

TEMA 1

MEJORA DEL RENDIMIENTO DEL PROCESADOR

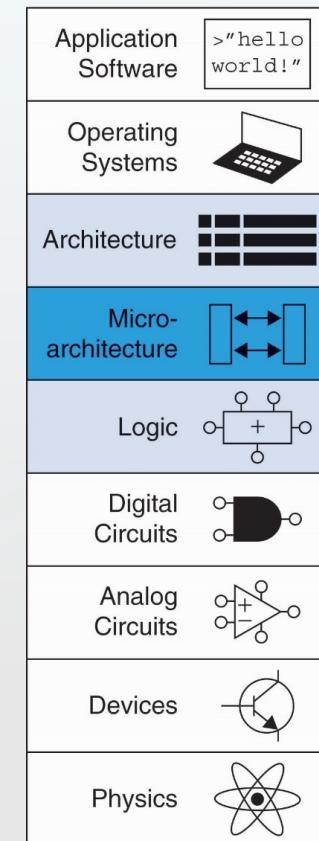
Transparencias basadas en:

- José Manuel Mendías Cuadros. Fundamentos de los Computadores II. UCM.
- Libro Patterson-Hennessy, capítulo 4
- Libro Harris&Harris



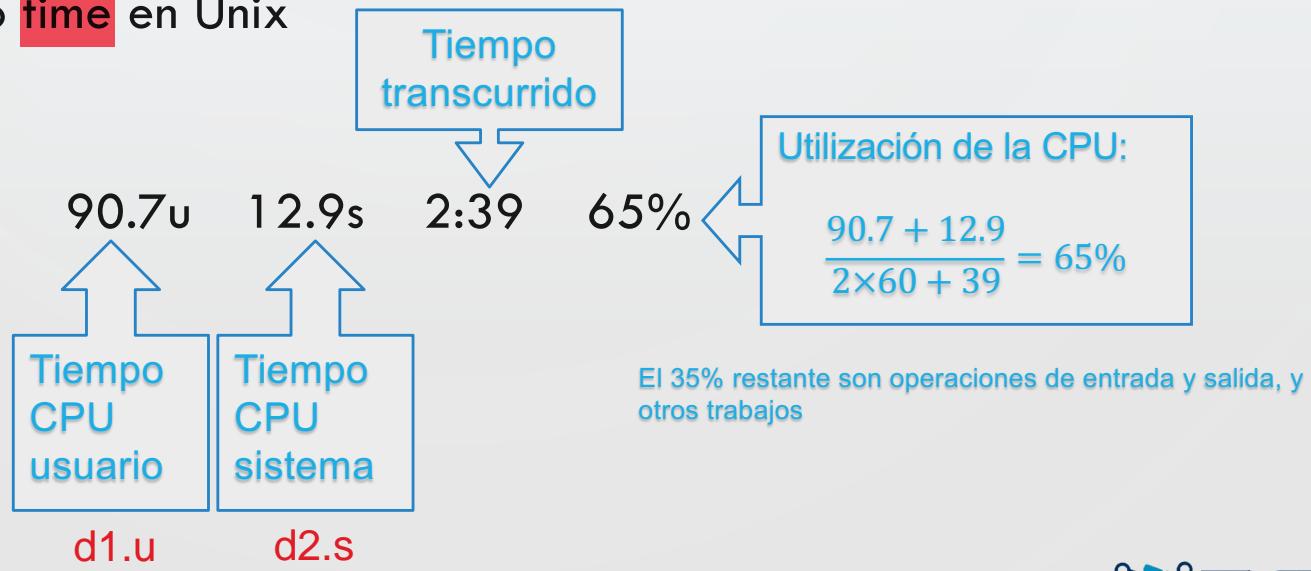
MICROARQUITECTURA

- **Microarquitectura:** cómo implementar una arquitectura (ISA) en hardware
- Vamos a comparar el **rendimiento** de diferentes microarquitecturas para un mismo ISA (RISC-V):
 - Procesador monociclo
 - Procesador multiciclo
 - Procesador segmentado
- El reto para el diseñador de una microarquitectura es satisfacer requisitos de:
 - Coste
 - Consumo de potencia
 - Rendimiento



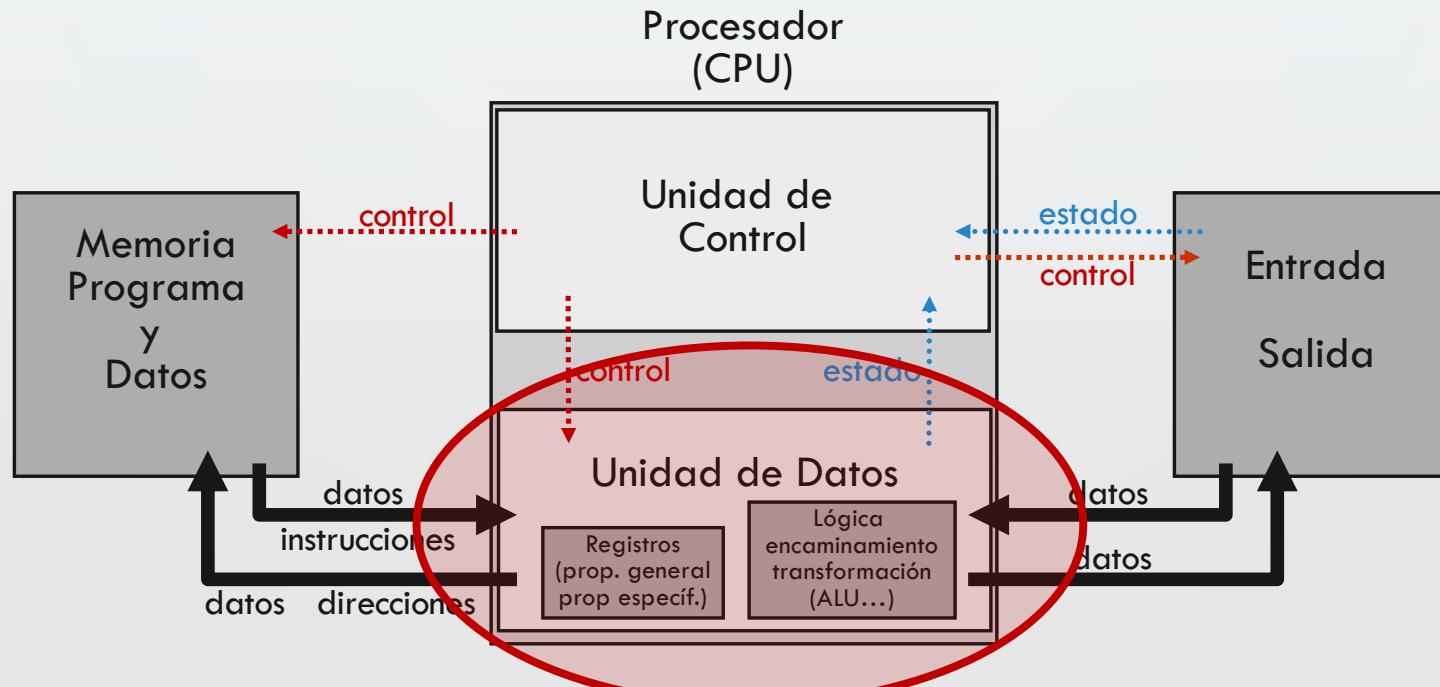
TIEMPO DE EJECUCIÓN DE UN PROGRAMA

- Mediremos el rendimiento usando el **tiempo de CPU**:
 - Se calcula a partir del tiempo empleado en completar un trabajo en el procesador
 - No incluye el tiempo empleado en operaciones de entrada y salida, ni otros trabajos
 - Una parte del tiempo es del usuario, y otra parte del sistema
- Ejemplo: comando **time** en Unix



EL PROCESADOR

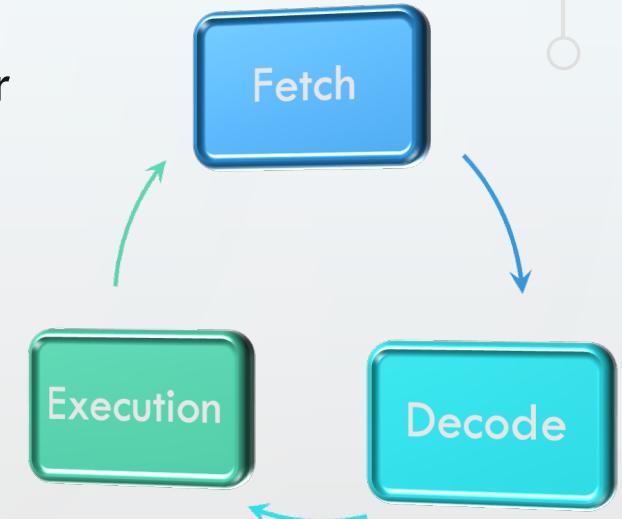
- Un procesador es un circuito **secuencial** gobernado por una señal de reloj (CLK) que sincroniza los cambios de estado
- **OBJETIVO:** ejecutar las acciones indicadas por las instrucciones



CICLO DE INSTRUCCIÓN DEL PROCESADOR

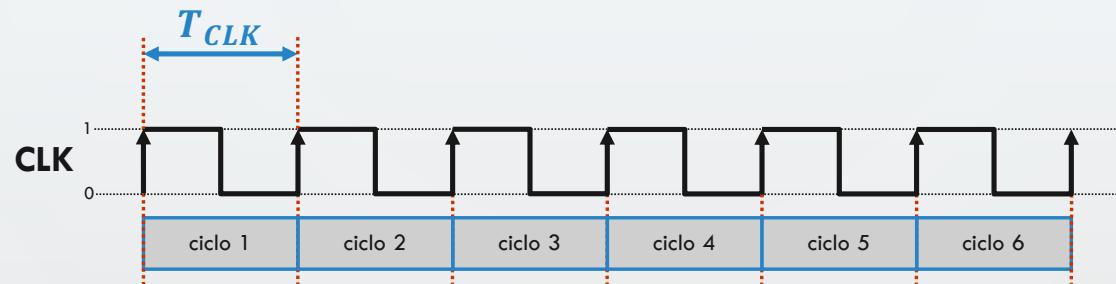
- **Búsqueda de la instrucción (fetching):** el procesador lee de memoria la instrucción a ejecutar. Debe conocer la posición de memoria donde se encuentra dicha instrucción (**PC – Program Counter**).
- **Decodificación de la instrucción:** Debe determinar el tipo de instrucción y los operandos que utiliza a partir de la instrucción binaria leída de memoria (**Formato de instrucción**).
- **Ejecución de la instrucción:** Ejecuta la acción expresada por la instrucción con los operandos especificados.

Ojo, el ciclo de instrucción no es lo mismo que el ciclo de reloj



TIEMPO DE CPU

- El tiempo de ejecución en la CPU (T_{CPU}) es el resultado de multiplicar el número total de ciclos de reloj (NC) por la duración del ciclo (periodo) de la señal de reloj (T_{CLK})



$$T_{CPU} = NC \times T_{CLK}$$

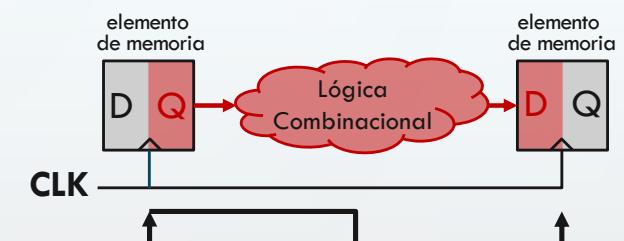
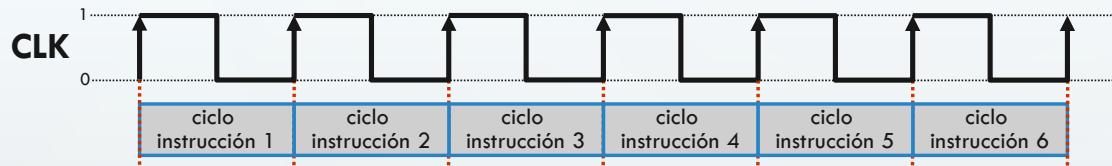
- NC y T_{CLK} dependen de la microarquitectura del procesador y la tecnología empleada
 - En las próximas secciones veremos como varían esos valores con diferentes implementaciones: monociclo, multiciclo y segmentado

PROCESADOR MONOCICLO

- Ciclo de instrucción del procesador monociclo
- Diseño del camino de datos y de la unidad de control
- Tiempo de ciclo

PROCESADOR MONOCICLO

- Todas las transferencias/transformaciones implicadas en cada instrucción se ejecutan en un único ciclo de reloj (entre dos flancos de la señal CLK)



- El reloj llega a todos los elementos de memoria del procesador
- En cada flanko (subida) se cargan los nuevos valores en todos ellos (comienza ciclo instrucción)
- Los nuevos valores se propagan/transforman a través de los circuitos combinacionales y se quedan a las entradas de los elementos de memoria (pero no se cargan)
- Cuando se han estabilizado, puede llegar el siguiente flanko de reloj que carga los nuevos valores en los elementos de memoria y da comienzo a la siguiente instrucción

REPERTORIO RISC-V REDUCIDO

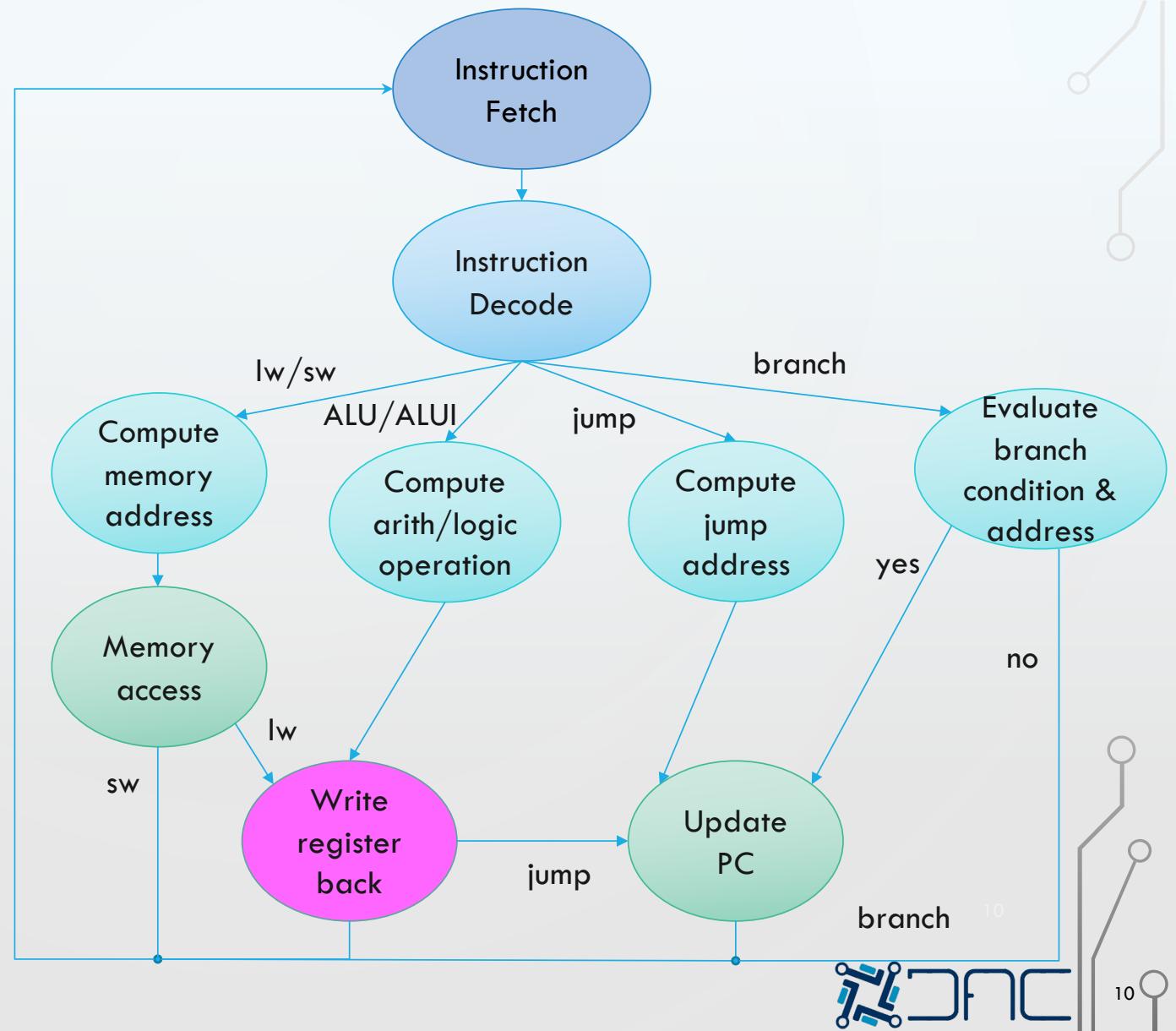
Vamos a diseñar la unidad de datos monociclo de un procesador **RISC-V** (32 bits) con un **conjunto reducido de instrucciones**

Memoria	lw rd, imm _{12b} (rs1)	$rd \leftarrow \text{MEM}[rs1 + sExt(\text{imm})]$	tipo-I
	sw rs2, imm _{12b} (rs1)	$\text{MEM}[rs1 + sExt(\text{imm})] \leftarrow rs2$	tipo-S
ALU	add rd, rs1, rs2	$rd \leftarrow rs1 + rs2$	tipo-R
	sub rd, rs1, rs2	$rd \leftarrow rs1 - rs2$	tipo-R
ALU Inmediato	and rd, rs1, rs2	$rd \leftarrow rs1 \& rs2$	tipo-R
	or rd, rs1, rs2	$rd \leftarrow rs1 rs2$	tipo-R
Saltos	addi rd, rs1, imm _{12b}	$rd \leftarrow rs1 + sExt(\text{imm})$	tipo-I
	andi rd, rs1, imm _{12b}	$rd \leftarrow rs1 \& sExt(\text{imm})$	tipo-I
Saltos	ori rd, rs1, imm _{12b}	$rd \leftarrow rs1 sExt(\text{imm})$	tipo-I
	beq rs1, rs2, imm _{13b}	$PC \leftarrow \begin{cases} (PC + sExt(\text{imm}_{12:1}) \ll 1) \\ \text{else } (PC + 4) \end{cases}$	tipo-B
	jal rd, imm _{21b}	$PC \leftarrow PC + sExt(\text{imm}_{20:1} \ll 1), rd \leftarrow PC + 4$	tipo-J

CICLO DE INSTRUCCIÓN

Memoria	<code>lw rd, imm_{12b}(rs1)</code> <code>sw rs2, imm_{12b}(rs1)</code>
	<code>add rd, rs1, rs2</code>
ALU	<code>sub rd, rs1, rs2</code> <code>and rd, rs1, rs2</code> <code>or rd, rs1, rs2</code>
	<code>addi rd, rs1, imm_{12b}</code>
ALU Inmediato	<code>andi rd, rs1, imm_{12b}</code> <code>ori rd, rs1, imm_{12b}</code>
	<code>jal rd, imm_{21b}</code>
Saltos	<code>beq rs1, rs2, imm_{13b}</code>

TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR



FORMATO DE INSTRUCCIONES

31	25 24	20 19	15 14	12 11	7 6	0	
funct7	rs2	rs1	funct3	rd	op		tipo-R
imm _{11:0}		rs1	funct3	rd	op		tipo-I
imm _{11:5}	rs2	rs1	funct3	imm _{4:0}	op		tipo-S
imm _{12,10:5}	rs2	rs1	funct3	imm _{4:1,11}	op		tipo-B
	imm _{20,10:1,11,19:12}			rd	op		tipo-J

Instrucción	Tipo	funct7 (31:25)	funct3 (14:12)	op (6:0)
Memoria	lw	I	-	010 0000011
		S	-	010 0100011
ALU	add	R	0000000	000
		sub	0100000	000
ALU Inmediato	and	R	0000000	111
		or	0000000	110
Saltos	addi	I	-	000
		andi	-	111 0010011
	ori	I	-	110
	beq	B	-	000 1100011
	jal	J	-	- 1101111

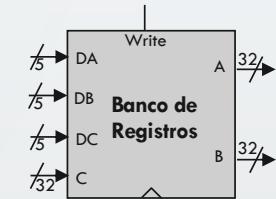
UNIDAD DE DATOS – ELEMENTOS HARDWARE

- Banco de registros (R)

- 32 registros de 32 bits
- Dos puertos de lectura (A,B) y otro de escritura (C)
 - Posibilidad de leer dos registros y escribir en un tercero a la vez

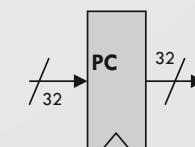
- Interfaz:

- DA: código registro a leer por A
- DB: código registro a leer por B
- DC: código registro a escribir con el contenido que entre por C



- PC

- Registro de 32 bits con carga paralela síncrona



UNIDAD DE DATOS – ELEMENTOS HARDWARE

- **Bancos de memoria (M)**

- 2 Bancos de memoria de 2^{30} palabras de 32 bits

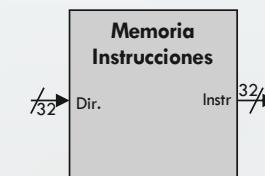
- Direccionables a nivel de byte

- Memoria Instrucciones:

- **Dir**: Dirección de memoria a leer

- **Instr**: Instrucción leída

- Sólo lectura



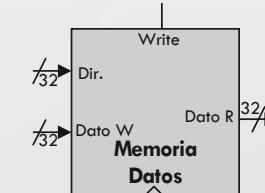
- Memoria de datos:

- **Dir**: Dirección de memoria a leer

- **Data W**: Dato a escribir en memoria

- **Data R**: Dato leído de memoria

- **Read**: lectura; **Write**: escritura; **CS**: chip select

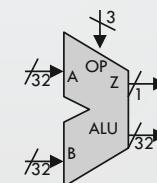


UNIDAD DE DATOS – ELEMENTOS HARDWARE

- ALU

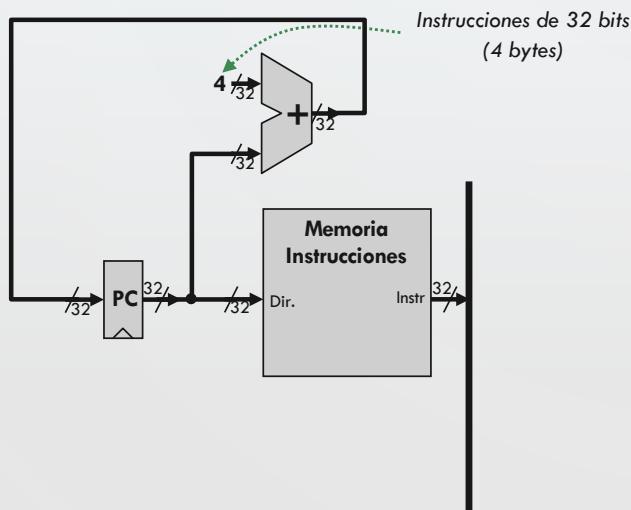
- Unidad aritmético-lógica de dos entradas de 32 bits
- Salida de estado Z (resultado 0)
- 8 operaciones:

OP	Nemo.	Operación
000	ADD	$ALU = A+B$
001	SUB	$ALU = A-B$
010	AND	$ALU = A \text{ AND } B$
011	OR	$ALU = A \text{ OR } B$
100	SLT	IF ($A < B$) then $ALU = 1$ else $ALU = 0$
101	SRL	$ALU = 0, A[31:1]$
110	SLL	$ALU = A[30:0], 0$
111	SRA	$ALU = A[31], A[31:1]$



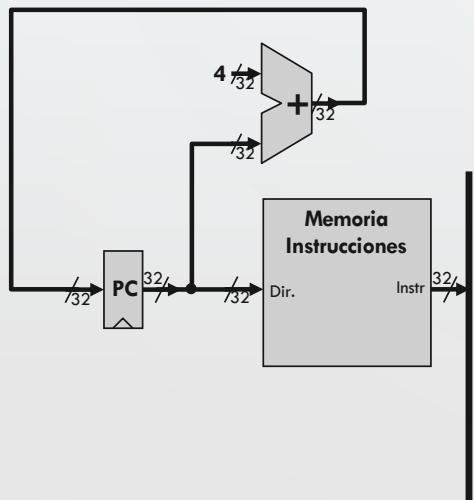
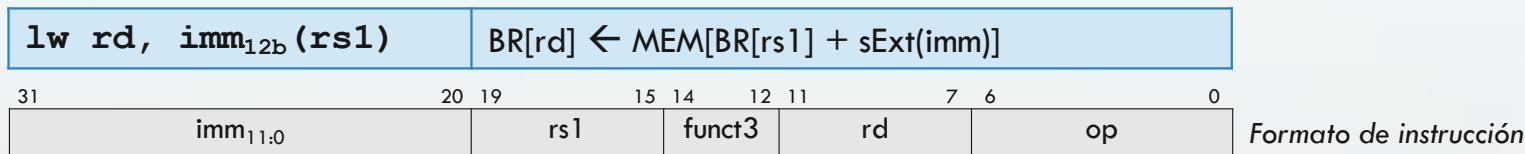
CAMINO DE DATOS

- Búsqueda secuencial de las instrucciones
 - PC contiene la dirección de memoria de la instrucción actual
 - PC debe incrementarse al final del ciclo para apuntar a la siguiente instrucción



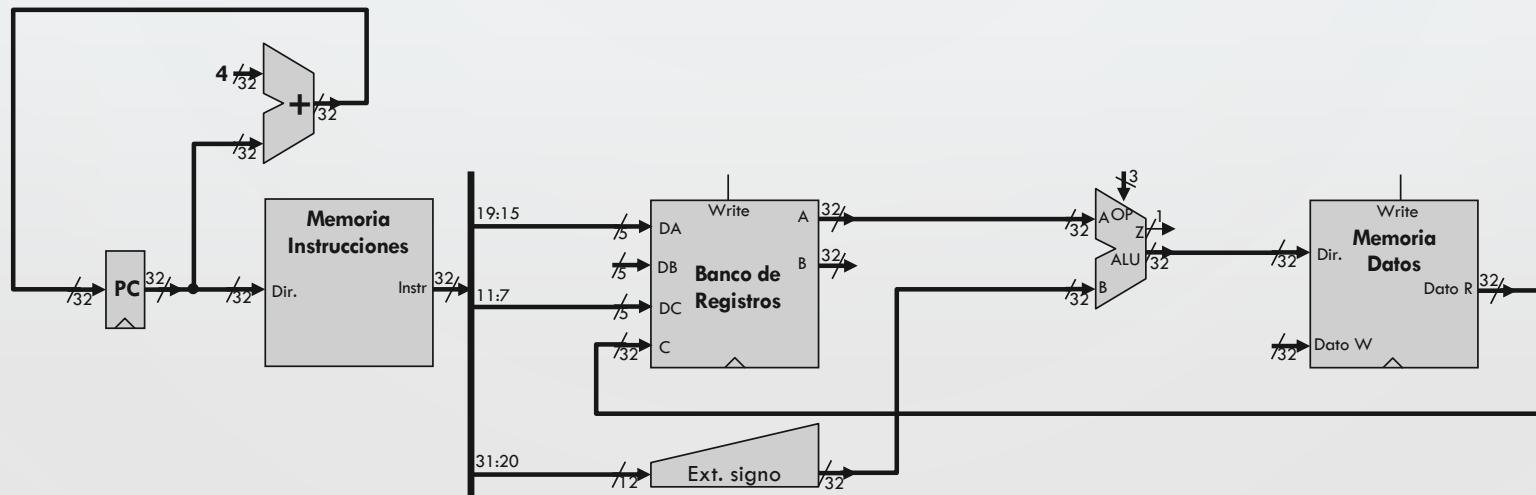
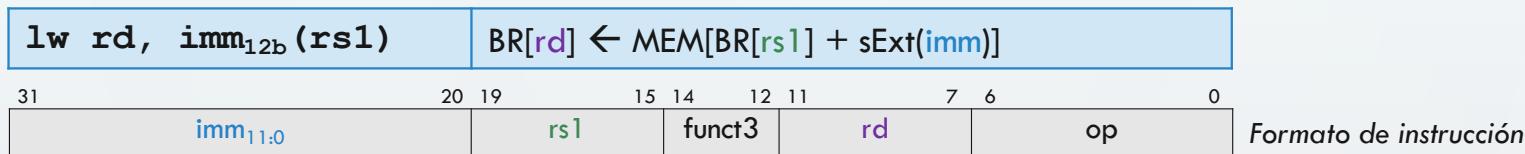
CAMINO DE DATOS

- Ruta de datos para la instrucción LW (Tipo I)



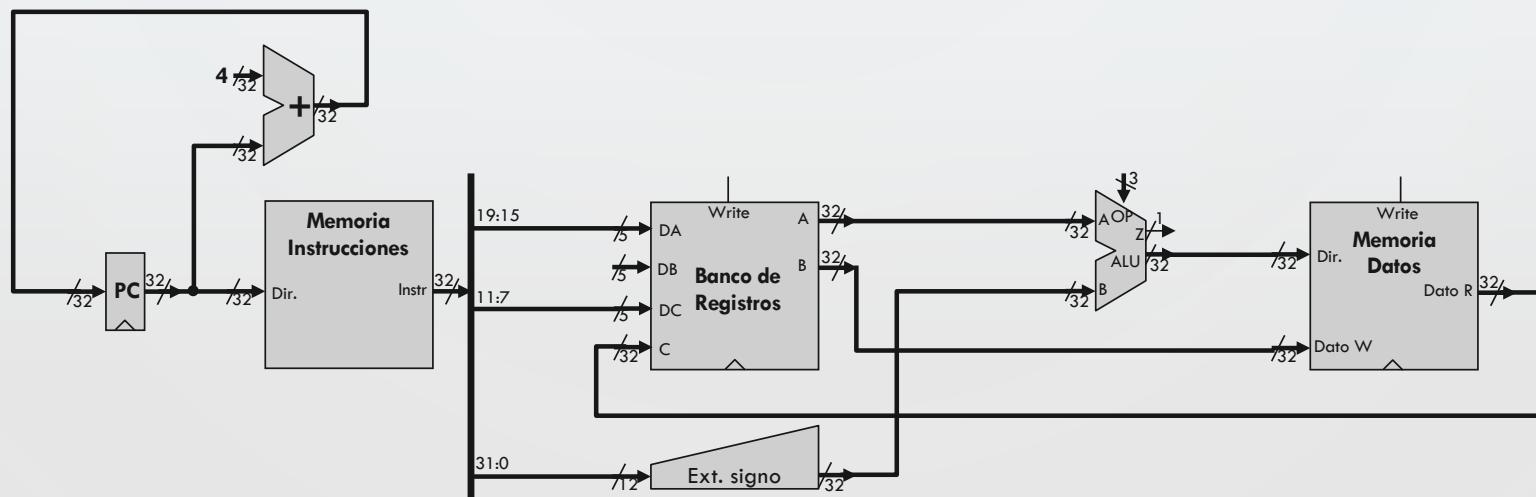
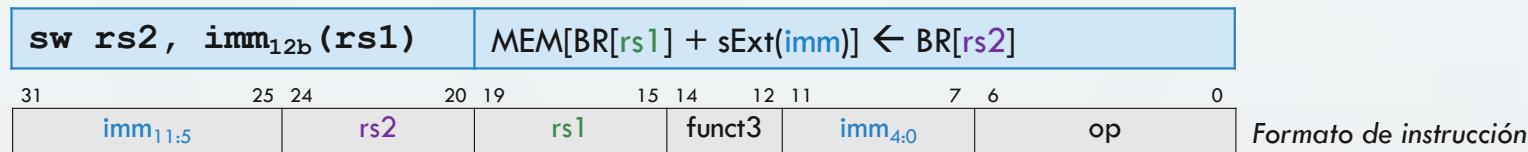
CAMINO DE DATOS

- Ruta de datos para la instrucción LW (Tipo I)



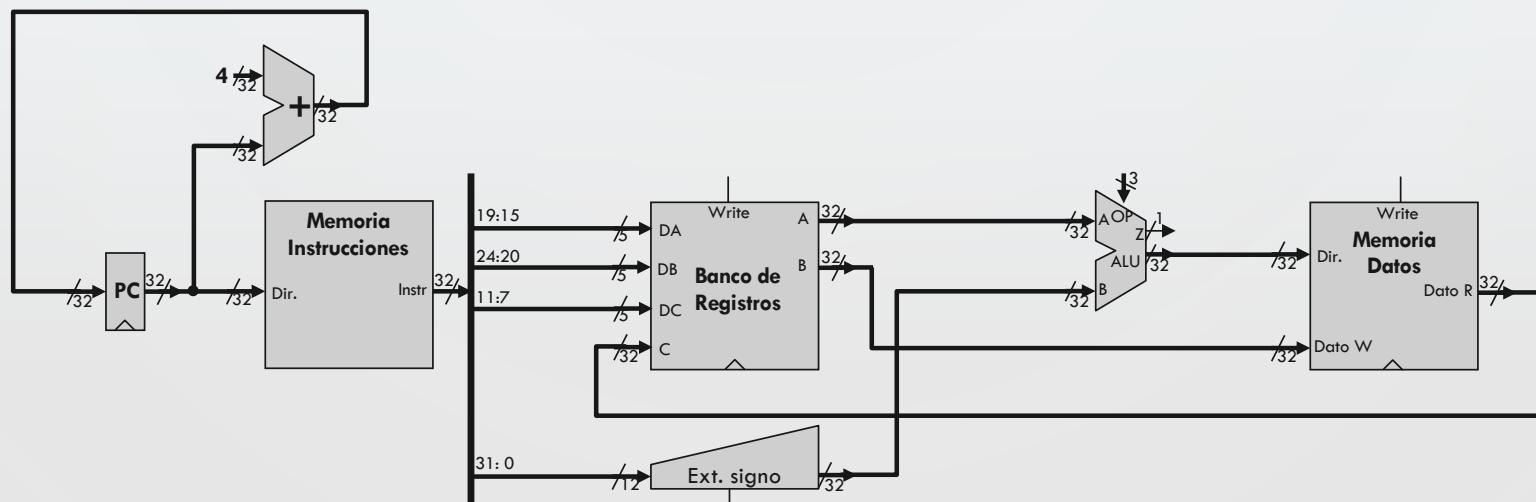
CAMINO DE DATOS

- Ruta de datos para la instrucción SW / LW (Tipo S)



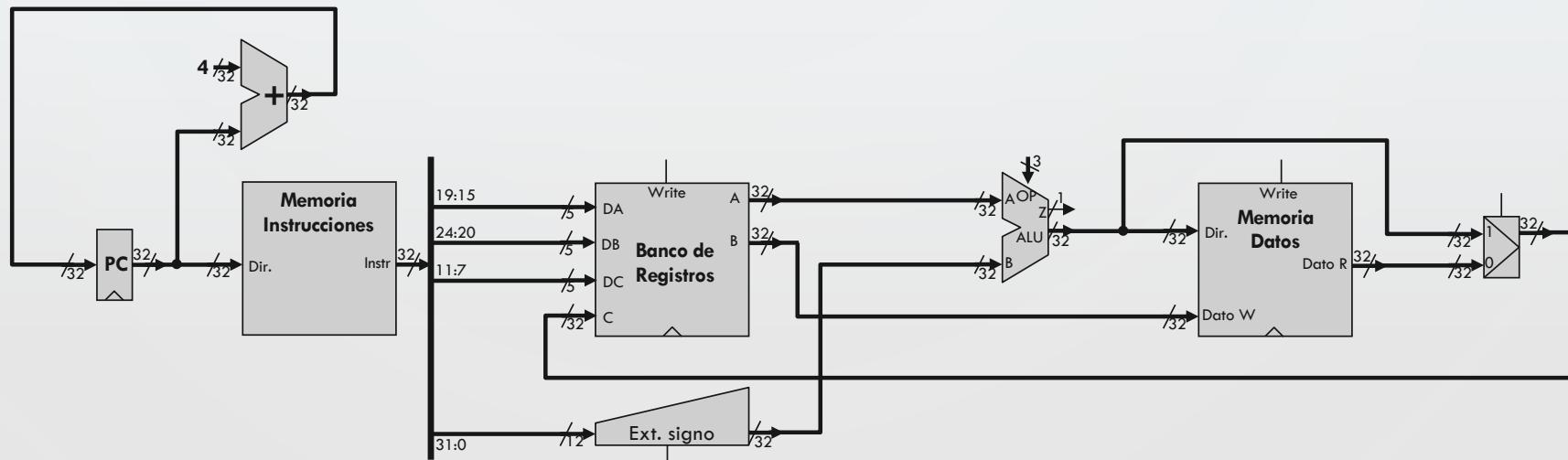
CAMINO DE DATOS

- Ruta de datos para la instrucción SW/LW (Tipo I y Tipo S)



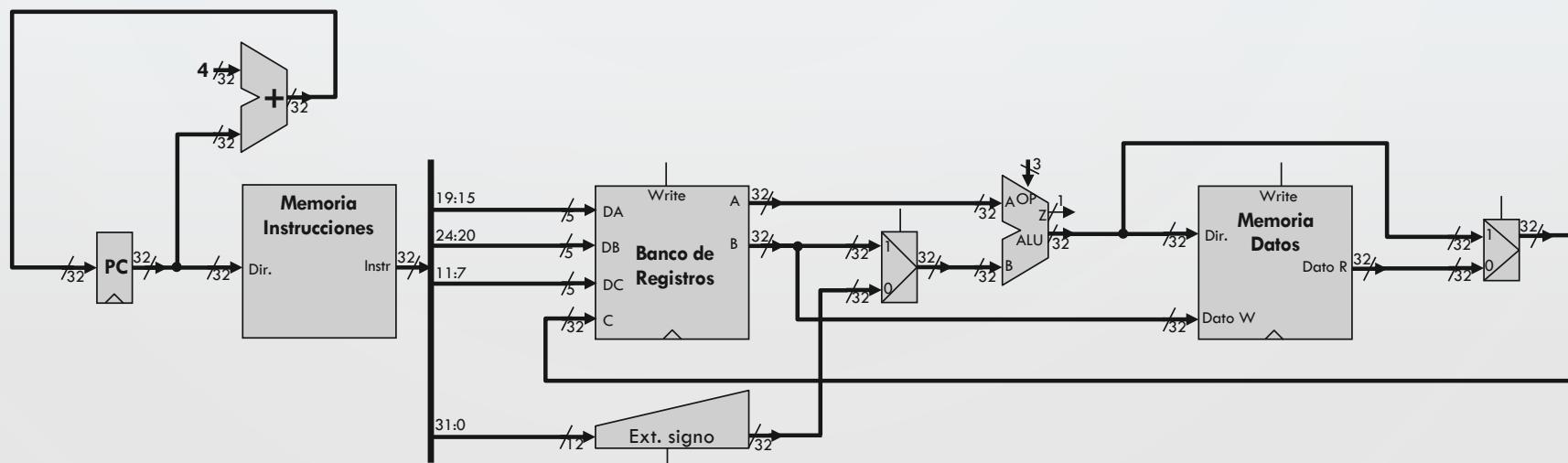
CAMINO DE DATOS

- Ruta de datos para la instrucción ALUI/SW/LW



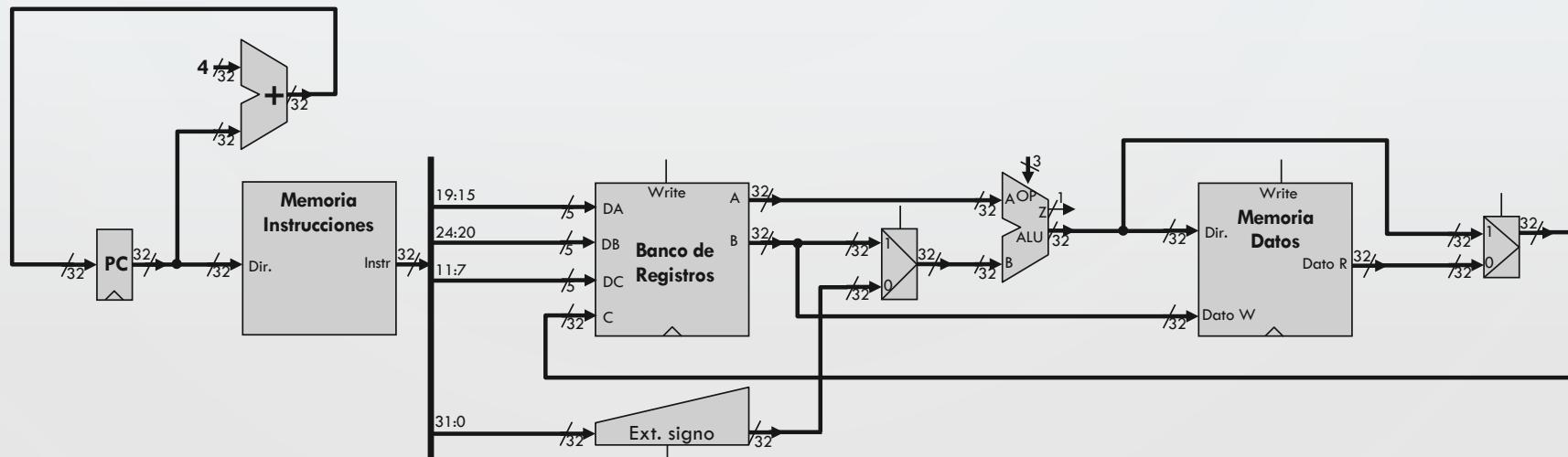
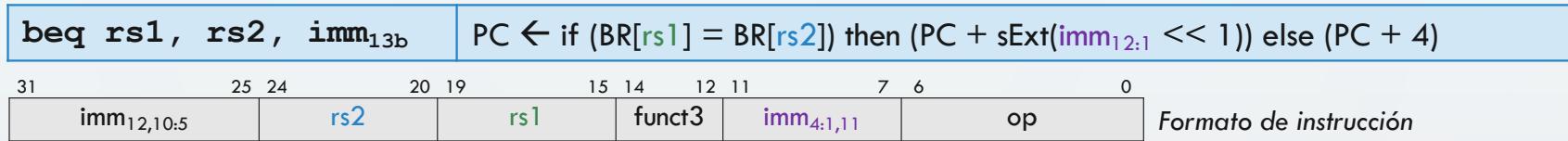
CAMINO DE DATOS

- Ruta de datos para la instrucción ALU/ALUI/SW/LW (Tipo I, R y S)



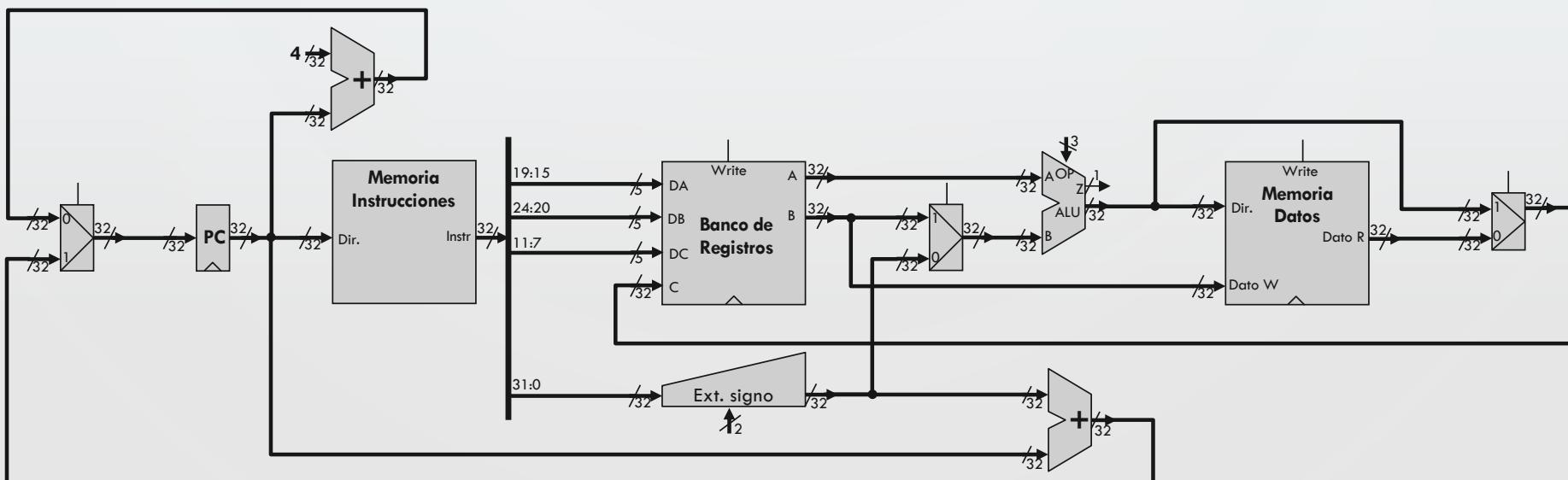
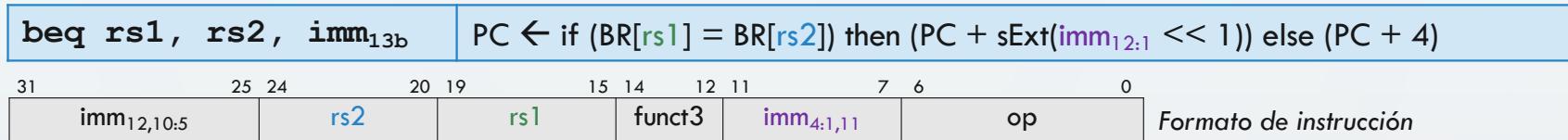
CAMINO DE DATOS

- Ruta de datos para la instrucción BEQ/ALU/ALUI/SW/LW (Tipo B)



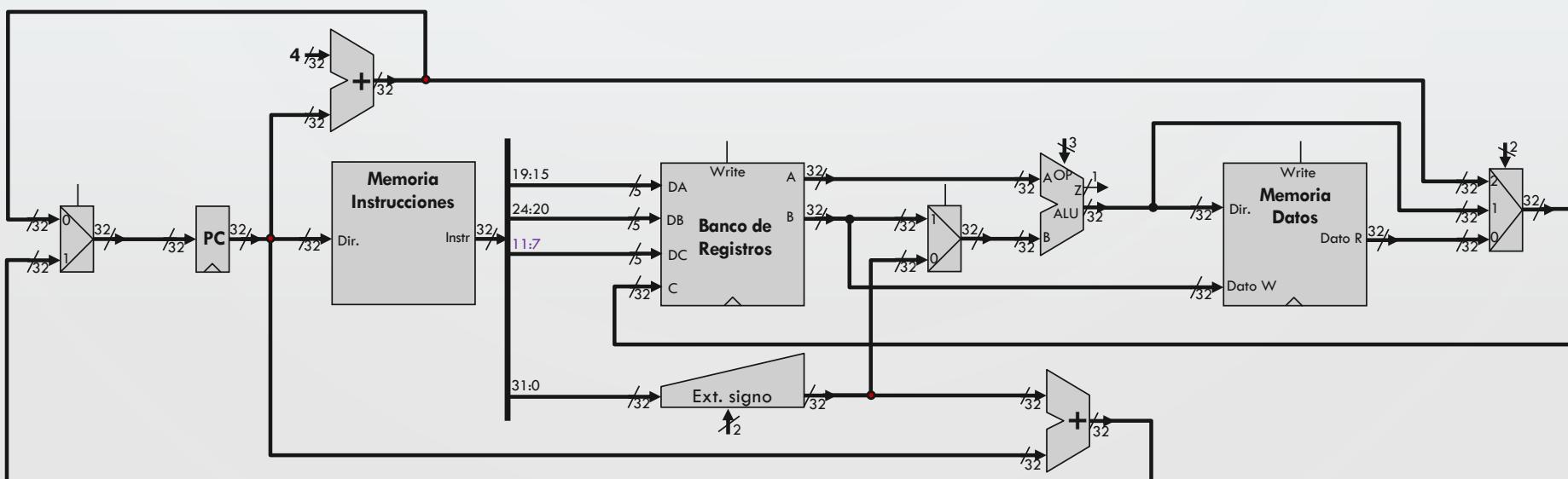
CAMINO DE DATOS

- Ruta de datos para la instrucción BEQ/ALU/ALUI/SW/LW (Tipos I,R,S,B)



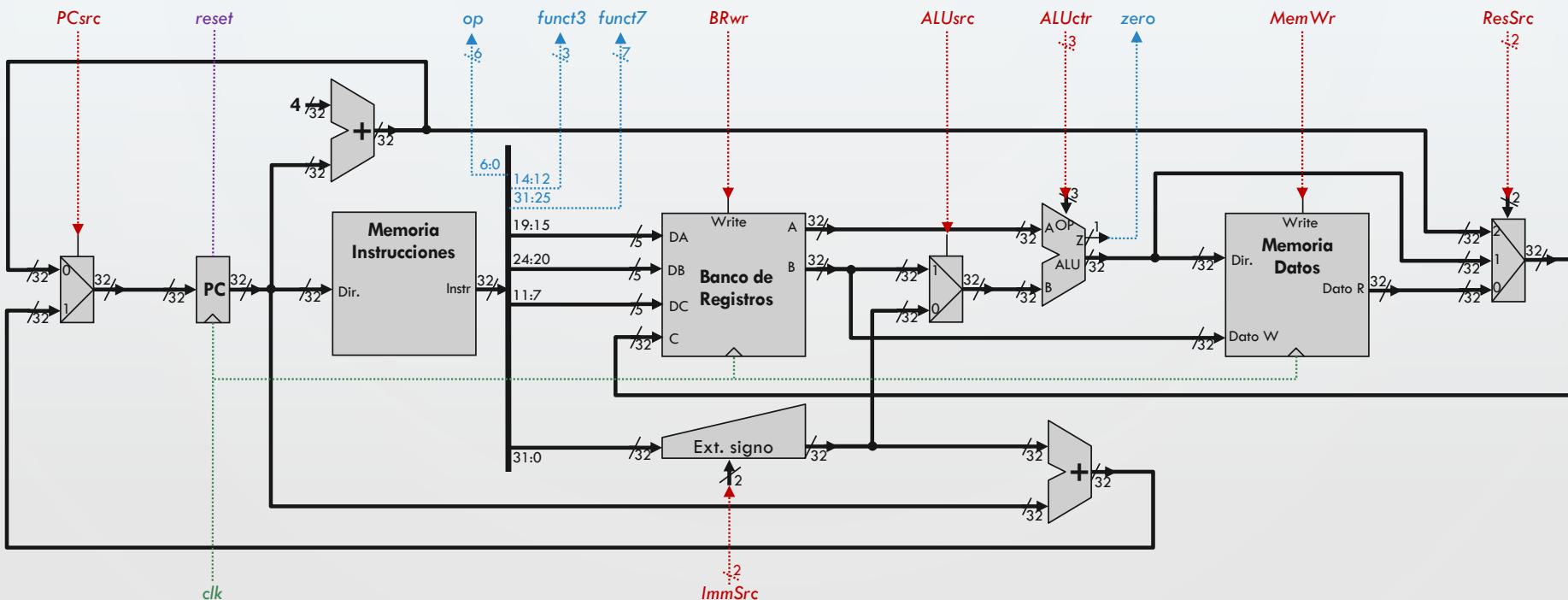
CAMINO DE DATOS

- Ruta de datos para la instrucción JAL/BEQ/ALU/ALUI/SW/LW (Tipo J)



CAMINO DE DATOS

- Señales de control, señales de estado, reloj y reset

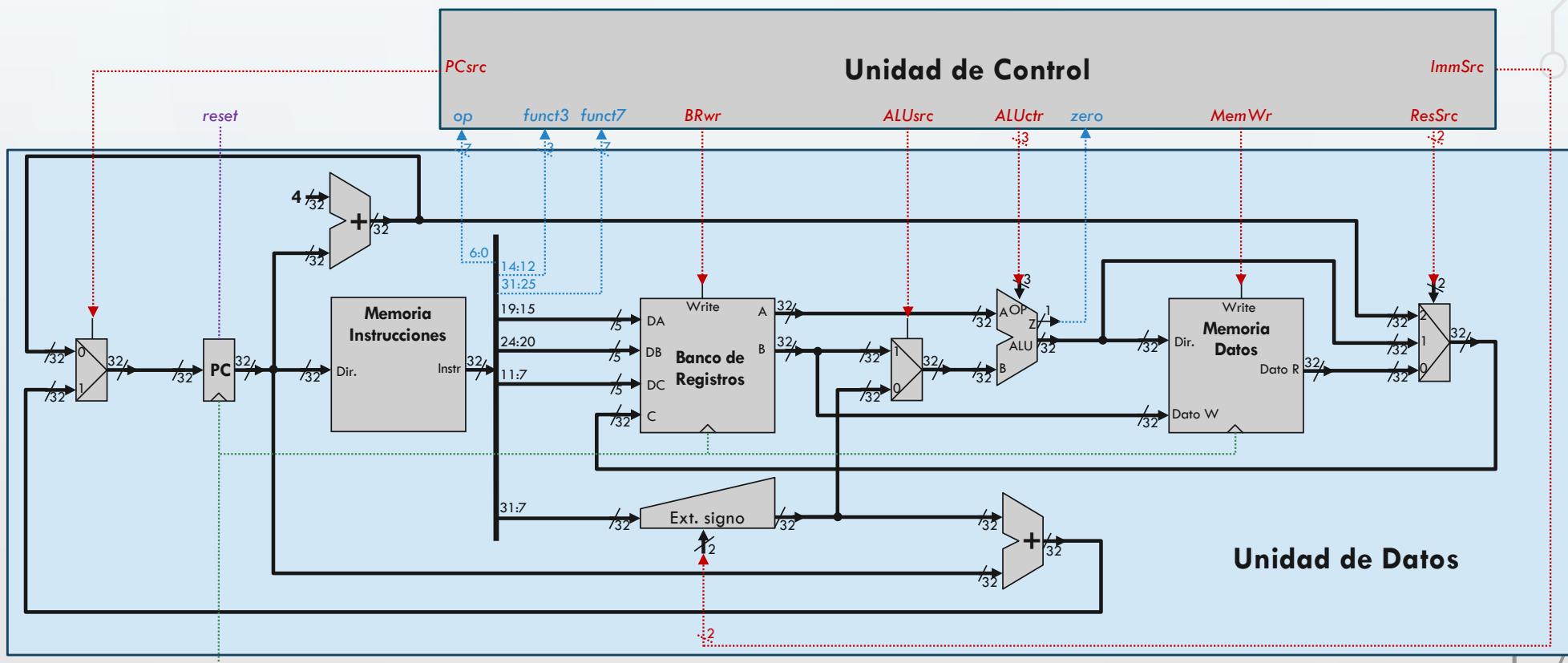


UNIDAD DE CONTROL

- Circuito combinacional que genera el valor de las señales de control en función de la instrucción actual y el estado de la unidad de datos
 - La instrucción y estado de la unidad de datos lo determina a partir de las señales de estado (`op`, `funct3`, `funct7`, `zero`)
 - Implementa lógica combinacional para cada una de las señales de control (`PCsrc`, `BRwr`, `ALUsrc`, `ALUctr`, `MemWr`, `ResSrc`, `ImmSrc`)

