

## Tema 2. Paralelismo a nivel de Instrucción

### ESTRUCTURA DE COMPUTADORES

Grupo de Arquitectura de Computadores (GAC)

# Índice

- 1 Objetivos del tema
- 2 Introducción a la segmentación de cauce
- 3 Dependencias y paralelismo a nivel de instrucción
- 4 Riesgos en la ejecución
- 5 Cauce segmentado en el MIPS
- 6 Repertorio de instrucciones para cauces segmentados
- 7 Conclusiones

# Índice

- 1 Objetivos del tema
- 2 Introducción a la segmentación de cauce
- 3 Dependencias y paralelismo a nivel de instrucción
- 4 Riesgos en la ejecución
- 5 Cauce segmentado en el MIPS
- 6 Repertorio de instrucciones para cauces segmentados
- 7 Conclusiones

# Objetivos del tema

En el curso pasado (**Fundamentos de Computadores**) se ha estudiado el diseño de un procesador multiciclo básico. En este curso, partiendo de los conceptos vistos anteriormente, se discutirá en detalle el concepto de segmentación de cauce utilizado en los computadores modernos para proporcionar altas prestaciones.

En este tema se estudiará:

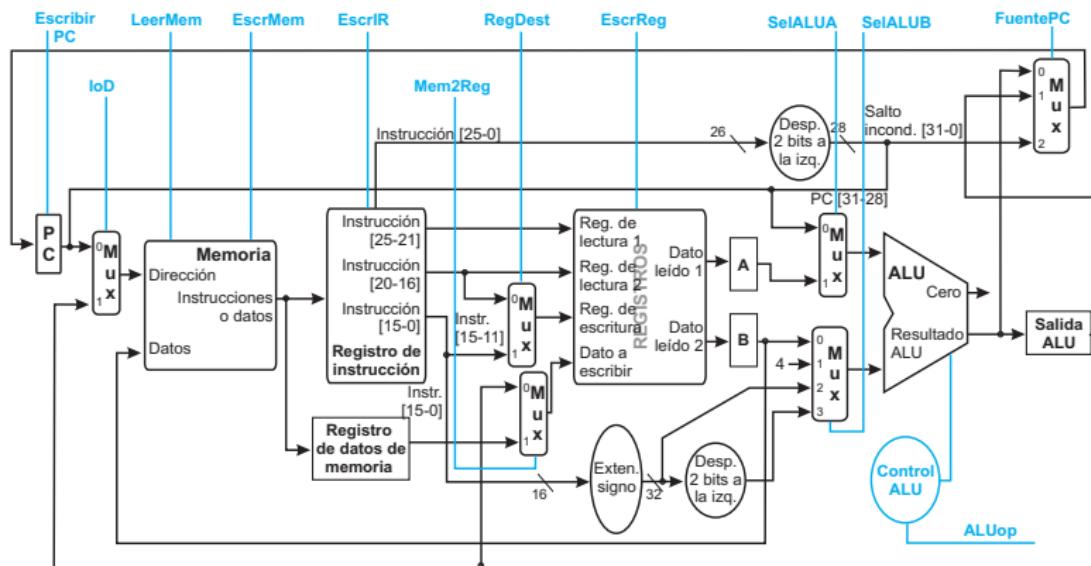
- La segmentación de cauce (**pipelining**) como forma de ejecución concurrente de instrucciones
- Como el paralelismo a nivel de instrucción determina las instrucciones que se pueden ejecutar en paralelo
- Diversos riesgos que ocasionan una degradación de las prestaciones de los procesadores segmentados y las formas de mitigar sus efectos
- Las implicaciones hardware y software de la segmentación de cauce
- La influencia de la segmentación de cauce en el diseño del repertorio de instrucciones

# Índice

- 1 Objetivos del tema
- 2 Introducción a la segmentación de cauce
- 3 Dependencias y paralelismo a nivel de instrucción
- 4 Riesgos en la ejecución
- 5 Cauce segmentado en el MIPS
- 6 Repertorio de instrucciones para cauces segmentados
- 7 Conclusiones

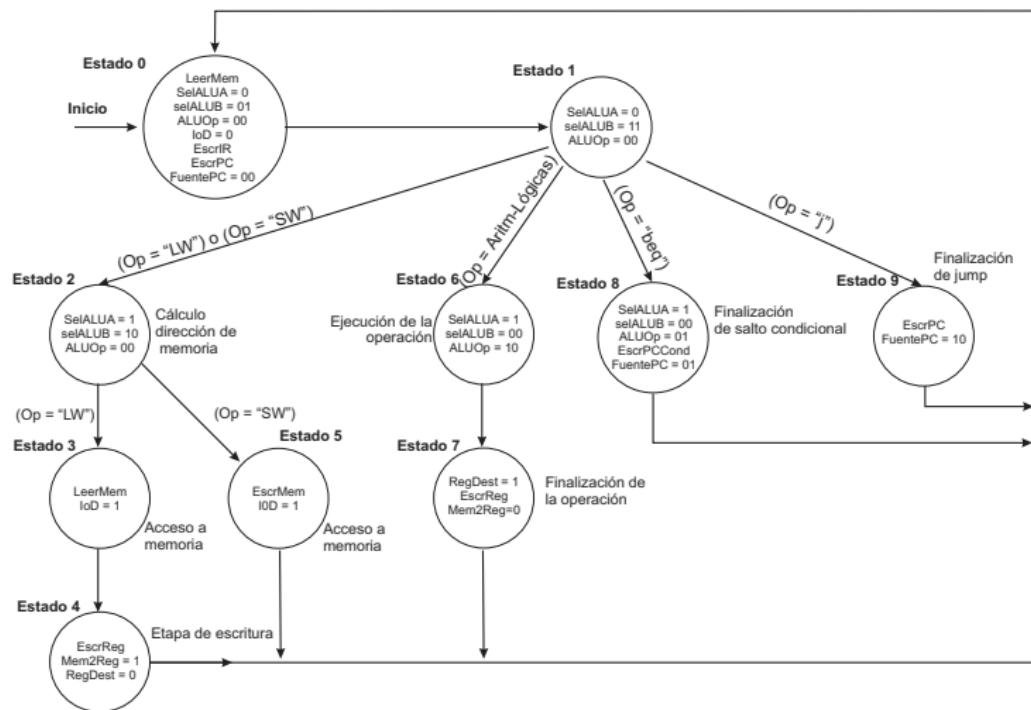
# Reaso: procesador multiciclo

Procesador MIPS multiciclo visto en FC:



