ESTRUCTURA DE COMPUTADORES

Tema 1. Segmentación

Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

ÍNDICE

1. ARM
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Operaciones multiciclo
7. Rendimiento en los procesadores

- **Bibliografía**
  - Hennessy, J. L., Patterson, D. “Computer Architecture: A Quantitative Approach”, 5th ed., Morgan Kaufmann, 2012. ISBN 978-0-12-383872-8. Capitulo 3,Apéndice C
  - Paterson, D . Hennessy, J. L. “Computer Organization and Design” , 5th ed., Morgan Kaufmann, 2014, ISBN 0780124077263. Capítulo 4.
  - Harris, S., Harris, D. "Digital Design and Computer Architecture. ARM Edition”, Elsevier 2015

ÍNDICE

1. **ARM**
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Operaciones multiciclo
7. 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
    - LDR, STR: 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**
- **3 formatos de instrucción:**
  - Instrucciones de procesamiento de datos
  - Instrucciones de memoria
  - Instrucciones de salto

FORMATO DE LAS INSTRUCCIONES

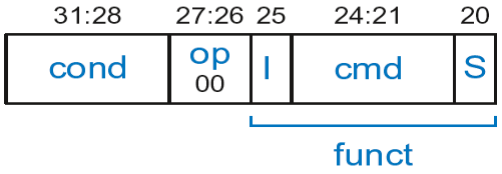
- **Instrucciones de procesamiento de datos**
  - Operandos:
    - Rn: primer registro fuente
    - Src2: segundo registro fuente o valor inmediato (ShiftOp)
    - Rd: registro destino
  - Campos de control:
    - Cond: especifica ejecución condicional
    - Op: código de operación: **00**
    - Funct: especifica la operación concreta a realizar

Data-processing

31:28	27:26	25:20	19:16	15:12	11:0
cond	op	funct	Rn	Rd	Src2
4 bits	2 bits	6 bits	4 bits	4 bits	12 bits

FORMATO DE LAS INSTRUCCIONES

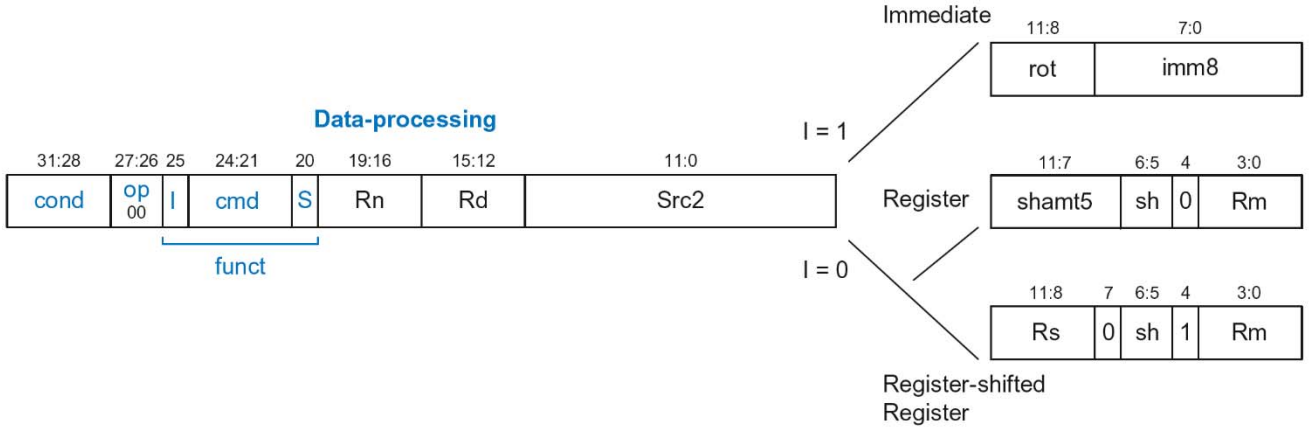
- **Instrucciones de procesamiento de datos**
  - ***funct*** está compuesto de ***cmd***, ***I***-bit, y ***S***-bit
    - ***cmd:*** especifica la instrucción de procesamiento de datos. Por ejemplo:
      - ***cmd*** = 0100_2 para ADD
      - ***cmd*** = 0010_2 para SUB



    - ***S-bit: 1 si activa los flags de condición***
      - ***S*** = 0: `SUB  R0, R5, R7`
      - ***S*** = 1: `ADDS R8, R2, R4` ó `CMP R3, #10`
    - ***I-bit***
      - ***I*** = 0: *Src2* es un registro
      - ***I*** = 1: *Src2* es un immediato

FORMATO DE LAS INSTRUCCIONES

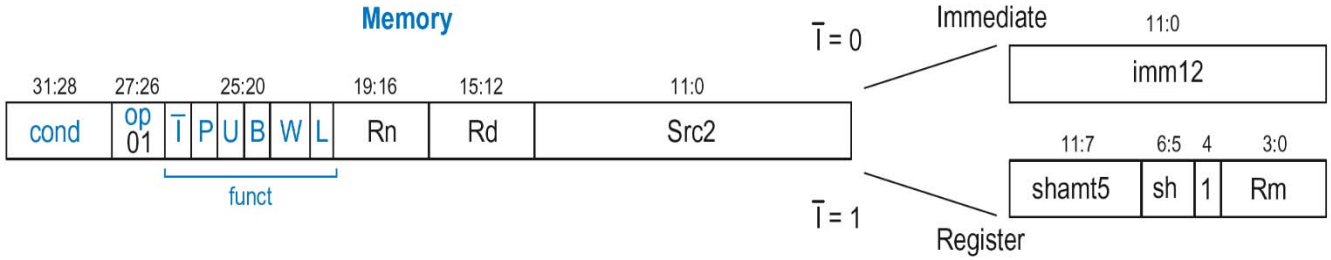
- **Instrucciones de procesamiento de datos**
  - ***Src2*** puede ser:
    - Inmediato
    - Registro
    - Registro con desplazamiento
      - ***sh*** indica el tipo de desplazamiento
      - ***shamt5***: Indica la cantidad de desplazamiento
      - ***rot*** : rotación a la derecha para crear constantes de 32 bits



FORMATO DE LAS INSTRUCCIONES

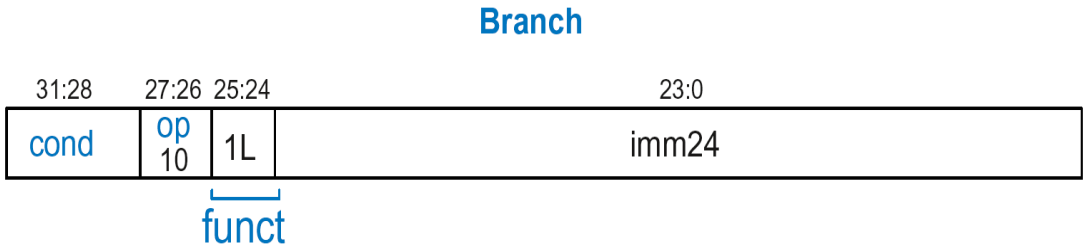
- **Instrucciones de carga y almacenamiento**
  - Codifican: LDR, STR, LDRB, STRB
    - ***op*** = 01_2
    - ***Rn*** = Registro base
    - ***Rd*** = destino (load), fuente (store)
    - ***Src2*** = desplazamiento
    - ***funct*** = 6 bits de control ***I*** (immediato o registro), ***U*** (suma o resta del desplazamiento), ***B*** y ***L*** (especifican el tipo de operación: LDR, STR, LDRB, STRB). ***P***, ***W*** indican algunas variaciones sobre el modo de direccionamiento que no hemos estudiado.

Tipo	L	B
STR	0	0
STRB	0	1
LDR	1	0
LDRB	1	1



FORMATO DE LAS INSTRUCCIONES

- **Instrucciones de salto: B y BL**
  - ***op*** = 10_2
  - ***imm24***: inmediato de 24 bits
  - ***func*** = Primer bit siempre 1
  - Bit ***L*** =1 para BL (branch and link) y ***L***=0 para B (branch)



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 de procesamiento de datos:**
    - **ADD, SUB, AND, ORR**
      - **Con Src2 en inmediato, pero sin desplazamiento**

31:28	27:26	25:20			19:16	15:12	rot	7:0
cond	op	1	cmd	S	Rn	Rd	0000	imm8

      - **Con Src2 en registro, pero sin desplazamiento**

31:28	27:26	25:20			19:16	15:12	11:7	6:5	4	3:0
cond	op	0	cmd	S	Rn	Rd	00000	00	0	Rm

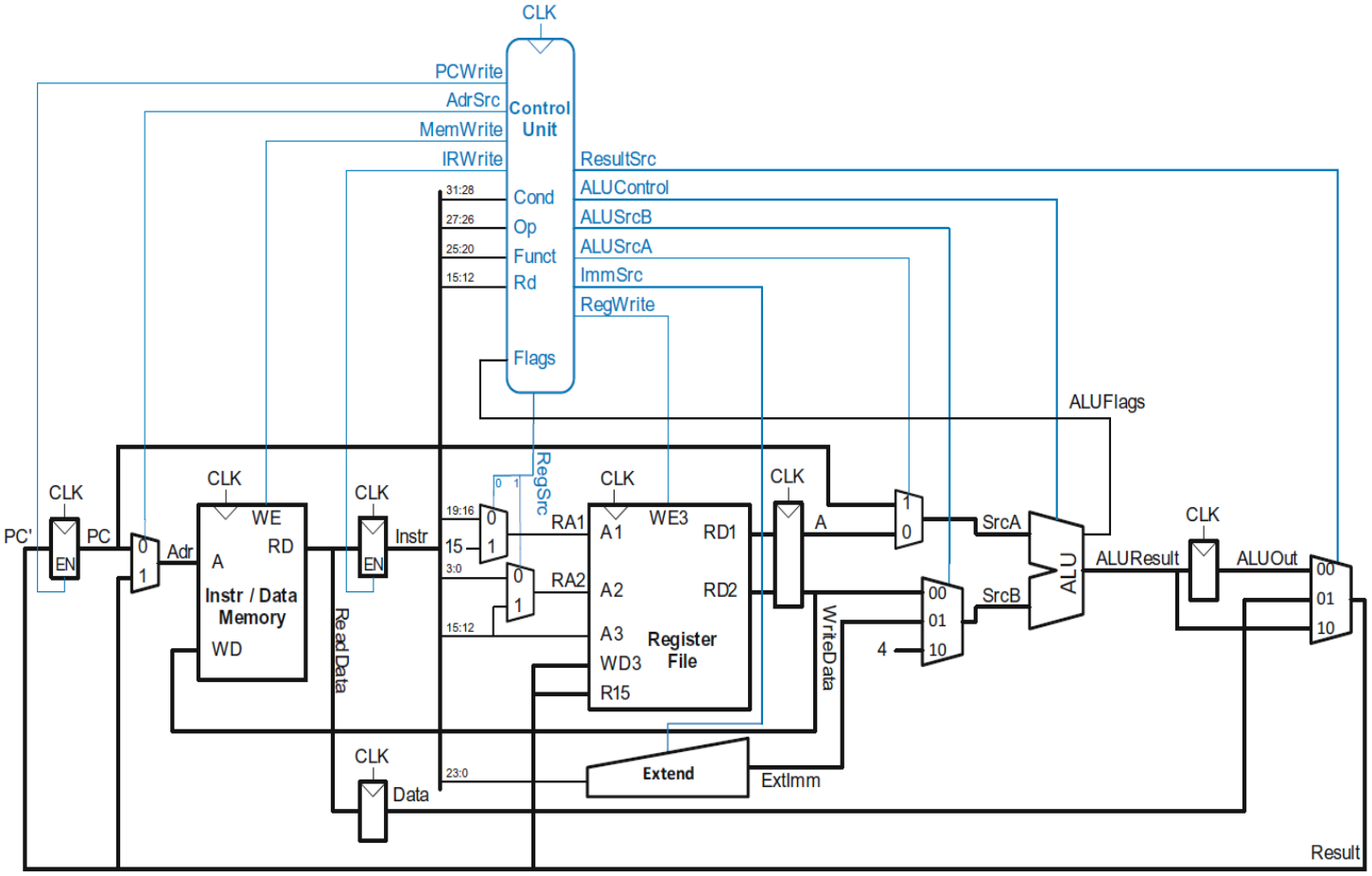
  - **Instrucciones de memoria:**
    - **LDR, STR**
      - **Con desplazamiento inmediato positivo**

31:28	27:26	25:20						19:16	15:12	11:0
cond	op	1	P	U	0	W	L	Rn	Rd	imm12

  - **Instrucción de salto:**
    - **B**

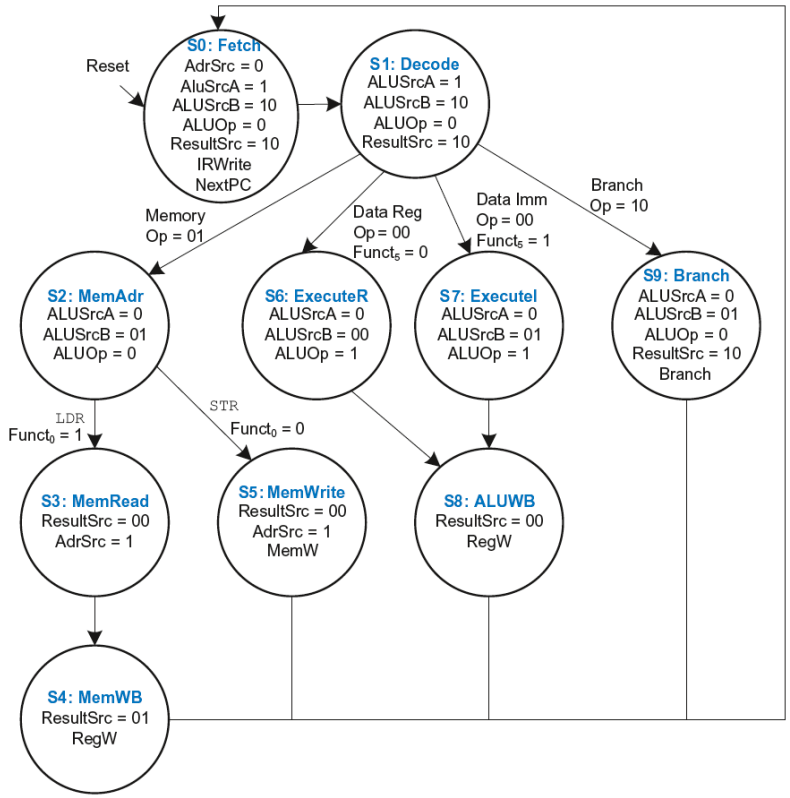
31:28	27:26	25:24	23:0
cond	op	1L	imm24

RUTA DE DATOS Y CONTROL MULTICICLO

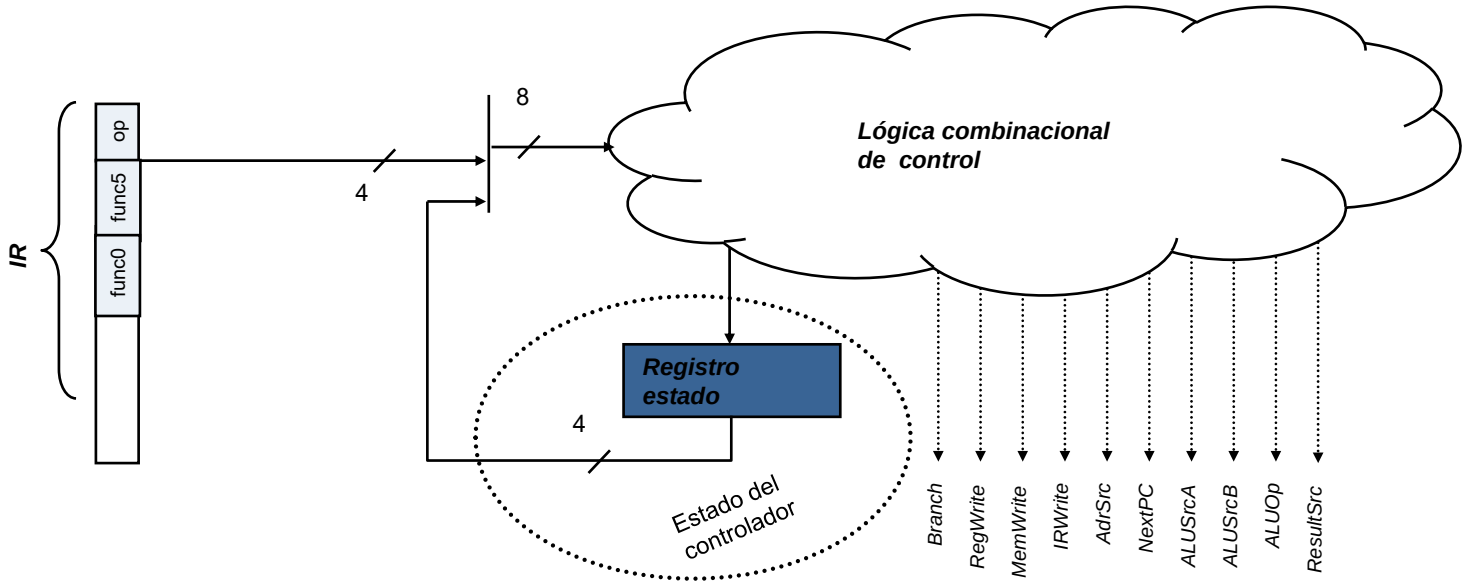


- Implementación mediante una ROM

State	Datapath µOp
Fetch	Instr ←Mem[PC]; PC ← PC+4
Decode	ALUOut ← PC +4
MemAdr	ALUOut ← Rn + Imm
MemRead	Data ← Mem[ALUOut]
MemWB	Rd ← Data
MemWrite	Mem[ALUOut] ← Rd
ExecuteR	ALUOut ← Rn op Rm
ExecuteI	ALUOut ← Rn op Imm
ALUWB	Rd ← ALUOut
Branch	PC ← R15 + offset