Un circuito combinacional es aquel que sus salidas en un momento determinado solo dependen de las entradas en dicho momento (no mantiene estado interno, no contiene elementos de memoria)

UNIDAD DE CONTROL



UNIDAD DE CONTROL EXTENSOR

- Realiza la extensión de signo del operando inmediato
 - 0111 (+7) → 00000111 (+7)
 - 1010 (-6) → 11111010 (-6)
- El tamaño del operando inmediato y su posición en la instrucción dependen del formato

31	25 24	20 19	15 14	12 11	7 6	0	
funct7	rs2	rs1	funct3	rd	op		tipo-R
imm _{11:0}		rs1	funct3	rd	op		tipo-I
imm _{11:5}	rs2	rs1	funct3	imm _{4:0}	op		tipo-S
imm _{12,10:5}	rs2	rs1	funct3	imm _{4:1,11}	op		tipo-B
imm _{20,10:1,11,19:12}				rd	op		tipo-J

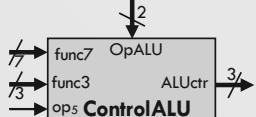
- Con la entrada sel del circuito se selecciona la extensión deseada



sel	Tipo Extensión	Operación
00	Tipo-I	OUT[31:0] = ExtSigno(IN[31:20])
01	Tipo-S	OUT[31:0] = ExtSigno(IN[31:25],IN[11:7])
10	Tipo-B	OUT[31:0] = ExtSigno(IN[31],IB[7],IN30:25],IN[11:8],0)
11	Tipo-J	OUT[31:0] = ExtSigno(IN[31],IN[19:12],IN[20],IN[30:21],0)

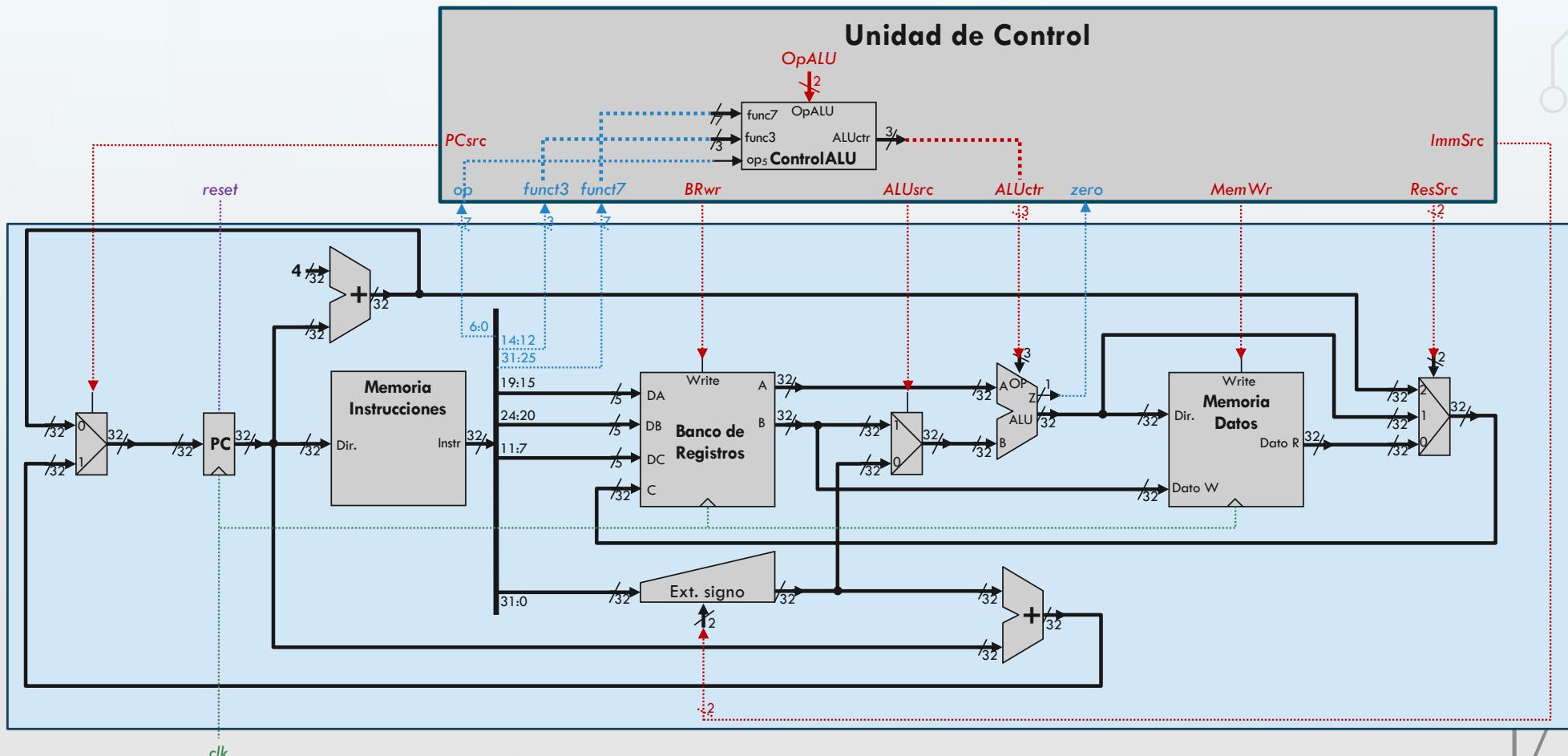
UNIDAD DE CONTROL ALU

- La operación que realiza la ALU (señal **ALUctr**):
 - a veces viene determinada sólo por el código de la instrucción (señal **op**)
 - LW y SW sumar para calcular la dirección de memoria
 - BEQ restar para comparar los registros
 - a veces depende además de la señal **funct3**
 - Las instrucciones ALUI dependen de **funct3** para suma, and, or
 - a veces depende también de la señal **funct7**
 - Las instrucciones ALU dependen de **funct3** y de **funct7** para suma, resta, and, or
- Vamos a usar un **circuito combinacional** que genere la señal **ALUctr** en función de **funct3** y **funct7**



OpALU	funct3	op ₅ , funct7 ₅	ALUctr	Instrucción
00	X	X	000 (ALU suma)	lw, sw
01	X	X	001 (ALU resta)	beq
		000, 01, 10	000 (ALU suma)	add
		000	001 (ALU resta)	sub
10	010	X	101 (ALU activar si menor)	slt
	110	X	011 (ALU or)	or
	111	X	010 (ALU and)	and

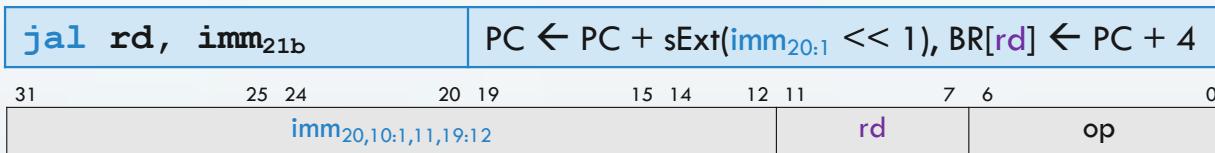
UNIDAD DE CONTROL MONOCICLO



TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR



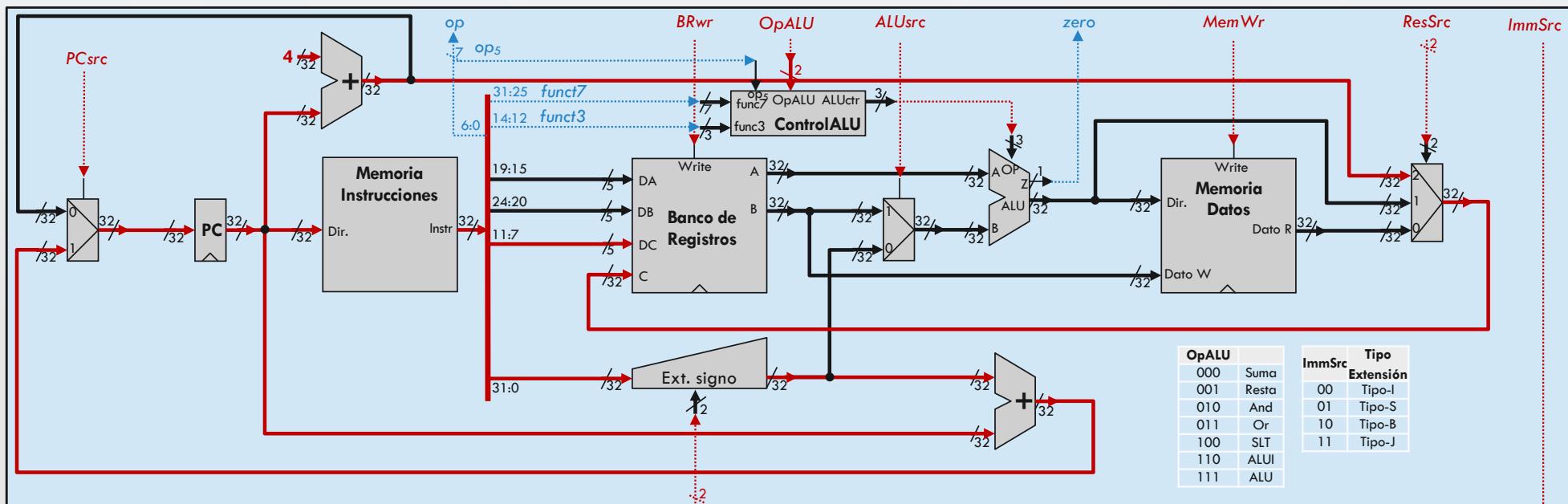
UNIDAD DE CONTROL MONOCICLO



Formato de instrucción Tipo-J

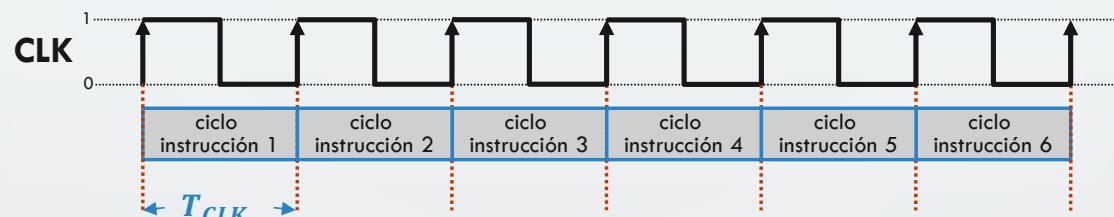
Tabla de activación de las señales de control

Instr.	PCsrc	BRwr	OpALU	ALUsrc	MemWr	ResSrc	ImmSrc
lw	0	1	00	0	0	00	00
sw	0	0	00	0	1	XX	01
ALUI	0	1	10	0	0	01	00
ALU	0	1	10	1	0	01	XX
beq	zero	0	01	1	0	XX	10
jal	1	1	XX	X	0	10	11



TIEMPO DE CICLO

- Ejecución de instrucciones sincronizada por señal de reloj
- Cada instrucción tarda un ciclo de reloj en ejecutarse



- El flanco de reloj carga los elementos de memoria:
 - PC
 - Registros del banco
 - Memoria RAM
- ¿Cuál es la máxima frecuencia a la que puede funcionar el procesador?
 - Mínimo T_{CLK} (período)

TIEMPO DE CICLO

- T_{CLK} tiene que ser suficiente para que de tiempo a realizar todas las transferencias de cada instrucción
- Cada elemento hardware introduce un retardo
 - Necesita un tiempo desde que cambian sus entradas hasta que su salida es estable
- El tiempo de ciclo (T_{CLK}) será el máximo de los caminos críticos de las transferencias entre registros que se realizan en el procesador
 - Camino crítico de una transferencia es el camino de datos de mayor retardo de todos los implicados en dicha transferencia

TIEMPO DE CICLO

- Podemos asumir retardos despreciables para los multiplexores, unidades de control, extensión de signo, acceso al PC, desplazamiento a la izquierda, transmisión por los cables, etc.

	Memoria Instrucciones	Lectura Registros	Operación ALU	Memoria Datos	Escritura Registros
Retardo	200	100	200	200	100

$$\text{ALU: } T = 200 + 100 + 200 + 100 = 600$$

$$\text{load: } T = 200 + 100 + 200 + 200 + 100 = 800$$

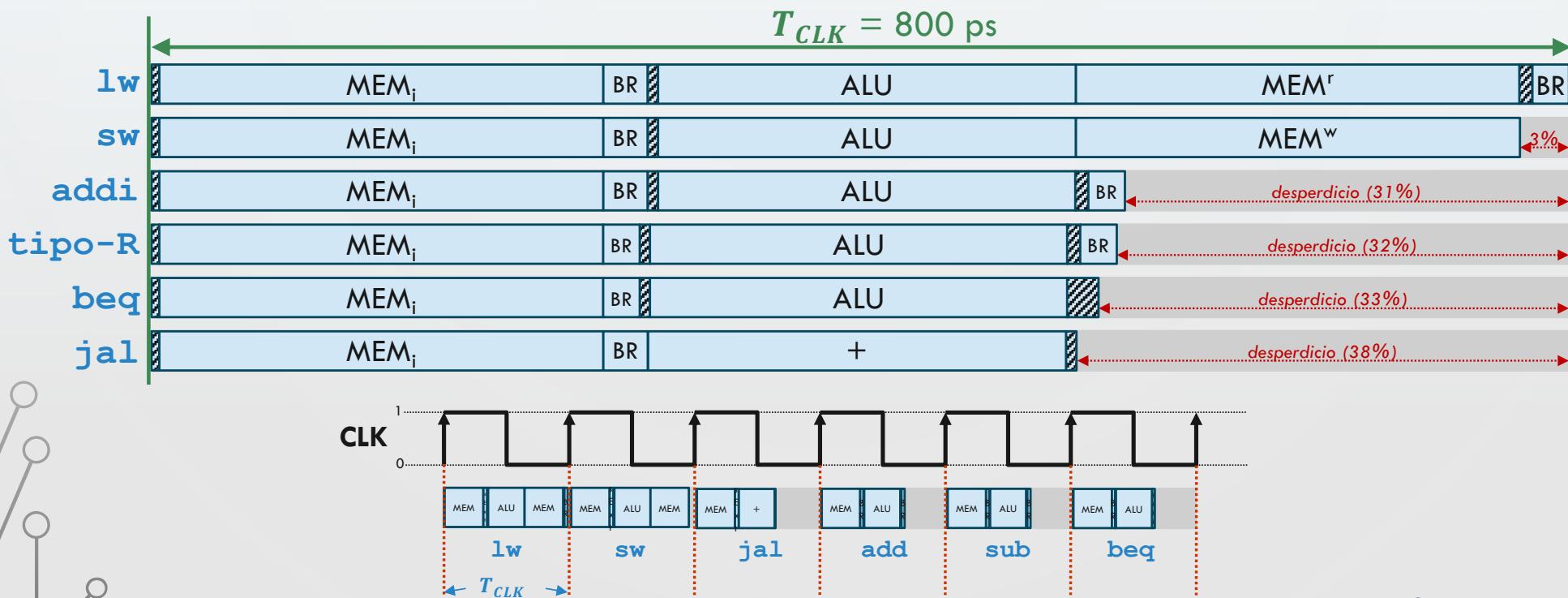
$$\text{store: } T = 200 + 100 + 200 + 200 = 700$$

$$\text{beq: } T = 200 + 100 + 200 = 500$$

$$\text{jump: } T = 200 + 100$$

TIEMPO DE CICLO

- El tiempo de ciclo (T_{CLK}) viene marcado por la instrucción más lenta
- Todas las instrucciones tardan lo mismo → se desperdicia tiempo en las más rápidas



DESVENTAJAS DEL MONOCICLO

- El reloj se ajusta a la instrucción más lenta:
 - Especialmente problemático para instrucciones más complejas como el producto en punto flotante
- Se desperdicia área ya que hay varias instancias de unidades funcionales (en un solo ciclo de reloj no se pueden compartir):
 - 3 sumadores/ALUs
 - memorias separadas para instrucciones y datos
- ¡No se optimiza el caso más frecuente, sino la instrucción más lenta!

EJECUCIÓN DE UN PROGRAMA

- $T_{CPU} = NC \times T_{CLK}$

Monociclo → número ciclos = número de instrucciones

```
        lw x1,desp(x0)
        lw x2,0(x1)
        lw x3,desp2(x0)
        lw x4,desp3(x0)
loop:   beq x3,x4,exit
        lw x4,0(x2)
        sw x4,0(x1)
        addi x2, x2, 4
        addi x1, x1, 4
        j loop
exit: ...
```

} 4 instrucciones
} 6 instrucciones x número iteraciones +
(salida del bucle) 1 instrucción
} **$4 + 6 \times Niter + 1$**

Para Niter = 10

$$T_{CPU} = 65 \text{ ciclos} \times T_{CLK} = 65 \times 800\text{ps} = 52000\text{ps}$$

PROCESADOR MULTICICLO: ¿UN CAMINO DE DATOS MAS EFICIENTE?

- Ciclo de instrucción del procesador multiciclo
- Diseño del camino de datos y de la unidad de control
- Tiempo de ciclo

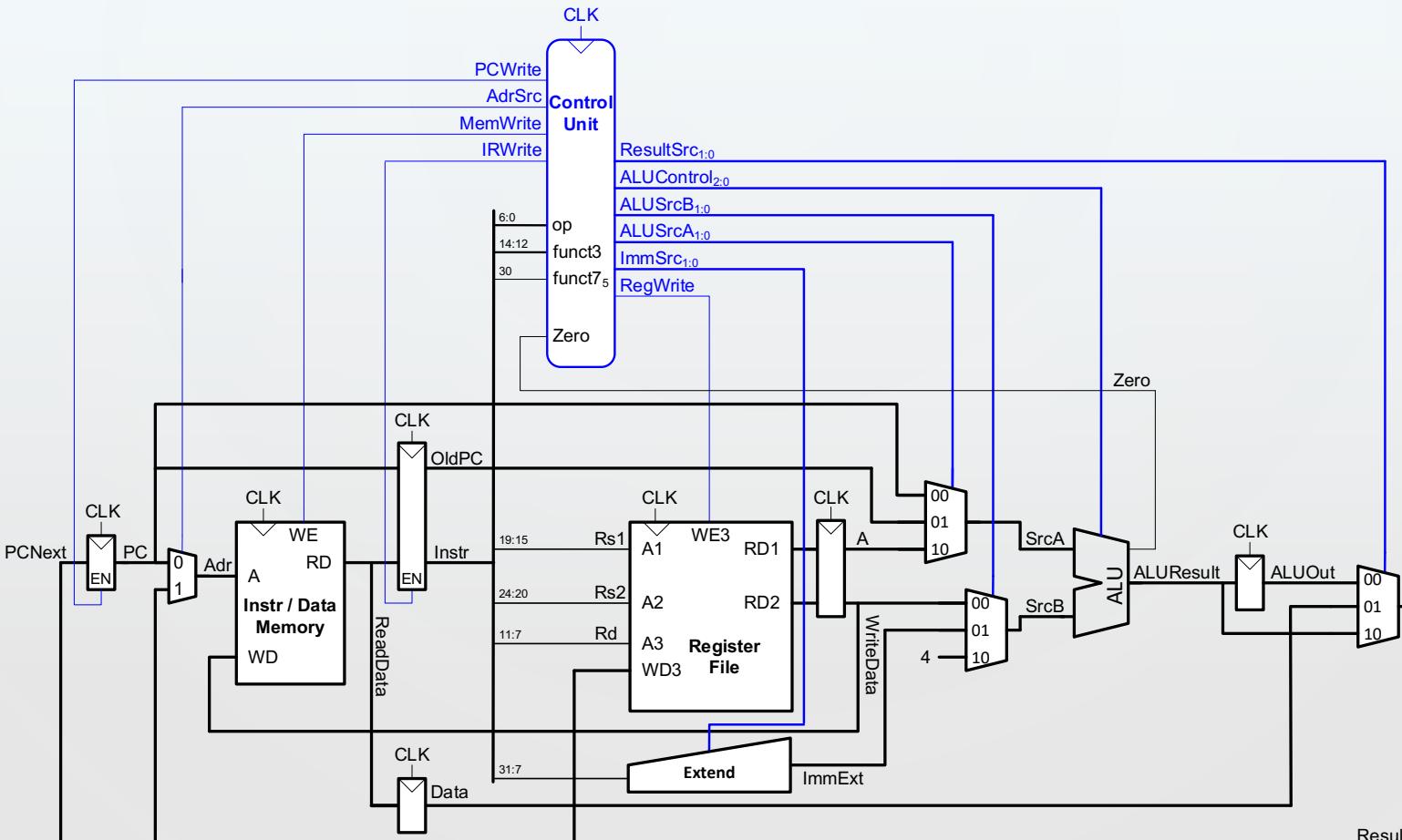
TIEMPO DE CICLO DEL PROCESADOR MULTICICLO

- El número de ciclos empleado para la ejecución dependerá cada instrucción
- El tiempo de ciclo más rápido: ya no depende de la instrucción más lenta, sino de la fase más lenta dentro del ciclo de instrucción: tiempo de ciclo de reloj de 200 ps

	Memoria Instrucciones	Lectura Registros	Operación ALU	Memoria Datos	Escritura Registros
Retardo	200	100	200	200	100

- La unidad de control se complica: ahora será una máquina secuencial que permita controlar los estados/fases asociados a cada ciclo de instrucción
 - Se puede implementar mediante microprogramación
- Se pueden reaprovechar elementos HW: ALU y sumadores, unificar las memorias de instrucciones y datos.
 - Pero tendremos que colocar registros entre etapas para propagar la información de estado

IMPLEMENTACIÓN DEL PROCESADOR MULTICICLO



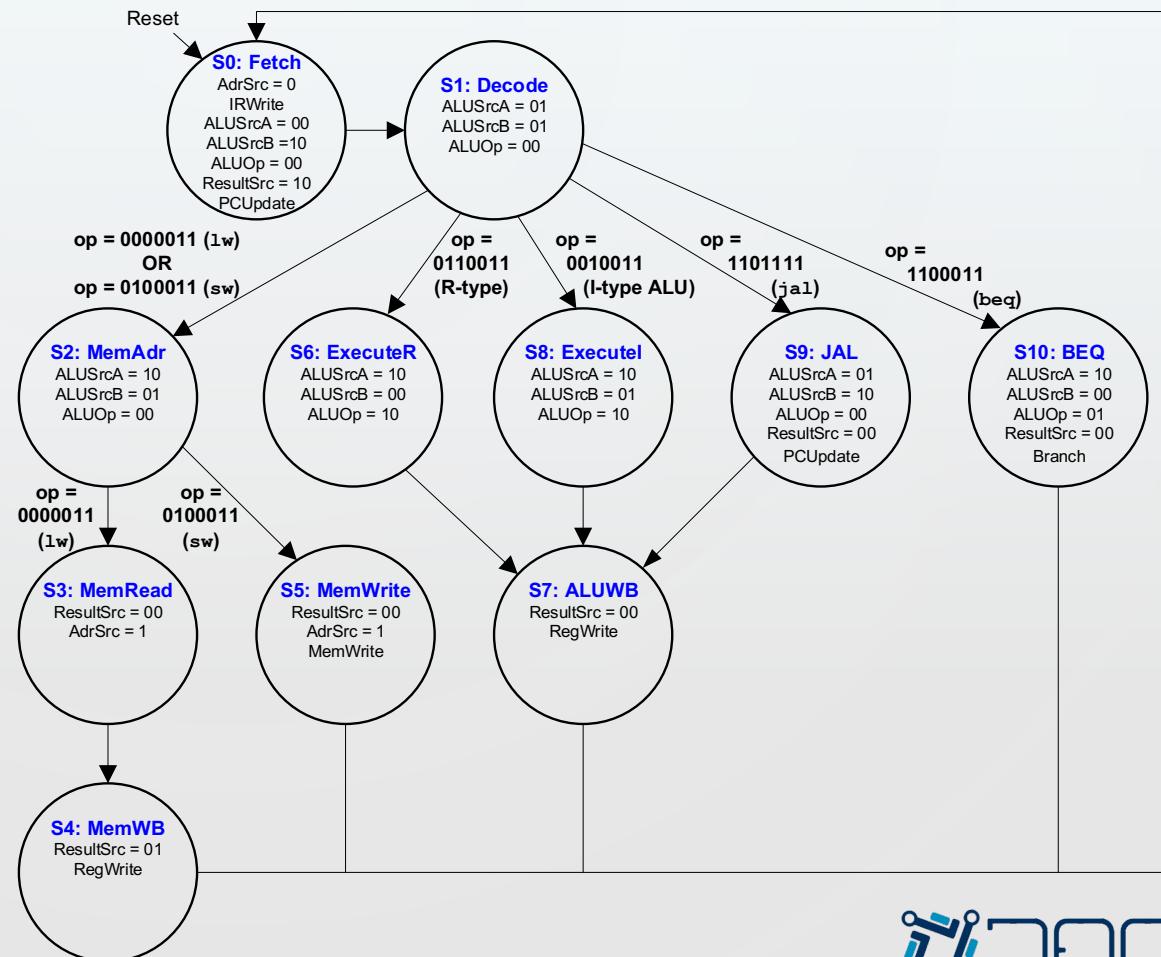
TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR



CICLO DE INSTRUCCIÓN DEL PROCESADOR MULTICICLO

State	Datapath µOp
Fetch	Instr ← Mem[PC]; PC ← PC+4
Decode	ALUOut ← PCTarget
MemAdr	ALUOut ← rs1 + imm
MemRead	Data ← Mem[ALUOut]
MemWB	rd ← Data
MemWrite	Mem[ALUOut] ← rd
ExecuteR	ALUOut ← rs1 op rs2
Executel	ALUOut ← rs1 op imm
ALUWB	rd ← ALUOut
BEQ	ALUResult = rs1-rs2; if Zero, PC = ALUOut
JAL	PC = ALUOut; ALUOut = PC+4

Instrucción	Ciclos
lw	5
sw	4
ALU/ALUI	4
jal	3
beq	3



EJECUCIÓN DE UN PROGRAMA

- $T_{CPU} = NC \times T_{CLK}$

Multiciclo → número ciclos depende del tipo de instrucción

```
lw x1,desp(x0)
lw x2,0(x1)
lw x3,desp2(x0)
lw x4,desp3(x0)
loop: beq x3,x4,exit
      lw x4,0(x2)
      sw x4,0(x1)
      addi x2, x2, 4
      addi x1, x1, 4
      j loop
exit: ...
```

$$\left. \begin{array}{l} \left. \begin{array}{l} \text{4 lw} \\ NC = 4 \times 5 \end{array} \right\} \\ \left. \begin{array}{l} \text{Niter} \times (1 \text{ beq} + 1 \text{ lw} + 1 \text{ sw} + 2 \text{ alui} + 1 \text{ jal}) + 1 \text{ beq} \\ NC = Niter \times (1 \times 3 + 1 \times 5 + 1 \times 4 + 2 \times 4 + 1 \times 3) \\ = Niter \times 23 + 3 \end{array} \right\} \end{array} \right\} 20 + 23 \times Niter + 3$$

Para Niter = 10

$$T_{CPU} = (20+230+3) \text{ ciclos} \times T_{CLK} = 253 \times 200\text{ps} = 50600\text{ps}$$

TIEMPO DE CPU EN EL PROCESADOR MULTICICLO

- El número de ciclos depende del número de instrucciones ejecutadas (NI) y el número medio (efectivo) de ciclos de reloj por instrucción (CPI_{ef})

$$T_{CPU} = NC \times T_{CLK} = NI \times CPI_{ef} \times T_{CLK}$$

- El valor de CPI_{ef} se puede calcular como la media de los valores de CPI individuales ponderados por la frecuencia de ese tipo de instrucción dentro del programa

$$CPI_{ef} = \sum_{i=0}^n \frac{NI_i}{NI} \times CPI_i \quad \text{con } NI = \sum_{i=0}^n NI_i$$

EJECUCIÓN DE UN PROGRAMA II

$$T_{CPU} = NI \times CPI_{ef} \times T_{CLK}$$

```
lw x1,desp(x0)
lw x2,0(x1)
lw x3,desp2(x0)
lw x4,desp3(x0)
loop: beq x3,x4,exit
      lw x4,0(x2)
      sw x4,0(x1)
      addi x2, x2, 4
      addi x1, x1, 4
      j loop
exit: ...
```

Para Niter = 10

$$NI = 4 + 10 \times 6 = 65 \text{ instrucciones}$$

Instrucción	NI_i / NI	CPI_i	$NI_i / NI \times CPI_i$
lw	14/65	5	14/65x5
sw	10/65	4	10/65x4
alu/alui	20/65	4	20/65x4
jal	10/65	3	10/65x3
beq	11/65	3	11/65x3
$CPI_{ef} = \sum_{i=0}^n \frac{NI_i}{NI} \times CPI_i$			= 3,892307

$$T_{CPU} = 65 \times 3,892307 \times 200 = 50600 \text{ ps}$$



OPTIMIZACIÓN DEL RENDIMIENTO

- Para maximizar el rendimiento (performance) hay que minimizar el tiempo de ejecución:

$$Performance_x = 1/T_{CPU_x}$$

- El tiempo de ejecución lo podemos medir ejecutando el programa, o a partir de $T_{CPU} = NI \times CPI_{ef} \times T_{CLK}$
 - El tiempo de ciclo de reloj T_{CLK} lo determina la microarquitectura y la tecnología usada
 - NI podemos conocerlo usando simuladores o contadores hardware y depende del repertorio de instrucciones, el lenguaje de programación y el algoritmo
 - El CPI_{ef} depende del tipo de instrucciones usados y de la implementación ISA

COMPARACIÓN DE RENDIMIENTOS

- Para comparar el rendimiento en X con el rendimiento en Y calculamos el **factor de mejora S**, también llamado **speed-up**

$$S = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{T_{CPU_Y}}{T_{CPU_X}}$$

- Si $S > 1$ hay una mejora, y si es < 1 hay un empeoramiento
- En forma porcentual se calcula como $S\% = (S - 1) \times 100$

MONOCICLO VS MULTICICLO

```
lw x1,desp(x0)
lw x2,0(x1)
lw x3,desp2(x0)
lw x4,desp3(x0)
loop: beq x3,x4,exit
lw x4,0(x2)
sw x4,0(x1)
addi x2, x2, 4
addi x1, x1, 4
j loop
exit: ...
```

Para Niter = 10

Monociclo: $T_{CPU_{mono}} = 52000ps$

Multiciclo: $T_{CPU_{multi}} = 50600ps$

$$S = \frac{T_{CPU_{mono}}}{T_{CPU_{multi}}} = \frac{52000ps}{50600ps} = 1,03$$

En forma percentual, la mejora es del 3%

USO DE BENCHMARKS

- Para comparar el rendimiento de forma más fiable se pueden usar conjuntos de programas representativos (**benchmarks**), como por ejemplo los SPEC
- Si hacemos un profiling de esos programas podemos obtener los valores de NI, CPI_{ef} y T_{CLK} para cada una de las arquitecturas
- Estos benchmarks permiten comparar el rendimiento de un procesador con respecto a otro procesador de referencia utilizando medidas como la media geométrica de los factores de mejora del tiempo de ejecución de cada programa (SPECspeed), o la ejecución simultánea de varias instancias del mismo programa (SPECrate)

SPEC CPU BENCHMARK

- Programas que representan una carga de trabajo típica
- Standard Performance Evaluation Corp (SPEC): spec.org
 - Develops benchmarks for CPU, I/O, Web, ...
- SPEC CPU2017: spec.org/cpu
 - Elapsed time de cada uno de los programas: sin I/O, enfocados en la medida del tiempo CPU
 - Reproducibilidad
 - Los tiempos medidos se normalizan respecto a una máquina de referencia
 - La métrica es la media geométrica (GM) de las ratios de mejora para cada programa
 - *SPECspeed2017_int* (integer)
 - *SPECspeed2017_int_base*
 - *SPECspeed2017_int_peak*
 - *SPECspeed2017_fp* (floating-point)
 - *SPECspeed2017_fp_base*
 - *SPECspeed2017_fp_peak*

$$GM = \sqrt[n]{\prod_{i=1}^n \text{SPEC ratio}_i}$$



BENCHMARK SPECINT2000

- En el caso del procesador monociclo, tenemos que $CPI_{ef} = 1$ y $T_{CLK} = 800ps$
- En el caso del procesador multiciclo, el valor de $T_{CLK} = 200ps$ y el valor de NI es el mismo que en el monociclo. Para calcular el valor de CPI_{ef} podemos usar los datos de la tabla:

$$CPI = 0.25 \times 5 + 0.1 \times 4 + 0.11 \times 3 + 0.02 \times 2 + 0.52 \times 4 = 4,1$$

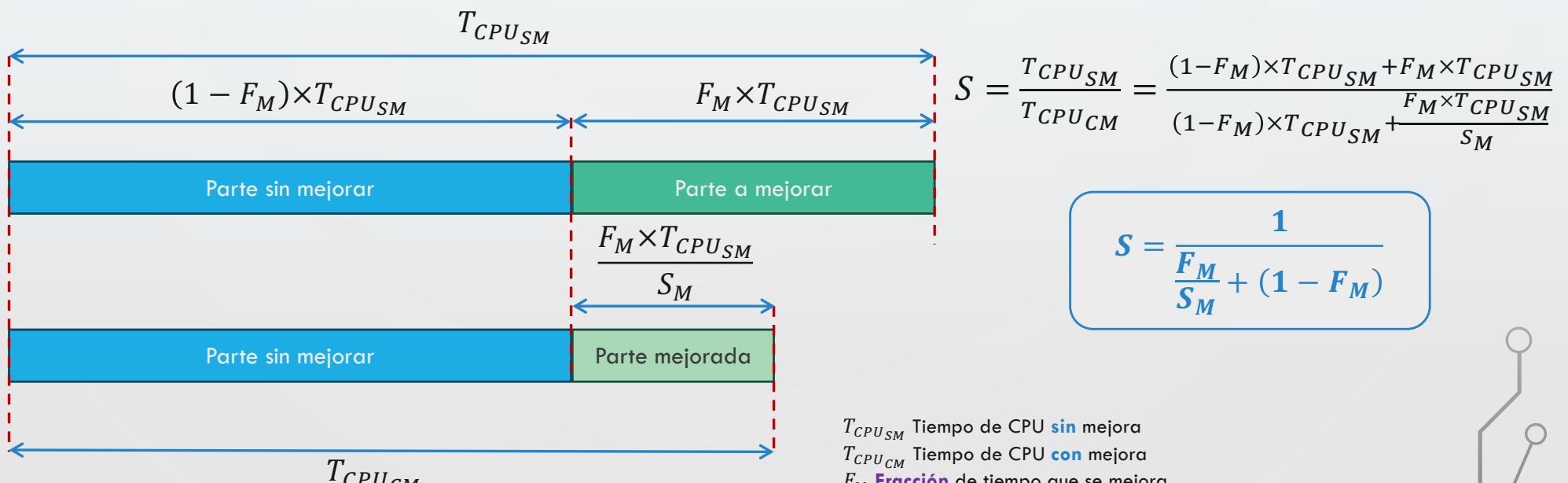
Instrucciones	Frecuencia
Loads	25%
Stores	10%
Branches	11%
Jumps	2%
ALU	52%

- Por tanto, el factor de mejora es:

$$S = \frac{T_{CPU\,mono}}{T_{CPU\,multi}} = \frac{NI \times 1 \times 800}{NI \times 4,1 \times 200} = 0,975 \rightarrow \text{¡No hay mejora!}$$

LEY DE AMDAHL

- Supongamos que somos capaces de mejorar los loads para que en lugar de 5 ciclos tarden sólo 4, ¿hay alguna forma de expresar la mejora total directamente?
- La ley de Amdahl permite calcular el factor de mejora total cuando se mejora sólo una parte del total



EJEMPLO DE LEY DE AMDAHL

- Supongamos que somos capaces de mejorar los loads para que en lugar de 5 ciclos tarden sólo 4, ¿hay alguna forma de expresar la mejora total directamente?
- La ley de Amdahl permite calcular el factor de mejora total cuando se mejora sólo una parte del total
- Los loads se mejoran de 5 a 4 ciclos, así que $S_M = \frac{5}{4} = 1,25$
- Para calcular la fracción de tiempo necesitamos saber cuantos ciclos del programa se dedican a loads:

$$F_M = \frac{NC_{loads}}{NC} = \frac{NI_{loads} \times 5}{NI \times 4,1} = \frac{0,25 \times NI \times 5}{NI \times 4,1} = 0,3048$$

- Aplicando la ley de Amdahl tenemos que el factor de mejora global será de

$$S = \frac{1}{\frac{F_M}{S_M} + (1 - F_M)} = \frac{1}{\frac{0,3048}{1,25} + (1 - 0,3048)} = 1,065$$

ASPECTOS IMPORTANTES DE LA LEY DE AMDAHL

- Tanto S como S_M son **factores de mejora** y se obtienen dividiendo el tiempo que se tardaba antes entre el tiempo que se tarda ahora
 - S es el factor de mejora total así que se dividen los T_{CPU} totales
 - S_M es el factor de mejora de la parte que se mejora, así que se dividen los T_{CPU} de esa parte antes y después de la mejora
 - S_M será mayor o igual que S , y ambos son números positivos
 - Si un factor de mejora es menor que 1, entonces se trata de un empeoramiento
- F_M es una **fracción de tiempo** y se obtiene dividiendo el tiempo que tarda la mejora entre el tiempo total
 - F_M será un número entre 0 y 1

ASPECTOS IMPORTANTES DE LA LEY DE AMDAHL

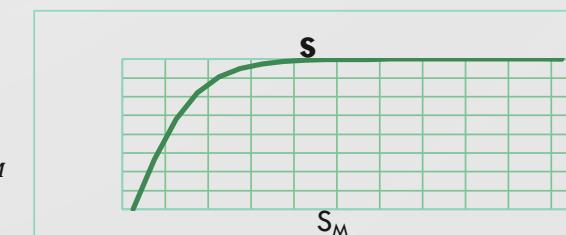
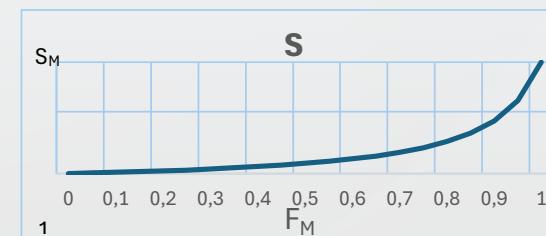
- La posibilidad de mejora está condicionada por el tiempo de ejecución que consume el evento modificado.
 - Corolario 1: hay que **acelerar el caso común**
- El impacto de una mejora en el rendimiento está restringido por la proporción del tiempo de ejecución en la que se utiliza la funcionalidad mejorada
 - Corolario 2: el **cuello de botella** (bottleneck) es el que **limitará la mejora**.
- Casos límite:

- $F_M = 0$. No hay mejora, así que $S = \frac{1}{\frac{0}{S_M} + (1-0)} = 1$

- $F_M = 1$. Todo se mejora, así que $S = \frac{1}{\frac{1}{S_M} + (1-1)} = S_M$

- $S_M = 1$. No hay mejora, así que $S = \frac{1}{\frac{F_M}{1} + (1-F_M)} = 1$

- $S_M = \infty$. Se alcanza la mejora máxima, así que $S = \frac{1}{\frac{F_M}{\infty} + (1-F_M)} = \frac{1}{1-F_M}$



EJEMPLO DE LEY DE AMDAHL (2)

- Un computador ejecuta un programa en 100 segundos, siendo las operaciones de multiplicación responsables del 80% de ese tiempo. ¿Cuánto habría que mejorar la velocidad de la multiplicación si se desea que el programa se ejecute 5 veces más rápido?
 - $F_M = 0.8$
 - $S=5$
- Aplicando la ley de Amdahl:: ???????

$$S = \frac{1}{\frac{F_M}{S_M} + (1 - F_M)} = \frac{1}{\frac{0.8}{S_M} + (0.2)} = 5$$

$$20 = \frac{80}{n} + 20$$

GENERALIZACIÓN DE LA LEY DE AMDAHL

- Si consideramos que el sistema está compuesto por N partes:
 - La parte i es responsable de la fracción F_{M_i} del tiempo total y tiene un factor de mejora S_{M_i}
 - Se cumple que $\sum_{i=1}^N F_{M_i} = 1$
- La ley de Amdahl generalizada viene dada por

$$S = \frac{1}{\sum_{i=1}^N \frac{F_{M_i}}{S_{M_i}}}$$

- Es fácil comprobar que, para dos partes ($F_{M_2} = 1 - F_{M_1}$), si una parte no se mejora ($S_{M_2} = 1$), entonces la ley generalizada se reduce a la ecuación vista antes

$$S = \frac{1}{\frac{F_{M_1}}{S_{M_1}} + \frac{F_{M_2}}{S_{M_2}}} = \frac{1}{\frac{F_{M_1}}{S_{M_1}} + (1 - F_{M_1})}$$

PROCESADOR SEGMENTADO: UN CAMINO DE DATOS MAS EFICIENTE

- Concepto de segmentación
- Ciclo de instrucción del procesador segmentado
- Diseño del camino de datos y de la unidad de control

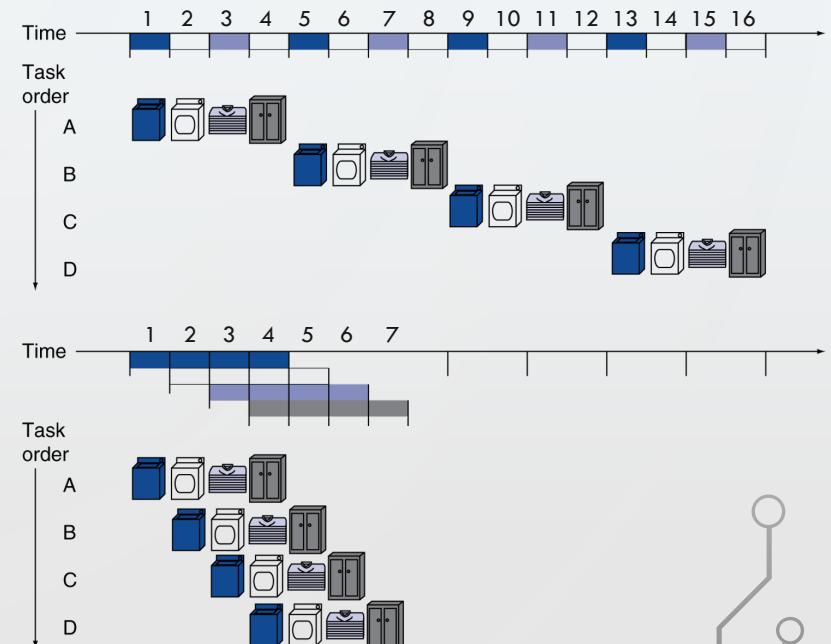
PARALELISMO A NIVEL DE INSTRUCCIÓN

- Una forma de mejorar el rendimiento consiste en ejecutar las instrucciones concurrentemente:
 - Iniciando varias instrucciones al mismo tiempo: **procesador superescalar**
 - Solapando parcialmente la ejecución de varias instrucciones: **procesador segmentado** (pipelining)



PARALELISMO A NIVEL DE INSTRUCCIÓN

- Ejemplo de procesamiento en una lavandería de varias coladas en las que hay que lavar, secar, planchar y guardar ropa
 - Suponemos que cada tarea tiene una duración similar
 - Los recursos de cada tarea están separados: lavadora, secadora, plancha y armario
 - No se puede empezar una tarea de una colada hasta acabar la tarea anterior
- Caso secuencial:
 - Tiempo (n coladas) = $4 \times n \rightarrow$ Tiempo(4) = 16h
 - Throughput (n coladas) = $\frac{n}{\text{Tiempo}(n)}$ → Throughput(4) = $\frac{4}{16} = 0.25 \text{ coladas/h}$
 - Cada electrodoméstico está el 75% del tiempo inactivo
- Caso segmentado:
 - Tiempo (n coladas) = $4 + (n - 1)$ → Tiempo(4) = $4 + 3 = 7\text{h}$
 - Throughput (4) = $\frac{4}{7} = 0.57 \text{ coladas/h}$
 - El throughput tiende a 1 conforme el número de coladas tiende a ∞
 - Throughput($n \rightarrow \infty$) = $\lim_{n \rightarrow \infty} \frac{n}{\text{Tiempo}(n)} = \lim_{n \rightarrow \infty} \frac{n}{4 + (n - 1)} = 1 \text{ colada/h}$
- Speed-up (n trabajos):
 - Speed-up (n) = $\frac{\text{Tiempo secuencial}(n)}{\text{Tiempo segmentado}(n)} = \frac{4 \times n}{4 + (n - 1)}$
 - Speed-up (4) = $\frac{4 \times 4}{4 + (4 - 1)} = \frac{16}{7} = 2.3$
 - Speed-up (non-stop) = $\lim_{n \rightarrow \infty} \frac{4 \times n}{4 + (n - 1)} = 4 = \text{número de etapas}$



ETAPAS DEL CICLO DE INSTRUCCIÓN

IF

Instruction Fetch, lectura de la instrucción de la memoria (MI)

ID

Instruction Decode, Decodificación de la instrucción y lectura de operandos del banco de registros (BR lectura)

EX

EXecute, Cálculo de la dirección de memoria u operación aritmético-lógica(ALU)

MEM

MEMory access, acceso al operando en memoria (MD)

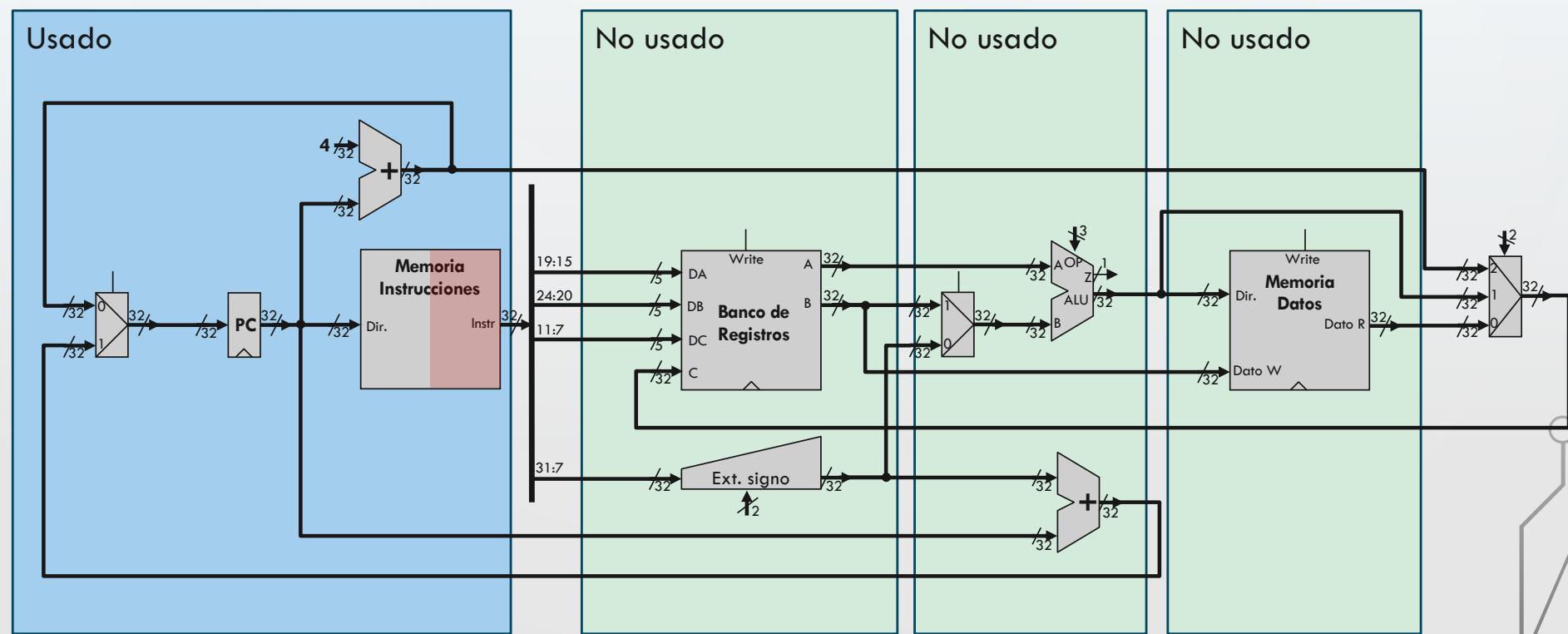
WB

Write Back, Escritura del dato en el registro destino (BR escritura)

PROCESADOR SEGMENTADO

- Etapa IF (Instruction Fetch)

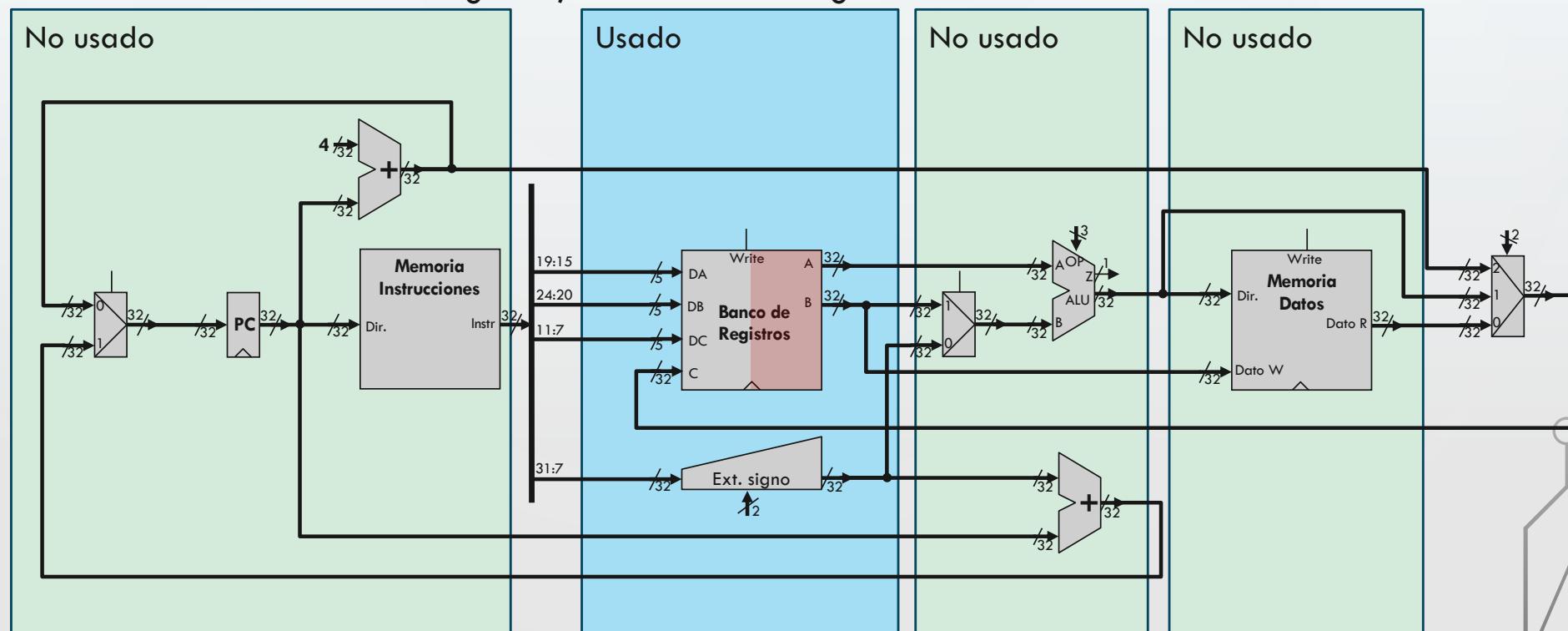
- Lectura de la instrucción de la memoria de instrucciones



PROCESADOR SEGMENTADO

- Etapa ID (Instruction Decode)

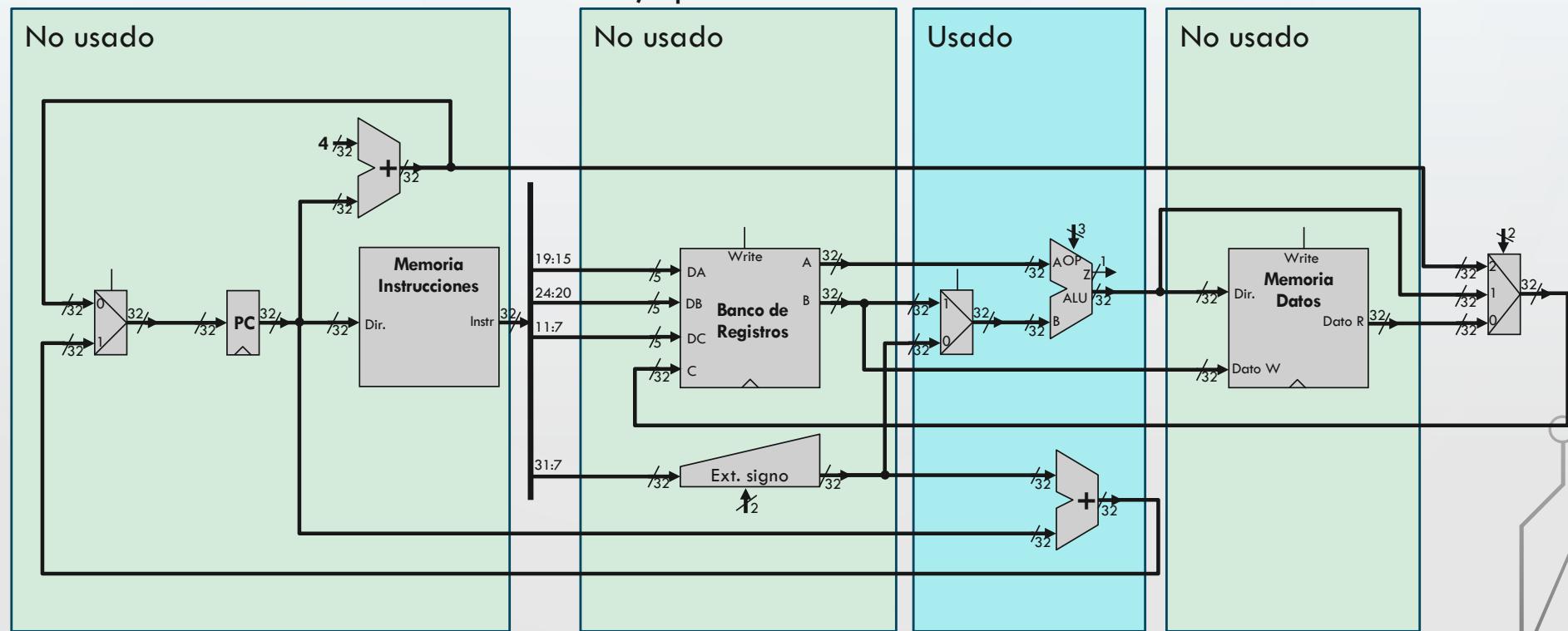
- Lectura del banco de registros, extensiones de signo