# Reaso: procesador multiciclo

Procesador MIPS multiciclo visto en FC:



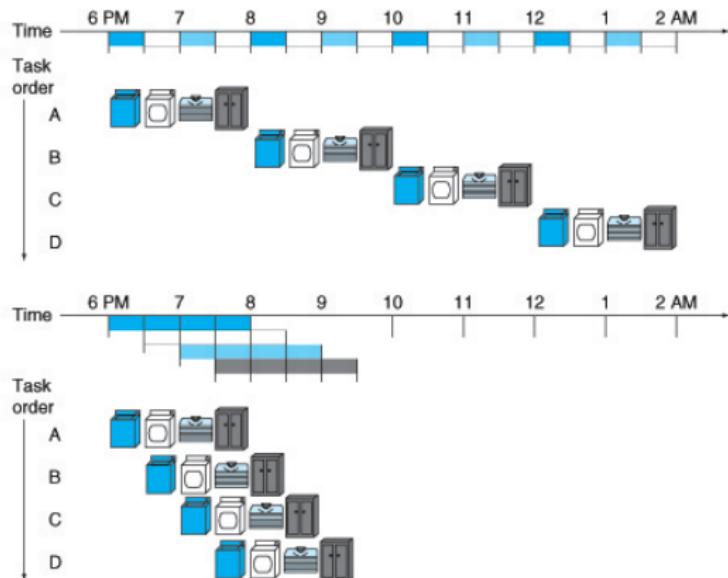
# Introducción

Hay dos posibilidades para incrementar el rendimiento:

- Mejorar la tecnología de los circuitos integrados para reducir el ciclo de reloj
  - ▶ Reduce el tiempo necesario para ejecutar cada una de las etapas
- Mejorar la organización del hardware para que pueda realizar más de una operación al mismo tiempo
  - ▶ Se incrementa el número de operaciones que puede realizar por segundo
  - ▶ No cambia el tiempo necesario para realizar una operación determinada

# Segmentación de cauce

## Concepto de segmentación



# Segmentación de cauce

- La segmentación del cauce:
  - ▶ Persigue aumentar las prestaciones
  - ▶ Divide la ejecución de una instrucción en etapas
  - ▶ Permite solapar la ejecución de distintas instrucciones en diferentes etapas
  - ▶ Permite comenzar una instrucción en cada ciclo de reloj (caso ideal)
- Etapas de una instrucción MIPS en el camino de datos:
  - ▶ **IF**: búsqueda de instrucciones
  - ▶ **ID**: decodificación de instrucciones y lectura del banco de registros
  - ▶ **EX**: ejecución o cálculo de direcciones
  - ▶ **MEM**: acceso a memoria para buscar un dato
  - ▶ **WB**: escritura de resultados

# Segmentación de cauce

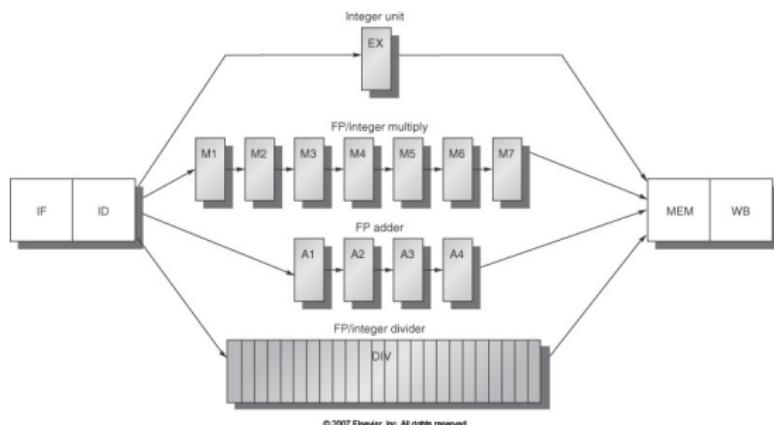
Comparación entre la implementación multiciclo y la segmentada:

Instrucciones	1	2	3	4	5	6	7	8	9	10	11	12
lw \$1,16(\$2)	IF	ID	EX	MEM	WB							
addi \$2,\$2,4						IF	ID	EX	MEM			
beq \$3,\$0,loop										IF	ID	EX

Instrucciones	1	2	3	4	5	6	7	8	9	10	11	12
lw \$1,16(\$2)	IF	ID	EX	MEM	WB							
addi \$2,\$2,4		IF	ID	EX	MEM	WB						
beq \$3,\$0,loop			IF	ID	EX	MEM	WB					

# Operaciones multiciclo

- Algunas operaciones tardan más de un ciclo de reloj en la etapa de ejecución
- Las instrucciones en punto flotante tienen la misma segmentación que las enteras, pero con las siguientes modificaciones:
  - ▶ La etapa EX tiene diferente latencia dependiendo de la instrucción
  - ▶ Existen diversas unidades funcionales para cada tipo de operación



# Índice

- 1 Objetivos del tema
- 2 Introducción a la segmentación de cauce
- 3 Dependencias y paralelismo a nivel de instrucción
- 4 Riesgos en la ejecución
- 5 Cauce segmentado en el MIPS
- 6 Repertorio de instrucciones para cauces segmentados
- 7 Conclusiones

## Definición

El paralelismo a nivel de instrucción (*Instruction Level Parallelism-ILP*) es la capacidad de procesar instrucciones en paralelo. Viene determinado por el número de instrucciones que pueden solaparse en las etapas de un procesador.

