ESTRUCTURA DE COMPUTADORES

Tema 1. Segmentación

Dpto. Arquitectura de Computadores y Automática

Universidad Complutense de Madrid

ÍNDICE

1. RISC-V
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Rendimiento del procesador segmentado
7. Operaciones multiciclo
8. Rendimiento en los procesadores

Bibliografía

- Computer Organization and Design. The Hardware/Software Interface. RISC-V edition. David A. Patterson & John L. Hennessy, Morgan Kaufmann 2018.
- Digital design and computer architecture, RISC-V Edition. S. Harris, D. Harris, Morgan Kaufmann 2022.
- Computer architecture: A quantitative approach. John L. Hennessy & David A. Patterson , Morgan Kaufmann 2017, 6ª ed.

ÍNDICE

1. **RISC-V**
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Rendimiento del procesador segmentado
7. Operaciones multiciclo
8. Rendimiento en los procesadores

FORMATO DE LAS INSTRUCCIONES

Principios de diseño

1. **La regularidad facilita la simplicidad de diseño**
2. **Hacer rápido el caso común**
3. **Más pequeño es más rápido**
4. **Buen diseño exige buenos compromisos**

- **Principio de diseño 1: La regularidad facilita la simplicidad de diseño**
  - Datos de 32-bits, instrucciones de 32-bits
  - Por simplicidad de diseño se preferiría un sólo formato, pero las instrucciones tienen diferentes necesidades
- **Principio de diseño 4: Buen diseño exige buenos compromisos**
  - Tener varios formatos da flexibilidad
    - `add`, `sub`: usan 3 registros como operandos
    - `lw`, `sw`: usan 2 registros y una constante como operandos
  - El número de formatos debe mantenerse reducido para cumplir con los principios 1 y 4

Formato de las instrucciones

- **Instrucciones de 32 bits**
- **Pocos formatos de instrucción:**

7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
funct7	rs2	rs1	funct3	rd	op	**R-Type**
$imm_{11:0}$		rs1	funct3	rd	op	**I-Type**
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	**S-Type**
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	**B-Type**
$imm_{31:12}$				rd	op	**U-Type**
$imm_{20,10:1,11,19:12}$				rd	op	**J-Type**
20 bits				5 bits	7 bits	

DISEÑO DE LA RUTA DE DATOS

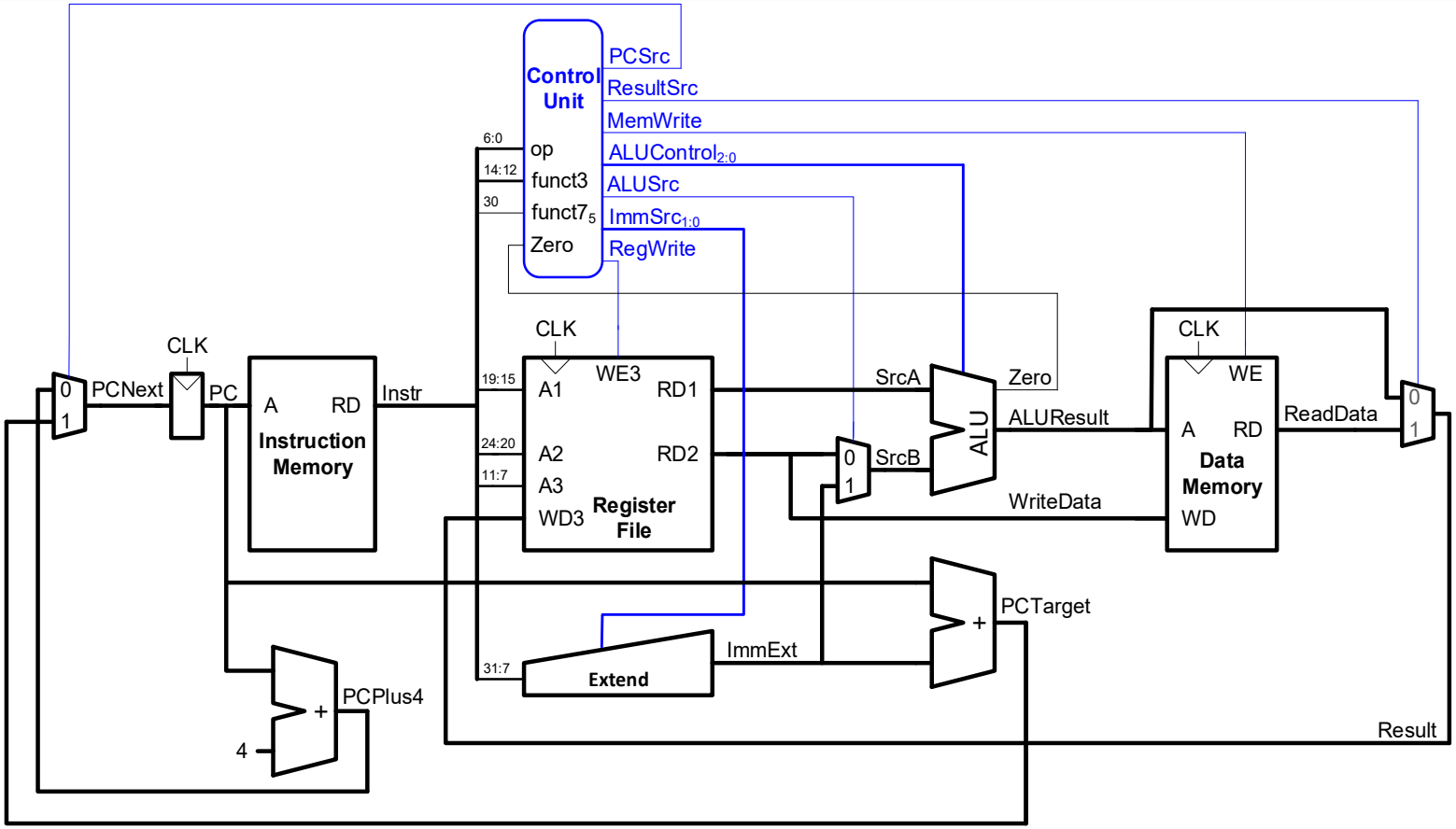
Metodología de diseño

- **Paso 1**: **Analizar el repertorio de instrucciones** para obtener los requisitos de la ruta de datos.
  - La ruta de datos debe incluir tantos **elementos de almacenamiento** como registros sean visibles por el programador. Además puede tener otros elementos de almacenamiento transparentes.
  - La ruta de datos debe incluir tantos tipos de **elementos operativos** como tipos de operaciones de cálculo se indiquen en el repertorio de instrucciones.
  - El significado de cada instrucción vendrá dado por un conjunto de transferencias entre registros. La ruta de datos debe ser capaz de soportar dichas transferencias.
- **Paso 2: Establecer la metodología de temporización.**
  - **Monociclo (CPI = 1)**: todas las transferencias entre registros implicadas en una instrucción se realizan en un único ciclo de reloj.
  - **Multiciclo (CPI > 1)**: las transferencias entre registros implicadas en una instrucción se reparten entre varios ciclos de reloj.
  - **Segmentada.** ?
- **Paso 3: Seleccionar el conjunto de módulos** (de almacenamiento, operativos e interconexión) que forman la ruta de datos.
- **Paso 4: Ensamblar la ruta de datos** de modo que se cumplan los requisitos impuestos por el repertorio, **localizando los puntos de control.**
- **Paso 5: Determinar los valores de los puntos de control** analizando las transferencias entre registros incluidas en cada instrucción.
- **Paso 6: Diseñar la lógica de control.**

Diseño de la ruta de datos

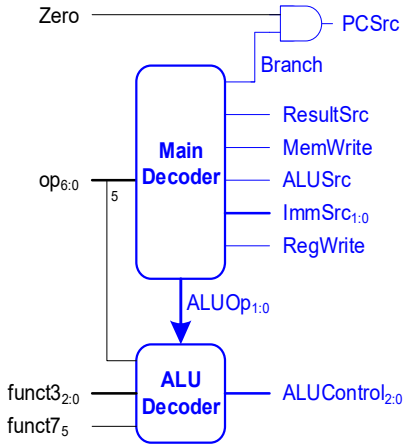
- **Vamos a implementar un subconjunto de todo el repertorio:**
  - **Instrucciones aritmético lógicas tipo-R**
    - `add`
    - `sub`
    - `and`
    - `or`
    - `slt`
  - **Instrucciones de memoria:**
    - `lw`
    - `sw`
  - **Instrucciones de salto:**
    - `beq`

RUTA DE DATOS Y CONTROL MONOCICLO

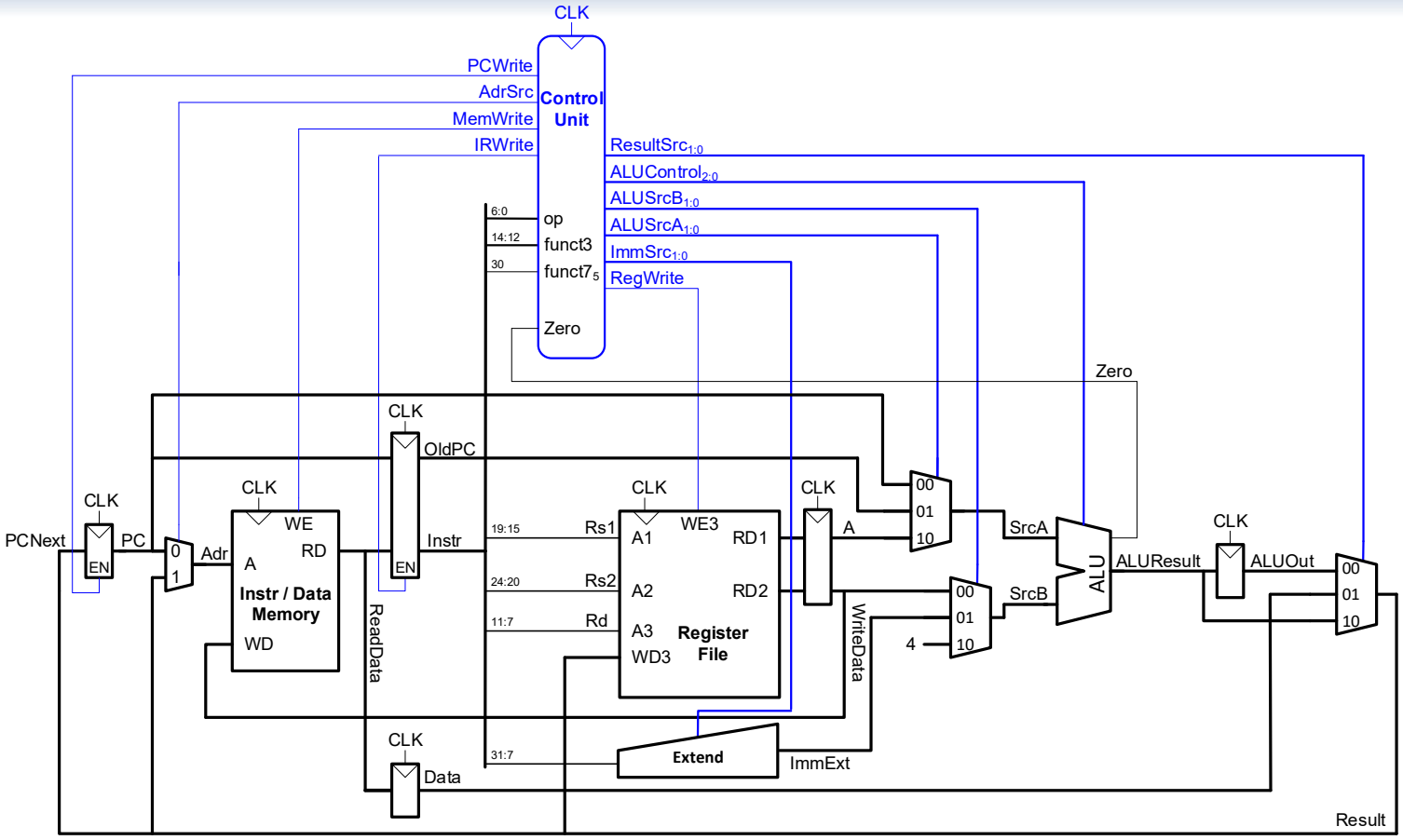


RUTA DE DATOS Y CONTROL MONOCICLO

op	Instr.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	`lw`	1	00	1	0	1	0	00
35	`sw`	0	01	1	1	X	0	00
51	R-type	1	XX	0	0	0	0	10
99	`beq`	0	10	0	0	X	1	01



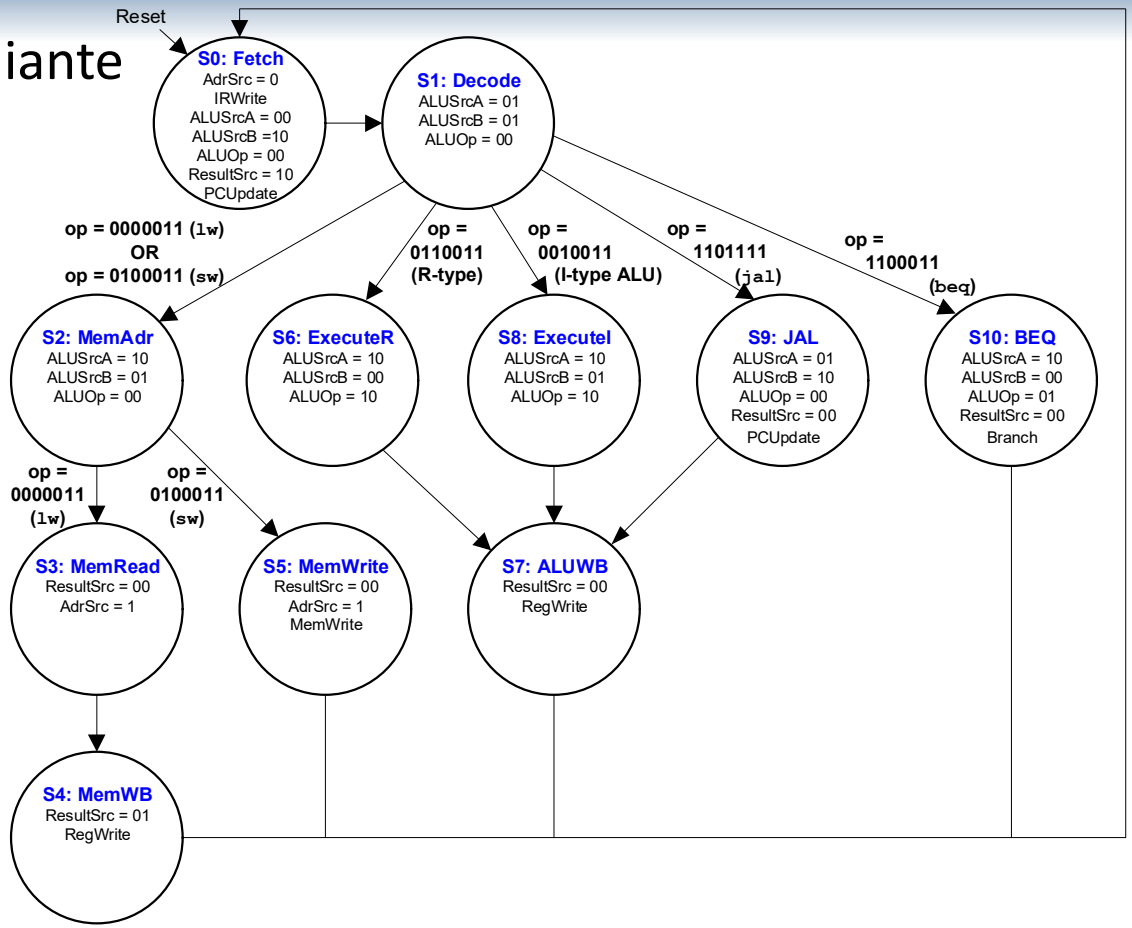
RUTA DE DATOS Y CONTROL MULTICICLO



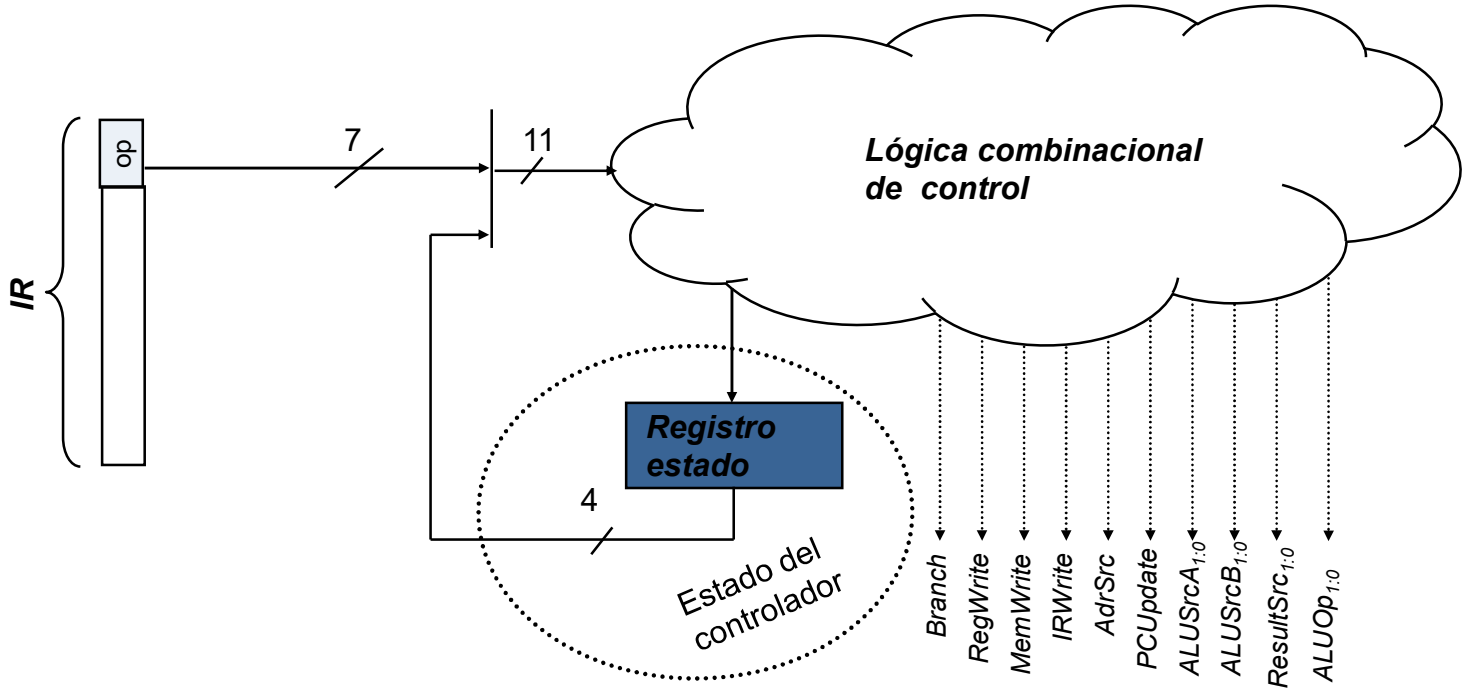
UNIDAD DE CONTROL MULTICICLO

- Implementación mediante una ROM

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
ExecuteI	ALUOut ← rs1 op imm
ALUWB	rd ← ALUOut
BEQ	ALUResult = rs1-rs2; if Zero, PC ← ALUOut
JAL	PC ← ALUOut; ALUOut ← PC+4