PROCESADOR SEGMENTADO

- Etapa EX (Execution)

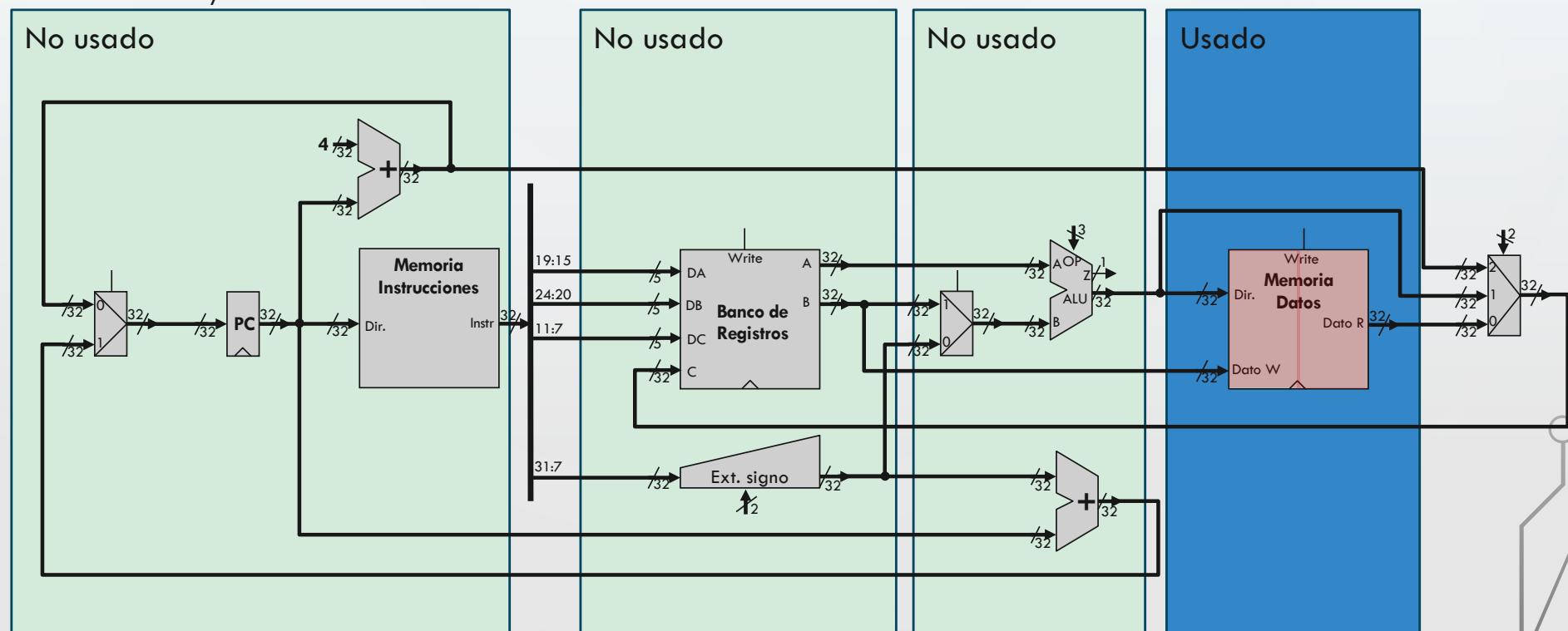
- Cálculo de la dirección de memoria, operación ALU



PROCESADOR SEGMENTADO

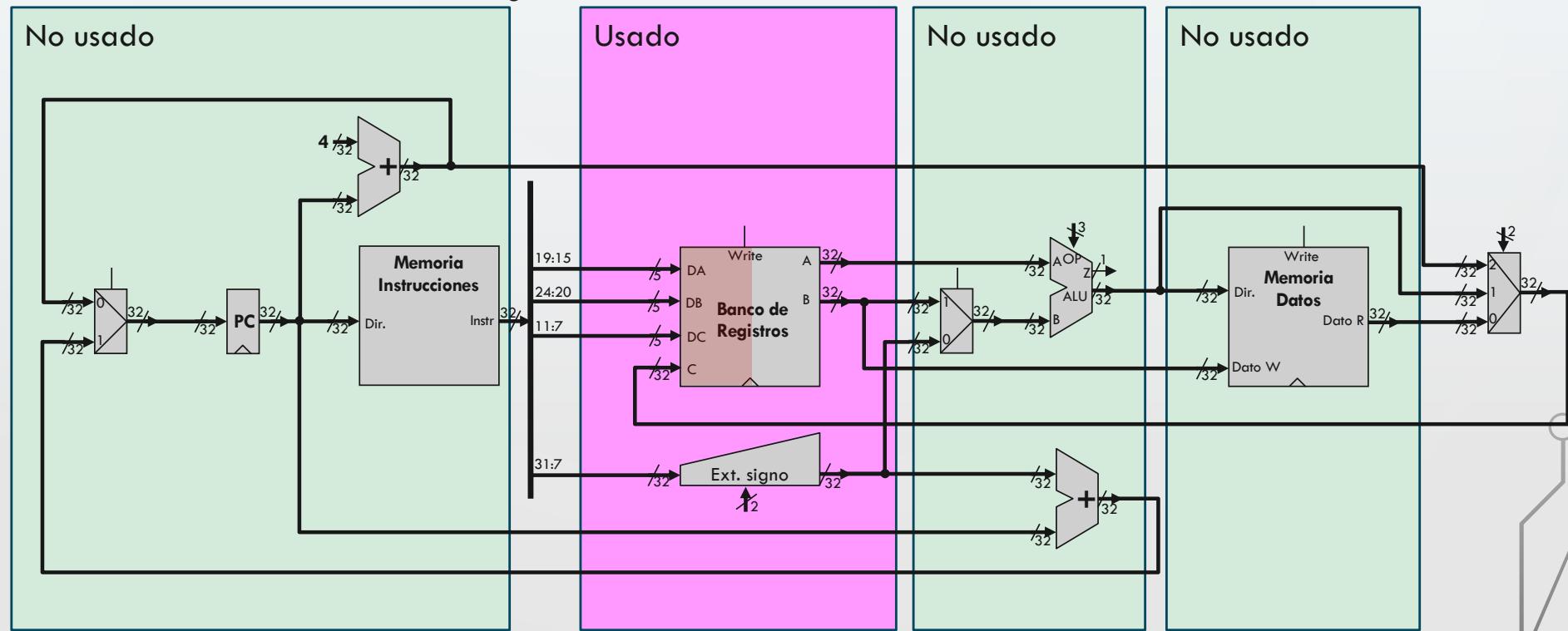
- Etapa MEM (Memory)

- Lectura/escritura en memoria de datos



PROCESADOR SEGMENTADO

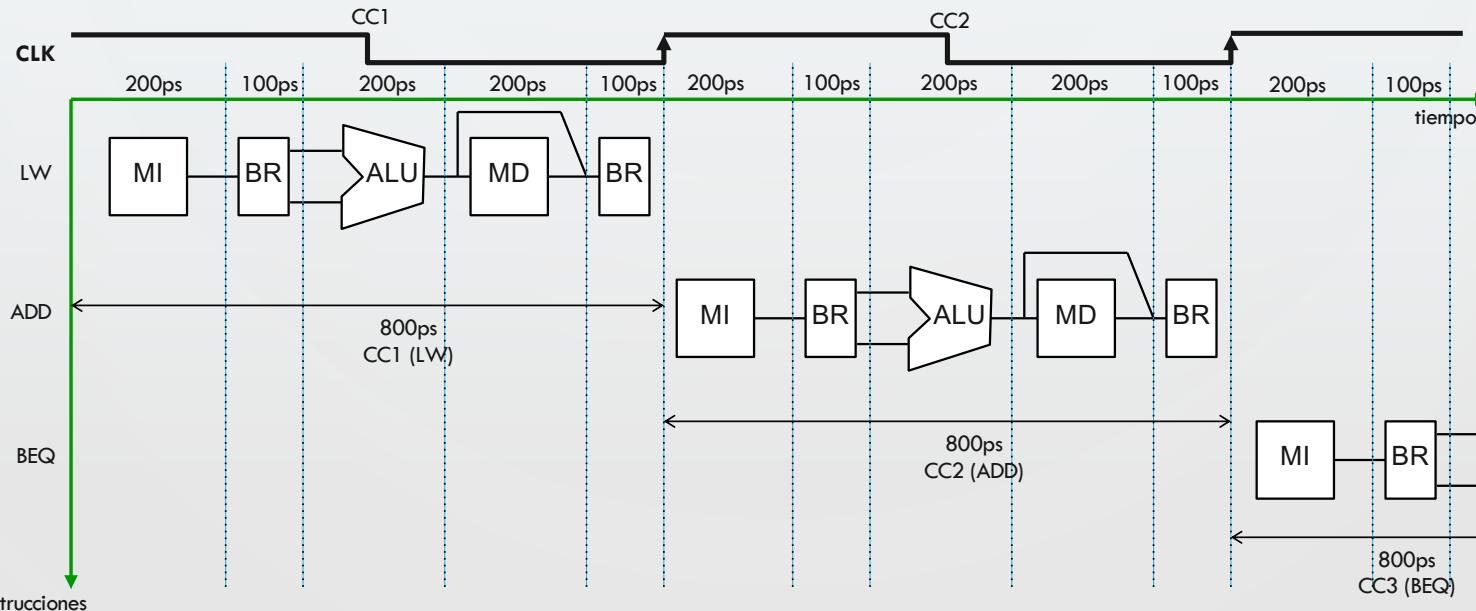
- Etapa WB (Write Back)
- Escritura en el banco de registros



REDUCCIÓN DEL TIEMPO DE EJECUCIÓN

- Procesador **Monociclo**

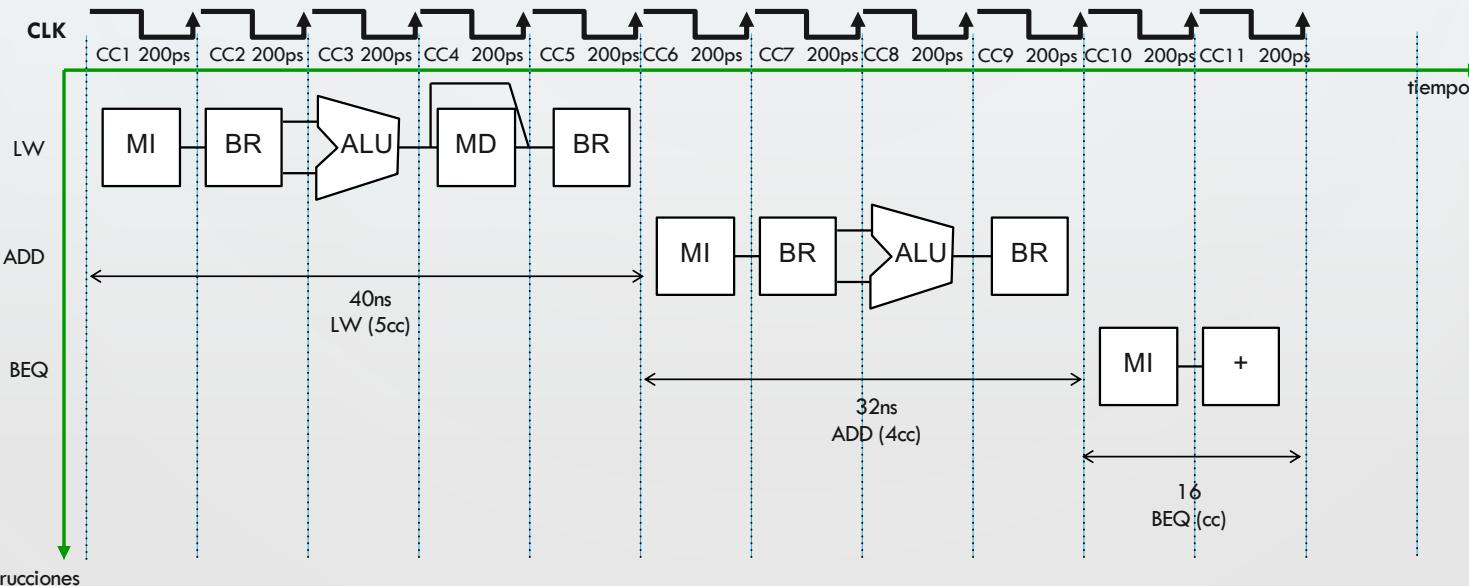
- Todas las instrucciones tardan **un ciclo de reloj**



REDUCCIÓN DEL TIEMPO DE EJECUCIÓN

- Procesador Multiciclo

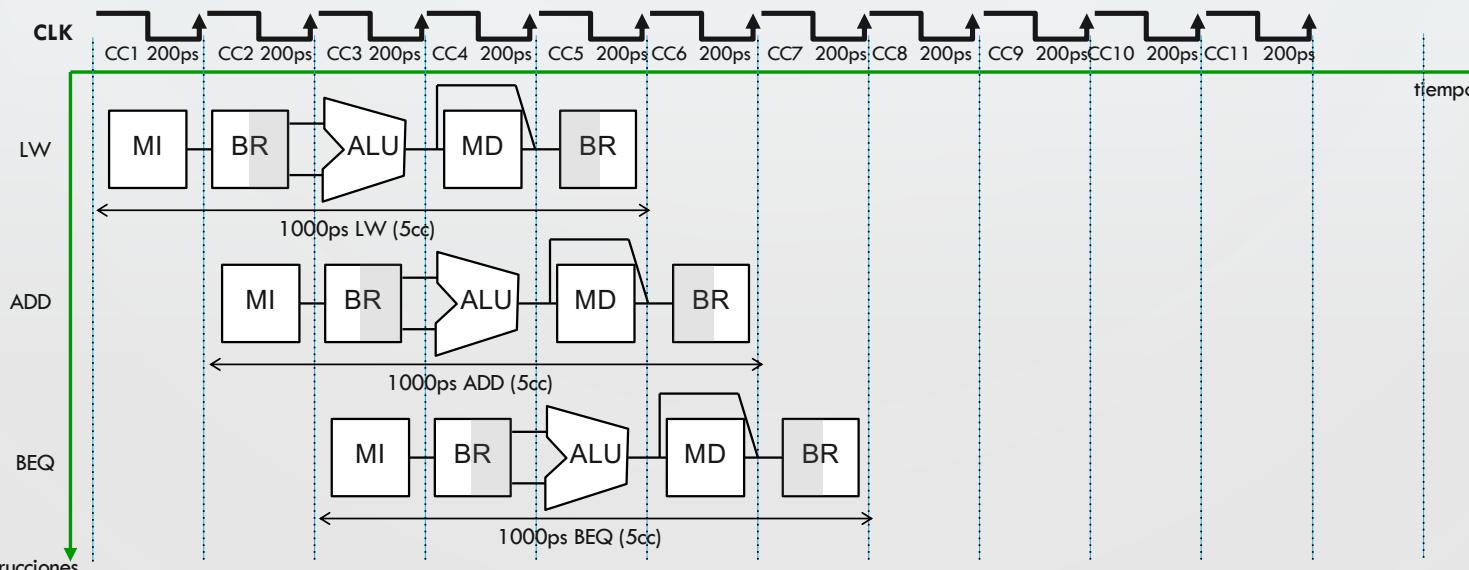
- La ejecución de una instrucción emplea **un número variable ciclos de reloj**



REDUCCIÓN DEL TIEMPO DE EJECUCIÓN

- Procesador Segmentado

- Las instrucciones se dividen en una **serie de etapas**, cada uno utiliza un ciclo
- Se ejecutan en paralelo distintas etapas de instrucciones consecutivas
- Ciclo de reloj → tiempo del camino crítico de la etapa más lenta



BENCHMARK SPECINT2000

- En el caso del procesador segmentado tenemos que $T_{CLK} = 200ps$, $CPI_{ef} \cong 1$ y el valor de NI es el mismo que en el monociclo.
- Por tanto, el factor de mejora es:

$$S = \frac{T_{CPU_{mono}}}{T_{CPU_{seg}}} = \frac{NI \times 1 \times 800}{NI \times 1 \times 200} = 4$$

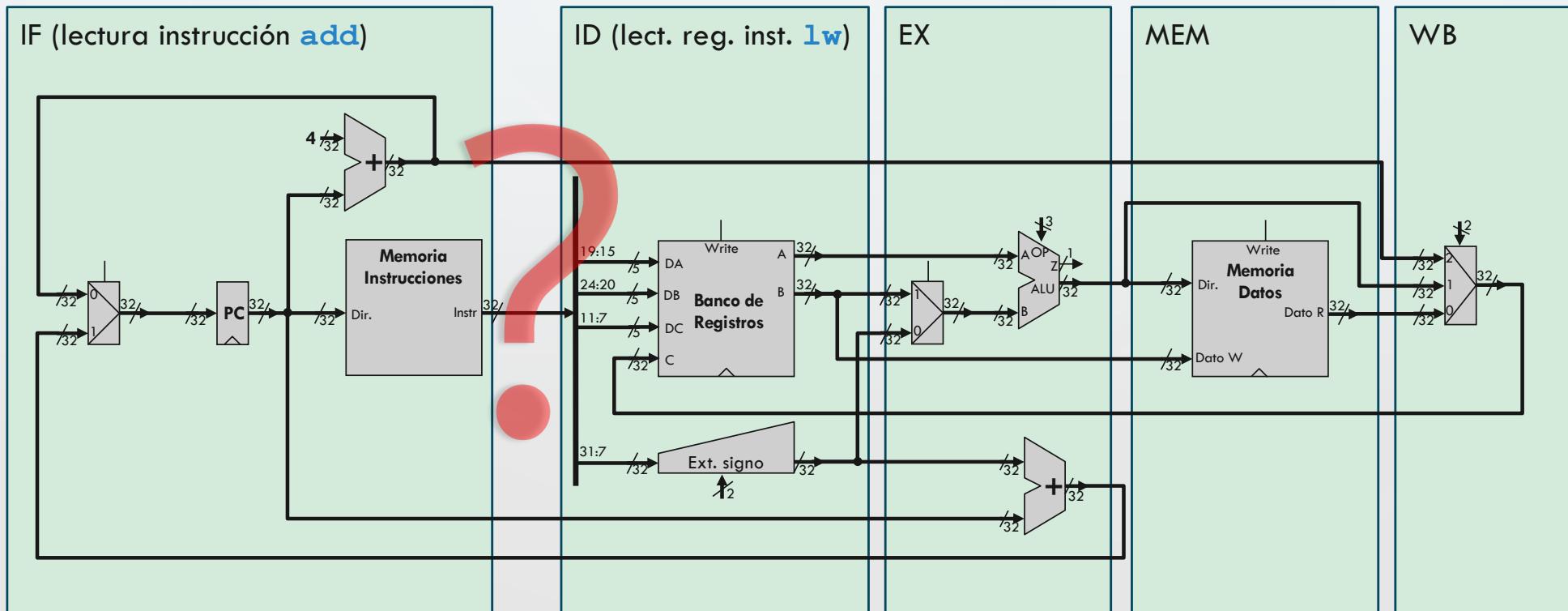
- Si las etapas estuvieran perfectamente equilibradas, entonces el factor de mejora sería igual al número de etapas. Hay otros factores que nos alejan del speed-up ideal:
 - Overhead de la implementación segmentada (introducción de los registros de segmentación)
 - Los riesgos producidos por las dependencias del código pueden hacer que $CPI > 1$
- La aceleración se consigue por el incremento de la productividad (throughput):
 - La latencia (tiempo empleado en la ejecución de una instrucción) no se reduce, puede incluso aumentar.

IMPLEMENTACIÓN DEL PROCESADOR SEGMENTADO

- El diseño de la arquitectura (nivel ISA) hace sencilla la implementación segmentada:
 - Todas las instrucciones son del mismo tamaño (32 bits)
 - Se puede hacer la búsqueda en la 1^a etapa y decodificar en la 2^a
 - Pocos formatos de instrucción y muy homogéneos
 - La lectura del banco de registros puede empezar en la 2^a etapa, en paralelo con la decodificación
 - Sólo se accede a memoria en los loads y stores
 - Se puede utilizar la etapa de ejecución (ALU) para calcular la dirección efectiva de memoria
 - Cada instrucción escribe como mucho un resultado (i.e. cambia el estado de la máquina) y lo hace en las últimas etapas del cauce (MEM o WB)
 - Los operandos deben estar alineados en memoria de forma que la transferencia de un dato requiera un solo acceso a memoria

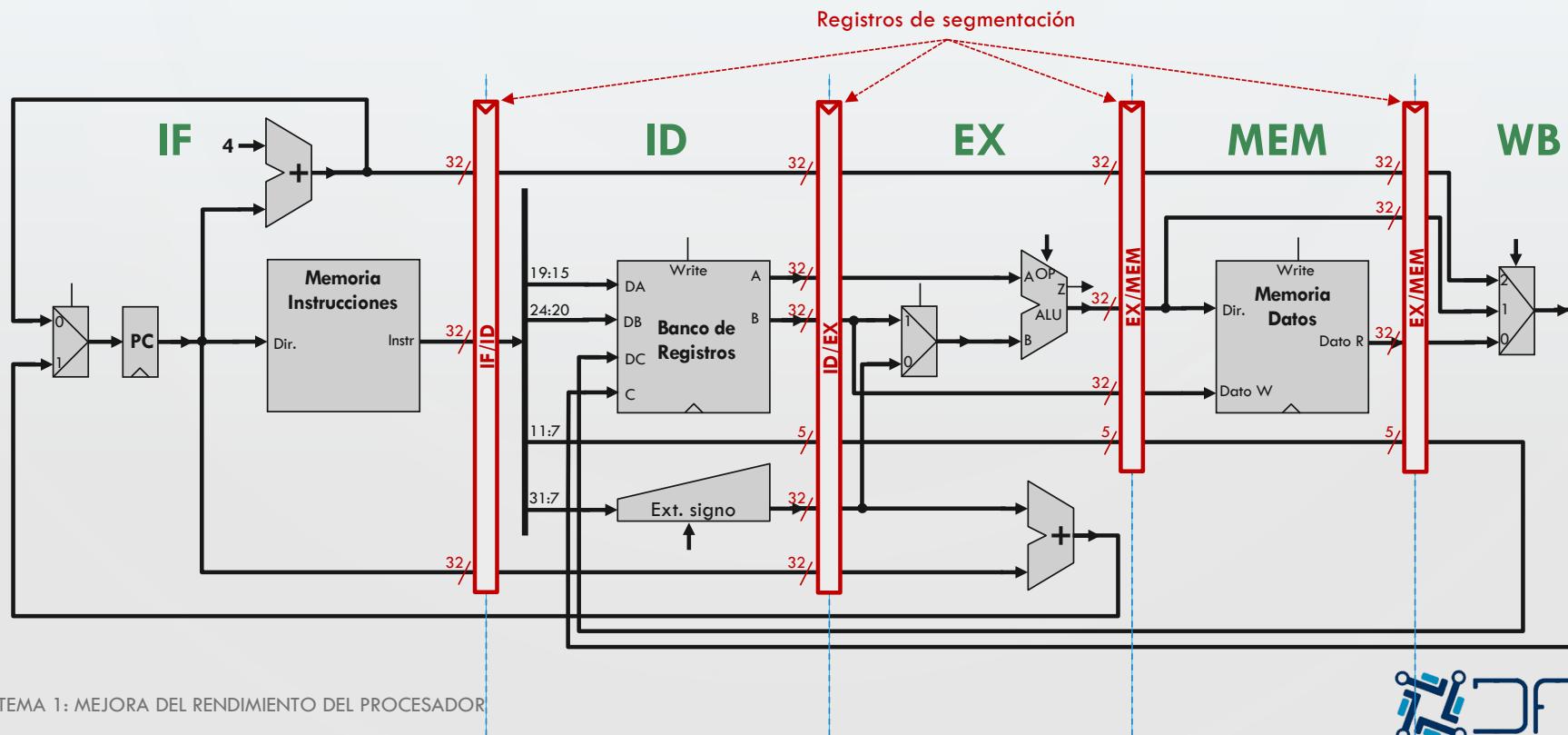
DISEÑO DEL CAMINO DATOS

- ¿Cómo ejecutar varias instrucciones a la vez en las distintas etapas? ¿Cómo leer la instrucción **add** de memoria (IF) y a la vez leer del banco de registros (ID) lo indicado en una **lw** anterior?



DISEÑO DEL CAMINO DATOS

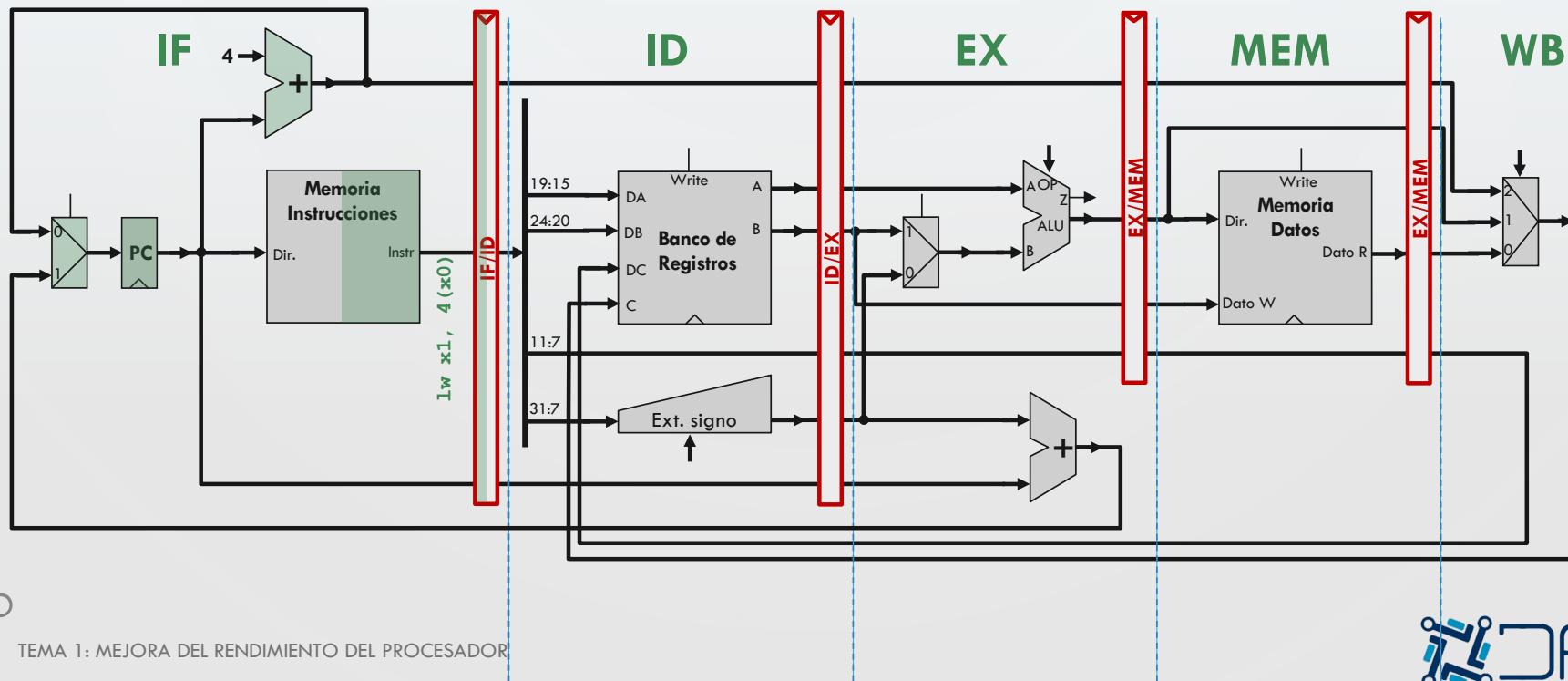
- Desacoplo de etapas mediante **Registros de Segmentación**
- Mantienen estables las entradas a cada fase para la instrucción a ejecutar durante el ciclo de reloj
- Los datos se van moviendo de etapa a etapa en cada ciclo de reloj



DISEÑO DEL CAMINO DE DATOS: CICLO 1

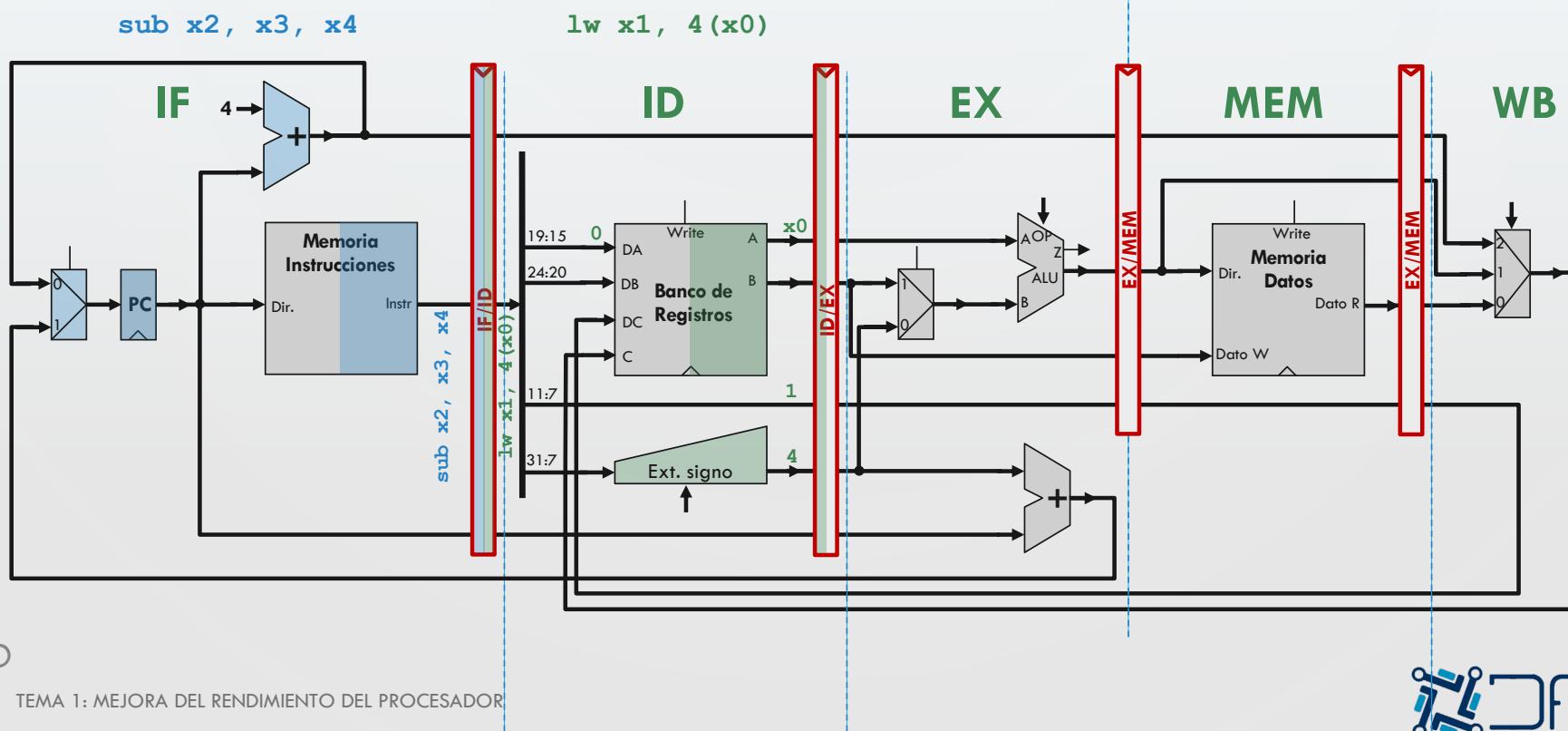
	1	2	3	4	5
lw x1, 4(x0)	IF	ID	EX	MEM	WB
sub x2, x3, x4	IF	ID	EX	MEM	
sw x5, 0(x6)		IF	ID	EX	
add x7, x8, x9			IF	ID	
beq x10, x11, 30				IF	

lw x1, 4(x0)



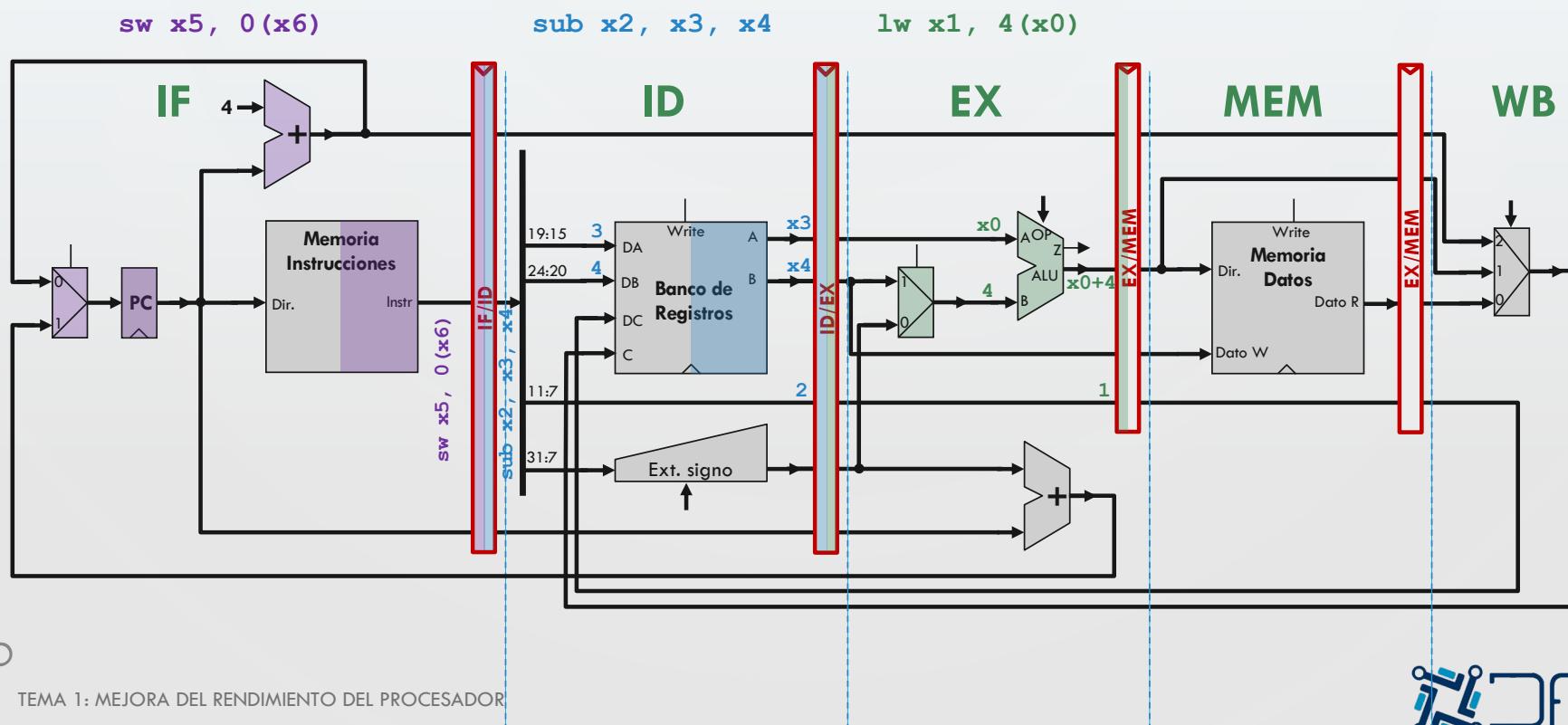
DISEÑO DEL CAMINO DE DATOS: CICLO 2

	1	2	3	4	5
lw x1, 4(x0)	IF	ID	EX	MEM	WB
sub x2, x3, x4		IF	ID	EX	MEM
sw x5, 0(x6)			IF	ID	EX
add x7, x8, x9				IF	ID
beq x10, x11, 30					IF



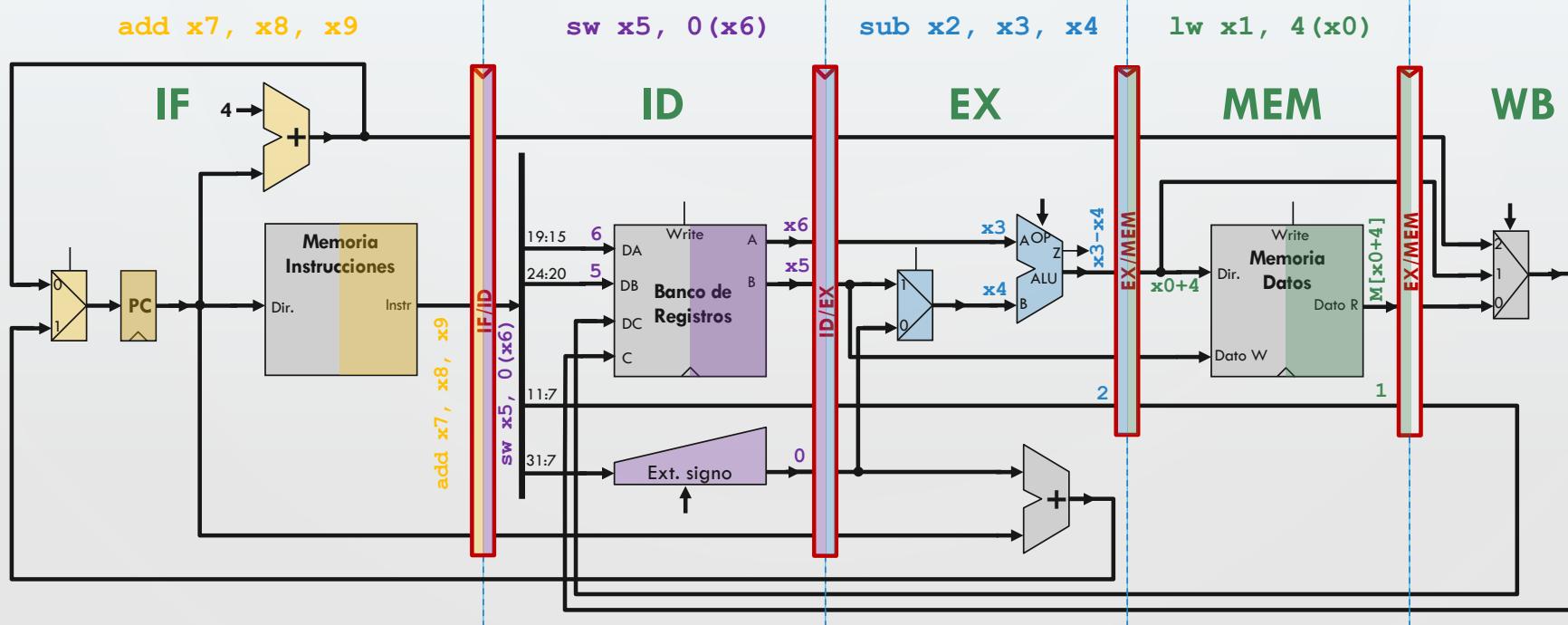
DISEÑO DEL CAMINO DE DATOS: CICLO 3

	1	2	3	4	5
lw x1, 4(x0)	IF	ID	EX	MEM	WB
sub x2, x3, x4		IF	ID	EX	MEM
sw x5, 0(x6)			IF	ID	EX
add x7, x8, x9				IF	ID
beq x10, x11, 30					IF



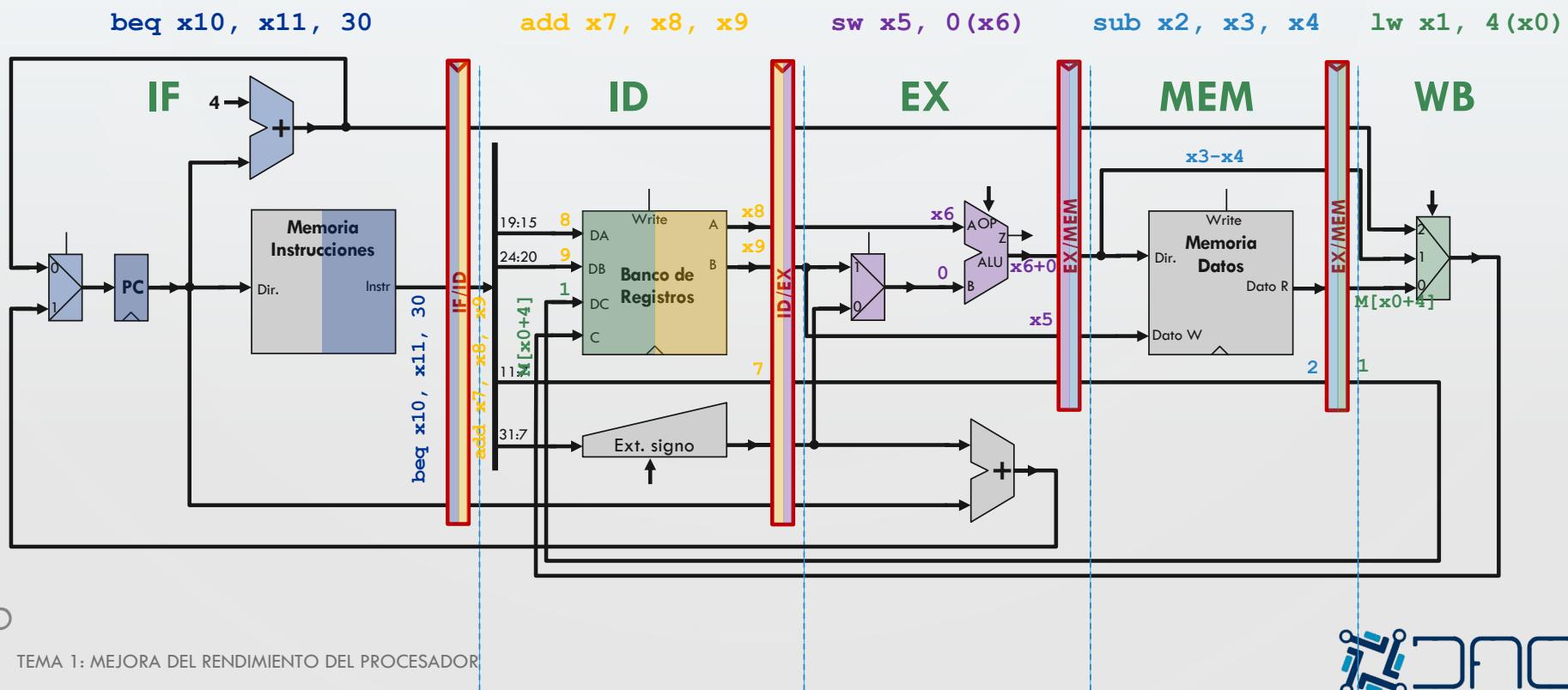
DISEÑO DEL CAMINO DATOS: CICLO 4

	1	2	3	4	5
lw x1, 4(x0)	IF	ID	EX	MEM	WB
sub x2, x3, x4		IF	ID	EX	MEM
sw x5, 0(x6)			IF	ID	EX
add x7, x8, x9				IF	ID
beq x10, x11, 30					IF



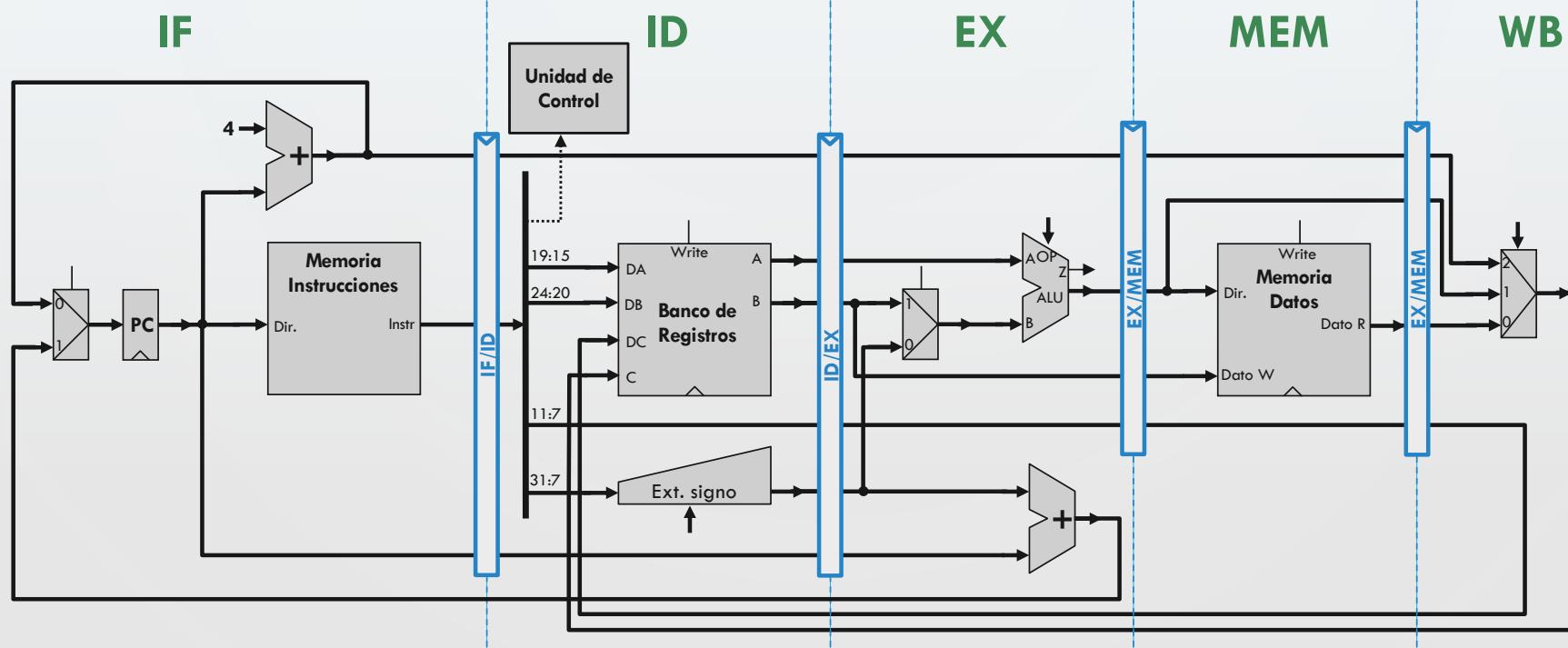
DISEÑO DEL CAMINO DATOS: CICLO 5

	1	2	3	4	5
lw x1, 4(x0)	IF	ID	EX	MEM	WB
sub x2, x3, x4		IF	ID	EX	MEM
sw x5, 0(x6)			IF	ID	EX
add x7, x8, x9				IF	ID
beq x10, x11, 30					IF



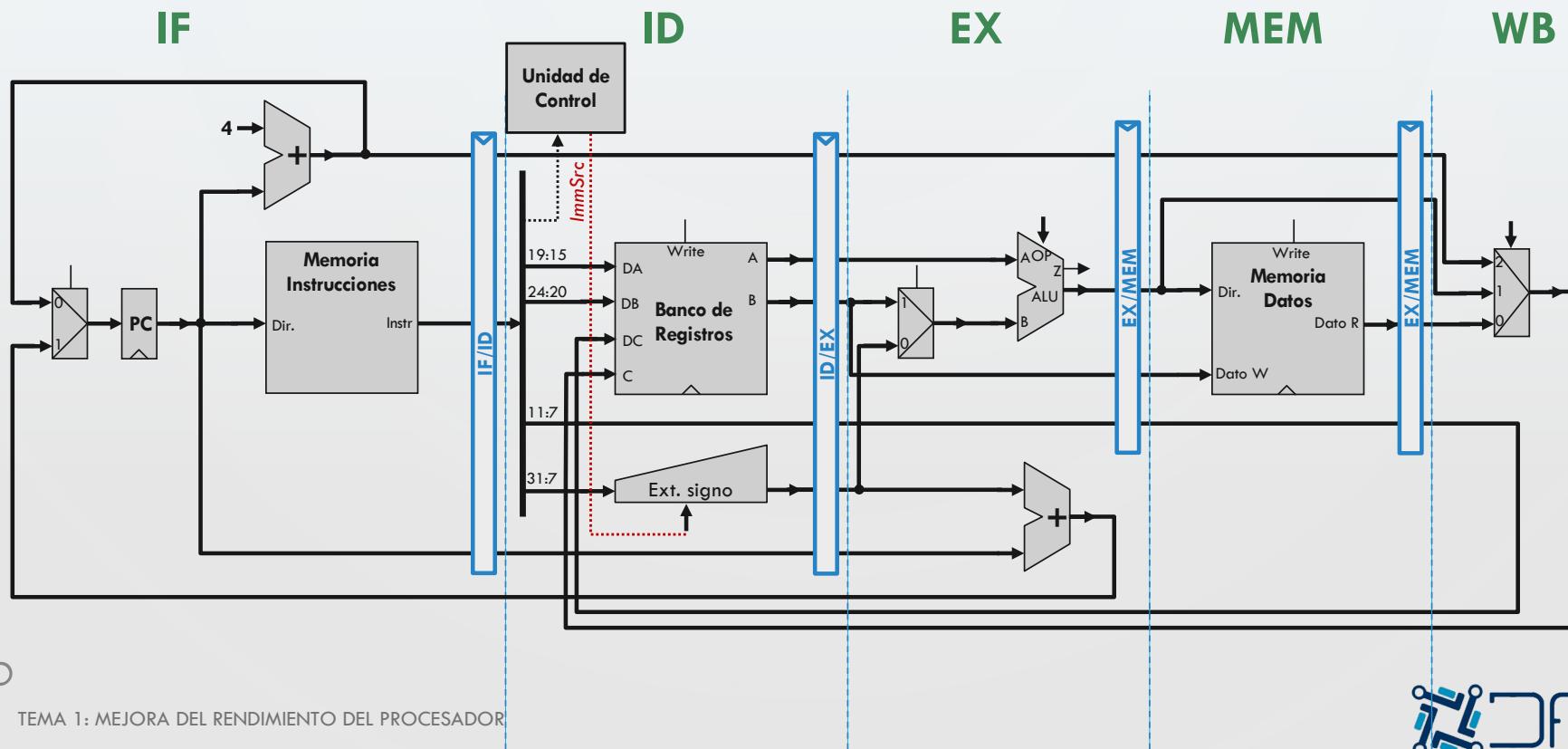
UNIDAD DE CONTROL

- Las señales de control se generan en **ID**, una vez leída y decodificada la instrucción
- El valor de las señales de control es **exactamente el mismo que para la implementación monociclo**
- Según la unidad funcional que controlan actuarán en una etapa distinta



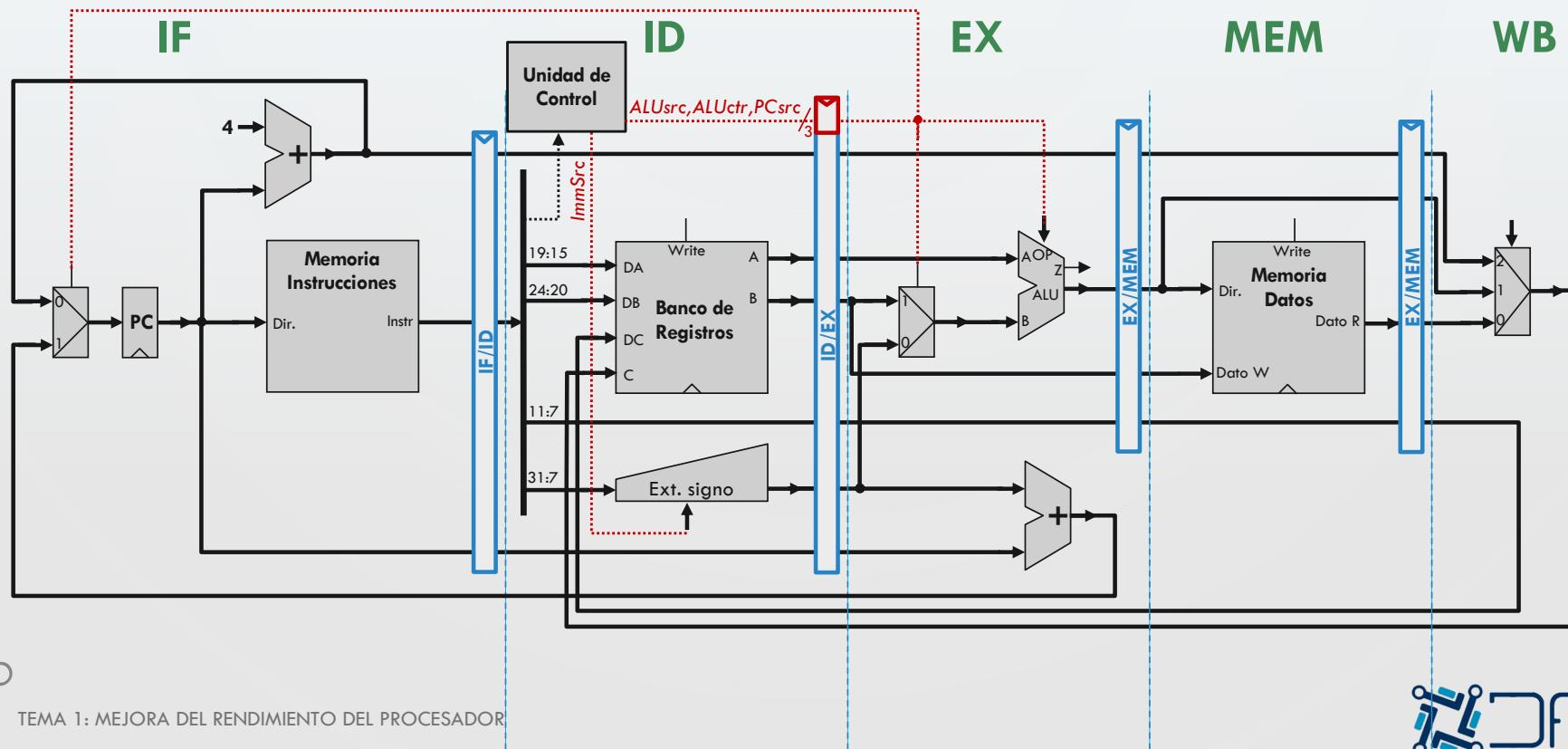
DISEÑO DE LA UNIDAD DE CONTROL

Etapa	Señales de control
ID	ImmSrc
EX	ALUsrc, ALUctr, PCsrc
MEM	MemWr
WB	ResSrc, BRwr