Dos instrucciones son dependientes si se deben ejecutar en orden:

- ① Indican la posibilidad de un riesgo (una instrucción no se puede ejecutar en el ciclo de reloj que le correspondería)
- ② Determinan el orden en el cual se deben calcular los resultados.
- ③ Establecen la cantidad máxima de paralelismo que se puede obtener.

Hay tres tipos de dependencias:

- de datos o verdaderas
- de nombre
- de control

## Dependencias de datos

Una instrucción depende de los resultados de una instrucción anterior, de forma que ambas no podrían ejecutarse de forma solapada.

Una instrucción  $j$  tiene una dependencia de datos con la instrucción  $i$  si ocurre cualquiera de las situaciones siguientes:

- La instrucción  $i$  produce un resultado que puede ser usado por la instrucción  $j$ .

$$i : lwc1 \quad \$f0, \quad 0(\$r1)$$
$$j : add.d \quad \$f2, \quad \$f0, \quad \$f4$$


- La instrucción  $j$  tiene una dependencia de datos con la instrucción  $k$ , y la instrucción  $k$  tiene una dependencia de datos con la instrucción  $i$ .

$$i : lwc1 \quad \$f0, \quad 0(\$r1)$$
$$k : mul.d \quad \$f4, \quad \$f0, \quad \$f10$$
$$j : add.d \quad \$f2, \quad \$f6, \quad \$f4$$


## Dependencias de nombre

Una dependencia de nombre se produce, cuando dos instrucciones usan el mismo registro o la misma posición de memoria, pero no hay flujo de datos entre las instrucciones asociadas a ese nombre.

Hay dos tipos de dependencias de nombre entre una instrucción  $i$  que preceda a una instrucción  $j$  en el orden del programa:

- ① Se produce una **antidependencia** entre la instrucción  $i$  y la  $j$  cuando la instrucción  $j$  escribe un registro o una posición de memoria que lee la instrucción  $i$ .

$$\begin{array}{lll} i : \text{mul.d} & \$f4, & \$f0, \quad \$f10 \\ & & \swarrow \\ j : \text{add.d} & \$f0, & \$f6, \quad \$f8 \end{array}$$

- ② Una **dependencia de salida** se produce cuando la instrucción  $i$  y la instrucción  $j$  escriben en el mismo registro o en la misma posición de memoria.

$$\begin{array}{lll} i : \text{mul.d} & \$f0, & \$f4, \quad \$f10 \\ & & \Downarrow \\ j : \text{add.d} & \$f0, & \$f6, \quad \$f8 \end{array}$$

## Dependencia de nombre

Una dependencia de nombre no es una dependencia verdadera, las instrucciones involucradas en una dependencia de nombre pueden ejecutarse simultáneamente o ser reordenadas.

Si se cambia el nombre desaparece el conflicto **renombrado de registros**. Puede hacerse estáticamente por el compilador o dinámicamente por el hardware.

## Dependencias de control

Una dependencia de control determina el orden de una instrucción con respecto a una instrucción de salto.

Tipo de restricciones:

- ① Una instrucción con dependencia de control con un salto, no se puede mover de forma que su ejecución ya no esté controlada por el salto.

```
if P1{  
    S1;  
}
```

S1 es dependiente de P1, es decir, sólo se ejecuta si se cumple P1.

- ② Una instrucción que no tiene una dependencia de control, no se puede poner de forma que su ejecución sea controlada por el salto.

```
S2  
if P1{  
    S1;  
}
```

S2 no es dependiente de P1, no se puede colocar en la zona de dependencia de P1.

# Dependencias de control

En un procesador segmentado básico, hay dos formas de tratar las dependencias de control:

- Las instrucciones se ejecutan siguiendo el orden del programa.
- Ejecutar instrucciones que no deberían ser ejecutadas aunque se viole el control de dependencias.

*add \$t1, \$t2, \$t3*

*beq \$s0, \$zero, skip*

*sub \$t4, \$t5, \$t6*

*add \$t5, \$t4, \$t7*

*skip : or \$s1, \$s2, \$s3*

Si el valor de  $\$t4$  no es usado después de skip permite mover la instrucción sub a antes del salto.

No se puede violar las dependencias de control en los siguientes casos

- Flujo de datos

*add \$t1, \$t2, \$t3*

*beq \$s0, \$zero, skip*

*sub \$t1, \$t5, \$t6*

*skip :* ...

*or \$t7, \$t1, \$s3*

La instrucción **or** tiene una dependencia de datos con **sub** y **add**.

- Comportamiento de las excepciones

*add \$t1, \$t2, \$t3*

*beq \$s0, \$zero, skip*

*lw \$t7, 0(\$s3)*

*skip :* ...

# Índice

- 1 Objetivos del tema
- 2 Introducción a la segmentación de cauce
- 3 Dependencias y paralelismo a nivel de instrucción
- 4 Riesgos en la ejecución
- 5 Cauce segmentado en el MIPS
- 6 Repertorio de instrucciones para cauces segmentados
- 7 Conclusiones

# Riesgos en la ejecución

- El CPI ideal de un procesador segmentado es uno
- Sin embargo, hay situaciones, denominadas **riesgos (hazards)**, que impiden que se ejecute la siguiente instrucción del flujo de instrucciones durante su ciclo de reloj.