UNIDAD DE CONTROL MULTICICLO



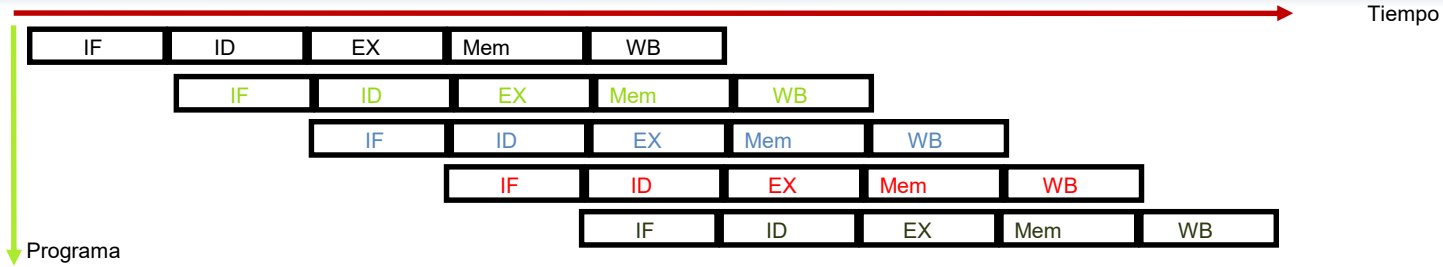
UNIDAD DE CONTROL MULTICICLO

Estado actual	op	Instrucción	Estado siguiente	Branch	RegWrite	MemWrite	IRWrite	PCUpdate	AdrSrc	$ALUSrcA_{1:0}$	$ALUSrcB_{1:0}$	$ResultSrc_{1:0}$	$ALUOp_{1:0}$
0000	X		0001	0	0	0	1	1	0	00	10	10	00
0001	0000011	lw	0010	0	0	0	0	0	X	01	01	X	00
0001	0100011	sw	0010	0	0	0	0	0	X	01	01	X	00
0001	0110011	Tipo R	0110	0	0	0	0	0	X	01	01	X	00
0001	0010011	Tipo I ALU	1000	0	0	0	0	0	X	01	01	X	00
0001	1101111	jal	1001	0	0	0	0	0	X	01	01	X	00
0001	1100011	beq	1010	0	0	0	0	0	X	01	01	X	00
0010	0000011	lw	0011	0	0	0	0	0	X	10	01	X	00
0010	0100011	sw	0101	0	0	0	0	0	X	10	01	X	00
0011	X		0100	0	0	0	0	0	1	X	X	00	X
0100	X		0000	0	1	0	0	0	X	X	X	01	X
0101	X		0000	0	0	1	0	0	1	X	X	00	X
0110	X		0111	0	0	0	0	0	X	10	00	X	10
0111	X		0000	0	0	0	1	0	X	X	X	00	X
1000	X		0111	0	0	0	0	0	X	10	01	X	10
1001	X		0011	0	0	0	0	1	X	01	10	00	00
1010	X		0000	1	0	0	0	0	X	10	00	00	01

ÍNDICE

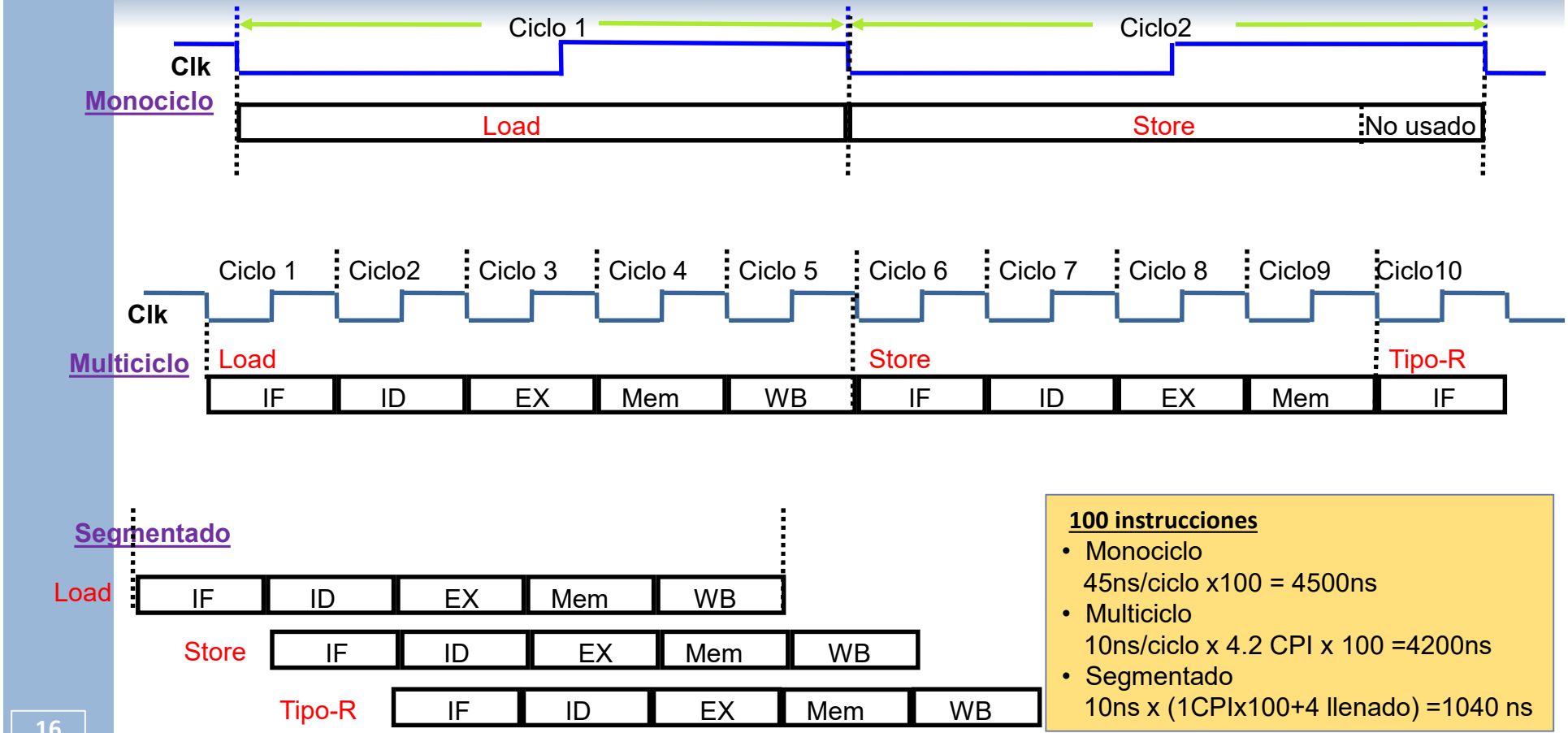
1. RISC-V
2. **Segmentación**
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Rendimiento del procesador segmentado
7. Operaciones multiciclo
8. Rendimiento en los procesadores

SEGMENTACIÓN



- Cada etapa opera en paralelo con otras etapas pero sobre instrucciones diferentes
- Una instrucción para ejecutarse tiene que atravesar todas y cada una de las etapas del pipeline
- El ciclo de reloj viene determinado por la etapa más lenta
- El orden de las etapas es el mismo para todas las instrucciones
  - A partir del ciclo 5
    - Sale una instrucción cada ciclo de reloj
    - CPI=1
  - Los 4 primeros ciclos se llaman de *llenado del pipeline*.
  - $CPI_{Ideal}=1$

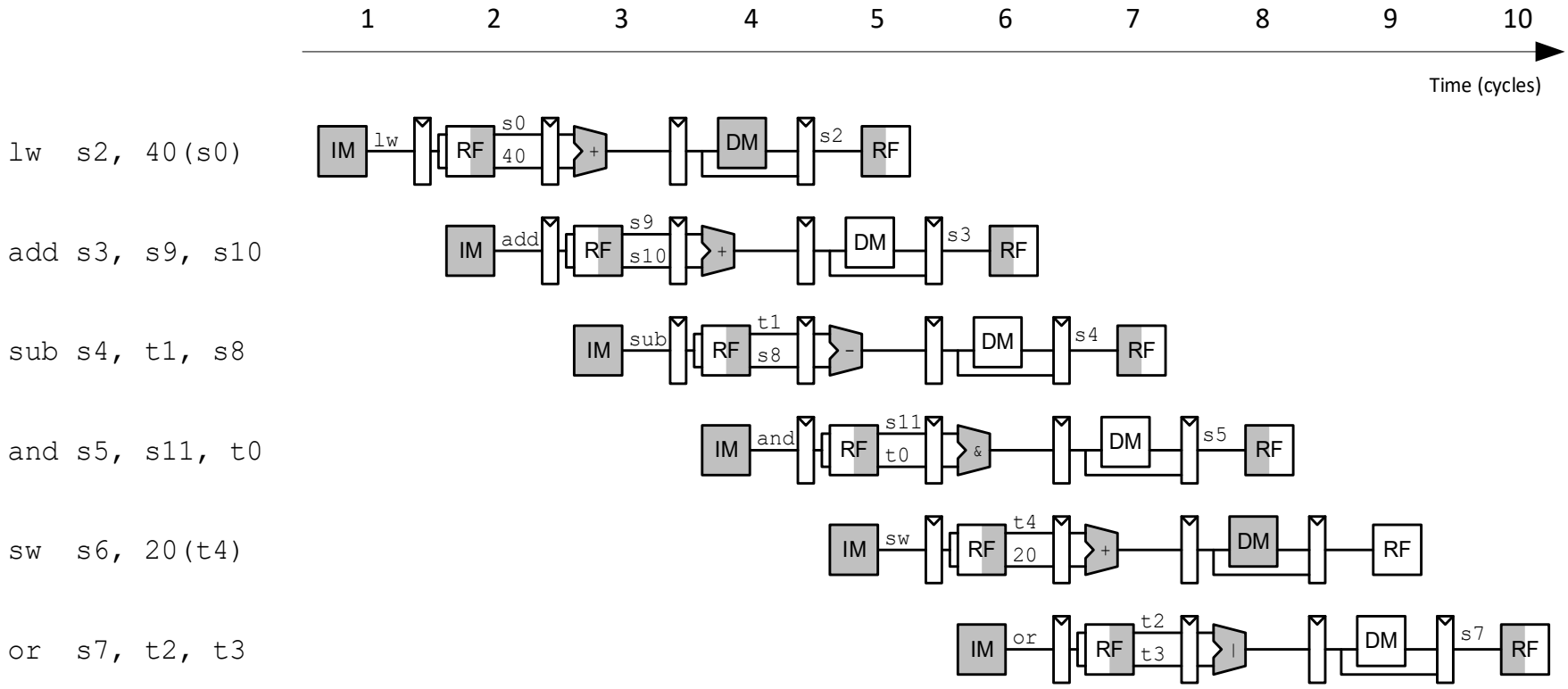
Segmentación



100 instrucciones

- Monociclo
  45ns/ciclo x100 = 4500ns
- Multiciclo
  10ns/ciclo x 4.2 CPI x 100 =4200ns
- Segmentado
  10ns x (1CPIx100+4 llenado) =1040 ns

- Representación gráfica del pipeline:



SEGMENTACIÓN

- ¿Qué características facilitan la segmentación?
  - Todas las instrucciones de igual anchura
  - Pocos formatos de instrucción
  - Búsqueda de operandos en memoria sólo en operaciones de carga y almacenamiento
- ¿Qué dificulta la segmentación?
  - **Riesgos**: Situaciones que impiden que en cada ciclo se inicie la ejecución de una nueva instrucción
    - **Estructurales**. Se producen cuando dos instrucciones tratan de utilizar el mismo recurso en el mismo ciclo.
    - **De datos**. Se producen al intentar utilizar un dato antes de que esté actualizado. Mantenimiento del orden estricto de lecturas y escrituras.
    - **De control**. Se producen al intentar tomar una decisión sobre una condición todavía no evaluada.

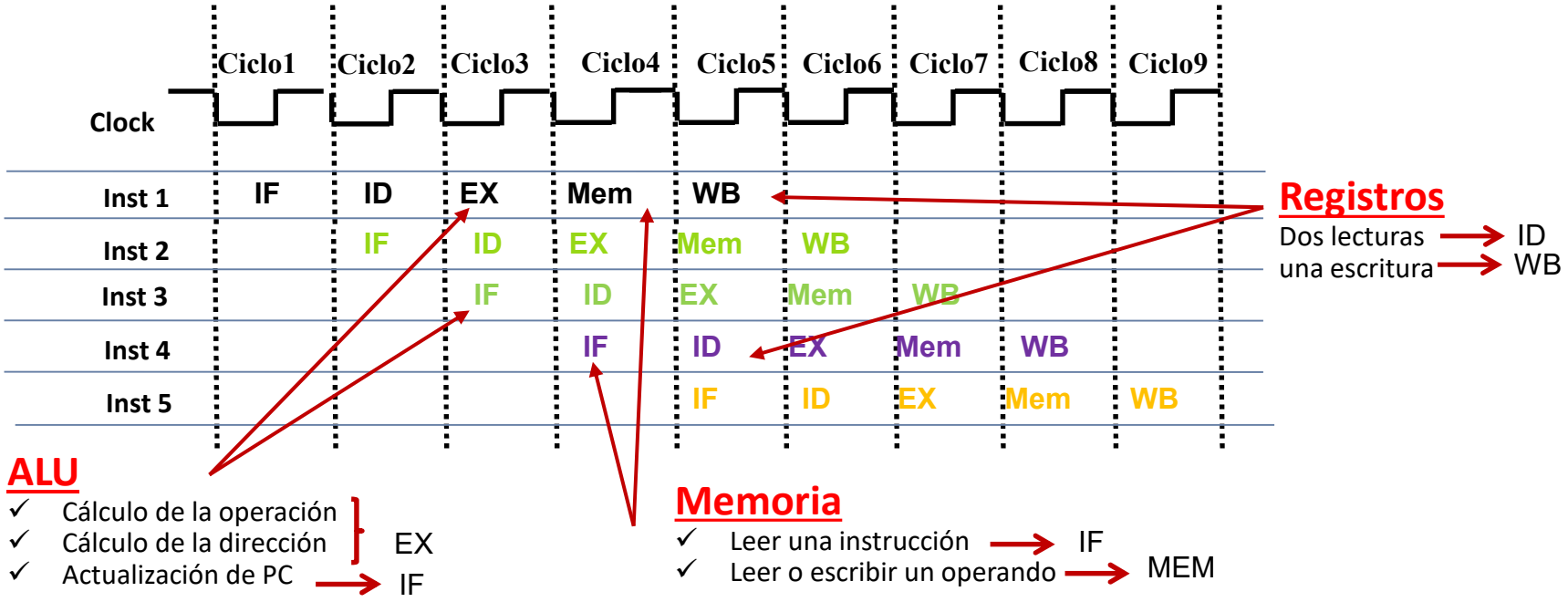
    Los riesgos se deben detectar y resolver
  - **Gestión de interrupciones**

ÍNDICE

1. RISC-V
2. Segmentación
3. **Riesgos estructurales**
4. Riesgos de datos
5. Riesgos de control
6. Rendimiento del procesador segmentado
7. Operaciones multiciclo
8. Rendimiento en los procesadores

Segmentación: Riesgos estructurales

- Se producen cuando dos instrucciones tratan de utilizar el mismo recurso en el mismo ciclo
  - ***Objetivo:*** **Ejecutar sin conflicto cualquier combinación de instrucciones**