UNIDAD DE CONTROL MULTICICLO



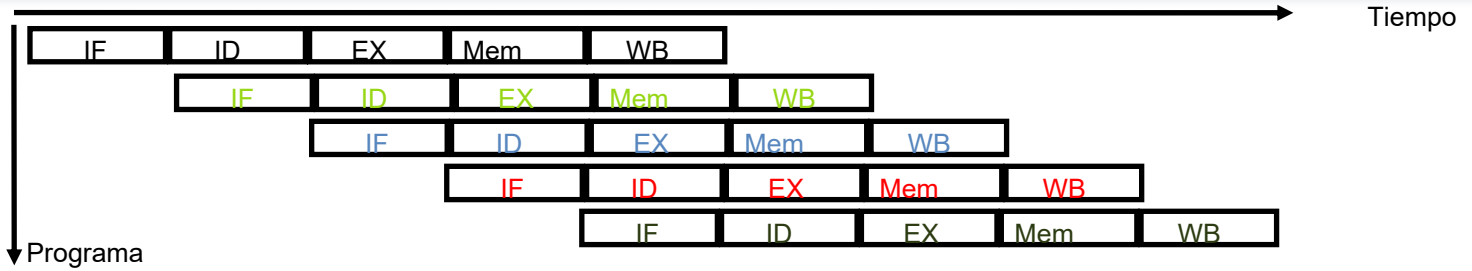
UNIDAD DE CONTROL MULTICICLO

Estado actual	op	Func0	Func5	Estado siguiente	Branch	RegWrite	MemWrite	IRWrite	NextPC	AdrSrc	ALUSrcA	ALUSrcB1	ALUSrcB0	ResultSrcB1	ResultSrcB0	ALUOp
0000	x	x	x	0001	0	0	0	1	1	0	1	1	0	1	0	0
0001	01	x	x	0010	0	0	0	0	0	X	1	1	0	1	0	0
0001	00	x	0	0110	0	0	0	0	0	X	1	1	0	1	0	0
0001	00	X	1	0111	0	0	0	0	0	X	1	1	0	1	0	0
0001	10	X	x	1001	0	0	0	0	0	X	1	1	0	1	0	0
0010	x	1	x	0011	0	0	0	0	0	X	0	0	1	X	X	0
0010	x	0	x	0101	0	0	0	0	0	X	0	0	1	X	X	0
0011	x	x	x	0100	0	0	0	0	0	**0**	X	X	X	**0**	**0**	X
0100	x	x	x	0000	0	**1**	0	0	0	X	X	X	X	**0**	**1**	X
0101	x	x	x	0000	0	0	**1**	0	0	**1**	X	X	X	**0**	**1**	X
0110	x	x	x	1000	0	0	0	0	0	X	0	0	0	X	X	1
0111	x	x	x	1000	0	0	0	0	0	X	0	0	1	X	X	1
1000	x	x	x	0000	0	1	0	0	0	X	X	X	X	0	0	X
1001	x	x	x	0000	1	0	0	0	0	X	0	0	1	1	0	0

ÍNDICE

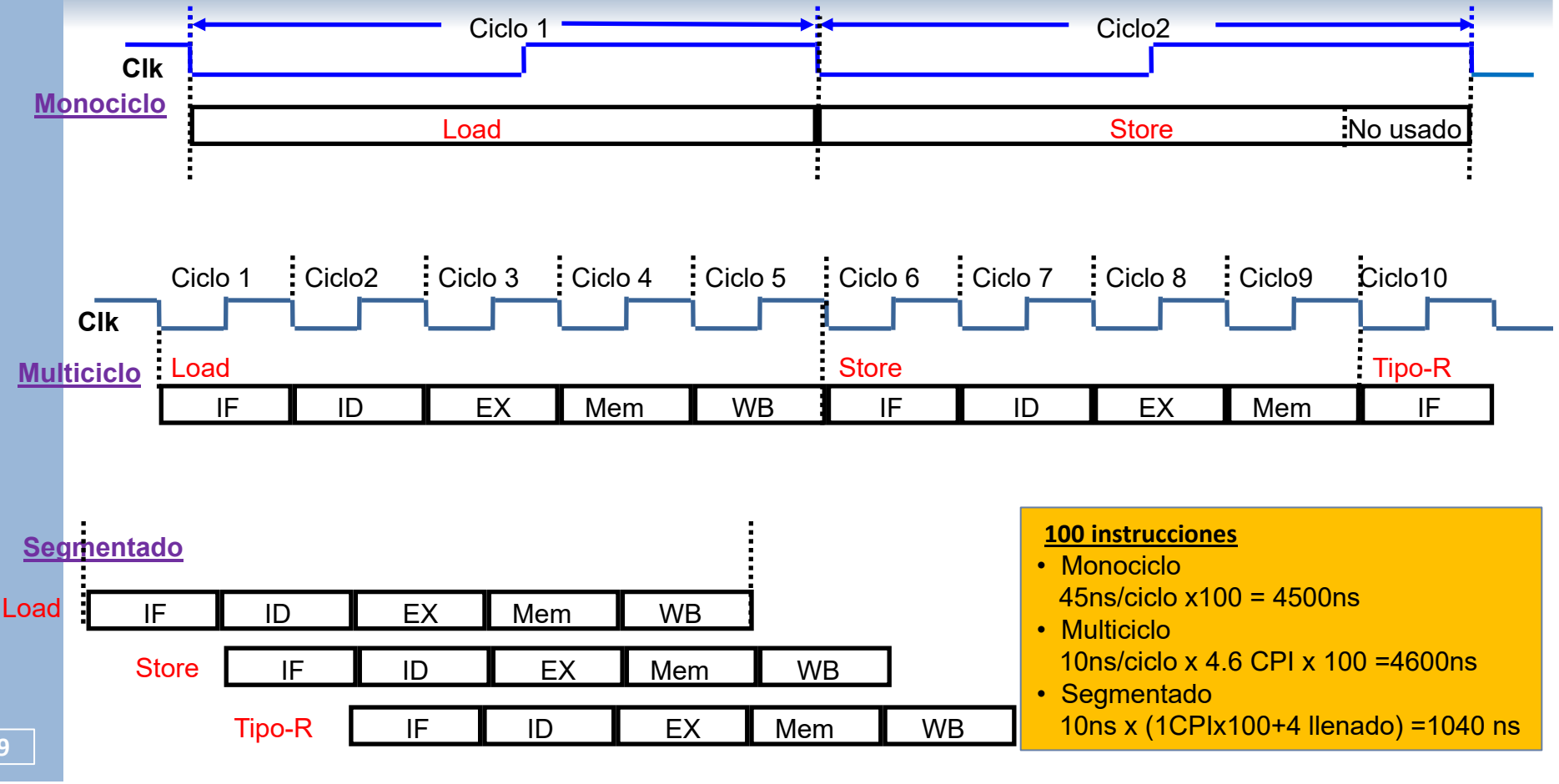
1. ARM
2. **Segmentación**
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Operaciones multiciclo
7. Rendimiento en los procesadores

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.
  - $CPI_{Ideal}=1$

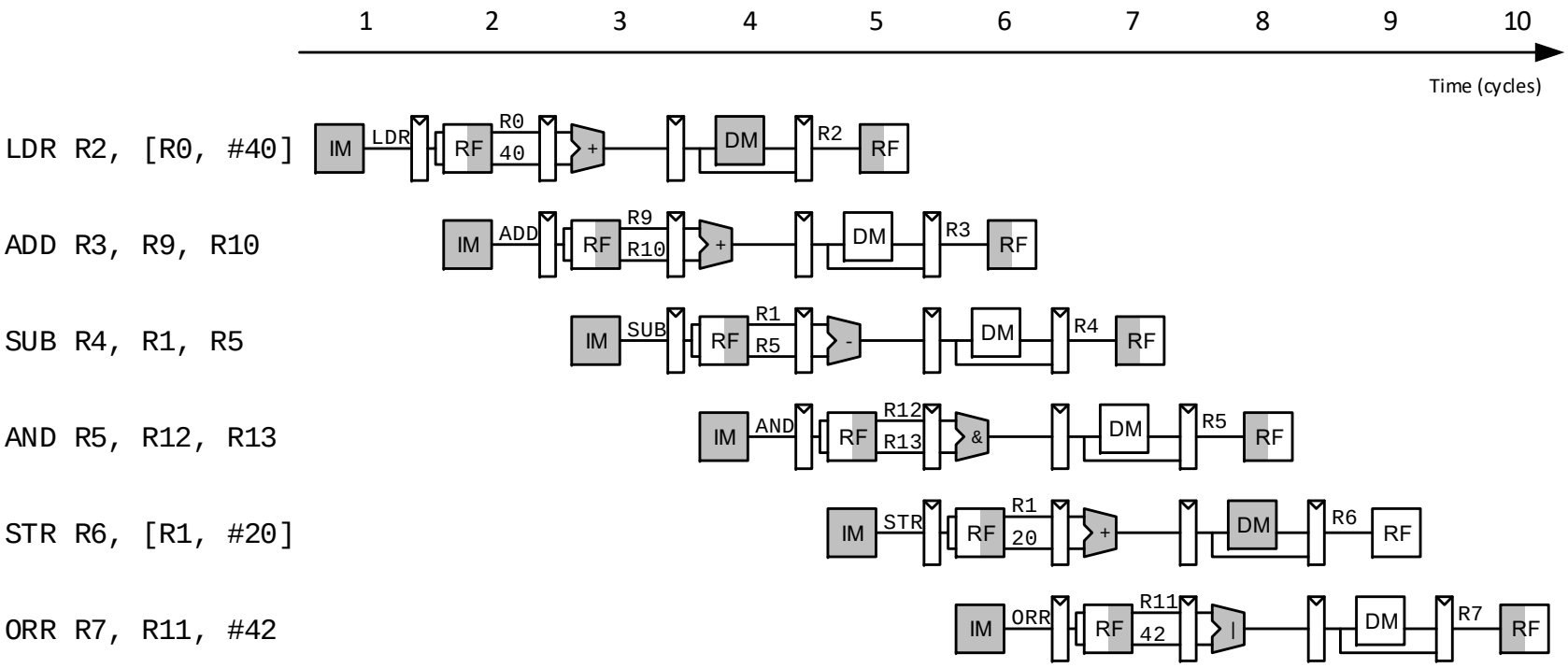
SEGMENTACIÓN



100 instrucciones

- Monociclo
  45ns/ciclo x100 = 4500ns
- Multiciclo
  10ns/ciclo x 4.6 CPI x 100 =4600ns
- Segmentado
  10ns x (1CPIx100+4 llenado) =1040 ns

- Representación gráfica del pipeline:



SEGMENTACIÓN

- ¿Qué facilita 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.

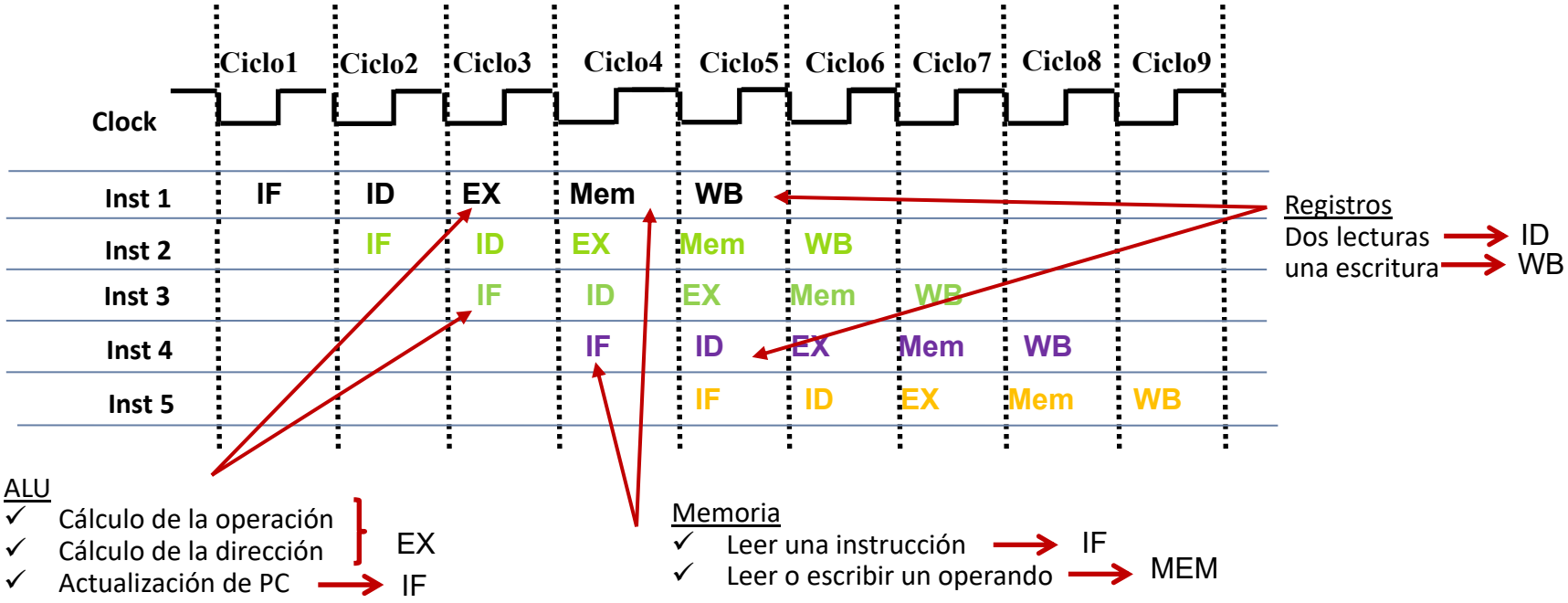
      Los riesgos se deben detectar y resolver
  - **Gestión de interrupciones**

ÍNDICE

1. ARM
2. Segmentación
3. **Riesgos estructurales**
4. Riesgos de datos
5. Riesgos de control
6. Operaciones multiciclo
7. 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**



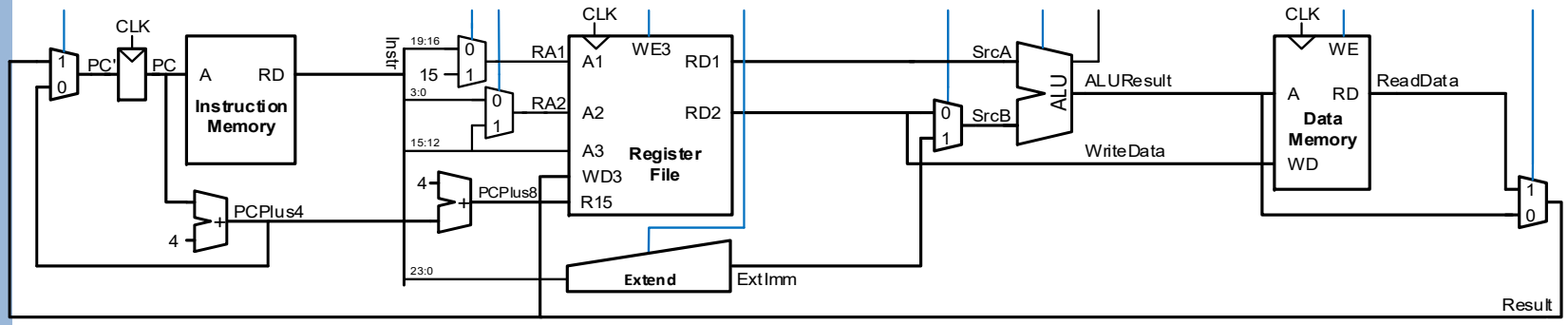
SEGMENTACIÓN: RIESGOS ESTRUCTURALES

- ¿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 B.R. no es problema porque tiene puertos para leer de dos registros y escribir en un registro a la vez.