## Tipos de riesgos

- ▶ **Estructurales:** el hardware no puede soportar la combinación de instrucciones que se quieren ejecutar en el mismo ciclo
- ▶ **De control:** surgen del problema de determinar la instrucción correcta que se tiene que ejecutar después de un salto
- ▶ **De datos:** surgen de la existencia de dependencias entre las instrucciones

# Riesgos estructurales

Los recursos deben ser capaces de satisfacer la demanda de uso

- Ejemplo 1: solo existe una unidad de división

div.s \$f10,\$f12,\$f14

div.s \$f16, \$f18,\$f20

- Ejemplo 2: Conflicto de escritura, existe sólo un puerto de escritura en el archivo de registros

Instruction	Clock cycle number										
	1	2	3	4	5	6	7	8	9	10	11
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
...	IF	ID	EX	MEM	WB						
...	IF	ID	EX	MEM	WB						
ADD.D F2,F4,F6		IF	ID	A1	A2	A3	A4	MEM	WB		
...		IF	ID	EX	MEM	WB					
...		IF	ID	EX	MEM	WB					
L.D F2,0(R2)			IF	ID	EX	MEM	WB				

Solución: la replicación

# Riesgos de control

Aparecen cuando surge la necesidad de tomar una decisión basada en los resultados de una instrucción mientras las otras se están ejecutando

## Solución:

- Primera opción: detener el cauce hasta que se toma la decisión
- Segunda opción: aplicar una técnica de predicción
- Tercera opción: aplicar una técnica de salto retardado

# Riesgos de datos

- Se deben a la alteración temporal de la ejecución de las instrucciones que produce la segmentación

## Tipos de riesgos de datos

- ▶ **RAW (read after write)**:  $j$  intenta leer un dato antes de que  $i$  lo escriba. Se corresponde con una dependencia verdadera
- ▶ **WAW (write after write)**:  $j$  intenta escribir un operando antes de que éste sea escrito por  $i$ . Se corresponde con una dependencia de salida
- ▶ **WAR (write after read)**:  $j$  trata de escribir en su destino antes de que este sea leído por  $i$ . Se corresponde con una antidependencia

# Técnicas básicas de tratamiento de los riesgos de datos

- **Software:** el compilador ordena la secuencia de instrucciones de acuerdo a las siguientes reglas:
  - ▶ genera una secuencia de instrucciones independientes, haciendo que los riesgos desaparezcan
  - ▶ inserta instrucciones **nop** cuando no existen instrucciones independientes

*add \$1, \$2, \$3*

*sub \$7, \$1, \$2*

*add \$8, \$2, \$6*

*sub \$10, \$6, \$2*

*add \$1 \$2, \$3*

*add \$8, \$2, \$6*

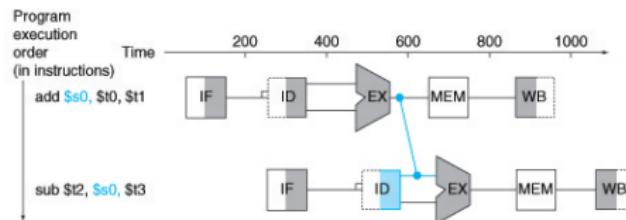
*sub \$10, \$6, \$2*

*sub \$7, \$1, \$2*

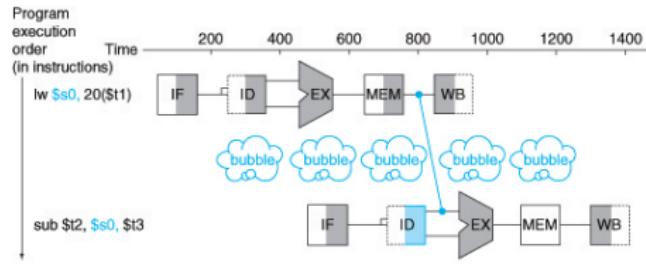
# Técnicas básicas de tratamiento de los riesgos de datos

- **Hardware:**

- ▶ **Anticipación:** el hardware adelanta el dato que hace falta una vez éste está calculado, sin necesidad de esperar a la escritura en el registro



- ▶ **Bloqueo:** el hardware para la ejecución de las instrucciones en el cauce para esperar a que el dato necesario esté disponible

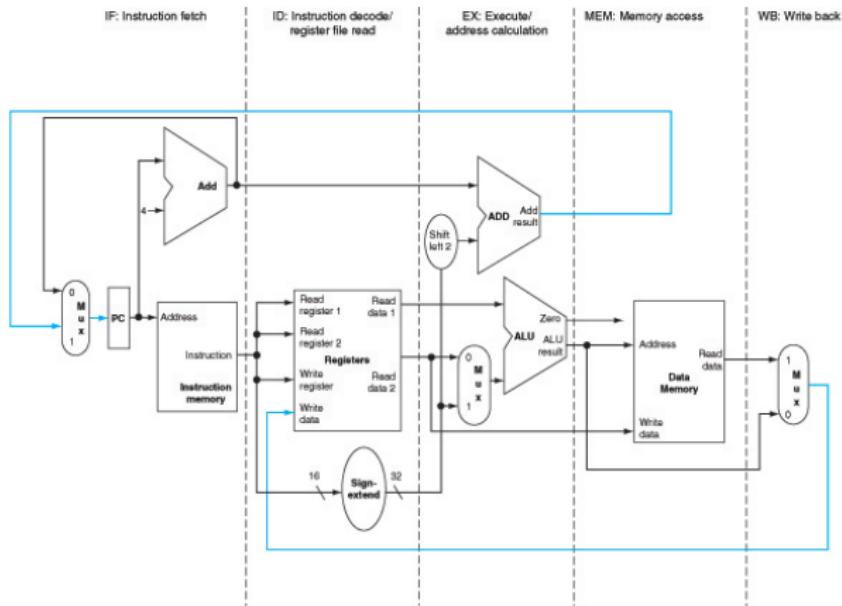


# Índice

- 1 Objetivos del tema
- 2 Introducción a la segmentación de cauce
- 3 Dependencias y paralelismo a nivel de instrucción
- 4 Riesgos en la ejecución
- 5 Cauce segmentado en el MIPS
- 6 Repertorio de instrucciones para cauces segmentados
- 7 Conclusiones