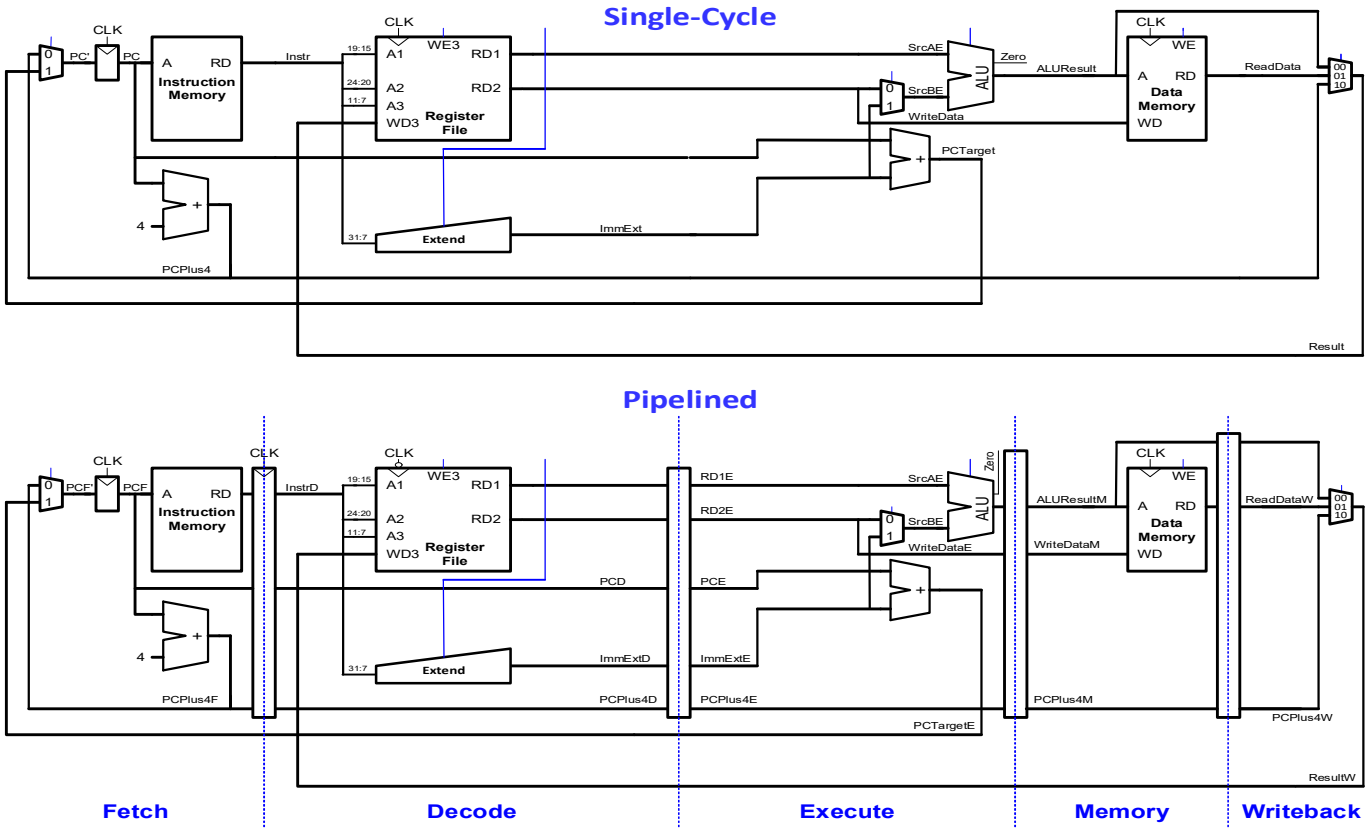


SEGMENTACIÓN: RIESGOS ESTRUCTURALES

- ¿Cómo resolver los riesgos estructurales?
  - Duplicar los recursos que se necesitan en el mismo ciclo:
    - ALU y dos sumadores;
    - memoria de instrucciones y de datos separadas.
  - El Banco de Registros no es problema porque tiene puertos para leer de dos registros y escribir en un registro a la vez.

RUTA DE DATOS SEGMENTADA

Se añaden registros para separar las etapas a la ruta de datos monociclo



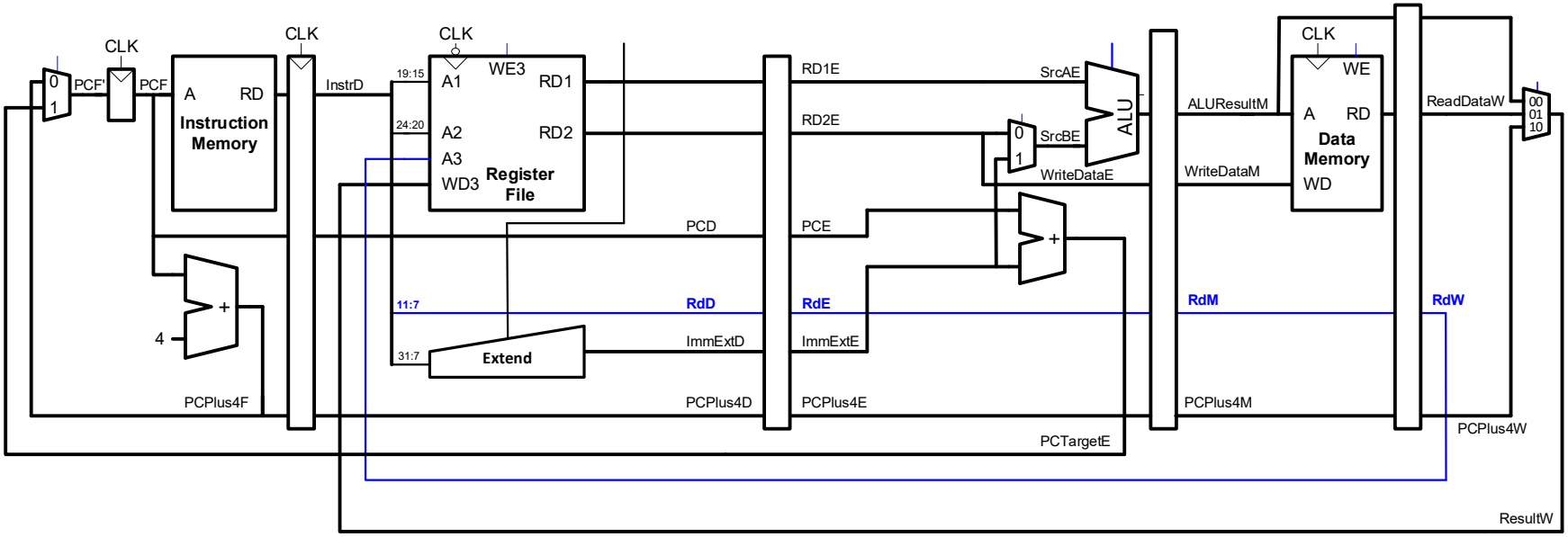
Las señales en el procesador con pipeline se etiquetan añadiendo la inicial del estado (ej. PC**F**, PC**D**, PC**E**).

Todas las señales asociadas con una instrucción deben avanzar al unisono a través del pipeline.

¿Hay algo erroneo en esta ruta de datos?

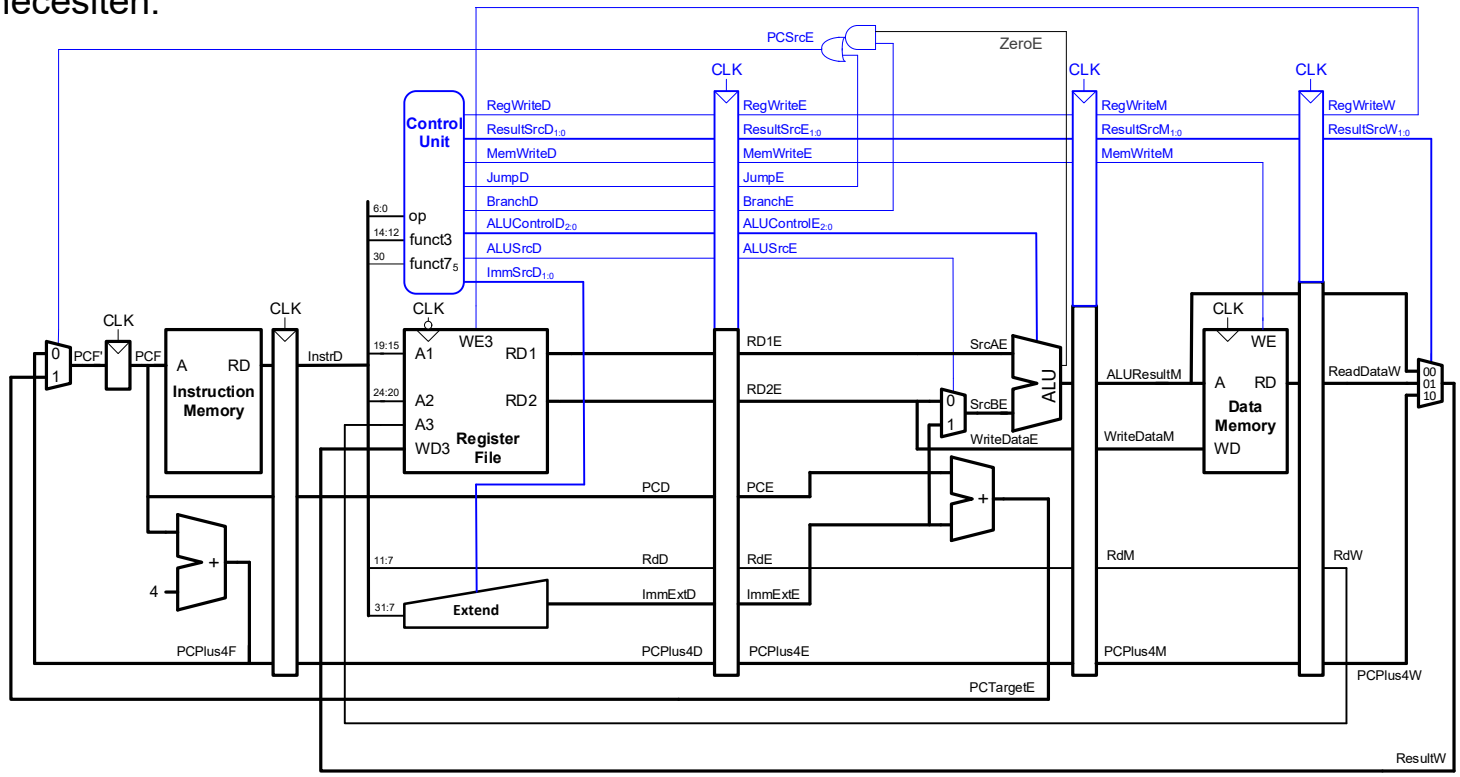
RUTA DE DATOS SEGMENTADA

- Ruta de datos corregida
  - ***Rd*** debe llegar al mismo tiempo que ***Result***
  - El banco de registros se escribe en el **flanco de bajada** del *CLK*

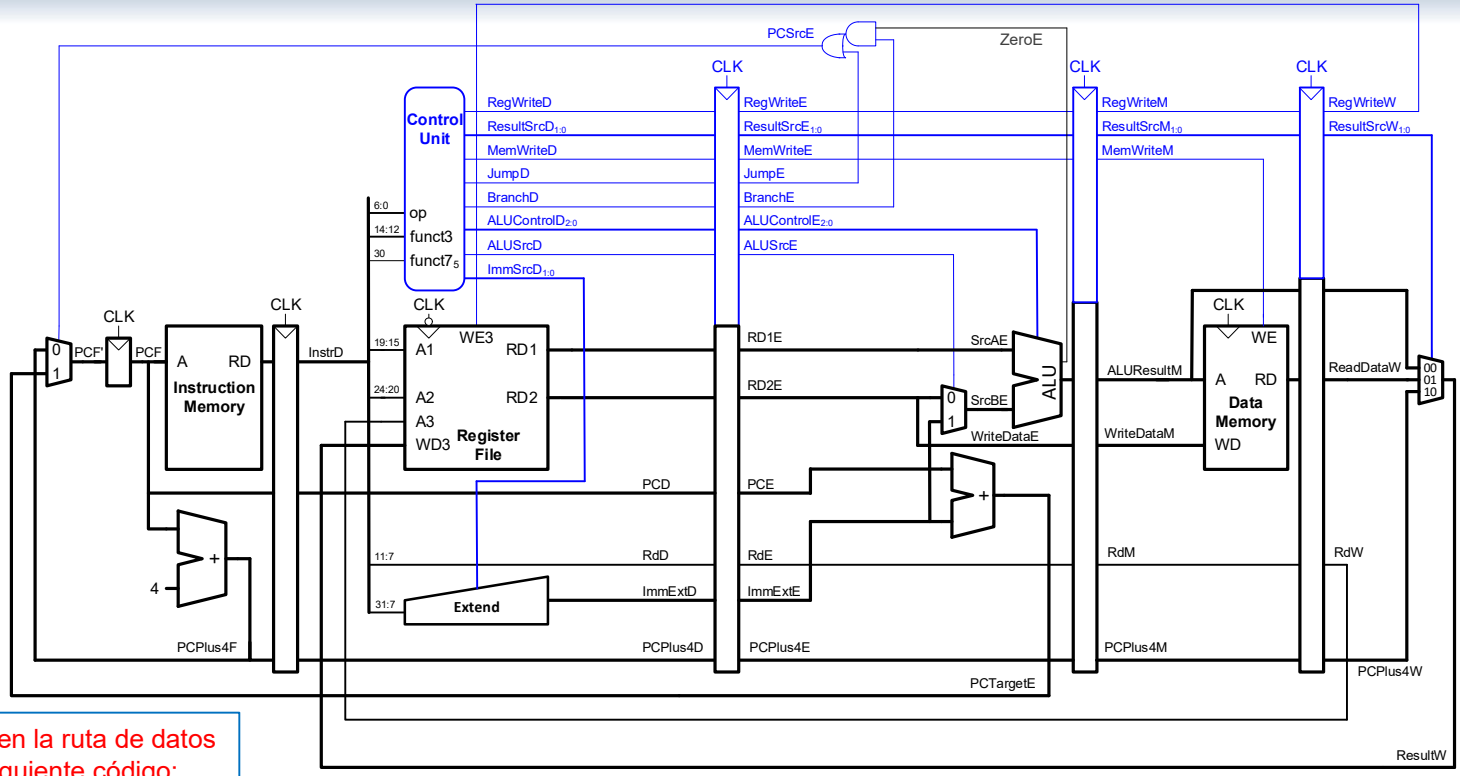


UNIDAD DE CONTROL SEGMENTADA

- La unidad de control es similar a la de la ruta monociclo.
- Los valores de los puntos de control se van propagando hasta la etapa del pipeline en que se necesiten.



UNIDAD DE CONTROL SEGMENTADA



Ver que ocurre en la ruta de datos al ejecutar el siguiente código:

```
add  x3, x9, x11
lw   x5, 4(x7)
sub  x5, x6, x8
ori  x6, x7, 4
```

ÍNDICE

1. RISC-V
2. Segmentación
3. Riesgos estructurales
4. **Riesgos de datos**
5. Riesgos de control
6. Rendimiento del procesador segmentado
7. Operaciones multiciclo
8. Rendimiento en los procesadores

Riesgos de datos

- Se produce un riesgo de datos si existe dependencia entre los datos de dos instrucciones que se ejecutan concurrentemente
- Este tipo de riesgos aumentan en operaciones multiciclo
- Tres tipos diferentes:
  - **Lectura después de escritura (LDE)**
  - **Escritura después de lectura (EDL)**
  - **Escritura después de escritura (EDE)**

RIESGOS DE DATOS

- **Lectura después de escritura** (LDE)

  *add **x1**, x2, x3* *– escribe el registro x1*

  *add x4, **x1**, x2* *—lee el registro x1*

  - Se produce riesgo si x1 se lee antes de que lo escriba la primera instrucción

- **Escritura después de lectura** (EDL)

  *add x1, **x4**, x3* *– lee el registro x4*

  *add **x4**, x5, x2* *—escribe el registro x4*

  - Se produce riesgo si x4 se escribe antes de que lo lea la primera instrucción

- **Escritura después de escritura** (EDE)

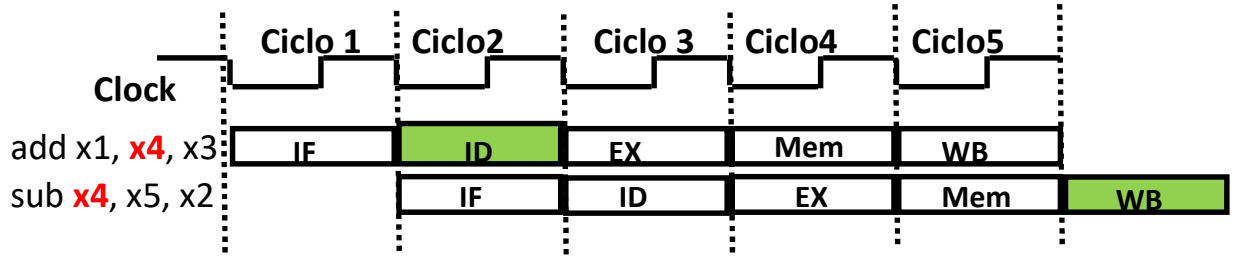
  *add **x4**, x2, x3* *– escribe el registro x4*

  *add **x4**, x1, x2* *—escribe el registro x4*

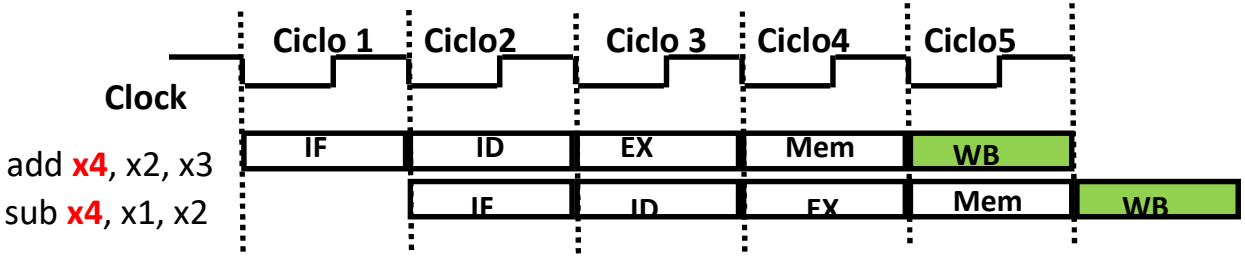
  - Se produce riesgo si x4 de la primera instrucción se escribe después de que lo escriba la segunda instrucción