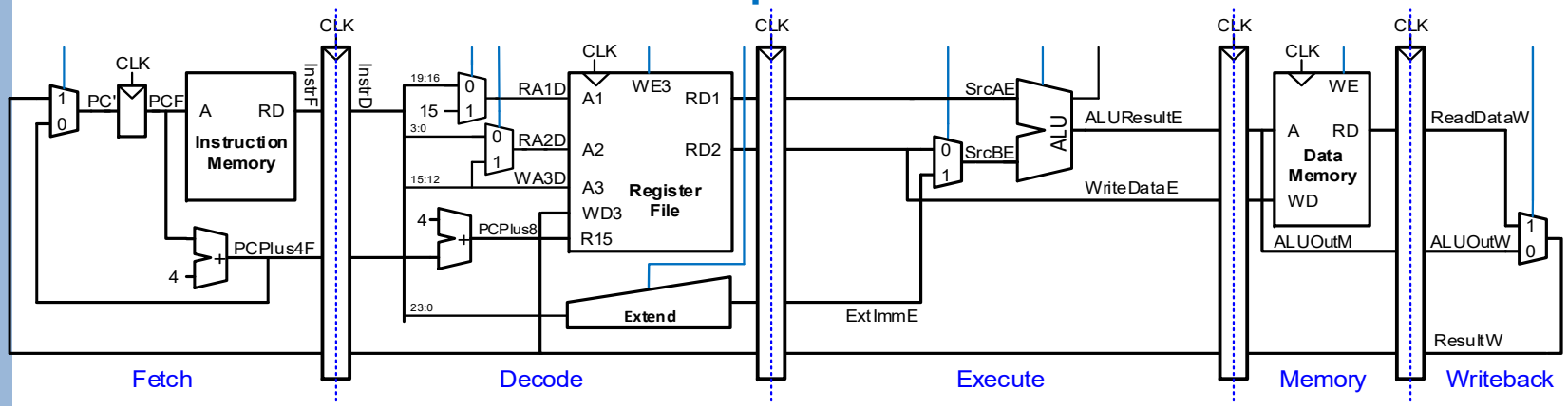
RUTA DE DATOS SEGMENTADA

Se añaden registros para separar las etapas

Single-Cycle

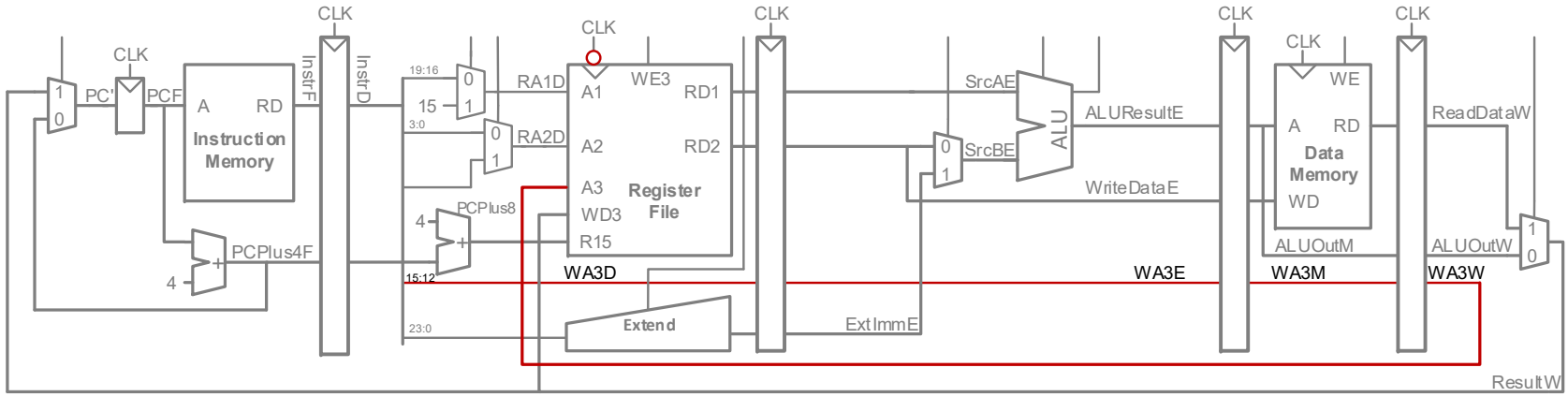


Pipelined



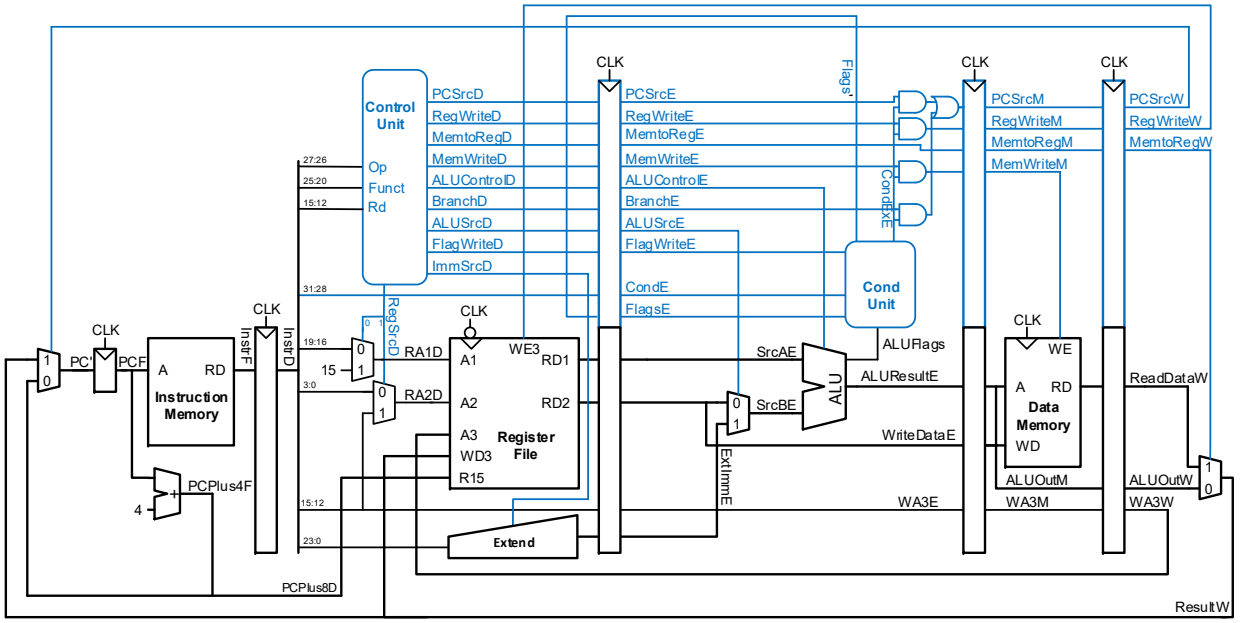
RUTA DE DATOS SEGMENTADA

- Ruta de datos corregida

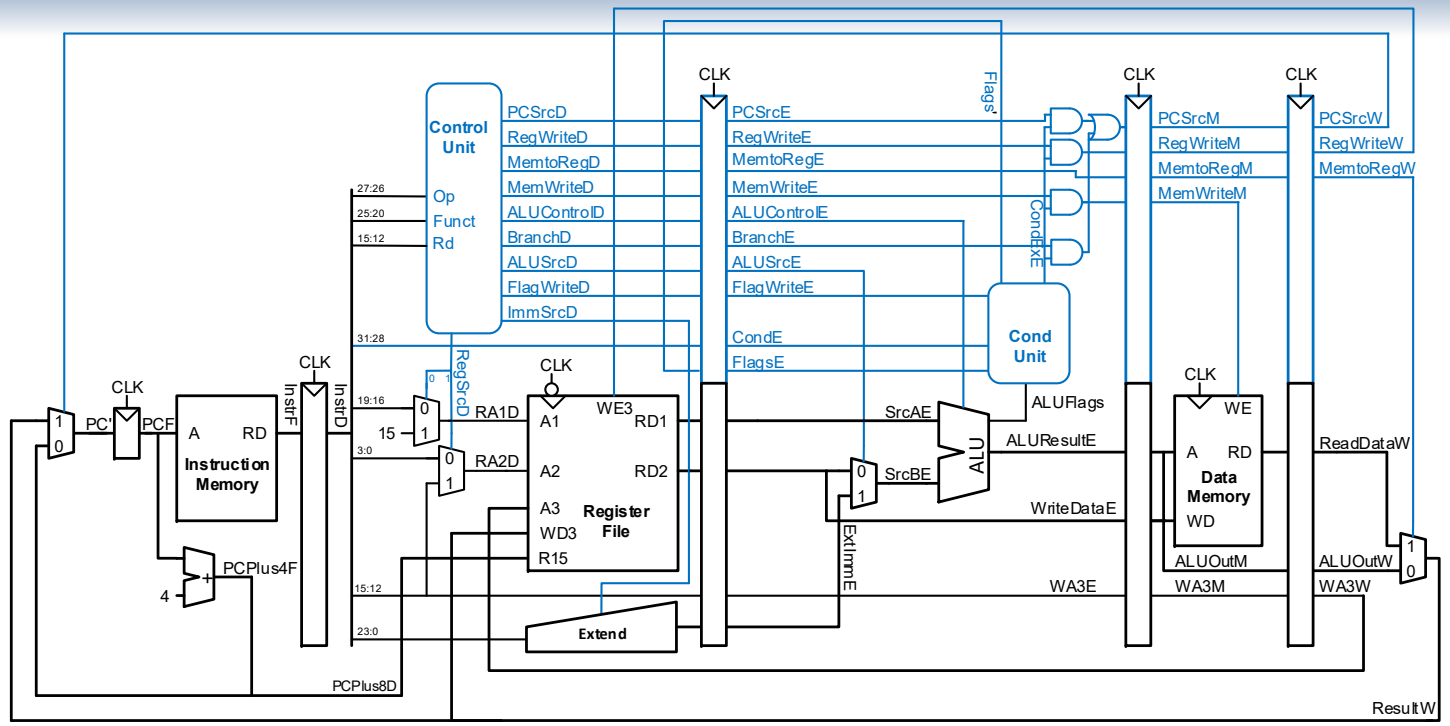


UNIDAD DE CONTROL SEGMENTADA

- La unidad de control es igual 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



ADD R3, R9, R11
LDR R5, [R2, #40]
SUB R5, R5, R4
OR R6, R7, R8

ÍNDICE

1. MIPS
2. Segmentación
3. Riesgos estructurales
4. **Riesgos de datos**
5. Riesgos de control
6. Operaciones multiciclo
7. Rendimiento en los procesadores

SEGMENTACIÓN : RIESGOS DE DATOS

- **Riesgos de datos:**
  - Se produce un riesgo 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)**

SEGMENTACIÓN: RIESGOS DE DATOS

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

  *ADD **R1**,R2,R3 – escribe el registro R1*

  *ADD R4,**R1**,R2—lee el registro R1*

  - Se produce riesgo si R1 se lee antes de que lo escriba la primera instrucción

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

  *ADD R1,**R4**,R3 – lee el registro R4*

  *ADD **R4**,R5,R2—escribe el registro R4*

  - Se produce riesgo si R4 se escribe antes de que lo lea la primera instrucción

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

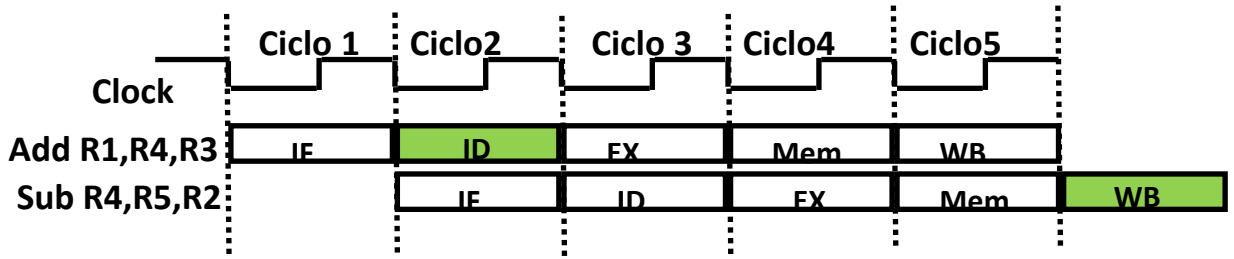
  *ADD **R4**,R2,R3 – escribe el registro R4*

  *Add **R4**,R1,R2—escribe el registro R4*

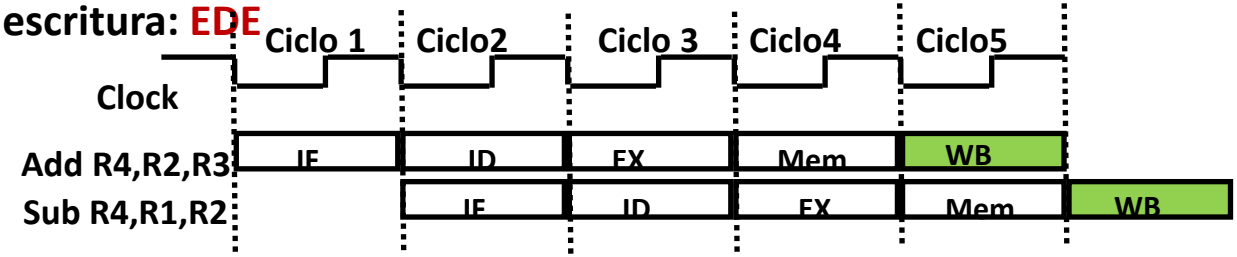
  - Se produce riesgo si R4 de la segunda instrucción se escribe antes de que lo escriba la primera instrucción

RIESGOS DE DATOS: EDL Y EDE

- **Escritura después de lectura: EDL**

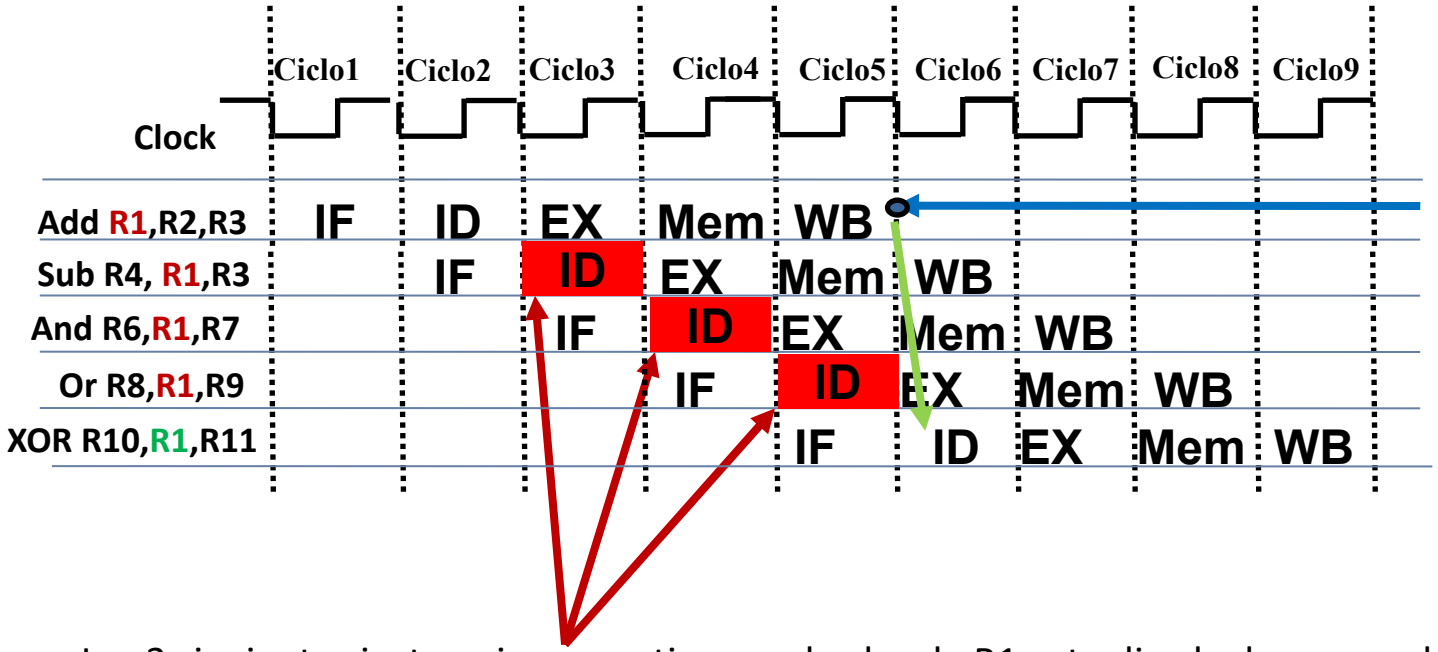


- **Escritura después de escritura: EDE**



- **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 BR en la última etapa

RIESGOS DE DATOS: LDE



R1 se escribe al empezar el ciclo6

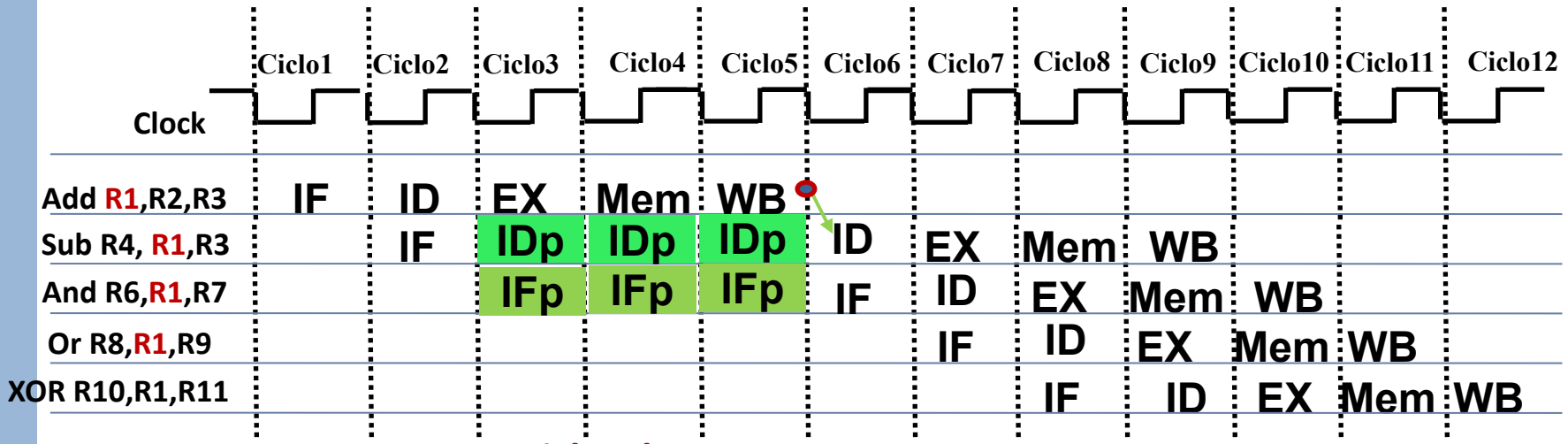
Las 3 siguientes instrucciones no tienen el valor de R1 actualizado, leen un valor antiguo

¿Cómo resolver los riesgos LDE?