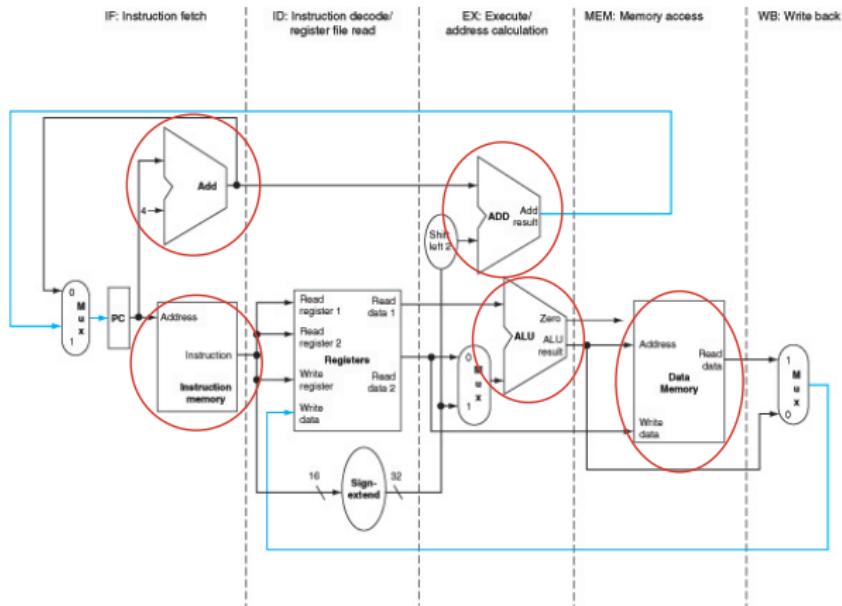
# Camino de datos segmentado básico

- Se necesita hardware adicional (frente al multiciclo) para soportar la ejecución de varias etapas simultáneamente



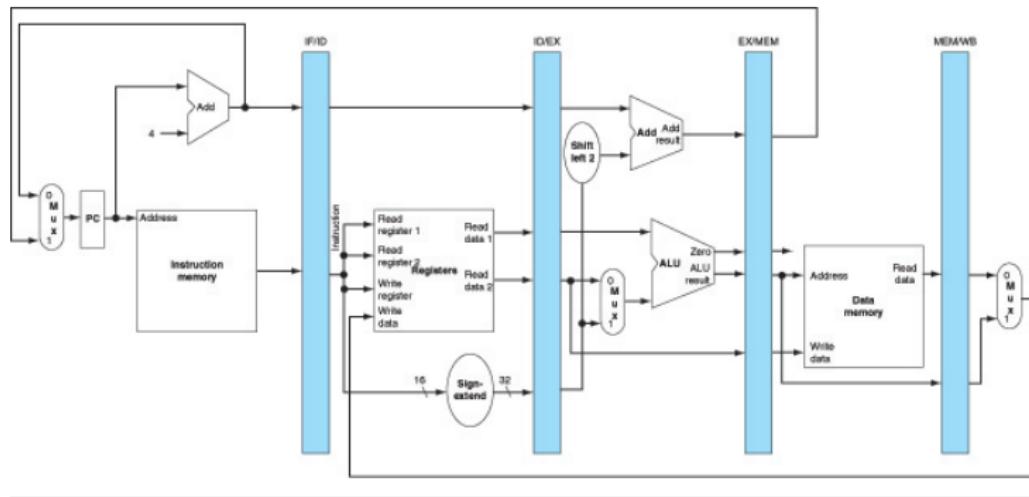
# Camino de datos segmentado básico

- Se necesita hardware adicional (frente al multiciclo) para soportar la ejecución de varias etapas simultáneamente



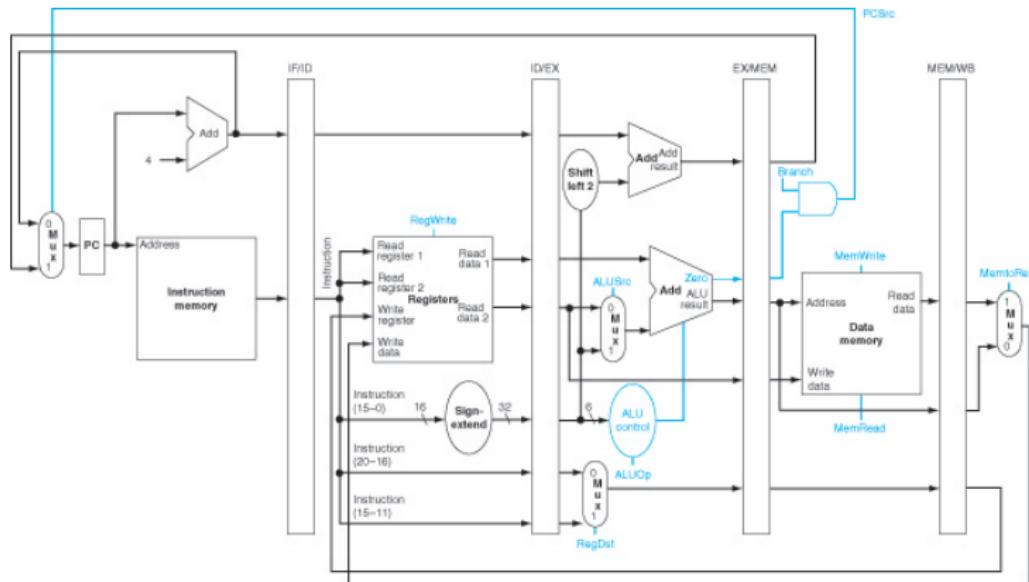
# Camino de datos segmentado básico

- Para conservar los valores de una instrucción para ser usados en etapas siguientes se usan **registros de segmentación**



# Unidad de control

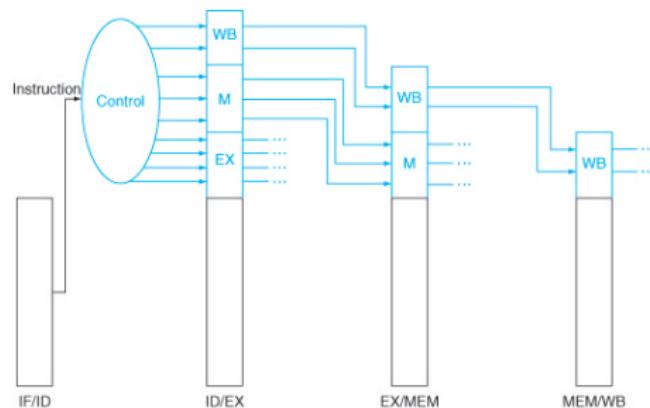
- Camino de datos segmentado con las señales de control identificadas