RIESGOS DE DATOS: EDL Y EDE

- **Escritura después de lectura: EDL**



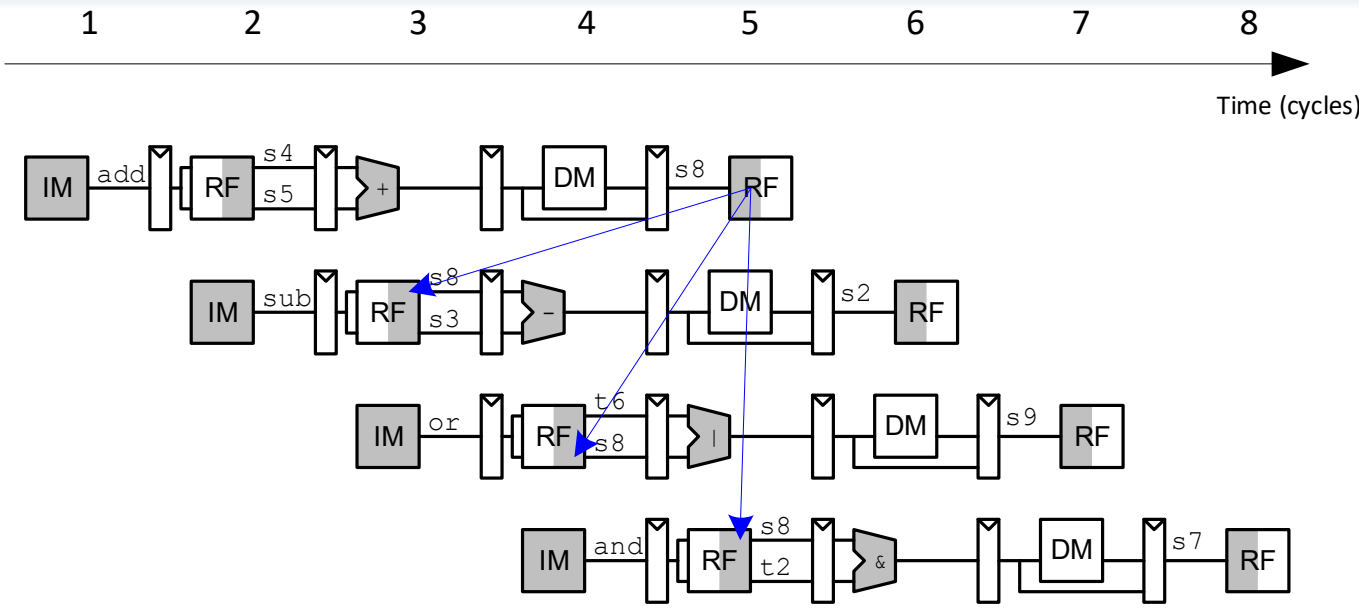
- **Escritura después de escritura: EDE**



- **Estos riesgos NO se dan en el pipeline cuando todas las instrucciones tienen igual duración**
  - Se leen los registros en el final de la segunda etapa
  - Las instrucciones escriben en el banco de registros en la última etapa

RIESGOS DE DATOS: LDE

```
add s8, s4, s5
sub s2, s8, s3
or  s9, t6, s8
and s7, s8, t2
```

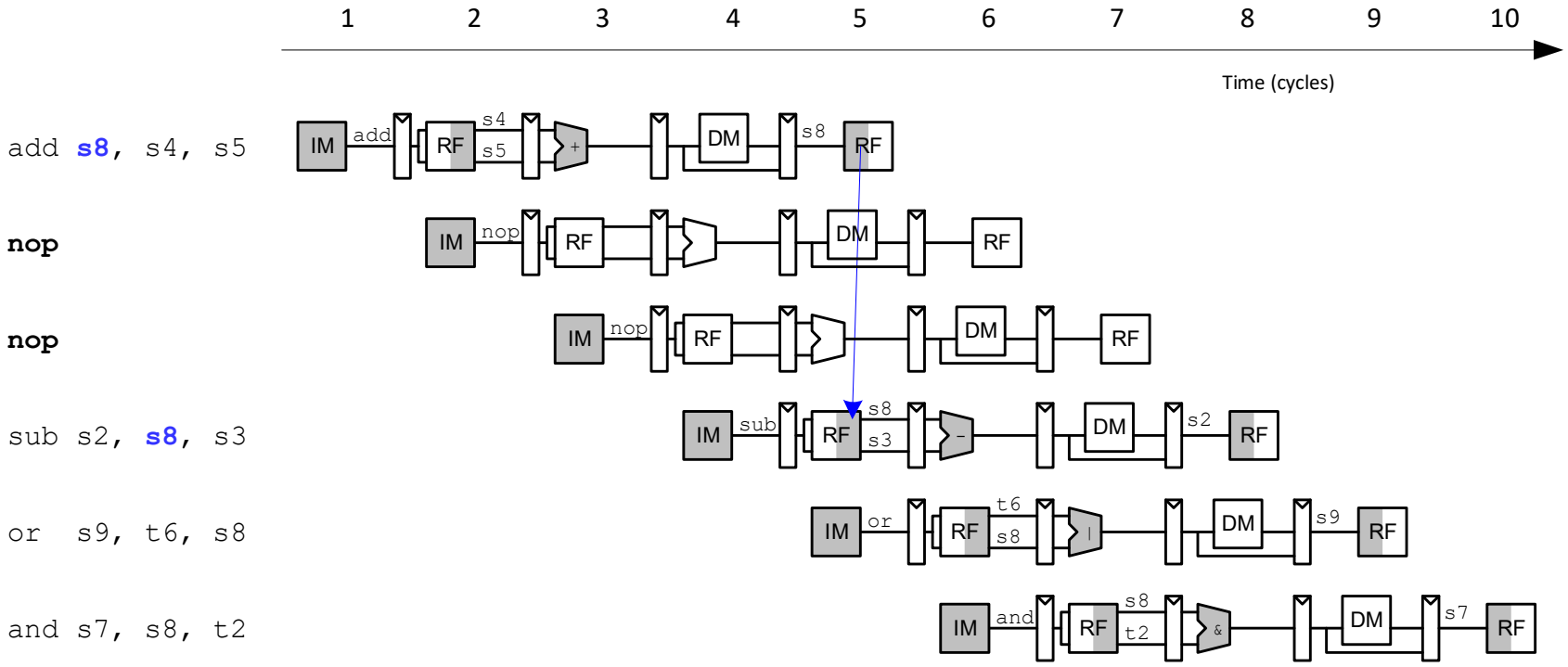


Las 2 siguientes instrucciones no tienen el valor de **s8** actualizado, leen un valor antiguo.
La instrucción *and* ya puede leer el valor actualizado dado que en el BR se escribe en la primera mitad del ciclo y se lee en la segunda mitad.

RIESGOS DE DATOS: LDE

¿Cómo resolver los riesgos LDE? Sin modificar el hardware

- Introducir instrucciones Nop entre las dependencias
- Reordenar el código para que las dependencias se alejen



RIESGOS DE DATOS: LDE

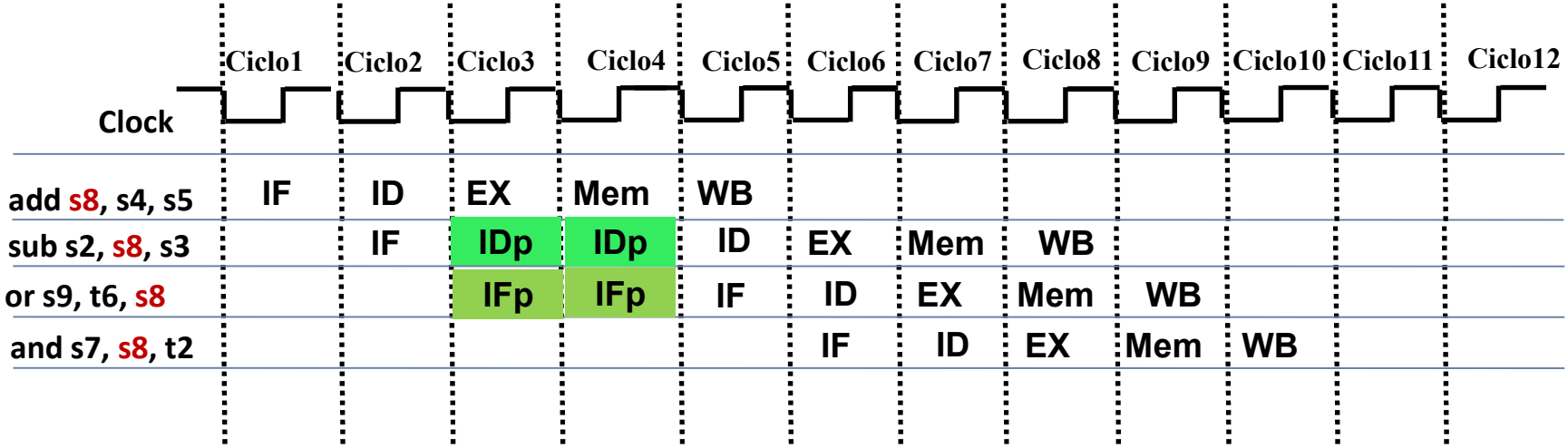
- ¿Cómo resolver los riesgos LDE?: Modificando el hardware

Solución 1: Detener el pipeline en la etapa de decodificación

- ¿Cómo afectan las paradas a la ejecución de un programa?
  - Las instrucciones que están en etapas anteriores a la etapa de parada también se paran
  - Las instrucciones que están en etapas posteriores a la etapa de parada siguen ejecutándose

¿Cómo resolver los riesgos LDE?: Modificando el hardware

Solución 1: Detener el pipeline en la etapa de decodificación



2 ciclos de espera

IDp ✓ID_P **parada por riesgo LDE** entre instrucción 1 e instrucción 2

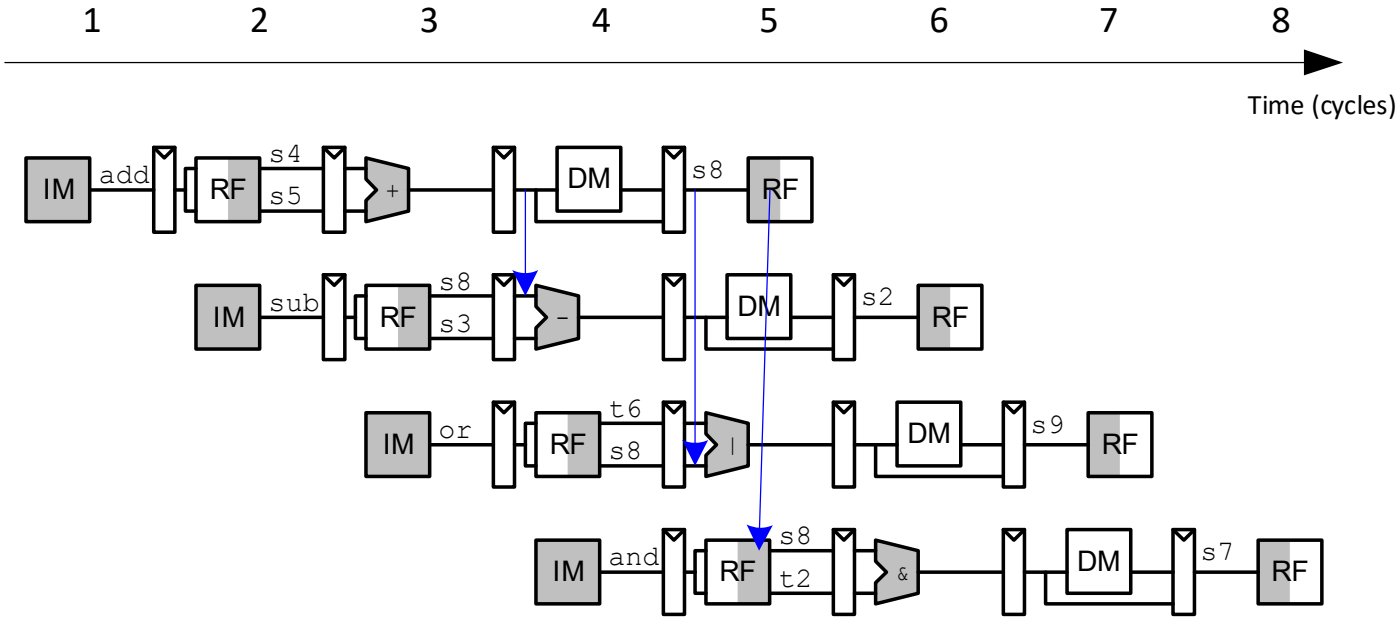
IFp ✓IF_P **parada por riesgo estructural**, la etapa ID está ocupada por la instrucción anterior

¿Cómo resolver los riesgos LDE?: Modificando el hardware

Solución 2: Cortocircuito (forwarding)

- Los datos están disponibles en los buses internos antes de que se escriban en el BR
- Realimentar los datos desde los buses internos a la etapa EX

```
add s8, s4, s5
sub s2, s8, s3
or  s9, t6, s8
and s7, s8, t2
```



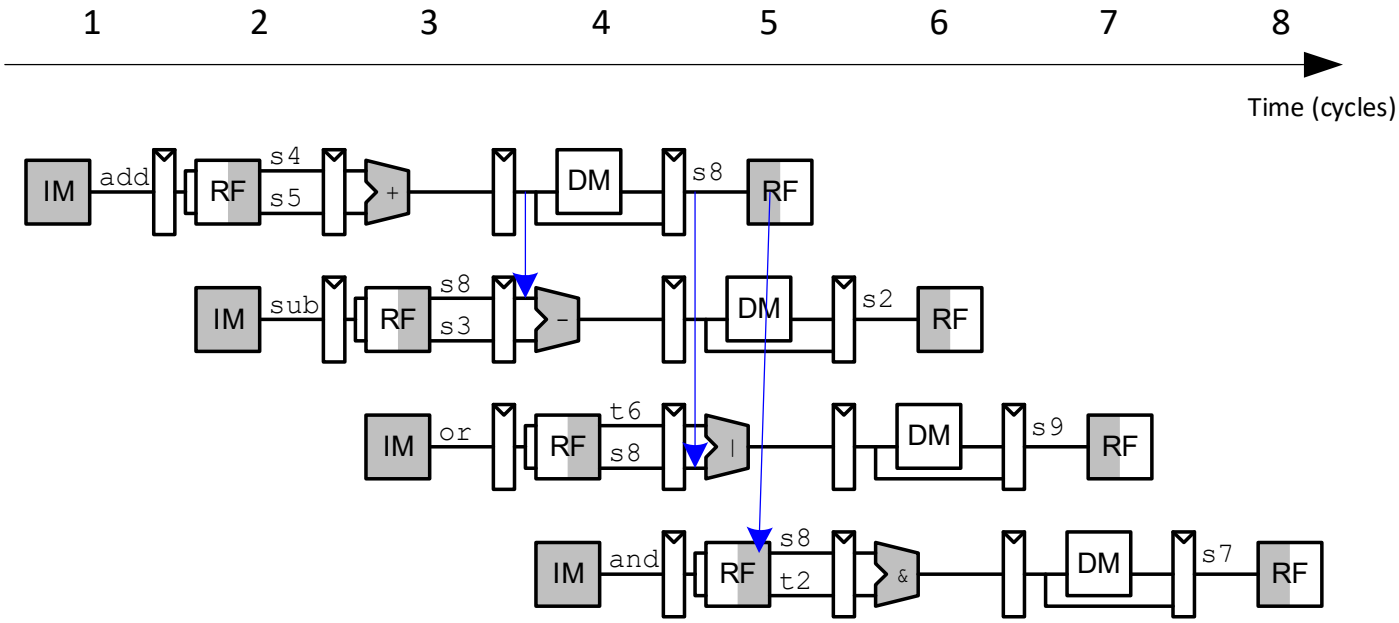
¿Cómo resolver los riesgos LDE? Cortocircuito (forwarding)

- Comprobar si el registro fuente en la etapa Execute coincide con el registro destino de la instrucción en la etapa Memoria o Writeback.