- **Solución 1: Detener el pipeline hasta que los datos estén disponibles**
  - ¿En qué etapa se realiza la parada?
    - Depende del diseño de la ruta de datos
    - Nosotros vamos a suponer que **las paradas se realizan en 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
- **Solución 2: Reordenar código**
  - Si se puede, no siempre es posible
  - Puede minimizar las paradas

¿Cómo resolver los riesgos LDE?

- **Solución 1: Detener el pipeline** (en la etapa de decodificación)



3 ciclos de espera

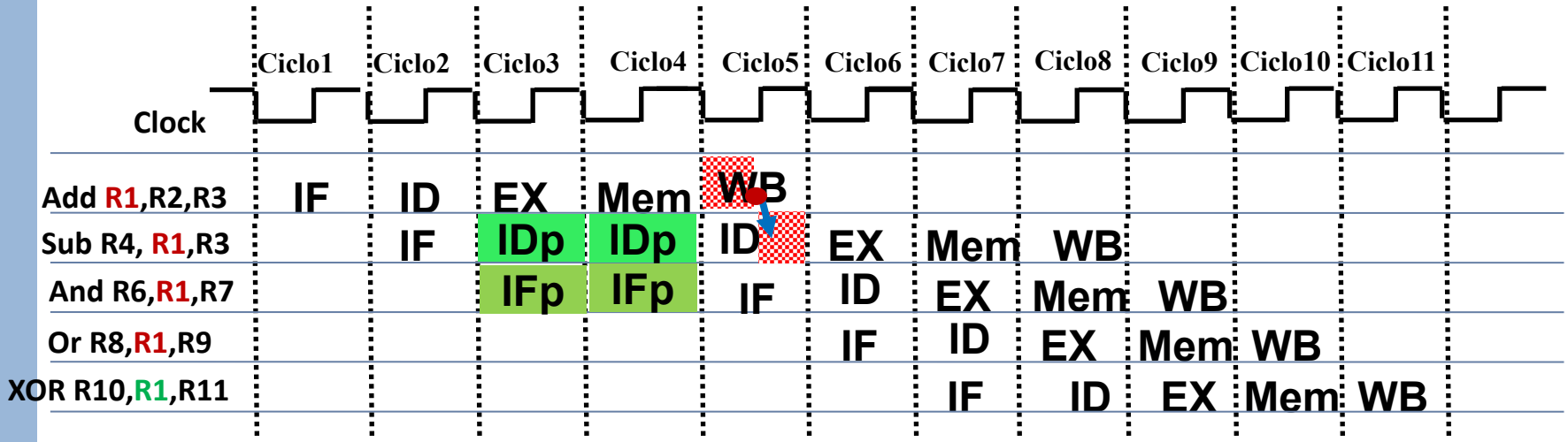
Sólo se está ejecutando la primera instrucción

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?

- **Mejora**: el **Banco de Registros escribe en la primera mitad del ciclo y lee en la segunda mitad del ciclo**



2 ciclos de espera

Sólo se está ejecutando la primera instrucción

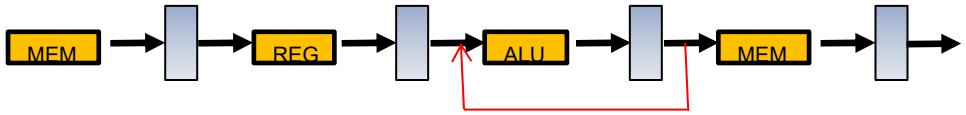
IDp ID_P **parada por riesgo LDE** entre instrucción 1 e instrucción 2

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

¿Cómo resolver los riesgos LDE? Solución 3: Cortocircuito (forwarding)

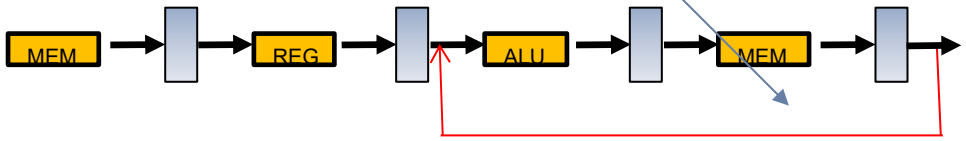
- *¿ Cuándo está listo el operando ?* Enviar el dato cuando esté calculado a las etapas que lo necesiten sin esperar a WB
  - Situación:

    add **r1**,r2,r3
    sub r4,**r1**,r3
    and r6,r5,r7

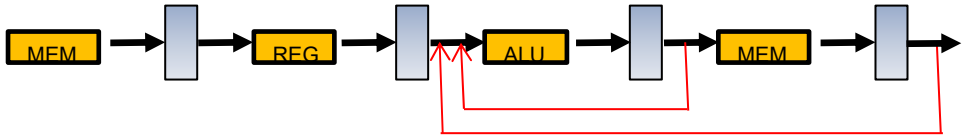
  - Situación:

    add **r1**,r2,r3
    sub r4,r5,r3
    and r6,**r1**,r7

  - Situación:

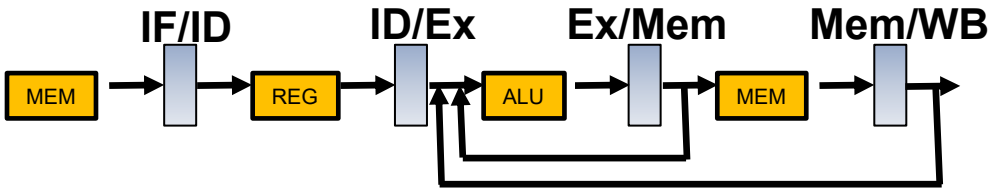
    add **r1**,r2,r3
    sub r4,**r1**,r3
    and r6,**r1**,r7

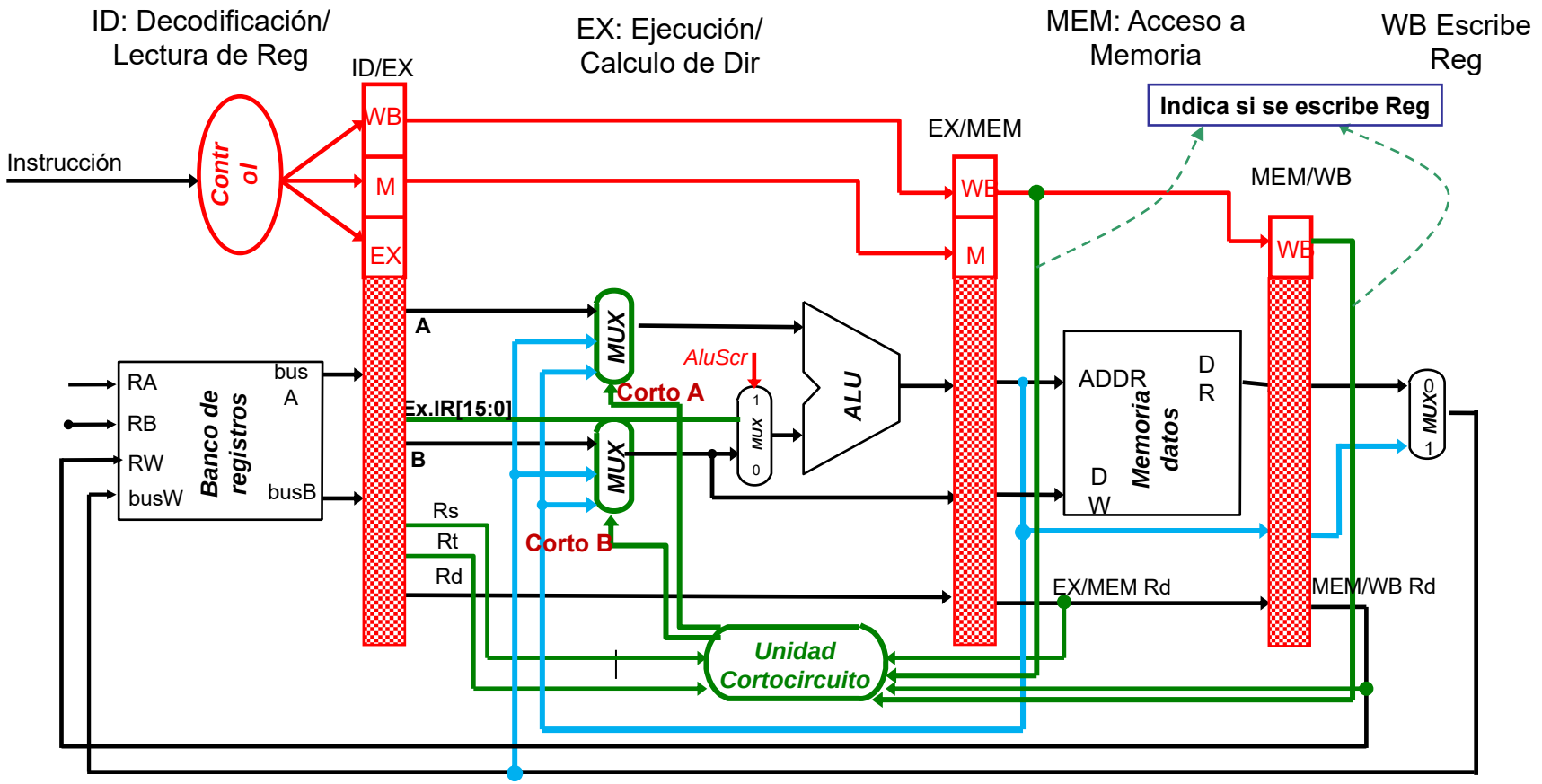
- **Para implementar el cortocircuito necesitamos dos caminos de datos:**
  - Desde el registro de pipeline EX/MEM (salida de la ALU) a entrada ALU
  - Desde el registro de pipeline MEM/WB (salida de la memoria) a entrada ALU

¿Cómo resolver los riesgos LDE? Solución 3: Cortocircuito (forwarding)

- **Información necesaria para implementar el cortocircuito:**
  - Registro a escribir en última etapa
    - EX/MEM.Rd
    - MEM/WB.Rd
  - Registros que se leen en segunda etapa (Rn y Rm)
    - ID/EX.Rn
    - ID/EX.Rm
  - Información sobre si se escribe en el banco de registros
    - EX/MEM.RegWrite
    - MEM/WB.RegWrite

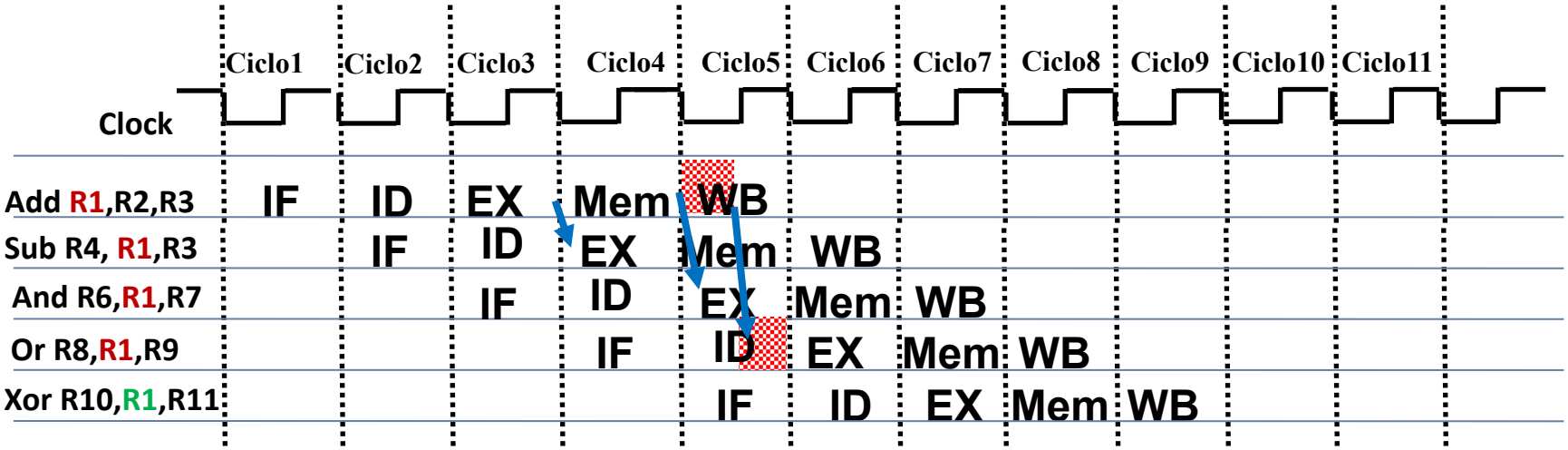


Riesgos LDE: Implementación del cortocircuito



¿Cómo resolver los riesgos LDE?

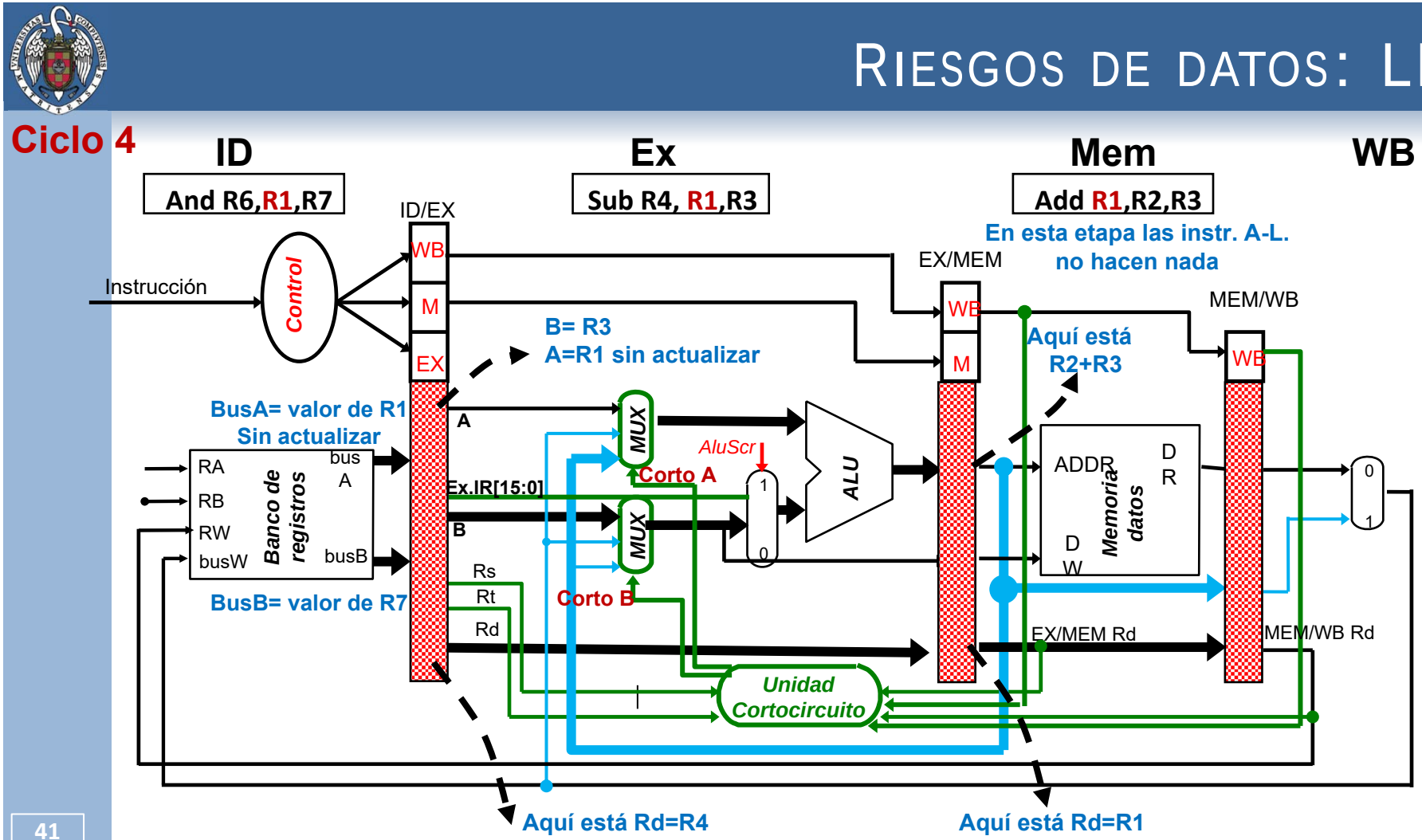
- **Solución 3:** Cortocircuito (forwarding)