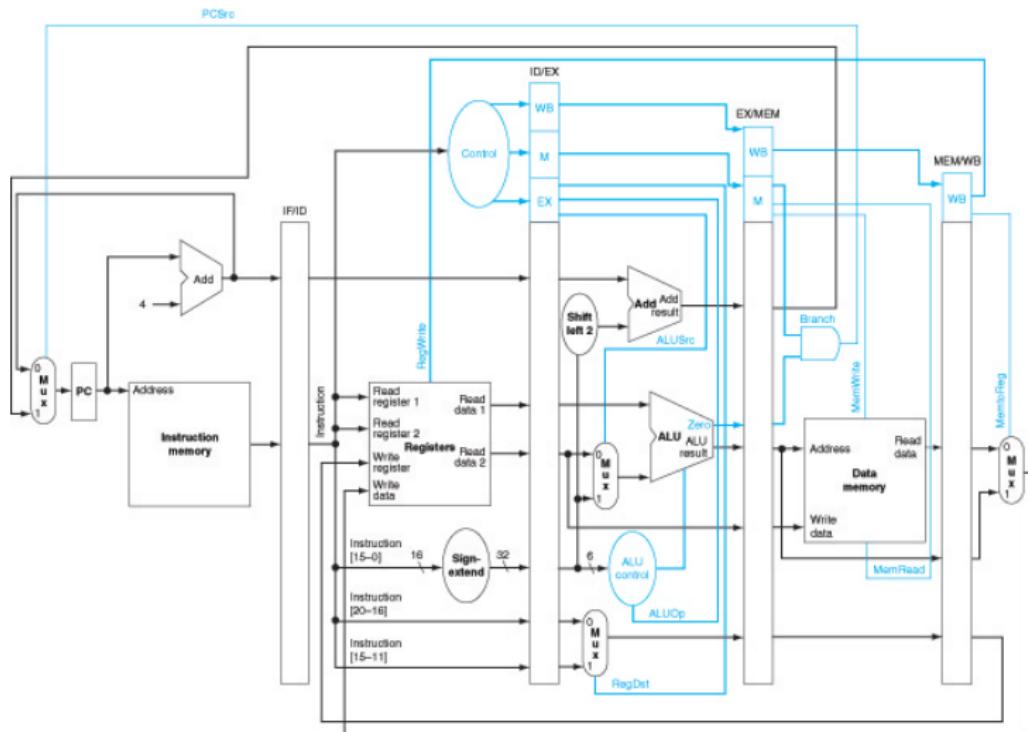
# Unidad de control

Se dividen las líneas de control en grupos

- IF: las señales de control necesarias están siempre activas
- ID: las señales de control necesarias están siempre activas
- EX: **RegDst, ALUOp y ALUSrc**
- MEM: **Brancht , MemRead y MemWrite**
- WB: **MemtoReg y RegWrite**

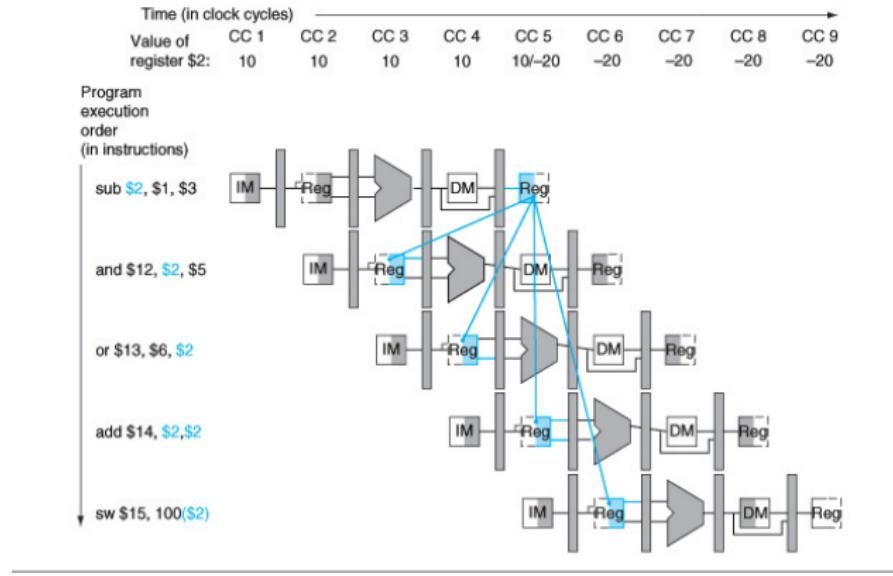


# Unidad de control



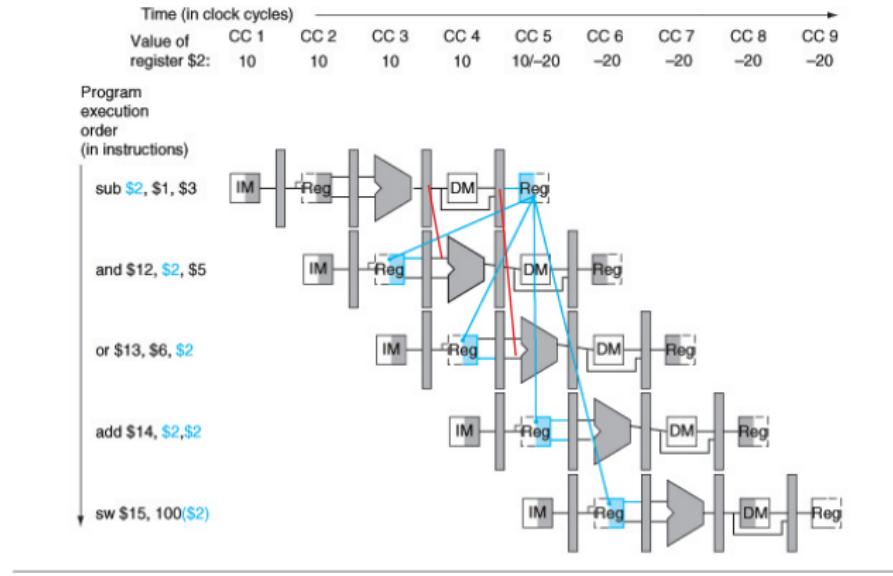
# Anticipación

- Permite obtener las entradas de la ALU de cualquier registro de segmentación, no solo del ID/EX