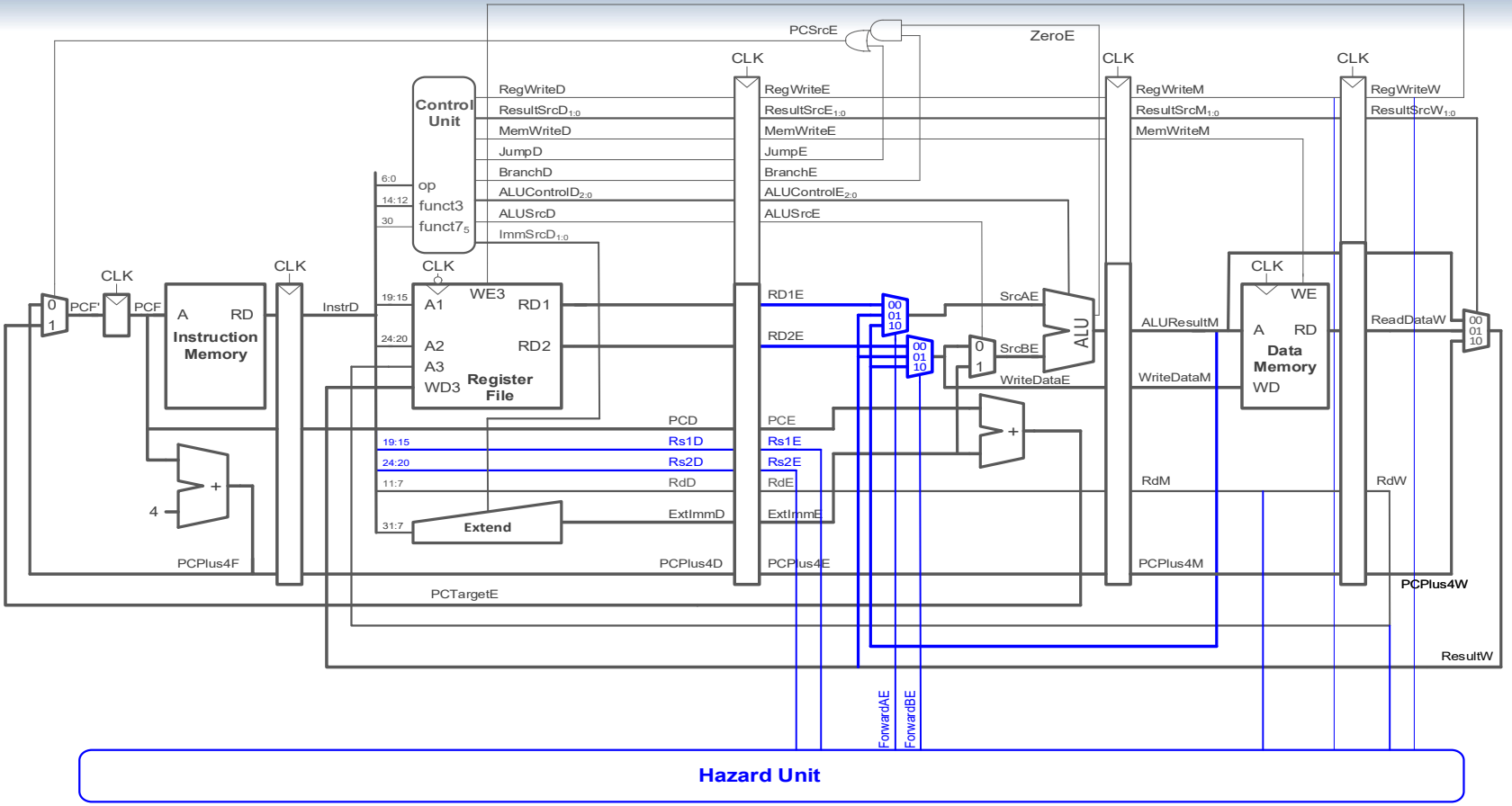
  Si es así, realimentar el resultado

  Necesitaremos multiplexores a la entrada de la ALU

```
add s8, s4, s5
sub s2, s8, s3
or  s9, t6, s8
and s7, s8, t2
```



LDE: UNIDAD DE DETECCIÓN DE CONFLICTOS



RIESGOS DE DATOS

- **Caso 1:** ¿*Rs1* ó *Rs2 de la etapa* **Execute** coincide con *Rd* de la etapa **Memory**?
  Realimenta desde la etapa Memory
- **Caso 2:** ¿*Rs1* ó *Rs2 de la etapa* **Execute** coincide con *Rd* de la etapa **WriteBack**?
  Realimenta desde la etapa Writeback
- **Caso 3:** En los otros casos usa el valor leído desde el banco de registros (como siempre)

Ecuaciones para *Rs1*:

```
if      ((Rs1E == RdM) AND RegWriteM)        // Caso 1
                ForwardAE = 10
else if ((Rs1E == RdW) AND RegWriteW)        // Caso 2
                ForwardAE = 01
else            ForwardAE = 00               // Caso 3
```

*Las ecuaciones para **ForwardBE** son similares (reemplaza **Rs1E** con **Rs2E**)*

RIESGOS DE DATOS

- **Caso 1:** ¿*Rs1* ó *Rs2 de la etapa* **Execute** coincide con *Rd* de la etapa **Memory**?
  Realimenta desde la etapa Memory
- **Caso 2:** ¿*Rs1* ó *Rs2 de la etapa* **Execute** coincide con *Rd* de la etapa **WriteBack**?
  Realimenta desde la etapa Writeback
- **Caso 3:** En los otros casos usa el valor leido desde el banco de registros (como siempre)

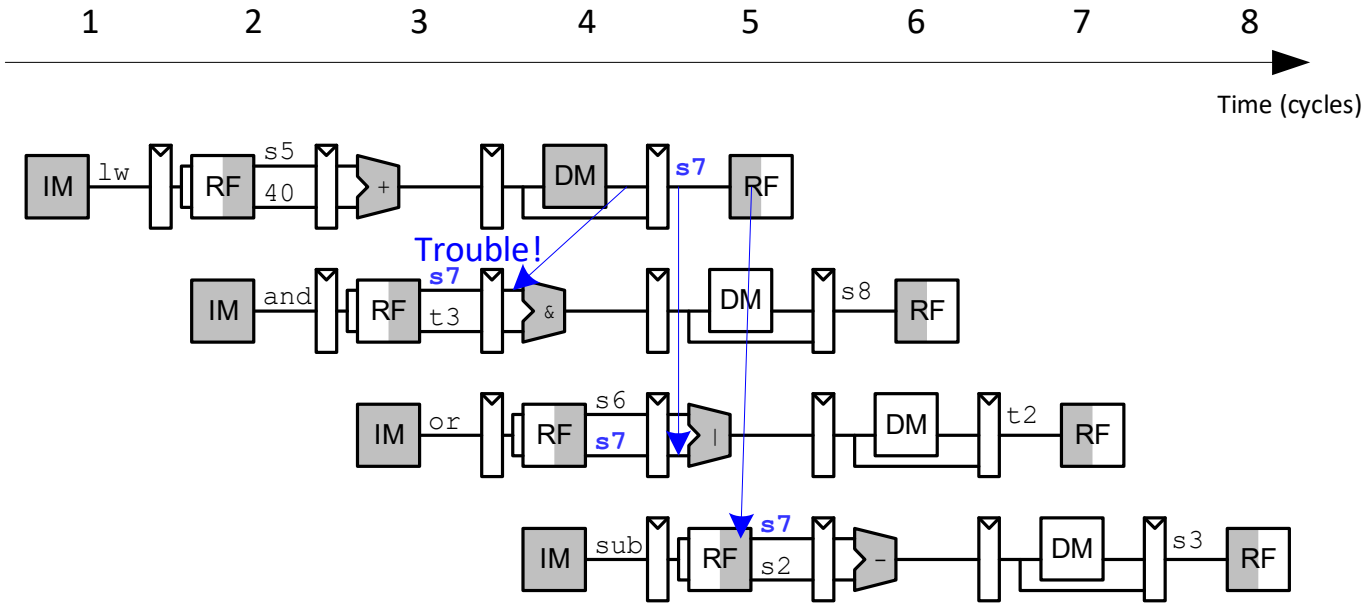
Ecuaciones para *Rs1*:

```
if      ((Rs1E == RdM) AND RegWriteM) AND (Rs1E != 0)     // Caso 1
                ForwardAE = 10
else if ((Rs1E == RdW) AND RegWriteW) AND (Rs1E != 0)     // Caso 2
                ForwardAE = 01
else            ForwardAE = 00                            // Caso 3
```

*Las ecuaciones para **ForwardBE** son similares (reemplaza **Rs1E** con **Rs2E**)*

RIESGOS DE DATOS TRAS INSTRUCCIÓN LW

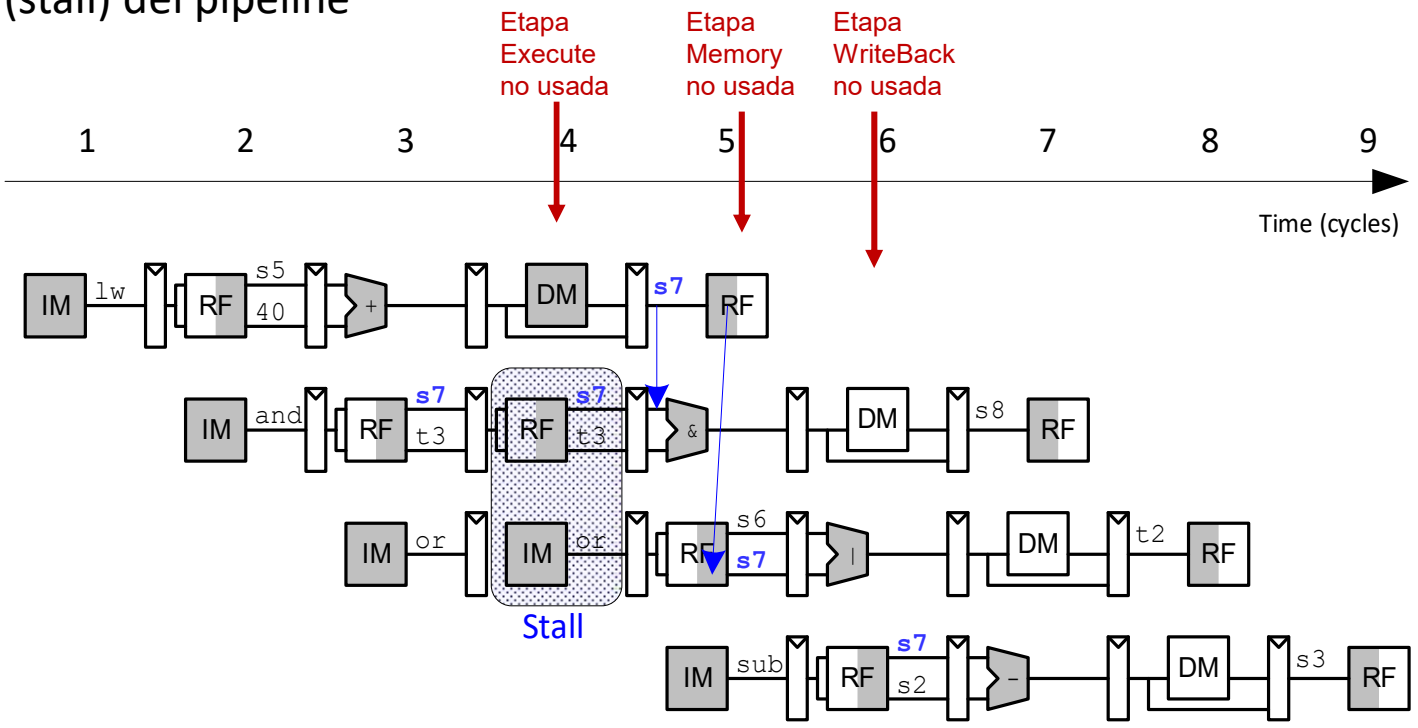
```
lw   s7, 40(s5)
and s8, s7, t3
or   t2, s6, s7
sub s3, s7, s2
```



Riesgos de datos tras instrucción lw

- Solución a los riesgos de datos tras instrucción lw
  - Parada (stall) del pipeline

```
lw   s7, 40(s5)
and  s8, s7, t3
or   t2, s6, s7
sub  s3, s7, s2
```



RIESGOS DE DATOS TRAS INSTRUCCIÓN LW

- Lógica del *stall*

  ¿Es algún **registro fuente en la etapa Decode** el mismo que el **registro destino en la etapa Execute**?

  y

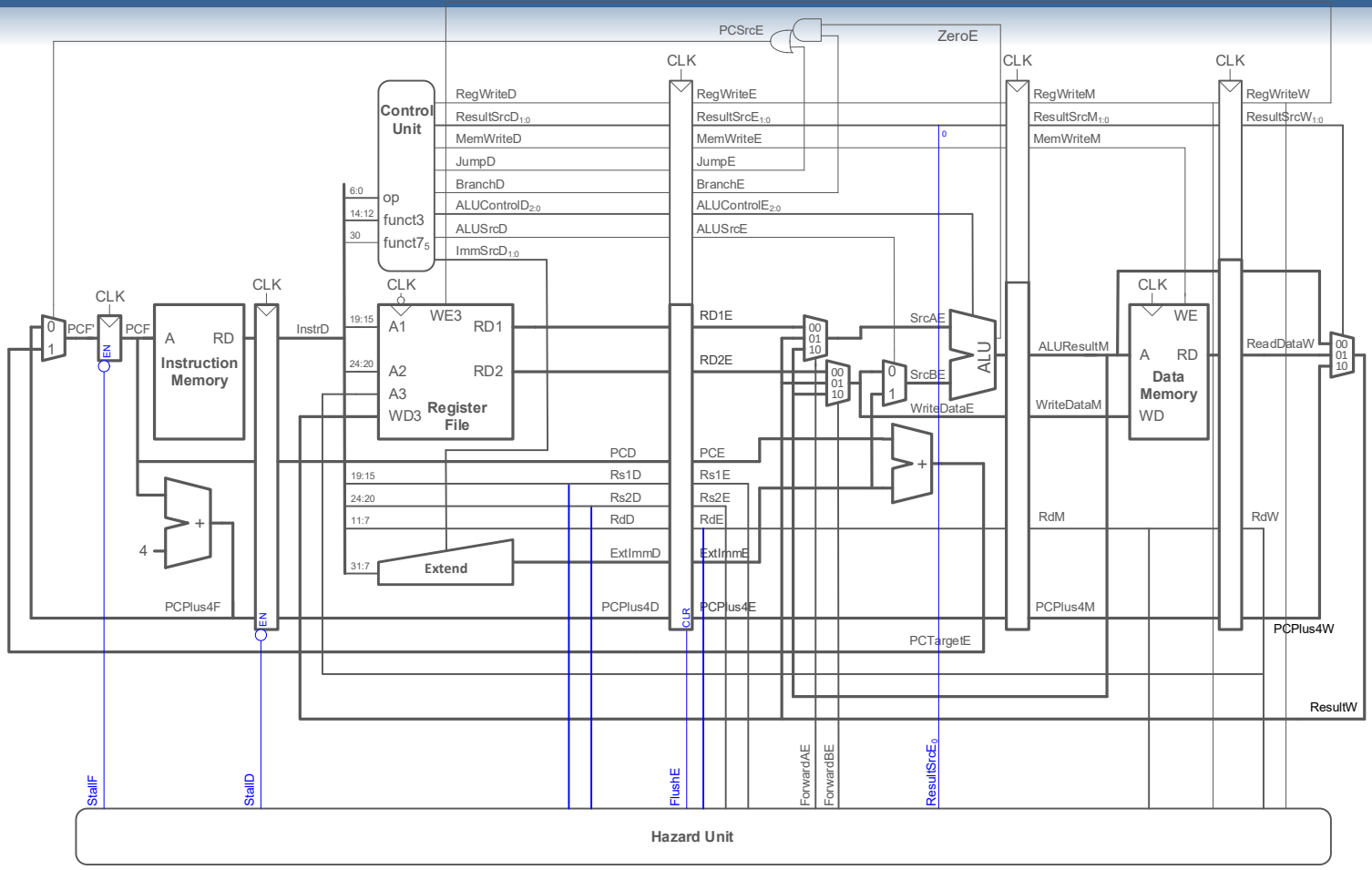
  ¿Es la instrucción en la etapa **Execute una `lw`**?

- Detener las etapas Fetch y Decode y anular la etapa Execute:
  - Para detener las etapas Fetch y Decode se descapacita la carga en los registros del pipeline (señal enable EN)
  - Para anular una etapa se ponen 0s en todos los puntos de control de la etapa (señal clear CLR). Se introduce una burbuja (bubble) en el pipeline.

  $$lwStall = ((Rs1D == RdE) \text{ OR } (Rs2D == RdE)) \text{ AND } ResultSrcE_0$$

  $$StallF = StallD = FlushE = lwStall$$

RIESGOS DE DATOS TRAS INSTRUCCIÓN LW



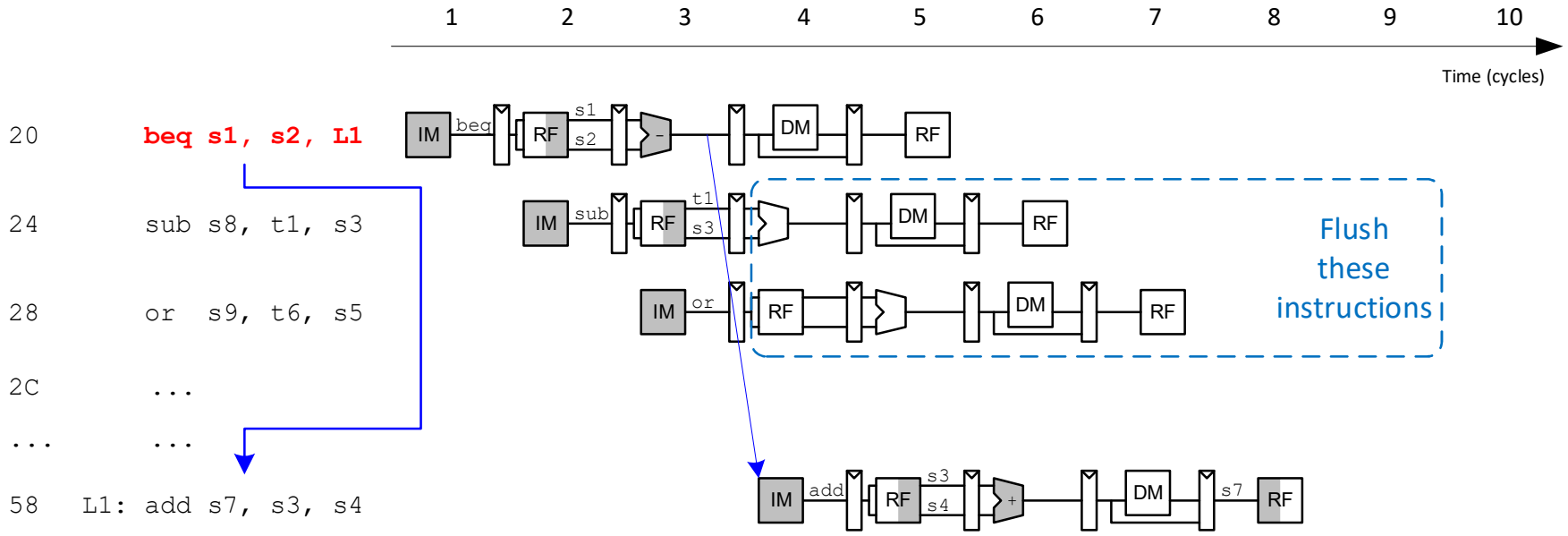
ÍNDICE

1. RISC-V
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. **Riesgos de control**
6. Rendimiento del procesador segmentado
7. Operaciones multiciclo
8. Rendimiento en los procesadores

RIESGOS DE CONTROL

- **beq:**
  - El salto no se determina hasta la etapa Execute.
  - Las dos Instrucciones siguientes al salto se buscan antes de que el salto se resuelva.
  - Estas dos instrucciones deben ser descartadas (flush) si el salto se produce.

