MEM	WB				
Instrucción i+2			IF	ID	EX	MEM	WB			
Instrucción i+3				Deten	IF	ID	EX	MEM	WB	
Instrucción i+4						IF	ID	EX	MEM	WB
Instrucción i+5							IF	ID	EX	MEM
Instrucción i+6								IF	ID	EX

Riesgos estructurales

- **Ejemplo**. Veamos cuanto puede costar el riesgo estructural load. Supongamos que las referencias a datos constituyen un 40% de la mezcla, y que el CPI ideal de la máquina segmentada, ignorando el riesgo estructural es 1. Suponemos que la máquina con el riesgo estructural tiene una frecuencia de reloj que es 1.05 veces mayor que la frecuencia de reloj que la máquina sin el riesgo. Descartando otras perdidas de rendimiento. ¿La segmentación con el riesgo estructural es más o menos rápida que sin él, y en que magnitud?.

Calculamos el tiempo medio por instrucción:

Tiempo medio por instrucción = CPI x Ciclo de reloj

El tiempo medio por instrucción para la máquina ideal es (al no tener detenciones):

Tiempo medio por instrucción ideal = Ciclo de reloj ideal

Riesgos estructurales

- **Ejemplo**. Veamos cuanto puede costar el riesgo estructural load. Supongamos que las referencias a datos constituyen un 40% de la mezcla, y que el CPI ideal de la máquina segmentada, ignorando el riesgo estructural es 1. Suponemos que la máquina con el riesgo estructural tiene una frecuencia de reloj que es 1.05 veces mayor que la frecuencia de reloj que la máquina sin el riesgo. Descartando otras perdidas de rendimiento. ¿ La segmentación con el riesgo estructural es más o menos rápida que sin él, y en que magnitud?.

Tiempo medio por instrucción para la máquina con riesgo estructural es:

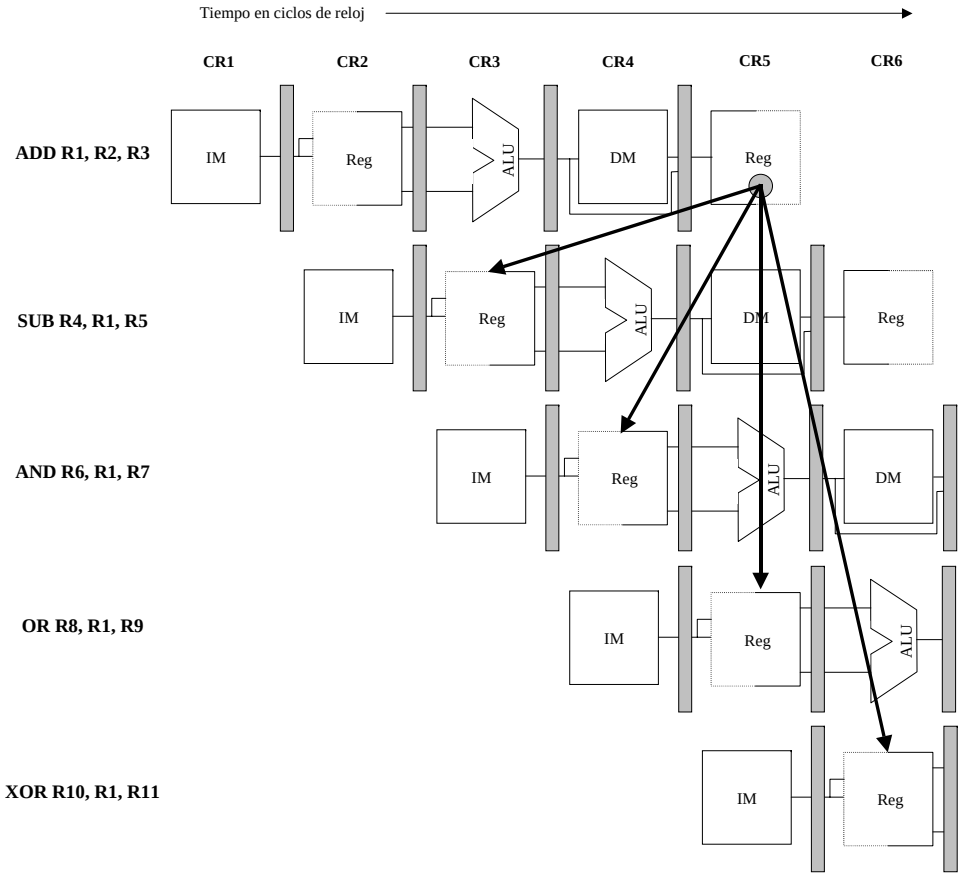
Tiempo medio por instrucción = CPI x Ciclo de reloj

$$(1+0.4\cdot 1)\cdot\frac{Ciclo\ \ de\ \ reloj_{ideal}}{1.05}=1.3\cdot Ciclo\ \ de\ \ reloj_{ideal}$$

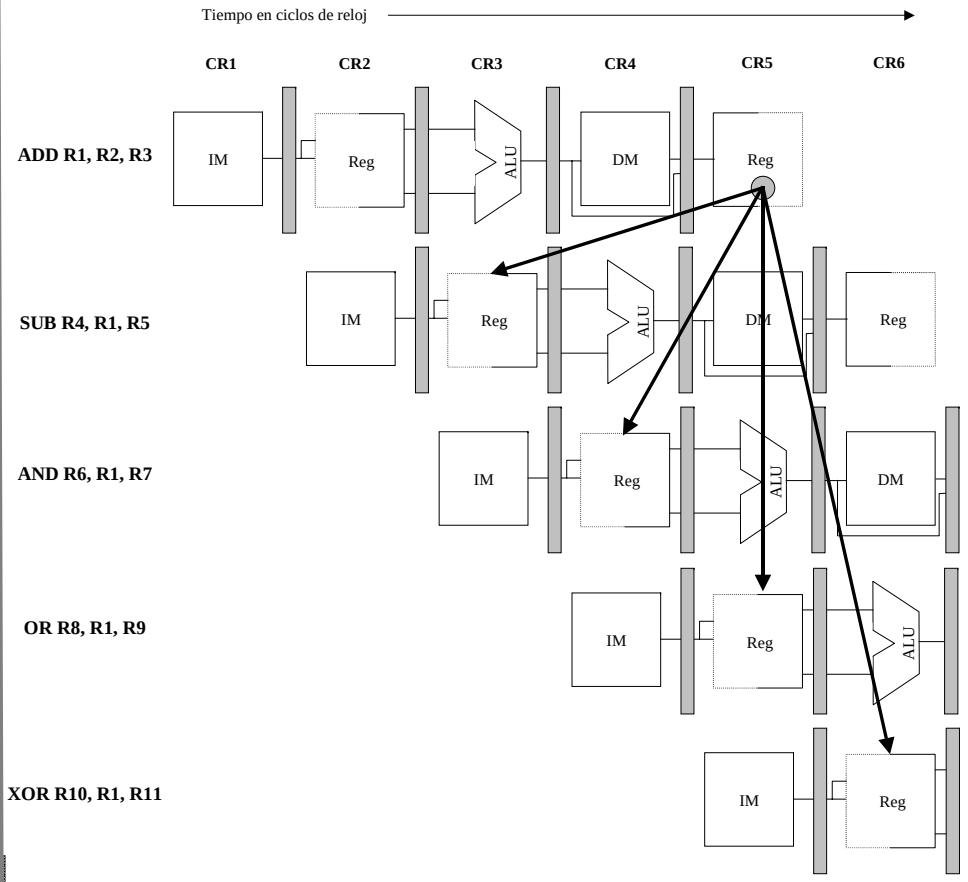
La máquina sin el riesgo estructural es 1.3 veces más rápida

Riesgos por dependencia de datos

- Los riesgos de datos ocurren cuando la segmentación cambia el orden de los accesos lectura/escritura a los operandos

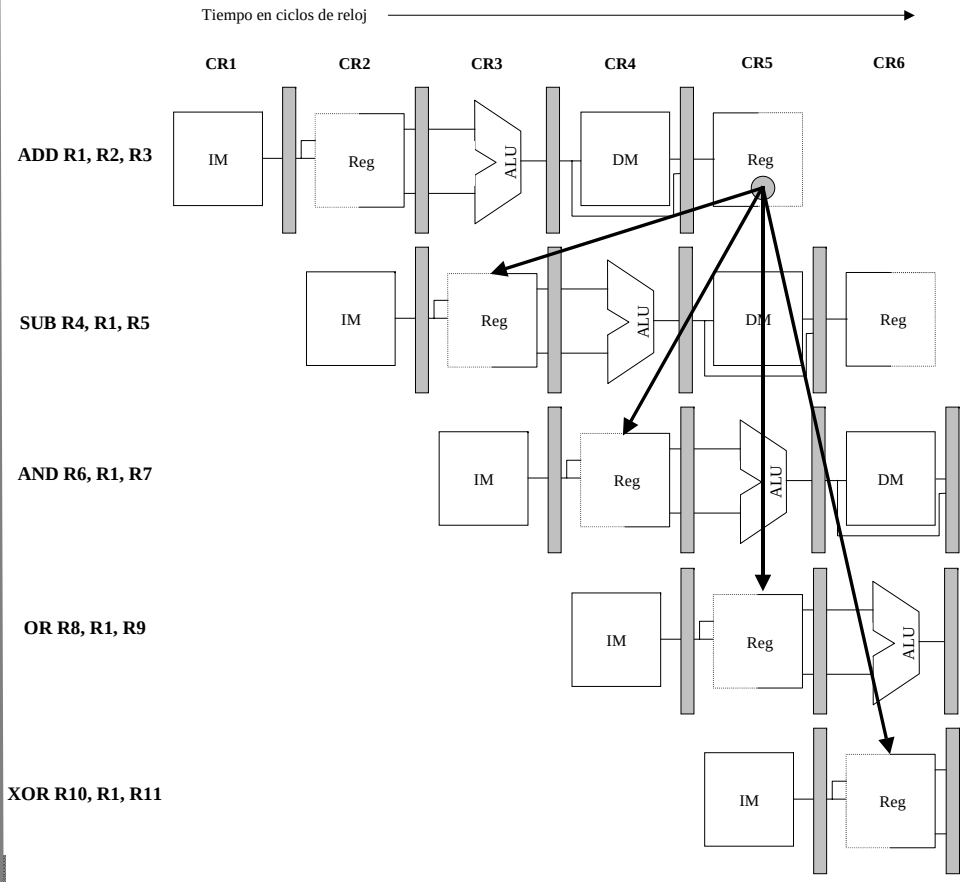


Riesgos por dependencia de datos



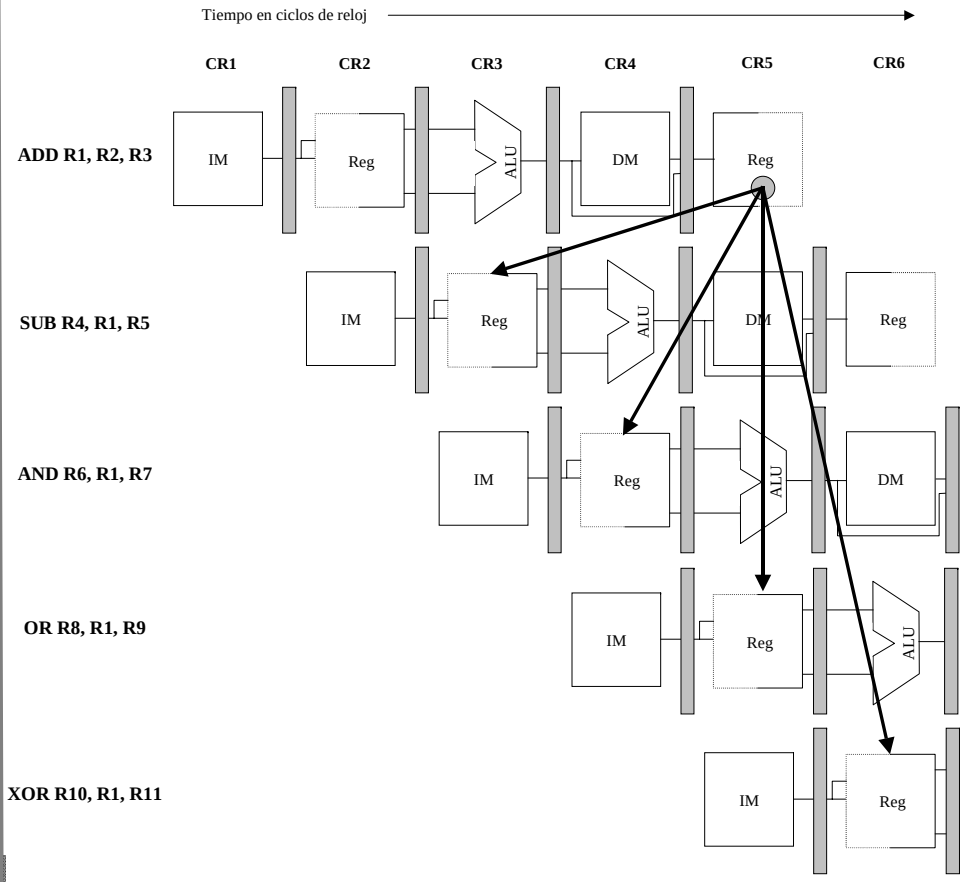
- Primera instrucción (SUB)
- La instrucción ADD escribe el valor de R1 en la etapa WB, pero la instrucción SUB lee el valor durante su etapa ID. La instrucción SUB leerá el valor erróneo e intentará utilizarlo.
- El valor usado por la instrucción SUB ni siquiera es determinístico

Riesgos por dependencia de datos



- Riesgo de datos de la segunda instrucción (AND)
- La instrucción AND. la escritura de R1 no se completa hasta el final del ciclo 5. Por lo tanto la instrucción AND que lee el registro durante el ciclo 4 recibirá resultados erróneos.

Riesgos por dependencia de datos



- Riesgo de datos de la tercera instrucción (OR)
- La instrucción OR puede hacerse funcionar sin que incurra en un riesgo con una técnica de implementación simple. Las lecturas del banco de registros en la segunda mitad del ciclo y las escrituras en la primera mitad.
- La instrucción XOR funciona adecuadamente.