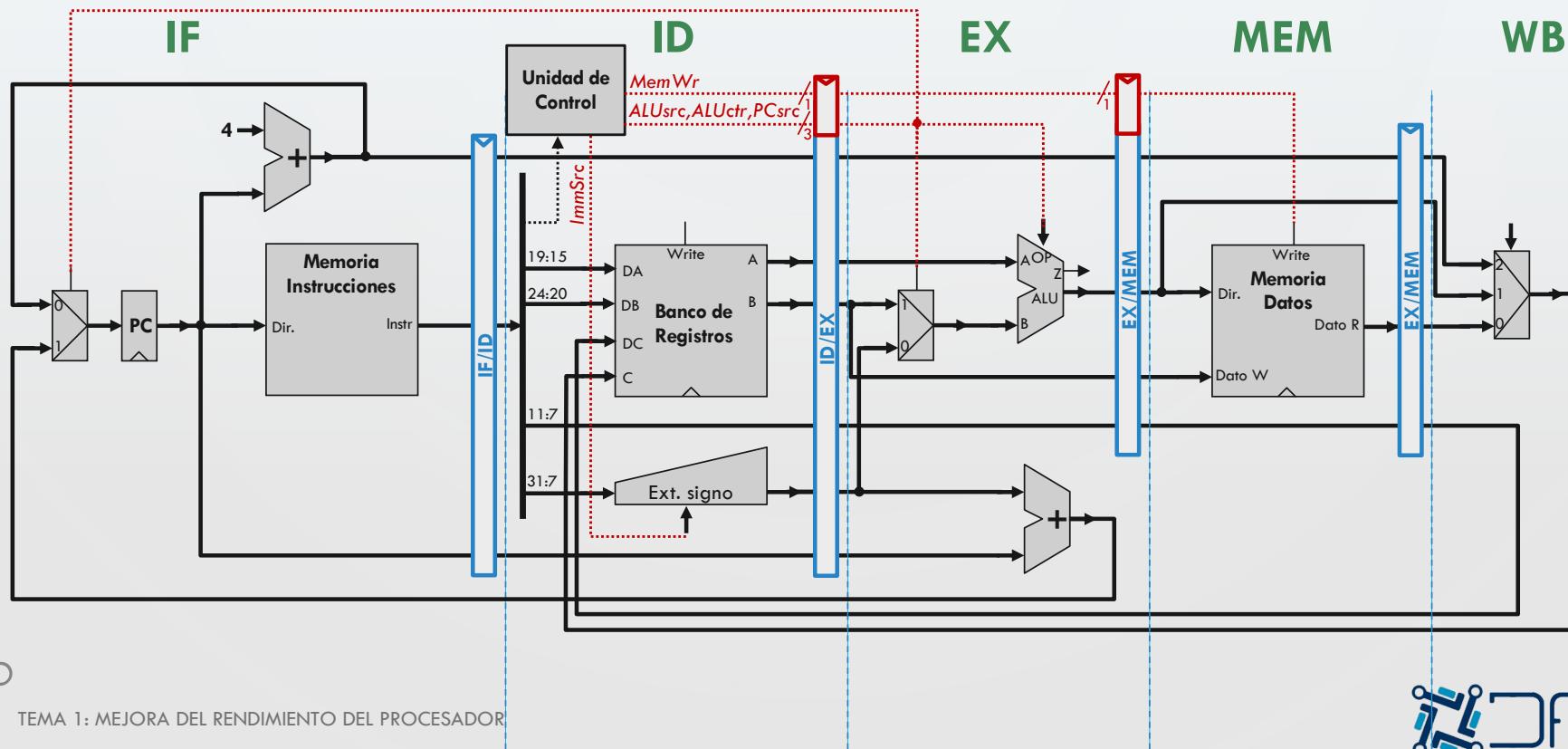
DISEÑO DE LA UNIDAD DE CONTROL

Etapa	Señales de control
ID	ImmSrc
EX	ALUsrc, ALUctr, PCsrc
MEM	MemWr
WB	ResSrc, BRwr



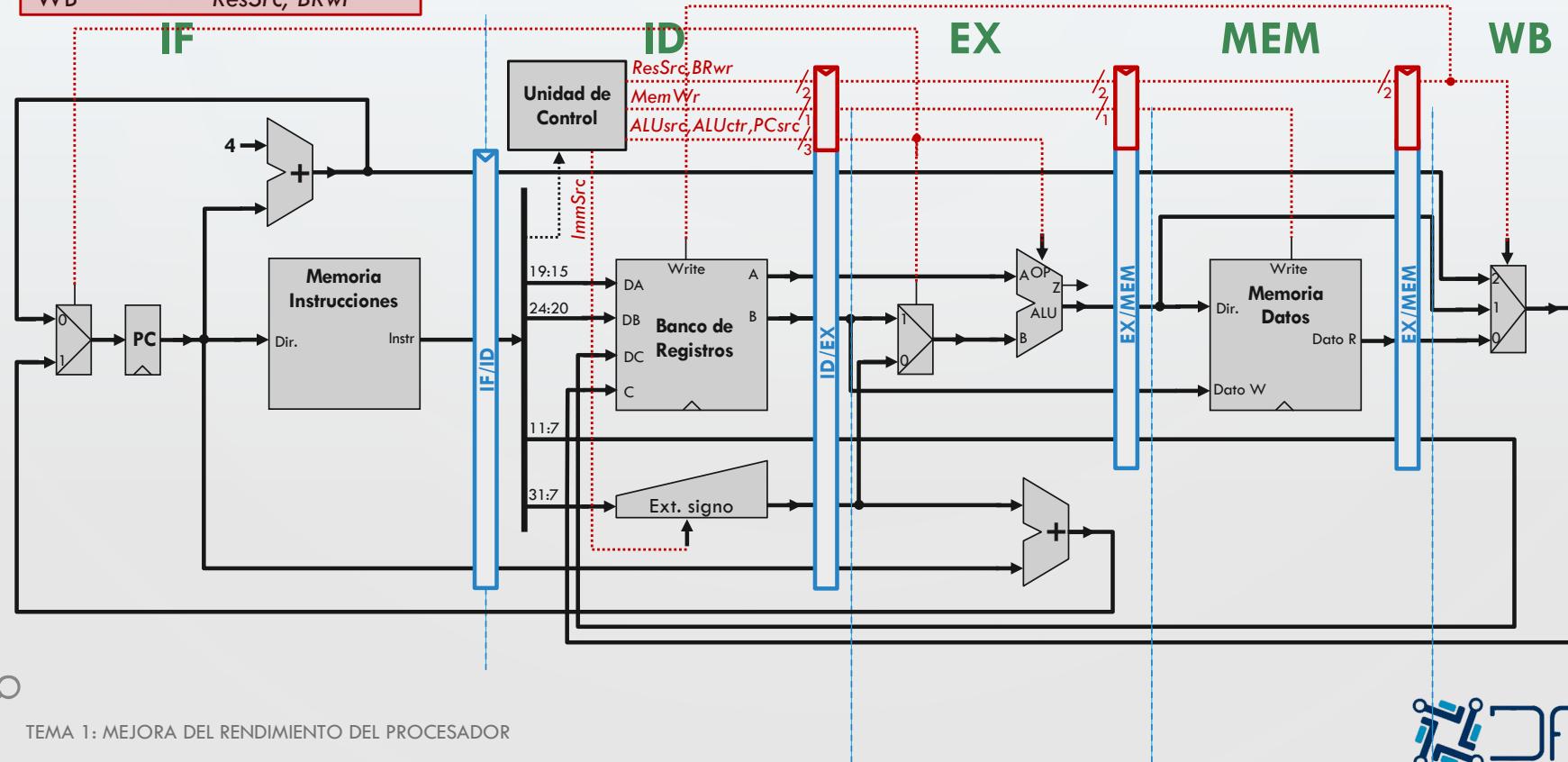
DISEÑO DE LA UNIDAD DE CONTROL

Etapa	Señales de control
ID	ImmSrc
EX	ALUsrc, ALUctr, PCsrc
MEM	MemWr
WB	ResSrc, BRwr



DISEÑO DE LA UNIDAD DE CONTROL

Etapa	Señales de control
ID	ImmSrc
EX	ALUsrc, ALUctr, PCsrc
MEM	MemWr
WB	ResSrc, BRwr



PROCESADOR SEGMENTADO: RIESGOS POR DEPENDENCIAS EN EL CÓDIGO

- Dependencias y riesgos
- Riesgos estructurales
- Riesgos de datos
- Riesgos de control

TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR

DEPENDENCIAS Y RIESGOS

- **Dependencia:** circunstancia que restringe la ejecución de un código (sólo depende de su **semántica**)

Las dependencias entre instrucciones *pueden* dar lugar:

- A la ejecución incorrecta del código si no actuamos sobre ellas.
- A conflictos en la ejecución

... en función de sobre qué microarquitectura se ejecuten

- En un **procesador segmentado** las dependencias pueden dar lugar a conflictos (**RIESGOS**) entre las instrucciones que están simultáneamente en ejecución:

- **No suceden en procesadores monociclo/multiciclo** porque una instrucción no comienza a ejecutarse hasta que la anterior no ha finalizado.

TIPOS DE DEPENDENCIAS

Supongamos que i precede a j:

- **Dependencia de datos:** se debe respetar el orden de acceso al mismo operando (registro, posición de memoria) en las instrucciones i y j
 - **Dependencia verdadera**
 - i escribe el operando y j lo lee
 - **Antidependencia**
 - i lee el operando y j lo escribe
 - **Dependencia de salida**
 - i y j escriben el operando
- **Dependencia de recursos:** i y j utilizan el mismo recurso
- **Dependencia de control:** i decide si se ejecuta o no j (saltos condicionales)

} Dependencias falsas o dependencias por nombre

DEPENDENCIAS DE DATOS

- La **distancia** entre i y j es el número de instrucciones que hay entre ambas más uno:
 - Distancia 1: i y j son consecutivas
 - Distancia 2: hay otra instrucción entre i y j
 - Y así sucesivamente
- Dependencia **verdadera** o de flujo
 - Una instrucción j depende de otra i si se verifica que:
 - La instrucción i produce un resultado que es usado por j, es decir, hay un flujo de información desde la instrucción i hacia la instrucción j
 - Ejemplo:

i: ld **f0** 0 (r1)
j: addd **f4** **f0**, f2
k: sd 0 (r1), **f4**

j depende de i
k depende de j

- Para anotar la dependencia hay que indicar las instrucciones implicadas y el operando causante de la dependencia

DEPENDENCIAS DE DATOS

- **Antidependencia:**

- Una instrucción j depende de otra i por antidependencia si:
 - Si i precede a j e i lee un registro o posición de memoria que es escrito luego por j

- Ejemplo:

i: addd f4, **f0**, f2
j: ld **f0**, 0 (r1)

- Dependencia de **salida**

- Una instrucción j depende de otra i por dependencia de salida si:
 - i precede a j y las dos instrucciones (j e i) escriben en el mismo registro o posición de memoria

- Ejemplo:

i: addd **f0**, f4, f2
j: ld **f0**, 0 (r1)

TIPOS DE RIESGOS (HAZARDS)

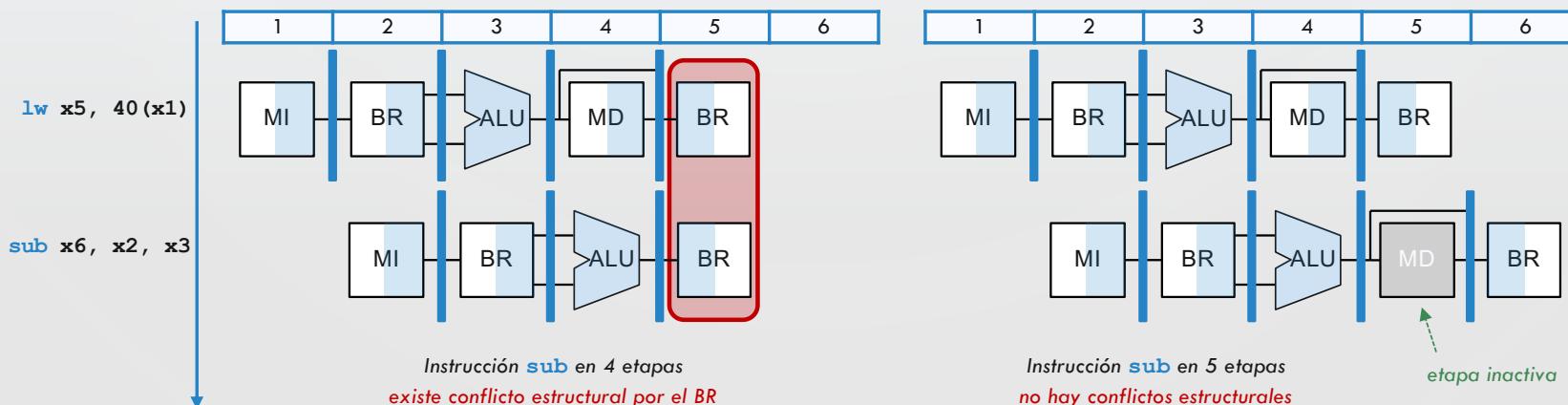
- **Estructurales**: causado por una dependencia de recursos. Una instrucción necesita usar un recurso hardware que está siendo utilizado por una instrucción lanzada con anterioridad
- **De datos**: causado por una dependencia de datos.
 - **RAW** (Read AfterWrite): causado por dependencia verdadera
 - **WAR** (Write after Read): causado por antidependencia
 - **WAW** (Write after Write): causado por dependencia de salida
- **De control**: causado por dependencia de control. Debe leerse de memoria la siguiente instrucción a ejecutar pero su dirección todavía no ha sido decidida/calculada por una instrucción de salto lanzada con anterioridad

RIESGOS POSIBLES EN NUESTRO RISCV SEGMENTADO

- Para saber qué riesgos se van a producir durante la ejecución de un código debemos conocer los detalles de la microarquitectura sobre la que se va a ejecutar
- ¿Qué riesgos pueden aparecer en nuestro RISCV segmentado?
 - Estructurales: ¿?
 - De datos: ¿?
 - RAW (Read After Write): ¿?
 - WAR (Write after Read): ¿?
 - WAW (Write after Write): ¿?
 - De control: ¿?

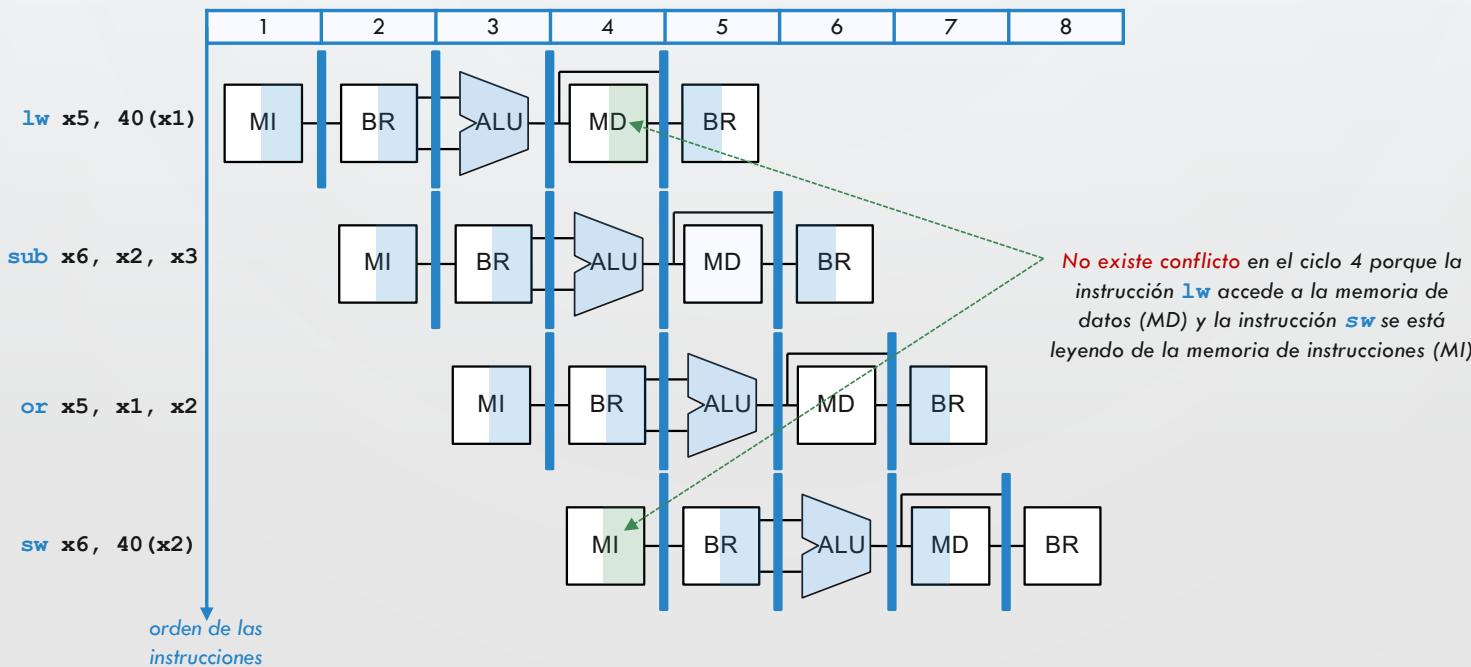
RIESGOS ESTRUCTURALES

- Este procesador segmentado **carence de conflictos estructurales** porque:
 - No hay recursos compartidos**
 - Simultáneamente puede **incrementarse el PC** (sumador, etapa IF), así **como calcular la dirección efectiva** (ALU, etapa EX) y la **condición de salto** (sumador, etapa EX)
 - Todas las instrucciones pasan por las 5 etapas**
 - Añadiendo **etapas inactivas** cuando sea necesario para evitar los conflictos.



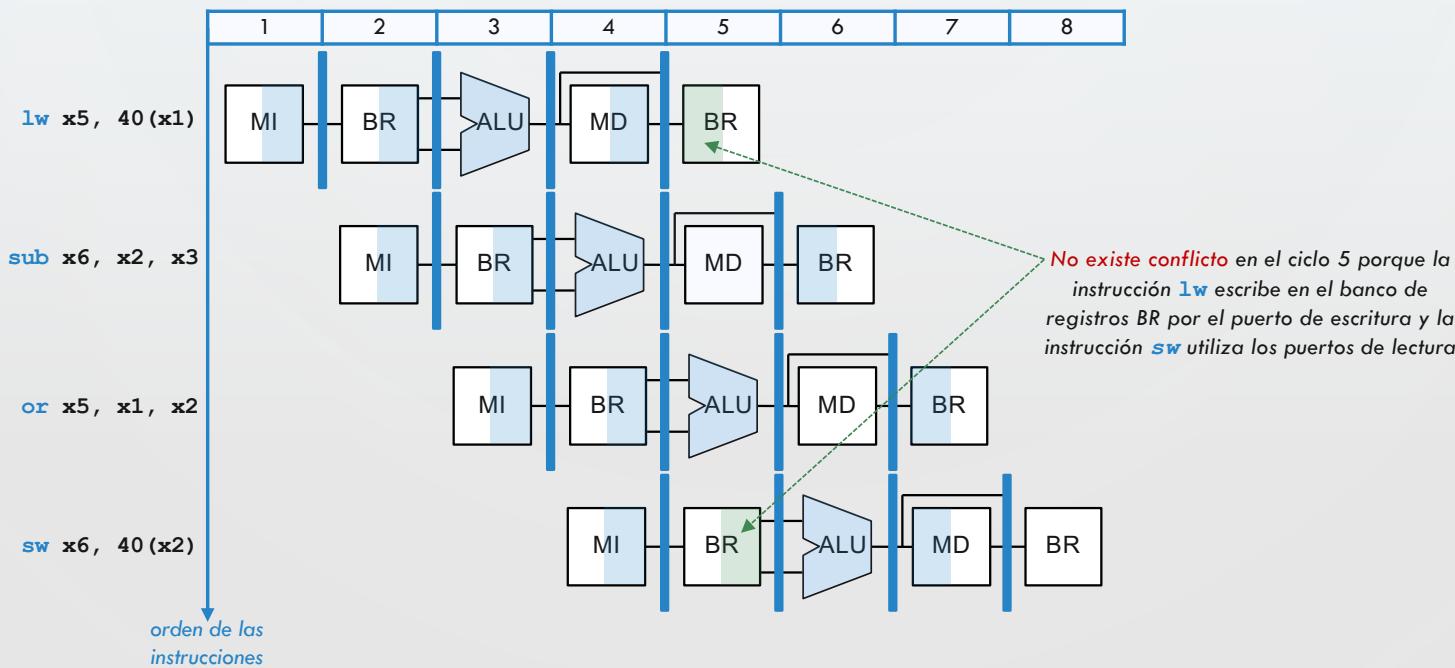
RIESGOS ESTRUCTURALES

- Este procesador segmentado **carence de conflictos estructurales** porque:
 - Hay memorias distintas para instrucciones y datos
 - Simultáneamente puede leerse instrucciones (MI, etapa IF) y datos (MD, etapa MEM)



RIESGOS ESTRUCTURALES

- Este procesador segmentado **carence de conflictos estructurales** porque:
 - El banco de registros tiene triple puerto
 - Simultáneamente pueden leerse 2 registros (etapa ID) y escribirse 1 (etapa WB)

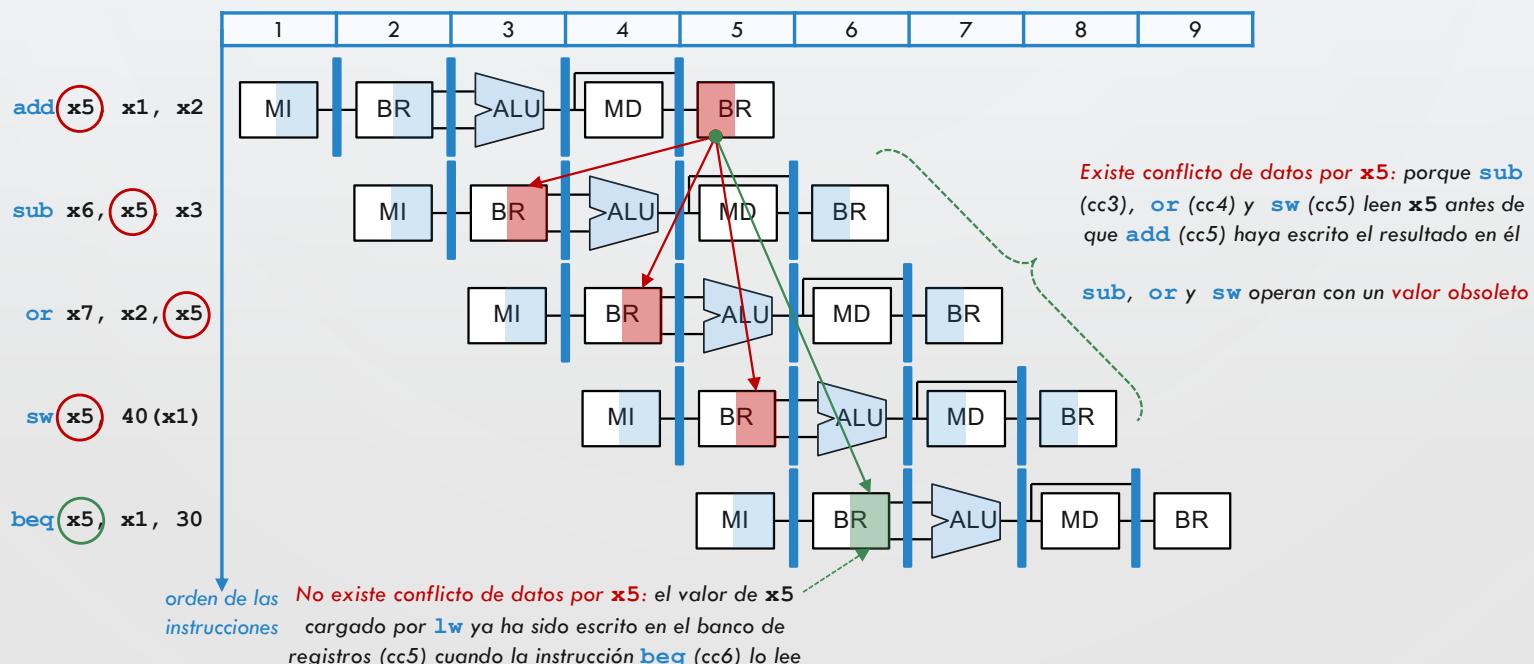


RIESGOS POSIBLES EN NUESTRO RISCV SEGMENTADO

- ¿Qué riesgos pueden aparecer en nuestro RISCV segmentado?
 - Estructurales: NO
 - De datos: ¿?
 - RAW (Read AfterWrite): ¿?
 - WAR (Write after Read): ¿?
 - WAW (Write after Write): ¿?
 - De control: ¿?

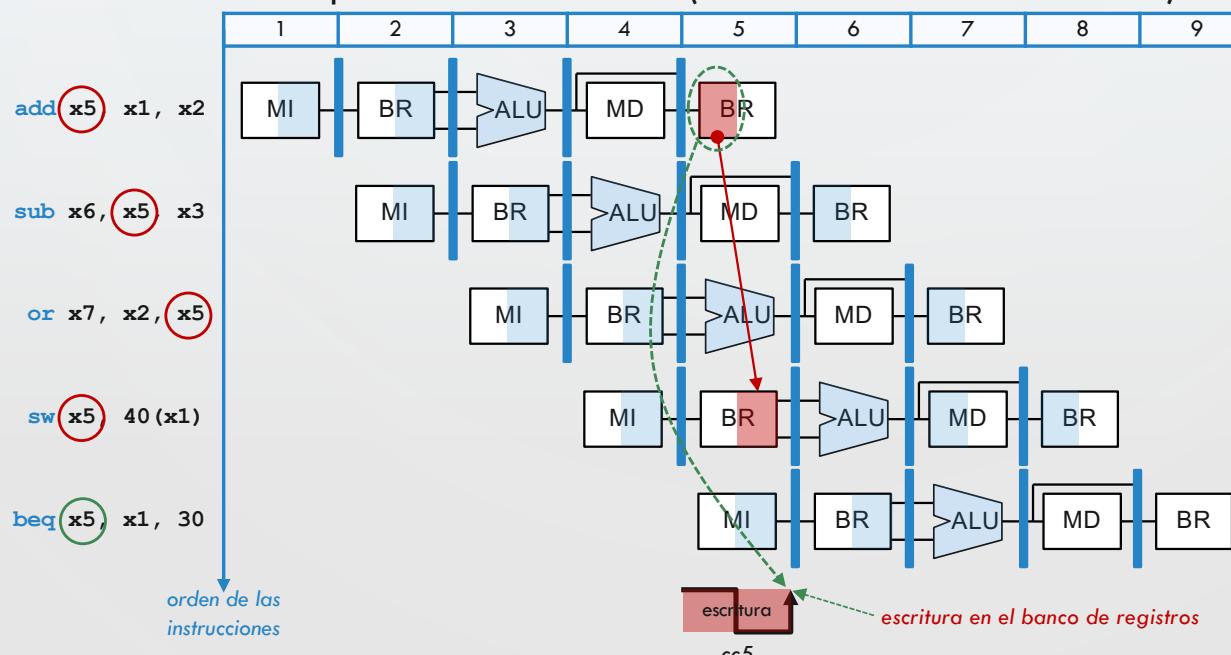
RIESGOS DE DATOS RAW

- Este procesador segmentado **tiene conflictos de datos** al ejecutar cualquier instrucción que lea un registro que sea escrito por cualquiera de las 3 instrucciones anteriores (distancia ≤ 3)



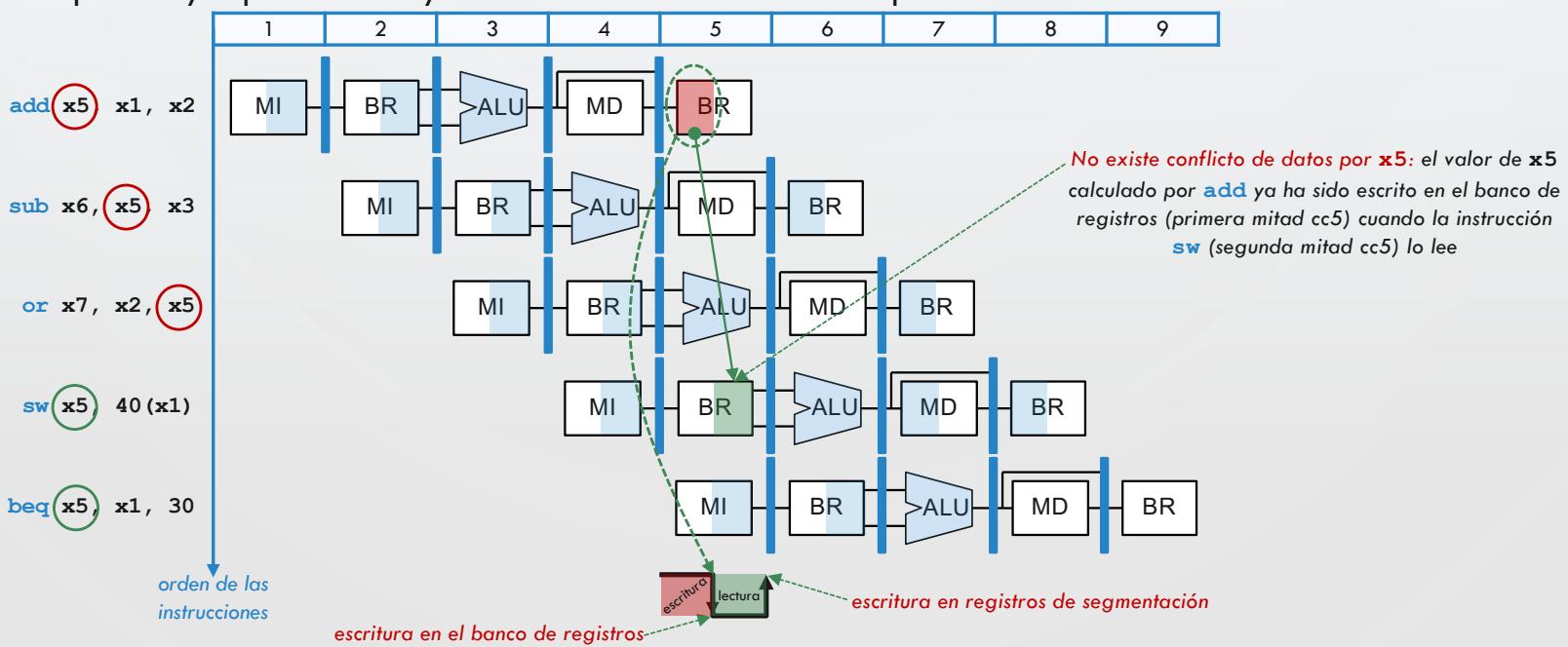
RIESGOS DE DATOS RAW

- ¿Por qué el acceso a **x5** por la instrucción **sw** en el cc5 es **incorrecto** si la instrucción **add** lo escribe en ese mismo ciclo?
- Los registros se escriben **al final del ciclo** (flanco de subida de clk), por lo que el valor leído por **sw** durante el cc5 es **justo el valor anterior** (aún no se ha escrito el nuevo)



RIESGOS DE DATOS RAW

- HW alternativo para reducir su impacto: **Anticipación/Adelantamiento en el banco de registros:**
 - Usar la primera mitad del ciclo para la escritura en el banco de registros (flanco de bajada)
 - Usar la segunda mitad del ciclo para la lectura del banco de registros
 - Es posible ya que escritura y lectura del BR < mitad del tiempo de ciclo



RIESGOS POSIBLES EN NUESTRO RISCV SEGMENTADO

- ¿Qué riesgos pueden aparecer en nuestro RISCV segmentado?
 - Estructurales: NO
 - De datos: SÍ
 - RAW (Read After Write): SÍ
 - WAR (Write after Read): ¿?
 - WAW (Write after Write): ¿?
 - De control: ¿?

RIESGOS DE DATOS WAR

- No provocan riesgos en nuestro RISC-V de 5 etapas
- Aunque exista una antidependencia entre dos instrucciones i y j
 - La instrucción i siempre lee sus operandos antes de que j escriba



- Operandos en registros: i lee de x1 y j escribe en x1.
 - La instrucción i lee en ID y j escribe en WB cuatro ciclos más tarde
- Operandos en memoria:
 - i es un lw de una posición de memoria y j es un sw que escribe en la misma posición
 - Aunque hay una antidependencia entre i y j, no hay riesgo porque i lee primero y j escribe luego
- En otras arquitecturas si pueden aparecer riesgos WAR
 - Si se permiten escrituras al comienzo del cauce y lecturas al final
 - En arquitecturas con ejecución fuera de orden, o con instrucciones de duración variable

RIESGOS POSIBLES EN NUESTRO RISCV SEGMENTADO

- ¿Qué riesgos pueden aparecer en nuestro RISCV segmentado?
 - **Estructurales:** NO
 - **De datos:** Sí
 - RAW (Read After Write): Sí
 - WAR (Write after Read): **NO**
 - WAW (Write after Write): ¿?
 - **De control:** ¿?

RIESGOS DE DATOS WAW

- No producen riesgos en la ejecución de instrucciones en nuestro RISCV de 5 etapas
- Aunque exista dependencia de salida entre i y j
 - Las escrituras de i y j se realizan en orden (primero i y luego j)



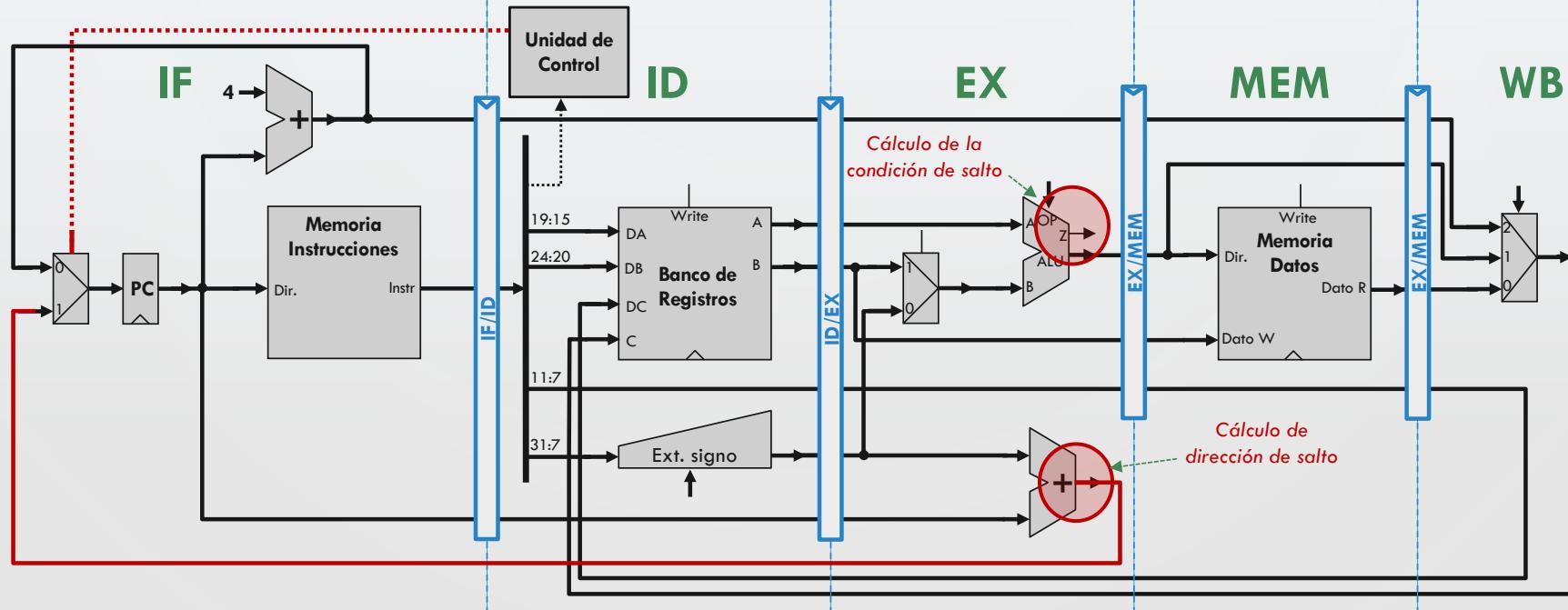
- En otras arquitecturas sí pueden aparecer.
- Una solución sencilla para los riesgos WAR y WAW consiste en cambiar el nombre del operando implicado

RIESGOS POSIBLES EN NUESTRO RISCV SEGMENTADO

- ¿Qué riesgos pueden aparecer en nuestro RISCV segmentado?
 - **Estructurales:** NO
 - **De datos:** Sí
 - RAW (Read After Write): Sí
 - WAR (Write after Read): NO
 - WAW (Write after Write): **NO**
 - **De control:** ¿?

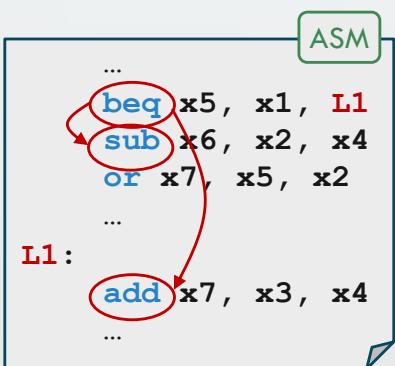
RIESGOS DE CONTROL

- Este procesador segmentado **tiene conflictos de control** al ejecutar instrucciones de salto, porque **debe leerse la siguiente instrucción**:
 - Antes de decidir si saltar o no (instrucción **beq**)
 - Antes de calcular la dirección de la instrucción a la que saltar en caso de hacerlo (instrucciones **beq** y **jal**)



RIESGOS DE CONTROL

- Después de la instrucción de salto se empiezan a ejecutar **dos instrucciones** que posiblemente sean **erróneas** (salto condicional) o definitivamente erróneas (salto incondicional)



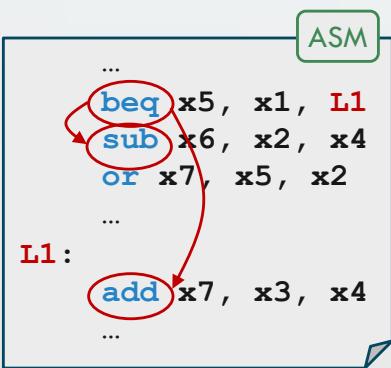
beq x5, x1, L1

1	2	3	4	5	6	7	8	9
MI								

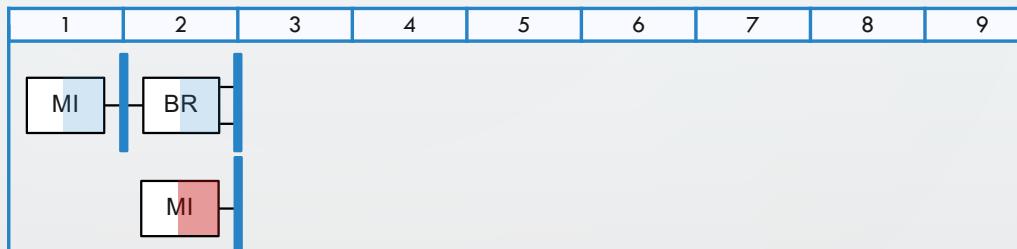
orden de las
instrucciones

RIESGOS DE CONTROL

- Después de la instrucción de salto se empiezan a ejecutar **dos instrucciones** que posiblemente sean **erróneas** (salto condicional) o definitivamente erróneas (salto incondicional)
 - la terminación del salto** es en EX: etapa en la que se actualizará PC con el valor correcto



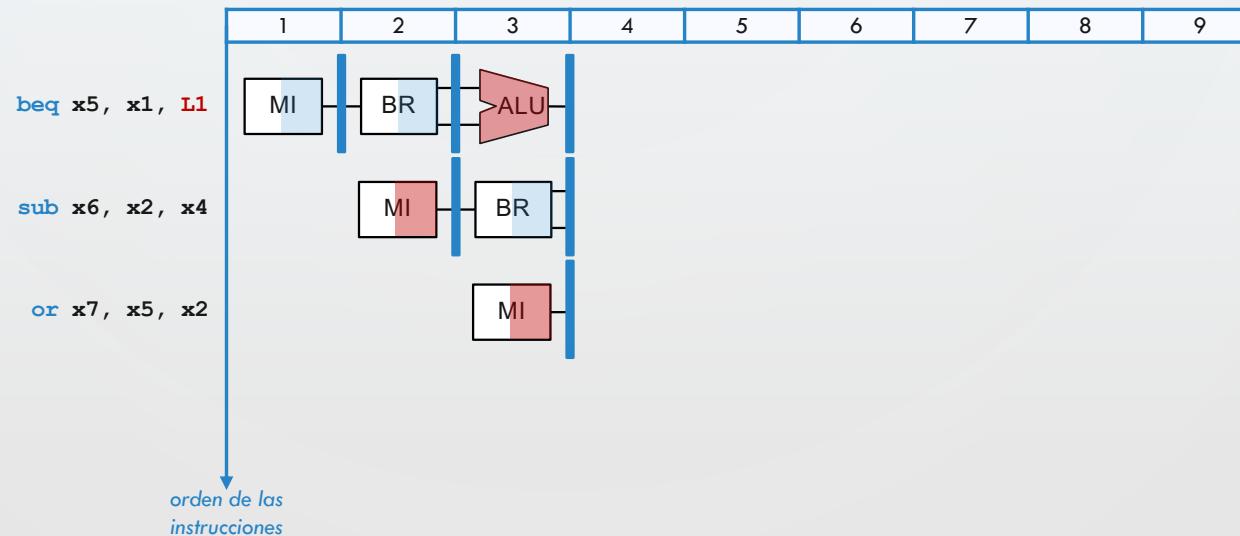
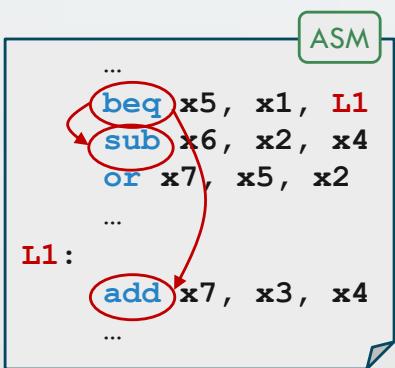
beq x5, x1, L1
sub x6, x2, x4



orden de las
instrucciones

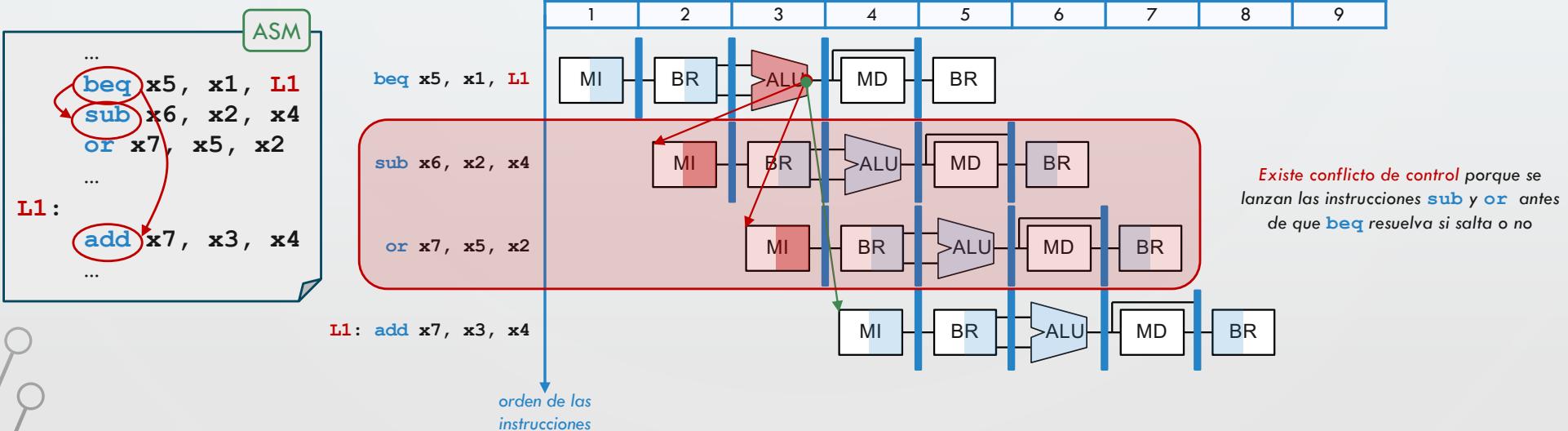
RIESGOS DE CONTROL

- Después de la instrucción de salto se empiezan a ejecutar **dos instrucciones** que posiblemente sean **erróneas** (salto condicional) o definitivamente erróneas (salto incondicional)



RIESGOS DE CONTROL

- Después de la instrucción de salto se empiezan a ejecutar **dos instrucciones** que posiblemente sean **erróneas** (salto condicional) o definitivamente erróneas (salto incondicional)



RIESGOS POSIBLES EN NUESTRO RISCV SEGMENTADO

- ¿Qué riesgos pueden aparecer en nuestro RISCV segmentado?
 - **Estructurales:** NO
 - **De datos:** SÍ
 - RAW (Read After Write): SÍ
 - WAR (Write after Read): NO
 - WAW (Write after Write): NO
 - **De control:** SÍ

PROCESADOR SEGMENTADO: SOLUCIONES SOFTWARE PARA LOS RIESGOS

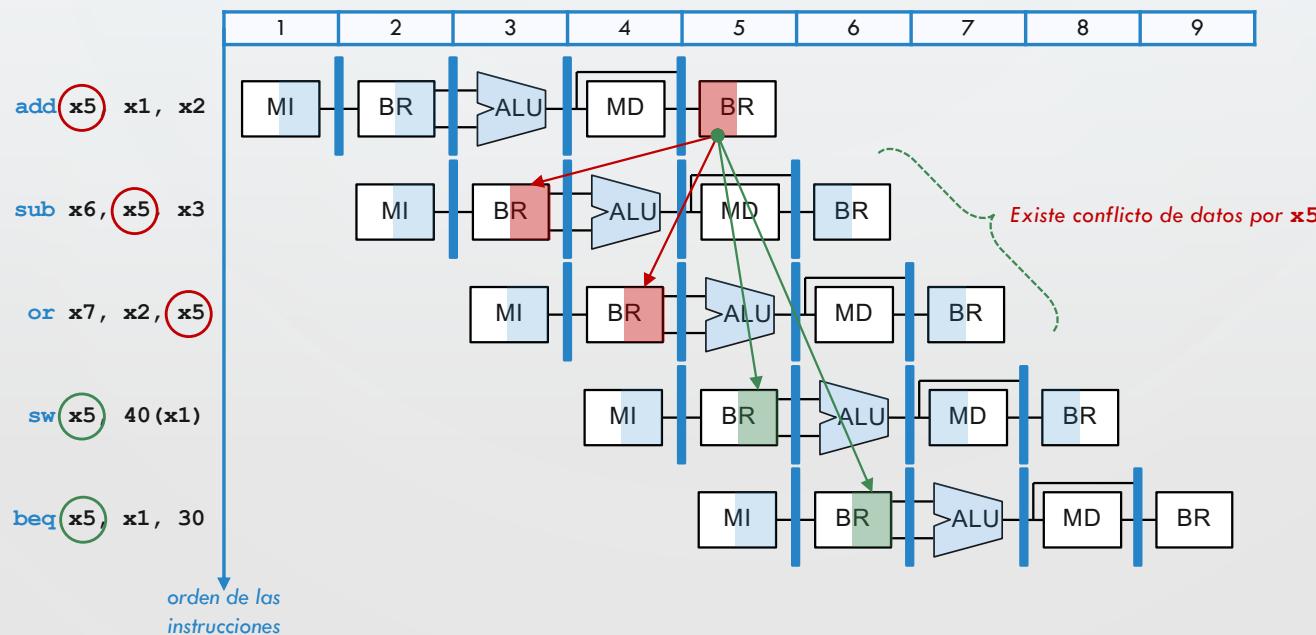
- Resolución de riesgos RAW:
 - Inserción de NOPs
 - Reordenación de código
- Resolución de riesgos de control
 - Inserción de NOPs
 - Salto retardado

RIESGOS DE DATOS RAW

- Soluciones software a los riesgos RAW: el programador o el compilador deben implementarlas para asegurar la correcta ejecución del código
 - **NOPs**
 - Insertar instrucciones NOPs entre las instrucciones dependientes para dar tiempo a que la escritura se realice y la lectura se haga así correctamente
 - Cada NOP introducido es un ciclo perdido
 - Hacen falta 2 NOPs para dependencias a distancia 1, y 1 NOP para distancia 2
 - **Reordenación**: insertar otras instrucciones del código entre las dependientes siempre que la semántica del código se mantenga
 - No siempre es posible por las dependencias
 - No se pierden ciclos

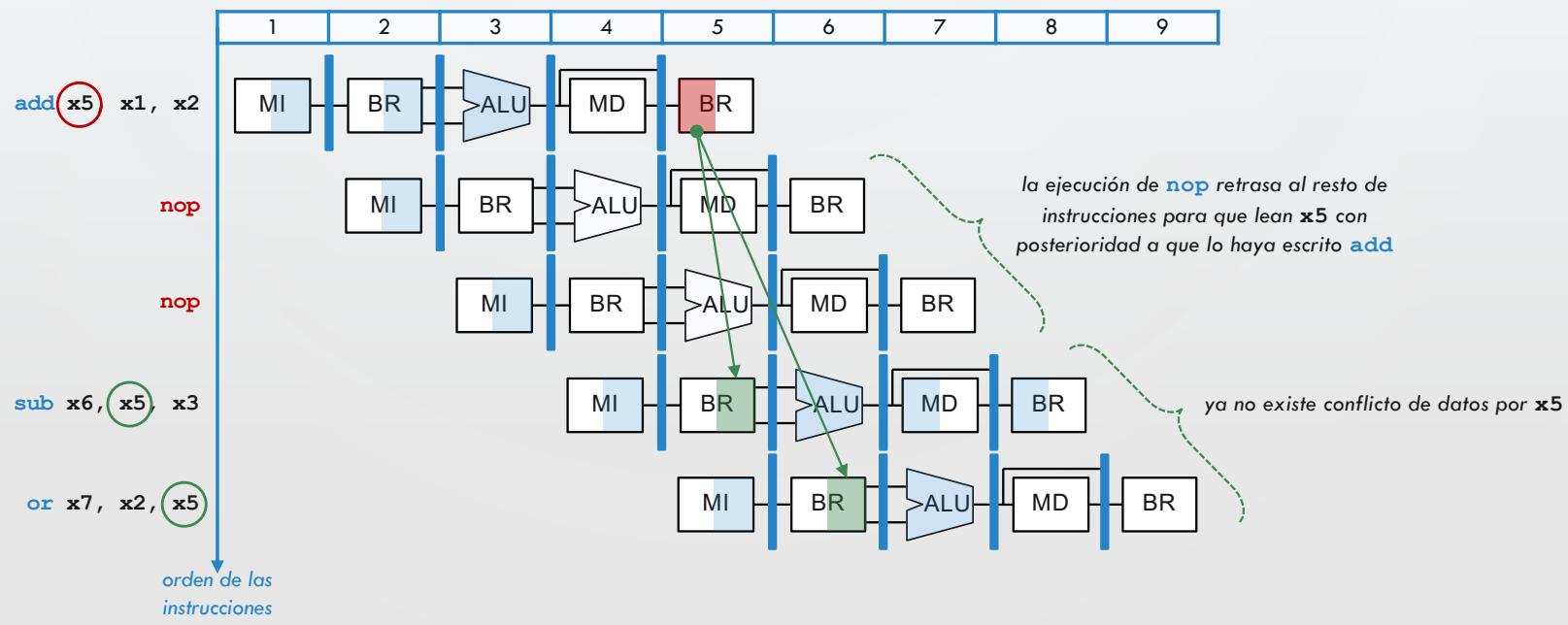
INSERCIÓN DE NOPS

- Se pueden separar las instrucciones que producen el riesgo insertando instrucciones **nop**



INSERCIÓN DE NOPS

- Se pueden separar las instrucciones que producen el riesgo insertando instrucciones **nop**



EJEMPLO DE INSERCIÓN DE NOPs

- Detectar los riesgos de datos y solucionarlos introduciendo instrucciones NOPs

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9
and x7, x6, x8	IF	ID	EX	M	WB				
add x4, x3, x2		IF	ID	EX	M	WB			
lw x5, 20(x4)			IF	ID	EX	M	WB		
sub x7, x5, x7				IF	ID	EX	M	WB	
or x3, x1, x2					IF	ID	EX	M	WB

EJEMPLO DE INSERCIÓN DE NOPs

- Detectar los riesgos de datos y solucionarlos introduciendo instrucciones NOPs

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9
and x7, x6, x8	IF	ID	EX	M	WB				
add x4, x3, x2		IF	ID	EX	M ^{x4}	WB			
lw x5, 20(x4)			IF	ID	EX	M	WB		
sub x7, x5, x7				IF	ID	EX	M	WB	
or x3, x1, x2					IF	ID	EX	M	WB

riesgo de datos



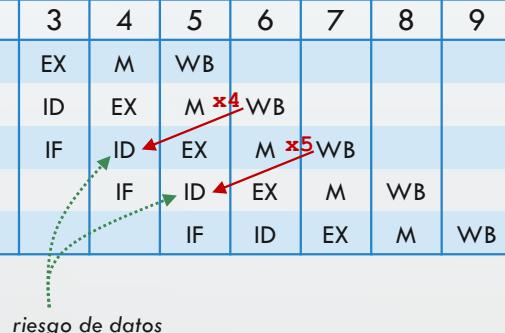
EJEMPLO DE INSERCIÓN DE NOPs

- Detectar los riesgos de datos y solucionarlos introduciendo instrucciones NOPs

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9
and x7, x6, x8	IF	ID	EX	M	WB				
add x4, x3, x2		IF	ID	EX	M ^{x4}	WB			
lw x5, 20(x4)			IF	ID	EX	M ^{x5}	WB		
sub x7, x5, x7				IF	ID	EX	M	WB	
or x3, x1, x2					IF	ID	EX	M	WB

riesgo de datos



EJEMPLO DE INSERCIÓN DE NOPs

- Detectar los riesgos de datos y solucionarlos introduciendo instrucciones NOPs

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9
and x7, x6, x8	IF	ID	EX	M	WB				
add x4, x3, x2		IF	ID	EX	M ^{x4}	WB			
lw x5, 20(x4)			IF	ID	EX	M ^{x5}	WB		
sub x7, x5, x7				IF	ID	EX	M	WB	
or x3, x1, x2					IF	ID	EX	M	WB

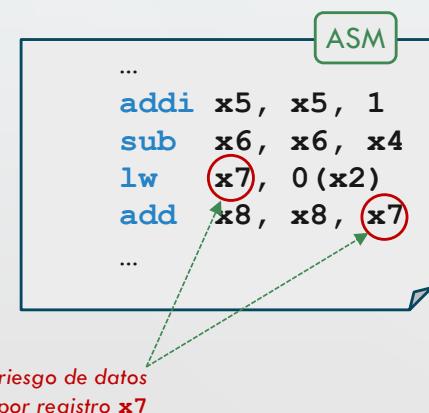
riesgo de datos

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX	M	WB ^{x4}							
nop			IF	ID	EX	M	WB						
nop				IF	ID	EX	M	WB					
lw x5, 20(x4)					IF	ID	EX	M	WB ^{x5}				
nop						IF	ID	EX	M	WB			
nop							IF	ID	EX	M	WB		
sub x7, x5, x7								IF	ID	EX	M	WB	
or x3, x1, x2									IF	ID	EX	M	WB