- Penalización por mala predicción del salto:
  - El número de instrucciones buscadas cuando el salto se toma (en este caso 2)



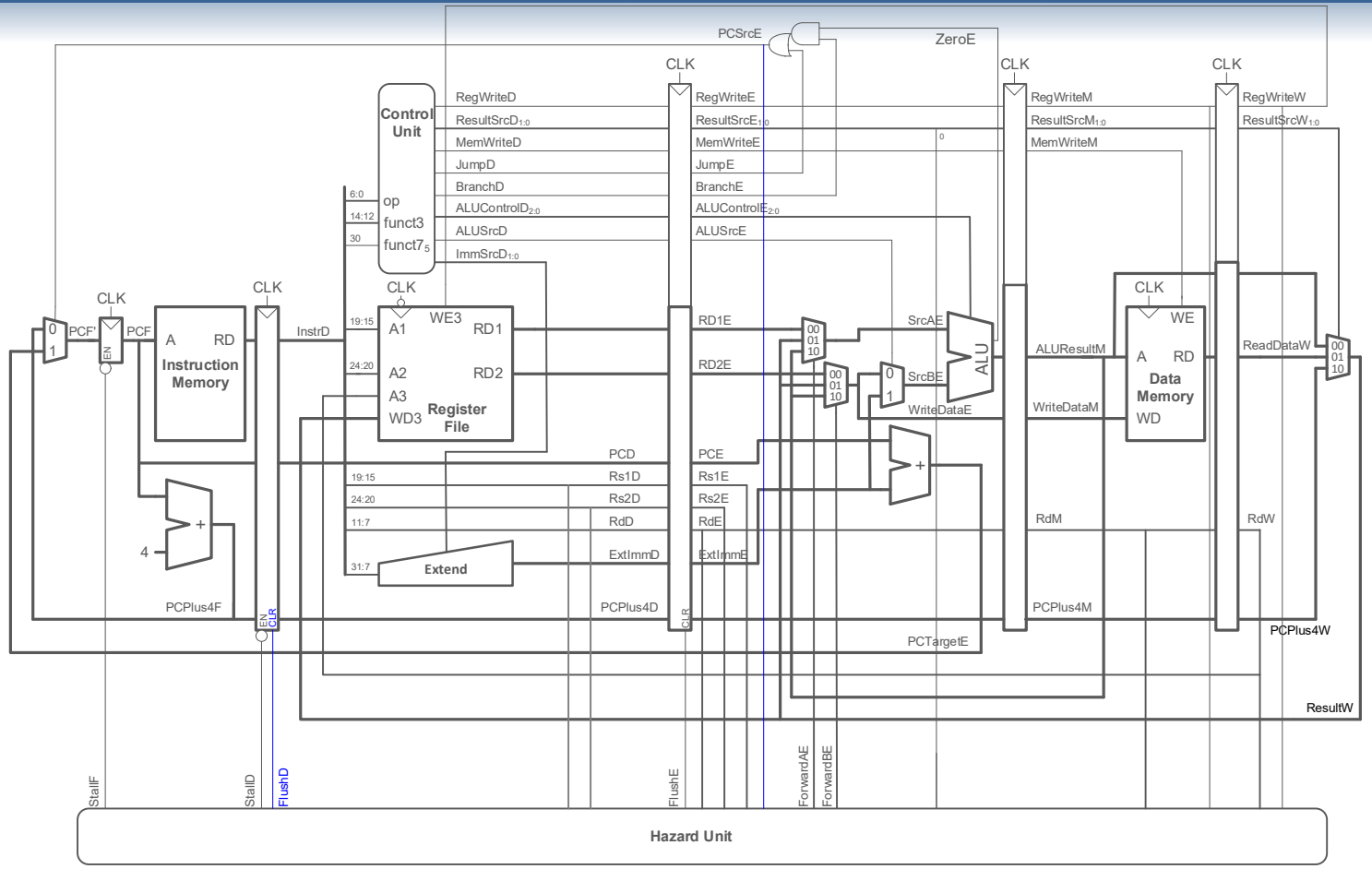
RIESGOS DE CONTROL

- Si en la etapa Execute se comprueba que el salto se toma, se necesita descartar las instrucciones que se encuentran en las etapas Fetch y Decode
  - Esto se hace poniendo a “0” los registros del pipeline Decode y Execute usando *FlushD* y *FlushE*
- Ecuaciones:

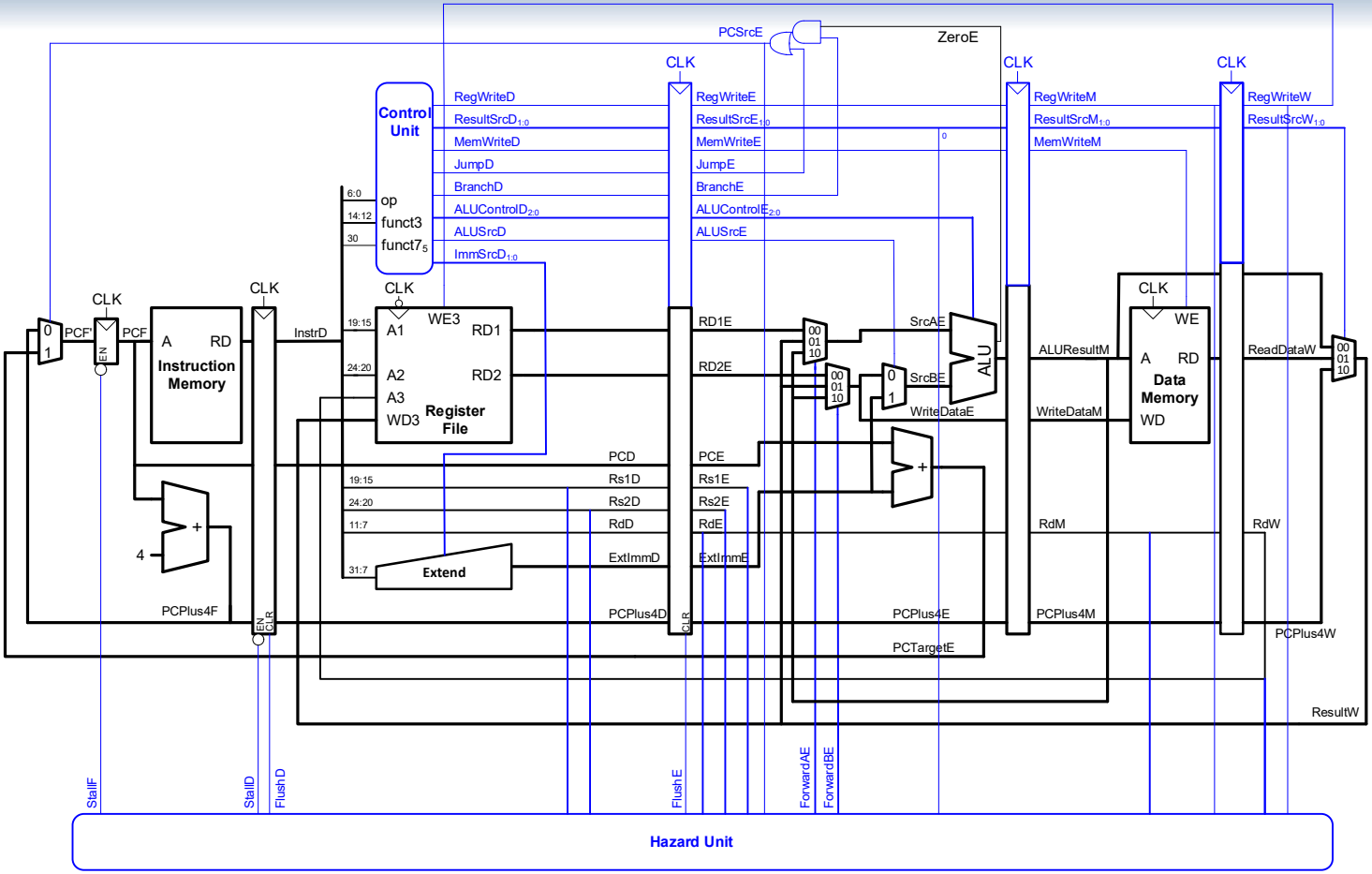
  FlushD = PCSrcE

  FlushE = lwStall OR *PCSrcE*

RIESGOS DE CONTROL



RISC-V: DETECCIÓN Y SOLUCIÓN DE RIESGOS



RESUMEN DE CONTROL DE RIESGOS

- Realimenta para resolver riesgos de datos cuando es posible:

if ((Rs1E == RdM) & RegWriteM) & (Rs1E != 0) then
ForwardAE = 10
else if ((Rs1E == RdW) & RegWriteW) & (Rs1E != 0) then
ForwardAE = 01
else
ForwardAE = 00

- Parada cuando ocurre un riesgo de datos tras un load:

$lwStall = ResultSrcE_0$ & ((Rs1D == RdE) | (Rs2D == RdE))

StallF = lwStall

StallD = lwStall

- Flush cuando un salto de toma o un load introduce una burbuja:

FlushD = PCSrcE

FlushE = lwStall | PCSrcE

ÍNDICE

RENDIMIENTO DEL PROCESADOR SEGMENTADO: CPI

- **SPECINT2000 benchmark:**
  - 25% loads
  - 10% stores
  - 11% saltos condicionales
  - 2 % saltos incondicionales
  - 52% tipo R
- Supongamos:
  - 40% de loads usados por la siguiente instrucción
  - 50% de saltos que se toman
- **¿Cuál es el CPI?** (Idealmente es 1)

load store Salto cond Salto incond Tipo R

CPI = (0,25*0,6*1) +(0,25*0.4*2) + (0.1*1) + (0.11*0,5*3)+ (0,11*0,5*1) + (0.02*3) + (0.52*1) =

= 1.25.

RENDIMIENTO DEL PROCESADOR SEGMENTADO: TC

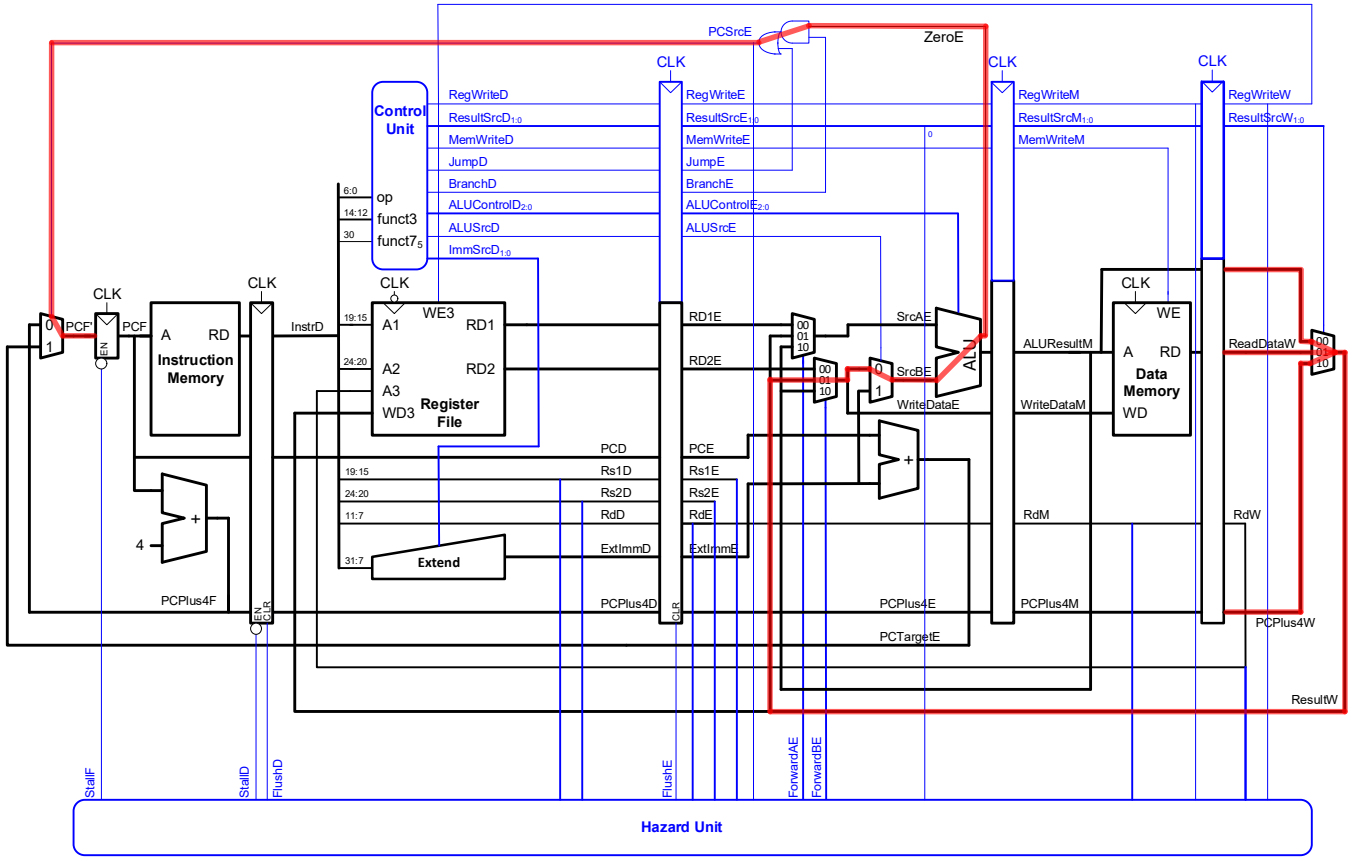
- Tiempo de ciclo.
  - Camino crítico del procesador segmentado:

$T_{c\_pipelined}$ = máximo de

$t_{pcq} + t_{mem} + t_{setup}$	**Fetch**
$2(t_{RFread} + t_{setup})$	**Decode**
$t_{pcq} + 4t_{mux} + t_{ALU} + t_{AND\text{-}OR} + t_{setup}$	**Execute**
$t_{pcq} + t_{mem} + t_{setup}$	**Memory**
$2(t_{pcq} + t_{mux} + t_{RFwrite})$	**Writeback**

Las etapas Decode y Writeback **usan el banco de registros** en cada ciclo por lo que cada una de ellas solo dispone de la mitad del tiempo para realizar su tarea.

CAMINO CRÍTICO: ETAPA EXECUTE



CAMINO CRÍTICO: ETAPA EXECUTE

Elemento	Parámetro	Retardo (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexor	t_{mux}	30
Puertas AND-OR	t_{AND-OR}	20
ALU	t_{ALU}	120
Decodificador	t_{dec}	25
Unidad de extensión	t_{ext}	35
Read memoria	t_{mem}	200
Read Banco de registros	t_{BRread}	100
Setup Banco de registros	$t_{BRsetup}$	60

$$T_{c\_pipelined} = t_{pcq} + 4t_{mux} + t_{ALU} + t_{AND-OR} + t_{setup} = 40 + 4(30) + 120 + 20 + 50 = 350 \text{ ps}$$

Esta es la etapa más lenta

RENDIMIENTO DEL PROCESADOR SEGMENTADO

Programa con 100.000 millones de instrucciones

Tiempo de ejecución = (# instrucciones) * CPI * T_c
= $(100 * 10^9)*(1.25)*(350 * 10^{-12})$
= **44 segundos**

Tiempo de ejecución monociclo = **80.5 segundos**
Tiempo de ejecución multiciclo = **155 segundos**

ÍNDICE

1. RISC-V
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Rendimiento del procesador segmentado
7. **Operaciones multiciclo**
8. Rendimiento en los procesadores

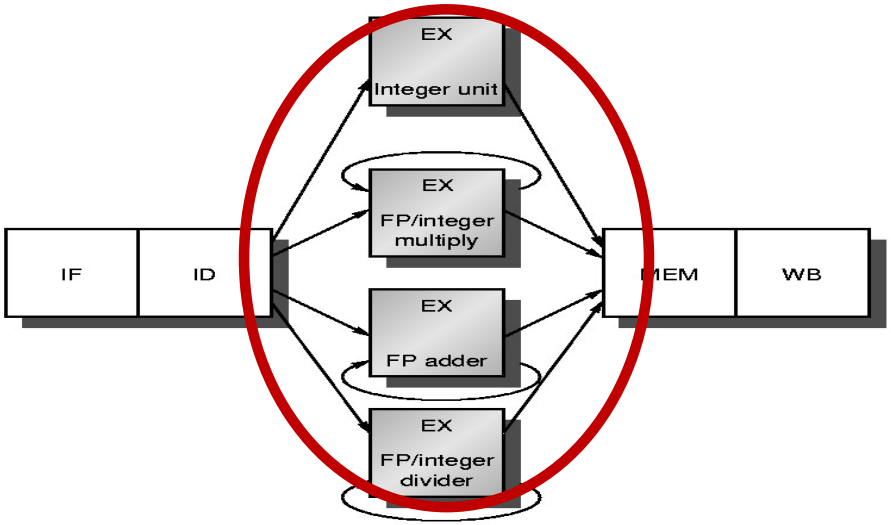
OPERACIONES MULTICICLO

- Cualquier procesador actual tiene en su repertorio operaciones sobre datos en Punto Flotante (PF):
  - Estas operaciones son más complejas y requieren más de un ciclo de ejecución
  - A veces, las instrucciones de multiplicación y división de enteros también requieren más ciclos

- **Latencia de las UFs**: Nº de ciclos que trascurren entre una instrucción que genera un dato y una instrucción que usa el resultado
- **Intervalo de iniciación**: Nº de ciclos que deben pasar entre dos ejecuciones de un mismo tipo de instrucción. Dependerá de si la UF es segmentada o no

Operaciones Multiciclo

Unidad funcional	Latencia	Intervalo de iniciación
ALU entera	0	1
Memoria de datos	1	1
Suma PF	3	1
Multiplicación PF	6	1
División PF	24	25 (no segmentada)