Riesgos por dependencia de datos

La técnica del adelantamiento

- Forwarding, bypassing, short-circuiting
- **La clave**: el resultado no se necesita realmente en la instrucción SUB hasta después de que la ADD realmente los produce
- Si el resultado puede moverse desde donde lo produce la ADD (el registro EX/MEM) hasta donde lo necesita la SUB (los multiplexores de entrada de la ALU) entonces la necesidad de detención puede evitarse

Riesgos por dependencia de datos

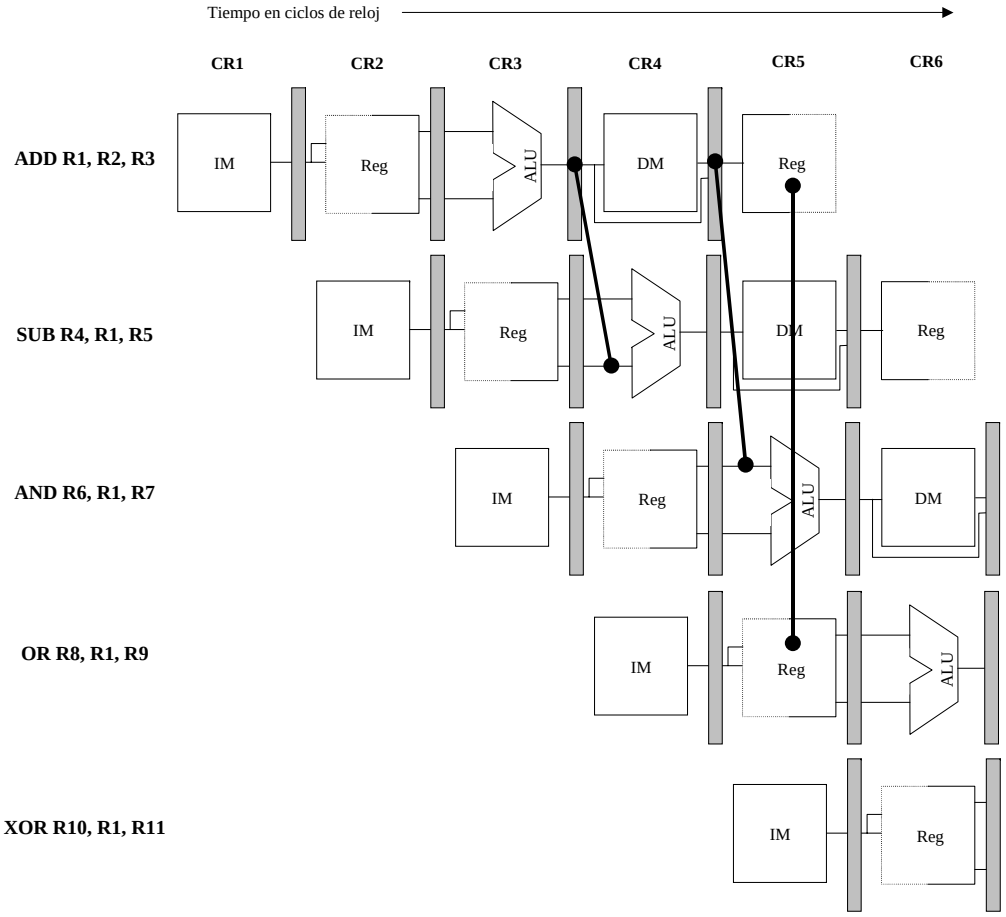
La técnica del adelantamiento

- **El adelantamiento trabaja** de la siguiente forma:
- **1.** El resultado de la ALU del registro EX/MEM siempre se realimenta a los multiplexores de entrada de la ALU.
- **2.** Si el hardware de adelantamiento detecta que la instrucción ALU previa ha escrito el registro correspondiente a una fuente de la operación ALU en curso, la lógica de control selecciona el valor adelantado como valor de entrada a la ALU en lugar del valor leído en el banco de registros
- **Completamente determinístico**
- Si la SUB se detiene, la ADD se completará y el adelantamiento no será activado

Riesgos por dependencia de datos

La técnica del adelantamiento

- La figura siguiente muestra el ejemplo con las rutas de adelantamiento

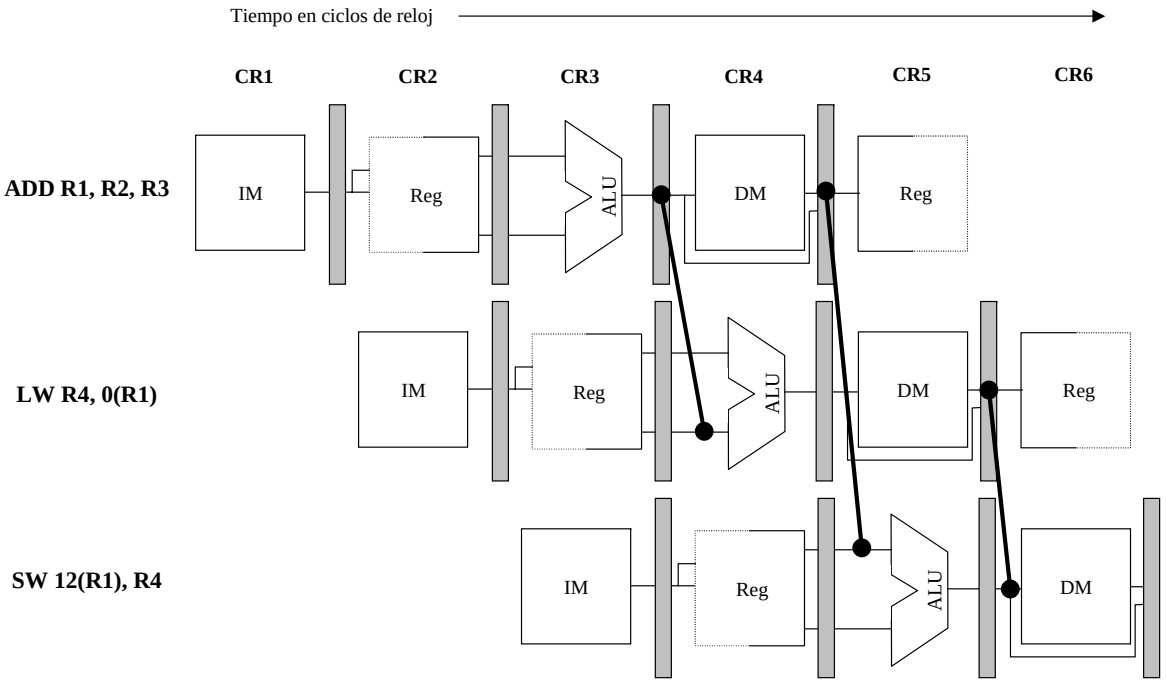


- Las entradas para las instrucciones SUB y AND se adelantan desde los registros intermedios EX/MEM y MEM/WB
- La OR recibe el resultado mediante adelantamiento a través del banco de registros

Riesgos por dependencia de datos

Generalización del adelantamiento

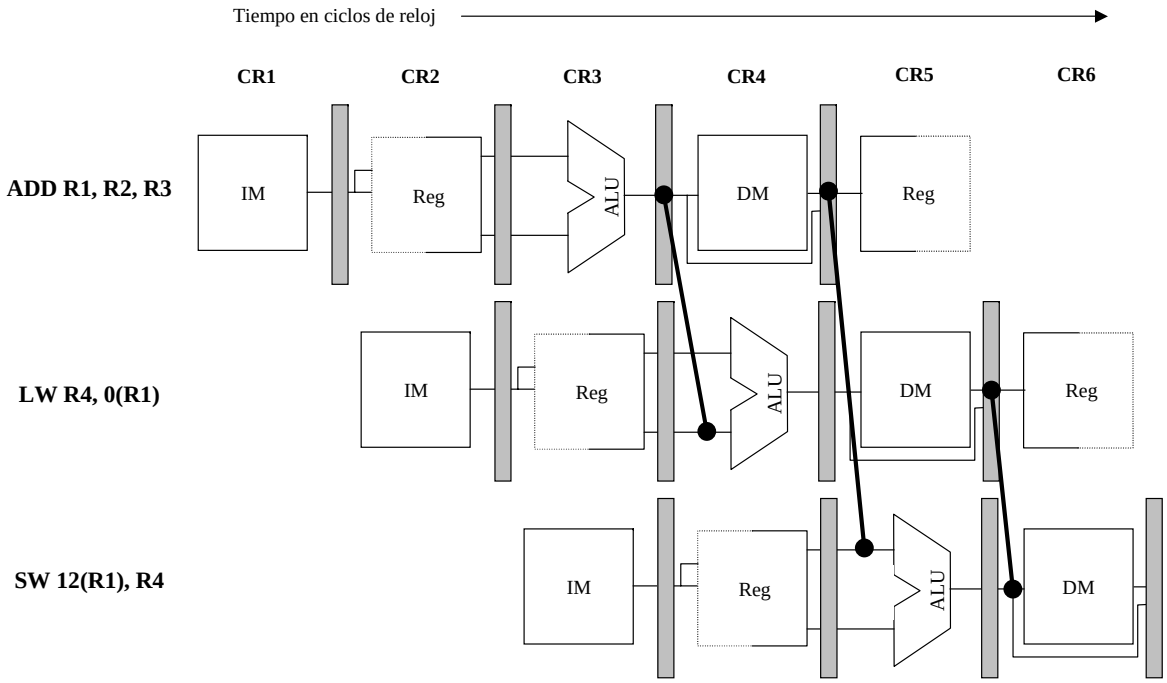
- El adelantamiento puede generalizarse para incluir el paso de resultados desde la salida de una unidad a la entrada de otra



Riesgos por dependencia de datos

Generalización del adelantamiento

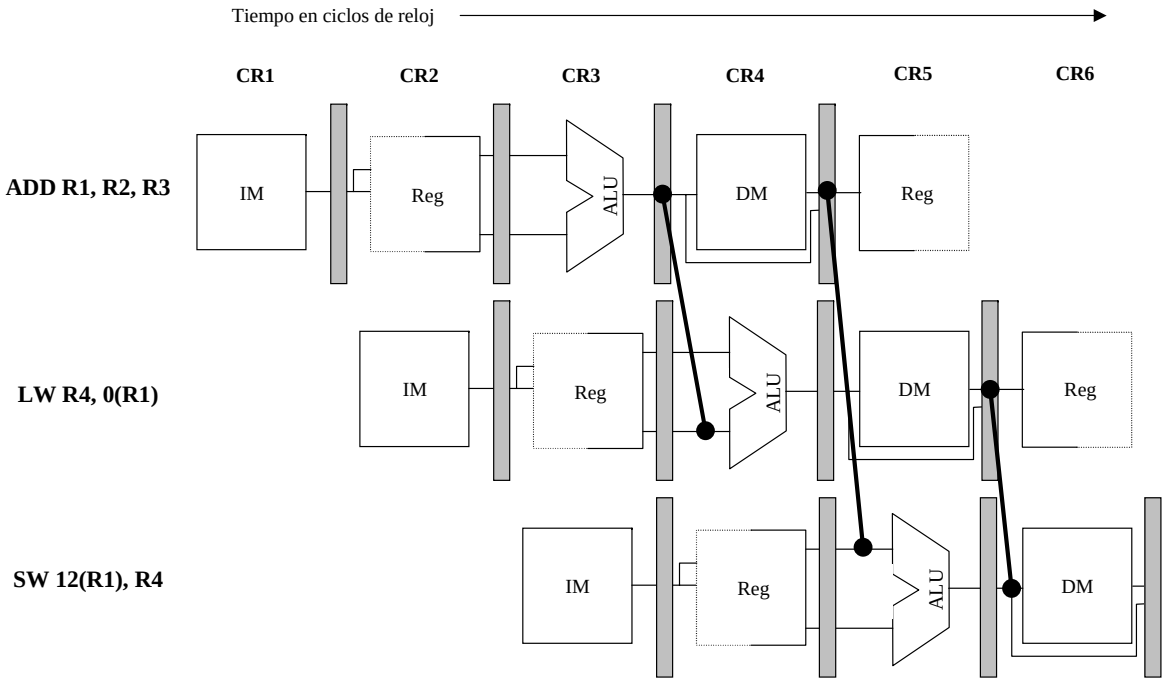
- STORE requiere un operando durante MEM. El resultado de la instrucción LOAD se adelanta desde la salida de la memoria en MEM/WB a la entrada de la memoria para ser almacenado.



Riesgos por dependencia de datos

Generalización del adelantamiento

- La salida de la ALU se adelanta a la entrada de la ALU para el cálculo de la dirección tanto en la LOAD como en la STORE



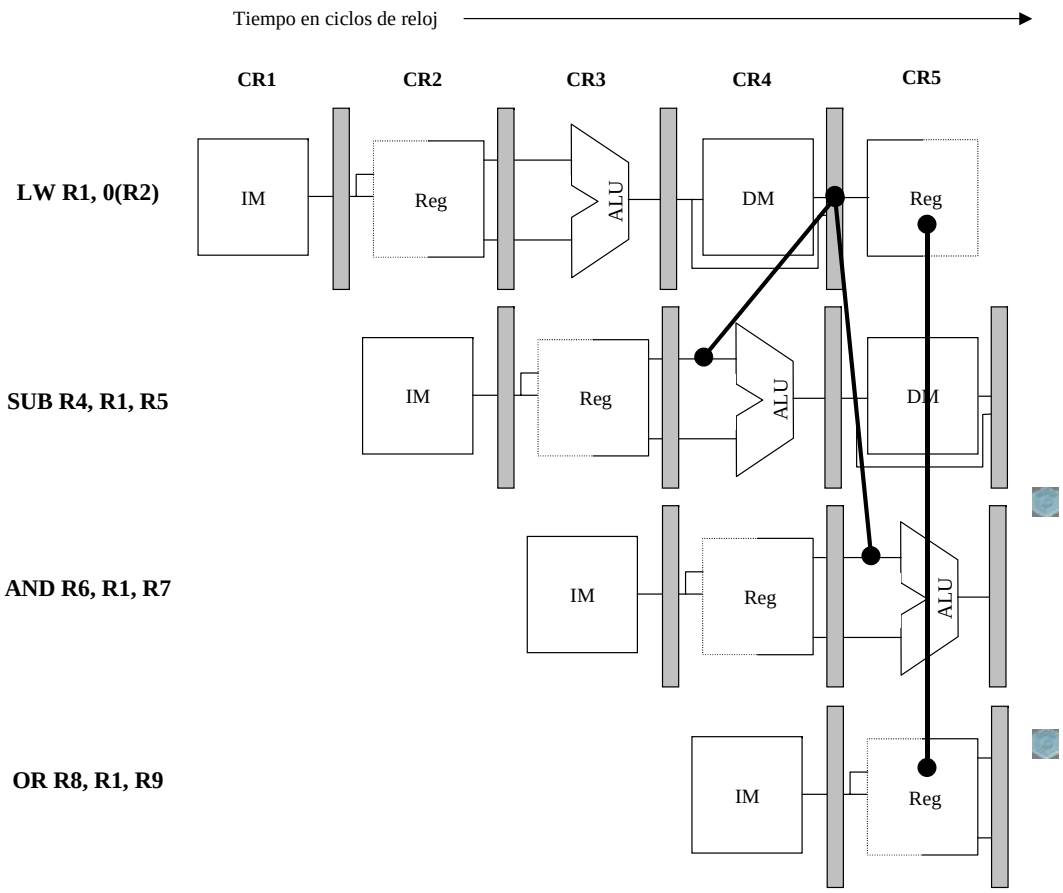
Riesgos por dependencia de datos

Generalización del adelantamiento

- En MIPS, podemos requerir una ruta de adelantamiento desde cualquier registro intermedio hacia la entrada de cualquier unidad funcional.
- Como tanto la ALU como la memoria de datos aceptan operandos, se necesitan rutas de adelantamiento hacia sus entradas desde los registros intermedios ALU/MEM y MEM/WB.
- MIPS usa una unidad de detección de cero que opera durante el ciclo EX, y el adelantamiento a esa unidad puede ser también necesario.