Tiempo de ejecución = 13 ciclos

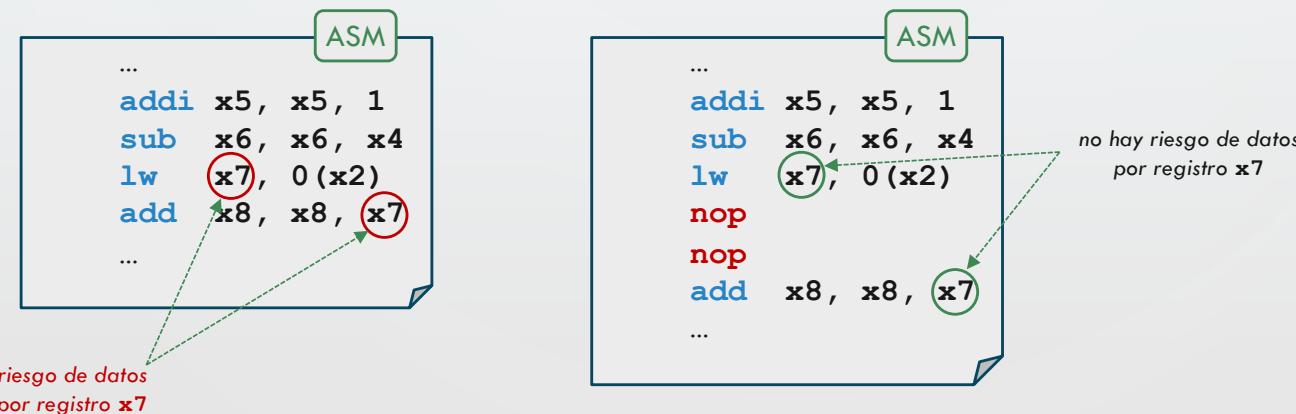
REORDENACIÓN DE CÓDIGO

- Se pueden separar las instrucciones que producen el riesgo **reordenando el código**
 - Respetando las dependencias para mantener semántica del código



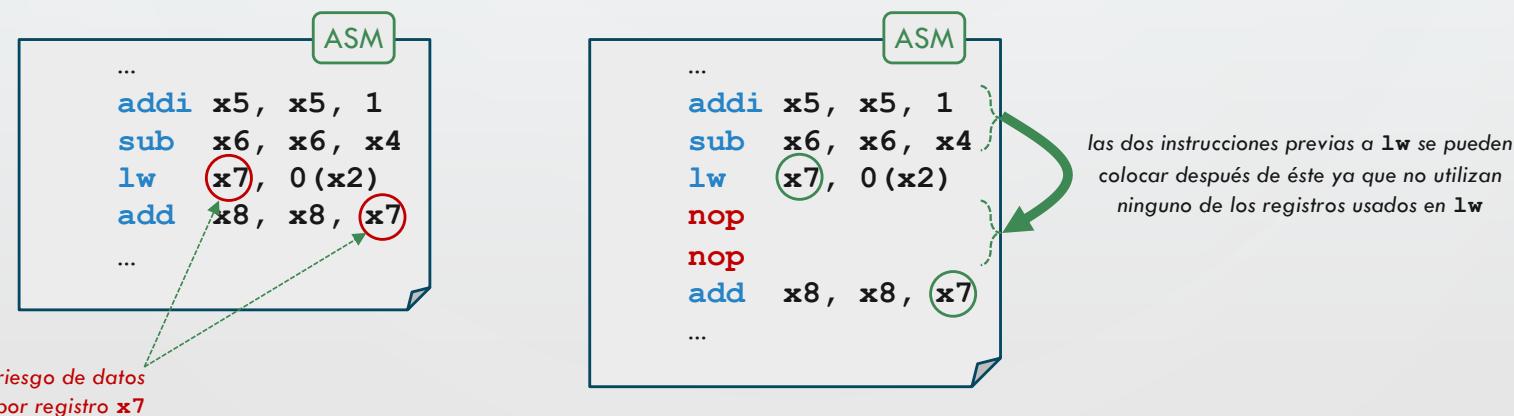
REORDENACIÓN DE CÓDIGO

- Se pueden separar las instrucciones que producen el riesgo **reordenando el código**
 - Respetando las dependencias para mantener semántica del código



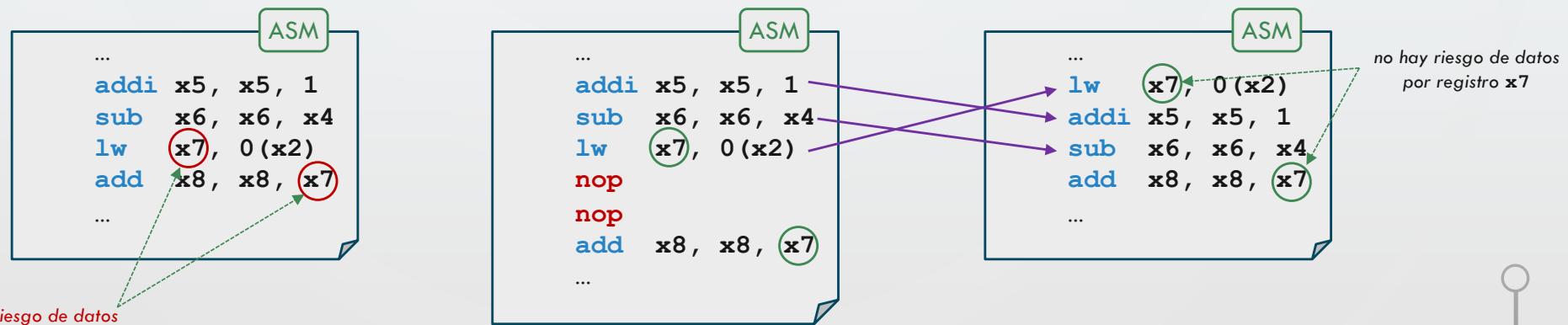
REORDENACIÓN DE CÓDIGO

- Se pueden separar las instrucciones que producen el riesgo **reordenando el código**
 - Respetando las dependencias para mantener semántica del código



REORDENACIÓN DE CÓDIGO

- Se pueden separar las instrucciones que producen el riesgo **reordenando el código**
 - Respetando las dependencias para mantener semántica del código



- Cuando no se puede reordenar, la **solución software (NOPs)** penaliza el tiempo de ejecución en 1 ciclo por cada **nop**

EJEMPLO DE REORDENACIÓN DE CÓDIGO

- Detectar los riesgos de datos y solucionarlos introduciendo el menor número de instrucciones NOPs posible (reordenación)

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9
<code>and x7, x6, x8</code>	IF	ID	EX	M	WB				
<code>add x4, x3, x2</code>		IF	ID	EX	M x4	WB			
<code>lw x5, 20(x4)</code>			IF	ID	EX	M x5	WB		
<code>sub x7, x5, x7</code>				IF	ID	EX	M	WB	
<code>or x3, x1, x2</code>					IF	ID	EX	M	WB

riesgo de datos

	1	2	3	4	5	6	7	8	9	10	11	12	13
<code>and x7, x6, x8</code>	IF	ID	EX	M	WB								
<code>add x4, x3, x2</code>		IF	ID	EX	M	WB x4							
<code>nop</code>			IF	ID	EX	M	WB						
<code>nop</code>				IF	ID	EX	M	WB					
<code>lw x5, 20(x4)</code>					IF	ID	EX	M	WB x5				
<code>nop</code>						IF	ID	EX	M	WB			
<code>nop</code>							IF	ID	EX	M	WB		
<code>sub x7, x5, x7</code>								IF	ID	EX	M	WB	
<code>or x3, x1, x2</code>									IF	ID	EX	M	WB

Tiempo de ejecución = 13 ciclos

EJEMPLO DE REORDENACIÓN DE CÓDIGO

- Detectar los riesgos de datos y solucionarlos introduciendo el menor número de instrucciones NOPs posible (reordenación)

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX	M	WB	x4						
nop			IF	ID	EX	M	WB						
nop				IF	ID	EX	M	WB					
lw x5, 20(x4)					IF	ID	EX	M	WB	x5			
nop						IF	ID	EX	M	WB			
nop							IF	ID	EX	M	WB		
sub x7, x5, x7							IF	ID	EX	M	WB		
or x3, x1, x2								IF	ID	EX	M	WB	

Tiempo de ejecución = 13 ciclos

EJEMPLO DE REORDENACIÓN DE CÓDIGO

- Detectar los riesgos de datos y solucionarlos introduciendo el menor número de instrucciones NOPs posible (reordenación)

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX	M	WB	x4						
nop			IF	ID	EX	M	WB						
nop				IF	ID	EX	M	WB					
lw x5, 20(x4)					IF	ID	EX	M	WB	x5			
nop						IF	ID	EX	M	WB			
nop							IF	ID	EX	M	WB		
sub x7, x5, x7								IF	ID	EX	M	WB	
or x3, x1, x2									IF	ID	EX	M	WB

Tiempo de ejecución = 13 ciclos

	1	2	3	4	5	6	7	8	9	10	11		
add x4, x3, x2	IF	ID	EX	M	WB	x4							
and x7, x6, x8		IF	ID	EX	M	WB							
or x3, x1, x2			IF	ID	EX	M	WB						
lw x5, 20(x4)				IF	ID	EX	M	WB	x5				
nop					IF	ID	EX	M	WB				
nop						IF	ID	EX	M	WB			
sub x7, x5, x7							IF	ID	EX	M	WB		

Tiempo de ejecución = 11 ciclos

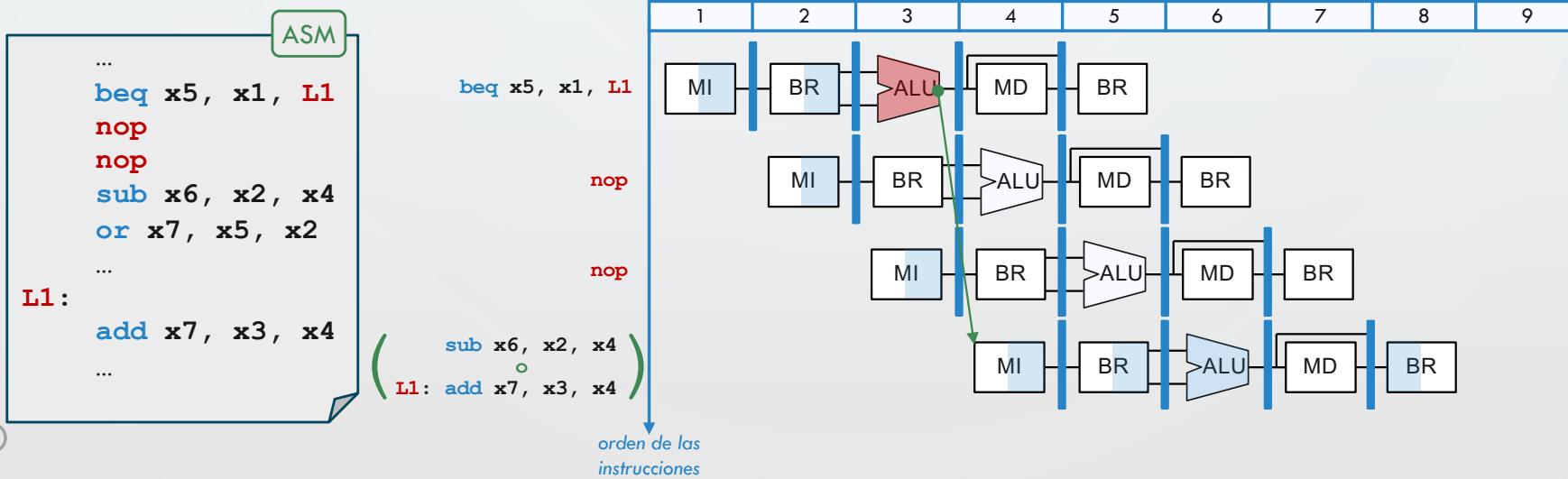


RIESGOS DE CONTROL

- Soluciones software a los riesgos de control: el programador o el compilador deben implementarlas para asegurar la correcta ejecución del código
 - NOPs
 - Insertar instrucciones NOPs detrás de las instrucciones de salto (programador o compilador) para llenar los huecos de retardo
 - 1 NOP tras la instrucción de salto si ésta termina en ID; 2 NOPs si termina en EX; 3 NOPs si termina en MEM
 - Cada NOP es un ciclo perdido
 - Salto retardado
 - Rellenar los huecos de retardo colocando instrucciones **independientes** al salto tras el mismo
 - No siempre es posible por las dependencias
 - Evitamos la pérdida de ciclos

INSERCIÓN DE NOPS

- Los riesgos de control pueden resolverse por software insertando 2 instrucciones **nop** después de cada instrucción de salto



EJEMPLO DE INSERCIÓN DE NOPs

- Detectar los riesgos de datos y control, y solucionarlos introduciendo instrucciones NOPs.

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

EJEMPLO DE INSERCIÓN DE NOPs

- Detectar los riesgos de datos y control, y solucionarlos introduciendo instrucciones NOPs.

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB		WB		WB			
add x3, x3, x2		IF	ID ← EX	M							
addi x1, x1, 4			IF	ID EX	M	WB					
addi x4, x4, -1				IF ID	EX M x4 WB						
bne x4, x0, L					IF ID ← EX M	WB					
addi x5, x5, 10					IF ID EX M x5 WB						
sw x3, 0(x5)						WB					

riesgo de datos

EJEMPLO DE INSERCIÓN DE NOPs

- Detectar los riesgos de datos y control, y solucionarlos introduciendo instrucciones NOPs.

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID ← EX	M	WB					
addi x4, x4, -1				IF	ID ← EX	M x4 WB					
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF	ID ← EX	M x5 WB			
sw x3, 0(x5)							IF	ID ← EX	M	WB	

riesgo de datos

riesgo de control

EJEMPLO DE INSERCIÓN DE NOPs

- Detectar los riesgos de datos y control, y solucionarlos introduciendo instrucciones NOPs.

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	EX	M	WB			
addi x5, x5, 10						IF	ID	EX	M x5 WB		
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2															
nop			IF	ID	EX	M	WB													
nop				IF	ID	EX	M	WB												
add x3, x3, x2					IF	ID	EX	M	WB											
addi x1, x1, 4						IF	ID	EX	M	WB										
addi x4, x4, -1							IF	ID	EX	M	WB x4									
nop								IF	ID	EX	M	WB								
nop									IF	ID	EX	M	WB							
bne x4, x0, L										IF	ID	EX	M	WB						
nop											IF	ID	EX	M	WB					
nop												IF	ID	EX	M	WB				
L: lw x2, 0(x1)													IF	ID	EX	M	WB			
nop														IF	ID	EX	M	WB		
nop															IF	ID	EX	M	WB	
...																IF	ID	EX	M	WB

Tiempo de ejecución iteración bucle = 11 ciclos



EJEMPLO DE INSERCIÓN DE NOPs

- Detectar los riesgos de datos y control, y solucionarlos introduciendo instrucciones NOPs.

Diagrama de ejecución simplificado

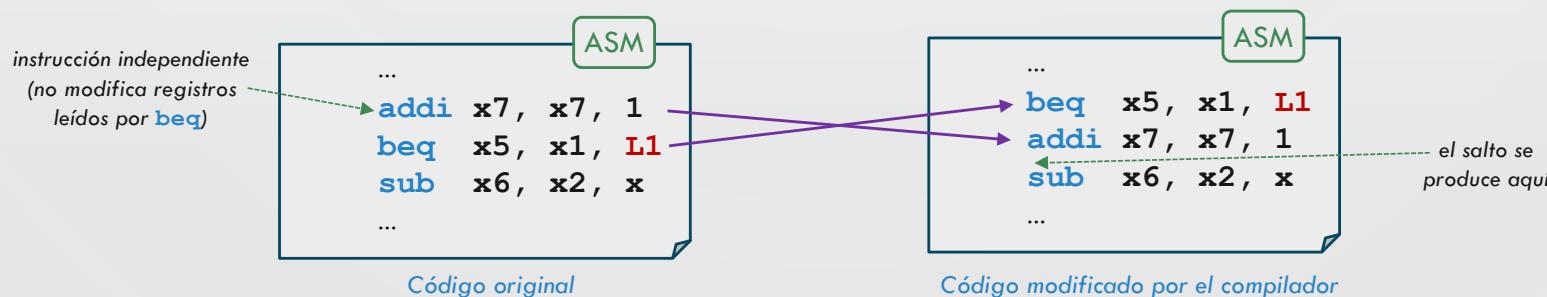
	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF	ID ← EX	M x5 WB			
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2														
nop		IF	ID	EX	M	WB													
nop			IF	ID	EX	M	WB												
add x3, x3, x2				IF	ID	EX	M	WB											
addi x1, x1, 4					IF	ID	EX	M	WB										
addi x4, x4, -1						IF	ID	EX	M	WB x4									
nop							IF	ID	EX	M	WB								
nop								IF	ID	EX	M	WB							
bne x4, x0, L									IF	ID	EX	M	WB						
nop										IF	ID	EX	M	WB					
nop											IF	ID	EX	M	WB				
addi x5, x5, 10												IF	ID	EX	M	WB x5			
nop													IF	ID	EX	M	WB		
nop														IF	ID	EX	M	WB	
sw x3, 0(x5)															IF	ID	EX	M	WB

SALTO RETARDADO

- Con la resolución de los saltos adelantada a la etapa ID, se pueden **eliminar completamente las paradas con los saltos retardados**:
 - Siempre se ejecuta la siguiente instrucción (secuencial) después de un salto (se elimina la parada por HW)
 - El salto tiene efecto siempre **después de la siguiente instrucción**
 - El compilador **move una instrucción (independiente)** anterior al salto a la posición siguiente al mismo (ocultando así el retardo del salto)



SALTO RETARDADO

- Problemas del salto retardado:
 - No siempre es posible encontrar instrucciones independientes por lo que hay que insertar NOPs
 - Para pipelines más profundos, el número de instrucciones a insertar después del salto es mayor y es más complicado encontrarlas
 - Han perdido popularidad frente a soluciones HW más flexibles:
 - Predicción de salto

EJEMPLO DE USO DE TÉCNICAS SW

- Transformamos el código inicial para que pueda ejecutarse sin riesgos en nuestro cauce segmentado utilizando técnicas SW

```
lw x1,desp(x0)
lw x2,0(x1)
lw x3,desp2(x0)
lw x4,desp3(x0)
loop: beq x3,x4,exit
      lw x4,0(x2)
      sw x4,0(x1)
      addi x2, x2, 4
      addi x1, x1, 4
      j loop
exit: ...
```

EJEMPLO DE USO DE TÉCNICAS SW

- Transformamos el código inicial para que pueda ejecutarse sin riesgos en nuestro cauce segmentado utilizando técnicas SW

```
lw x1,desp(x0)
lw x2,0(x1)
lw x3,desp2(x0)
lw x4,desp3(x0)
loop: beq x3,x4,exit
      lw x4,0(x2)
      sw x4,0(x1)
      addi x2, x2, 4
      addi x1, x1, 4
      j loop
exit: ...
```

```
lw x1,desp(x0)
lw x3,desp2(x0)
lw x4,desp3(x0)
lw x2,0(x1)
loop: beq x3,x4,exit
      lw x4,0(x2)
      sw x4,0(x1)
      addi x2, x2, 4
      addi x1, x1, 4
      j loop
exit: ...
```

EJEMPLO DE USO DE TÉCNICAS SW

- Transformamos el código inicial para que pueda ejecutarse sin riesgos en nuestro cauce segmentado utilizando técnicas SW

~~Reordenar~~

```
lw x1,desp(x0)
lw x3,desp2(x0)
lw x4,desp3(x0)
lw x2,0(x1)
loop: beq x3,x4,exit
      lw x4,0(x2)
      sw x4,0(x1)
      addi x2, x2, 4
      addi x1, x1, 4
      j loop
exit: ...
```

Insertar NOP

```
lw x1,desp(x0)
lw x3,desp2(x0)
lw x4,desp3(x0)
lw x2,0(x1)
loop: beq x3,x4,exit
      nop
      lw x4,0(x2)
      sw x4,0(x1)
      addi x2, x2, 4
      addi x1, x1, 4
      j loop
exit: ...
```

EJEMPLO DE USO DE TÉCNICAS SW

- Transformamos el código inicial para que pueda ejecutarse sin riesgos en nuestro cauce segmentado utilizando técnicas SW

```
lw x1,desp(x0)
lw x3,desp2(x0)
lw x4,desp3(x0)
lw x2,0(x1)
nop
loop: beq x3,x4,exit
      lw x4,0(x2)
      sw x4,0(x1)
      addi x2, x2, 4
      addi x1, x1, 4
      j loop
exit: ...
```

Insertar NOPs

```
lw x1,desp(x0)
lw x3,desp2(x0)
lw x4,desp3(x0)
lw x2,0(x1)
nop
loop: beq x3,x4,exit
      nop
      nop
      lw x4,0(x2)
      sw x4,0(x1)
      addi x2, x2, 4
      addi x1, x1, 4
      j loop
exit: ...
```

EJEMPLO DE USO DE TÉCNICAS SW

- Transformamos el código inicial para que pueda ejecutarse sin riesgos en nuestro cauce segmentado utilizando técnicas SW

```
lw x1,desp(x0)
lw x3,desp2(x0)
lw x4,desp3(x0)
lw x2,0(x1)
nop
loop: beq x3,x4,exit
nop
nop
lw x4,0(x2)
sw x4,0(x1)
addi x2, x2, 4
addi x1, x1, 4
j loop
exit: ...
```

Insertar NOPs

```
lw x1,desp(x0)
lw x3,desp2(x0)
lw x4,desp3(x0)
lw x2,0(x1)
nop
loop: beq x3,x4,exit
nop
nop
lw x4,0(x2)
nop
nop
sw x4,0(x1)
addi x2, x2, 4
addi x1, x1, 4
j loop
exit: ...
```

EJEMPLO DE USO DE TÉCNICAS SW

- Transformamos el código inicial para que pueda ejecutarse sin riesgos en nuestro cauce segmentado utilizando técnicas SW

```
lw x1,desp(x0)
lw x3,desp2(x0)
lw x4,desp3(x0)
lw x2,0(x1)
nop
loop: beq x3,x4,exit
nop
nop
lw x4,0(x2)
nop
nop
sw x4,0(x1)
addi x2, x2, 4
addi x1, x1, 4
j loop
exit: ...
```

Reordenar
(Salto retardado)

```
lw x1,desp(x0)
lw x3,desp2(x0)
lw x4,desp3(x0)
lw x2,0(x1)
nop
loop: beq x3,x4,exit
nop
nop
lw x4,0(x2)
nop
nop
sw x4,0(x1)
j loop
addi x2, x2, 4
addi x1, x1, 4
exit: ...
```

EJEMPLO DE USO DE TÉCNICAS SW

- Transformamos el código inicial para que pueda ejecutarse sin riesgos en nuestro cauce segmentado utilizando técnicas SW

Monociclo/multiciclo

```
lw x1,desp(x0)
lw x2,0(x1)
lw x3,desp2(x0)
lw x4,desp3(x0)
loop: beq x3,x4,exit
lw x4,0(x2)
sw x4,0(x1)
addi x2, x2, 4
addi x1, x1, 4
j loop
```

exit: ...

Monociclo: $T_{cpu} = 52000ps$
Multiciclo: $T_{cpu} = 50600ps$

Segmentado

```
lw x1,desp(x0)
lw x3,desp2(x0)
lw x4,desp3(x0)
lw x2,0(x1)
nop
loop: beq x3,x4,exit
nop
nop
lw x4,0(x2)
nop
nop
sw x4,0(x1)
j loop
addi x2, x2, 4
addi x1, x1, 4
```

exit: ...



EJEMPLO DE USO DE TÉCNICAS SW

- Tiempo de ciclo del cauce segmentado: 200ps
- Ciclos para llenar el cauce: 4 ciclos. Una vez lleno el cauce, CPI=1

```

lw x1,desp(x0)
lw x3,desp2(x0)
lw x4,desp3(x0)
lw x2,0(x1)
nop
loop: beq x3,x4,exit
nop
nop
lw x4,0(x2)
nop
nop
sw x4,0(x1)
j loop
addi x2, x2, 4
addi x1, x1, 4
exit: ...
    
```

5 instrucciones

10 iteraciones x 10 instrucciones
→ 100 instrucciones

Salida del bucle:
BEQ + 2 NOPs → 3 instrucciones

$$CPI_{ef} = \frac{NC}{NI - \#NOPs} = \frac{112}{108 - 43} = 1,72$$

$$NC = 4 + (5 + 10 \times 10 + 3) = 112 \text{ ciclos}$$

$$T_{CPU_{seg}} = NC \times T_{CLK} = 112 \times 200 = 22400 \text{ ps}$$

$$S = \frac{T_{CPU_{mono}}}{T_{CPU_{seg}}} = \frac{52000 \text{ ps}}{22400 \text{ ps}} = 2,32$$

$$S = \frac{T_{CPU_{multi}}}{T_{CPU_{seg}}} = \frac{50600 \text{ ps}}{22400 \text{ ps}} = 2,26$$

PROCESADOR SEGMENTADO: SOLUCIONES HARDWARE PARA LOS RIESGOS

- Resolución de riesgos RAW:
 - Parada
 - Anticipación
- Resolución de riesgos de control
 - Parada
 - Predicción estática y dinámica

RIESGOS DE DATOS RAW

- Soluciones hardware a los riesgos RAW: el HW garantiza la correcta ejecución del código
 - **Parada** (=detención, stall, inserción burbujas):
 - Detener la ejecución uno o más ciclos hasta que se resuelva el riesgo
 - Pérdida de ciclos de reloj
 - Implica la implementación de una lógica para insertar burbujas (\cong NOPs hardware)
 - **Anticipación** (=adelantamiento, forwarding, bypassing, activación de cortocircuitos)
 - Se activan caminos de adelantamiento o cortocircuitos implementados en el HW
 - Anticipar el resultado que produce una etapa en una instrucción a otra etapa que consume ese resultado para otra instrucción
 - No hay por qué esperar hasta que un resultado se escribe en el banco de registros
 - En cuanto está disponible se puede adelantar (anticipar) a la etapa que lo necesita: se lleva desde el registro de segmentación que lo contiene a la entrada de la unidad funcional consumidora
 - Reduce la pérdida de ciclos

PARADA DEL CAUCE

- Podemos hacer que el hardware **PARE** las instrucciones que producen el riesgo evitando tener que meter instrucciones NOPs
- La **solución** consiste en **parar (stall)** el pipeline uno o dos ciclos para retrasar la instrucción que necesita el dato (y sucesivas)
 - **Dos ciclos** si la instrucción que necesita el dato está a distancia 1

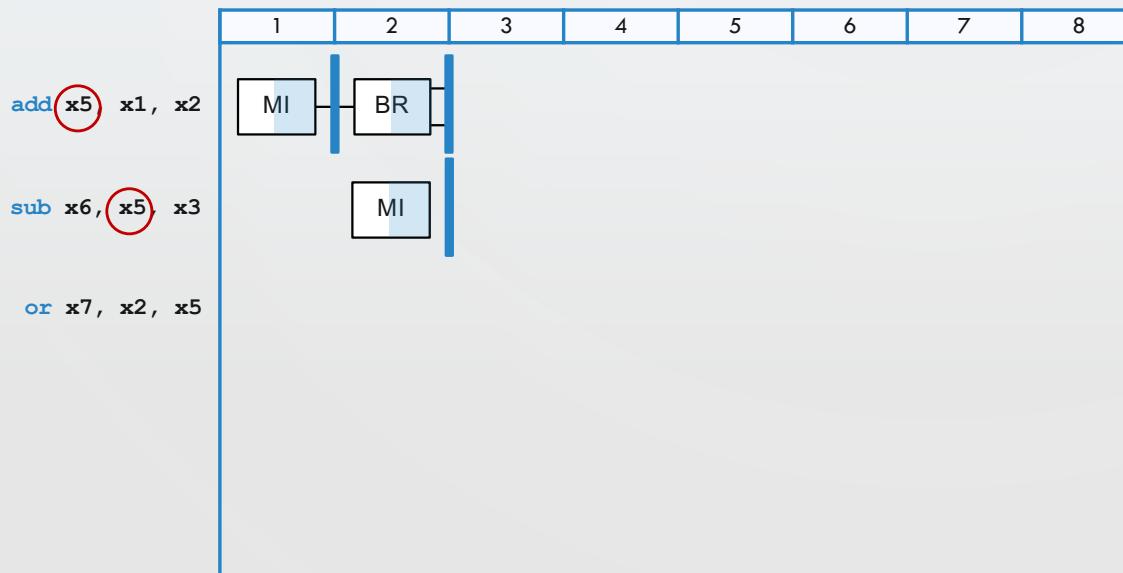
```
add x5, x1, x2
sub x6, x5, x3
or  x7, x2, x5
```

- **Un ciclo** si está a distancia 2

```
add x5, x1, x2
sub x6, x7, x3
or  x7, x2, x5
```

EJEMPLO DE PARADA

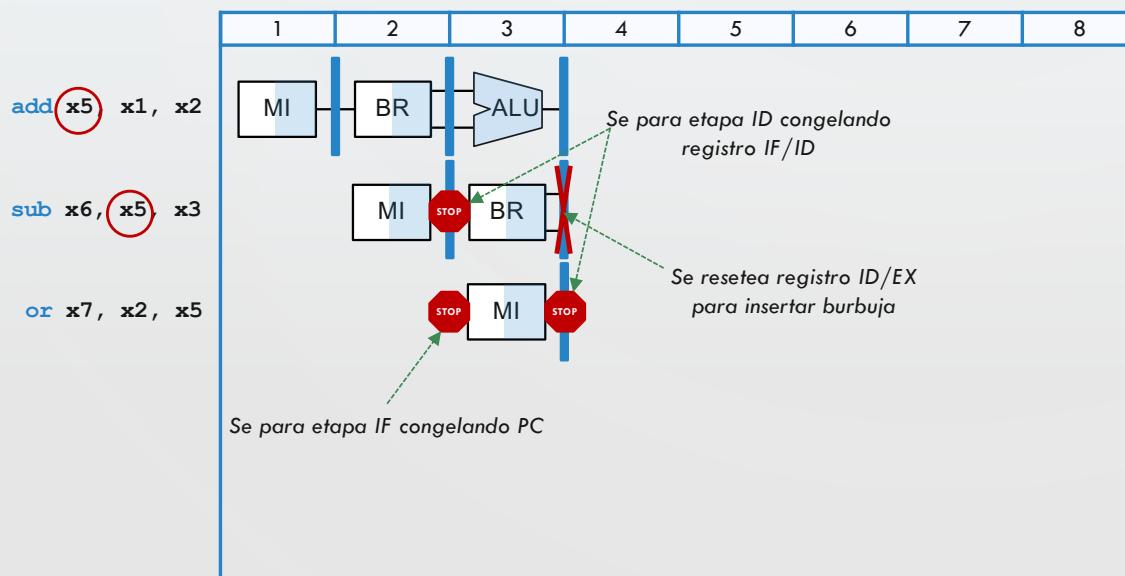
- Ejemplo de instrucción que usa dato generado en instrucción anterior
- La **solución** consiste en **parar (stall)** el pipeline dos ciclos para retrasar la instrucción que necesita el dato (y sucesivas)



TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR

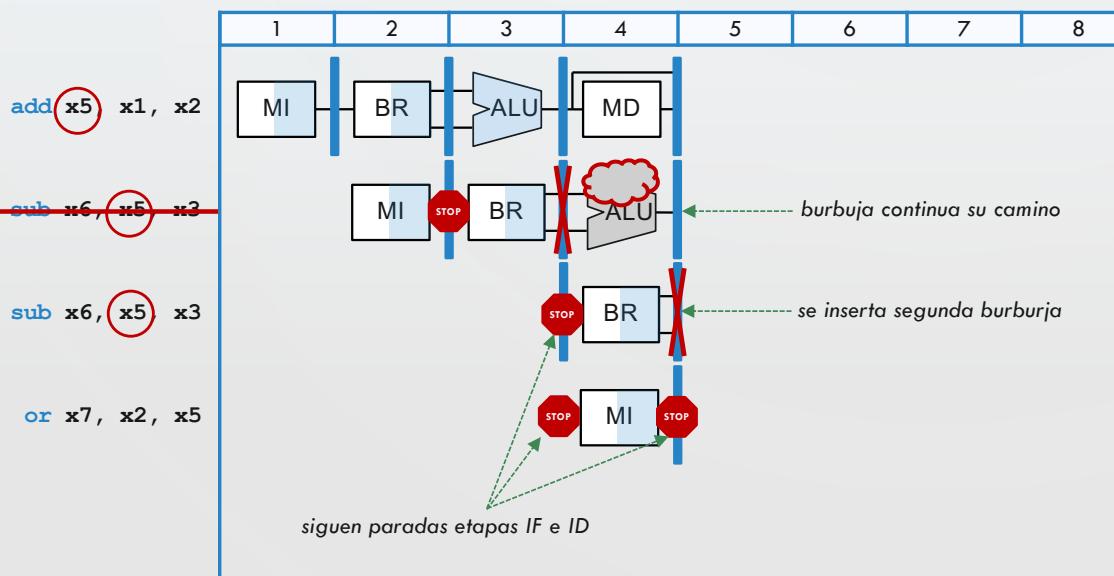
EJEMPLO DE PARADA

- Ejemplo de instrucción que usa dato generado en instrucción anterior
- La **solución** consiste en **parar (stall)** el pipeline dos ciclos para retrasar la instrucción que necesita el dato (y sucesivas)
 - Ciclo 3 (**sub** está en ID): se detecta el conflicto. Las instrucciones **sub** y **or** se **paran** y se inserta una “burbuja” de no operación en la etapa EX



EJEMPLO DE PARADA

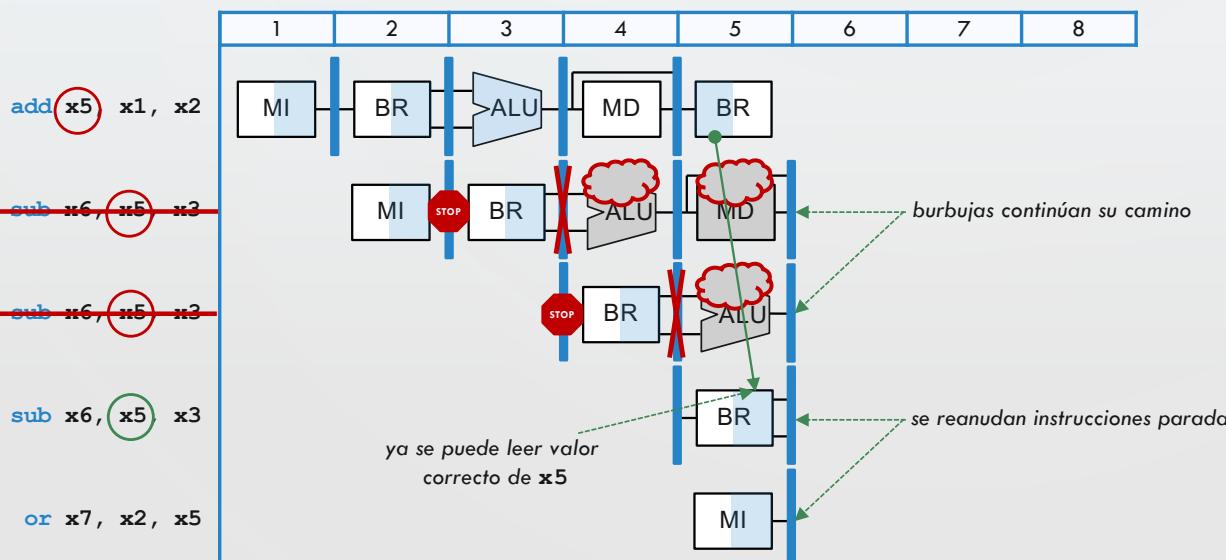
- Ejemplo de instrucción que usa dato generado en instrucción anterior
- La **solución** consiste en **parar (stall)** el pipeline dos ciclos para retrasar la instrucción que necesita el dato (y sucesivas)
 - **Ciclo 4:** la instrucción **add** está en la etapa **MEM**, las instrucciones **sub** y **or** siguen paradas y se inserta otra “burbuja”



TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR

EJEMPLO DE PARADA

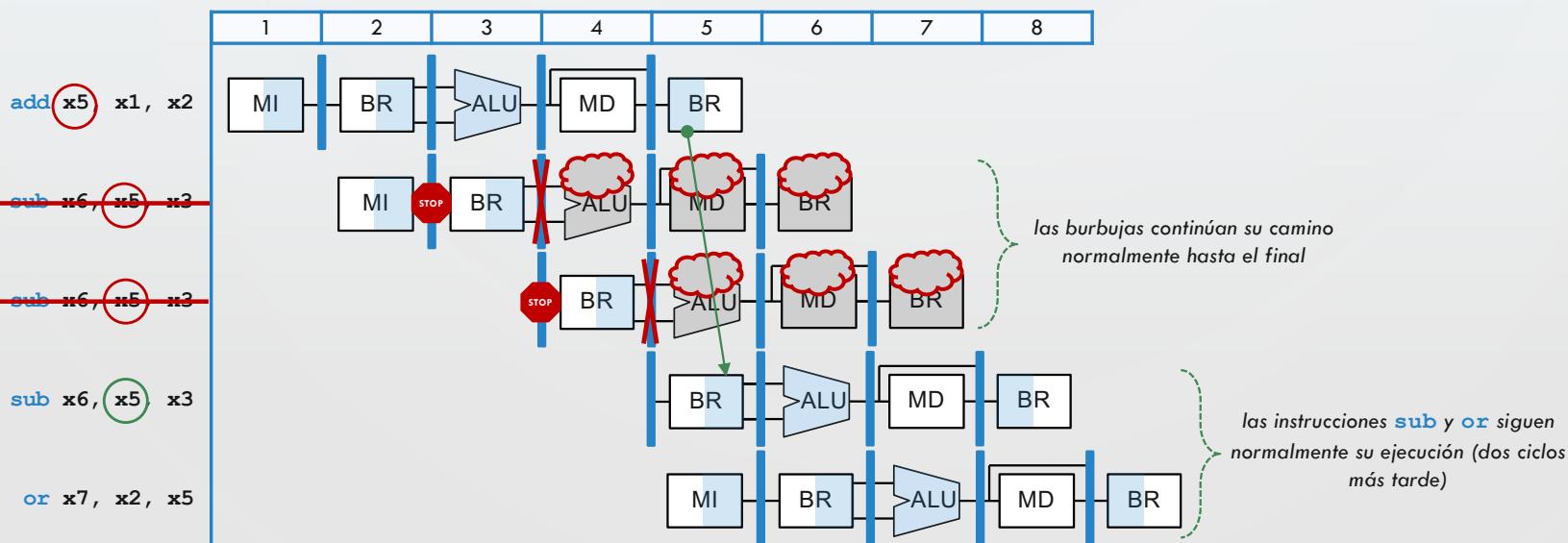
- Ejemplo de instrucción que usa dato generado en instrucción anterior
- La **solución** consiste en **parar (stall)** el pipeline dos ciclos para retrasar la instrucción que necesita el dato (y sucesivas)
 - Ciclo 5: la instrucción **add** está en la etapa WB y puede anticipar el dato a la etapa BR de **sub**, por lo que **sub y or** se reanudan



TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR

EJEMPLO DE PARADA

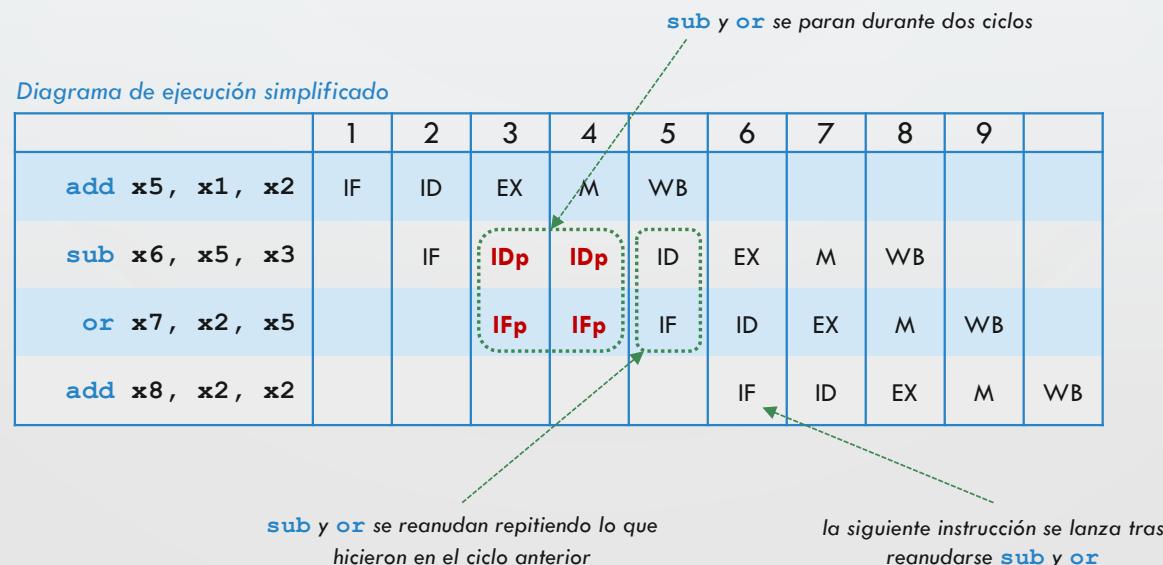
- Ejemplo de instrucción que usa dato generado en instrucción anterior
- La solución consiste en parar (stall) el pipeline dos ciclos para retrasar la instrucción que necesita el dato (y sucesivas)
 - Ciclos sucesivos: el pipeline avanza normalmente



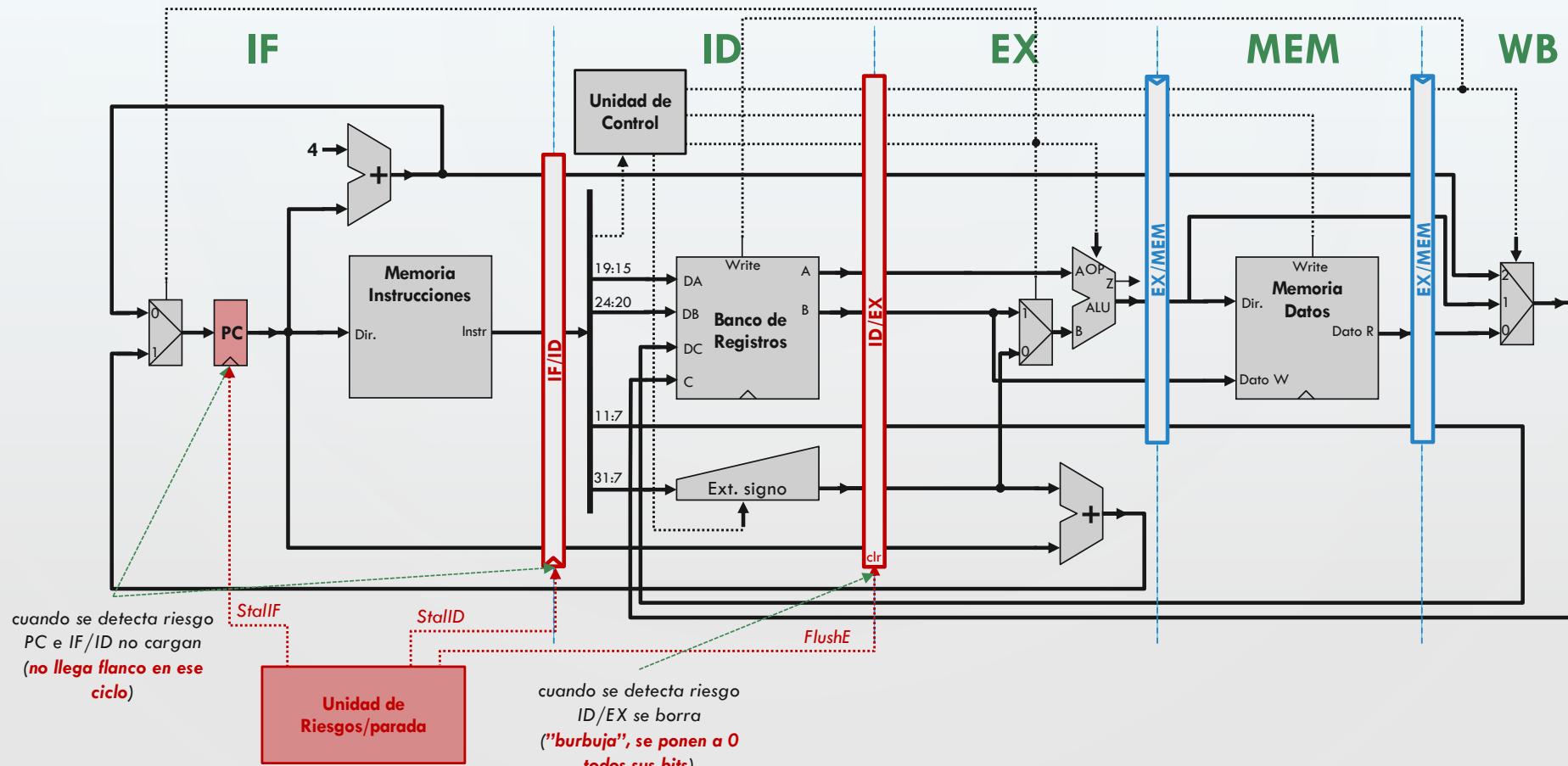
TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR

DIAGRAMA DE EJECUCIÓN DE LA PARADA

- En **diagramas de ejecución simplificados**, las paradas se indican marcando las etapas que paran (las “burbujas” quedan implícitas)



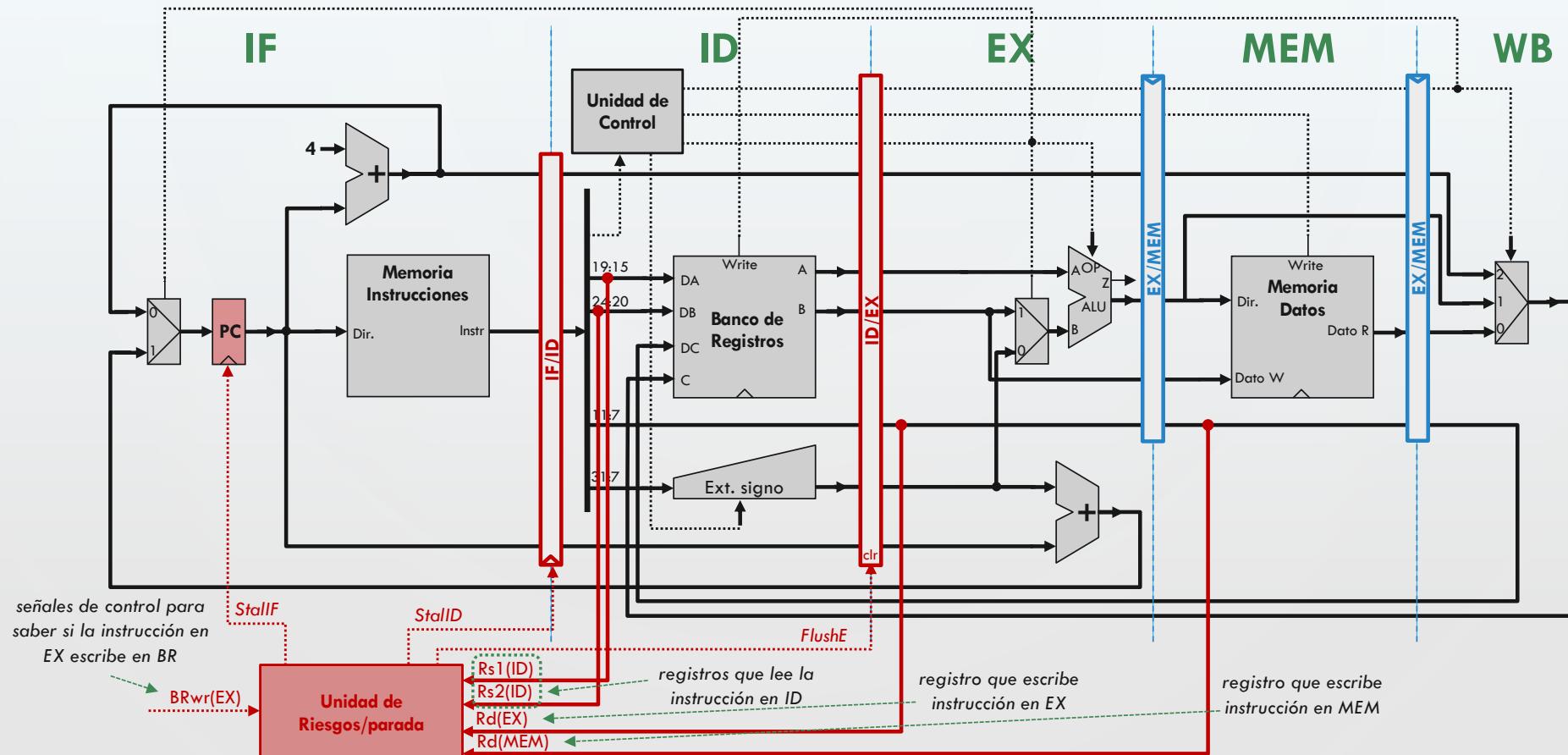
HARDWARE PARA LA PARADA



TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR



HARDWARE PARA LA DETECCIÓN Y PARADA



TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR



EJEMPLO DE DIAGRAMA DE EJECUCIÓN DE LA PARADA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8													
add x4, x3, x2													
lw x5, 20(x4)													
sub x7, x5, x7													
or x3, x1, x2													

EJEMPLO DE DIAGRAMA DE EJECUCIÓN DE LA PARADA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX										
add x4, x3, x2		IF	ID										
lw x5, 20(x4)			IF										
sub x7, x5, x7													
or x3, x1, x2													

EJEMPLO DE DIAGRAMA DE EJECUCIÓN DE LA PARADA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M									
add x4, x3, x2		IF	ID	EX									
lw x5, 20(x4)			IF	IDp									
sub x7, x5, x7				IFp									
or x3, x1, x2													

EJEMPLO DE DIAGRAMA DE EJECUCIÓN DE LA PARADA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX	M								
lw x5, 20(x4)			IF	IDp	IDp								
sub x7, x5, x7				IFp	IFp								
or x3, x1, x2													

EJEMPLO DE DIAGRAMA DE EJECUCIÓN DE LA PARADA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX	M	WB x4							
lw x5, 20(x4)			IF	IDp	IDp	ID							
sub x7, x5, x7				IFp	IFp	IF							
or x3, x1, x2													

EJEMPLO DE DIAGRAMA DE EJECUCIÓN DE LA PARADA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX	M	WB x4							
lw x5, 20(x4)			IF	IDp	IDp	ID	EX						
sub x7, x5, x7				IFp	IFp	IF	IDp						
or x3, x1, x2							IFp						

EJEMPLO DE DIAGRAMA DE EJECUCIÓN DE LA PARADA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX	M	WB x4							
lw x5, 20(x4)			IF	IDp	IDp	ID	EX	M	WB x5				
sub x7, x5, x7				IFp	IFp	IF	IDp	IDp	ID	EX	M	WB	
or x3, x1, x2						IFp	IFp	IF	ID	EX	M	WB	

Tiempo de ejecución = 13 ciclos

EJEMPLO DE DIAGRAMA DE EJECUCIÓN DE LA PARADA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX	M	WB x4							
lw x5, 20(x4)			IF	IDp	IDp	ID	EX	M	WB x5				
sub x7, x5, x7				IFp	IFp	IF	IDp	IDp	ID	EX	M	WB	
or x3, x1, x2						IFp	IFp	IF	ID	EX	M	WB	

Tiempo de ejecución = 13 ciclos

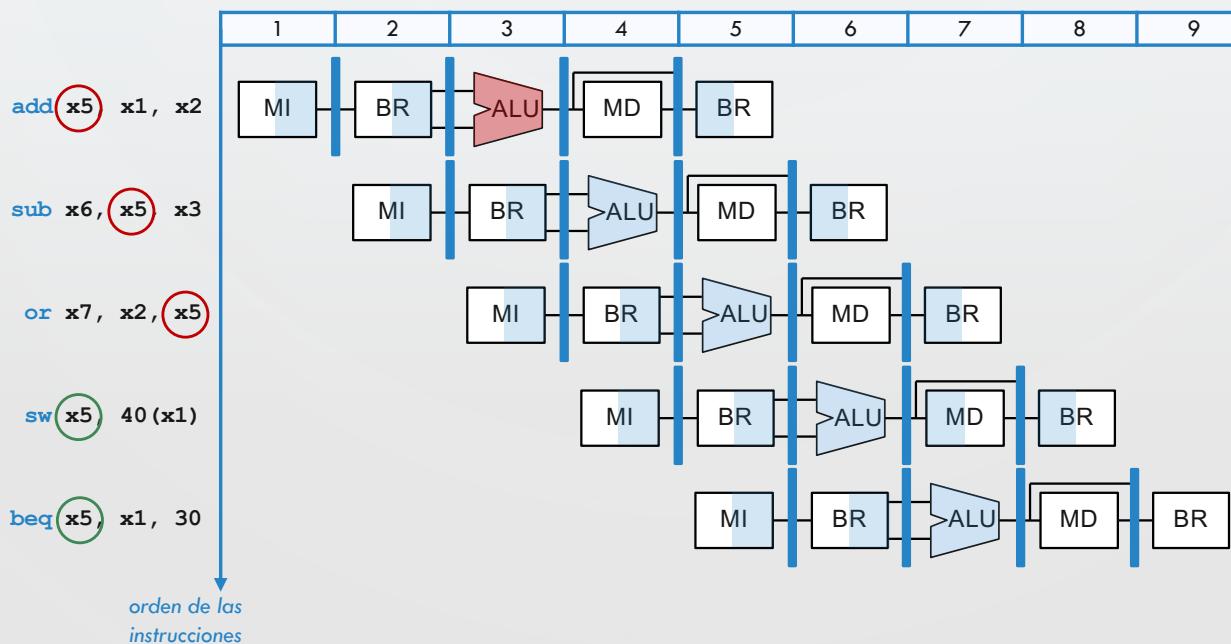
Diagrama de ejecución simplificado (reordenando)

	1	2	3	4	5	6	7	8	9	10	11	12	13
add x4, x3, x2	IF	ID	EX	M	WB x4								
and x7, x6, x8		IF	ID	EX	M	WB							
lw x5, 20(x4)			IF	IDp	ID	EX	M	WB x5					
or x3, x1, x2				IFp	IF	ID	EX	M	WB				
sub x7, x5, x7					IF	IDp	ID	EX	M	WB			