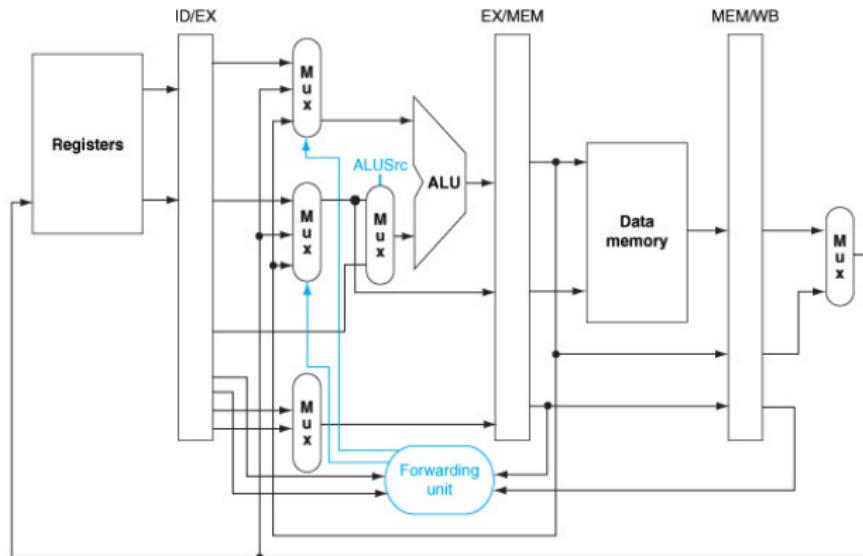
# Anticipación

- Permite obtener las entradas de la ALU de cualquier registro de segmentación, no solo del ID/EX



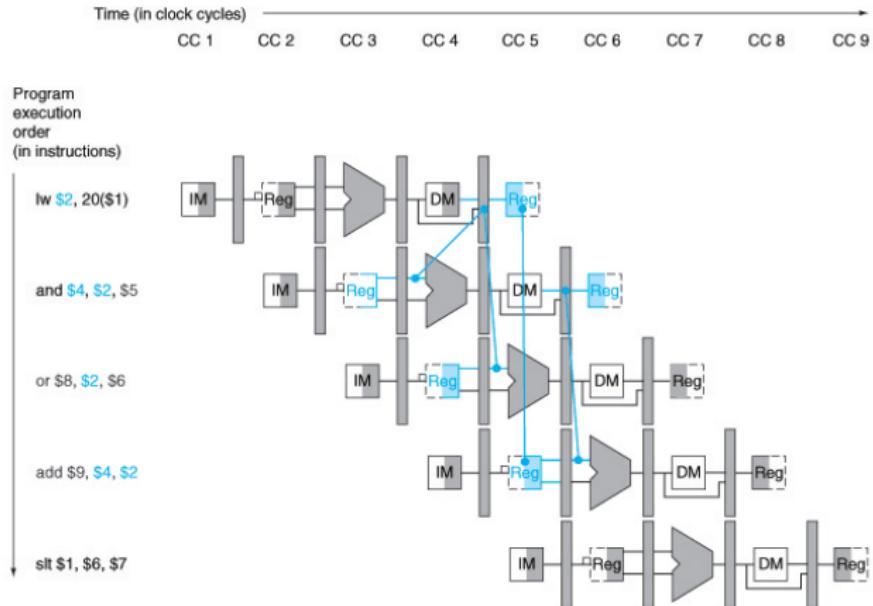
# Anticipación

- Supone añadir multiplexores adicionales a la entrada de la ALU y añadir lógica de control => Unidad de Anticipación



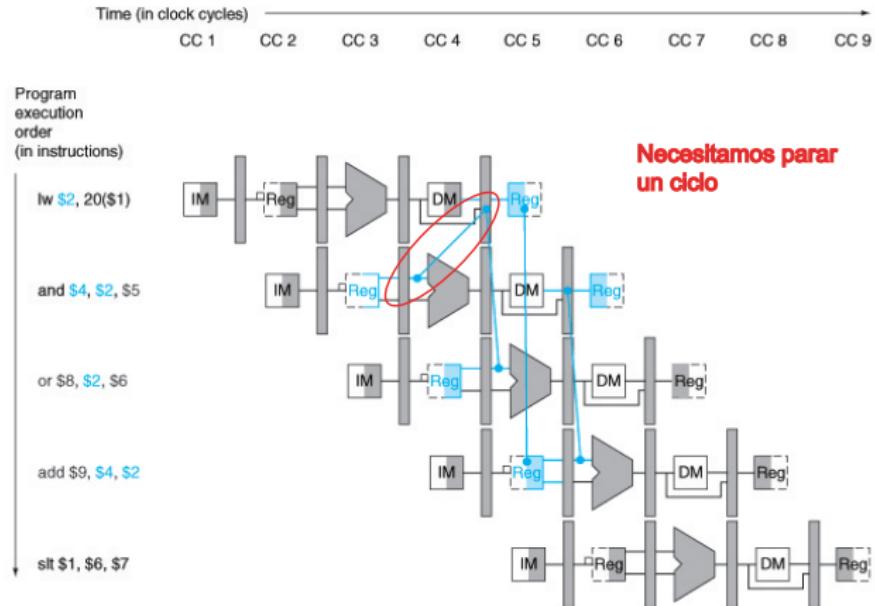
# Bloqueo

- En algunas situaciones la anticipación no evita el riesgo RAW y se hace necesario bloquear el flujo de instrucciones en el cauce