Riesgos por dependencia de datos

Riesgos de datos que requieren detenciones



- LW no tiene el dato hasta el final del ciclo 4 (su ciclo MEM), mientras que la instrucción SUB necesita el dato en el inicio de ese ciclo
- Podemos adelantar a la ALU desde los registros MEM/WB para AND
- OR no tiene problema, ya que recibe el valor a través del banco de registros

Riesgos por dependencia de datos

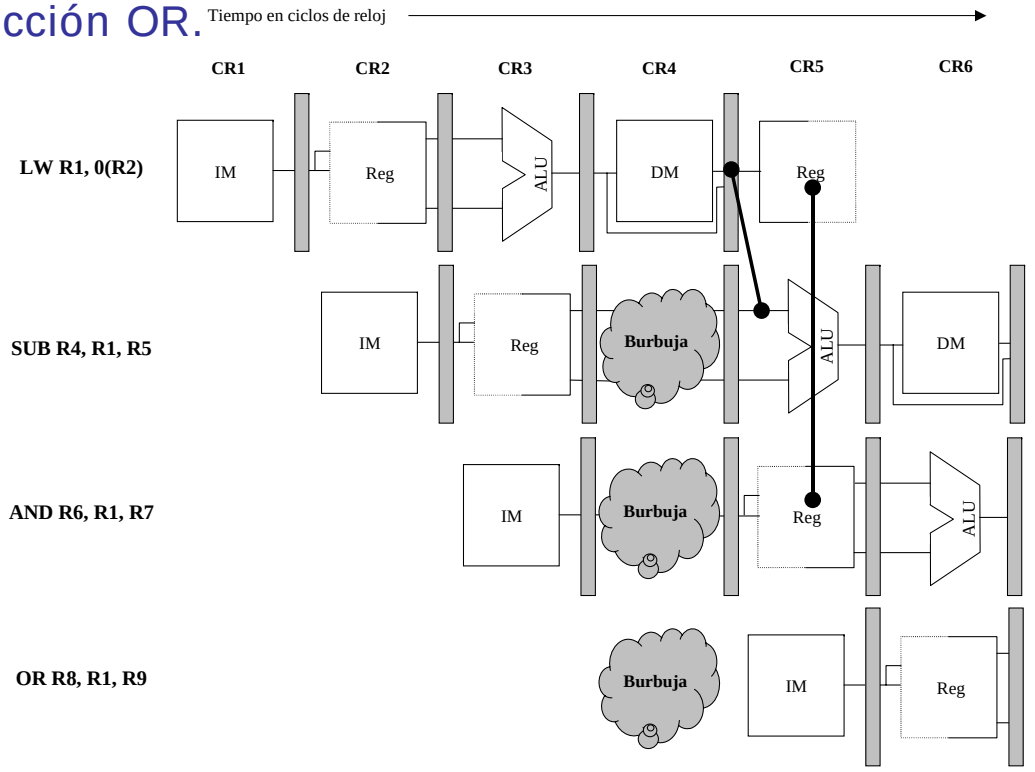
Hardware de interbloqueo

- Necesitamos **añadir hardware de interbloqueo** del cauce, para preservar el patrón de ejecución correcto.
- En general, el hardware de interbloqueo, detecta un riesgo y detiene el cauce hasta que el riesgo desaparece.
- En este caso, el interbloqueo detiene el cauce, empezando por la instrucción que quiere usar el dato hasta que la instrucción fuente los produce.
- Este interbloqueo del cauce introduce una detención o burbuja, como se hacía para los riesgos estructurales.

Riesgos por dependencia de datos

Hardware de interbloqueo

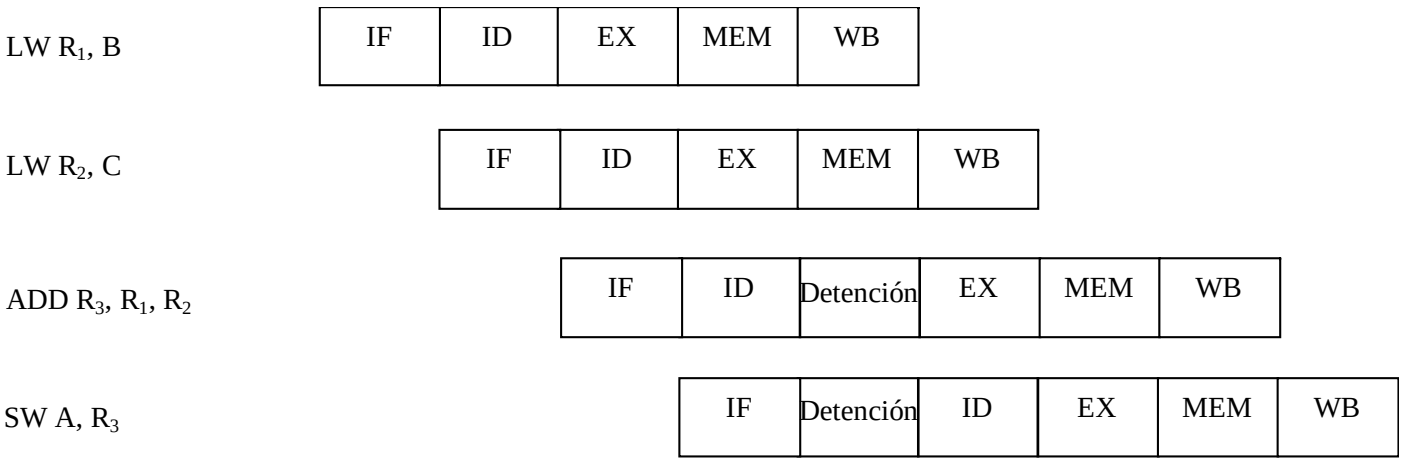
- Observamos el cauce con la detención y el adelantamiento legal.
- El adelantamiento a la instrucción AND va ahora a través del banco de registros y no se necesita adelantamiento para la instrucción OR.



Riesgos por dependencia de datos

Planificación del compilador para los riesgos de datos

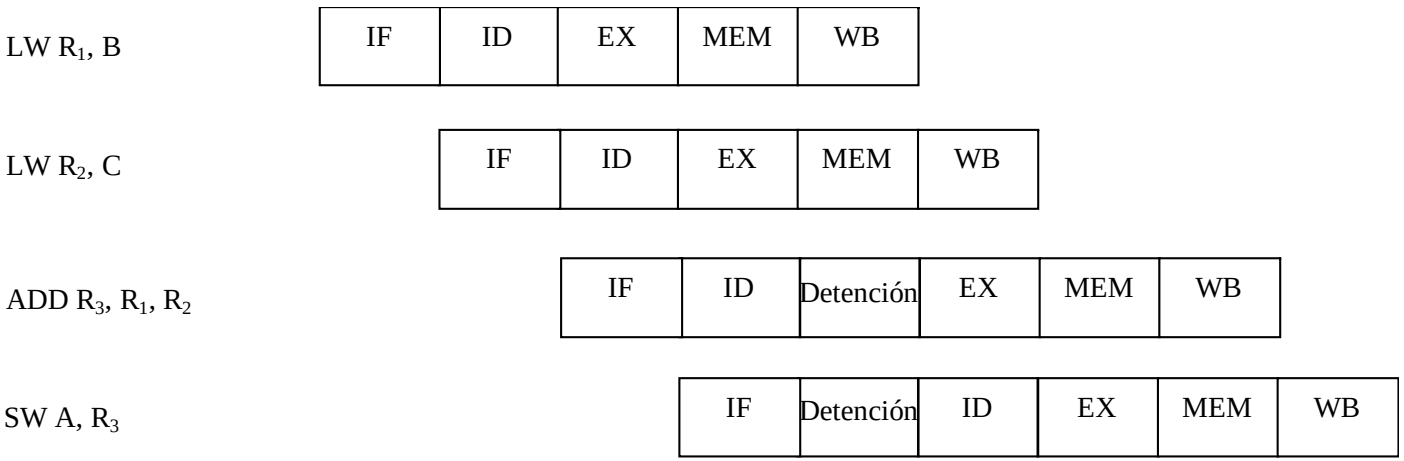
- Muchos tipos de detenciones son muy frecuentes.
- El patrón general de generación de código para una sentencia del tipo A=B+C produce una detención para la carga del valor del segundo dato C.
- El almacenamiento de A no provoca otra detención, ya que el resultado de la suma puede adelantarse a la memoria de datos para que sea usado por la store.



Riesgos por dependencia de datos

Planificación del compilador para los riesgos de datos

- En lugar de permitir que el cauce se detenga, el compilador puede intentar planificar la segmentación para evitar estas detenciones mediante la reorganización del código para eliminar los riesgos.
- Por ejemplo, el compilador puede intentar evitar generar código con una instrucción load seguida por el uso inmediato del registro destino de la load. Esta técnica se llama planificación del cauce o planificación de las instrucciones.



Riesgos por dependencia de datos

Ejemplo

Generar código de DLX que evite detenciones del cauce para la siguiente secuencia:

a = b + c
d = e - f

```
LW Rb, b
LW Rc, c
LW Re, e              ;intercambio con la siguiente instrucción
ADD Ra, Rb, Rc
LW Rf, f
SW a, Ra              ;se intercambia almacenamiento/carga
SUB Rd, Re, Rf
SW d Rd
```

Evitamos los interbloqueos:

LW Rc,c → ADD Ra, Rb, Rc
LW Rf,f → SUB Rd, Re, Rf

Riesgos por dependencia de datos

- **Implementación de la detección de riesgos por dependencias de datos**

  - **Emisión:** proceso de evolucionar desde la fase ID a la EX
  - Para DLX todos los riesgos de datos pueden detectarse durante ID
  - En caso de riesgo se detiene la emisión de la instrucción
  - Podemos determinar que adelantamiento será necesario durante ID y fijar el control apropiado

Riesgos por dependencia de datos

- **Situaciones detectables por el hardware de detección de riesgos al comparar los destinos y fuentes de instrucciones.**
  - La tabla muestra que la única comparación requerida es entre el destino y las fuentes de las dos instrucciones siguientes

Situación	Secuencia de código ejemplo	Acción

No dependencia	LW **R_1**, 45(R_2)
	ADD R_5, R_6, R_7
	SUB R_8, R_6, R_7
	OR R_9, R_6, R_7
		No hay riesgo. No existe dependencia sobre R_1 en las tres instrucciones siguientes.

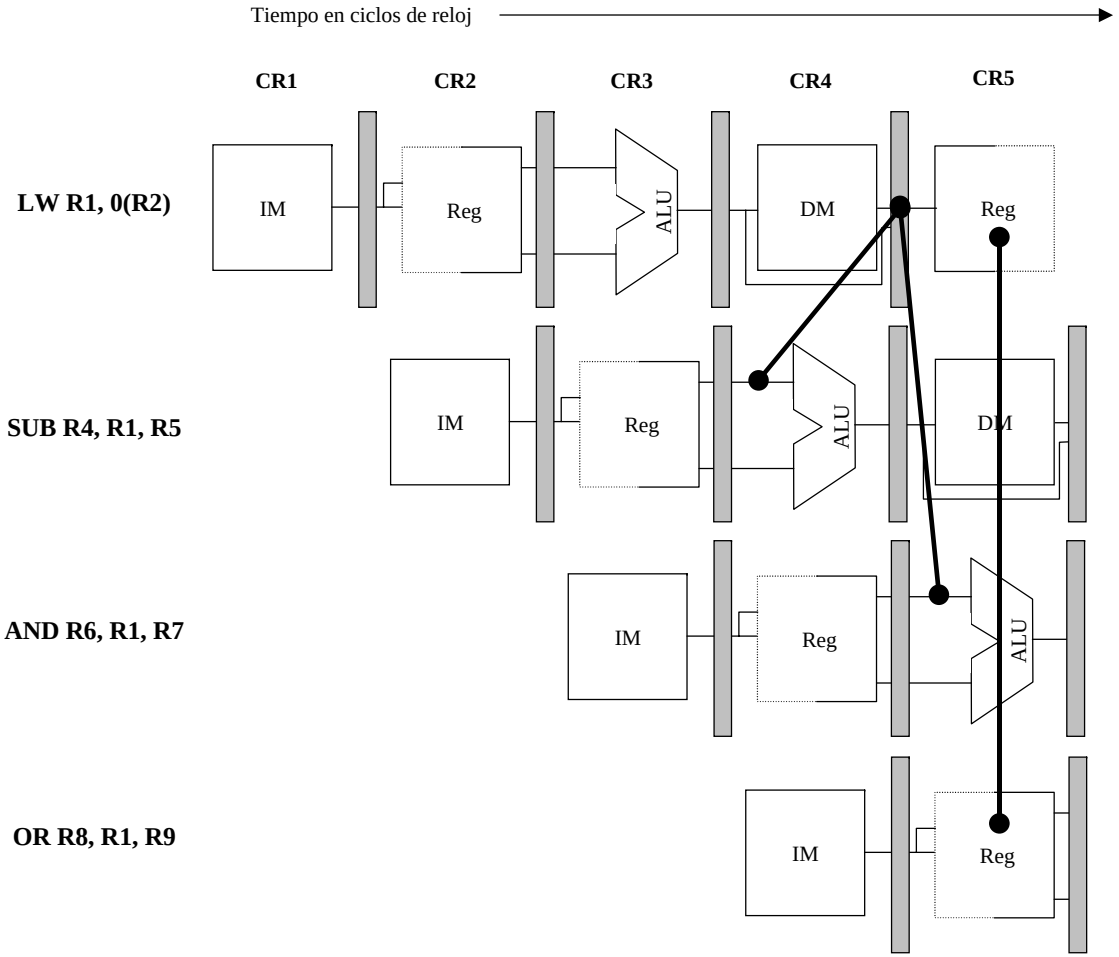
Dependencia que requiere detención	LW **R_1**, 45(R_2)
	ADD R_5, **R_1**, R_7
	SUB R_8, R_6, R_7
	OR R_9, R_6, R_7
		Los comparadores detectan el uso de R_1 en ADD y detienen ADD (y SUB y OR) antes que ADD comience EX

Dependencia superada por adelantamiento	LW **R_1**, 45(R_2)
	ADD R_5, R_6, R_7
	SUB R_8, **R_1**, R_7
	OR R_9, R_6, R_7
		Los comparadores detectan el uso de R_1 en SUB y adelantan el resultado de la carga a la ALU en el instante en que SUB comienza EX.