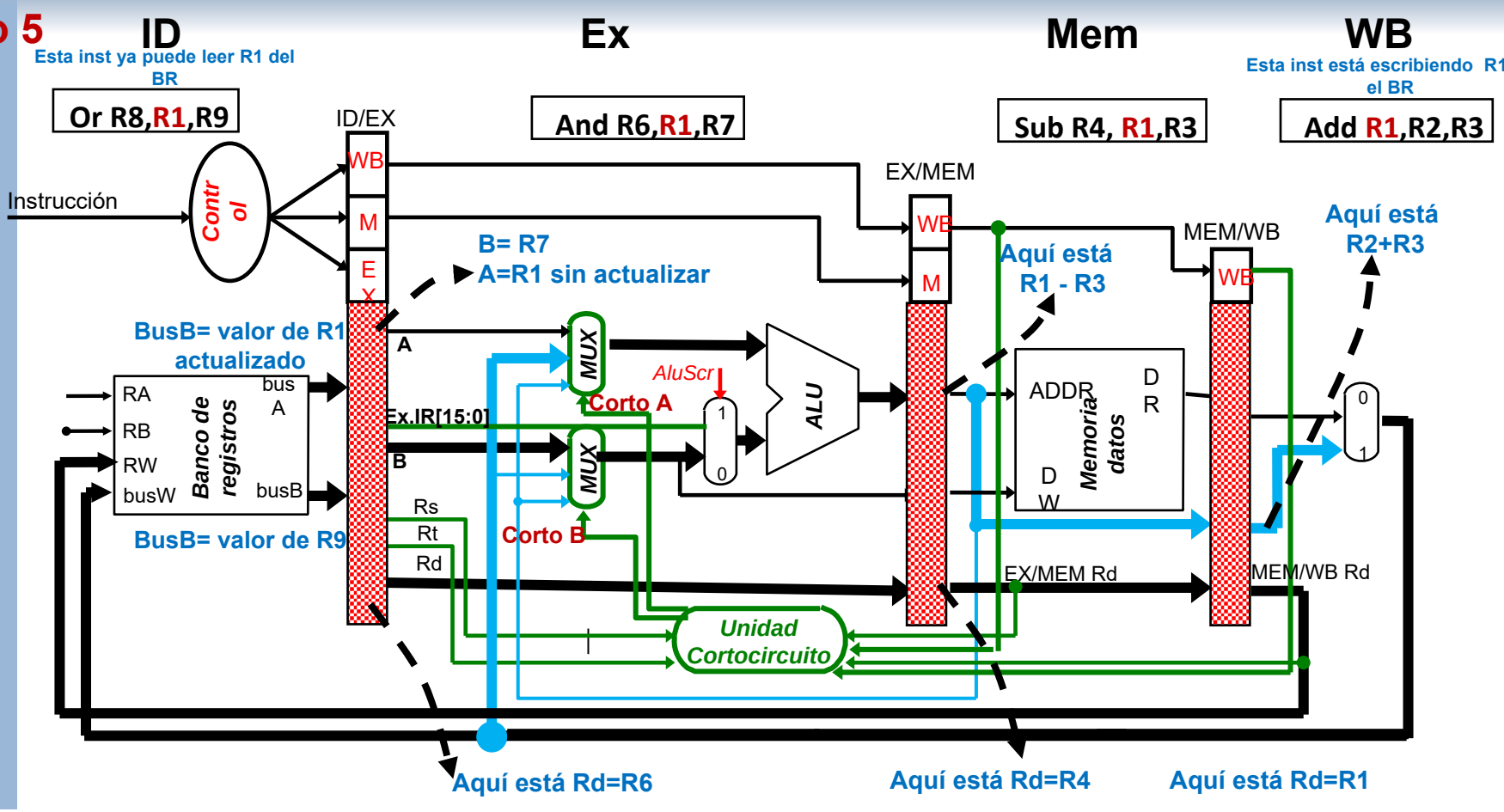
No hay ciclos de espera

Se pueden ejecutar todas las instrucciones sin problemas

RIESGOS DE DATOS: LDE

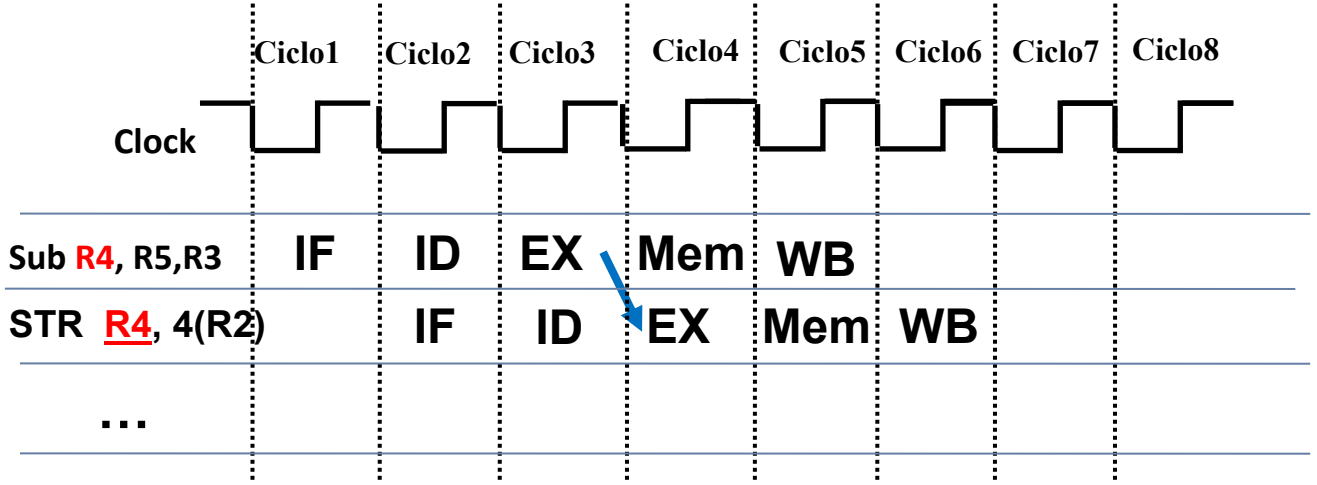


RIESGOS DE DATOS: LDE



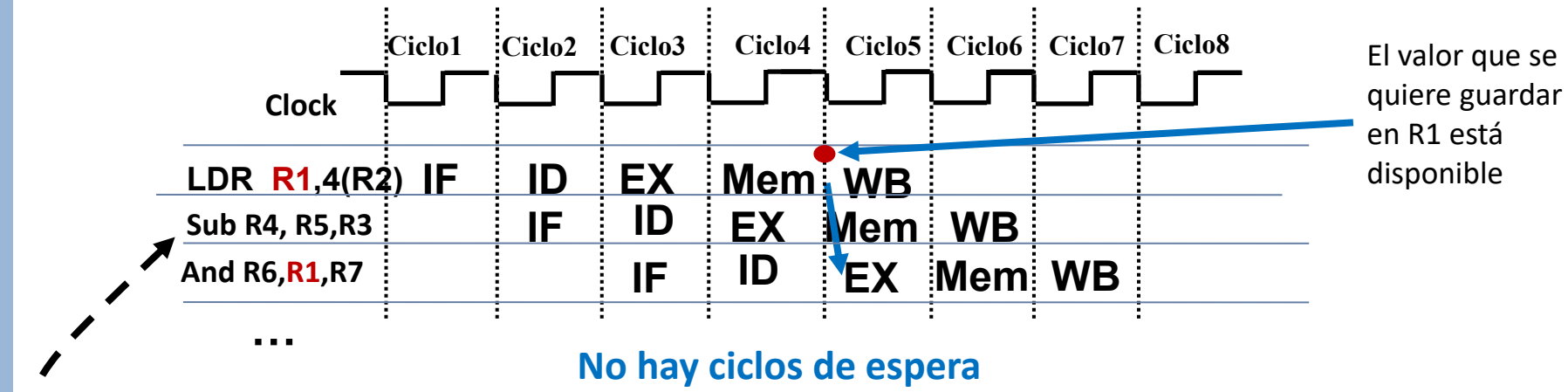
Riesgo LDE: ejemplo 1 con instrucción store

- El store escribe en memoria en la etapa **Mem** pero en nuestra ruta de datos el dato que va a escribir en memoria lo tiene que tener en la etapa **Ex**
  - Se puede resolver con cortocircuito



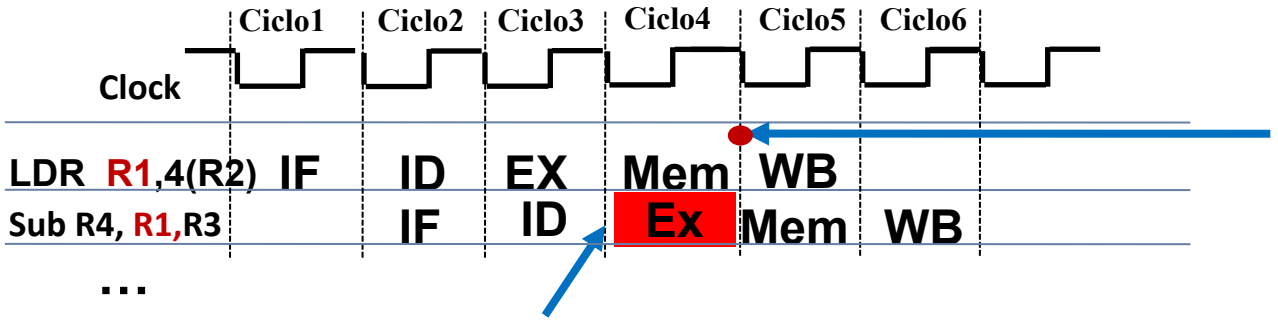
No hay ciclos de espera

- **Riesgo LDE:** caso particular, **el dato lo proporciona un Load**



- Se puede resolver sólo con cortocircuito si hay una instrucción intermedia

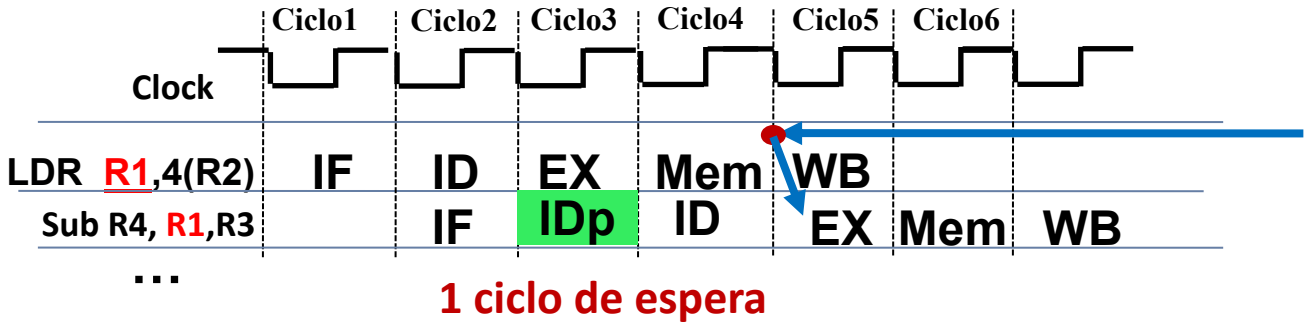
- **Riesgo LDE:** caso particular, **el dato lo proporciona un Load**



- No se puede resolver sólo con el cortocircuito

Riesgo LDE: caso particular, **el dato lo proporciona un Load**

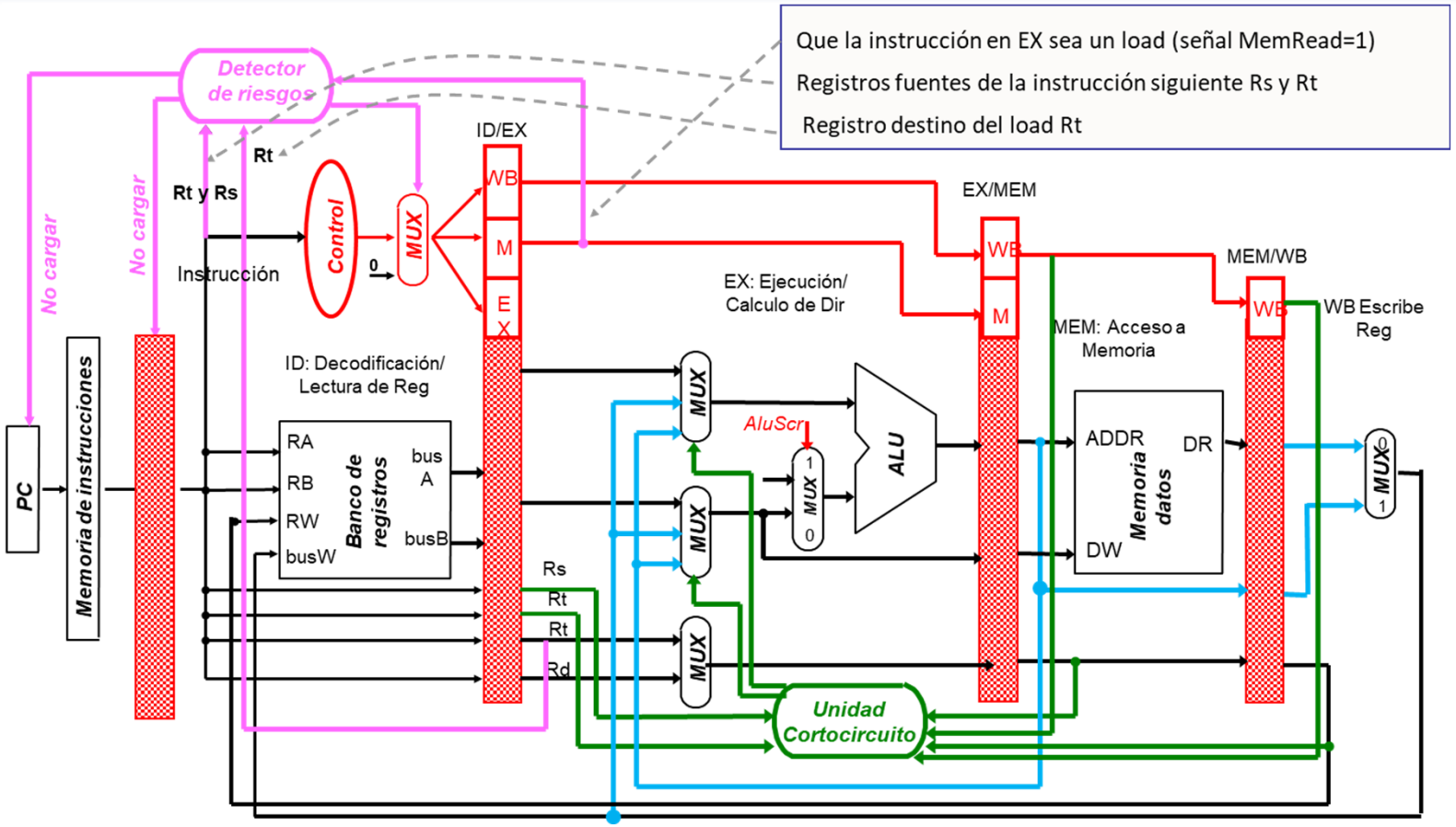
- **Solución HW:** Detección del riesgo y parada del procesador un ciclo



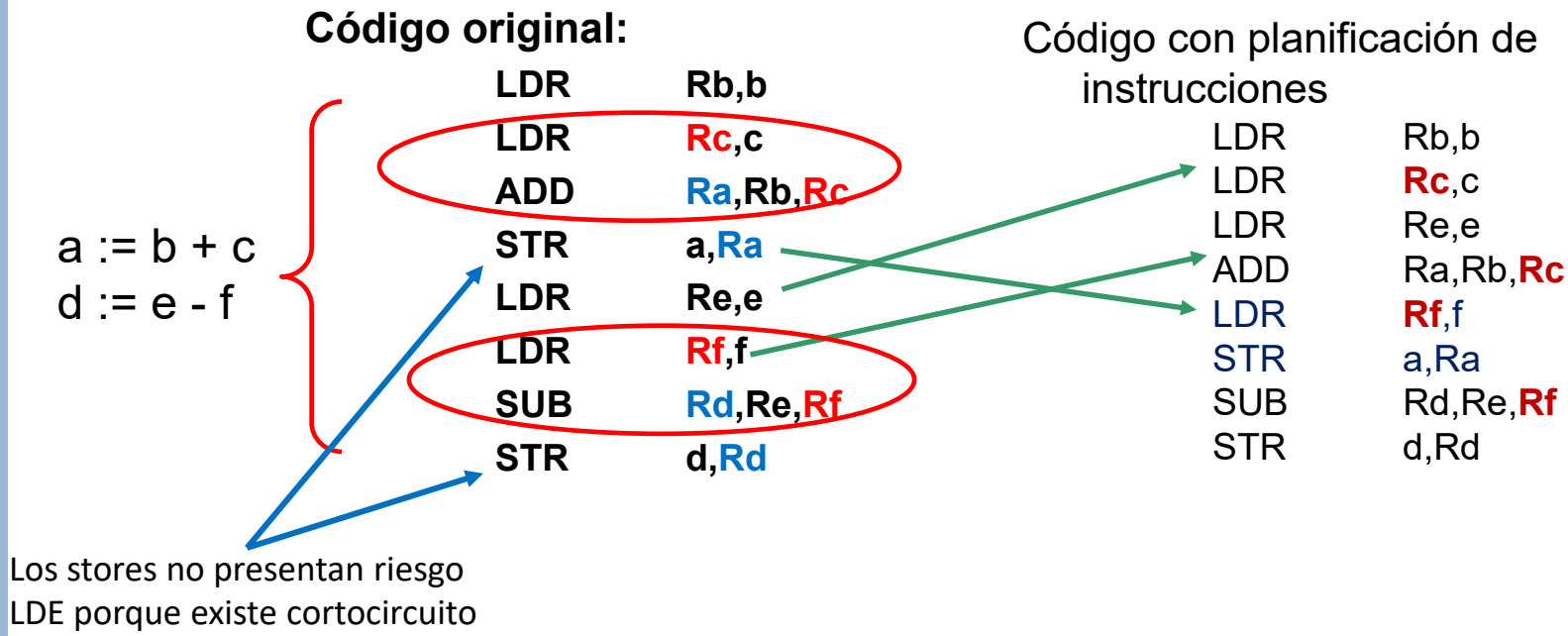
El valor que se quiere guardar en R1 está disponible