INSTRUCCIONES EN PUNTO FLOTANTE DEL RISC-V

- Trabajan sobre datos en un banco de registros diferente: f0 a f31

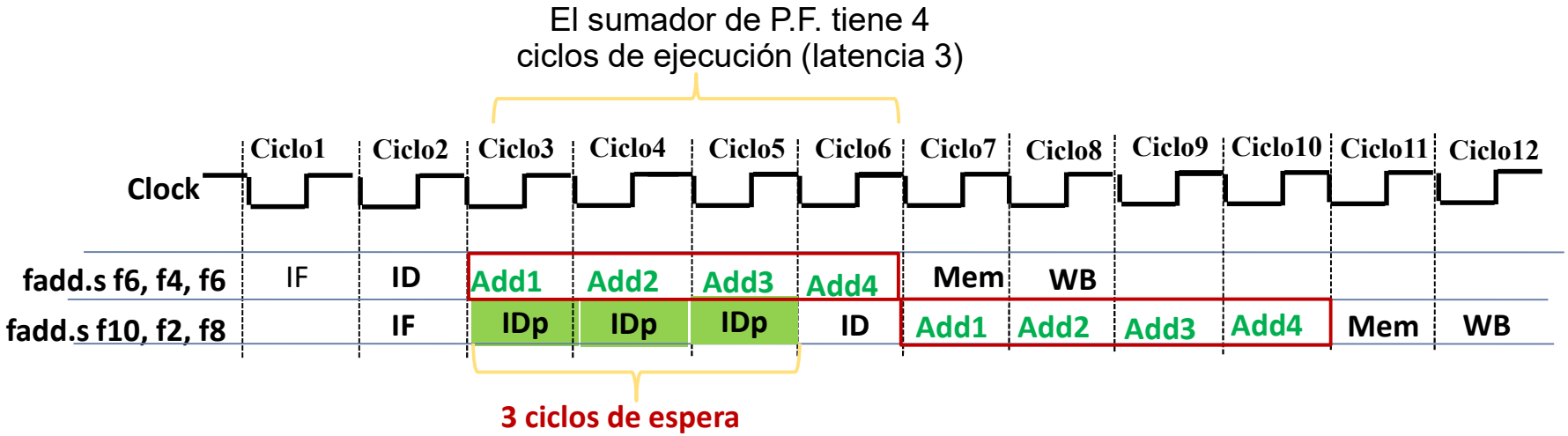
	Operación	Ejemplo	Significado
flw	load	flw f0, 4(x5)	f0 = Memoria[x5+4]
fsw	store	fsw f0, 8(x5)	Memoria[x5+8]= f0
fadd.s	Suma	fadd.s f0,f1, f2	f0 = f1 + f2
fsub.s	Resta	fsub.s f0, f1, f2	f0 = f1 - f2
fmul.s	Multiplicación	fmul.s f0, f1, f2	f0 = f1 * f2
fdiv.s	División	fdiv.s f0, f1, f2	f0 = f1 / f2
feq.s (*)	Compara si igual	feq.s x5, f1, f2	X5 = 1 si f1 == f2 X5 = 0 si f1 != f2
flt.s (*)	Compara si menor que	flt.s x5, f1, f2	X5 = 1 si f1 < f2 X5 = 0 si f1 >= f2
fle.s (*)	Compara si menor o igual	fle.s x5, f1, f2	X5 = 1 si f1 <= f2 X5 = 0 si f1 > f2

(*) Estas instrucciones permiten realizar saltos sobre comparaciones en PF usando las instrucciones enteras beq y bne

OPERACIONES MULTICICLO

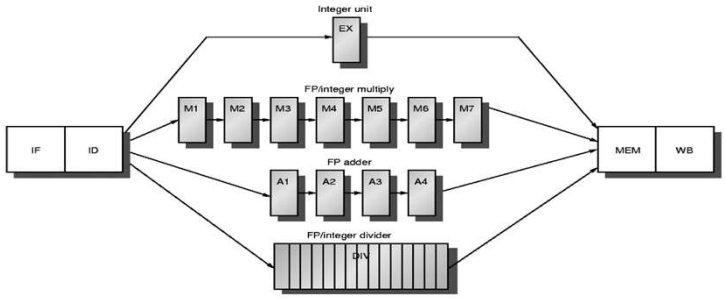
- Problemas:
  - Riesgos estructurales
  - **Mayor penalización** de los riesgos LDE
  - Aparecen riesgos EDE y EDL
  - Problemas con la **finalización fuera de orden**

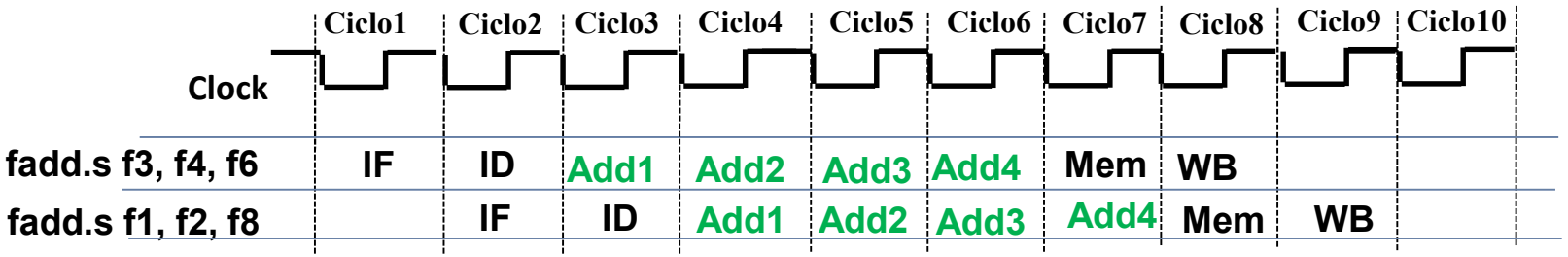
- **Riesgo estructural**: dos instrucciones necesitan la misma UF
  - Hay que esperar que la UF (no segmentada) haya acabado la ejecución de la primera operación



IDp ✓ID_P parada por riesgo estructural, se necesita el sumador y no está disponible

- **Riesgos estructurales. Solución:**
  - Segmentar las UFs con latencia > 1
  - La división no suele estar segmentada: se tiene que detectar el riesgo y realizar paradas
  - Sólo hay que esperar a que la UF haya acabado la operación asociada al primer ciclo de ejecución de la primera instrucción



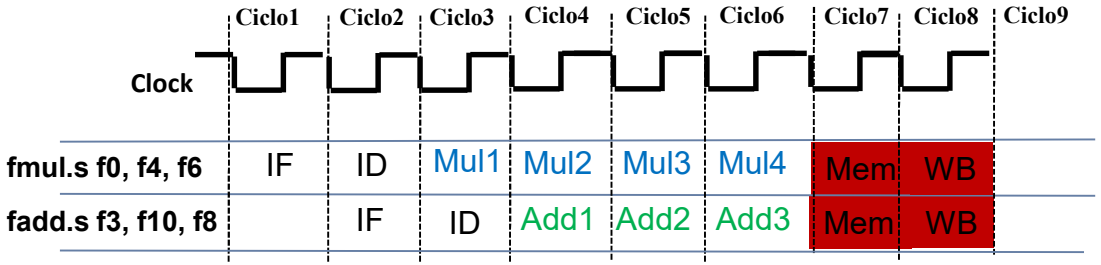


0 ciclos de espera

Riesgo estructural

- Dos instrucciones no pueden acceder a la vez a la etapa Mem
- Dos instrucciones no pueden acceder a la vez a la etapa WB

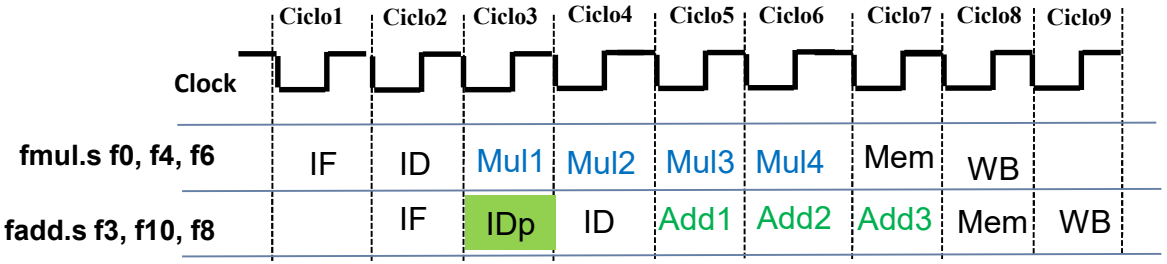
Unidad Funcional	Tiempo de ejecución
FP add	3
FP multiplicador	4



Solución:

- Detener la segunda instrucción en la etapa de decodificación

Unidad Funcional	Tiempo de ejecución
FP add	3
FP multiplicador	4

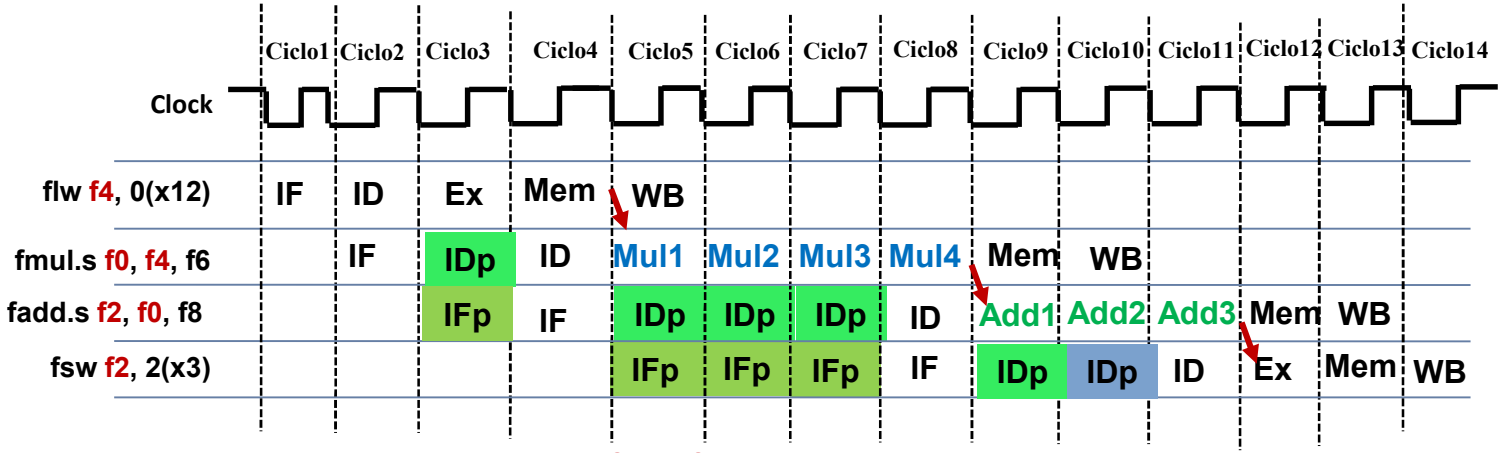


IDp ✓ID_P **parada por riesgo estructural, dos instrucciones van a acceder a la vez a la memoria de datos**

OPERACIONES MULTICICLO: RIESGOS LDE

Ejemplo:

Unidad Funcional	Tiempo de ejecución
FP add	3
FP multiplicador	4



6 ciclos de espera

IDp **parada por riesgo LDE**

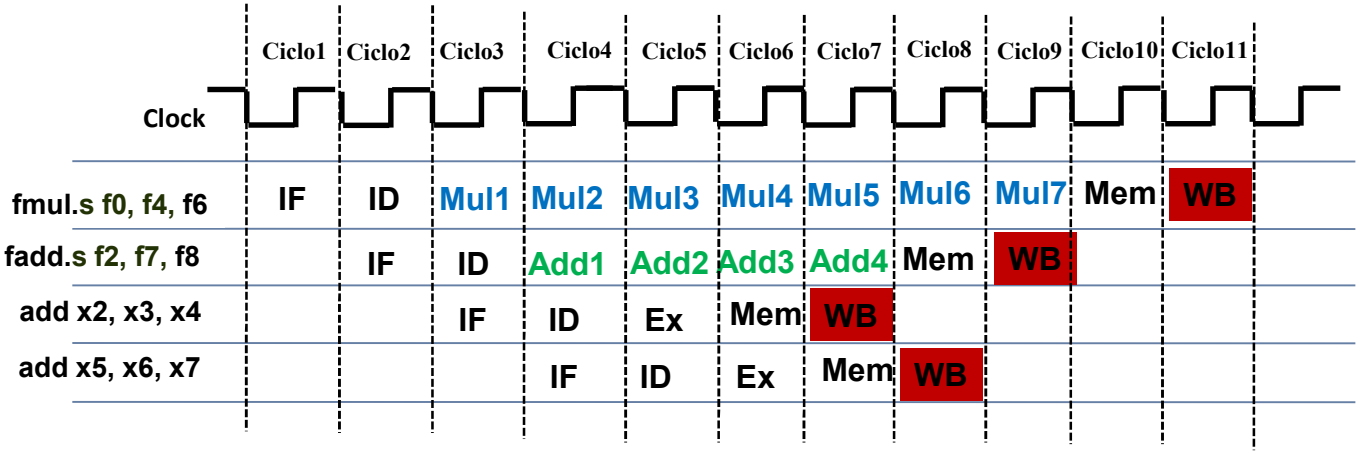
IFp **parada por riesgo estructural**, la etapa está ocupada por la instrucción anterior

IDp **parada por riesgo estructural**, acceso simultaneo a memoria

Finalización fuera de orden

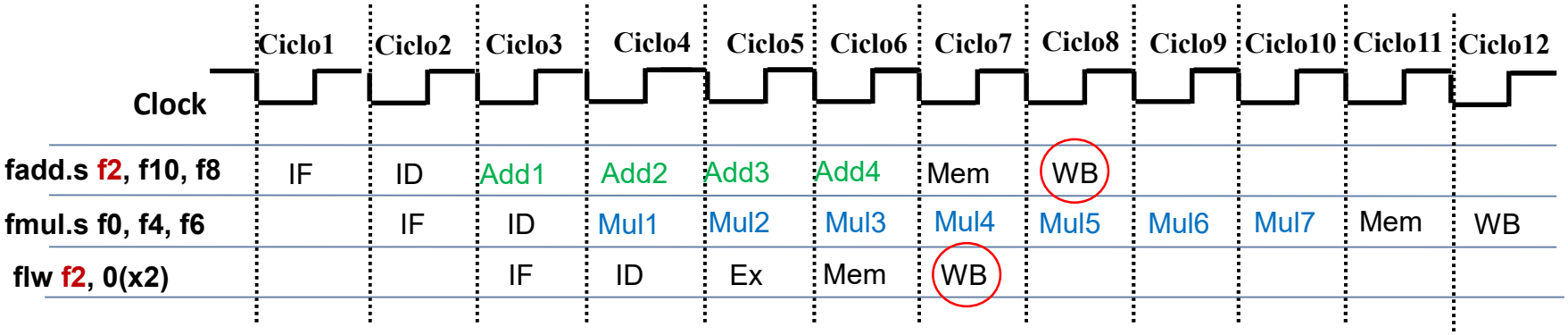
- Las instrucciones pueden acabar en un orden diferente al de lanzamiento
- **Problemas**:
  - Conflictos por escritura simultánea en el banco de registros (riesgo estructural)
  - Aparecen riesgos de escritura después de escritura (EDE)

Unidad Funcional	Tiempo de ejecución
FP add	4
FP multiplicador	7



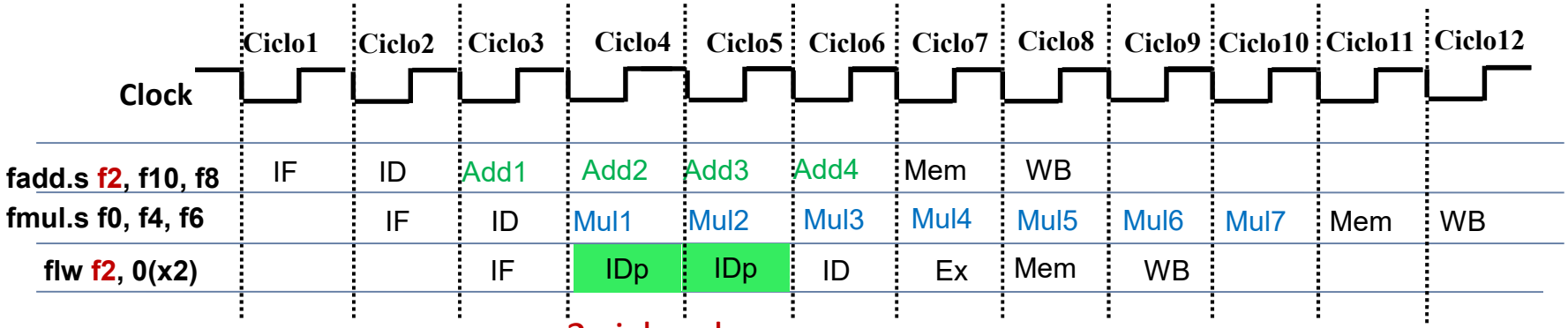
OPERACIONES MULTICICLO: RIESGOS EDE

- Problema: Escritura después de escritura



OPERACIONES MULTICICLO: RIESGOS EDE

- Solución 1
  - Detener en la etapa ID la instrucción que provoca el riesgo (la segunda)
  - El número de paradas depende de la longitud de la primera instrucción y de la distancia entre ambas