Dependencia con accesos en orden	LW **R_1**, 45(R_2)
	ADD R_5, R_6, R_7
	SUB R_8, R_6, R_7
	OR R_9, **R_1**, R_7
		No se requiere acción porque la lectura de R_1 por OR se presenta en la segunda mitad de la fase ID, mientras que la escritura del dato cargado se presentó en la primera mitad.

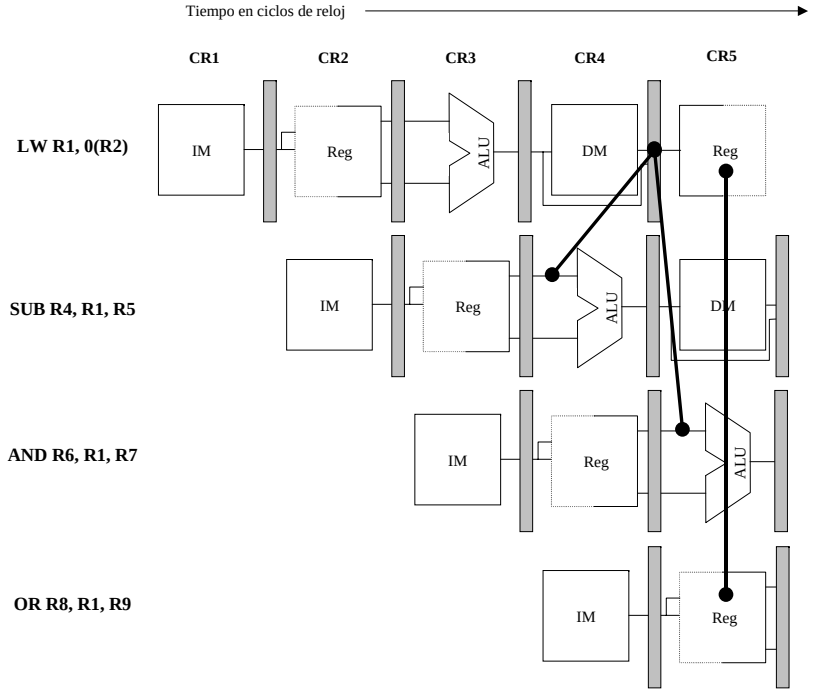
Riesgos por dependencia de datos

Implementación del interbloqueo load



Riesgos por dependencia de datos

Implementación del interbloqueo load



Campo código de operación de ID/EX ($ID/EX.IR_{0..5}$)	Campo código de operación de IF/ID ($IF/ID.IR_{0..5}$)	Comparación de campos de operandos
Load	ALU registro registro	$ID/EX.IR_{11..15}$==$IF/ID.IR_{6..10}$
Load	ALU registro registro	$ID/EX.IR_{11..15}$==$IF/ID.IR_{11..15}$
Load	Load, store, ALU inm, o salto	$ID/EX.IR_{11..15}$==$IF/ID.IR_{6..10}$

Instrucción tipo I

6	5	5	16
Cód ope	Rs1	Rd	Inmediato

Instrucción tipo R

6	5	5	5	11
Cód ope	Rs1	Rs2	Rd	func

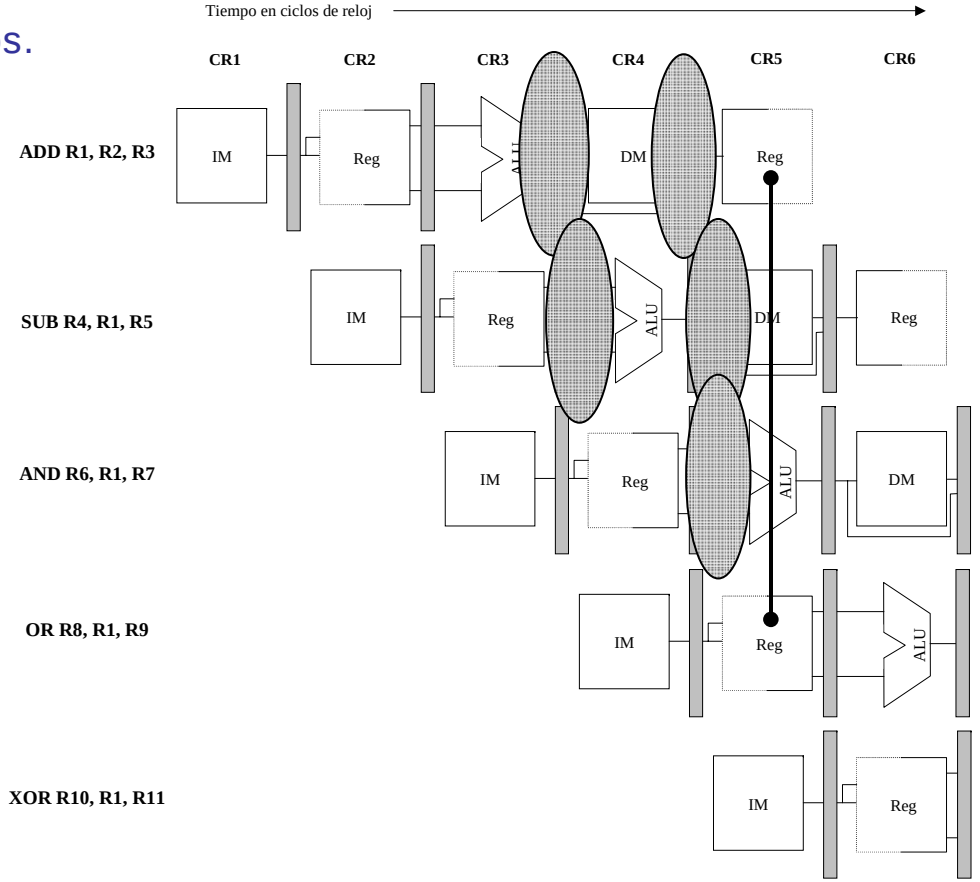
Instrucción tipo J

6	26
Cód ope	Desplazamiento añadido al PC

Riesgos por dependencia de datos

Implementación de la lógica de adelantamiento

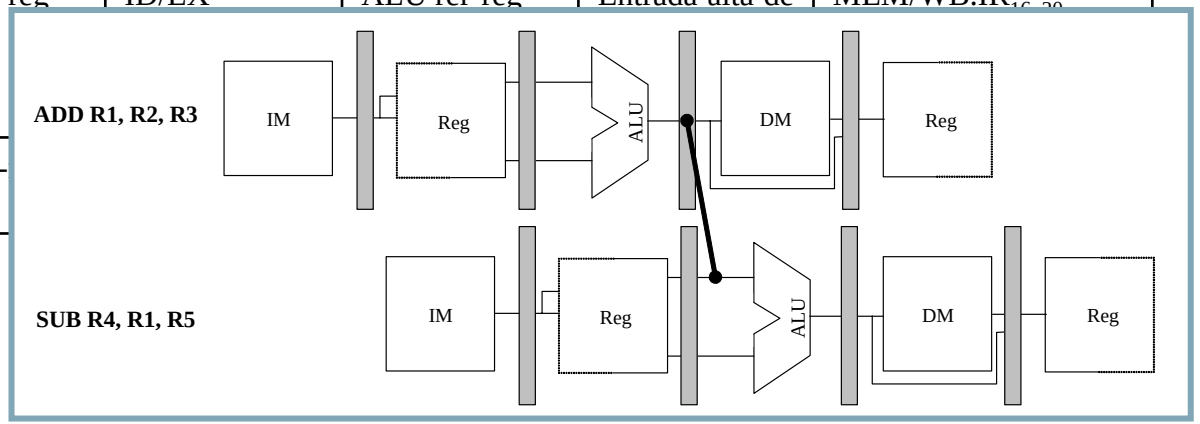
- Todo adelantamiento, lógicamente, ocurre:
  - Desde la salida de la ALU o la memoria de datos
  - Hacia la entrada de la ALU, la memoria de datos o la unidad de detección de ceros.



Riesgos por dependencia de datos

Implementación de la lógica de adelantamiento

Registro intermedio con la instrucción fuente	Código de operación de la instrucción fuente	Registro intermedio con la instrucción destino	Código de operación de la instrucción destino	Destino del resultado adelantado	Comparación (si igual entonces adelantar)
EX/MEM	ALU rer-reg	ID/EX	ALU rer-reg ALU inm Load, store Branch	Entrada alta de la ALU	$EX/MEM.IR_{16..20} = ID/EX.IR_{6..10}$
EX/MEM	ALU rer-reg	ID/EX	ALU rer-reg	Entrada baja de la ALU	$EX/MEM.IR_{16..20} = ID/EX.IR_{11..15}$
MEM/WB	ALU rer-reg	ID/EX	ALU rer-reg	Entrada alta de	$MEM/WB.IR_{16..20} =$
MEM/WB	ALU rer-				



Instrucción tipo I

6	5	5	16
Cód ope	Rs1	Rd	Inmediato

Instrucción tipo R

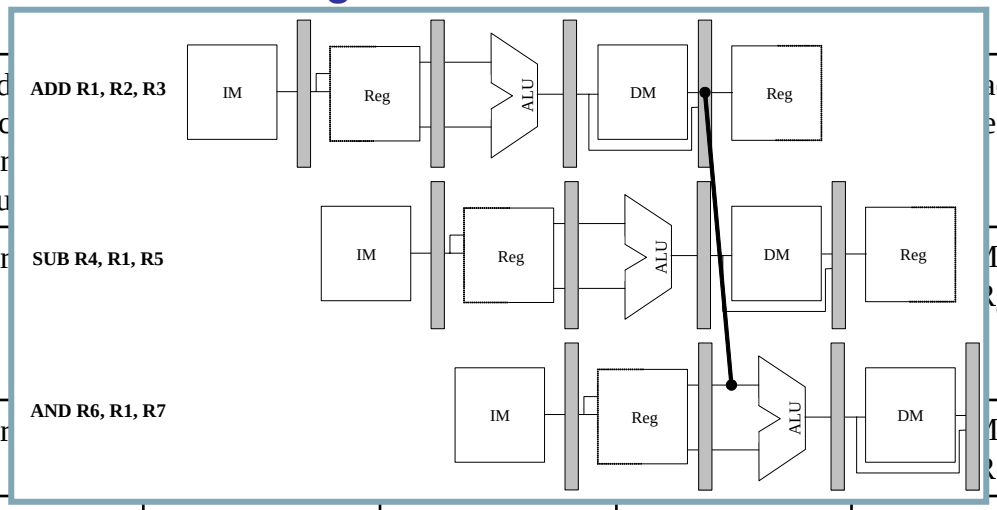
6	5	5	5	11
Cód ope	Rs1	Rs2	Rd	func

Instrucción tipo J

6	26
Cód ope	Desplazamiento añadido al PC

Riesgos por dependencia de datos

Implementación de la lógica de adelantamiento



Registro intermedio con la instrucción fuente	Cód operac instr fu				ación (si igual es adelantar)
EX/MEM	ALU r				M.IR$_{16..20}$= R$_{6..10}$
EX/MEM	ALU r				M.IR$_{16..20}$= R$_{11..15}$
MEM/WB	ALU rer-reg	ID/EX	ALU rer-reg ALU inm Load, store Branch	Entrada alta de la ALU	MEM/WB.IR$_{16..20}$= ID/EX.IR$_{6..10}$
MEM/WB	ALU rer-reg	ID/EX	ALU rer-reg	Entrada baja de la ALU	MEM/WB.IR$_{16..20}$= ID/EX.IR$_{11..15}$

Instrucción tipo I

6	5	5	16
Cód ope	Rs1	Rd	Inmediato

Instrucción tipo R

6	5	5	5	11
Cód ope	Rs1	Rs2	Rd	func

Instrucción tipo J

6	26
Cód ope	Desplazamiento añadido al PC

Riesgos por dependencia de datos

Implementación de la lógica de adelantamiento

Registro intermedio con la instrucción fuente	Código de operación de la instrucción fuente	Registro intermedio con la instrucción destino	Código de operación de la instrucción destino	Destino del resultado adelantado	Comparación (si igual entonces adelantar)
EX/MEM	ALU inm	ID/EX	ALU rer-reg ALU inm Load, store Branch	Entrada alta de la ALU	$EX/MEM.IR_{11..15} = ID/EX.IR_{6..10}$
EX/MEM	ALU inm	ID/EX	ALU rer-reg	Entrada baja de la ALU	$EX/MEM.IR_{11..15} = ID/EX.IR_{11..15}$
MEM/WB	ALU inm	ID/EX	ALU rer-reg ALU inm Load, store Branch	Entrada alta de la ALU	$MEM/WB.IR_{11..15} = ID/EX.IR_{6..10}$
MEM/WB	ALU inm	ID/EX	ALU rer-reg	Entrada baja de la ALU	$MEM/WB.IR_{11..15} = ID/EX.IR_{11..15}$

Instrucción tipo I

6	5	5	16
Cód ope	Rs1	Rd	Inmediato

Instrucción tipo R

6	5	5	5	11
Cód ope	Rs1	Rs2	Rd	func

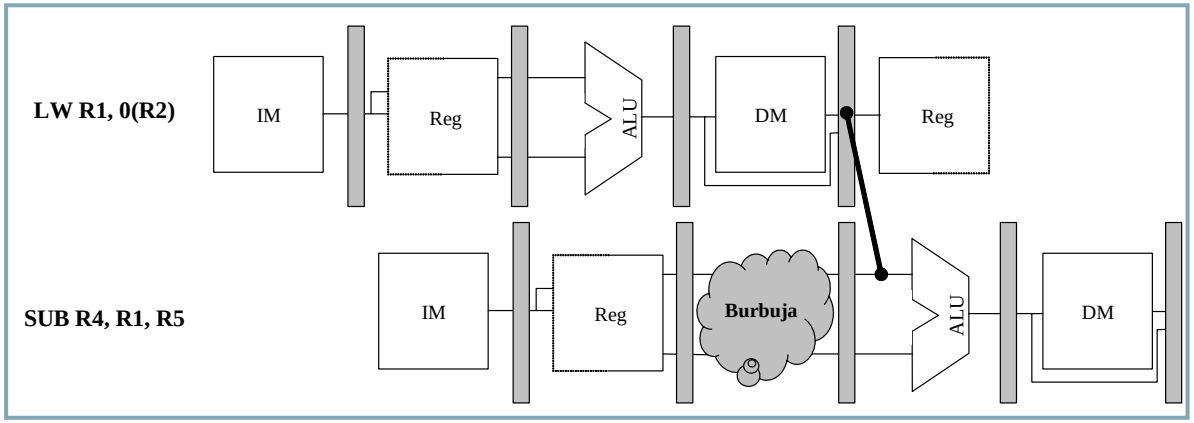
Instrucción tipo J

6	26
Cód ope	Desplazamiento añadido al PC

Riesgos por dependencia de datos

Implementación de la lógica de adelantamiento

Registro intermedio con la instrucción fuente	Código de operación de la instrucción fuente	Registro intermedio con la instrucción destino	Código de operación de la instrucción destino	Destino del resultado adelantado	Comparación (si igual entonces adelantar)
MEM/WB	Load	ID/EX	ALU rer-reg ALU inm Load, store Branch	Entrada alta de la ALU	$MEM/WB.IR_{11..15} = ID/EX.IR_{6..10}$
MEM/WB	Load	ID/EX	ALU rer-reg	Entrada baja de la ALU	$MEM/WB.IR_{11..15} = ID/EX.IR_{11..15}$