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

- Detección de riesgos y parada del procesador un ciclo



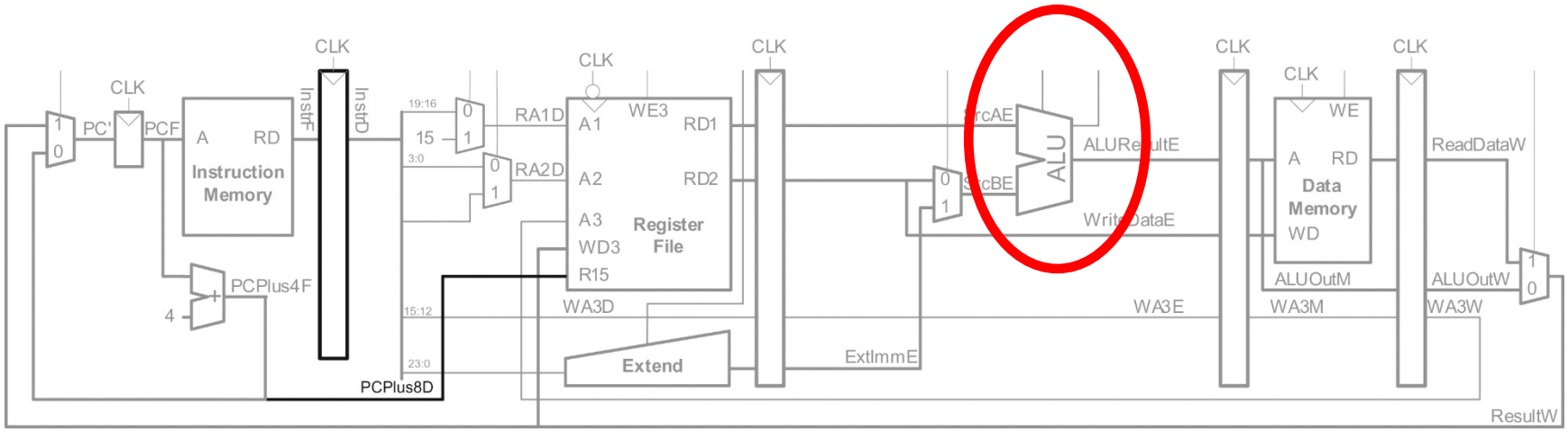
- **Riesgo LDE:** caso particular, **el dato lo proporciona un Load**
  - **Solución SW**: Anticipar el Load en la planificación de instrucciones que hace el compilador



ÍNDICE

1. MIPS
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. **Riesgos de control**
6. Operaciones multiciclo
7. Rendimiento en los procesadores

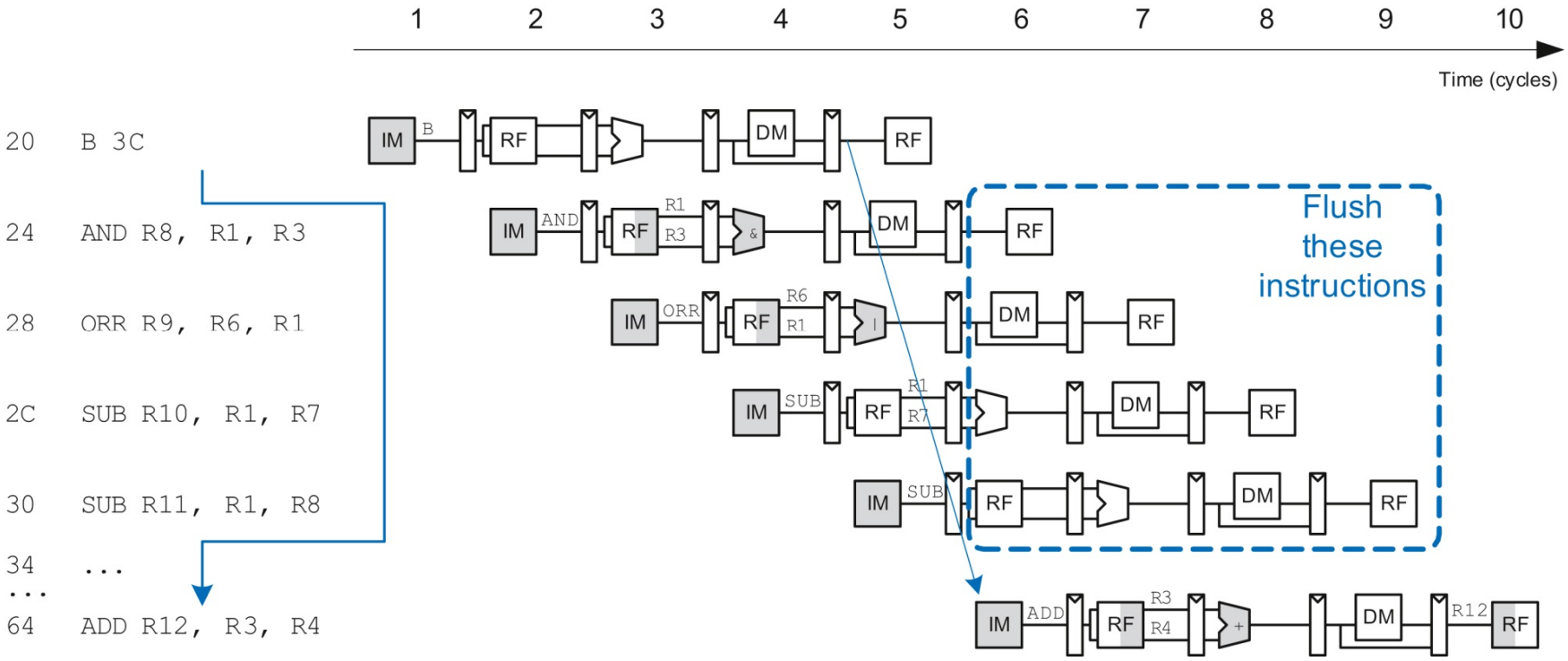
- Riesgos de control: ¿Por qué aparecen?
  - Para saltar se necesita haber calculado la dirección de salto
  - Aparece el riesgo porque en la ruta de datos que hemos estudiado la dirección de salto se calcula en EX



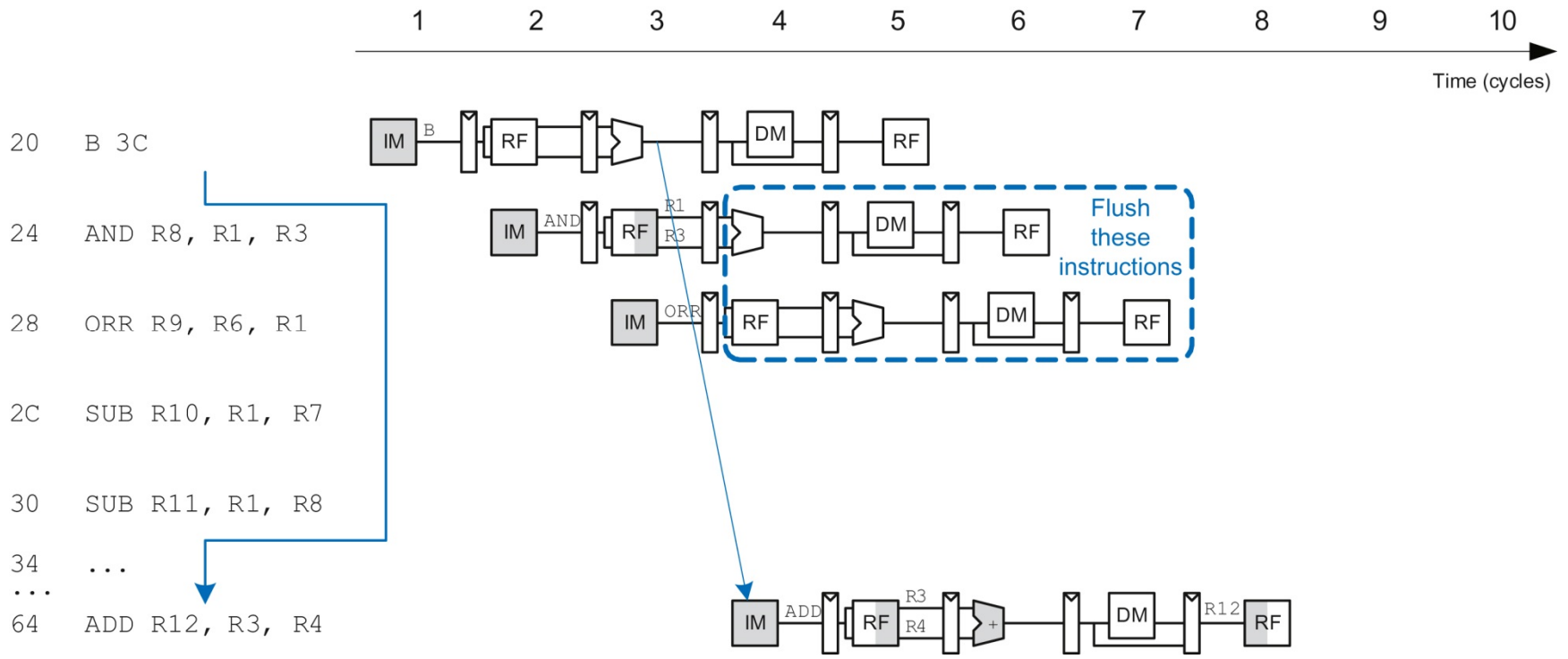
Solución 1: Seguir ejecutando instrucciones hasta que se conozca el destino del salto. Descartar las instrucciones comenzadas erróneamente.

Nota: ¿Cómo se comporta esta solución para saltos condicionales?

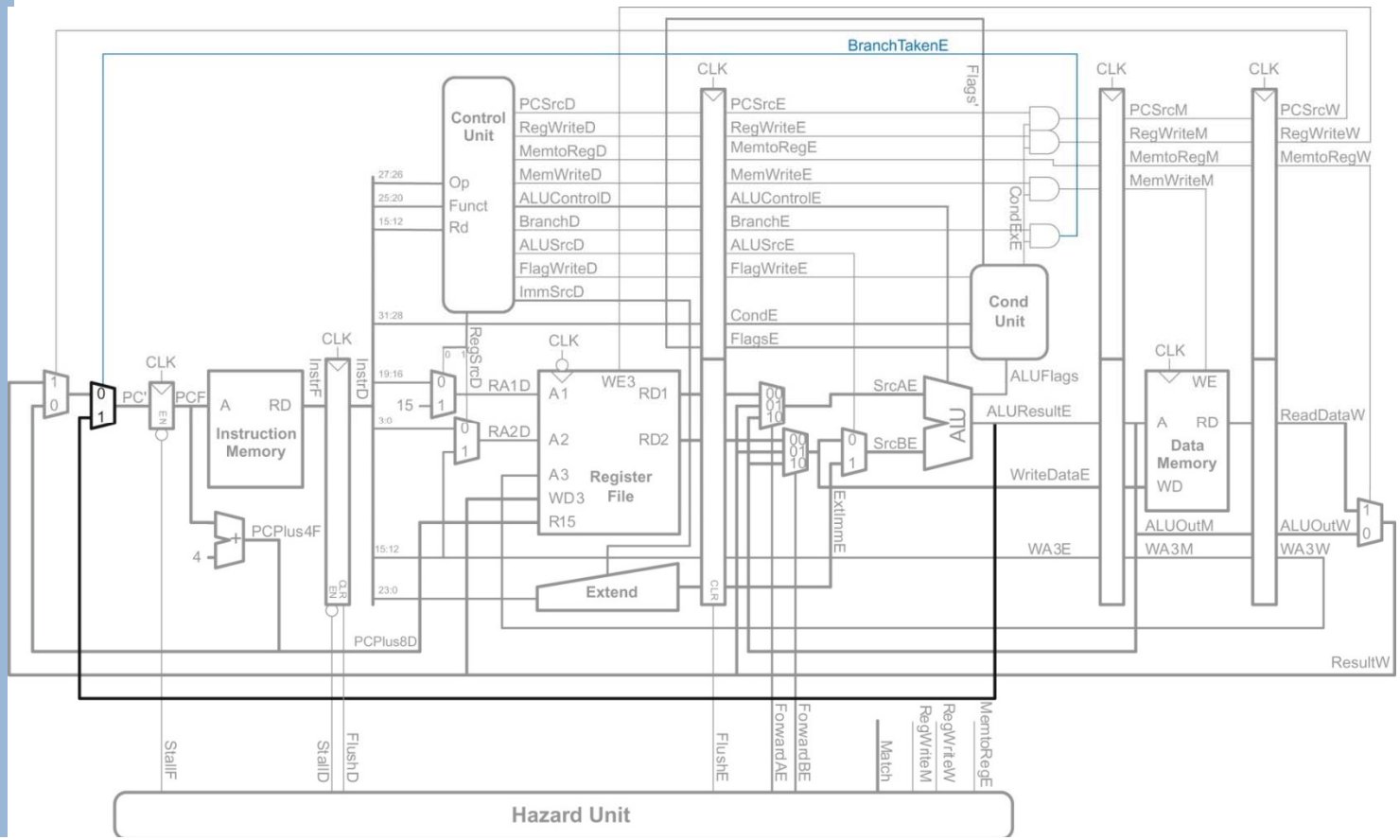
Solución 2 Detener el pipeline durante 4 ciclos hasta que se conozca el destino del salto



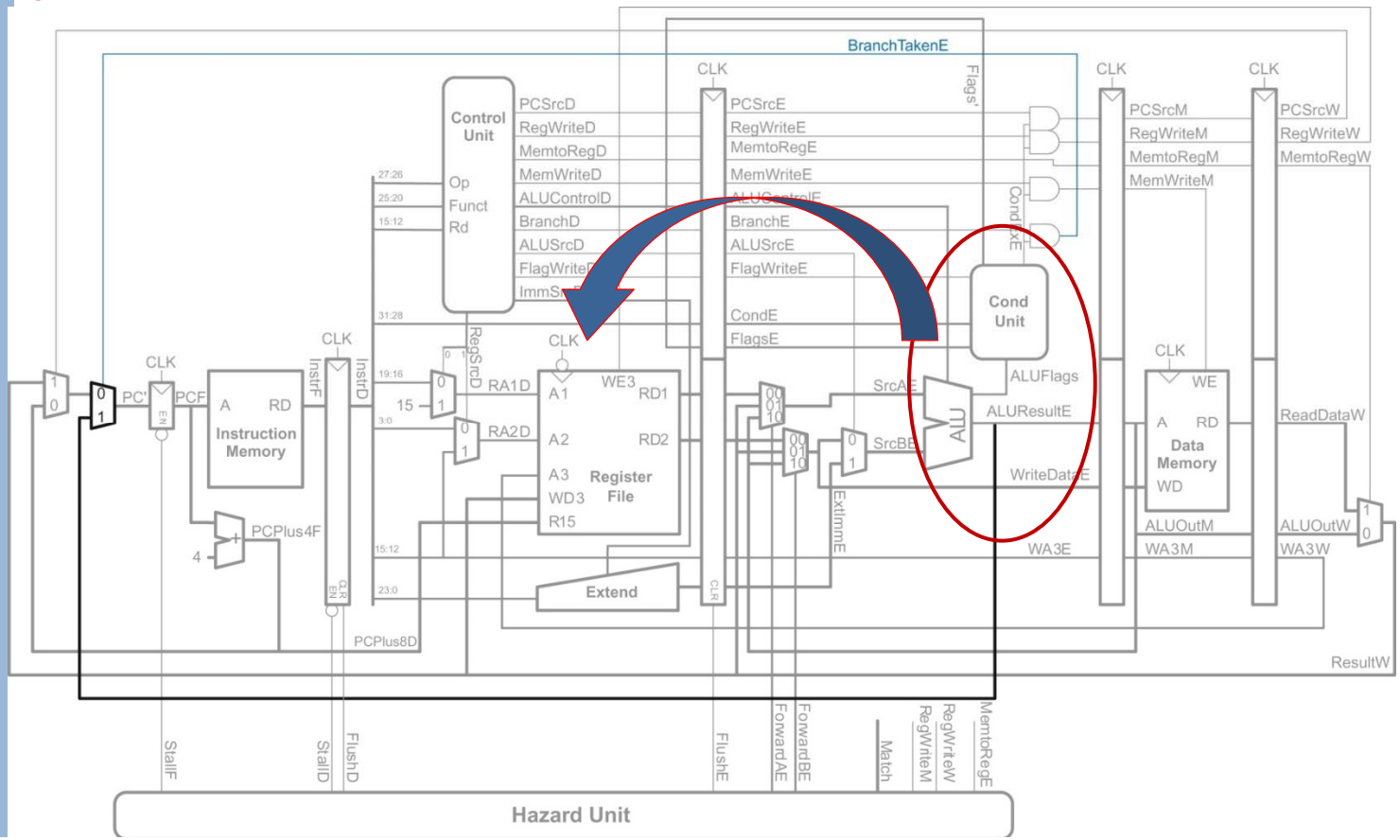
Solución 3: Como la dirección de salto se conoce en la etapa 3, cortocircuitarla hacia el PC, de forma que sólo haya que descartar dos instrucciones



Solución 3: Modificaciones a la ruta de datos



¿Podría reducirse aún más el número de ciclos de penalización?