Tiempo de ejecución = 11 ciclos

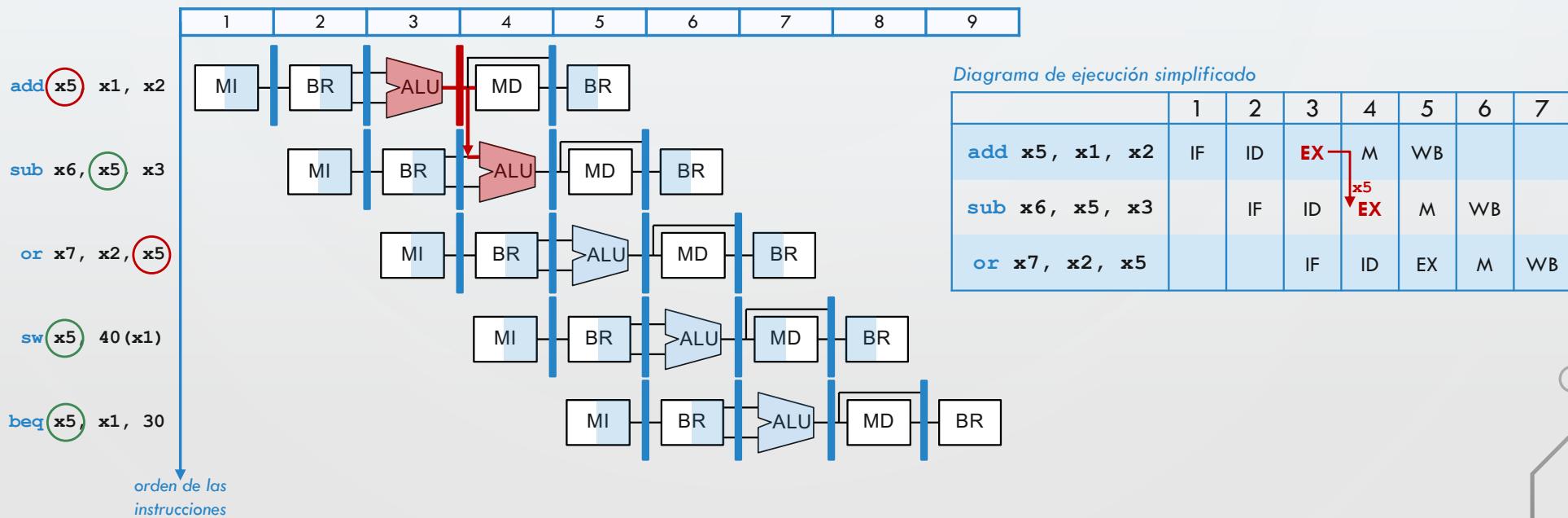
ANTICIPACIÓN

- Existe una **solución hardware** que **evita la penalización** de las paradas ya que:
 - La instrucción **add** usa la ALU en el ciclo 3 para calcular la suma (que se almacenará en **x5**)
 - Las instrucciones siguientes necesitan el dato en los ciclos 4, 5 (posteriores al 3)
 - **El dato está disponible desde el ciclo 4**, y **puede anticiparse** (*forward*) sin necesidad de esperar a que pueda leerse del BR



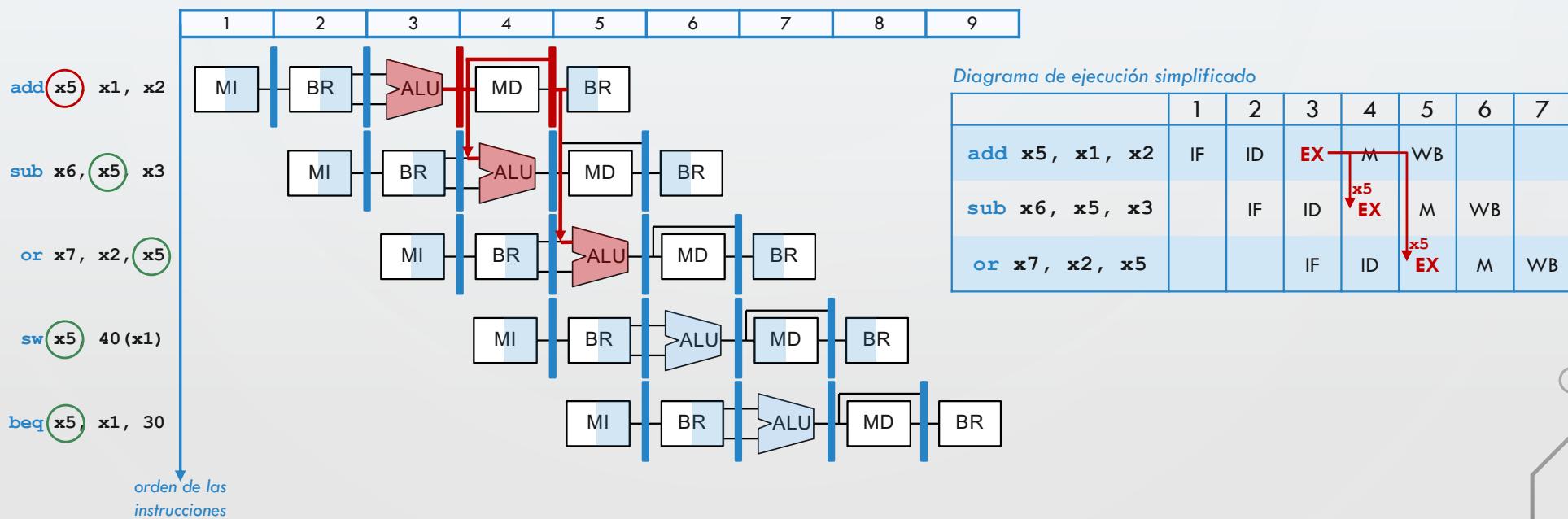
ANTICIPACIÓN EJECUCIÓN-EJECUCIÓN

- **add-sub:** se añaden caminos para **anticipar** el dato desde el registro de segmentación **EX/MEM** a cualquiera de las entradas de la ALU (**etapa EX**)



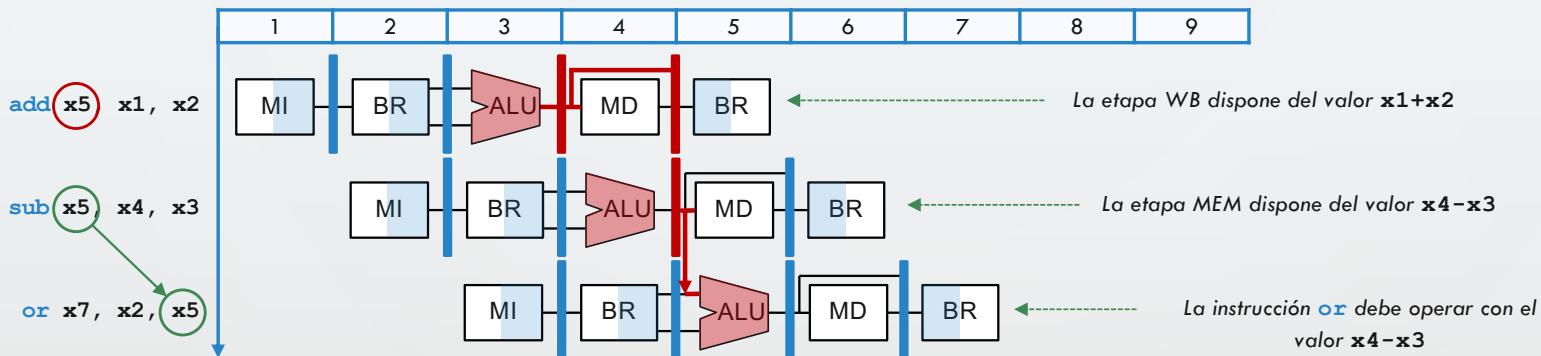
ANTICIPACIÓN MEMORIA-EJECUCIÓN

- **add-or**: se añaden caminos para **anticipar** el dato desde el registro de segmentación **MEM/WB** a cualquiera de las entradas de la ALU (**etapa EX**)

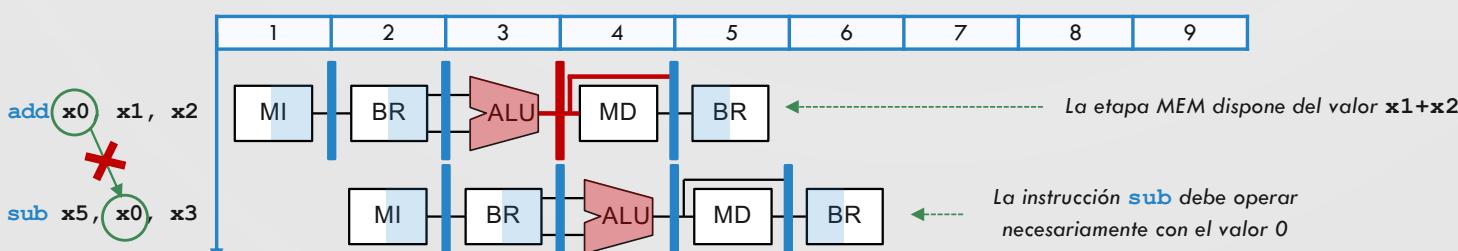


ANTICIPACIÓN DESDE MEMORIA O DESDE EJECUCIÓN

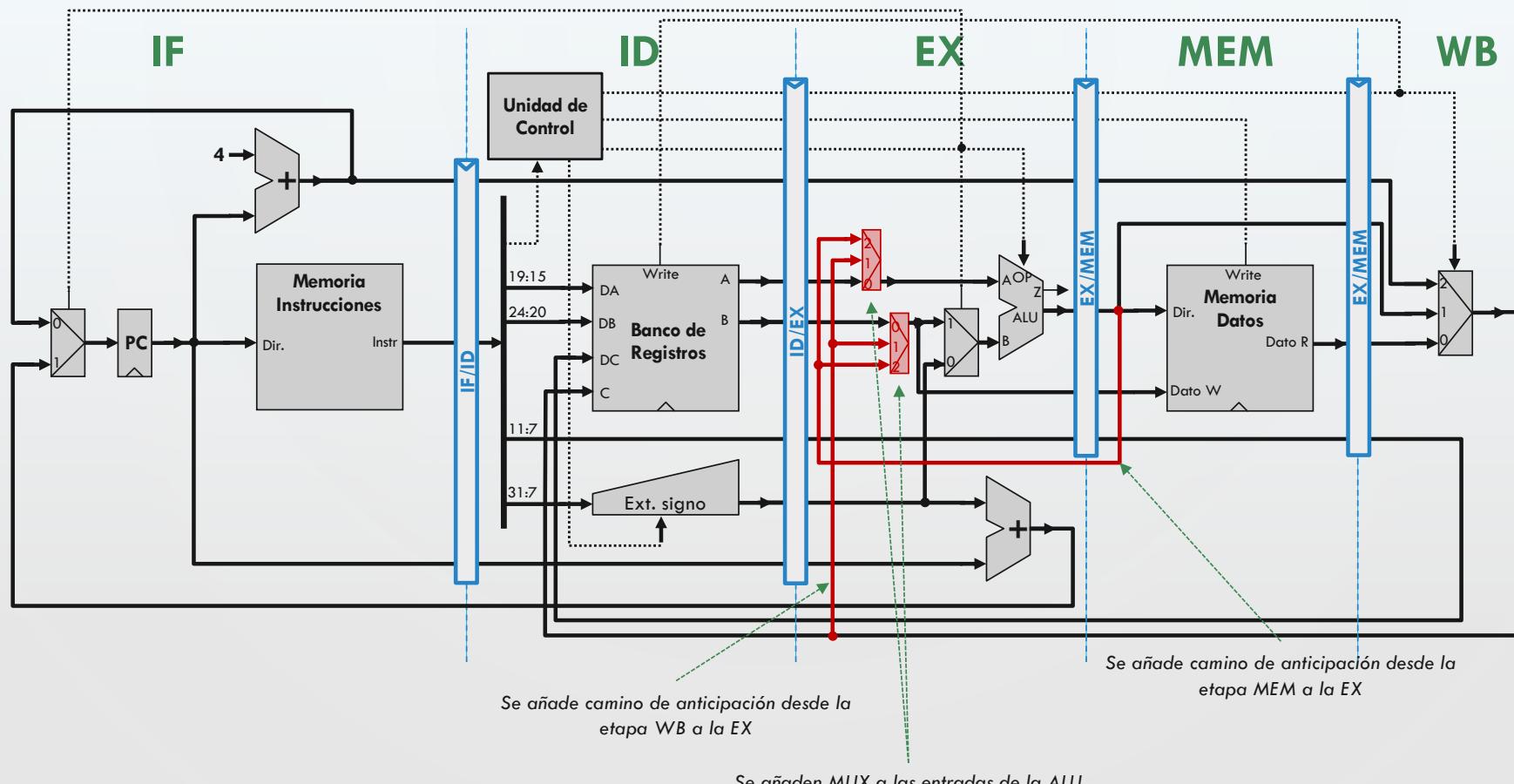
- En caso de que un registro pueda anticiparse desde EX/MEM y MEM/WB:
 - Deberá hacerlo desde MEM/WB por disponer ésta el valor más actualizado



- Además, el registro **x0** nunca deberá anticiparse



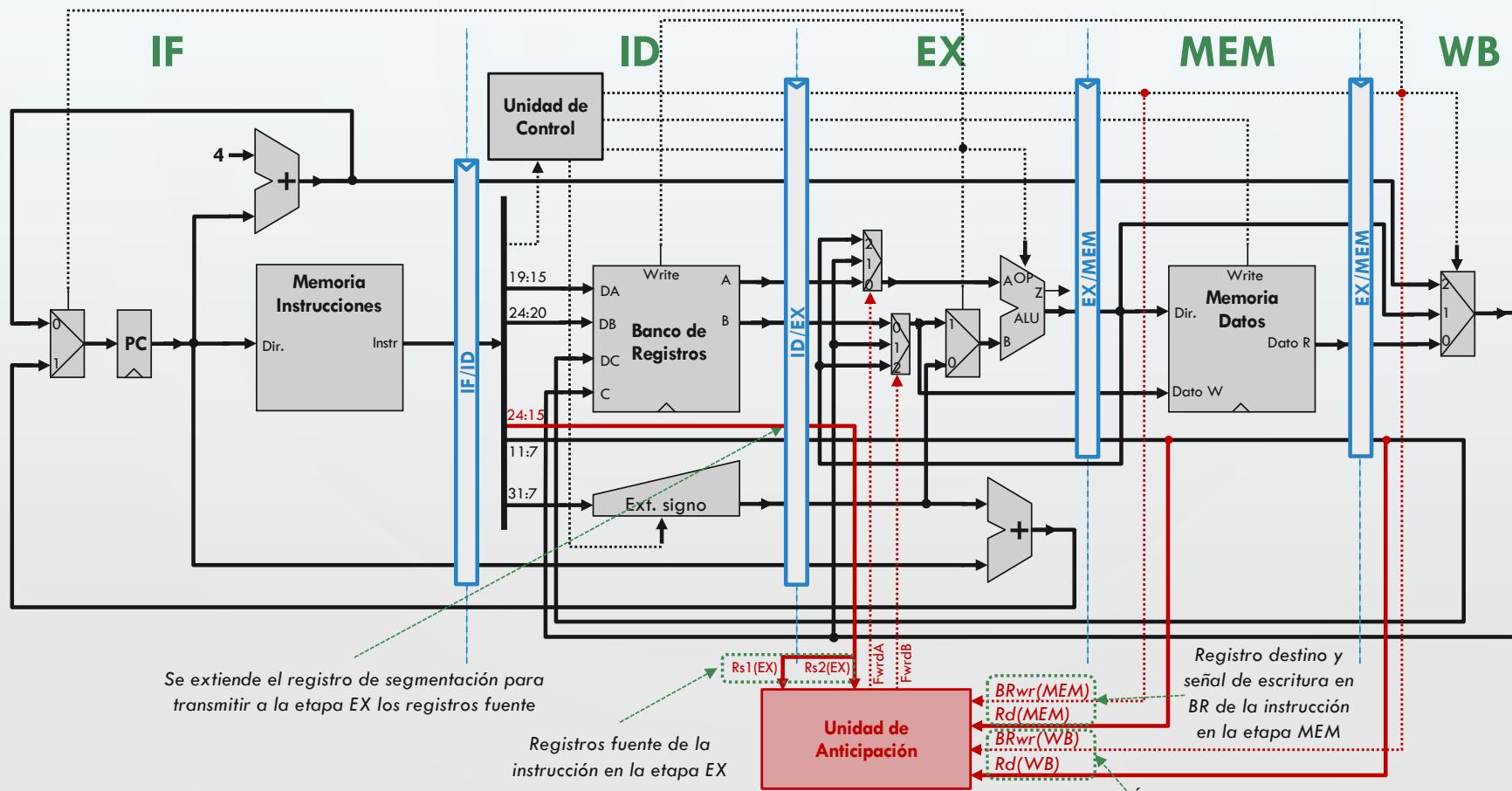
HARDWARE PARA LA ANTICIPACIÓN



UNIDAD DE ANTICIPACIÓN

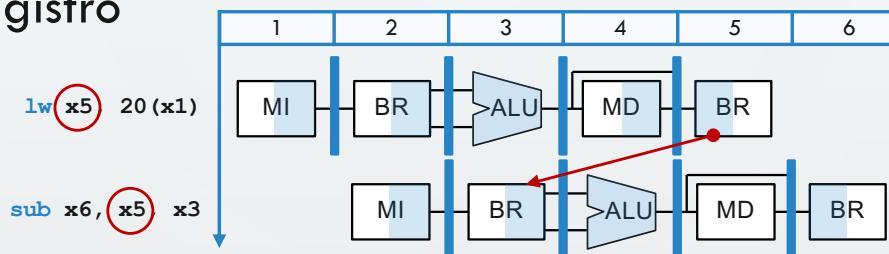
- Circuito combinacional que **controla los MUX de anticipación** para que la ALU opere con datos producidos en las etapas EX o MEM
- Para decidir de dónde provienen los datos **necesita conocer**:
 - Registro fuente 1 de la instrucción en la etapa EX
 - Registro fuente 2 de la instrucción en la etapa EX
 - Registro destino de la instrucción en la etapa MEM
 - Si la instrucción en la etapa MEM escribe en el BR
 - Registro destino de la instrucción en la etapa WB
 - Si la instrucción en la etapa WB escribe en el BR

HARDWARE PARA LA ANTICIPACIÓN

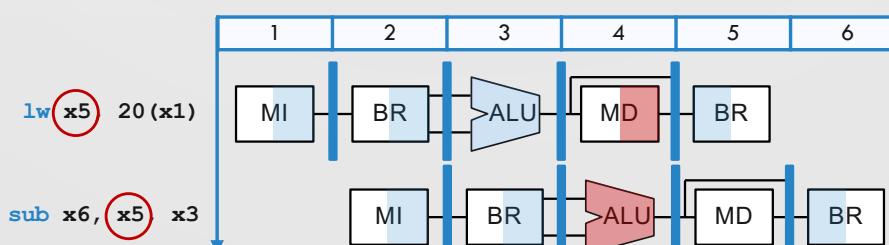


TRATAMIENTO DE RIESGOS LOAD-USO

- Cuando a una instrucción `lw` que carga un registro le sigue otra instrucción que lee ese mismo registro



- El dato requerido **no puede anticiparse** porque:
 - La instrucción `lw` lee el dato de memoria en el ciclo 4
 - La instrucción siguiente necesita el dato en ese mismo ciclo



TRATAMIENTO DE RIESGOS LOAD-USO

- La solución consiste en parar (**stall**) el pipeline un ciclo para retrasar la instrucción que necesita el dato (y sucesivas) y este pueda anticiparse
 - Ciclo 3 (**sub** está en ID): se detecta el conflicto. Las instrucciones **sub** y **or** se paran y se inserta una “burbuja” de no operación en la etapa EX
 - Ciclo 4: la instrucción **lw** lee el dato de memoria y **sub** y **or** se reanudan
 - Ciclo 5: el dato se anticipa desde la etapa WB de **lw** a la etapa EX de **sub**
 - Ciclos sucesivos: el pipeline avanza normalmente
- Penaliza la ejecución del programa en **un ciclo por riesgo lw**

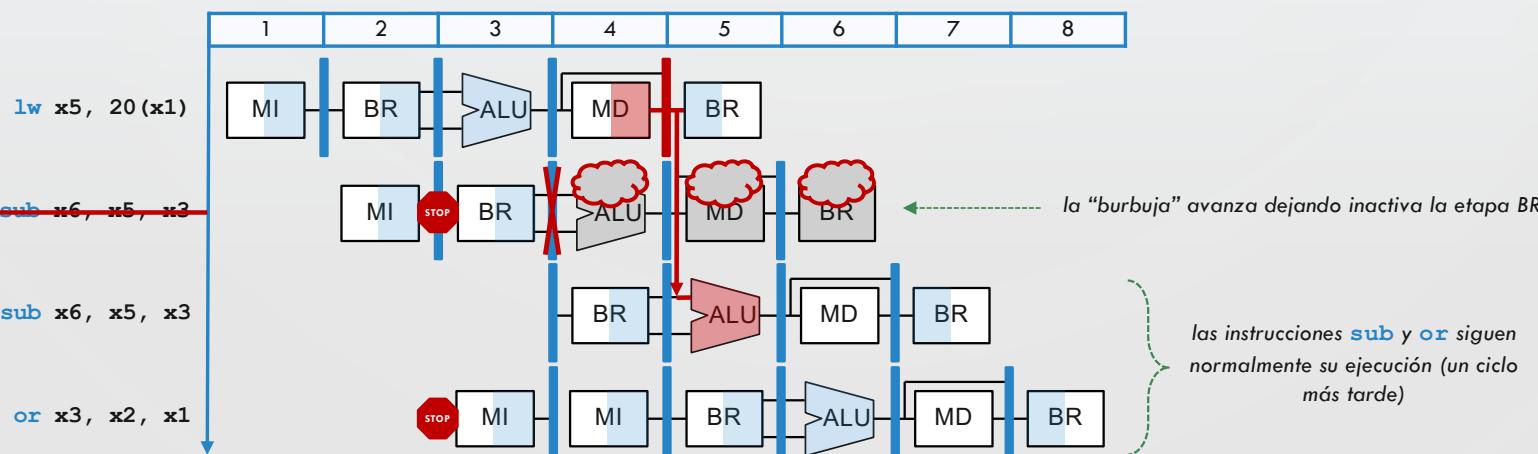
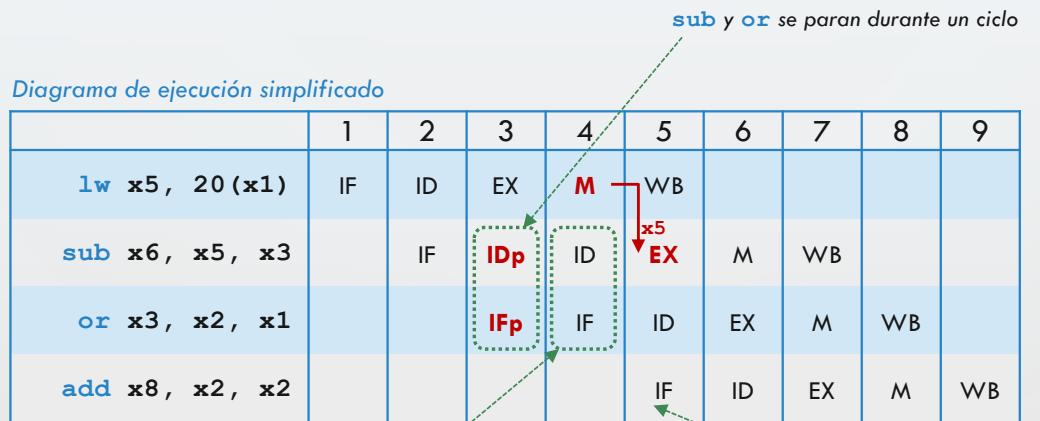
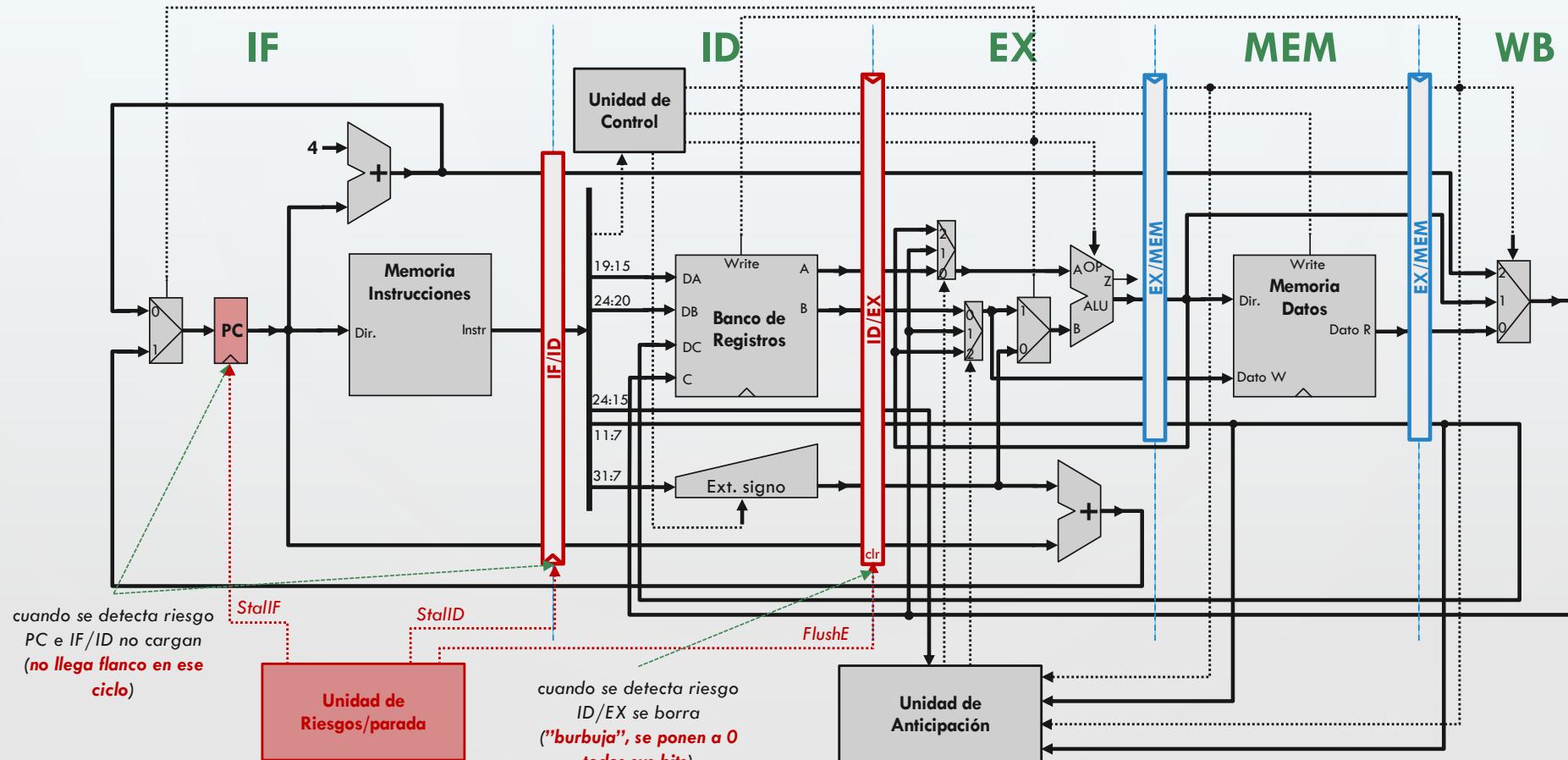


DIAGRAMA DE EJECUCIÓN DE RIESGOS LOAD-USO

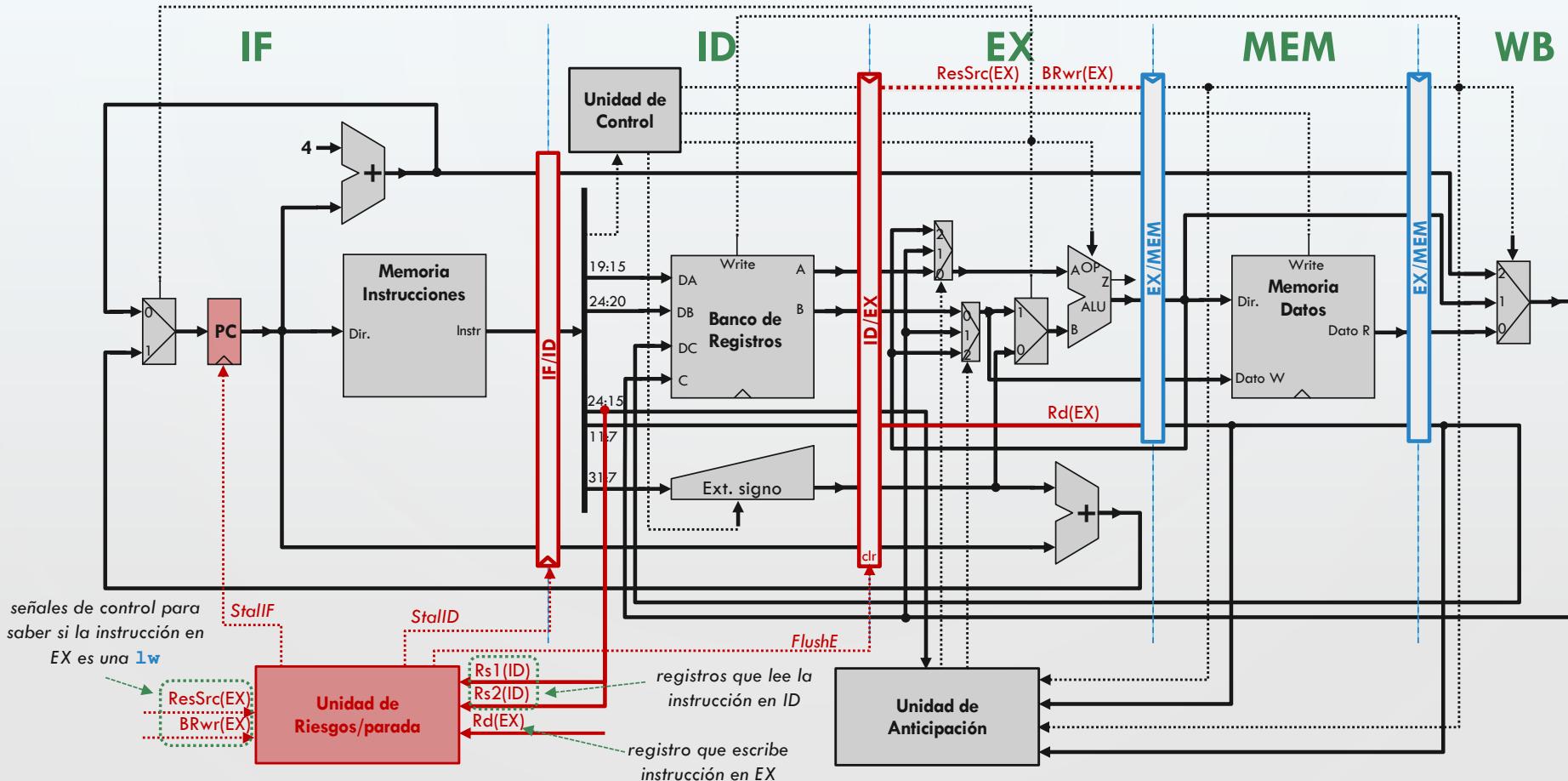
- En **diagramas de ejecución simplificados**, las paradas se indican marcando las etapas que paran (las “burbujas” quedan implícitas)



HARDWARE PARA LA DETECCIÓN, PARADA Y ANTICIPACIÓN



HARDWARE PARA LA DETECCIÓN, PARADA Y ANTICIPACIÓN



CASOS ESPECIALES DE RIESGOS LOAD-USO

- La solución presentada realiza, en algunos casos, **paradas innecesarias**
- En los siguientes casos **no es necesario realizar paradas**:
 - Cuando **x0** es el registro destino de la carga de memoria: **lw x0 , 20(x1)**
 - Cuando **a la instrucción lw le sigue una instrucción sw que almacena en memoria el mismo registro cargado por la primera**
 - Es un caso común que se usa, por ejemplo, para copiar arrays
 - El **dato, disponible desde el ciclo 5**, puede anticiparse de **MEM/WB → MEM** sin parar

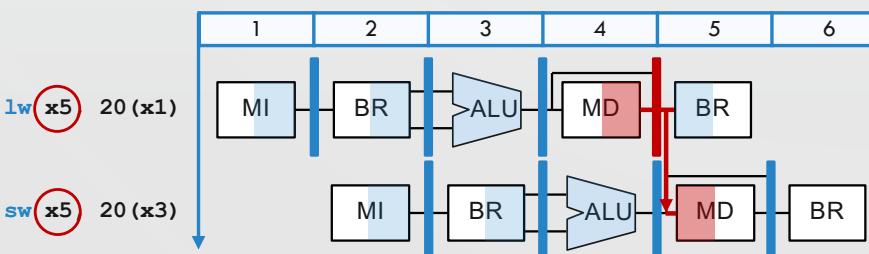


Diagrama de ejecución simplificado

	1	2	3	4	5	6
lw x5, 20(x1)	IF	ID	EX	M	WB	
sw x5, 20(x3)		IF	ID	EX	x5 M	WB

EJEMPLO DE ANTICIPACIÓN

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2													
lw x5, 20(x4)													
sub x7, x5, x7													
or x3, x1, x2													

EJEMPLO DE ANTICIPACIÓN

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX	M	WB							
lw x5, 20(x4)													
sub x7, x5, x7													
or x3, x1, x2													

EJEMPLO DE ANTICIPACIÓN

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX	* ^{x4} M	WB							
lw x5, 20(x4)			IF	ID	EX	M	WB						
sub x7, x5, x7													
or x3, x1, x2													

EJEMPLO DE ANTICIPACIÓN

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX ^{*4}	M	WB							
lw x5, 20(x4)			IF	ID	EX	M ^{*5}	WB						
sub x7, x5, x7				IF	IDp	EX	M	WB					
or x3, x1, x2													

EJEMPLO DE ANTICIPACIÓN

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX ^{*4}	M	WB							
lw x5, 20(x4)			IF	ID	EX ^{*5}	M	WB						
sub x7, x5, x7				IF	IDp	ID	EX	M	WB				
or x3, x1, x2					IFp	IF	ID	EX	M	WB			

Tiempo de ejecución = 10 ciclos
(Tiempo con NOPs = 13 ciclos,
Tiempo con reordenación y NOPs = 11 ciclos)

REORDENACIÓN + ANTICIPACIÓN

- Dado un programa ensamblador, las **paradas por riesgo de datos** con instrucciones **lw** son **inevitables por HW**
 - Pero **pueden evitarse por SW** reordenando el código para que a una instrucción **lw** nunca le siga una instrucción que utilice el registro que carga
 - Esta es una de las **optimizaciones** que aplican los compiladores

C

```
...
int a, b, c;
int d, e;
...
c = a + b;
e = d + b;
...
```

a → x1 d → x4
b → x2 e → x5
c → x3

asignación de variables a registros

ASM

```
...
lw x1, 0(x31)
lw x2, 4(x31)
add x3, x1, x2
sw x3, 8(x31)
lw x4, 12(x31)
add x5, x4, x2
sw x5, 16(x31)
...
```

2 paradas
(13 ciclos)

ASM

```
...
lw x1, 0(x31)
lw x2, 4(x31)
lw x4, 12(x31)
add x3, x1, x2
sw x3, 8(x31)
add x5, x4, x2
sw x5, 16(x31)
...
```

0 paradas
(11 ciclos)

DIAGRAMA DE EJECUCIÓN DE REORDENACIÓN + ANTICIPACIÓN

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos. Reduce el número de ciclos utilizando reordenación.

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX ^{*4}	M	WB							
lw x5, 20(x4)			IF	ID	EX	M ^{*5}	WB						
sub x7, x5, x7				IF	IDp	ID	EX	M	WB				
or x3, x1, x2				IFp	IF	ID	EX	M	WB				

Tiempo de ejecución = 10 ciclos
(Tiempo con NOPs = 13 ciclos,
Tiempo con reordenación y NOPs = 11 ciclos)



DIAGRAMA DE EJECUCIÓN DE REORDENACIÓN + ANTICIPACIÓN

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos. Reduce el número de ciclos utilizando reordenación.

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX ^{x4}	M	WB							
lw x5, 20(x4)			IF	ID	EX	M ^{x5}	WB						
sub x7, x5, x7				IF	IDp	ID	EX	M	WB				
or x3, x1, x2				IFp	IF	ID	EX	M	WB				

Tiempo de ejecución = 10 ciclos

(Tiempo con NOPs = 13 ciclos,

Tiempo con reordenación y NOPs = 11 ciclos)

	1	2	3	4	5	6	7	8	9	10	11	12	13
and x7, x6, x8	IF	ID	EX	M	WB								
add x4, x3, x2		IF	ID	EX ^{x4}	M	WB							
lw x5, 20(x4)			IF	ID	EX	M ^{x5}	WB						
or x3, x1, x2				IF	ID	EX	M	WB					
sub x7, x5, x7				IF	ID	EX	M	WB					

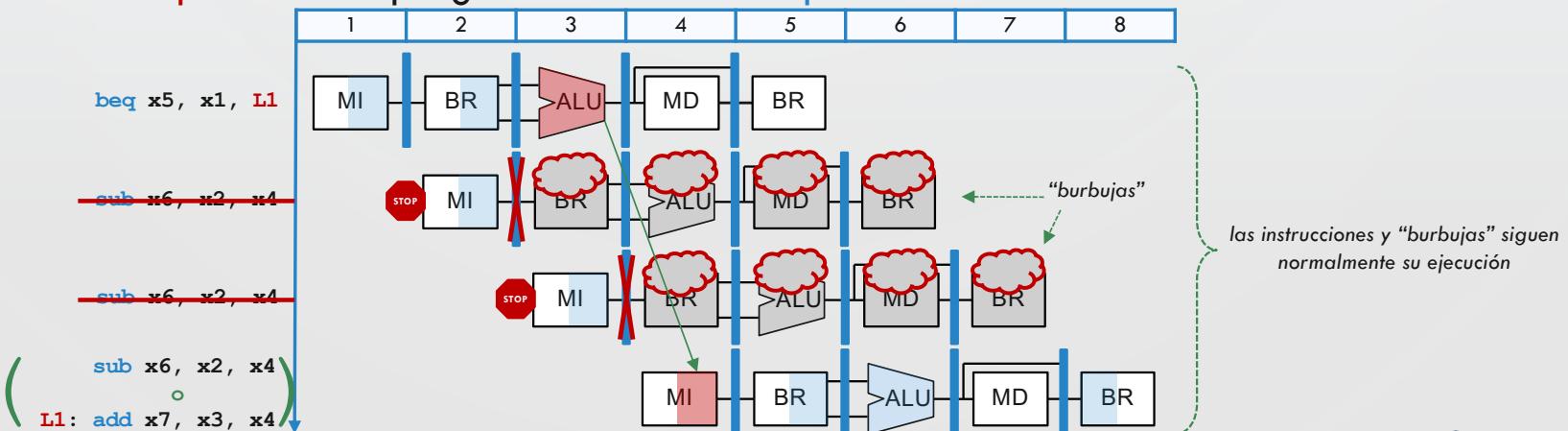
Tiempo de ejecución = 9 ciclos

RIESGOS DE CONTROL

- Soluciones Hardware: ni el programador ni el compilador se tienen que preocupar de la correcta ejecución del código
 - Detener el cauce segmentado (actualización de PC) tantos ciclos como requiera la etapa de terminación del salto
 - No mejora el rendimiento (la espera la realiza el HW)
 - Podemos reducir la penalización modificando el hardware para adelantar la etapa de resolución de los saltos
 - Como mucho a la etapa ID
 - Puede afectar a la resolución de los riesgos RAW
 - Puede afectar al tiempo de ciclo al meter nuevos retardos en la etapa ID (junto con los de control)
 - Mejora el rendimiento pues reduce los ciclos de espera
 - Predictores de saltos
 - Sólo hay penalización cuando el predictor falla, en caso de acierto no se pierden ciclos
 - Estáticos: siempre hacemos la misma predicción (salto tomado/salto no tomado)
 - Dinámicos: la predicción cambia de forma dinámica

PARADA DEL CAUCE

- La solución HW consiste en parar (stall) el pipeline durante dos ciclos para retrasar la ejecución de nuevas instrucciones hasta que el salto esté resuelto
 - Ciclo 2 (**beq** está en ID): se detecta el conflicto. La instrucción **sub** se para y se inserta una “burbuja” de no operación en la etapa ID
 - Ciclo 3 (**beq** está en EX): el conflicto continúa. La instrucción **sub** permanece parada, se inserta otra “burbuja” y la instrucción **beq** resuelve el salto
 - Ciclo 4 (**beq** está en MEM): se lanza la instrucción que corresponda
- Penaliza la ejecución del programa en dos ciclos por instrucción de salto



PARADA DEL CAUCE

- La solución HW consiste en parar (stall) el pipeline durante dos ciclos para retrasar la ejecución de nuevas instrucciones hasta que el salto esté resuelto
 - Ciclo 2 (**beq** está en ID): se detecta el conflicto. La instrucción **sub** se para y se inserta una “burbuja” de no operación en la etapa ID
 - Ciclo 3 (**beq** está en EX): el conflicto continúa. La instrucción **sub** permanece parada, se inserta otra “burbuja” y la instrucción **beq** resuelve el salto
 - Ciclo 4 (**beq** está en MEM): se lanza la instrucción que corresponda
- Penaliza la ejecución del programa en dos ciclos por instrucción de salto

Diagrama de ejecución simplificado (condición falsa)

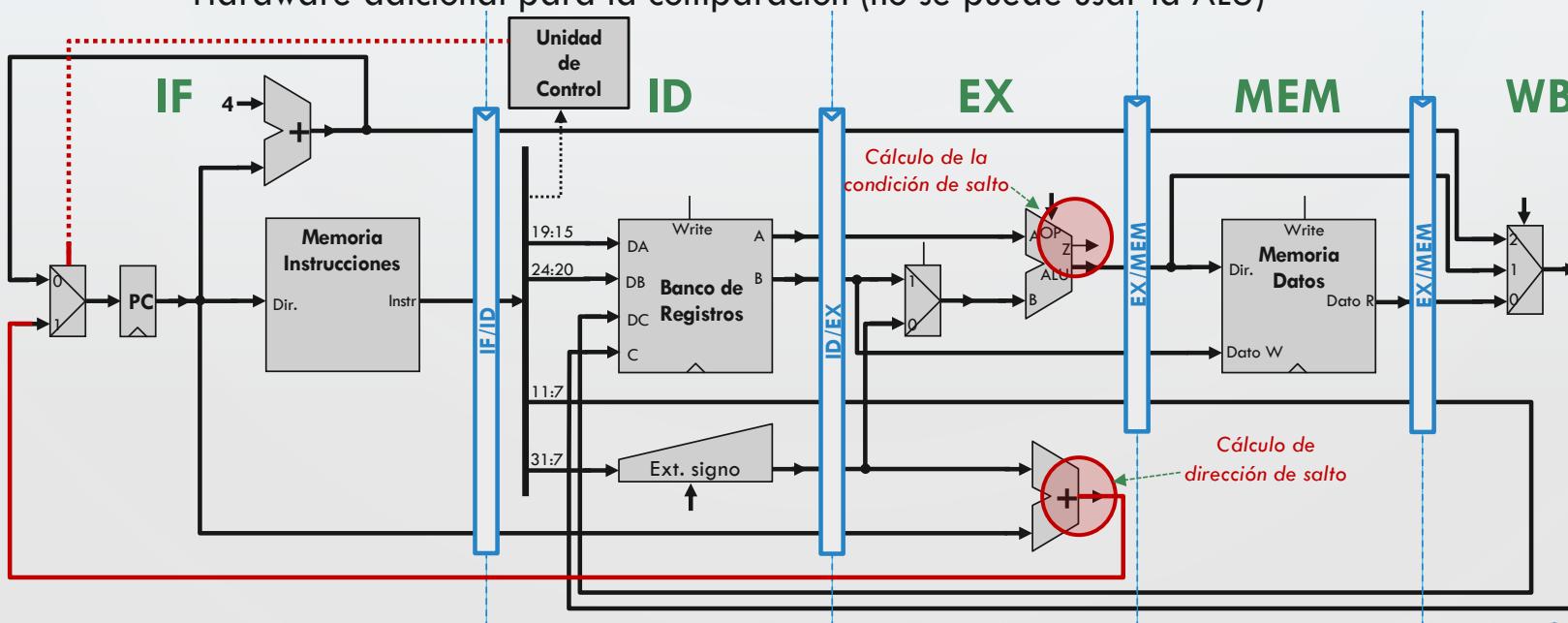
	1	2	3	4	5	6	7	8
beq x5, x1, L1	IF	ID	EX	M	WB			
sub x6, x2, x4		IFp	IFp	IF	ID	EX	M	WB

Diagrama de ejecución simplificado (condición verdadera)

	1	2	3	4	5	6	7	8
beq x5, x1, L1	IF	ID	EX	M	WB			
sub x6, x2, x4		IFp	IFp	X				
L1: add x7, x3, x4				IF	ID	EX	M	WB

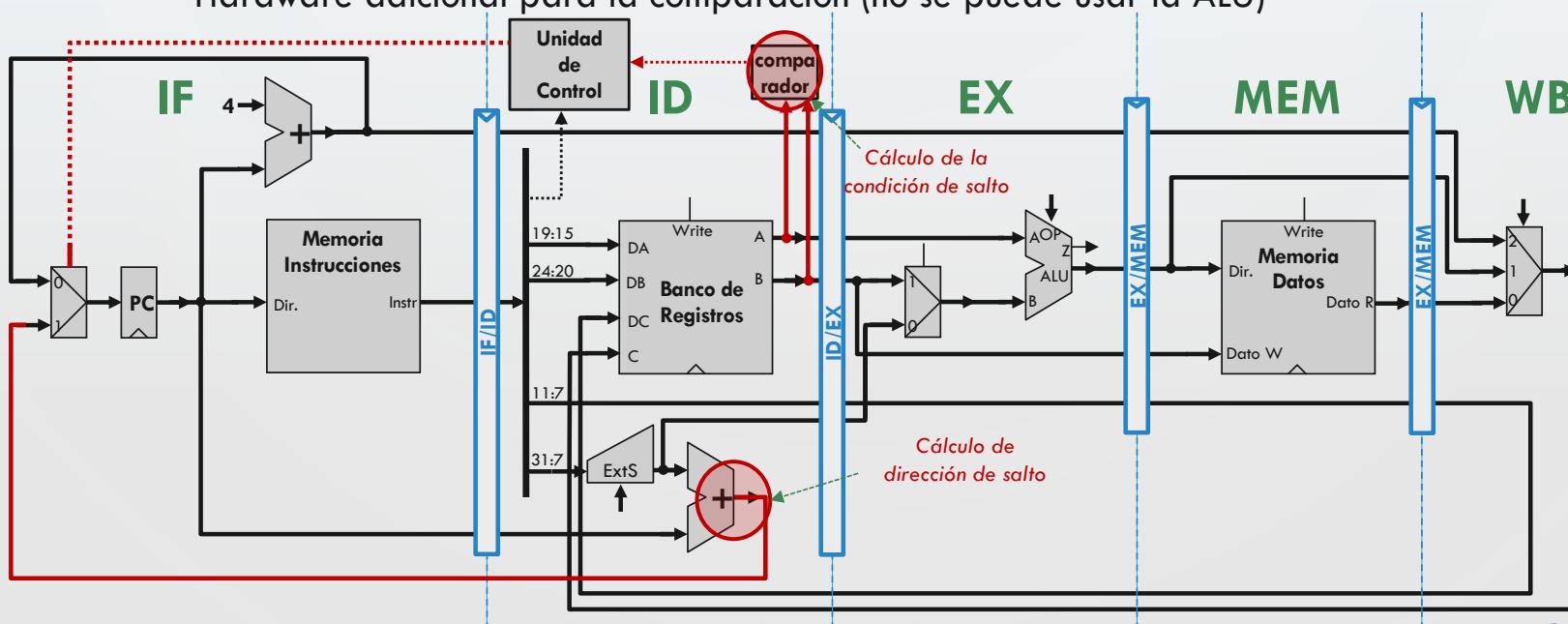
HARDWARE PARA ADELANTAR SALTOS A ID

- Intentar mover los saltos a etapas más tempranas: por ejemplo, a ID
 - Mover cálculo de la dirección: desplazar sumador a etapa ID
 - Mover cálculo de la condición:
 - Lectura de los registros + comparación en el mismo ciclo
 - Hardware adicional para la comparación (no se puede usar la ALU)



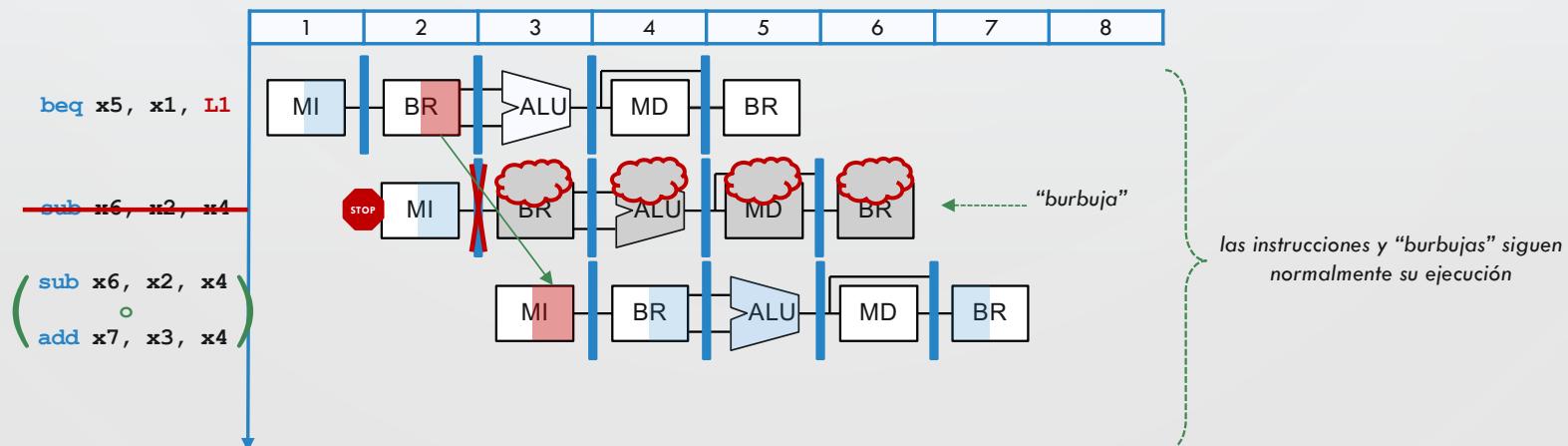
HARDWARE PARA ADELANTAR SALTOS A ID

- Intentar mover los saltos a etapas más tempranas: por ejemplo, a ID
 - Mover cálculo de la dirección: desplazar sumador a etapa ID
 - Mover cálculo de la condición:
 - Lectura de los registros + comparación en el mismo ciclo
 - Hardware adicional para la comparación (no se puede usar la ALU)



PARADA DEL CAUCE CON SALTOS EN ID

- Con esta optimización solo hay que **parar (stall)** el pipeline durante un ciclo para retrasar la ejecución de nuevas instrucciones hasta que el salto esté resuelto
 - Ciclo 2 (`beq` está en ID):** se detecta el conflicto. La instrucción **sub** se para y se inserta una “burbuja” de no operación en la etapa ID y la instrucción `beq` resuelve el salto
 - Ciclo 3 (`beq` está en EX):** se lanza la instrucción que corresponda
- Penaliza la ejecución del programa en un ciclo por instrucción de salto**



PARADA DEL CAUCE CON SALTOS EN ID

- Con esta optimización solo hay que **parar (stall)** el pipeline durante un ciclo para retrasar la ejecución de nuevas instrucciones hasta que el salto esté resuelto
 - Ciclo 2 (beq está en ID):** se detecta el conflicto. La instrucción **sub** se para y se inserta una “burbuja” de no operación en la etapa ID y la instrucción **beq** resuelve el salto
 - Ciclo 3 (beq está en EX):** se lanza la instrucción que corresponda
- Penaliza la ejecución del programa en un ciclo por instrucción de salto**

Diagrama de ejecución simplificado (condición falsa)

	1	2	3	4	5	6	7	8
beq x5, x1, L1	IF	ID	EX	M	WB			
sub x6, x2, x4		IFp	IF	ID	EX	M	WB	

Diagrama de ejecución simplificado (condición verdadera)

	1	2	3	4	5	6	7	8
beq x5, x1, L1	IF	ID	EX	M	WB			
sub x6, x2, x4		IFp	X					
L1: add x7, x3, x4			IF	ID	EX	M	WB	

PROBLEMAS ASOCIADOS A LOS SALTOS EN ID

- Necesidad de nueva anticipación a la etapa ID desde EX/MEM (a la entrada del comparador)

	1	2	3	4	5	6	7
sub x5, x5, x3	IF	ID	EX	M	WB		
or x3, x2, x1		IF	ID	EX x5	M	WB	
beq x5, x1, L1			IF	ID	EX	M	WB

- Necesidad de parada de uno o dos ciclos (stalls) si hay dependencia verdadera a distancia 1

- Parada de un ciclo si se produce en etapa EX (add, ...)