Instrucción tipo I

6	5	5	16
Cód ope	Rs1	Rd	Inmediato

Instrucción tipo R

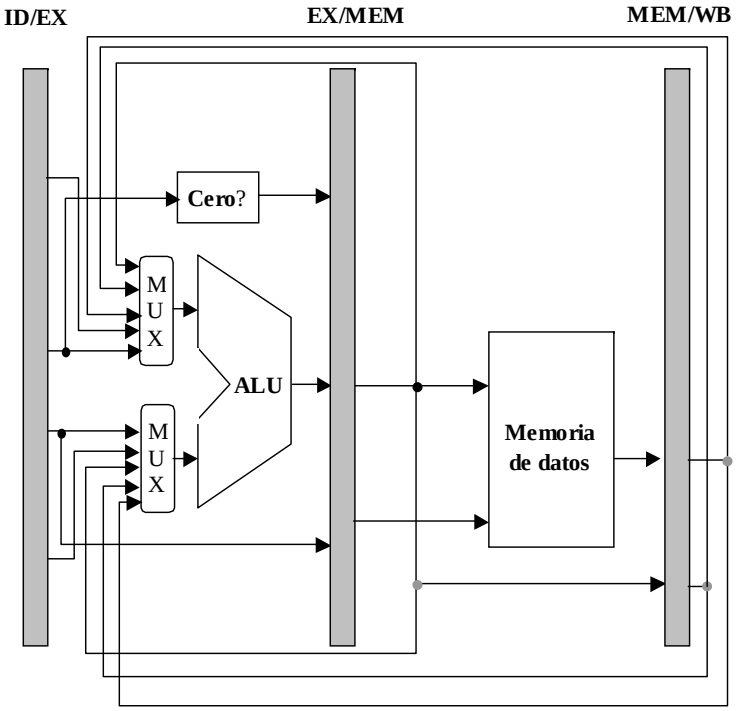
6	5	5	5	11
Cód ope	Rs1	Rs2	Rd	func

Instrucción tipo J

6	26
Cód ope	Desplazamiento añadido al PC

Riesgos por dependencia de datos

- Adelantar los resultados a la ALU requiere la incorporación de
  - Tres entradas adicionales en cada multiplexor de la ALU
  - Tres rutas a las nuevas entradas. Las rutas corresponden al adelantamiento
    - (1) la salida de la ALU al final de EX
    - (2) la salida de la ALU al final de la etapa MEM
    - (3) la salida de la memoria al final de la etapa MEM



Riesgos de control

- Pueden provocar mayor pérdida de rendimiento que los riesgos por dependencia de datos.

- Cuando se ejecuta un salto pueden ocurrir dos cosas, en función de la evaluación de la condición de salto (realizada en fase EX):

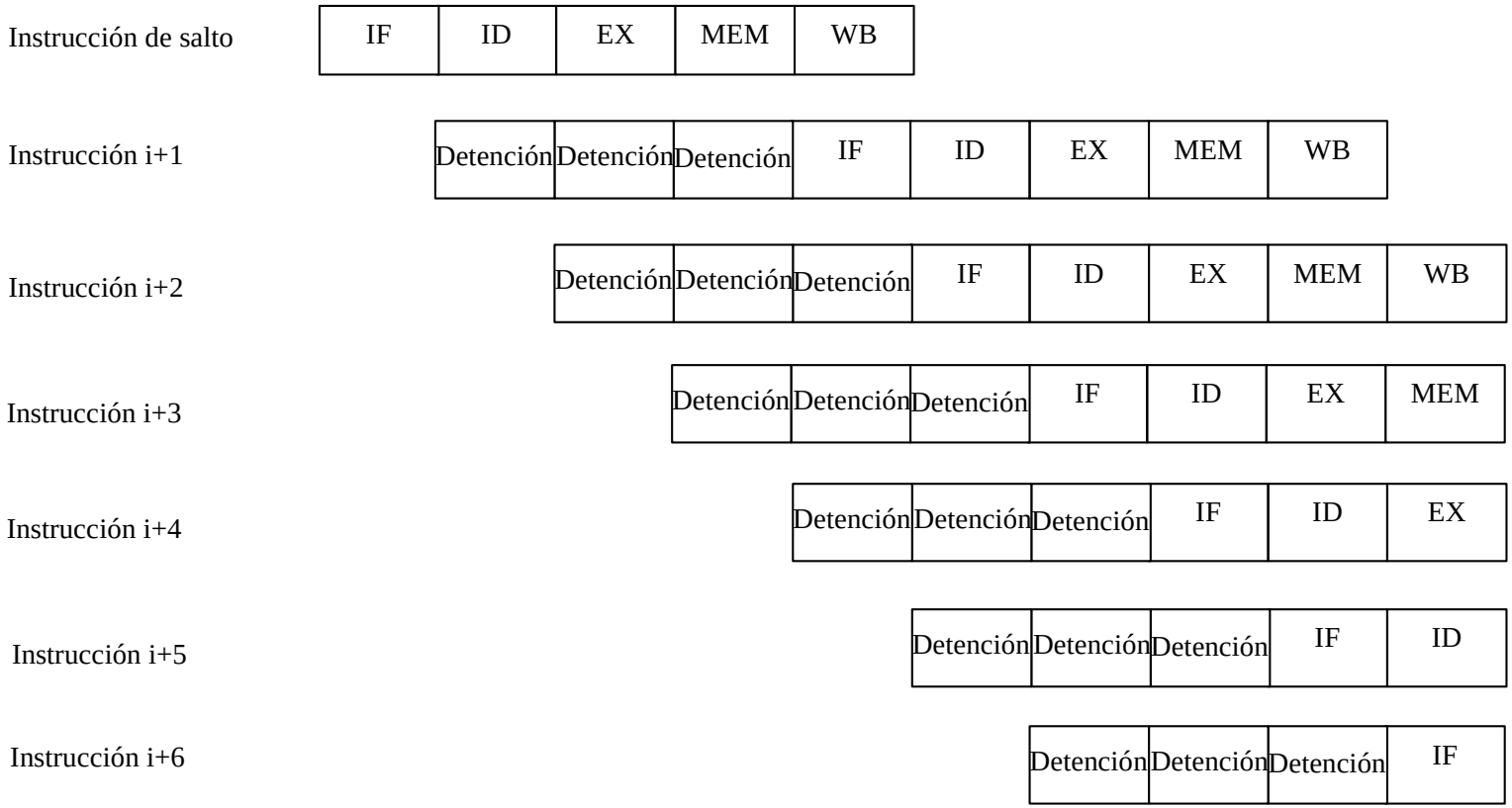
  - **El salto es efectivo**: El PC cambia su valor por una dirección calculada por la ALU al final de MEM, tras calcular la dirección y la comparación

    $ALUoutput \leftarrow PC + ((IR_{16})^{16} \# \# IR_{16..31})$

  - **El salto no es efectivo**: El PC cambia su valor por PC+4

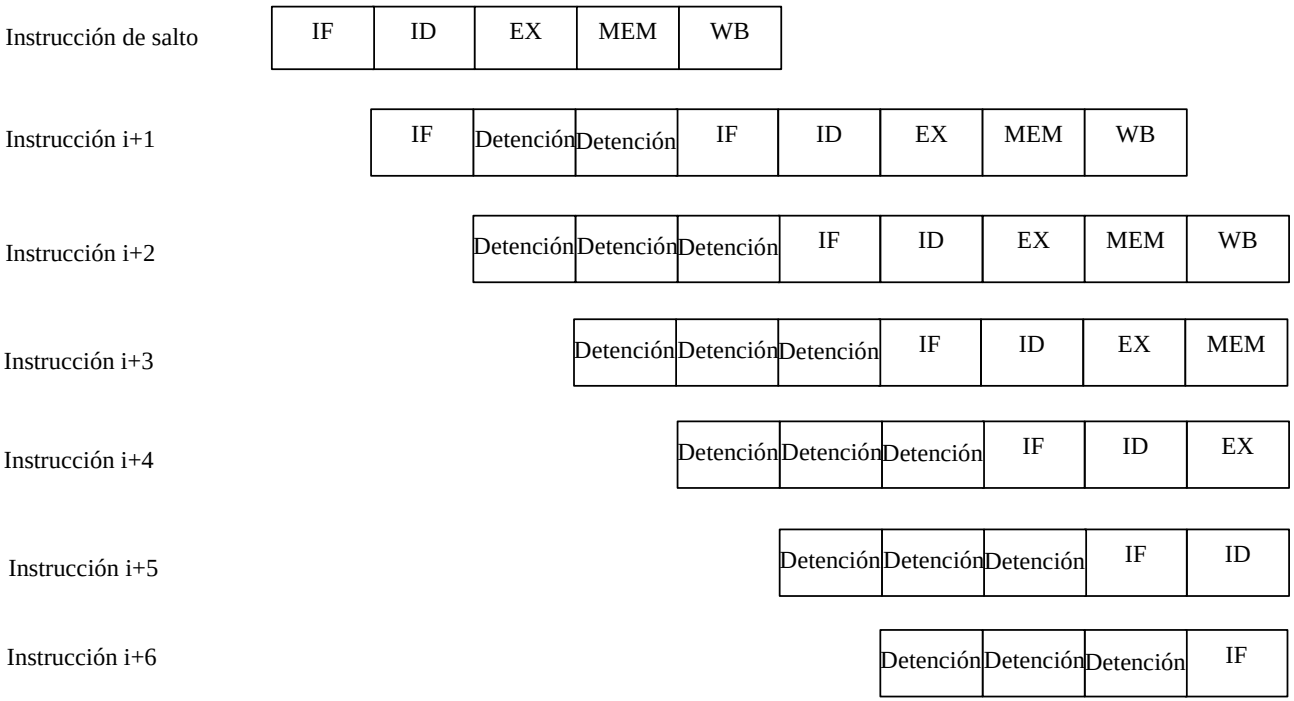
Riesgos de control

- **El método más simple:** detener el cauce tan pronto como detectemos el salto antes de alcanzar la etapa MEM, que determina el nuevo PC.



Riesgos de control

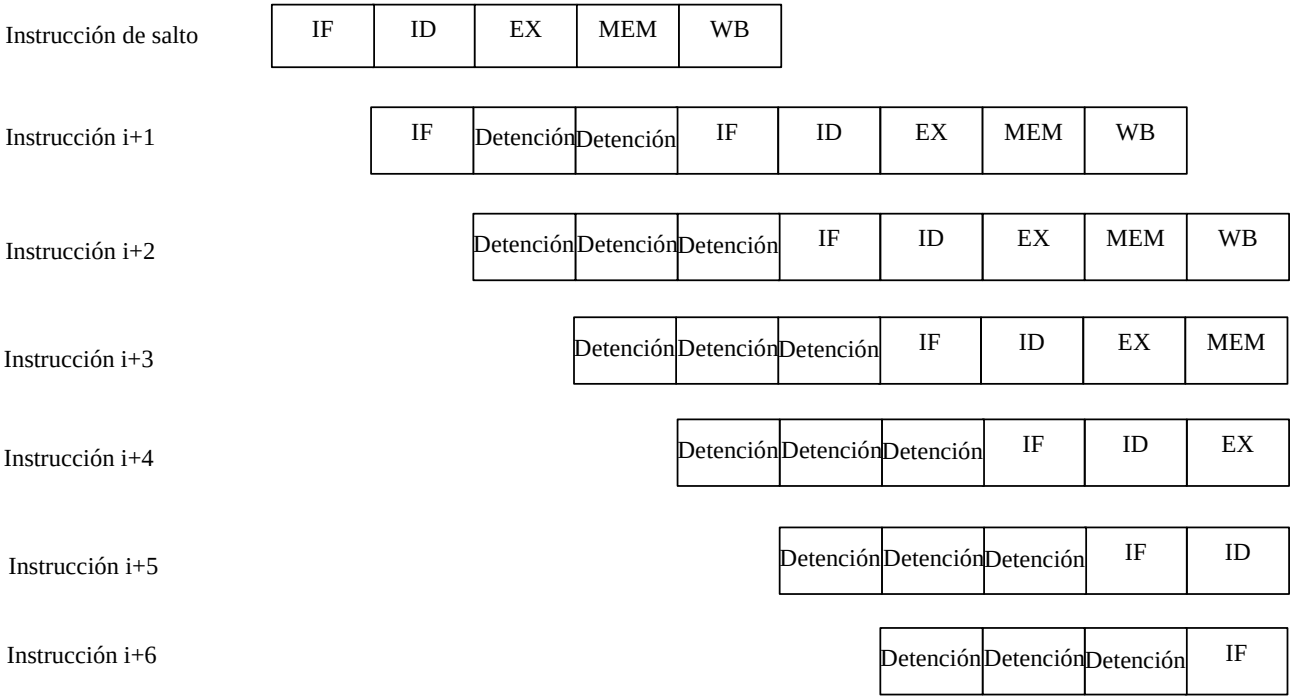
- La detención no ocurre hasta después de la etapa ID (no queremos parar el cauce hasta que sepamos que la instrucción es un salto).



- El ciclo IF de la instrucción siguiente al salto debe repetirse en cuanto conocemos el resultado del salto. Por eso, el primer ciclo IF es esencialmente una detención, porque nunca realiza trabajo útil.