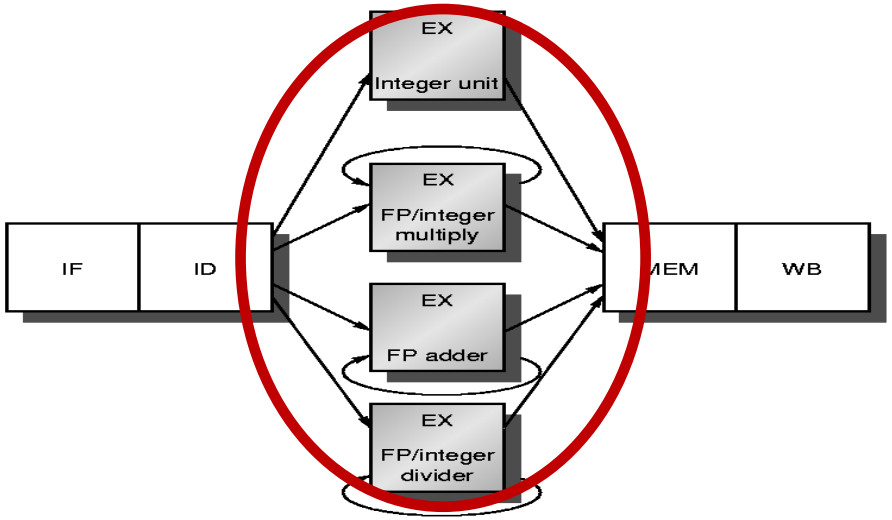
ÍNDICE

1. MIPS
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. **Operaciones multiciclo**
7. Rendimiento en los procesadores

OPERACIONES MULTICICLO

- ¿ Qué ocurre si las instrucciones tienen diferente duración?
  - Esto ocurre cuando la operaciones requieren más de un ciclo de ejecución
    - Las operaciones en punto flotante
    - La multiplicación y división de enteros
  - **Latencia de las UFs** : Nº de ciclos de duración de una instrucción en una UF

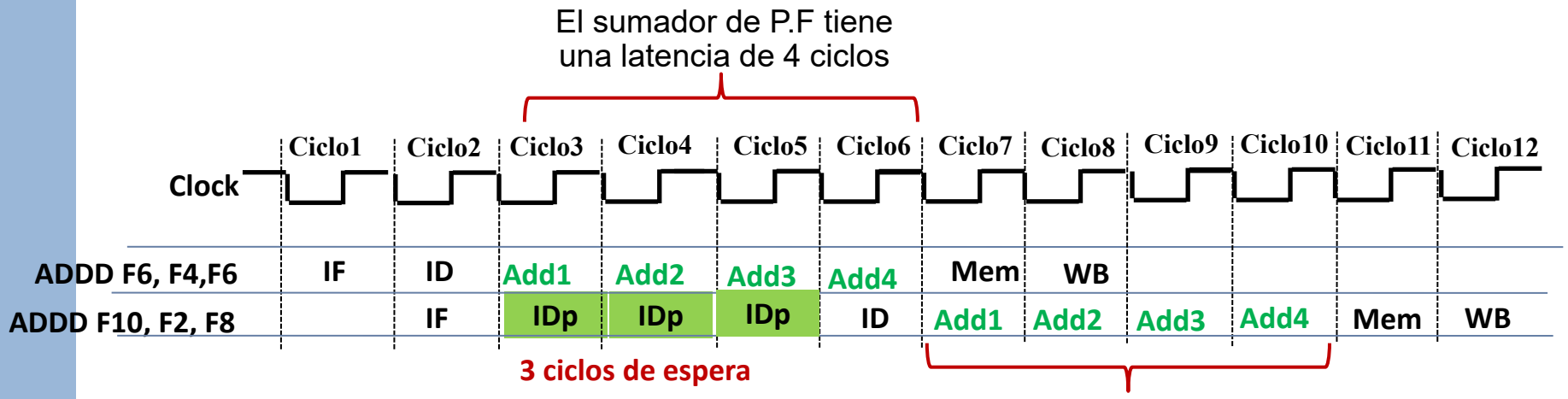
Unidad funcional	Latencia
ALU entera	1
Suma PF	4
Multiplicación PF	7
División PF	24



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 haya acabado la operación de la primera instrucción

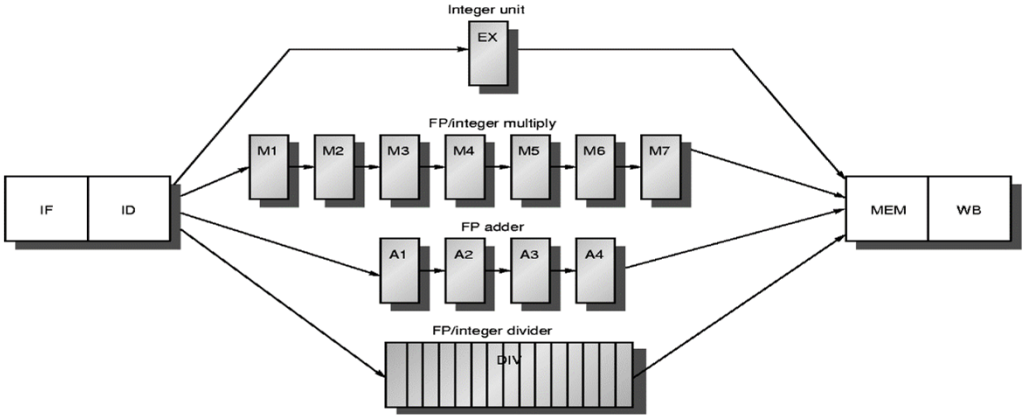


IDp ✓ID_P parada por riesgo estructural, se necesita el sumador y no está disponible

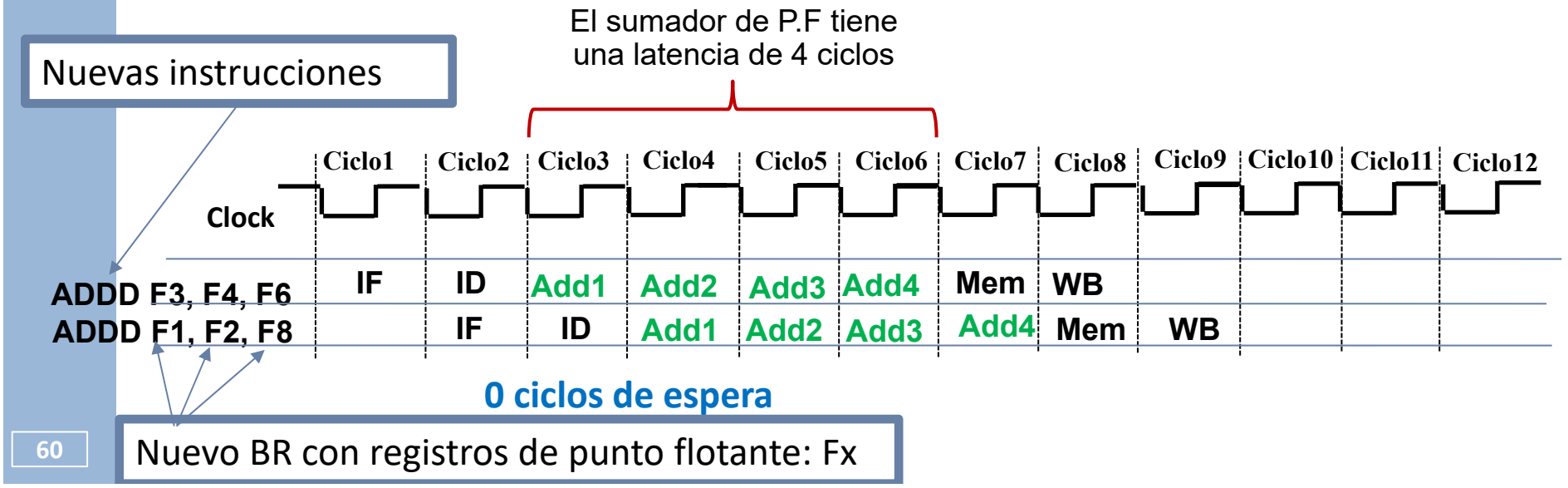
Riesgos estructurales

- Solución: Segmentar las UFs con latencia > 1
- **Intervalo de iniciación:** Nº de ciclos que tiene que esperar una instrucción para poder utilizar una UF que está utilizando otra
- La división no suele estar segmentada: se tiene que detectar el riesgo y realizar paradas

Unidad funcional	Latencia	Intervalo de iniciación
ALU entera	1	1
Suma PF	4	1
Multiplicación PF	7	1
División PF	24	24



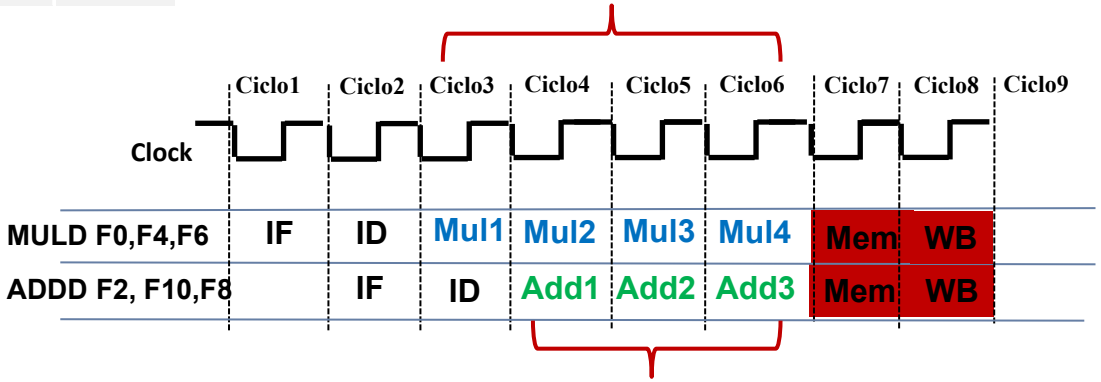
- **Riesgos estructurales**
  - Solución: Segmentar las UF con latencia > 1
  - Sólo hay que esperar que la UF haya acabado la operación asociada al primer ciclo de ejecución de la primera instrucción



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

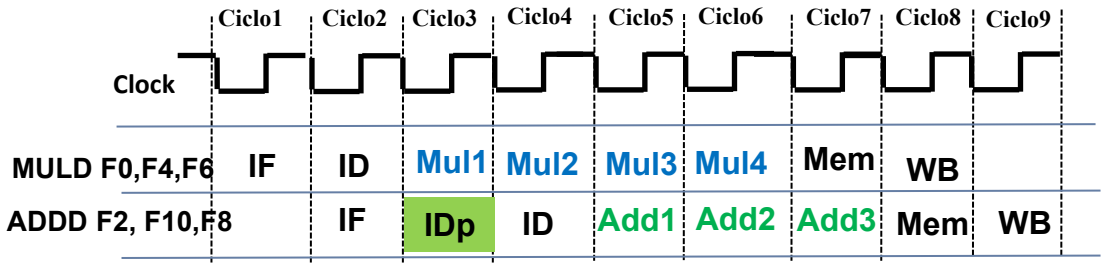
Unidad Funcional	Latencia
FP add	3
FP multiplicador	4



Solución 1:

- **Detener** la segunda instrucción en la **etapa de decodificación**

Unidad Funcional	Latencia
FP add	3
FP multiplicador	4

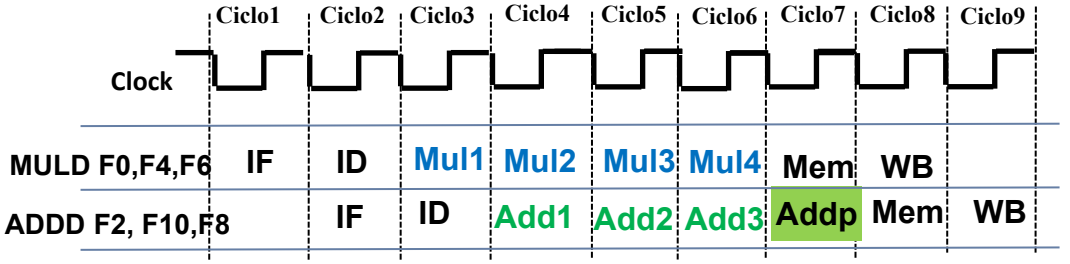


IDp ✓ID_P parada por **riesgo estructural,** dos instrucciones van a acceder a la vez a la memoria de datos

Solución 2:

- **Detener** las instrucciones conflictivas **al final de la fase de ejecución**
  - Necesidad de establecer prioridades de acceso
    - Dar mayor prioridad a la unidad de mayor latencia o a la que comienza más tarde
  - Lógica de detección y generación de paradas en dos puntos diferentes

Unidad Funcional	Latencia
FP add	3
FP multiplicador	4



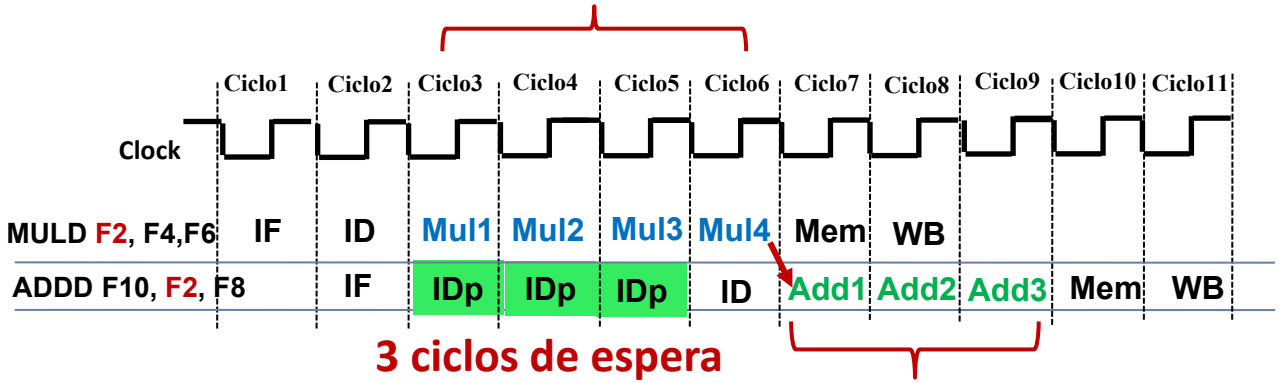
Addp ✓ID_P parada por **riesgo estructural,** dos instrucciones van a acceder a la vez a la memoria de datos

OPERACIONES MULTICICLO: RIESGOS LDE

Qué pasa cuando hay riesgo LDE y el dato lo proporciona una instrucción Aritmético-Lógica

- El **cortocircuito NO elimina todas las paradas**

Unidad Funcional	Latencia
FP add	3
FP multiplicador	4



IDp D_P **parada por riesgo LDE** entre instrucción 1 e instrucción 2

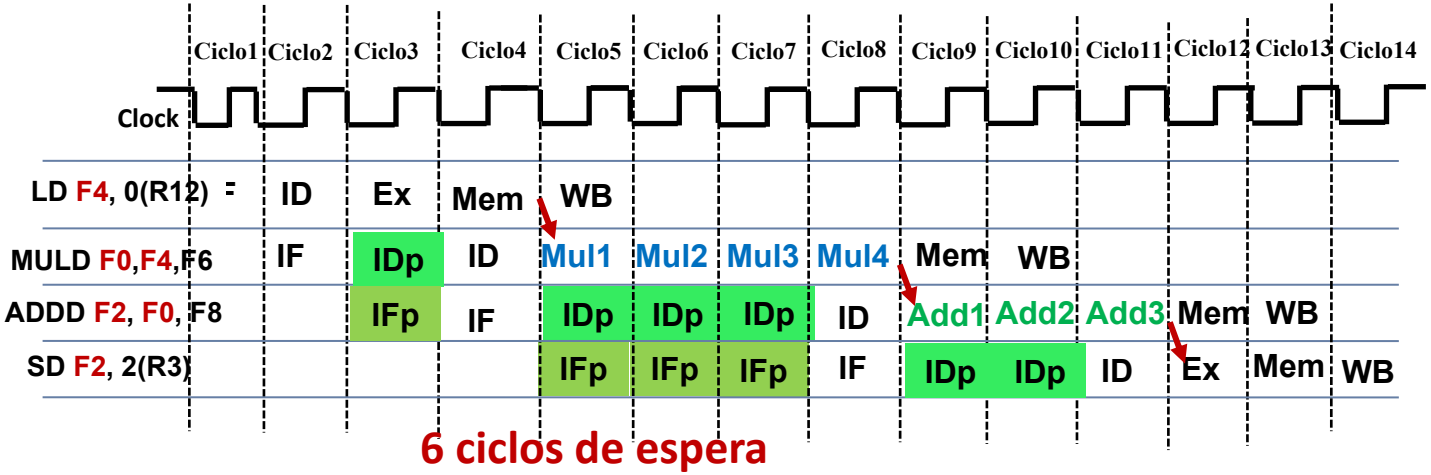
64

OPERACIONES MULTICICLO: RIESGOS LDE

Qué pasa cuando hay Riesgo LDE y el dato lo proporciona un load

- La instrucción que depende del load además tiene una parada
  - El **cortocircuito NO elimina todas las paradas**