	1	2	3	4	5	6	7	8	9
sub x5, x5, x3	IF	ID	EX	M	WB				
or x5, x2, x1		IF	ID	EX	M	WB			
beq x5, x1, L1			IF	IDp x5	ID	EX	M	WB	

- Parada de dos ciclos si se carga en etapa MEM (lw)

	1	2	3	4	5	6	7	8	9
sub x5, x5, x3	IF	ID	EX	M	WB				
lw x5, 20(x1)		IF	ID	EX	M	WB			
beq x5, x1, L1			IF	IDp	IDp	ID	EX	M	WB

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M	x5 WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF																		

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID																	
add x3, x3, x2		IF																	

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← EX	ID	EX	M x5 WB		
sw x3, 0(x5)							IF ← EX	ID	M	WB	

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	IDp															
add x3, x3, x2		IF	IDp																
addi x1, x1, 4			IDp																

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M															
add x3, x3, x2		IF	IDp	IDp															
addi x1, x1, 4			IDp	IDp															

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						ID	EX	M	x5 WB		
sw x3, 0(x5)							EX	M		WB	

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB _{x2}														
add x3, x3, x2		IF	ID _p	ID _p	IF _p	IF													
addi x1, x1, 4																			

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2														
add x3, x3, x2		IF	IDp	IDp	ID	EX													
addi x1, x1, 4			IFp	IFp	IF	ID													
addi x4, x4, -1						IF													

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB	x2													
add x3, x3, x2		IF	IDp	IDp	ID	EX	M												
addi x1, x1, 4			IFp	IFp	IF	ID	EX												
addi x4, x4, -1						IF	ID												
bne x4, x0, L							IF												

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB	x2													
add x3, x3, x2		IF	IDp	IDp	ID	EX	M	WB											
addi x1, x1, 4			IFp	IFp	IF	ID	EX	M											
addi x4, x4, -1						IF	ID	EX											
bne x4, x0, L							IF	IDp											
addi x5, x5, 10								IFp											

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB	x2													
add x3, x3, x2		IF	IDp	IDp	ID	EX	M	WB											
addi x1, x1, 4			IFp	IFp	IF	ID	EX	M	WB										
addi x4, x4, -1						IF	ID	EX	M										
bne x4, x0, L							IF	IDp	IDp										
addi x5, x5, 10								IFp	IFp										

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2														
add x3, x3, x2		IF	IDp	IDp	ID	EX	M	WB											
addi x1, x1, 4			IFp	IFp	IF	ID	EX	M	WB										
addi x4, x4, -1						IF	ID	EX	M	WB x4									
bne x4, x0, L							IFp	IDp	IDp	ID	WB x4								
addi x5, x5, 10							IFp	IDp	IDp	IDp	ID								

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2														
add x3, x3, x2		IF	IDp	IDp	IF	EX	M	WB											
addi x1, x1, 4			IFp	IFp		ID	EX	M	WB										
addi x4, x4, -1						IF	ID	EX	M	WB x4									
bne x4, x0, L							IF	IDp	IDp	IFp	IFp	IF	EX						
addi x5, x5, 10								IFp	IFp	IFp	IFp								

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2														
add x3, x3, x2		IF	IDp	IDp	IF	EX	M	WB											
addi x1, x1, 4			IDp	IDp	IF	ID	EX	M	WB										
addi x4, x4, -1						IF	ID	EX	M	WB x4									
bne x4, x0, L							IF	IDp	IDp	IF	EX	M	X						
addi x5, x5, 10								IFp	IFp	IFp	IFp	IF							
L: lw x2, 0(x1)																			

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2														
add x3, x3, x2		IF	IDp	IDp	ID	EX	M	WB											
addi x1, x1, 4			IFp	IFp	IF	ID	EX	M	WB										
addi x4, x4, -1						IF	ID	EX	M	WB x4									
bne x4, x0, L							IFp	IDp	IDp	ID	EX	M	WB						
addi x5, x5, 10							IFp	IFp	IFp	IFp	X	IF	ID	EX	M	WB			
L: lw x2, 0(x1)																			

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2														
add x3, x3, x2		IF	IDp	IDp	IF	EX	M	WB											
addi x1, x1, 4			IFp	IFp		ID	EX	M	WB										
addi x4, x4, -1						IF	ID	EX	M	WB x4									
bne x4, x0, L							IF	IDp	IDp	IFp	ID	EX							
addi x5, x5, 10								IFp	IFp	IFp	IFp								

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2														
add x3, x3, x2		IF	IDp	IDp	ID	EX	M	WB											
addi x1, x1, 4			IFp	IFp	IF	ID	EX	M	WB										
addi x4, x4, -1						IF	ID	EX	M	WB x4									
bne x4, x0, L						IF	IDp	IDp	IDp	ID	EX								
addi x5, x5, 10							IFp	IFp	IFp	IFp	IFp	M							

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						ID	EX	M	x5 WB		
sw x3, 0(x5)							IF	EX	M	WB	

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2														
add x3, x3, x2		IF	IDp	IDp	IF	EX	M	WB											
addi x1, x1, 4			IFp	IFp		ID	EX	M	WB										
addi x4, x4, -1						IF	ID	EX	M	WB x4									
bne x4, x0, L							IFp	IDp	IDp	IFp	IFp	EX	M	WB					
addi x5, x5, 10								IFp	IFp	IFp	IFp	IFp	IF	ID					
sw x3, 0(x5)																			

DIAGRAMA DE EJECUCIÓN CON PARADAS POR SALTOS

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa parada tanto para los riesgos de datos como para los de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4 WB			
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← EX	ID	EX	M	x5 WB	
sw x3, 0(x5)							ID ← EX	M	WB		

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2														
add x3, x3, x2		IF	IDp	IDp	ID ↓	EX	M	WB											
addi x1, x1, 4			IFp	IFp	IF	ID	EX	M	WB										
addi x4, x4, -1						ID	EX	M	WB x4	ID ↓	EX	M	WB						
bne x4, x0, L						IF	IDp	IDp	IDp	IFp	IFp	IFp	IF	ID	EX	IDp	IDp	WB x5	
addi x5, x5, 10							IF	IF	IF	IF	IF	IF	IF	ID	EX	IDp	IDp	WB	
sw x3, 0(x5)															WB ↓	EX	M	WB	

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M	x5 WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF																		

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa **anticipación** para los **riesgos de datos** y **parada** para los de **control** (con resolución del **salto en ID**).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2WB						
add x3, x3, x2		IF	ID	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M	x4WB			
bne x4, x0, L					IF	ID	x4WB		WB		
addi x5, x5, 10						IF	ID	EX	M	x5WB	
sw x3, 0(x5)						IF	ID	EX	M		WB

Diagrama de ejecución simplificado (salto tomado)

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M x5 WB			
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX																
add x3, x3, x2		IF	IDp																
addi x1, x1, 4			IFp																

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ←	EX	M	WB					
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ←	EX	M	WB		
addi x5, x5, 10						IF ←	ID	EX	M x5 WB		
sw x3, 0(x5)							IF	ID ←	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M	x5 WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB															
add x3, x3, x2		IF	IDp	ID ← EX															
addi x1, x1, 4			IFp	IF	ID														
addi x4, x4, -1					IF														

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← EX	ID	EX	M x5 WB		
sw x3, 0(x5)							IF ← EX	ID	M	WB	

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB															
add x3, x3, x2		IF	IDp	ID ← EX	M														
addi x1, x1, 4			IFp	IF	ID	EX													
addi x4, x4, -1					IF	ID													
bne x4, x0, L						IF ← EX	ID	EX											

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M x5 WB			
sw x3, 0(x5)							IF ← EX	M	WB		

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB															
add x3, x3, x2		IF	IDp IFp	ID	EX	M	WB												
addi x1, x1, 4				IF	ID	EX	M												
addi x4, x4, -1					IF	ID	EX												
bne x4, x0, L						IF	IDp IFp												
addi x5, x5, 10																			

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M ^{x2} WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M ^{x4} WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M	^{x5} WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M ^{x2} WB															
add x3, x3, x2		IF	IDp IFp	ID ← EX	M	WB													
addi x1, x1, 4				IF	ID	EX	M	WB											
addi x4, x4, -1					IF	ID	EX ^{x4} WB	M											
bne x4, x0, L						IF	IDp IFp	ID											
addi x5, x5, 10							IFp	IFp											

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M	x5 WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB															
add x3, x3, x2		IF	IDp	IDp	EX	M	WB												
addi x1, x1, 4				IF	ID	EX	M	WB											
addi x4, x4, -1					IF	ID	EX	x4 M	WB										
bne x4, x0, L						IF	IDp	IDp	ID	EX	WB								
addi x5, x5, 10							IFp	IFp	X	IF									
L: lw x2, 0(x1)																			

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M ^{x2} WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M ^{x4} WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M	^{x5} WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M ^{x2} WB															
add x3, x3, x2		IF	IDp	ID ← EX	M	WB													
addi x1, x1, 4			IFp	IF	ID	EX	M	WB											
addi x4, x4, -1					IF	ID	EX	M ^{x4} WB											
bne x4, x0, L						IF	IDp	ID ← EX	M	WB									
addi x5, x5, 10							IFp	IFp	X										
L: lw x2, 0(x1)								IF	ID	EX	M	WB							

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M ^{x2} WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M ^{x4} WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M	^{x5} WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M ^{x2} WB															
add x3, x3, x2		IF	IDp IFp	ID	EX	M	WB												
addi x1, x1, 4				IF	ID	EX	M	WB											
addi x4, x4, -1					IF	ID	EX ^{x4} WB	M											
bne x4, x0, L						IF	IDp IFp	ID											
addi x5, x5, 10							IFp	IFp											

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M ^{x2} WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M ^{x4} WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M	^{x5} WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M ^{x2} WB															
add x3, x3, x2		IF	IDp IFp	ID	EX	M	WB												
addi x1, x1, 4				IF	ID	EX	M	WB											
addi x4, x4, -1					IF	ID	EX ^{x4} WB	M	WB										
bne x4, x0, L						IF	IDp IFp	ID	EX										
addi x5, x5, 10							IFp	IF											

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M ^{x2} WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M ^{x4} WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M	^{x5} WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M ^{x2} WB															
add x3, x3, x2		IF	IDp	ID ← EX	M	WB													
addi x1, x1, 4			IFp	IF	ID	EX	M	WB											
addi x4, x4, -1					IF	ID	EX	M ^{x4} WB	M	WB									
bne x4, x0, L						IF	IDp	ID ← EX	IDp	IF	M	EX							
addi x5, x5, 10							IFp	IFp	IFp	IF	ID								
sw x3, 0(x5)																			

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M	x5 WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB															
add x3, x3, x2		IF	IDp	ID ← EX	M	WB													
addi x1, x1, 4			IFp	IF	ID	EX	M	WB											
addi x4, x4, -1					IF	ID	EX	x4 WB	M	WB									
bne x4, x0, L						IF	IDp	ID ← EX	M	WB									
addi x5, x5, 10							IFp	IFp → IF	ID	EX	M	WB							
sw x3, 0(x5)									IF	ID	IF	ID	EX	ID					

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M ^{x2} WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M ^{x4} WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M	^{x5} WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M ^{x2} WB															
add x3, x3, x2		IF	IDp	ID ← EX	M	WB													
addi x1, x1, 4			IFp	IF	ID	EX	M	WB											
addi x4, x4, -1					IF	ID	EX	^{x4} WB	M	WB									
bne x4, x0, L						IF	IDp	ID ← EX	IFp	IF	EX	M	WB						
addi x5, x5, 10							IFp	IFp ← IF	IF	ID	IF	ID	WB						
sw x3, 0(x5)									IF	EX ^{x5} WB	M	EX	ID						

DIAGRAMA DE EJECUCIÓN CON ANTICIPACIÓN Y SALTOS EN ID

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y parada para los de control (con resolución del salto en ID).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB							
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF ← ID	EX	M x5 WB			
sw x3, 0(x5)							IF ← EX	M	WB		

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M x2 WB															
add x3, x3, x2		IF	IDp IFp	ID ← EX	M	WB													
addi x1, x1, 4				IF	ID	EX	M	WB											
addi x4, x4, -1					IF	ID	EX x4	M	WB										
bne x4, x0, L						IF	IDp IFp	ID ← EX	IF	M	WB								
addi x5, x5, 10							IFp	ID ← IF	EX	M	WB								
sw x3, 0(x5)								IF ← EX	ID	IF	M	WB							

EJEMPLO DE USO DE TÉCNICAS HW

- Los riesgos se resuelven por HW: anticipación y predicción estática de salto no tomado

Diagrama de ejecución simplificado (primera iteración)

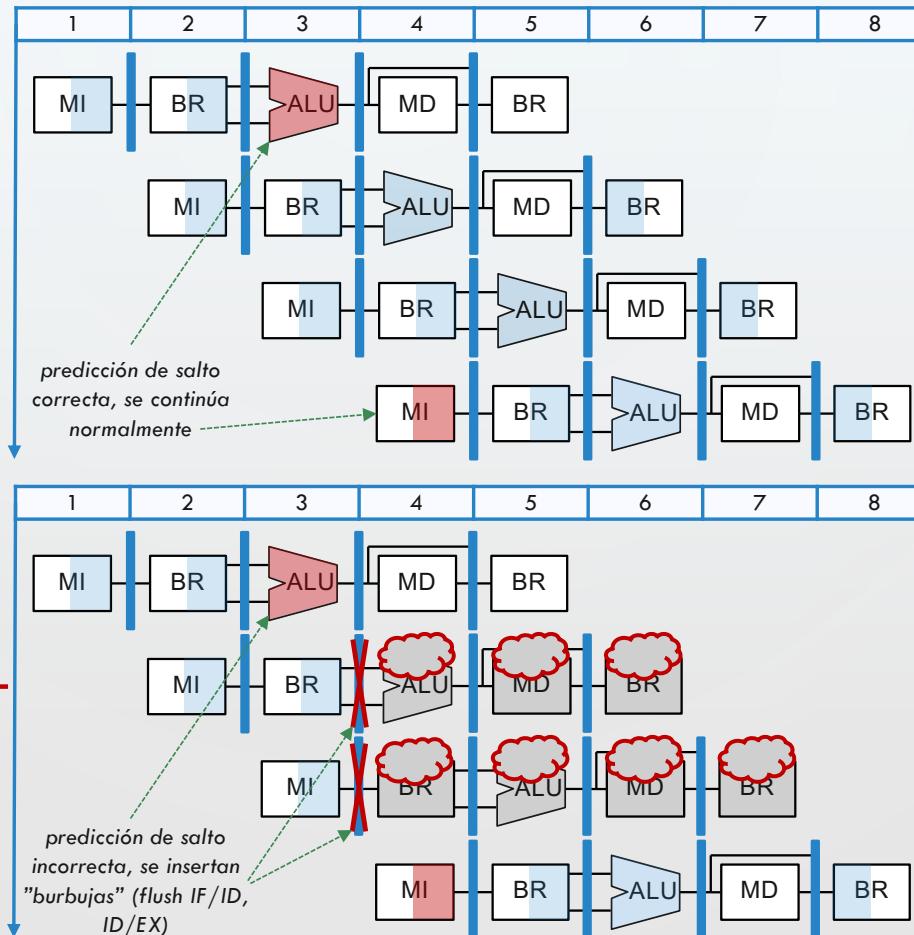
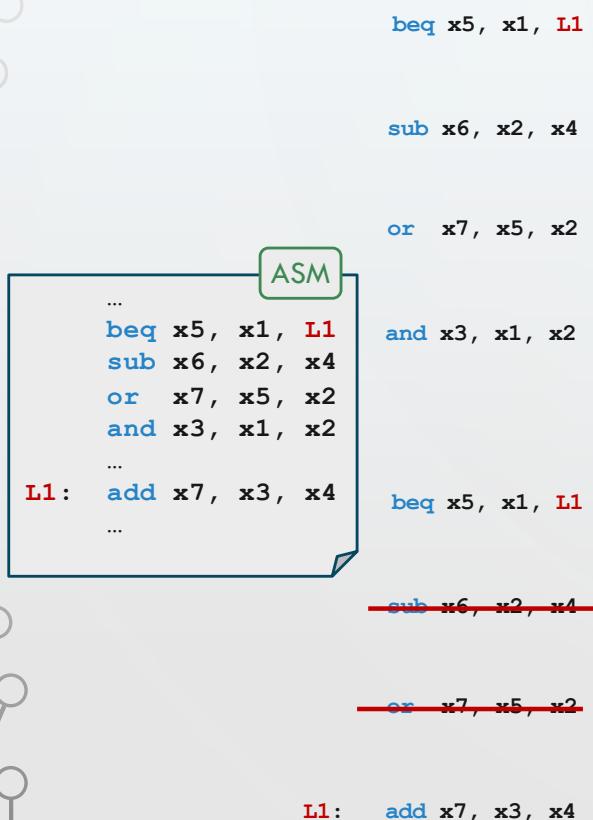
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
lw x1, desp(x0)	IF	ID	EX	M	WB														
lw x2, 0(x1)	IF	IF	IDp	WB															
lw x3, desp2(x0)			IFp	ID	EX	M	WB												
lw x4, desp3(x0)				IF	ID	EX	M	WB											
loop: beq x3, x4, exit					IF	IDp	WB												
lw x4, 0(x2)						IFp	ID	EX	M	WB									
sw x4, 0(x1)							IF	ID	WB										
addi x2, x2, 4								EX	M	WB									
addi x1, x1, 4								x4	WB										
j loop									EX	WB									
exit: ...										IF									
...										ID									

PREDICCIÓN DE SALTO ESTÁTICA

- Existe una **solución** mejor que la parada para las instrucciones de salto: **predecir que el salto NO se tomará**
 - La instrucción de salto y las siguientes se lanzan normalmente
 - Cuando se conoce la dirección/decisión de salto:
 - Si no hay que saltar, **no se hace nada**
 - Si hay que saltar, **se descartan (flush)** las instrucciones lanzadas después del salto, insertando “**burbujas**” de no operación y se lanza la instrucción que corresponda
 - **Penaliza la ejecución** del programa **sólo en el caso de fallar la predicción** (no siempre como en el caso de “parada”)
- **Predecir que el salto se tomará** es más compleja puesto que también hay que **predecir la dirección** a la que saltar
- Los predictores tienen mucho sentido en **pipelines profundos** donde el número de ciclos de **parada** después de un salto es **muy elevado**

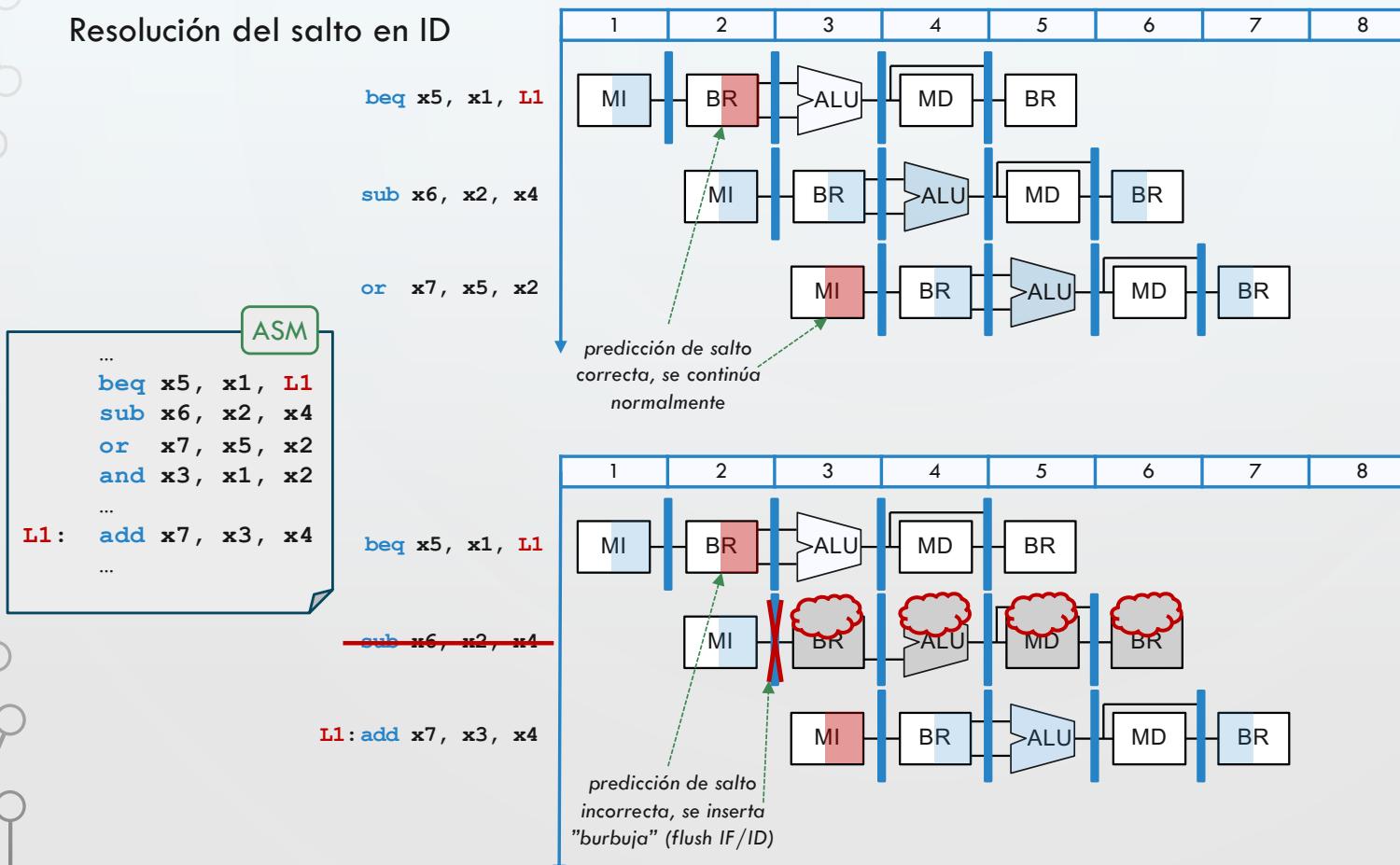
EJEMPLO DE PREDICCIÓN DE SALTO NO TOMADO

Resolución del salto en EX



EJEMPLO DE PREDICCIÓN DE SALTO NO TOMADO

Resolución del salto en ID



TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR

DIAGRAMA DE EJECUCIÓN DE SALTO NO TOMADO

- Diagramas de ejecución simplificados con fallo de predicción:
 - Sin adelantamiento de la decisión/cálculo de dirección de salto (EX)

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9
beq x5, x1, L1	IF	ID	EX	M	WB				
sub x6, x2, x4		IF	ID	X					
or x7, x5, x2			IF	X					
L1: add x7, x3, x4				IF	ID	EX	M	WB	

- Con adelantamiento de la decisión/cálculo de dirección de salto a ID

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9
beq x5, x1, L1	IF	ID	EX	M	WB				
sub x6, x2, x4		IF	X						
L1: add x7, x3, x4			IF	ID	EX	M	WB		

EJEMPLO CON ANTICIPACIÓN Y PREDICCIÓN ESTÁTICA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y predicción estática (no tomado) para los riesgos de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	x2 WB						
add x3, x3, x2		IF	ID ← EX	M	WB						
addi x1, x1, 4			IF	ID	EX	M	WB				
addi x4, x4, -1				IF	ID	EX	M x4 WB				
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF	ID	EX	M x5 WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

- Reordenamos para evitar paradas por riesgos de datos

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2						
addi x4, x4, -1		IF	ID	EX	M	WB x4					
addi x1, x1, 4			IF	ID	EX	M	WB				
add x3, x3, x2				IF	ID	EX	M	WB			
bne x4, x0, L					IF	ID ← EX	M	WB			
addi x5, x5, 10						IF	ID	EX	M x5 WB		
sw x3, 0(x5)							IF	ID ← EX	M	WB	

EJEMPLO CON ANTICIPACIÓN Y PREDICCIÓN ESTÁTICA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y predicción estática (no tomado) para los riesgos de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2						
addi x4, x4, -1		IF	ID	EX	M	WB x4					
addi x1, x1, 4			IF	ID	EX	M	WB				
add x3, x3, x2				IF	ID	EX	M	WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB														
addi x4, x4, -1		IF	ID	EX	M	WB													
addi x1, x1, 4			IF	ID	EX	M	WB												
add x3, x3, x2				IF	ID	EX	M	WB											
bne x4, x0, L					IF	ID	EX	M	WB										
addi x5, x5, 10						IF	ID	EX	M	x5 WB									

EJEMPLO CON ANTICIPACIÓN Y PREDICCIÓN ESTÁTICA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y predicción estática (no tomado) para los riesgos de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2						
addi x4, x4, -1		IF	ID	EX	M	WB x4					
addi x1, x1, 4			IF	ID	EX	M	WB				
add x3, x3, x2				IF	ID	EX	M	WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							ID	EX	M		WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB														
addi x4, x4, -1		IF	ID	EX	M	WB													
addi x1, x1, 4			IF	ID	EX	M	WB												
add x3, x3, x2				IF	ID	EX	M	WB											
bne x4, x0, L					IF	ID	EX	M	WB										
addi x5, x5, 10						IF	ID	EX	M	WB									
sw x3, 0(x5)							ID	EX	M	WB									

EJEMPLO CON ANTICIPACIÓN Y PREDICCIÓN ESTÁTICA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y predicción estática (no tomado) para los riesgos de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2						
addi x4, x4, -1		IF	ID	EX	M	WB x4					
addi x1, x1, 4			IF	ID	EX	M	WB				
add x3, x3, x2				IF	ID	EX	M	WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB														
addi x4, x4, -1		IF	ID	EX	M	WB													
addi x1, x1, 4			IF	ID	EX	M	WB												
add x3, x3, x2				IF	ID	EX	M	WB											
bne x4, x0, L					IF	ID	EX	M	WB										
addi x5, x5, 10						IF	ID	EX	M	WB									
sw x3, 0(x5)							IF	ID	EX	M	WB								
L: lw x2, 0(x1)								EX failo											

COMPARACIÓN DE PREDICCIÓN ESTÁTICA CON PARADA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y predicción estática (no tomado) para los riesgos de control (con resolución del salto en EX).

Diagrama de ejecución simplificado (salto tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB														
addi x4, x4, -1		IF	ID	EX	M	WB													
addi x1, x1, 4			IF	ID	EX	M													
add x3, x3, x2				IF	ID	EX													
bne x4, x0, L					IF	ID		EX	M	WB									
addi x5, x5, 10						IF		ID			X								
sw x3, 0(x5)									IF		X								
L: lw x2, 0(x1)										IF	ID	EX	M	WB					

Diagrama de ejecución simplificado (salto tomado) para HW sin predicción, solo parada

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB														
addi x4, x4, -1		IF	ID	EX	M	WB													
addi x1, x1, 4			IF	ID	EX	M													
add x3, x3, x2				IF	ID	EX													
bne x4, x0, L					IF	ID		EX	M	WB									
addi x5, x5, 10						IFp		ID		X									
L: lw x2, 0(x1)									IF		IF								

EJEMPLO CON ANTICIPACIÓN Y PREDICCIÓN ESTÁTICA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y predicción estática (no tomado) para los riesgos de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2						
addi x4, x4, -1		IF	ID	EX	M	WB x4					
addi x1, x1, 4			IF	ID	EX	M	WB				
add x3, x3, x2				IF	ID	EX	M	WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							ID	EX	M	WB	

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB														
addi x4, x4, -1		IF	ID	EX	M	WB													
addi x1, x1, 4			IF	ID	EX	M	WB												
add x3, x3, x2				IF	ID	EX													
bne x4, x0, L					IF	ID	EX												
addi x5, x5, 10						IF	ID	EX											

EJEMPLO CON ANTICIPACIÓN Y PREDICCIÓN ESTÁTICA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y predicción estática (no tomado) para los riesgos de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2						
addi x4, x4, -1		IF	ID	EX	M	WB x4					
addi x1, x1, 4			IF	ID	EX	M	WB				
add x3, x3, x2				IF	ID	EX	M	WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							ID	EX	M		WB

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB														
addi x4, x4, -1		IF	ID	EX	M	WB													
addi x1, x1, 4			IF	ID	EX	M	WB												
add x3, x3, x2				IF	ID	EX	M	WB											
bne x4, x0, L					IF	ID	EX	M	WB										
addi x5, x5, 10						IF	ID	EX	M	WB									
sw x3, 0(x5)							ID	EX	M	WB									

acertado

EJEMPLO CON ANTICIPACIÓN Y PREDICCIÓN ESTÁTICA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y predicción estática (no tomado) para los riesgos de control (con resolución del salto en EX).

Diagrama de ejecución simplificado

	1	2	3	4	5	6	7	8	9	10	11
L: lw x2, 0(x1)	IF	ID	EX	M	WB x2						
addi x4, x4, -1		IF	ID	EX	M	WB x4					
addi x1, x1, 4			IF	ID	EX	M	WB				
add x3, x3, x2				IF	ID	EX	M	WB			
bne x4, x0, L					IF	ID	EX	M	WB		
addi x5, x5, 10						IF	ID	EX	M	x5 WB	
sw x3, 0(x5)							IF	ID	EX	M	WB

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB														
addi x4, x4, -1		IF	ID	EX	M	WB													
addi x1, x1, 4			IF	ID	EX	M	WB												
add x3, x3, x2				IF	ID	EX	M	WB											
bne x4, x0, L					IF	ID	EX	M	WB										
addi x5, x5, 10						IF	ID	EX	x5	M	WB								
sw x3, 0(x5)							IF	ID	EX	M	WB								

COMPARACIÓN DE PREDICCIÓN ESTÁTICA CON PARADA

- Realiza el diagrama de ciclos para el siguiente código en un HW que implementa anticipación para los riesgos de datos y predicción estática (no tomado) para los riesgos de control (con resolución del salto en EX).

Diagrama de ejecución simplificado (salto NO tomado)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB														
addi x4, x4, -1		IF	ID	EX	M	WB													
addi x1, x1, 4			IF	ID	EX	M	WB												
add x3, x3, x2				IF	ID	EX	M	WB											
bne x4, x0, L					IF	ID	EX	EX	M	WB									
addi x5, x5, 10						IF	ID	EX	x5	M	WB								
sw x3, 0(x5)							IF	ID	EX		WB								
										M	WB								

Diagrama de ejecución simplificado (salto NO tomado) para HW sin predicción, solo parada

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L: lw x2, 0(x1)	IF	ID	EX	M	WB														
addi x4, x4, -1		IF	ID	EX	M	WB													
addi x1, x1, 4			IF	ID	EX	M	WB												
add x3, x3, x2				IF	ID	EX	M	WB											
bne x4, x0, L					IF	ID	EX	EX	M	WB									
addi x5, x5, 10						IFp	IFp	IF	ID	IF	EX	x5	M	WB					
sw x3, 0(x5)									IF	ID	EX	M	WB						
												M	WB						

PREDICCIÓN DE SALTO DINÁMICA

- Para **pipelines profundos** o **procesadores superescalares**, donde un fallo de predicción es muy costoso en términos de ciclos perdidos la **predicción estática no es adecuada**
- Utilizando más HW es posible predecir el comportamiento de los saltos dinámicamente durante la ejecución de los programas
- **Predicción de saltos Dinámica:** predice los saltos en tiempo de ejecución usando **información de run-time** (historia de saltos previos)

PREDICTOR DINÁMICO DE 1 BIT

- Se almacena información sobre los saltos (tomado/no tomado) para determinar la predicción actual: tomar/no tomar
- Se utiliza un Branch Prediction Buffer (o Branch History Table - BHT) en la etapa IF
 - Se puede direccionar con los bits menos significativos de PC
 - Almacena un bit que indica si la última vez se tomó (1) o no (0) el salto
 - Este bit se pasa a la etapa ID (IF/ID) para que se tome la decisión de salto (cuando ya se sabe si la instrucción es un salto)
- Si la predicción es errónea, se descartan (flush) las instrucciones incorrectas y se invierte el bit de predicción en la BHT
 - Un fallo de predicción no afecta a la corrección del programa sino a su rendimiento
 - La tasa de fallos de predicción de una BHT de 4096 bits varía entre un 1% y un 20%

EJEMPLO DE PREDICCIÓN DINÁMICA DE UN BUCLE

- Ejemplo de bucle (10 iteraciones)

ASM

```
...  
addi x5, x0, 10  
loop: addi x5, x5, -1  
...  
...  
beq x5, x0, loop  
...
```

	BHT(dir_beq)		acuerdo /fallo
	actual	siguiente	
x5 = 9	0		

predice NO TOMADO (*no salta*)

EJEMPLO DE PREDICCIÓN DINÁMICA DE UN BUCLE

- Ejemplo de bucle (10 iteraciones)

ASM

```
...  
addi x5, x0, 10  
loop: addi x5, x5, -1  
...  
...  
beq x5, x0, loop  
...
```

	BHT(dir_beq)		acuerdo /fallo
	actual	siguiente	
x5 = 9	0	1	X

predice NO TOMADO (no salta)

por lo que falla la predicción y se cambia BHT

EJEMPLO DE PREDICCIÓN DINÁMICA DE UN BUCLE

- Ejemplo de bucle (10 iteraciones)

ASM

```
...  
addi x5, x0, 10  
loop: addi x5, x5, -1  
...  
...  
beq x5, x0, loop  
...
```

	BHT(dir_beq)		acuerdo /fallo
	actual	siguiente	
x5 = 9	0	1	X
x5 = 8	1	1	✓

predice TOMADO (salta)

por lo que no se cambia BHT

EJEMPLO DE PREDICCIÓN DINÁMICA DE UN BUCLE

- Ejemplo de bucle (10 iteraciones)

ASM

```
...
    addi x5, x0, 10
loop: addi x5, x5, -1
...
...
    beq x5, x0, loop
...
```

	BHT(dir_beq)		acerto /fallo
	actual	siguiente	
x5 = 9	0	1	X
x5 = 8	1	1	✓
x5 = 7	1	1	✓
x5 = 6	1	1	✓
x5 = 5	1	1	✓
x5 = 4	1	1	✓
x5 = 3	1	1	✓
x5 = 2	1	1	✓
x5 = 1	1	1	✓
x5 = 0	1		

en la última iteración predice tomado

EJEMPLO DE PREDICCIÓN DINÁMICA DE UN BUCLE

- Ejemplo de bucle (10 iteraciones)

ASM

```
...  
addi x5, x0, 10  
loop: addi x5, x5, -1  
...  
...  
beq x5, x0, loop  
...
```

	BHT(dir_beq)		acerto /fallo
	actual	siguiente	
x5 = 9	0	1	X
x5 = 8	1	1	✓
x5 = 7	1	1	✓
x5 = 6	1	1	✓
x5 = 5	1	1	✓
x5 = 4	1	1	✓
x5 = 3	1	1	✓
x5 = 2	1	1	✓
x5 = 1	1	0	✓
x5 = 0	1	0	X

en la última iteración predice tomado

por lo que falla la predicción y
se cambia BHT

EJEMPLO DE PREDICCIÓN DINÁMICA DE UN BUCLE

- Ejemplo de bucle (10 iteraciones)

ASM

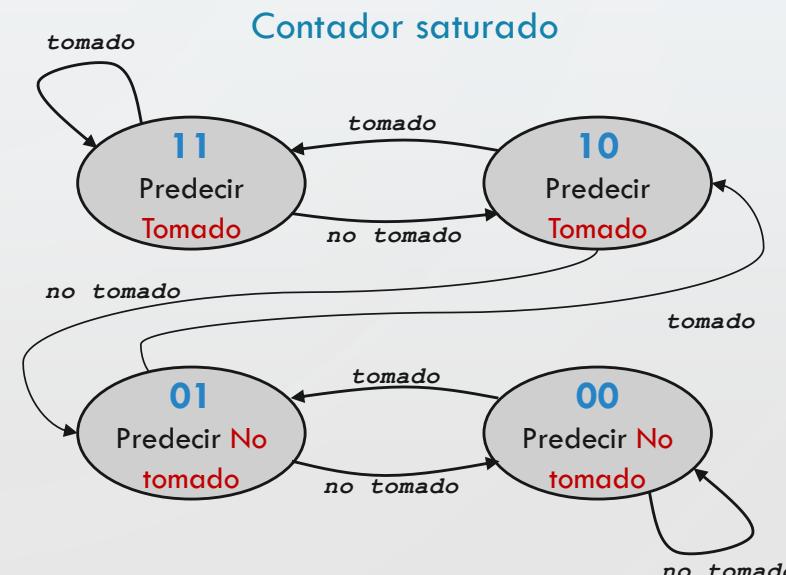
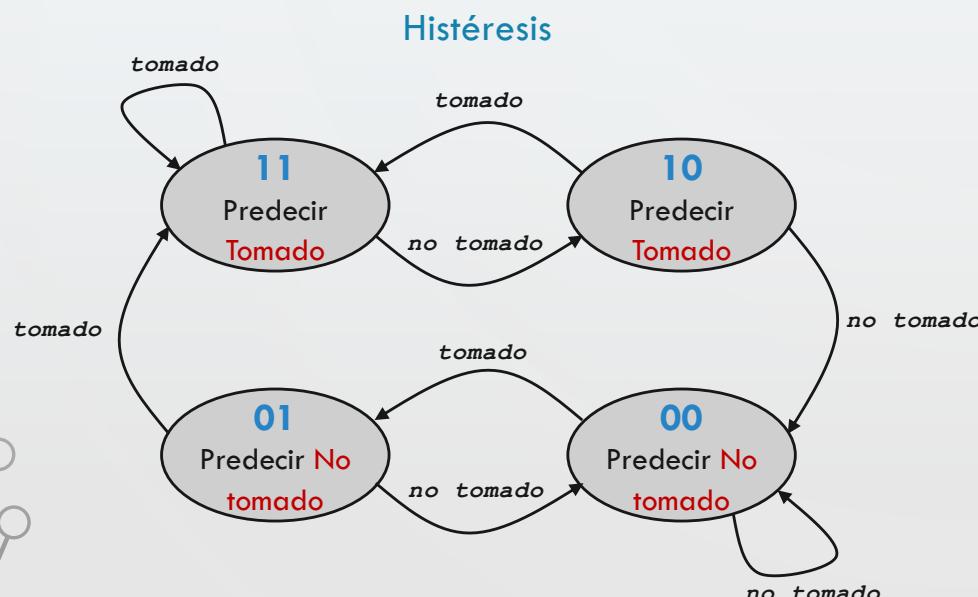
```
...  
addi x5, x0, 10  
loop: addi x5, x5, -1  
...  
...  
beq x5, x0, loop  
...
```

	BHT(dir_beq)		acerto /fallo
	actual	siguiente	
x5 = 9	0	1	X
x5 = 8	1	1	✓
x5 = 7	1	1	✓
x5 = 6	1	1	✓
x5 = 5	1	1	✓
x5 = 4	1	1	✓
x5 = 3	1	1	✓
x5 = 2	1	1	✓
x5 = 1	1	1	✓
x5 = 0	1	0	X

Para este caso concreto el predictor acierta un 80% para un salto que se toma un 90% de las veces

PREDICTOR DINÁMICO DE 2 BITS

- La **BHT** utiliza **dos bits** para hacer la predicción (0 – no tomado, 1– tomado)
 - Tiene que haber dos fallos consecutivos para cambiar el sentido de la predicción



EJEMPLO DE PREDICCIÓN DINÁMICA DE UN BUCLE

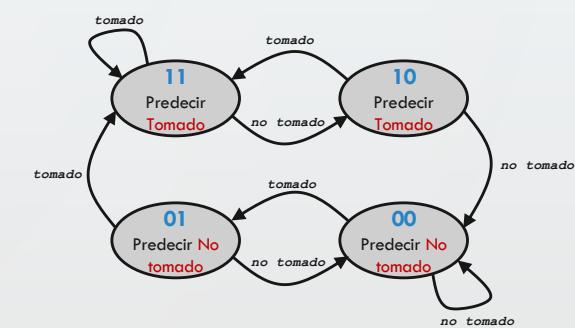
- Ejemplo de bucle (10 iteraciones)

ASM

```
...  
addi x5, x0, 10  
loop: addi x5, x5, -1  
...  
...  
beq x5, x0, loop  
...
```

	BHT(dir_beq)		acuerdo /fallo
	actual	siguiente	
x5 = 9	00		

predice NO TOMADO (no salta)



EJEMPLO DE PREDICCIÓN DINÁMICA DE UN BUCLE

- Ejemplo de bucle (10 iteraciones)

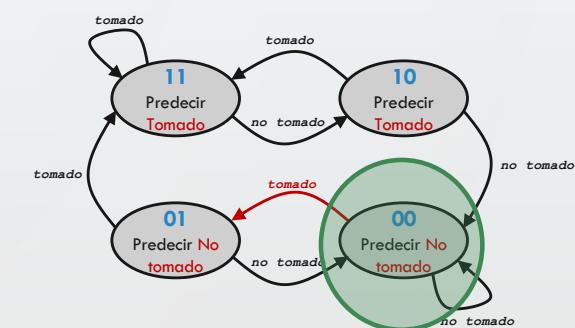
ASM

```
...  
addi x5, x0, 10  
loop: addi x5, x5, -1  
...  
...  
beq x5, x0, loop  
...
```

	BHT(dir_beq)		acuerdo /fallo
	actual	siguiente	
x5 = 9	00	01	X

predice NO TOMADO (no salta)

por lo que falla la predicción y se cambia BHT



EJEMPLO DE PREDICCIÓN DINÁMICA DE UN BUCLE

- Ejemplo de bucle (10 iteraciones)

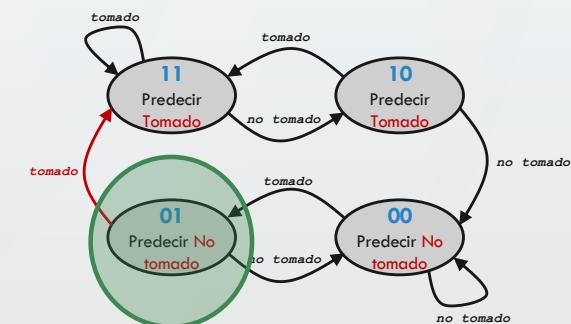
ASM

```
...  
addi x5, x0, 10  
loop: addi x5, x5, -1  
...  
...  
beq x5, x0, loop  
...
```

	BHT(dir_beq)		acerto /fallo
	actual	siguiente	
x5 = 9	00	01	X
x5 = 8	01	11	X

predice NO TOMADO (no salta)

por lo que falla otra vez y se cambia BHT



EJEMPLO DE PREDICCIÓN DINÁMICA DE UN BUCLE

- Ejemplo de bucle (10 iteraciones)

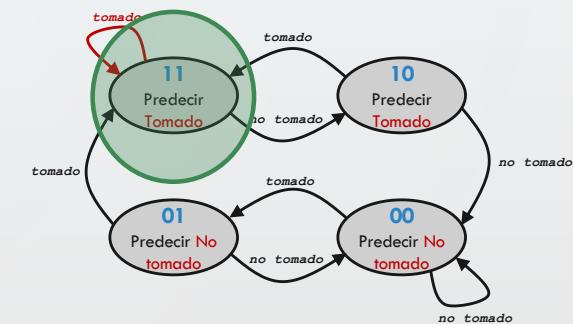
ASM

```
...  
addi x5, x0, 10  
loop: addi x5, x5, -1  
...  
...  
beq x5, x0, loop  
...
```

	BHT(dir_beq)		acuerdo /fallo
	actual	siguiente	
x5 = 9	00	01	X
x5 = 8	01	11	X
x5 = 7	11	11	✓

predice TOMADO (salta)

por lo que acierta y
no se cambia BHT



EJEMPLO DE PREDICCIÓN DINÁMICA DE UN BUCLE

- Ejemplo de bucle (10 iteraciones)

ASM

```

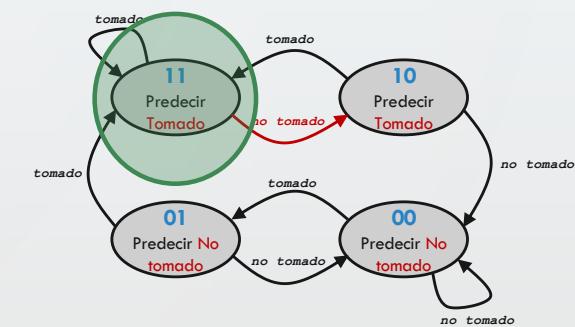
...
    addi x5, x0, 10
loop: addi x5, x5, -1
...
...
    beq x5, x0, loop
...

```

	BHT(dir_beq)		acuerdo /fallo
	actual	siguiente	
x5 = 9	00	01	X
x5 = 8	01	11	X
x5 = 7	11	11	✓
x5 = 6	11	11	✓
x5 = 5	11	11	✓
x5 = 4	11	11	✓
x5 = 3	11	11	✓
x5 = 2	11	11	✓
x5 = 1	11	11	✓
x5 = 0	11	10	X

predice TOMADO (salta)

por lo que no acierta y se cambia BHT (pero
no predicción aún)



EJEMPLO DE PREDICCIÓN DINÁMICA DE UN BUCLE

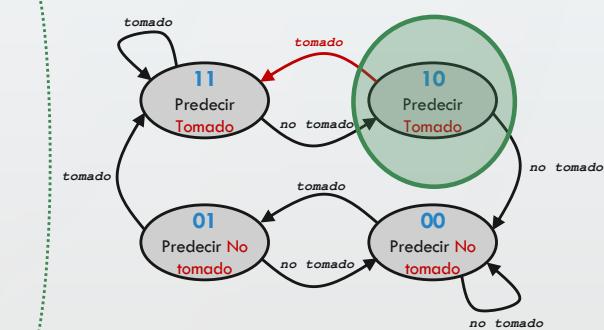
- Ejemplo de bucle (10 iteraciones)

ASM

```
...  
    addi x5, x0, 10  
loop: addi x5, x5, -1  
...  
...  
    beq x5, x0, loop  
...
```

	BHT(dir_beq)		acuerdo /fallo
	actual	siguiente	
x5 = 9	10	11	✓
x5 = 8	11	11	✓
x5 = 7	11	11	✓
x5 = 6	11	11	✓
x5 = 5	11	11	✓
x5 = 4	11	11	✓
x5 = 3	11	11	✓
x5 = 2	11	11	✓
x5 = 1	11	11	✓
x5 = 0	11	10	✗

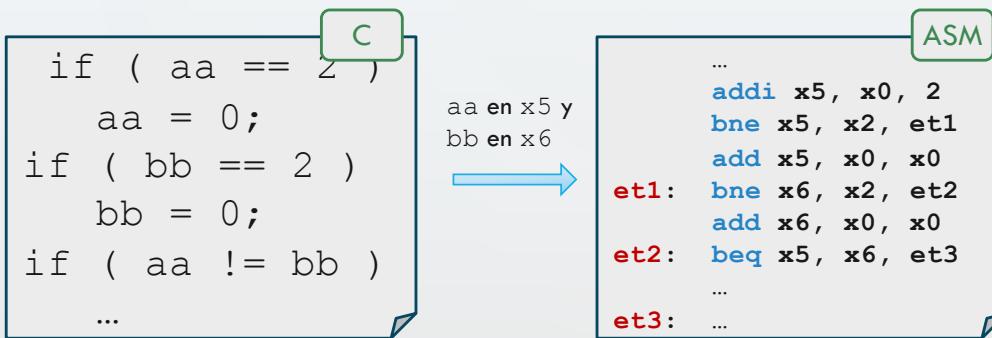
vuelve a predecir TOMADO y acierta



Para este caso concreto el predictor acierta un 90% para un salto que se toma un 90% de las veces

PREDICTOR CORRELADO

- Hay códigos en los que algunos saltos no dependen de su historia previa, sino de otros saltos: los saltos están **correlados**



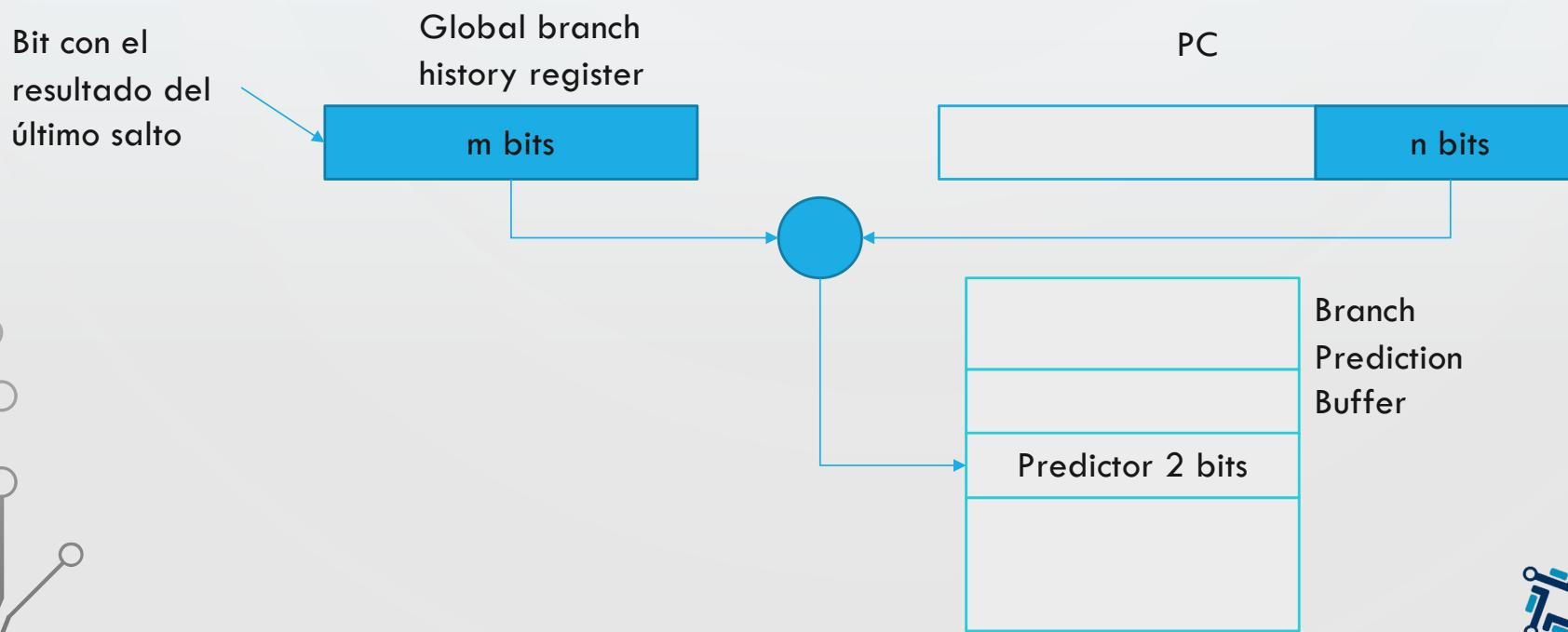
Si los dos primeros saltos no se toman, el tercero sí se toma

- Un predictor correlado mantiene el histórico de m saltos previos en un **Global Branch History Register** (GBHR), y un **Branch Prediction Buffer** (BPB) con 2^m predictores de salto de k -bits.