Riesgos de control

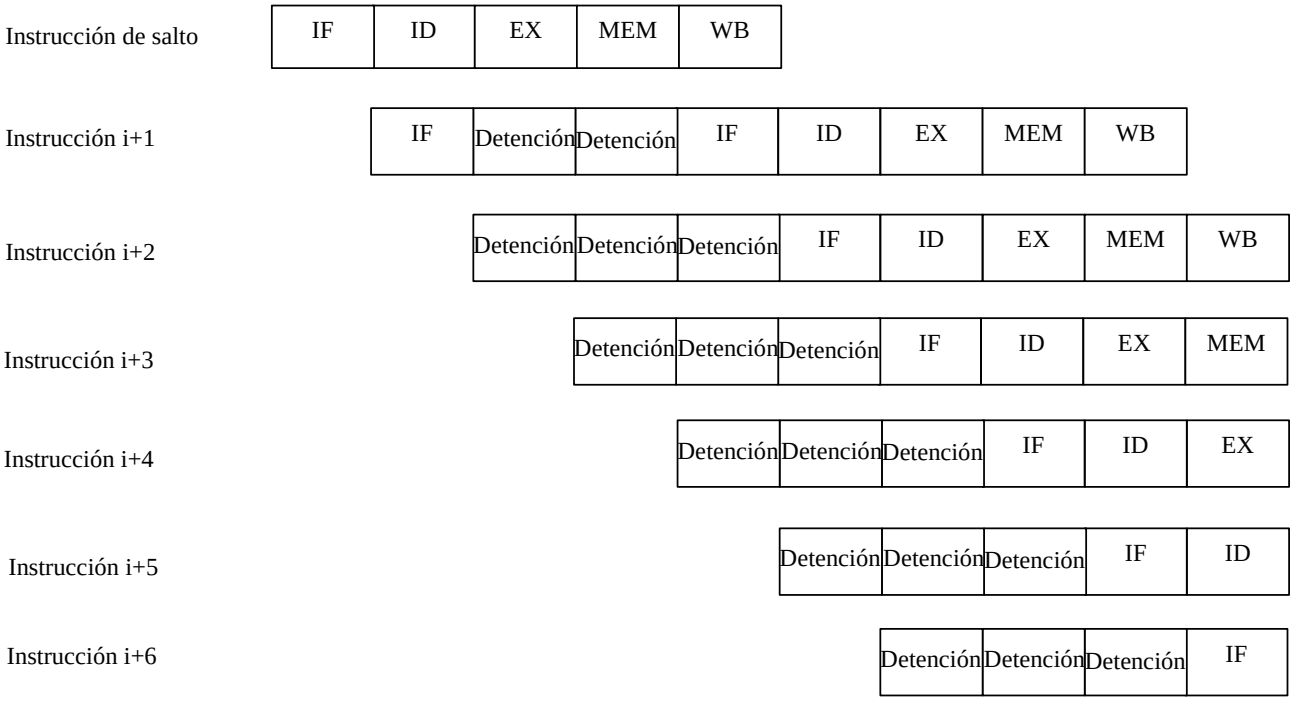
- La detención no ocurre hasta después de la etapa ID (no queremos parar el cauce hasta que sepamos que la instrucción es un salto).



- Un salto causa una detención de tres ciclos en el cauce MIPS: Un ciclo corresponde la repetición de IF y dos ciclos están inactivos.

Riesgos de control

- La detención no ocurre hasta después de la etapa ID (no queremos parar el cauce hasta que sepamos que la instrucción es un salto).



- Esta detención puede implementarse fijando el registro IF/ID a cero en los tres ciclos. Debemos resaltar que si el salto no es efectivo, entonces la repetición de la etapa IF es innecesaria.

Ejemplo

Supongamos una frecuencia de salto de un 30% y un CPI ideal de 1, ¿Qué relación de velocidad encontramos entre la máquina segmentada ideal y la máquina con detenciones por saltos?.

Ejemplo

Supongamos una frecuencia de salto de un 30% y un CPI ideal de 1, ¿Qué relación de velocidad encontramos entre la máquina segmentada ideal y la máquina con detenciones por saltos?.

CPIideal = 1

CPImáquina con detenciones por saltos = 0.3*4+0.7*1= 1.2 + 0.7 = 1.9

Relación de rendimiento = 1.9/1

Riesgos de control

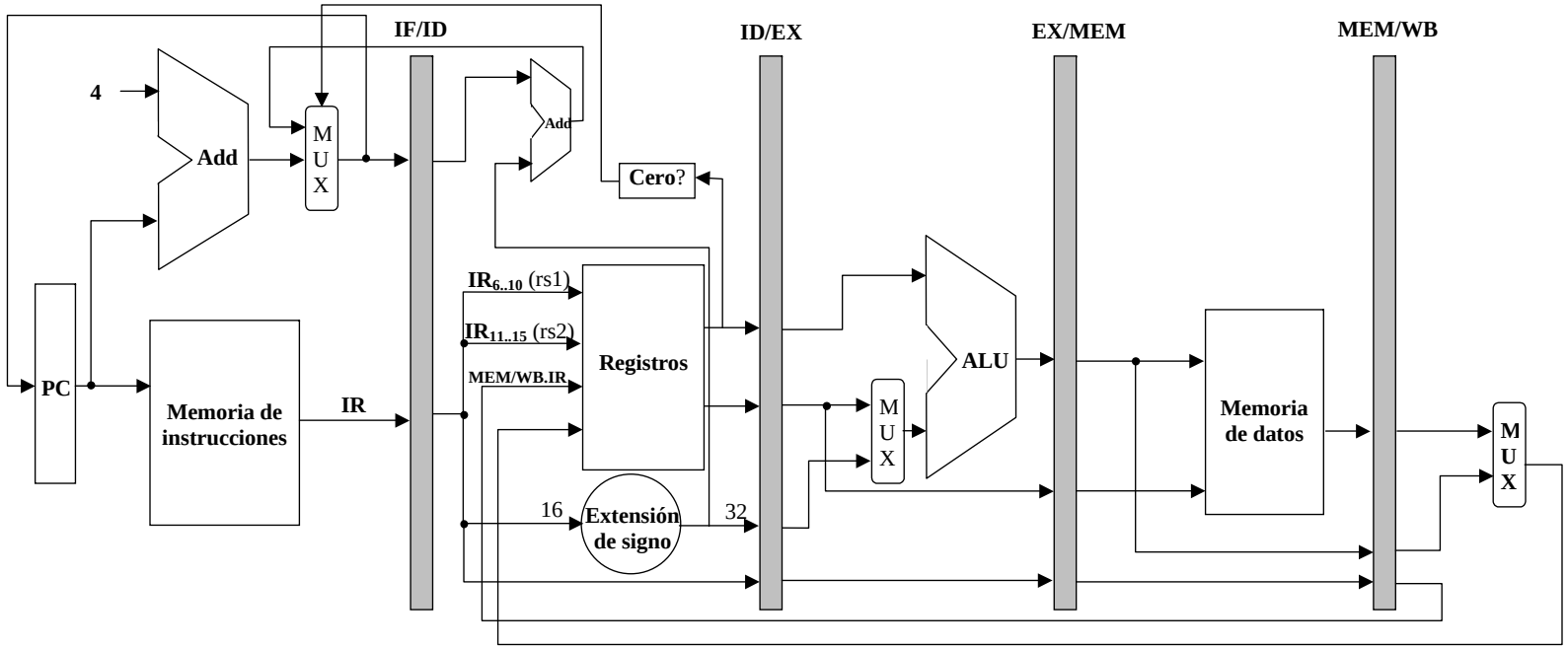
- **Reducción del número de ciclos de detención de salto**
  - La relación de rendimiento entre la máquina ideal y la máquina con detenciones por saltos es 1.9, por lo tanto la reducción del número de detenciones es crítica
- **Dos pasos** para la reducción del número de detenciones por salto:
  - 1. Evaluar la condición lo antes posible en el cauce
  - 2. Calcular el PC efectivo lo antes posible en el cauce
- Para optimizar el comportamiento del salto, deben realizarse ambas cosas

Riesgos de control

- En MIPS los saltos (BEQZ y BNEZ) requieren comprobar si un registro es igual a cero.
- Es posible completar esta decisión al final del ciclo ID moviendo el test cero a este ciclo.
- Para aprovechar este adelantamiento de la decisión de salto, ambos PCs (efectivo y no efectivo) deben calcularse previamente.
- Computar el destino del salto durante ID requiere un sumador adicional porque la ALU, no está disponible hasta EXE.

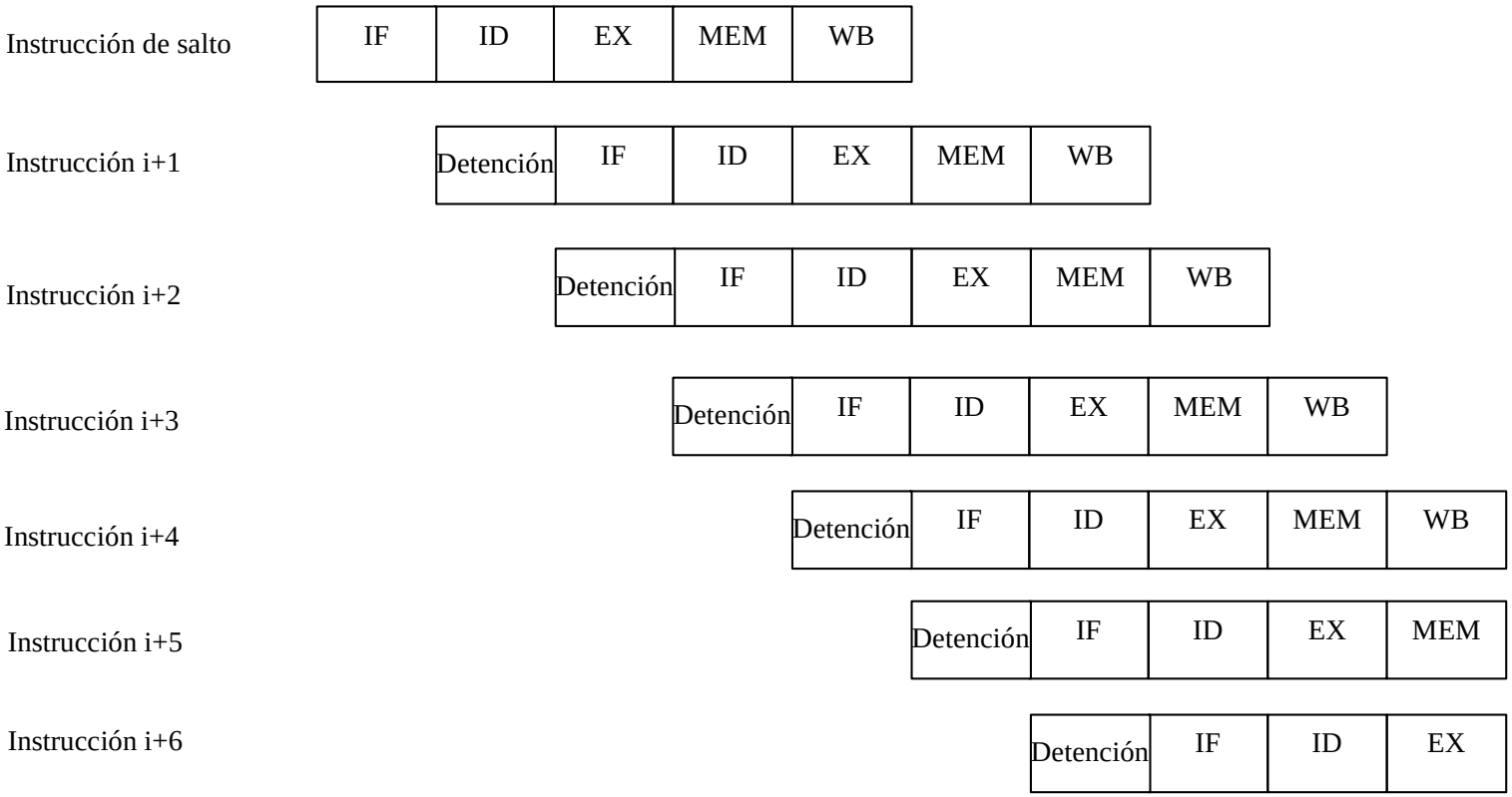
Riesgos de control

- Estructura revisada de la segmentación, se muestra el uso de un sumador separado para calcular la dirección destino del salto:



Riesgos de control

- En estas condiciones sólo es necesario un ciclo de detención en los saltos



Riesgos de control

- Tanto el cálculo de la dirección de salto como la evaluación de la condición se realiza para todas las instrucciones.
- Sólo se sustituye el PC en caso de evaluación positiva de la condición y de que la instrucción sea de salto.

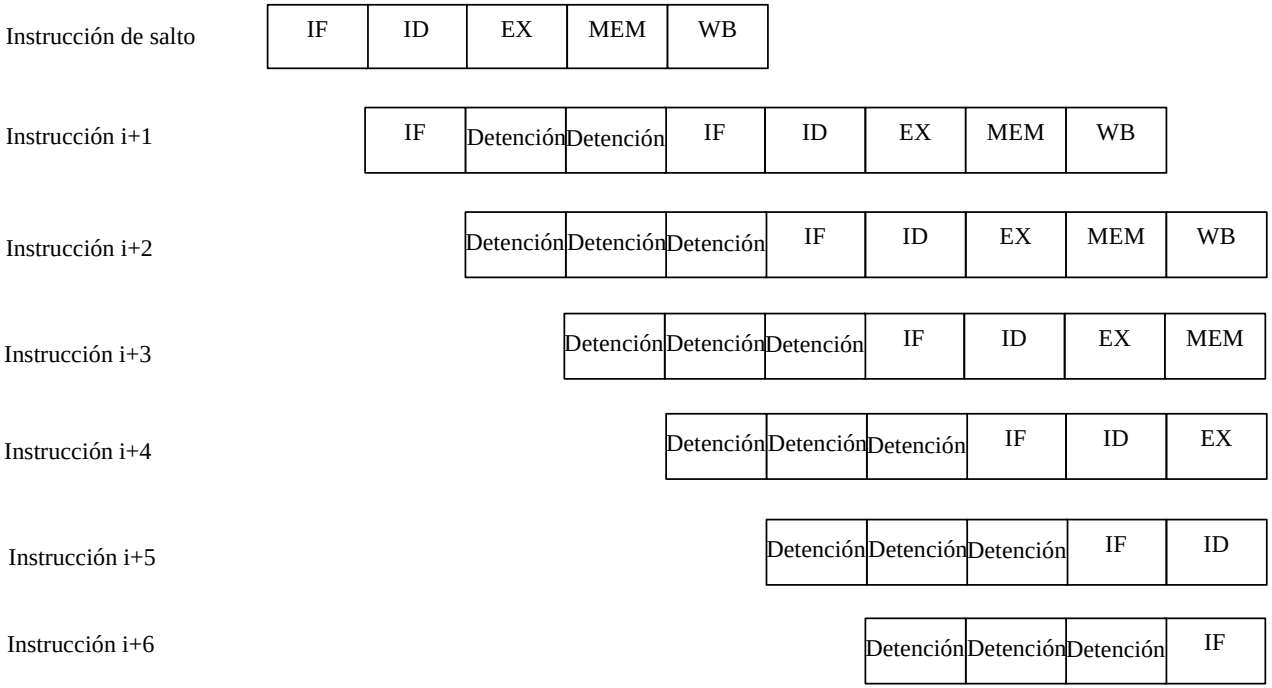
Etapa	Cualquier instrucción
IF	IF/ID.IR ← Mem[PC]; IF/ID.NPC, PC ← (if (Regs[IF/ID.IR$_{6..10}$] op 0) {IF/ID.NPC+(IF/ID.IR$_{16}$)^{16} ## IF/ID.IR$_{16..31}$} else {PC+4});
ID	ID/EX.A ← Regs [IF/ID.IR$_{6..10}$]; ID/EX.B ← Regs [IF/ID.IR$_{11..15}$]; ID/EX.IR ← IF/ID.IR ID/EX.Inm ← (IF/ID.IR$_{16}$)^{16} ## IF/ID.IR$_{16..31}$

- En general, cuanto más profundo es el cauce, peor es la penalización en ciclos de retardo para los saltos.