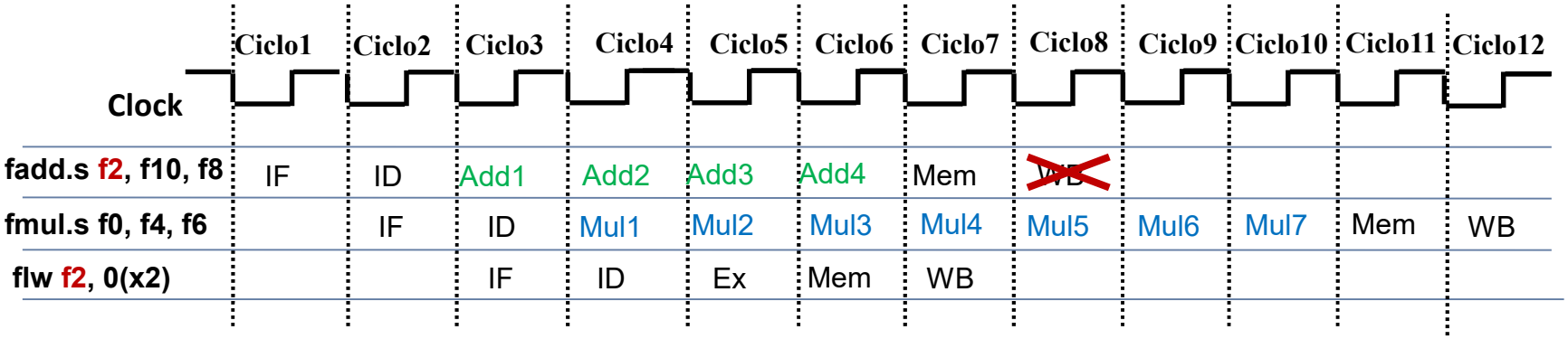


2 ciclos de espera

IDp ✓ID_P **parada por riesgo EDE** entre instrucción 1 e instrucción 2

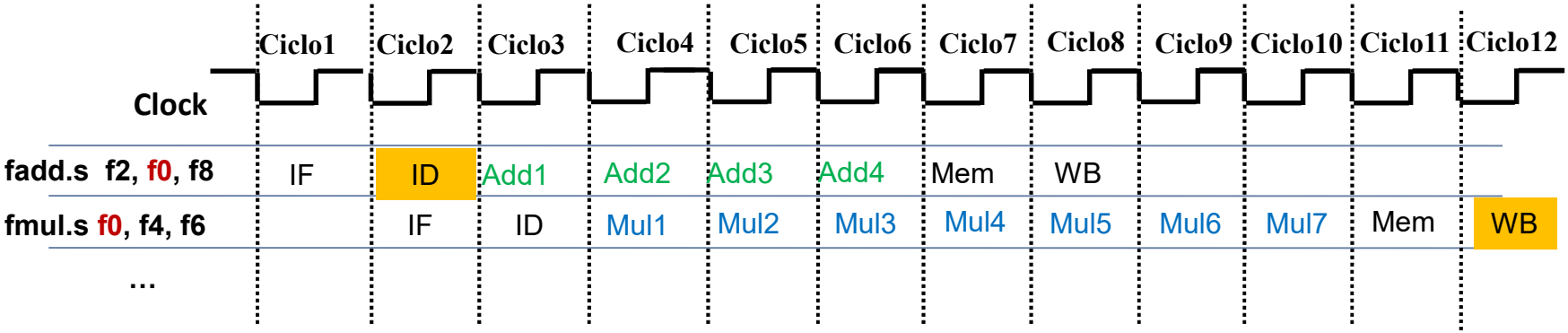
OPERACIONES MULTICICLO: RIESGOS EDE

- Solución 2
  - Inhibir la escritura de la primera instrucción



OPERACIONES MULTICICLO: RIESGOS EDL

- Estos riesgos NO se producen por construcción del pipeline
  - La lectura siempre se realiza en la etapa de decodificación
  - La escritura siempre se realiza en la última etapa



ÍNDICE

1. RISC-V
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Rendimiento del procesador segmentado
7. Operaciones multiciclo
8. **Rendimiento de los procesadores**

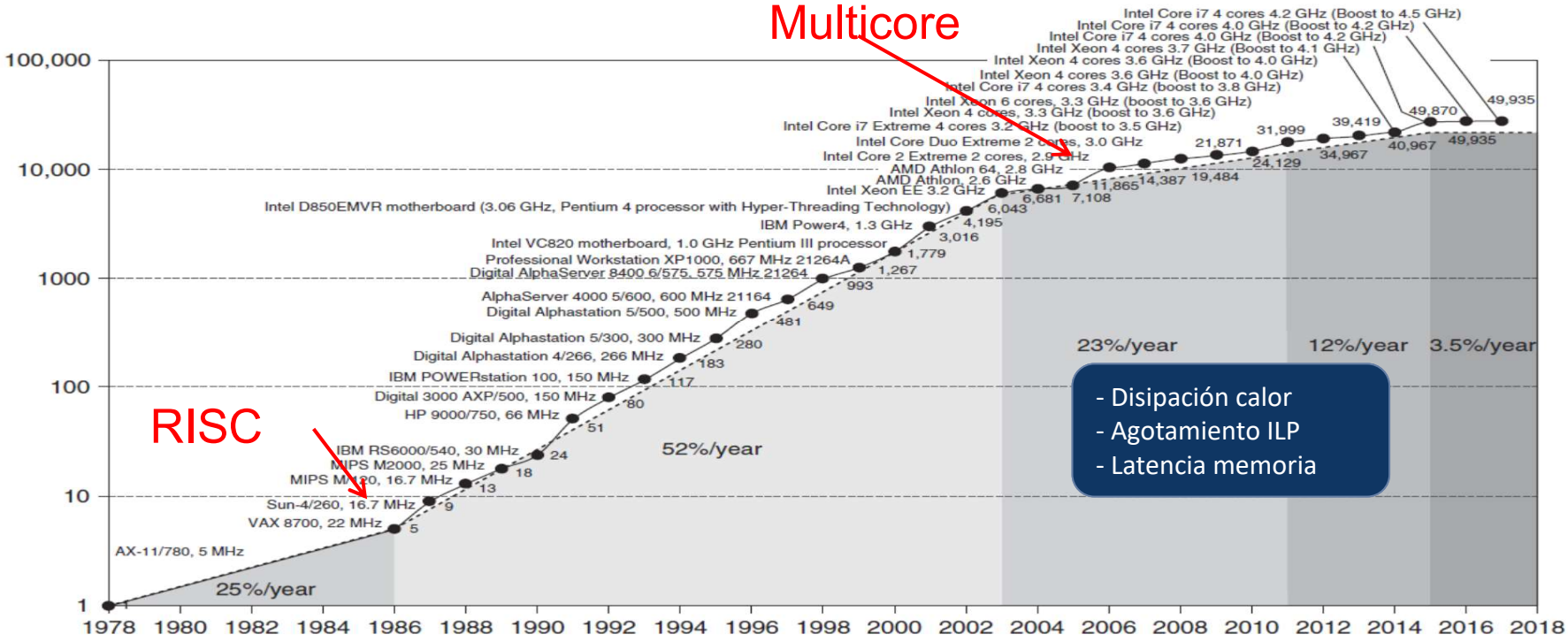
RENDIMIENTO

"Los buenos programadores se han preocupado siempre por el rendimiento de sus programas porque la rápida obtención de resultados es crucial para crear programas de éxito"

D. A. Patterson y J. L. Hennessy

CRECIMIENTO DEL RENDIMIENTO DE LOS PROCESADORES

Medida de rendimiento utilizada:
número de veces más rápido qué el VAX-11/780



RENDIMIENTO DE LOS PROCESADORES

- ¿Cuántos ciclos tarda en ejecutarse este programa?
  - Depende del procesador: por ejemplo en el RISC-V multiciclo

lw x1, 0(x0)	→ 5
lw x2, 4(x0)	→ 5
add x3, x1, x2	→ 4
beq x3, x5, 1	→ 3
sub x3, x3, x5	→ 4
sw x3, 8(x0)	→ 4

- ¿Y cuánto Tiempo?
  - Depende de la frecuencia del procesador

MEDIDAS DEL RENDIMIENTO

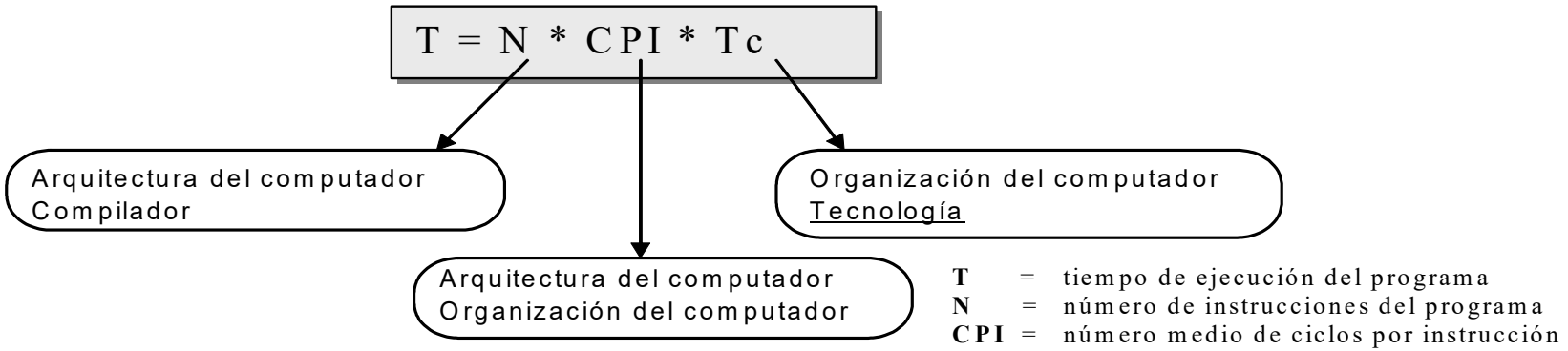
- Para poder comparar diferentes procesadores hace falta establecer una medida del rendimiento que permita cuantificar los resultados de la comparación
  - **Métrica**: establece la unidad de medida, que casi siempre es el tiempo, aunque hay que considerar dos aspectos diferentes del tiempo:
    - **Tiempo de ejecución:** tiempo que tarda en realizarse una tarea determinada
    - **Productividad** (throughput): tareas realizadas por unidad de tiempo
  - **Patrón de medida**: establece los programas que se utilizan para realizar la medida (benchmarks). Existen muchos posibles benchmarks aunque los más utilizados son:
    - Núcleos de programas reales: SPEC (www.spec.org)
    - Programas sintéticos

MEDIDAS DEL RENDIMIENTO: TIEMPO DE EJECUCIÓN

- **Tiempo de respuesta**: tiempo para completar una tarea (que percibe el usuario).
- **Tiempo de CPU**: tiempo que tarda en ejecutarse un programa, sin contar el tiempo de E/S o el tiempo utilizado para ejecutar otros programas. Se divide en:
  - ***Tiempo de CPU utilizado por el usuario:*** tiempo que la CPU utiliza para ejecutar el programa del usuario sin tener en cuenta el tiempo de espera debido a la E/S
  - ***Tiempo de CPU utilizado por el S.O:*** tiempo que el S.O. emplea para realizar su gestión interna.
- La función ***time*** de Unix produce: 90.7u 12.9s 2:39 65%, donde:
  - Tiempo de CPU del usuario = 90.7 segundos
  - Tiempo de CPU utilizado por el sistema = 12.9 segundos
  - Tiempo de CPU= 90.7 seg.+ 12.9seg = 103.6
  - Tiempo de respuesta = 2 minutos 39 segundos =159 segundos
  - Tiempo de CPU = 65% del tiempo de respuesta = 159 segundos*0.65 = 103.6
  - Tiempo esperando operaciones de E/S y/o el tiempo ejecutando otras tareas 35% del tiempo de respuesta = 159 segundos*0.35 = 55.6 segundos

MEDIDAS DEL RENDIMIENTO: TIEMPO DE EJECUCIÓN

- Tiempo de ejecución de un programa



- **CPI =** Ciclos medios por instrucción
  - Una instrucción necesita varios ciclos de reloj para su ejecución
  - Diferentes instrucciones tardan diferentes cantidades de tiempo
  - CPI = Es una suma ponderada del número de ciclos que tarda por separado cada tipo de instrucción

MEDIDAS DEL RENDIMIENTO: TIEMPO DE EJECUCIÓN

- Cálculo del CPI
  - El número total de ciclos de reloj de la CPU se calcula como:

$$CPI = \frac{\left(\sum_{i=1}^{n} CPI_i \bullet NI_i \right)}{NI} = \sum_{i=1}^{n} (CPI_i \bullet \frac{NI_i}{NI})$$

    - NI_i = número de veces que el grupo/tipo de instrucciones *i* es ejecutado en un programa
    - CPI_i = número medio de ciclos para el conjunto de instrucciones del grupo/tipo *i*

  - Podemos calcular el *CPI* multiplicando cada CPI_i individual por la fracción de ocurrencias de las instrucciones *i* en el programa.

SEGMENTACIÓN: PÉRDIDA DE RENDIMIENTO

- El CPI ideal es 1
- Hay pérdidas de rendimiento por las paradas del pipe

$CPI_{real} = CPI_{ideal}$ + Penalización media por instrucción =

$$= 1 + \sum_{i=1}^{\#tipos\ de\ instr} Penalizacion_i * Frec_i$$

- Caso de los saltos. Un programa típico 30% de saltos

  CPI = 1 + (2 x 0.3) = 1.6

$$Speep\_up = \frac{Nºinstrucciones * Nº\ etapas}{Nºinstrucciones * CPI} = \frac{5}{1{,}6} = 3{,}125$$

- Eficiencia:
  - $Speedup_{real} / Speedup_{ideal}$ =3.125 / 5 = 0.625

  Se pierde un 37.5 % respecto al caso ideal

MEDIDAS DEL RENDIMIENTO: MIPS

- **MIPS** (Millones de Instrucciones Por Segundo)

$$MIPS = \frac{NI}{Tiemp\ \ de\ ejecucion * 10^6} = \frac{1}{CPI * 10^6 * Tc}$$

- Dependen del repertorio de instrucciones, por lo que resulta un parámetro difícil de utilizar para comparar máquinas con diferente repertorio de instrucciones
- Varían entre programas ejecutados en el mismo computador
- Pueden variar inversamente al rendimiento, como ocurre en máquinas con hardware especial para punto flotante

MEDIDAS DEL RENDIMIENTO: MFLOPS

- **MFLOPS** (Millones de Operaciones en Punto FLotante Por Segundo)

$$MFLOPS = \frac{Numero\ de\ operaciones\ en\ punto\ flotante\ de\ un\ programa}{Tiempo\ de\ ejecucion\ *\ 10^6}$$

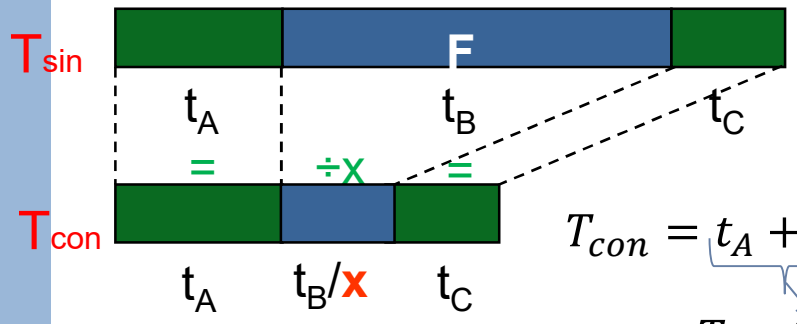
  - Existen operaciones en coma flotante rápidas (como la suma) o lentas (como la división), por lo que puede resultar una medida poco significativa.
  - Se han utilizado los MFLOPS normalizados, que dan distinto peso a las diferentes operaciones.
  - Por ejemplo: suma, resta, comparación y multiplicación se les da peso 1; división y raíz cuadrada peso 4; y exponenciación, trigonométricas, etc. peso 8:

MEDIDAS DEL RENDIMIENTO. LEY DE AMDAHL

- Ganancia de velocidad (*speedup*): Ley de Amdahl
  - *La mejora obtenida en el rendimiento global de un computador al utilizar un modo de ejecución más rápido está limitada por la fracción de tiempo que se puede utilizar dicho modo*
- Un principio básico: Hacer rápidas las funciones frecuentes.

$$\text{Speedup} = \frac{Tiempo\ de\ ejecucion\ sin\ mejora\ (Tsin)}{Tiempo\ de\ ejecucion\ con\ mejora\ (Tcon)}$$

- Qué speedup se obtendrá al aplicar una cierta mejora, M, que permite ejecutar una fracción, F, del código x veces más rápido.



$$T_{sin} = t_A + t_B + t_C \qquad F = \frac{t_B}{T_{sin}}$$

$$T_{con} = t_A + t_C + \frac{t_B}{x} = \quad \{\text{sustituyendo}\}$$

$$= T_{sin}\,(1 - F) + T_{sin}\frac{F}{x} = T_{sin}\left[(1 - F) + \frac{F}{x}\right]$$

MEDIDAS DE RENDIMIENTO: LEY DE AMDAHL

$$T_{con} = T_{sin}\left[(1-F) + \frac{F}{x}\right]$$

$$Speedup = \frac{T_{sin}}{T_{con}} = \frac{1}{(1-F) + \frac{F}{x}}$$

Ejemplo 1:. El 10% del tiempo de ejecución de mi programa es consumido por operaciones en PF. Se mejora la implementación de la operaciones PF reduciendo su tiempo a la mitad

$T_{con} = T_{sin}$ x (0.9 + 0.1 / **2**) = 0.95 x T_{sin}

$$Speedup = \frac{1}{0.95} = 1.053$$

Mejora de sólo un 5.3%

Ejemplo 2: Para mejorar la velocidad de una aplicación, se ejecuta una parte que consumía el 90% del tiempo sobre 100 procesadores en paralelo. El 10% restante no admite la ejecución en paralelo.

$T_{con} = T_{sin}$ x (0.1 + 0.9 / **100**) = 0.109 x T_{sin}

$$Speedup = \frac{1}{0.109} = 9.17$$

El uso de 100 procesadores sólo multiplica la velocidad por 9.17

Medidas de rendimiento: Ley de Amdahl

- Concepto de eficiencia (E)

$$\mathbf{E} = \frac{Speedup}{x} = \frac{\frac{1}{(1-F)+\frac{F}{x}}}{x} = \frac{1}{x(1-F)+F} = \frac{1}{x+F(1-x)}$$

El valor máximo posible de E es 1 (para lo que se necesitaría que F=1)

- Ampliación del Ejemplo 2:

Procesadores (x)	F	Speedup	Eficiencia
10	0.9	5.26	0,526 (52.6%)
100	0.9	9.17	0,0917 (9.17%)
1000	0.9	9.91	0.00991 (0.99%)

- Observaciones:

1. La fracción no paralelizable de un cálculo, (1-F), limita seriamente el Speedup, incluso cuando esta fracción es pequeña.
2. A partir de cierto punto, aumentar mucho el nº de procesadores apenas mejora el Speedup, por lo que se degradada mucho la Eficiencia.

MEDIDAS DE RENDIMIENTO: LEY DE AMDAHL

- ***Everyone knows Amdahl's Law but quickly forgets it!***

  Thomas Puzak (IBM's T. J. Watson Research Center)