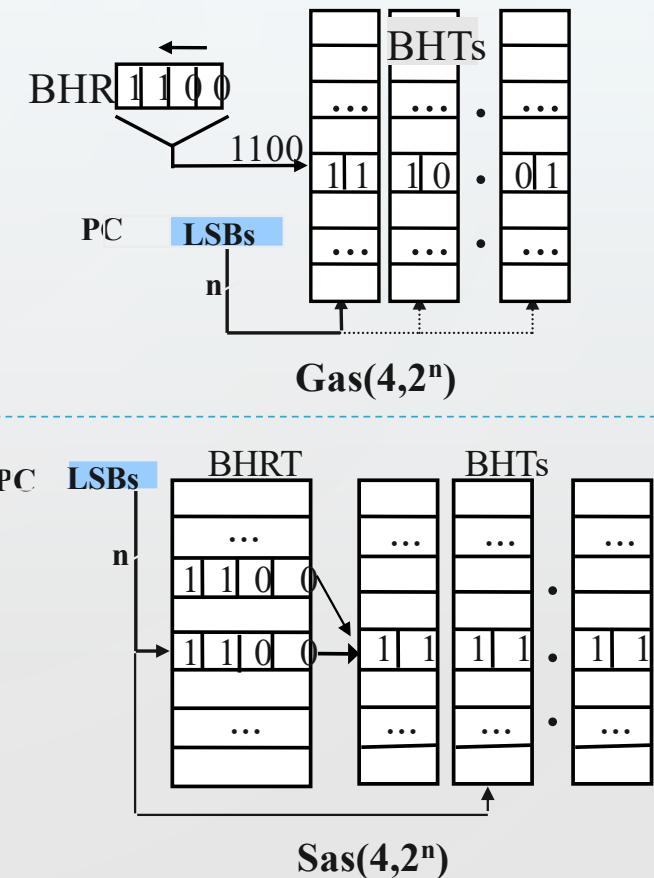
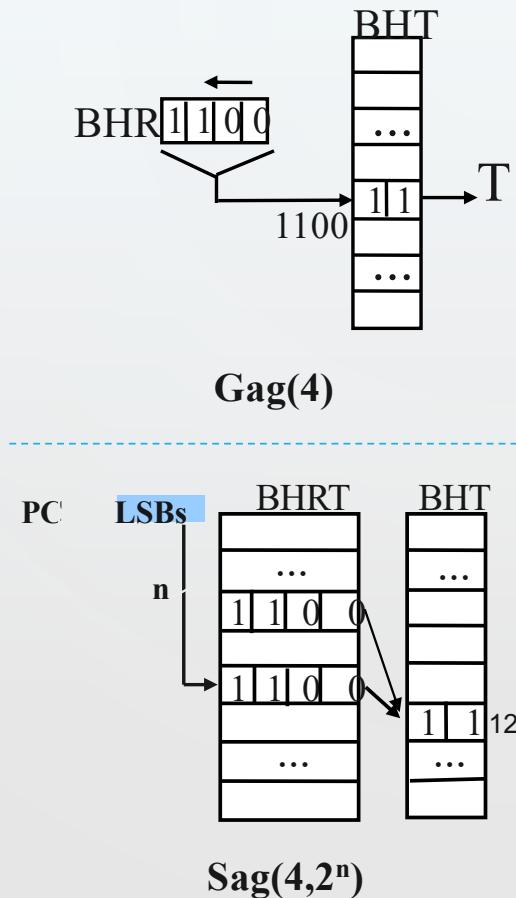
PREDICTOR GSHARE

- El predictor Gshare es un predictor correlado en el que los m bits del **GBHR** se combinan con los n bits menos significativos del **PC** ($m=n$) mediante una función XOR para direccionar la **BPB**



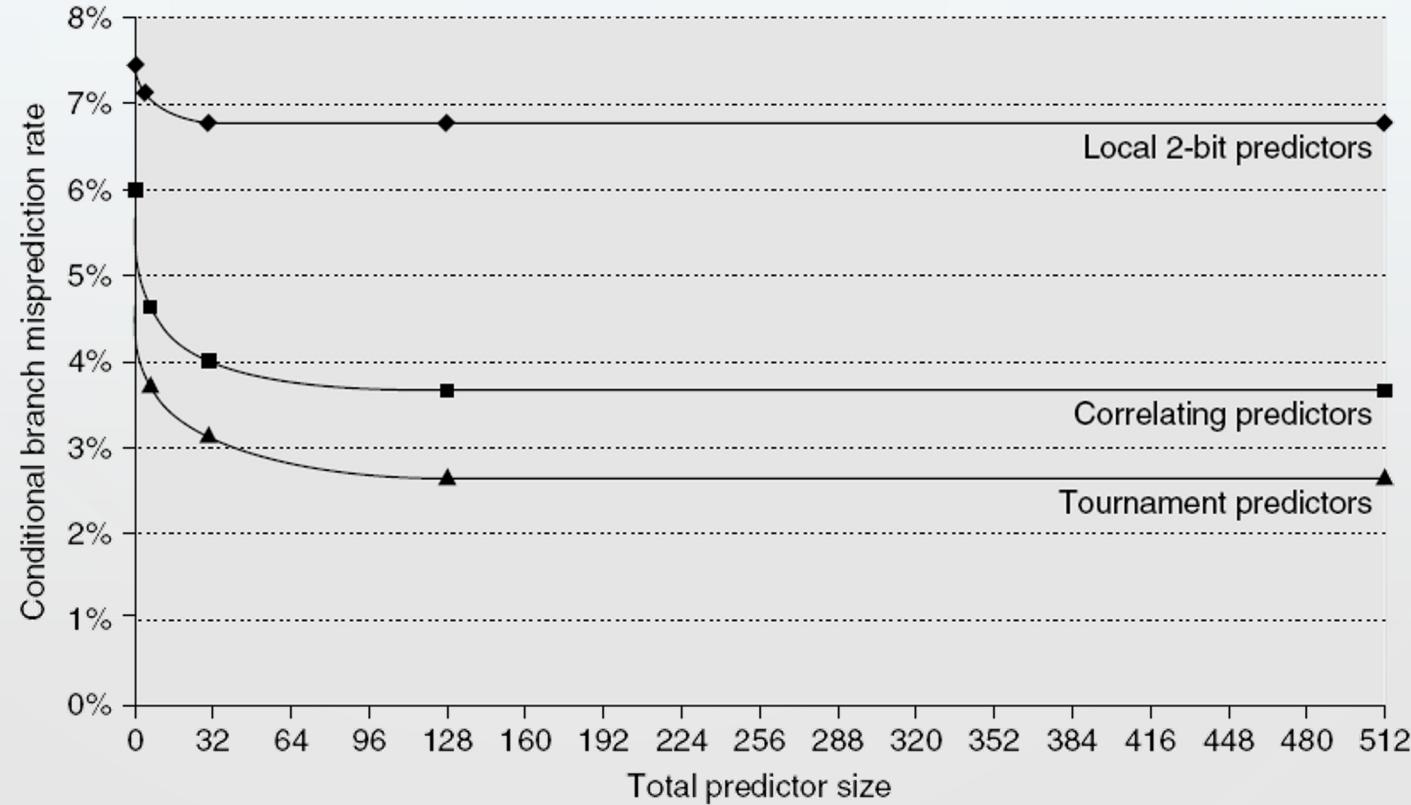
PREDICTORES DE DOS NIVELES

- Generalización del predictor correlado:
- Primer nivel BHR: histórico de los saltos predichos (por dirección, S) o de todos (globalmente, G)
 - Segundo nivel BHT: tabla de predicciones por dirección (s) o global (g)



PREDICTORES POR TORNEO (HÍBRIDOS)

Se combinan varios predictores y en cada momento se selecciona el que mejor resultado haya dado últimamente

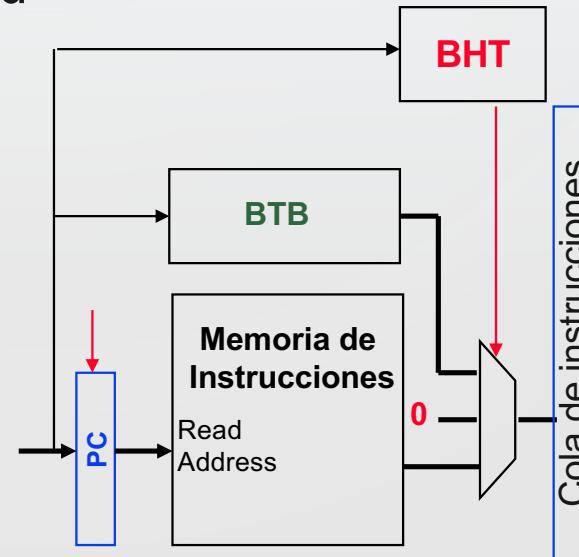


BUFFER DE DIRECCIONES DE SALTO (BTB)

- La **BHT** predice **CUANDO** se salta, pero no dice nada de **DONDE** se salta
 - Si se predice **salto tomado** hay que **calcular la dirección de salto**
 - Hay que **parar el pipeline** uno o más ciclos (dependiendo de la etapa en la que se calcule)
 - **Buffer de Direcciones de Salto** (Branch-Target Buffer **BTB**):
 - Memoria asociativa (cache) que **almacena la dirección/instrucción destino de los saltos** (más costosa que BHT)
 - Con BTB, si hay acierto en la predicción, también se sabe dirección/instrucción destino del salto **evitando la parada del pipeline**
 - En la etapa IF con el PC actual la BTB nos proporciona el siguiente PC

BUFFER DE DIRECCIONES DE SALTO (BTB)

- La **BTB** se direcciona con el PC y almacena la siguiente instrucción que se predice ejecutar
- La **BHT** también se direcciona con el PC y predice si se salta a la instrucción que predice la BTB, se hace un **flush** (por un fallo en la predicción) o se ejecuta la instrucción que se leyó de la memoria



IMPACTO DE LOS RIESGOS SOBRE EL CPI

- El **CPI efectivo** en un procesador segmentado es idealmente 1 pero **se incrementa** por culpa de los **riesgos** estructurales, de datos y de control
- En el procesador segmentado con todos los mecanismos hardware de resolución de riesgos vistos en clase hay dos casos que penalizan el CPI:
 - **Casos load-uso**: hay una penalización de 1 ciclo
 - **Predicciones incorrectas de salto**: hay una penalización de 2 ciclos si el salto se resuelve en EX, y de 1 ó 2 ciclos si se resuelve en ID
- Por ejemplo, para un procesador segmentado con anticipación, predicción estática de salto no tomado y resolución de saltos en EX, si $\#load_uso$ es el número de casos load-uso y $\#fallos_pred$ el número de fallos de predicción, entonces

$$CPI_{ef} = 1 + \frac{\#load_{uso} + \#fallos_{pred}}{NI}$$

EJEMPLO DE USO DE TÉCNICAS HW

- Los riesgos se resuelven por HW: anticipación y predicción estática de salto no tomado resuelto en EX

Diagrama de ejecución simplificado (primera iteración)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
lw x1, desp(x0)	IF	ID	EX	M ^{x2} WB															
lw x2, 0(x1)		IF	IDp	WB															
lw x3, desp2(x0)			IFp	IF	ID	EX	M	WB											
lw x4, desp3(x0)				IF	ID	EX	M	WB	M ^{x4} WB										
loop: beq x3, x4, exit					IF	IDp	ID	EX	M	WB					IF	ID	EX	M WB	
lw x4, 0(x2)						IFp	IF	ID	EX	M	WB				IF	ID	EX	M	
sw x4, 0(x1)							IF	ID	EX	M	WB				IF	ID	EX	M EX	
addi x2, x2, 4								IF	ID	EX	M	WB				IF	ID	EX M	
addi x1, x1, 4									IF	ID	EX	M	WB				IF	ID	
j loop										IF	ID	EX	M	WB				IF	
exit: ...											IF	ID	EX	M	WB				
...												IF	ID	EX	M	WB			

5 ciclos

Antes de entrar al bucle

9 ciclos

Primera iteración

EJEMPLO DE USO DE TÉCNICAS HW

- Los riesgos se resuelven por HW: anticipación y predicción estática de salto no tomado resuelto en EX

Diagrama de ejecución simplificado (segunda iteración y última iteración)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
lw x1, desp(x0)																			
lw x2, 0(x1)																			
lw x3, desp2(x0)																			
lw x4, desp3(x0)																			
loop: beq x3, x4, exit		IF	ID	EX <i>acíerto</i>	M	WB				IF	ID	EX <i>fallo</i>	M	WB					
lw x4, 0(x2)			IF	ID	EX	M	WB				IF	ID	EX <i>fallo</i>	M	WB				
sw x4, 0(x1)				IF	ID	EX	M	WB				IF	ID	EX <i>fallo</i>	M	WB			
addi x2, x2, 4	WB				IF	ID	EX	M	WB				IF	ID	EX <i>fallo</i>	M	WB		
addi x1, x1, 4	M	WB				IF	ID	EX	M	WB				IF	ID	EX <i>fallo</i>	M	WB	
j loop						IF	ID	EX	M	WB					IF				
exit: ...							IF	ID	EX	M	WB								
...								IF	ID	EX	M	WB							

Diagrama de ejecución simplificado (segunda iteración y última iteración). El diagrama muestra la ejecución de un bucle (loop) y su salida (exit). Los ciclos están divididos en "Iteraciones intermedias" (8 ciclos) y "Última iteración" (3 ciclos). Se indican las fases de IF (Instrucción), ID (Identificación), EX (Ejecución) y WB (Write Back). Se marcan los resultados de las predicciones: "acíerto" (verde) para el salto correcto y "fallo" (rojo) para el salto incorrecto. Los errores se resuelven en la fase EX.

$$T_{CPU_{seghw}} = NC \times T_{CLK} = (5 + 9 + 9 \times 8 + 3) \times 200 = 89 \times 200 = 17800\text{ps}$$

antes 1^a 2^a-10^a última

EJEMPLO DE USO DE TÉCNICAS HW

- Los riesgos se resuelven por HW: anticipación y predicción estática de salto no tomado resuelto en EX

```
load-uso    lw x1, desp(x0)
load-uso    lw x2, 0(x1)
load-uso    lw x3, desp2(x0)
load-uso    lw x4, desp3(x0)
loop:      beq x3, *4, exit
           lw x4, 0(x2)
           sw x4, 0(x1)
           addi x2, x2, 4
           addi x1, x1, 4
           j loop
exit: ...
```

} 4 instrucciones
+ 1 load-uso
10 iteraciones x 6 instrucciones
+ 1 beq (salida del bucle)
+ 1 load-uso
+ 10 fallos de predicción de j
+ 1 fallo de predicción de beq

$$NI = 4 + 6 \times 10 + 1 = 65$$
$$CPI_{ef} = 1 + \frac{2+11 \times 2}{65} = \frac{89}{65} = 1,37$$

$$T_{CPU_{seghw}} = 65 \times \frac{89}{65} \times 200 = 17800 \text{ ps}$$

$$S = \frac{T_{CPU_{mono}}}{T_{CPU_{seghw}}} = \frac{52000 \text{ ps}}{17800 \text{ ps}} = 2,92$$

$$S = \frac{T_{CPU_{multi}}}{T_{CPU_{seghw}}} = \frac{50600 \text{ ps}}{17800 \text{ ps}} = 2,84$$

$$S = \frac{T_{CPU_{segsw}}}{T_{CPU_{seghw}}} = \frac{22400 \text{ ps}}{17800 \text{ ps}} = 1,26$$

PARALELISMO AVANZADO

- Latencia vs Productividad
- Paralelismo a nivel de instrucción
 - Procesadores superescalares y supersegmentados
 - Planificación dinámica de instrucciones y especulación
- Paralelismo a nivel de datos
- Paralelismo a nivel de threads

TIEMPO DE RESPUESTA VS PRODUCTIVIDAD

“Si usted tuviera que arar un campo, ¿qué preferiría usar, dos bueyes fuertes o 1024 gallinas?”

Seymour Cray, padre de la supercomputación



TIEMPO DE RESPUESTA VS PRODUCTIVIDAD

Analicemos la respuesta extendiendo la analogía. Supongamos que queremos arar un campo de 10 hectáreas ($100m \times 1000m$), con surcos cada $0,250m$ (en total 4000 surcos)



Imagen generada con Perplexity AI

Queremos **reducir** el tiempo necesario para completar todos los surcos (**tareas**)
con un coste razonable

TIEMPO DE RESPUESTA VS PRODUCTIVIDAD

Solución básica: usar un azadón

- Supongamos que un hombre puede arar a una velocidad de 1 metro por minuto
- La **latencia** necesaria para completar un surco sería de **100 minutos**
 - Necesitamos hacer 4000 surcos, así que todas las tareas acabarían en 400000 minutos, es decir, 6666 horas o 277 días
 - La **productividad** será de $4 \cdot 10^3 / 4 \cdot 10^5 = 0.01 \text{ surcos/minuto}$



Imagen generada con Perplexity AI

TIEMPO DE RESPUESTA VS PRODUCTIVIDAD

Primera mejora: aumentar la velocidad de arado usando un caballo

- Supongamos que un caballo puede arar a una velocidad de 10 metros por minuto
- La **latencia** necesaria para completar un surco sería de **10 minutos**
 - Necesitamos hacer 4000 surcos, así que todas las tareas acabarían en 40000 minutos, es decir, unos 27 días
 - La **productividad** será de $4 \cdot 10^3 / 4 \cdot 10^4 = 0.1$ *surcos/minuto*



Imagen generada con Perplexity AI

TIEMPO DE RESPUESTA VS PRODUCTIVIDAD

Segunda mejora: añadir más puntas al arado y usar animales más fuertes (y probablemente lentos) para tirar de él

- Supongamos que una pareja de bueyes puede tirar de un arado de 8 puntas a una velocidad de 5 metros por minuto
- La **latencia** necesaria para completar un surco aumenta a **20 minutos**
 - Pero cada 20 minutos completamos 8 surcos, así que todas las tareas acabarían en 10000 minutos, es decir, unos 7 días
 - La **productividad** será de $4 \cdot 10^3 / 1 \cdot 10^4 = 0.4 \text{ surcos/minuto}$



Imagen generada con Perplexity AI

TIEMPO DE RESPUESTA VS PRODUCTIVIDAD

Tercera mejora: utilizar una multitud de animales débiles y lentos equipados con pequeños arados

- Supongamos que una gallina puede tirar de un arado a una velocidad de 0,1 metros por minuto

- La **latencia** necesaria para completar un surco aumenta a **1000 minutos**

- Pero cada 1000 minutos completamos 1024 surcos, así que todas las tareas acabarían en menos de 4000 minutos, es decir, unos 3 días

- La **productividad** será aproximadamente $\frac{4 \cdot 10^3}{4 \cdot 10^3} = 1 \text{ surco/minuto}$



Imagen generada con Perplexity AI

TIEMPO DE RESPUESTA VS PRODUCTIVIDAD

Solución	Latencia	Productividad	Tiempo total
Un hombre con una azada	100 min.	0.01 furrows/min.	400.000 min.
Un caballo con un arado	10 min.	0.1 furrows/min.	40.000 min.
Dos bueyes con un arado de 8 puntas	20 min.	0.4 furrows/min.	10.000 min.
1024 gallinas con pequeñas azadas	1000 min.	1 furrows/min.	4.000 min.

Los arquitectos de computadores alcanzan estos resultados

- Aumentando la frecuencia y escala de integración, y usando paralelismo a nivel de instrucción
- Usando hyper-threading, procesadores multinúcleo, etc
- Mediante paralelismo masivo, por ejemplo, con GPUs

PARALELISMO Y LA LEY DE AMDAHL

- La ley de Amdahl fue formulada inicialmente para predecir el rendimiento obtenido al parallelizar una fracción de un código ($F_M = F$) y ejecutarlo en P procesadores ($S_M = P$).
- Una consecuencia importante de esta ley es que el rendimiento del paralelismo estará limitado por

$$S = \frac{1}{1 - F + \frac{F}{P}} = \frac{P}{F + P \times (1 - F)} < \frac{1}{1 - F}$$

- Esta limitación es grave si solo se aprovecha el paralelismo para mejorar la ejecución secuencial del código → llega un punto en el que da igual el número de procesadores o núcleos en el caso de multiprocesadores

LEY DE GUSTAFSON

- La ley de Gustafson predice qué rendimiento se puede alcanzar si aumentamos el tamaño del programa que se ejecutaba secuencialmente. Es decir, en lugar de reducir el tiempo de la parte que se paralleliza, **se aumenta el número de tareas que se ejecutan en paralelo**.
- Si seq es el tiempo que tarda la parte que se ejecuta secuencialmente, par el tiempo de la parte que se paralleliza, y P el número de procesadores usado, entonces $F = \frac{par}{seq+par}$ y el rendimiento máximo S_g que se puede alcanzar es

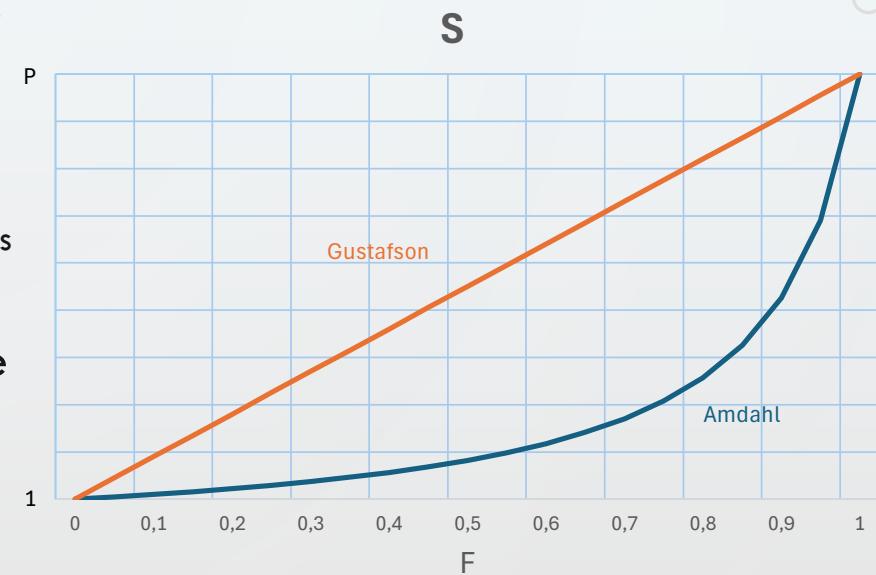
$$S_g = \frac{seq + P \times par}{seq + par} = 1 - F + F \times P$$

Ojo, el enfoque de la ley de Amdahl es distinto: $S = \frac{seq+par}{seq+\frac{par}{P}}$



GUSTAFSON VS AMDAHL

- Son leyes basadas en dos filosofías distintas, pero posiblemente complementarias, sobre como aprovechar el paralelismo
 - La ley de Amdahl predice el rendimiento para un tamaño fijo de programa → reducir latencia
 - La ley de Gustafson predice el rendimiento si aumentamos el tamaño del programa → aumentar productividad
- El tiempo que tarda la parte paralela cuando se ejecuta secuencialmente no suele ser igual que cuando se ejecuta en paralelo debido a
 - Cambios en el algoritmo usado
 - Sincronización entre las distintas partes
 - Competición por recursos hardware con otras instancias que se ejecutan en paralelo



EL PARALELISMO ESTÁ EN TODAS PARTES

Software

- Paralelismo a nivel de peticiones
Se asignan a computadores
- Paralelismo a nivel de threads
Se asignan a núcleos
- Paralelismo a nivel de instrucciones Aquí estamos
> 1 instrucción concurrentemente
- Paralelismo a nivel de datos
> 1 dato concurrentemente
- Paralelismo a nivel de Hardware
Todas las puertas lógicas funcionando al mismo tiempo

Hardware

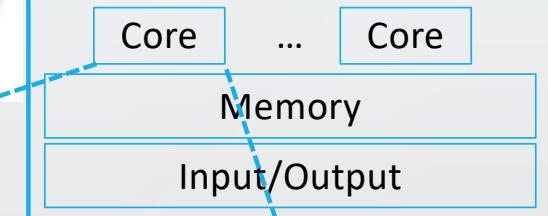
Warehouse Scale Computer



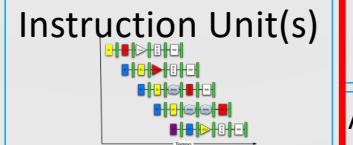
Smart Phone



Computer



Core

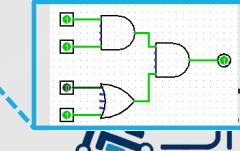


Functional Unit(s)

$$A_0+B_0 \quad A_1+B_1 \quad A_2+B_2 \quad A_3+B_3$$

Cache Memory

Logic Gates



PARALELISMO A NIVEL DE INSTRUCCIÓN

- La segmentación explota el paralelismo potencial entre instrucciones: ILP, Instruction-Level Parallelism
- Al explotar el ILP, el objetivo es minimizar el CPI al incluir los ciclos perdidos por riesgos estructurales, riesgos de datos y riesgos de control
- Podemos considerar dos métodos para aumentar el paralelismo potencial:
 - Aumentar la profundidad del cauce (más etapas) para solapar más instrucciones
 - Lanzar varias instrucciones al mismo tiempo en cada etapa del cauce (cauces paralelos)

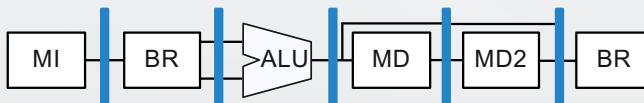
PARALELISMO A NIVEL DE INSTRUCCIÓN

- **Supersegmentación (superpipelining)**: se aumenta la profundidad del cauce, de forma que más instrucciones se solapan y se puede aumentar la frecuencia de reloj
 - Aparecen más riesgos, hay que añadir caminos de anticipación, y los retardos asociados a los registros de segmentación suponen una parte importante del ciclo de instrucción
- **Lanzamiento múltiple (multiple issue)**: se busca y ejecuta más de una instrucción al mismo tiempo, expandiendo cada etapa del cauce para acomodar múltiples instrucciones. El valor de $CPI < 1$ por lo que usaremos IPC, Instrucciones Por Ciclo
 - Hay que replicar componentes internos y las dependencias pueden impedir ejecutar dos instrucciones simultáneamente

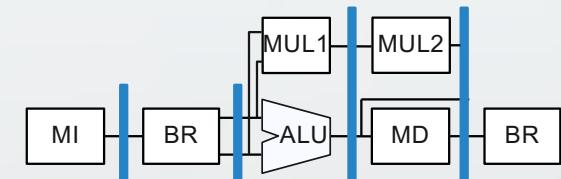
SUPERSEGMENTACIÓN

- Al incluir instrucciones más complejas es necesario más tiempo para completarlas. Podemos hacer el **reloj más lento** para acomodarlas, pero hay un impacto muy negativo en el rendimiento. Es mejor dividir la ejecución de dicha operación en **varios ciclos del pipeline**

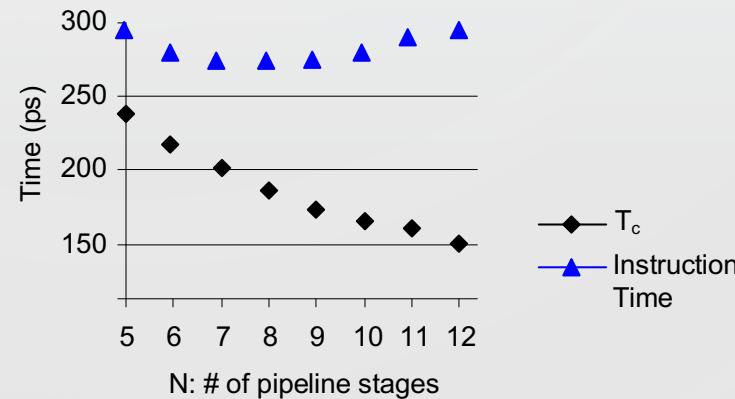
Ej. acceso a memoria de datos el doble de lento que el de memoria de instrucciones



Ej. multiplicación de dos ciclos, solapando con MD ya que no es usada en estas instrucciones



- Típicamente los cauces tienen entre 10 y 20 etapas



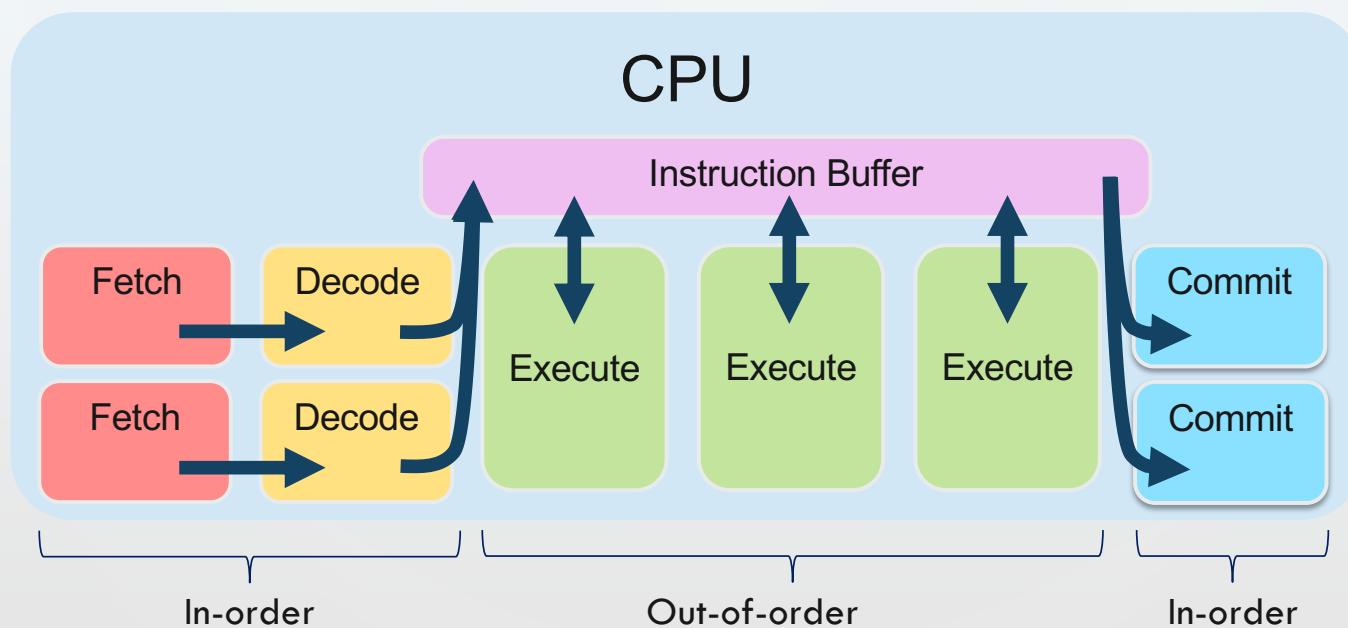
LANZAMIENTO MÚLTIPLE DE INSTRUCCIONES

- **Estático (Very Long Instruction Word, VLIW)**: se agrupan varias instrucciones en un paquete de emisión o ejecución (*issue packet*)
 - Un paquete es como una instrucción muy larga (VLIW) que incluye operaciones que se pueden lanzar juntas, cada una por una ranura de ejecución (*issue slot*)
 - El paquete no tiene porque incluir instrucciones para todas las ranuras
 - El compilador se encarga de formar el paquete y resolver los riesgos
- **Dinámico (superescalar)**: el procesador se encarga de lanzar varias instrucciones en tiempo de ejecución
 - El hardware tiene que resolver dependencias y posiblemente reordenar el código

PLANIFICACIÓN DINÁMICA DE INSTRUCCIONES

- Consiste en **ejecutar** las instrucciones **fueras de orden** (Out-of-Order, OoO)
 - Las instrucciones se **lanzan** (dispatch) y **finalizan** (commit) **en orden**
 - Si una instrucción no puede progresar por el cauce (por culpa de algún riesgo), no bloquea a las instrucciones posteriores
 - Aparecen riesgos estructurales, WAR y WAW que hay que resolver parando el cauce o usando técnicas de renombrado de registros
 - Si el riesgo es de control se puede **especular**, pero hace falta un hardware que permita finalizar en orden
 - Las excepciones deben tratarse de forma precisa
- Algunos de los métodos más conocidos son el **Scoreboard** y el **algoritmo de Tomasulo con especulación**

PLANIFICACIÓN DINÁMICA DE INSTRUCCIONES

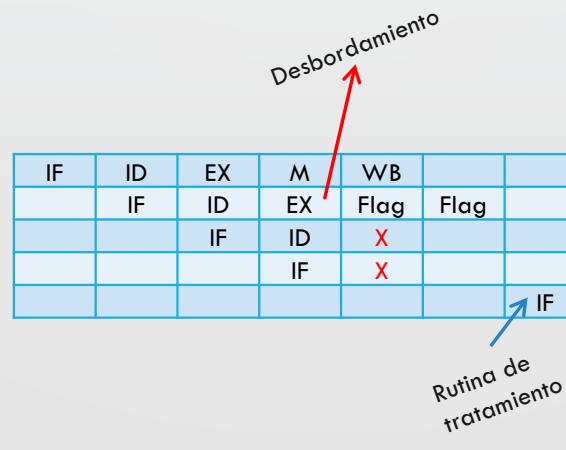


EXCEPCIONES

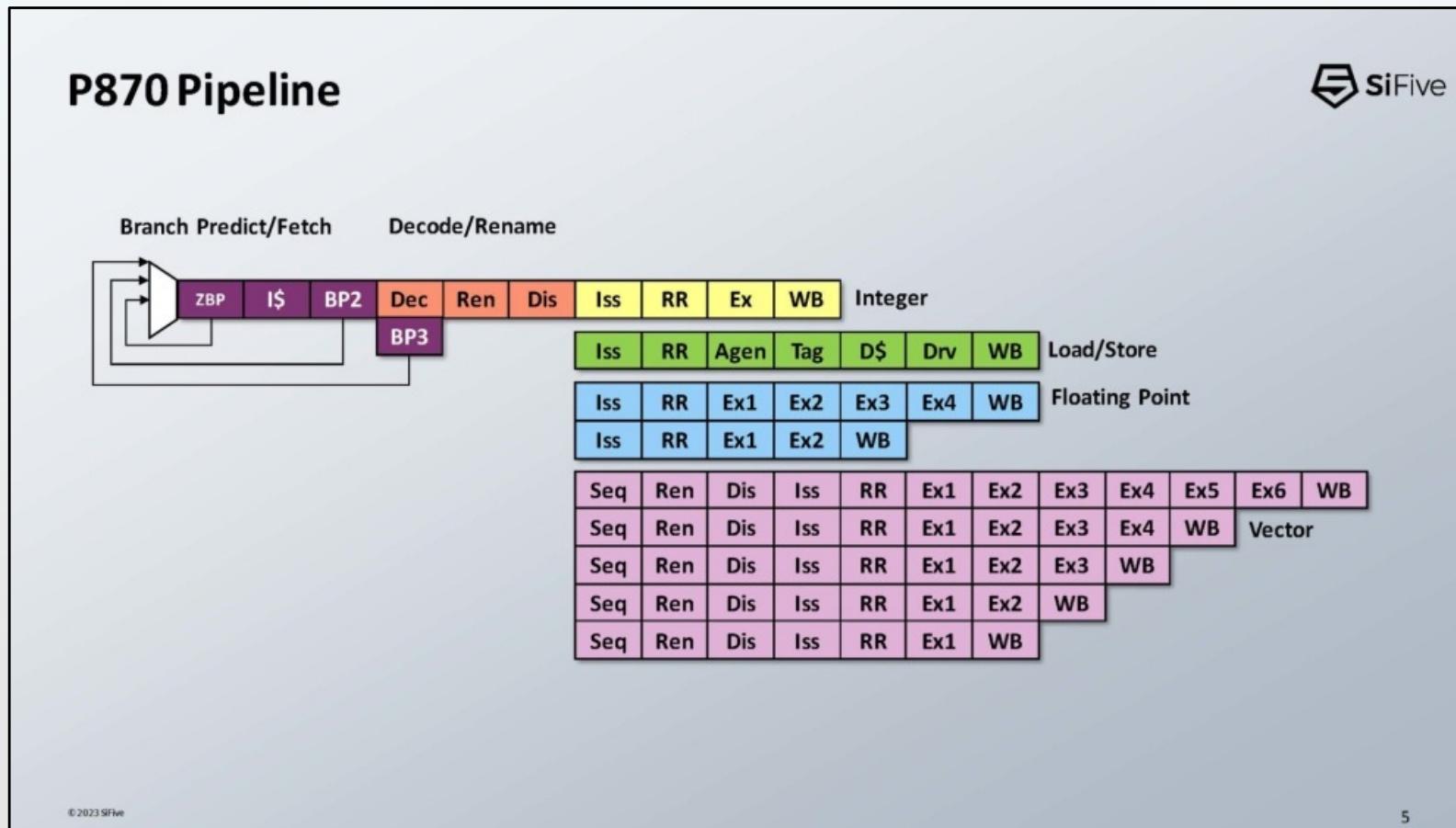
- Son eventos extraordinarios que pueden ocurrir
 - Durante la ejecución de instrucciones (síncronos con el programa), por ejemplo un desbordamiento durante una operación de suma
 - Por interrupciones externas, por ejemplo la pulsación de una tecla
 - Por malfuncionamiento del hardware, por ejemplo un fallo de alimentación
- Cuando ocurre una excepción, el hardware se encarga de atenderla mediante la ejecución de una rutina específica de tratamiento

EXCEPCIONES PRECISAS

- Se dice que una excepción es **precisa** si la rutina de tratamiento se ejecuta **tras la finalización** de las instrucciones previas
 - La ejecución fuera de orden complica tratar las excepciones de forma precisa
- Una forma de conseguirlo consiste en etiquetar (flag) la instrucción causante usando un registro especial (el registro Exception Status Register), poner a 0 el registro de segmentación, y esperar a la última fase (WB) para lanzar la rutina de tratamiento

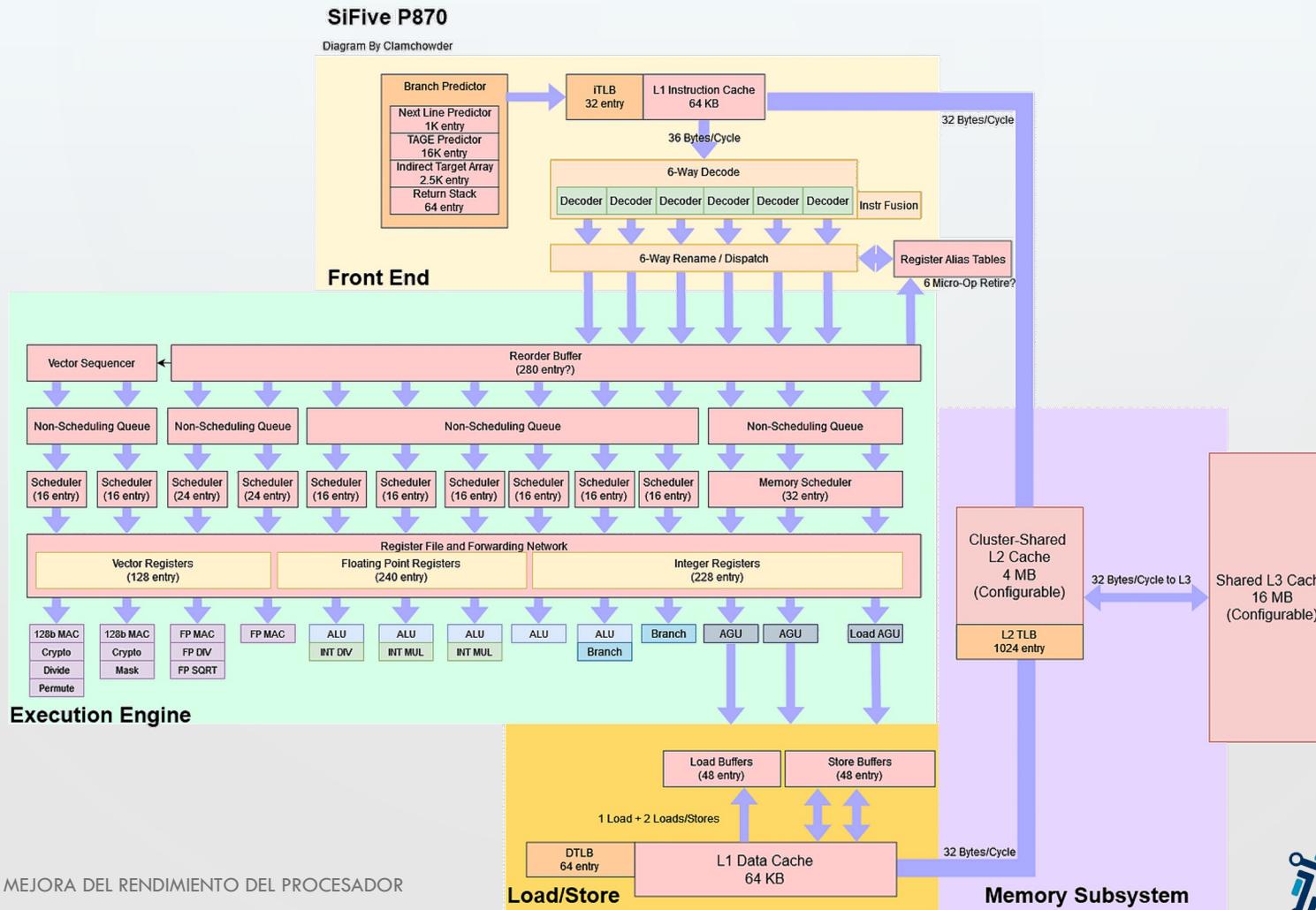


EJEMPLO REAL DE CAUCE DE RISC-V



TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR

EJEMPLO REAL DE MICROARQUITECTURA DE RISC-V



TEMA 1: MEJORA DEL RENDIMIENTO DEL PROCESADOR



PLANIFICACIÓN SOFTWARE DE INSTRUCCIONES

- Un **bloque básico** de código consiste en una secuencia de instrucciones que no tiene saltos salvo en la última instrucción, ni destinos de salto salvo en la primera instrucción
- La planificación software de instrucciones puede ser
 - **Local**, reordenando instrucciones dentro de bloques básicos
 - **Global**, reordenando o incluso fusionando bloques básicos
 - Planificación cíclica, cuando los bloques forman parte de un bucle
 - Planificación no cíclica, cuando se reordenan los bloques de un grafo de llamadas

DESENRROLLADO DE BUCLES

- Es una planificación global cíclica consistente en replicar el cuerpo del bucle
 - Se replica el cuerpo del bucle tantas veces como queramos (factor de desenrollado)
 - Se renombran registros para eliminar riesgos WAR y WAW
 - Se mueven todos los loads al principio y los stores al final

Original	Desenrollado de factor 2	Renombrado de registros	Reordenación del código
Loop:	Loop:	Loop:	Loop:
lw x10, 0(x1)	lw x10, 0(x1)	lw x10, 0(x1)	lw x10, 0(x1)
lw x11, 0(x2)	lw x11, 0(x2)	lw x11, 0(x2)	lw x11, 0(x2)
add x12, x11, x10	add x12, x11, x10	add x12, x11, x10	lw x13, 4(x1)
sw x12, 0(x1)	sw x12, 0(x1)	sw x12, 0(x1)	lw x14, 4(x2)
addi x1, x1, 4	lw x10, 4(x1)	lw x13, 4(x1)	add x12, x11, x10
addi x2, x2, 4	lw x11, 4(x2)	lw x14, 4(x2)	add x15, x13, x14
addi x3, x3, -1	add x12, x11, x10	add x15, x13, x14	addi x3, x3, -2
bne x3, x0, Loop	sw x12, 4(x1)	sw x12, 4(x1)	addi x1, x1, 8
	addi x1, x1, 8	addi x1, x1, 8	addi x2, x2, 8
	addi x2, x2, 8	addi x2, x2, 8	sw x12, -8(x1)
	addi x3, x3, -2	addi x3, x3, -2	sw x12, -4(x1)
	bne x3, x0, Loop	bne x3, x0, Loop	bne x3, x0, Loop

PARALELISMO A NIVEL DE DATOS

- Software*
- Paralelismo a nivel de peticiones
Se asignan a computadores
 - Paralelismo a nivel de threads
Se asignan a núcleos
 - Paralelismo a nivel de instrucciones
> 1 instrucción concurrentemente
 - Paralelismo a nivel de datos
> 1 dato concurrentemente
 - Paralelismo a nivel de Hardware
Todas las puertas lógicas
funcionando al mismo tiempo

Hardware

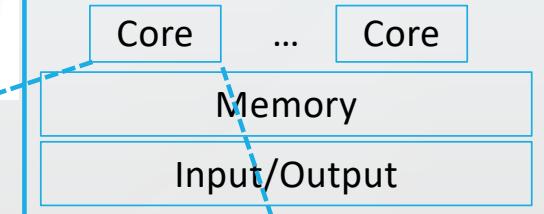
Warehouse Scale Computer



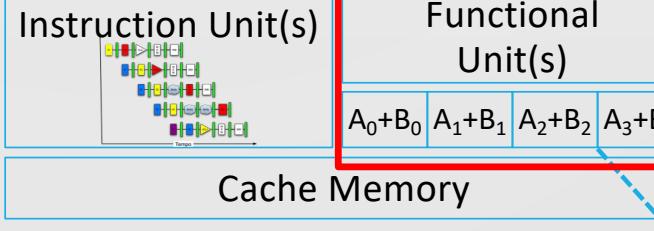
Smart Phone



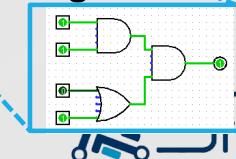
Computer



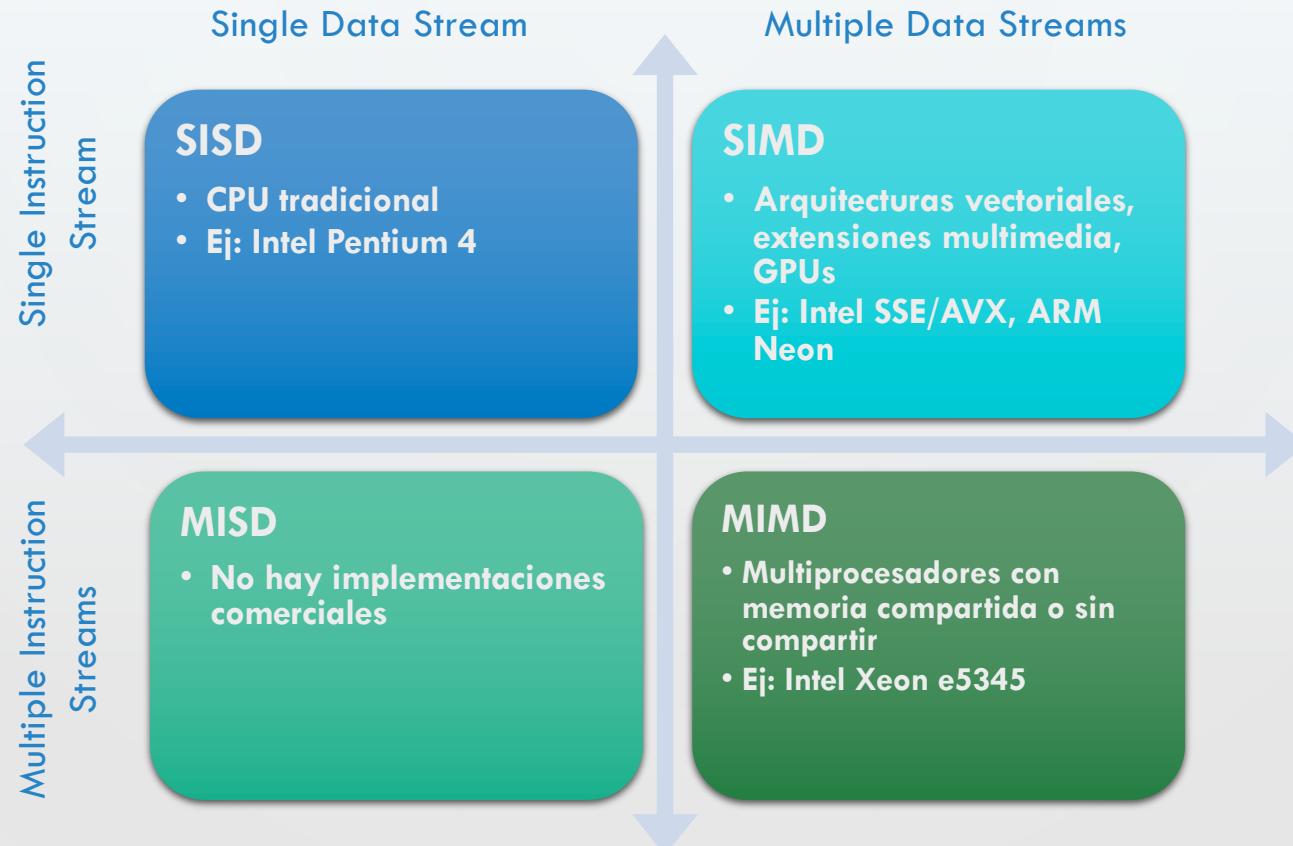
Core



Logic Gates

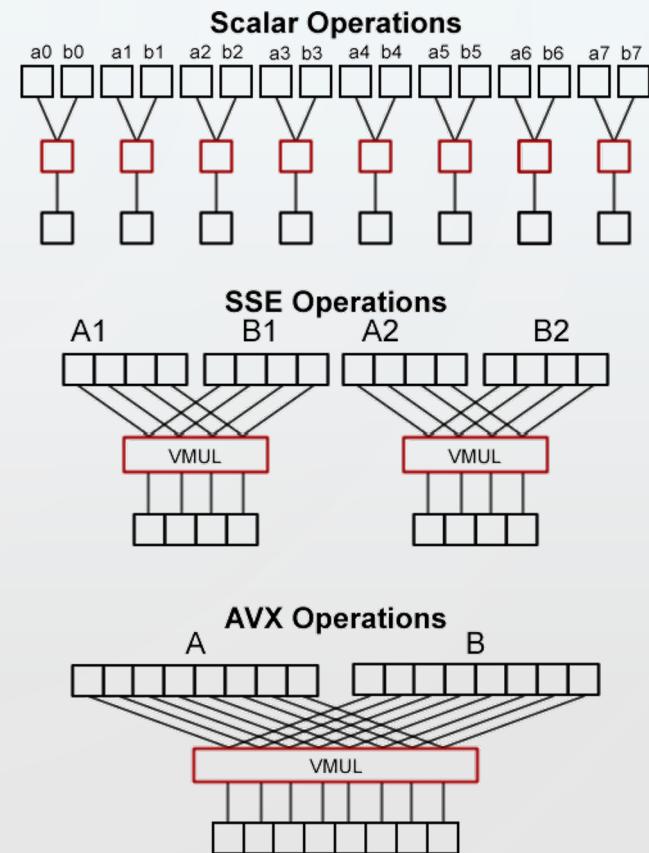


TAXONOMÍA DE FLYNN



ARQUITECTURAS VECTORIALES

- Fueron populares en el siglo pasado para procesar vectores, pero hoy en día han sido sustituidos por procesadores con extensiones vectoriales
- Estas extensiones cuentan con instrucciones que permiten leer y escribir varios valores de memoria en un solo acceso, y procesarlos al mismo tiempo



EXTENSIONES MULTIMEDIA

- SIMD, Single Instruction Multiple Data
 - Una sola instrucción es capaz de procesar varios datos al mismo tiempo
 - Ejemplo:

```
for (int i = 0; i <= N; i++) {  
    C[i] = A[i] + B[i];  
}
```

```
.L1:  
    ld    r4, [r1]  
    ld    r5, [r2]  
    add   r4, r4, r5  
    st    [r3], r4  
    add   r0, r0, #4  
    cmp   r0, #N  
    ble  .L1
```

r0: #0
r1: &A[i]
r2: &B[i]
r3: &C[i]

Código escalar

```
for (int i = 0; i <= N; i+=4) {  
    C[i:i+3] = A[i:i+3] + B[i:i+3];  
}
```

```
.L1:  
    vld   vr4, [r1]  
    vld   vr5, [r2]  
    vadd  vr4, vr4, vr5  
    vst   [r3], vr4  
    add   r0, r0, #16  
    cmp   r0, #N  
    ble  .L1
```

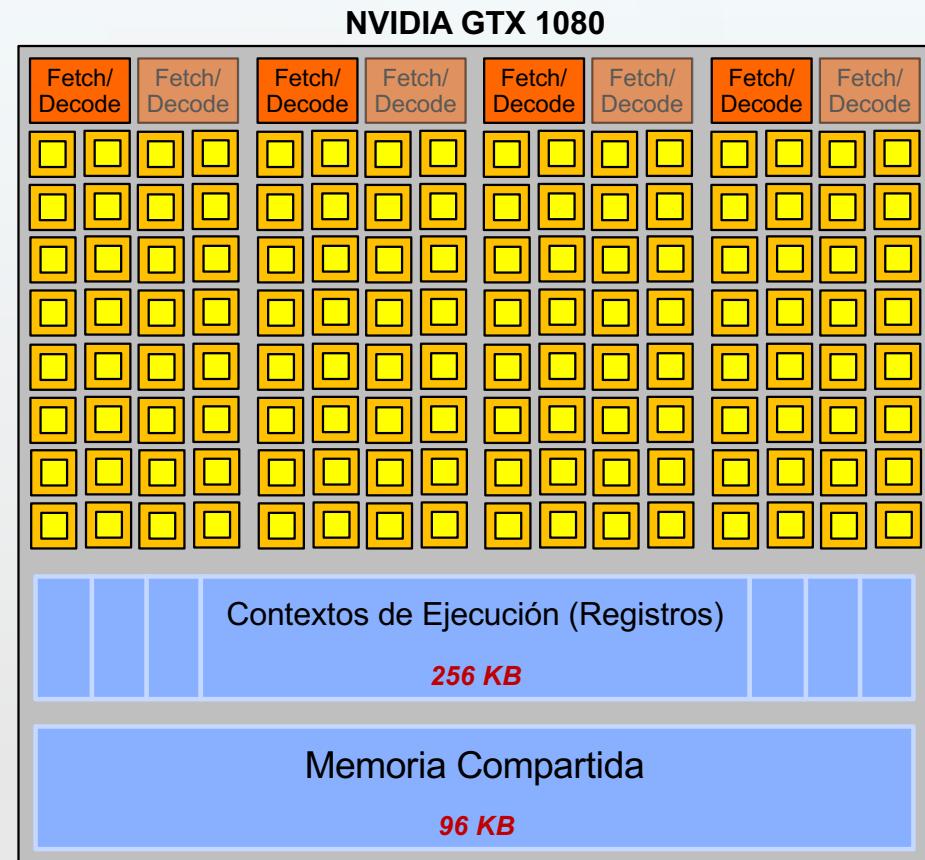
Código SIMD

Instrucción SIMD

Registro SIMD (almacena 4 valores)

GPUS

- El compilador genera un programa escalar, pero se ejecutan N instancias de ese programa en paralelo y sincronizado (típicamente N=32). A esto se le conoce como *warp*.
- El hardware ejecuta la misma instrucción de múltiples instancias sobre diferentes datos
- Una GPU es capaz de ejecutar cientos de warps simultáneamente



PARALELISMO A NIVEL DE THREADS

- Software*
- Paralelismo a nivel de peticiones
Se asignan a computadores
 - Paralelismo a nivel de threads
Se asignan a núcleos
 - Paralelismo a nivel de instrucciones
> 1 instrucción concurrentemente
 - Paralelismo a nivel de datos
> 1 dato concurrentemente
 - Paralelismo a nivel de Hardware
Todas las puertas lógicas
funcionando al mismo tiempo

Hardware

Warehouse Scale Computer



Smart Phone



Computer



Core ... Core

Memory

Input/Output

Core

Instruction Unit(s)

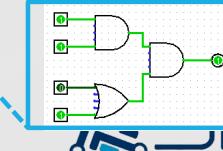


Functional Unit(s)

$$A_0+B_0 \quad A_1+B_1 \quad A_2+B_2 \quad A_3+B_3$$

Cache Memory

Logic Gates



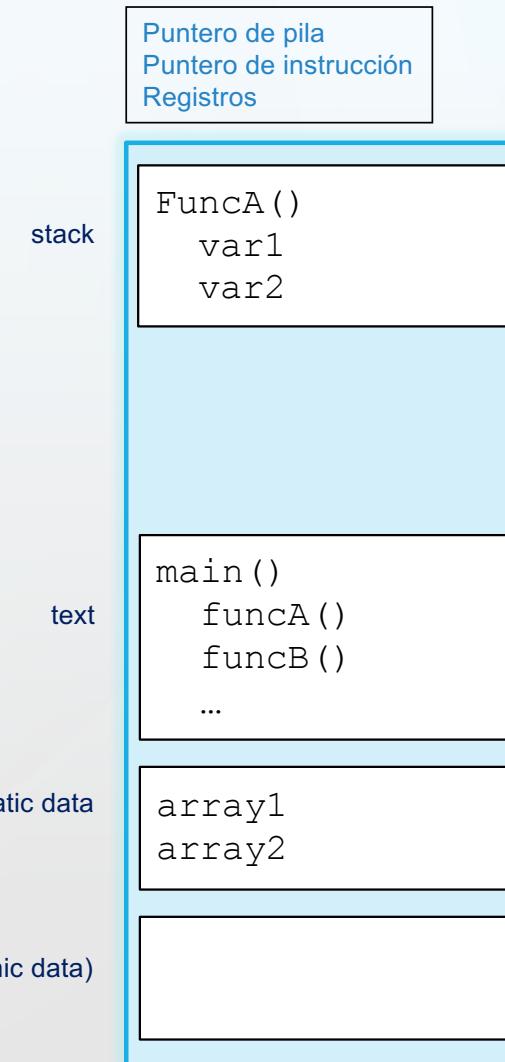
CONCEPTO DE PROCESO

- **Programa:**

- Descripción de como llevar a cabo algo (algoritmo)
- Compuesto por instrucciones y datos estáticos
- Almacenado en un fichero estático (imagen)

- **Proceso:**

- Un programa en ejecución, que progresa secuencialmente
- Está compuesto por su estado de ejecución (punteros y registros), el código (text) y la memoria de trabajo (estática y dinámica) static data
- El mismo programa puede estar ejecutándose varias veces al mismo tiempo (instancias)



CONCEPTO DE THREAD

- Idea clave: dividir el proceso en dos entidades
 - Thread: los recursos de planificación/ejecución que incluyen el puntero de instrucción, los registros y la pila
 - Proceso: resto de recursos
- Se dice que un **thread** o hebra/hilo es un **proceso ligero**
- Varios threads del mismo proceso pueden ejecutarse simultáneamente, compartiendo el mismo espacio de direcciones, pero manteniendo su propio control de flujo de ejecución

PROCESAMIENTO MULTIHILLO

- La implementación de los threads se puede hacer a nivel de software o hardware:
 - Threads controlados por el sistema operativo
 - Threads definidos por el programador usando alguna librería (por ejemplo, Pthreads)
 - Threads generados por el compilador (por ejemplo, OpenMP)
 - Threads controlados por hardware (por ejemplo, Intel Hyperthreading)
- Es complicado desarrollar software multihilo que sea eficiente, pero las ganancias en rendimiento pueden ser considerables

PROCESADOR SEGMENTADO: CONCLUSIONES

- Todos los **procesadores modernos** utilizan **pipeline** para aumentar el rendimiento
- La **frecuencia de reloj** viene limitada por la **etapa más lenta** del pipeline (por eso un diseño balanceado de las etapas es muy importante)
- Se deben **detectar y resolver los riesgos**
 - Riesgos estructurales: resueltos por un diseño correcto del pipeline
 - Riesgos de datos:
 - Paradas del pipeline (impactan en el rendimiento)
 - Anticipación – Forwarding (requiere soporte HW)
 - Riesgos de control: mover HW decisión de salto lo más pronto posible
 - Paradas del pipeline (impactan en el rendimiento)
 - Saltos retardados (requiere soporte del compilador)
 - Predicción estática y dinámica (requiere soporte HW)

DISTRIBUCIÓN DE CLASES

1	Presentación	1	6
2	Repaso monociclo	7	40
3	Multiciclo, Amdahl	41	60
4	Ripes, Resolución de ejercicios		
5	Segmentado	61	86
6	Riesgos	87	111
7	Soluciones SW	112	143
8	Soluciones HW de datos	144	183
9	Soluciones HW de control	184	238
10	Predictores dinámicos, ejercicio	239	263
11	Resolución de ejercicios		
12	Paralelismo avanzado, Going Faster	264	294
13	Going Faster		
14	Resolución de ejercicios y dudas		
15	Primer parcial		