Reducción de las penalizaciones de saltos en la segmentación

Congelación de la segmentación:

- El esquema más sencillo es detener todas las instrucciones después del salto, hasta conocer el destino correcto. La sencillez del esquema es su principal atractivo.

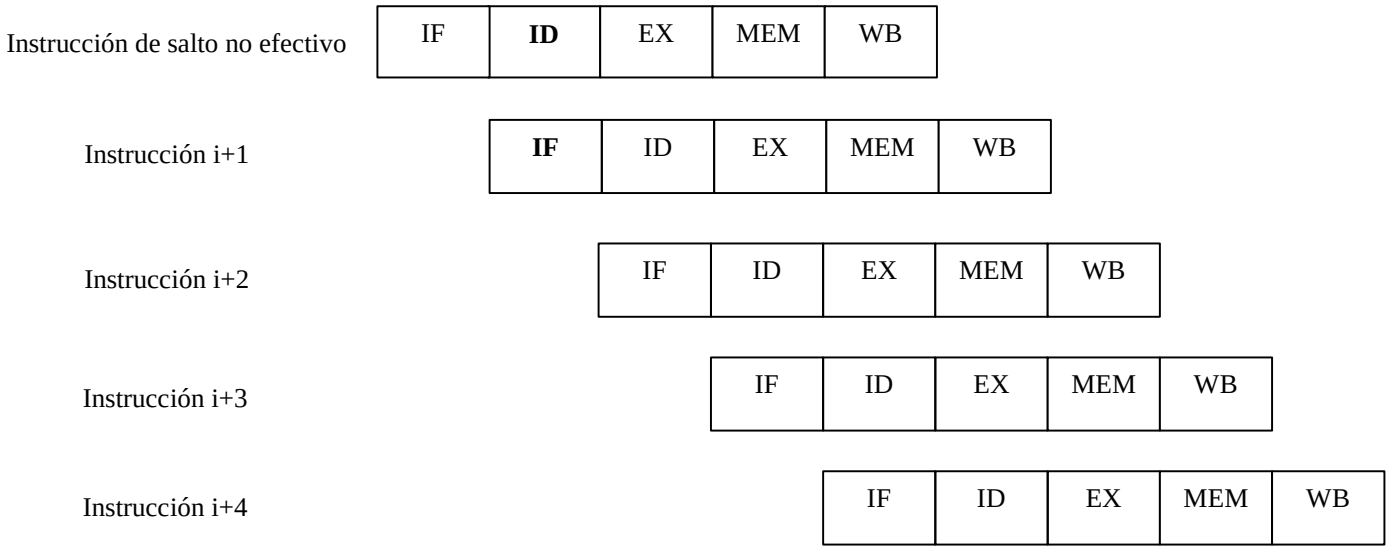


Riesgos de control

- **Reducción de las penalizaciones de saltos en la segmentación**
  - **Predecir el salto como no efectivo**
  - En este esquema permitimos que el hardware continúe como si el salto no se ejecutase.
  - La complejidad radica en no cambiar el estado de la máquina hasta que no se conozca el resultado del salto. Esto supone el conocimiento de cuando una instrucción cambia el estado y como deshacer el cambio.
  - Se implementa continuando la búsqueda de instrucciones como si no ocurriese nada extraordinario. Si el salto es efectivo detenemos la segmentación, recomenzamos las búsquedas y deshacemos los cambios de estado (una posibilidad simple es limpiar la segmentación).

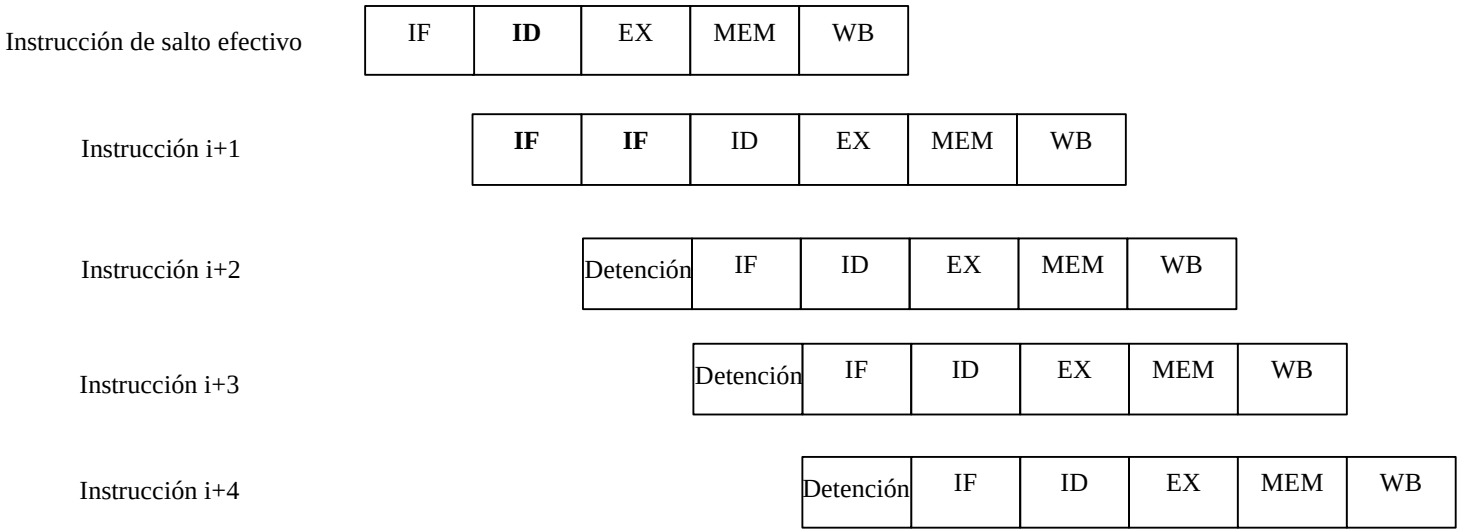
Riesgos de control

- **Reducción de las penalizaciones de saltos en la segmentación**
  - **Predecir el salto como no efectivo**
  - Cuando el salto no es efectivo: Se determina en ID, simplemente continuamos



Riesgos de control

- **Reducción de las penalizaciones de saltos en la segmentación**
  - **Predecir el salto como no efectivo**
  - Cuando el salto es efectivo durante ID, reanudamos la búsqueda en el destino del salto. Esto hace que todas las instrucciones que siguen al salto se detengan un ciclo de reloj.



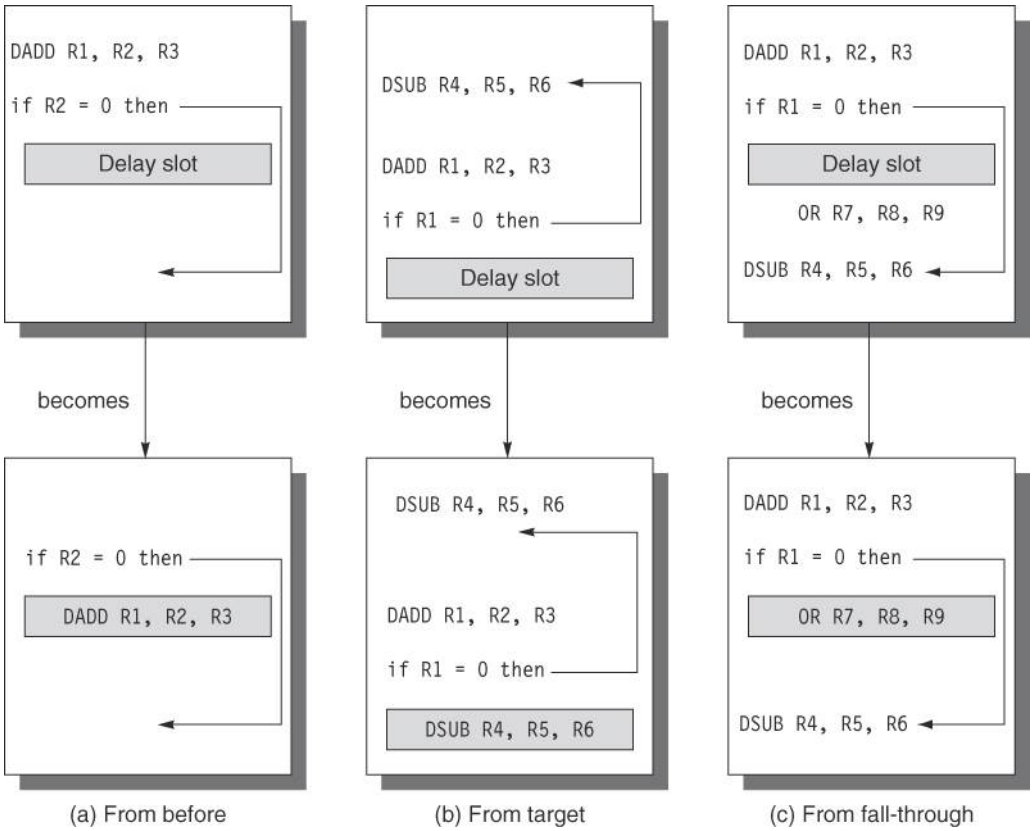
Riesgos de control

- **Reducción de las penalizaciones de saltos en la segmentación**
  - **Predecir el salto como efectivo**
  - Una vez decodificado el salto y calculada la dirección destino, suponemos que el salto se va a realizar y comenzamos la búsqueda y ejecución en el destino.
  - En MIPS no se conoce la dirección destino antes de conocer la evaluación del salto, por lo tanto esta estrategia no es útil.
  - En máquinas con códigos de condición más potentes (más lentas) el destino del salto se conoce antes que la evaluación del salto, es en esta situación donde el esquema tiene sentido.

Riesgos de control

- **Reducción de las penalizaciones de saltos en la segmentación**
  - **Saltos retardados**
    - Ejecutar instrucciones **independientes del salto** durante los ciclos de retardo (delay-slot)
    - Estas instrucciones se ejecutarán **siempre** (tanto si el salto es tomado como si no)
    - Trabajo del compilador: hacer que las instrucciones sean válidas y útiles
    - El número de instrucciones situadas a continuación de un salto que se ejecutan puede variar con la arquitectura

Planificación del salto retardado



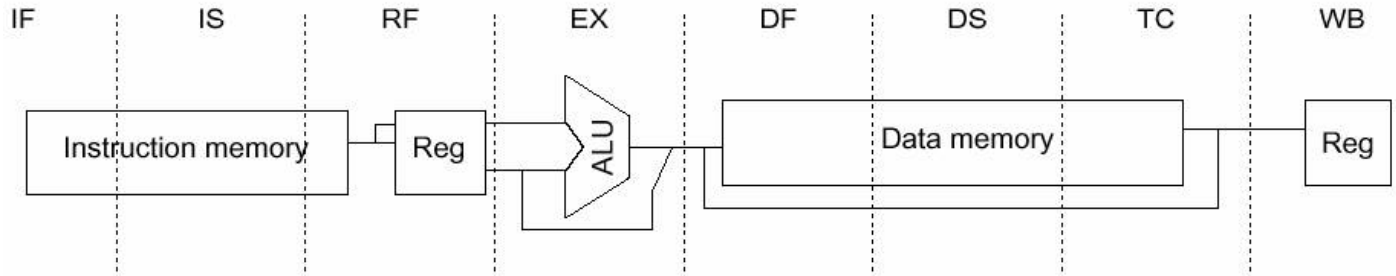
Riesgos de control

- **Planificación del salto retardado**
  - **Si las instrucciones se toman de antes del salto:**
    - no debe haber dependencia entre ellas y el salto
    - siempre mejora el rendimiento
  - **Si las instrucciones se toman del destino del salto:**
    - no debe afectar al programa el que se ejecuten si el salto no se toma
    - puede haber ocasiones en las que el código no se copie, sino que se duplique (al destino se llega desde varios puntos)
    - mejora el rendimiento si el salto se toma
  - **Si las instrucciones se toman a continuación del salto:**
    - no debe afectar al programa el que se ejecuten si el salto se toma
    - mejora el rendimiento si el salto no se toma

Tienen bastante utilidad en cauces sencillos, pero en cauces más complejos su efectividad es poca

Ejemplo: MIPS R4000

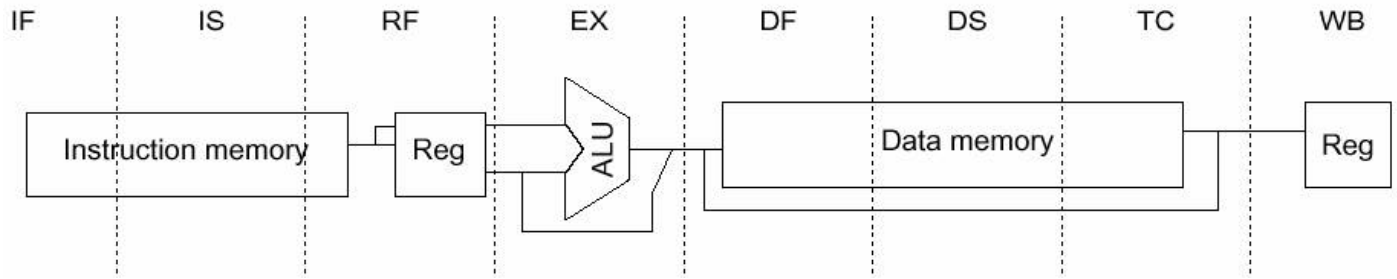
- Representativo de arquitecturas que aparecieron a partir de los 80
- Máquina de 64 bits
- Implementa el repertorio MIPS-3
- Cauce con 8 etapas (supersegmentación)
- Esto le permitía utilizar una frecuencia alta (100-200 MHz)
- Utilizado por NEC, Nintendo, Silicon Graphics, Sony...