Unidad Funcional	Latencia
FP add	3
FP multiplicador	4



6 ciclos de espera

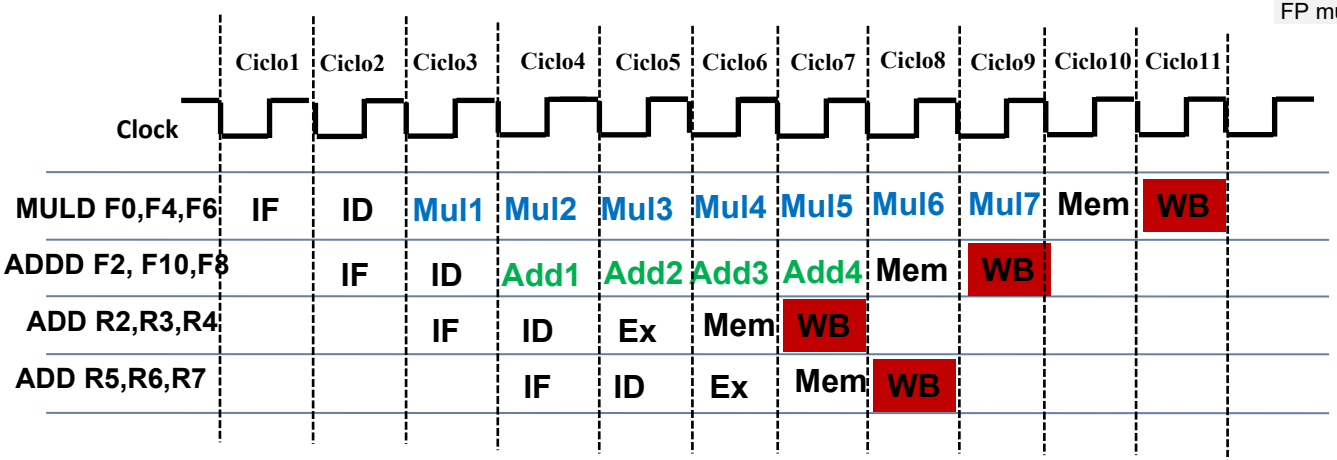
IDp **parada por riesgo LDE** entre instrucción 1 e instrucción 2

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

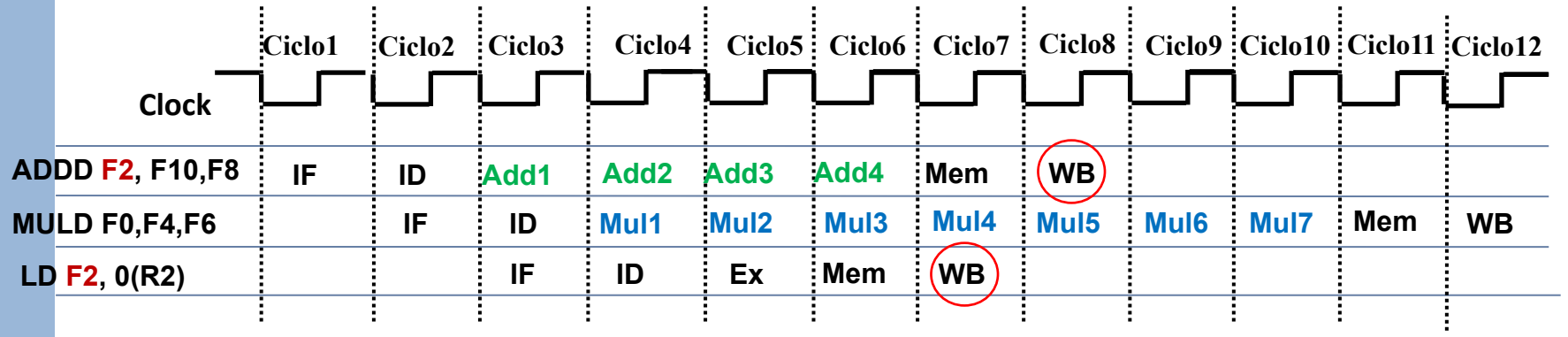
Finalización fuera de orden

- 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	Latencia
FP add	4
FP multiplicador	7

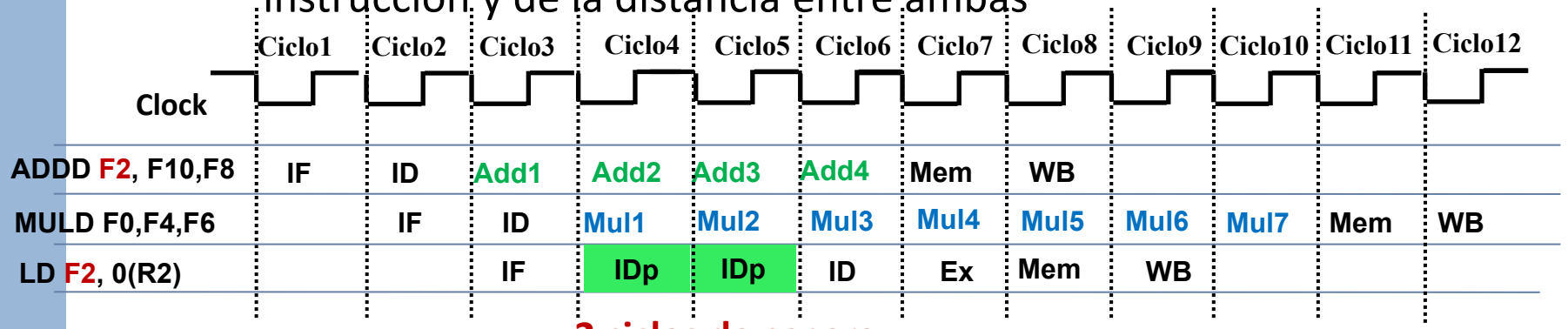


- Problema: Escritura después de escritura



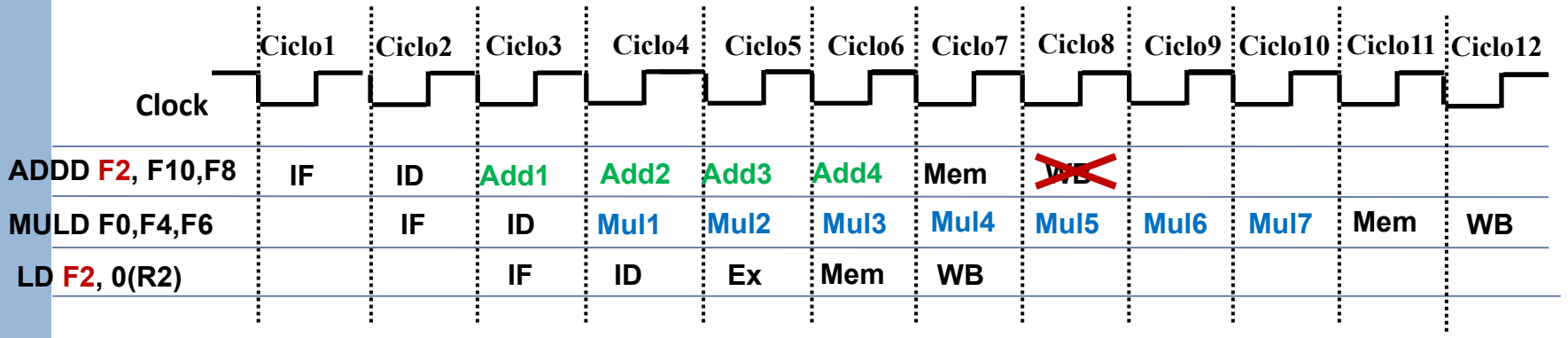
OPERACIONES MULTICICLO: RIESGOS EDE

- 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



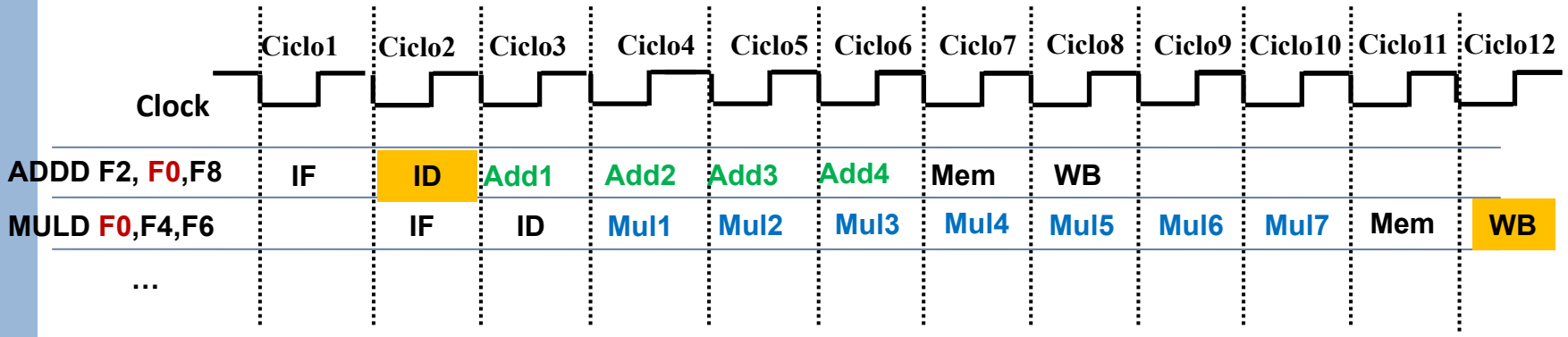
IDp ✓ID_P **parada por riesgo EDE** entre instrucción 1 e instrucción 2

- 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

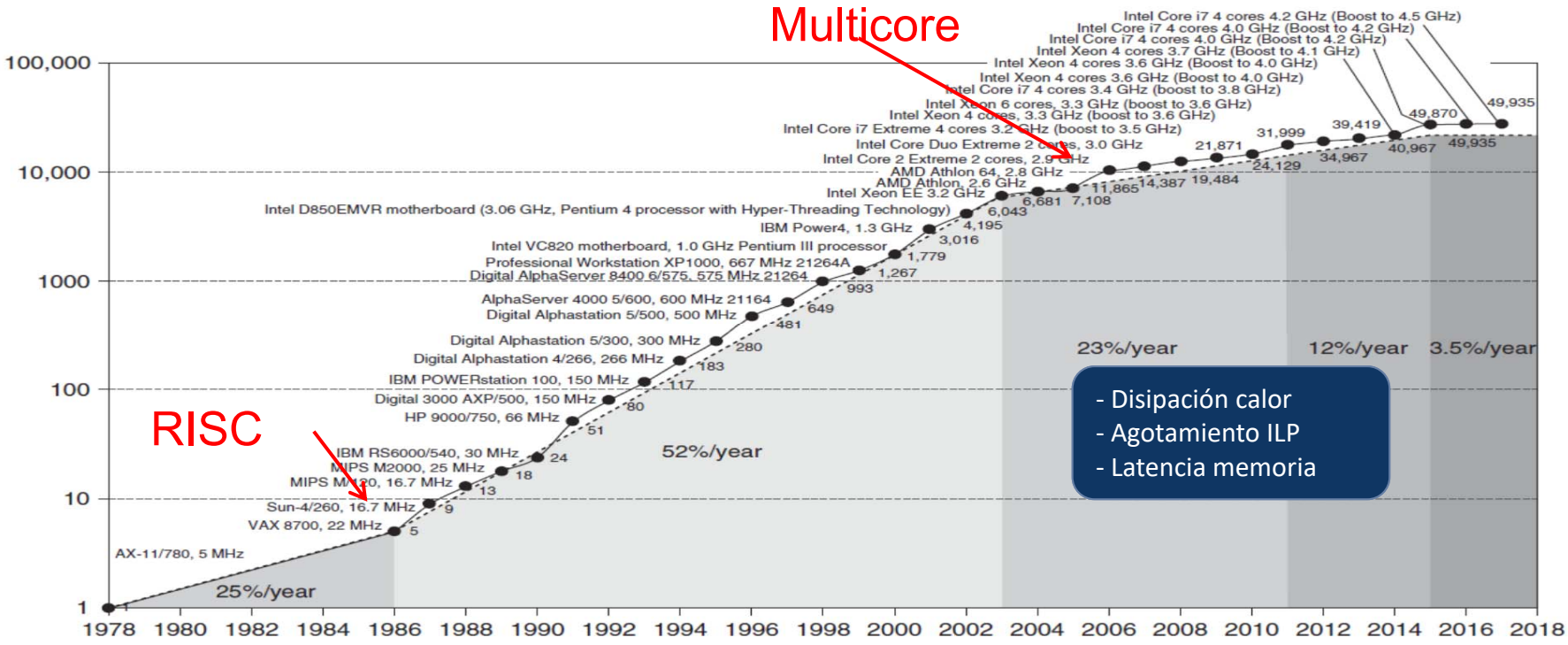
1. MIPS
2. Segmentación
3. Riesgos estructurales
4. Riesgos de datos
5. Riesgos de control
6. Operaciones multiciclo
7. **Rendimiento de los procesadores**

"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 MIPS multiciclo

lw r1, 0(r0) → 5

lw r2, 4(r0) → 5

add r3, r1, r2 → 4

beq r3, r5, 1 → 4

sub r3, r3, r5 → 4

sw r3, 8(r0) → 4

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

MEDIDAS DEL RENDIMIENTO

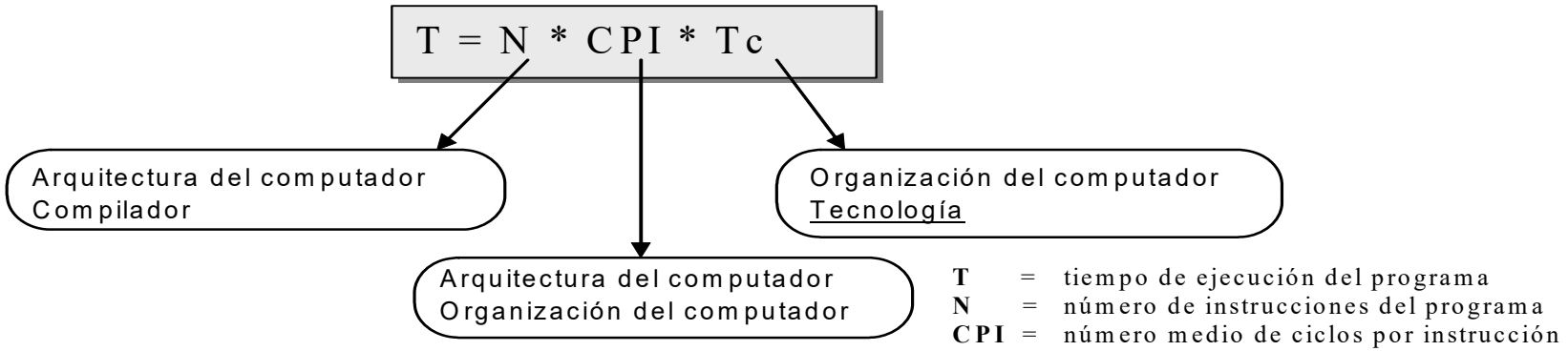
- Para poder comparar diferentes procesadores hace falta establecer una medida del rendimiento que permita cuantificar los resultados de la comparación
  - **Métrica**: establece la unidad de medida, que casi siempre es el tiempo, aunque hay que considerar dos aspectos diferentes del tiempo:
    - **Tiempo 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.

PÉRDIDA DE RENDIMIENTO

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

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

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

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

  CPI = 1 + (1 x 0.3) = 1.3

$$Speep\_up = \frac{Nºinstrucciones * Nº\ etapas}{Nºinstrucciones * CPI} = \frac{5}{1{,}3} = 3{,}84$$

- Eficiencia:
  - $Speedup_{real} / Speedup_{ideal} = 3.84 / 5 = 0.76$

  Se pierde un 24 % respecto al caso ideal

MEDIDAS DEL RENDIMIENTO: MIPS

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

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

- 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)

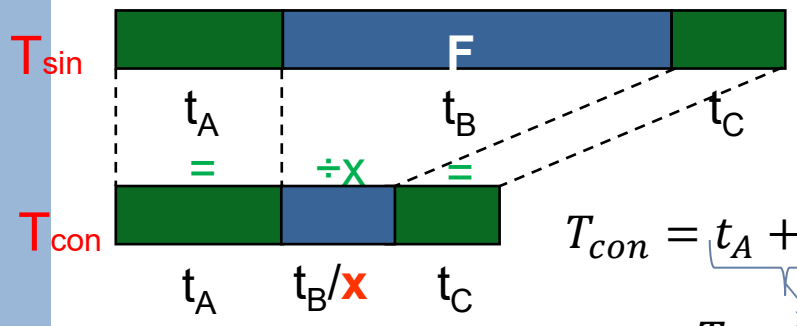
$$MFLOPS = \frac{Numero\ de\ operaciones\ en\ punto\ flotante\ de\ un\ programa}{Tiempo\ de\ ejecucion\ *\ 10^6}$$

  - 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:

- **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)