Ejemplo: MIPS R4000

- 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.
- RF: 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



Ejemplo: MIPS R4300

- Procesador de 64 bits utilizado para aplicaciones empotradas
- La segmentación del R4300 implementa el punto flotante mediante la extensión de la longitud del pipeline a través de la adición de múltiples etapas EX para las operaciones de punto flotante. Esto introduce una complejidad: la posibilidad de completar instrucciones fuera de orden. Una instrucción entera puede completarse y escribir su resultado antes de que termine una instrucción en punto flotante.
- Cuatro versiones:

Clock frequency (MHz)	Performance SPECInt 92	Performance SPECFP 92	Power consumption (W)
80	48	36	1.5
100	60	45	1.8
133	80	60	1.9
167	100	75	2.4

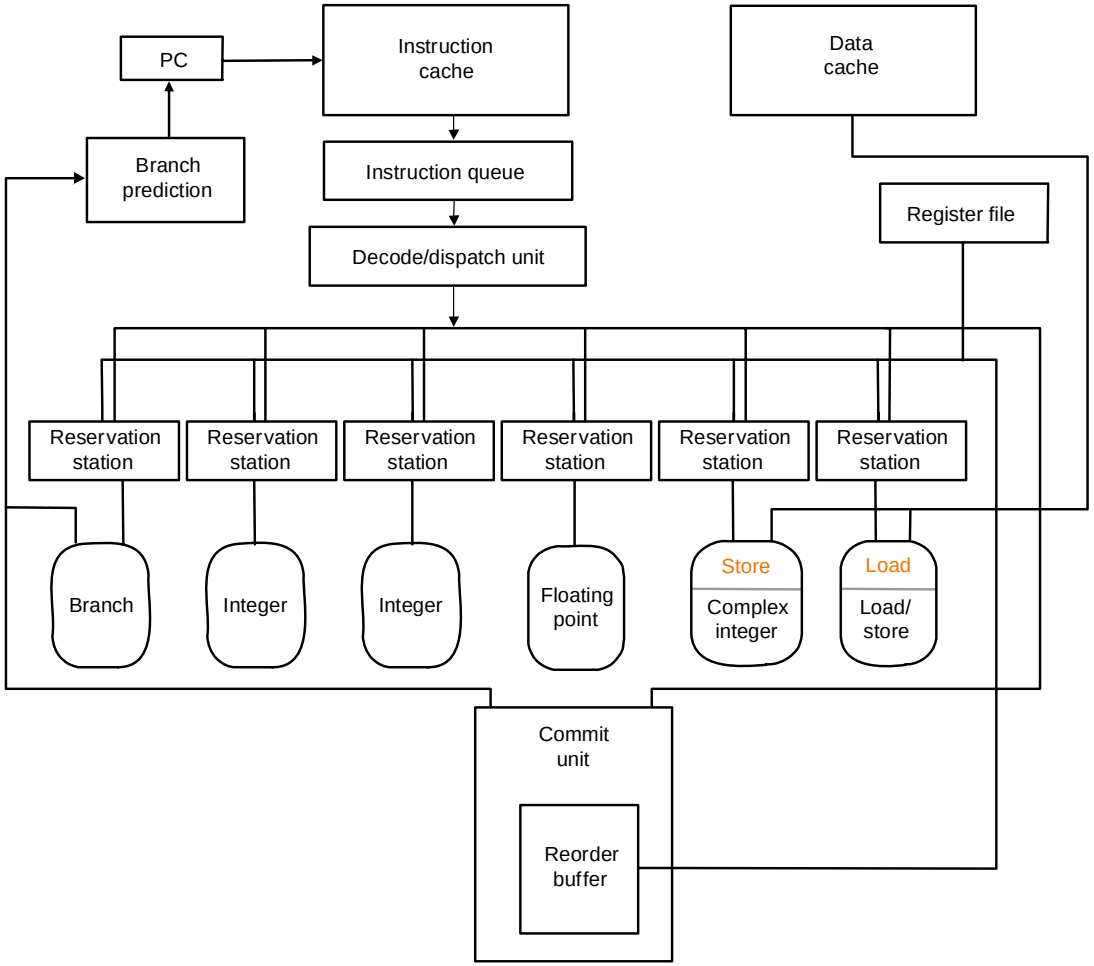
Ejemplo: Power PC 604

- El PowerPC 604 se introdujo en diciembre de 1994.
- El 604 es un procesador superescalar capaz de emitir cuatro instrucciones simultáneamente (3 enteras y una en coma flotante). El 604 cuenta con un cauce de 6 etapas y con seis unidades de ejecución (dos para operaciones enteras, dos para operaciones en coma flotante y dos para procesamiento de saltos) que pueden trabajar en paralelo, puede terminar hasta seis instrucciones en cada ciclo
- El PowerPC 604 contiene 3,6 millones de transistores y fue fabricado por IBM y Motorola. Funcionan a velocidades de entre 100 y 180 MHz.

Ejemplo: Pentium Pro

- El Pentium Pro se introdujo en 1995.
- El Pentium Pro es un procesador superescalar capaz de emitir tres instrucciones simultáneamente. El Pentium Pro cuenta con una pipeline de 6 etapas y con seis unidades de ejecución que pueden trabajar en paralelo, puede terminar hasta 5 instrucciones en cada ciclo.
- El pentium en vez de segmentar las instrucciones de tamaño variable 80x86,lo que hace es que la unidad de decodificación del Pentium Pro traduce las instrucciones intel a microoperaciones de 72 bits de longitud fija y envía estas microoperaciones al buffer de reordenación y a las estaciones de reserva.

Organización genérica del pipeline de los proceadores Pentium pro y Power PC 604



PowerPC 604 y Pentium Pro

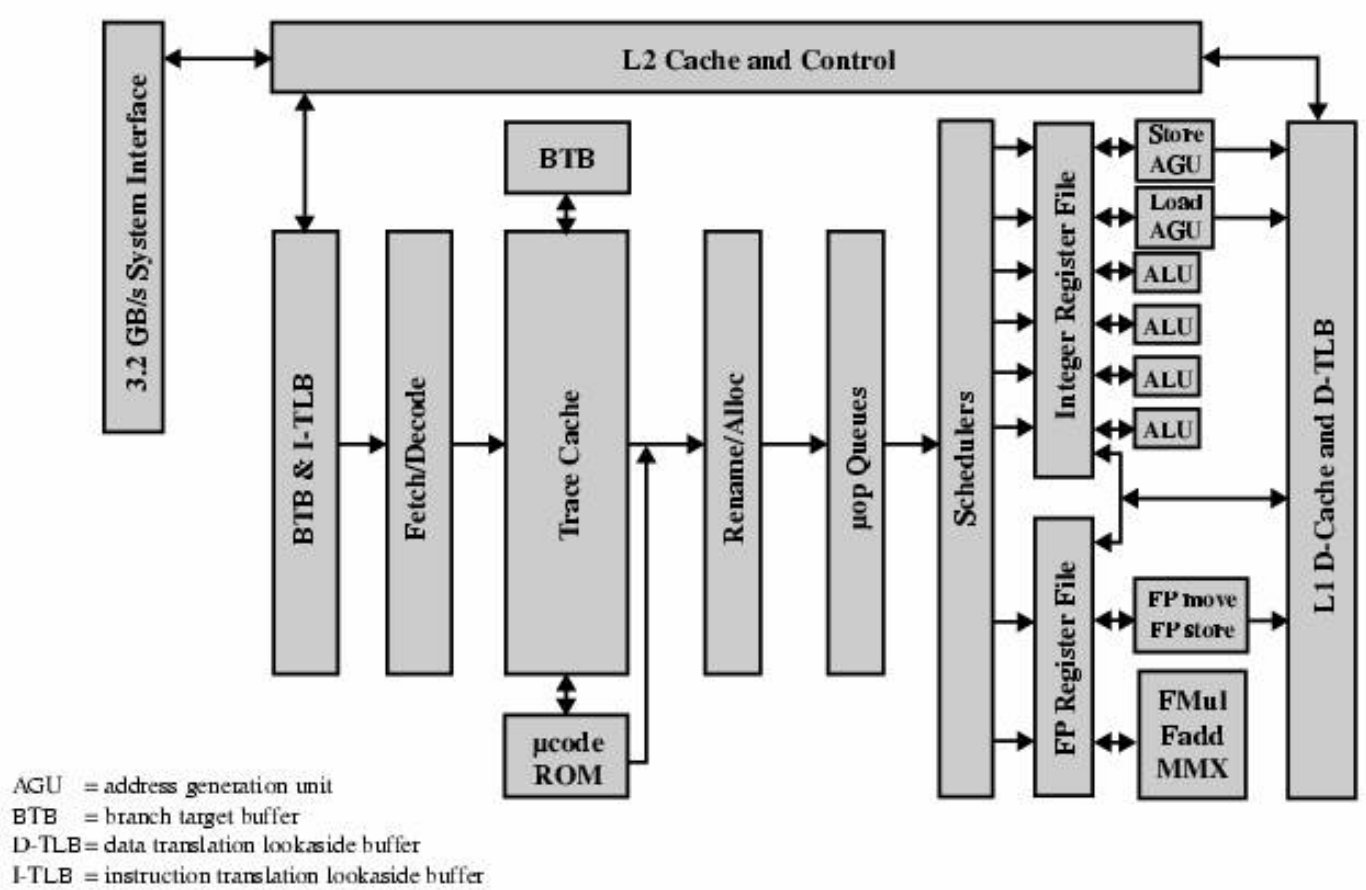
- La cache de instrucciones busca 16 bytes de instrucciones y los envía a una cola de instrucciones; cuatro para el PowerPC y un número de isntrucciones variables para el Pentium.
- Después unas cuantas de ellas se decodifican
- Ambos utiliza una tabla de historia de saltos de 512 entradas para predecir los saltos.
- La unidad de envío manda cada instrucción y sus operandos a la estación de reserva de una de las seis unidades funcionales. Esta unidad también reserva una entrada en el buffer de reordenación de la unidad de commit para esta instrucción.

Ejemplo: Pentium 4

- Pentium 4
- Introducido en el 2001 con ciclo de reloj de más de 1GHz
- El cauce del Pentium 4 tiene 20 etapas
- Tiene registros XMM (extended multi media) de 128 bits para dos operaciones de coma flotante empaquetadas de 64bits cada una
- Los registros de 128 bits manejan también enteros grandes
- Está equipado con un conjunto operaciones SIMD
- Revisions
  - Northwood (1/2002) – 21 etapas
  - Prescott (2/2004) – 31 etapas

Ejemplo: Pentium 4

Diagrama de bloques del Pentium 4

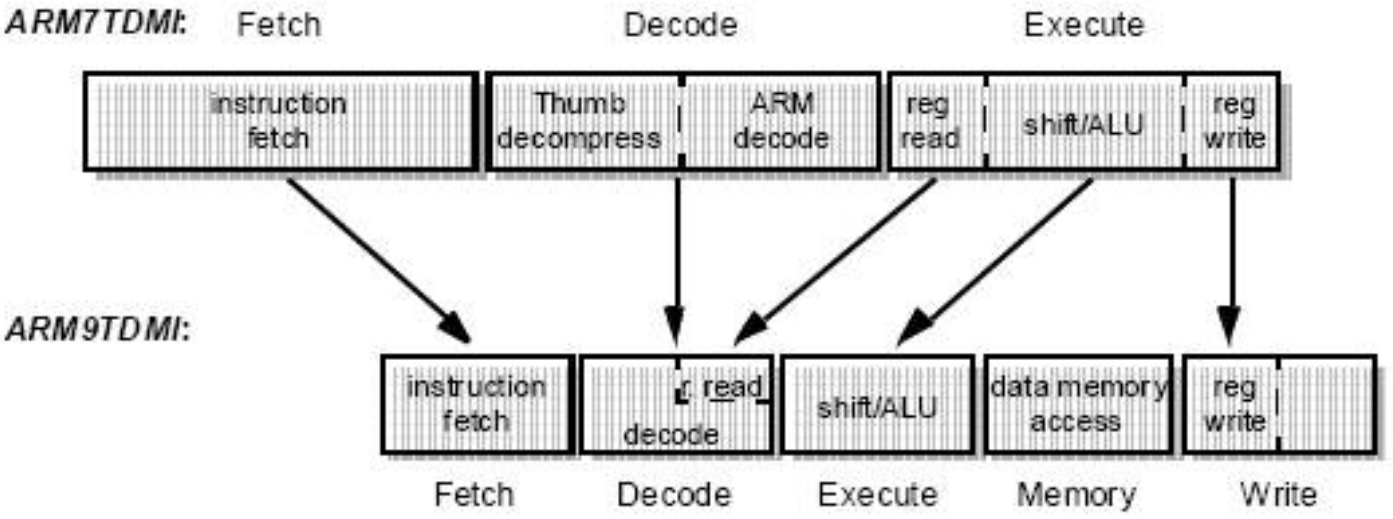


Ejemplo: Pentium 4

- **Operación del Pentium 4:**
- Busca instrucciones de memoria en orden
- Traduce las instrucciones en una o mas instrucciones RISC de longitud fija (micro oeraciones)
- Ejecuta las micro operaciones en un pipeline superescalar.
  - Las micro operaciones pueden ejecutarse fuera de orden
- Almacena los resultados de las micro peraciones en el conjunto de registros en el orden del flujo del programa original.
- Capa externa CISC con núcleo interno RISC
- El cauce del núcleo interno RISC tienen al menos 20 etapas
  - Algunas micro operaciones requieren múltiples etapas de ejecución (pipeline más larga)

Ejemplo: Procesadores ARM

- **ARM:** Advances RISC Machines
- La familia de procesadores ARM se dirige fundamentalmente al mercado de los sistemas embebidos
- El primer ARM se desarrolló en Inglaterra entre 1983 y 1985
- Han aparecido 5 versiones del repertorio de instrucciones ARM
- Hasta la versión ARM7 inclusive se implementaban con una cauce de 3 etapas, después pasaron a 5 etapas. Posteriormente se introdujo un cauce de 6 etapas para poder utilizar ciclos de reloj más pequeños.



Ejemplo: Procesadores ARM

- **ARM7: lanzamiento en 1994**
  - Objetivos: Teléfonos móviles, agendas, impresoras, cámaras, PDAs,
  - ARM7TDMI(-S) :Pipeline de 3 etapas.
  - ARM7EJ-S Pipeline de 5 etapas.
- **ARM9: lanzamiento en1997**
  - Objetivos: Teléfonos móviles, buscas, smartphones, decodificadores de TV,
  - ARM946E-S (1999) Pipeline de 5 etapas.
- **ARM11 : lanzamiento en 2000**
  - Objetivos: Cámaras digitales, smartphones, e-book readers, media centers, ...
  - ARM1136J(F)-S (2002) Pipeline de 8 etapas.