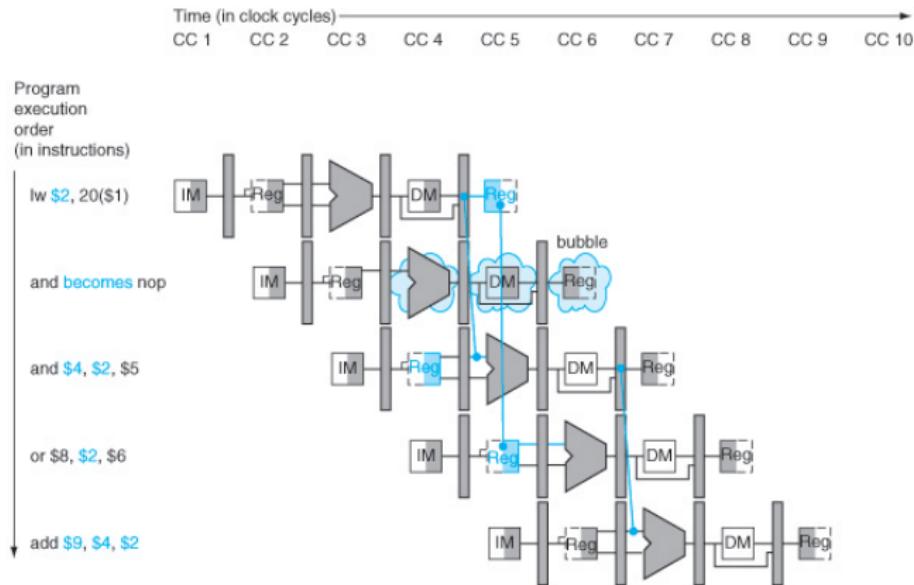
# Bloqueo

- En algunas situaciones la anticipación no evita el riesgo RAW y se hace necesario bloquear el flujo de instrucciones en el cauce

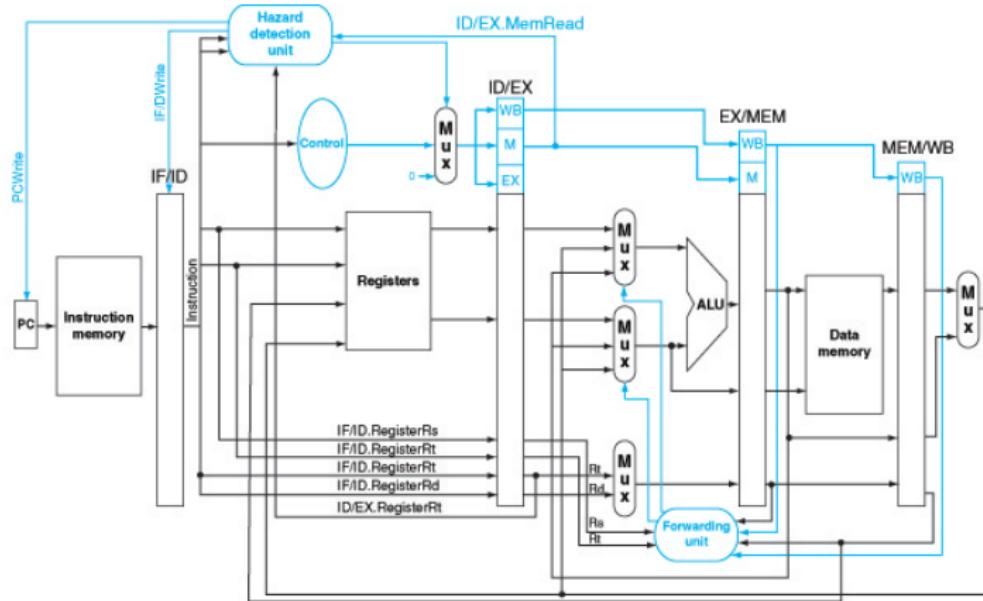


# Bloqueo

- En algunas situaciones la anticipación no evita el riesgo RAW y se hace necesario bloquear el flujo de instrucciones en el cauce



# Bloqueo + Anticipación



# Índice

- 1 Objetivos del tema
- 2 Introducción a la segmentación de cauce
- 3 Dependencias y paralelismo a nivel de instrucción
- 4 Riesgos en la ejecución
- 5 Cauce segmentado en el MIPS
- 6 Repertorio de instrucciones para cauces segmentados
- 7 Conclusiones

# Repertorio de instrucciones para cauces segmentados

- El diseño del repertorio de instrucciones puede afectar negativamente a la segmentación:
  - ▶ Las longitudes de las instrucciones y tiempos de ejecución variables pueden llevar a un desequilibrio entre las etapas de segmentación
  - ▶ Los modos de direccionamiento muy sofisticados generan problemas
- Caso de estudio: repertorio de instrucciones del MIPS
  - ▶ Todas las instrucciones son de la misma longitud: facilita la búsqueda de la instrucción en la primera etapa del cauce
  - ▶ Pocos formatos de instrucciones, muy regulares: facilita la lectura de operando en paralelo a la decodificación de la instrucción
  - ▶ Las operaciones de memoria se limitan a las cargas y almacenamientos: así se puede usar la etapa de ejecución para calcular la dirección de memoria y acceder a memoria en la siguiente etapa
  - ▶ Los operandos y las instrucciones están alineados en memoria: una transferencia entre memoria y procesador sólo necesitará una etapa

# Índice

- 1 Objetivos del tema
- 2 Introducción a la segmentación de cauce
- 3 Dependencias y paralelismo a nivel de instrucción
- 4 Riesgos en la ejecución
- 5 Cauce segmentado en el MIPS
- 6 Repertorio de instrucciones para cauces segmentados
- 7 Conclusiones

# Conclusiones

- El hardware del camino de datos y la lógica de control se complica:
  - ▶ Caminos de anticipación
  - ▶ Lógica para detección de conflictos de datos
  - ▶ Lógica para predicción de saltos
- La segmentación mejora la productividad, pero no el tiempo de ejecución de cada instrucción
- Incrementar la longitud del cauce segmentado no tienen por qué incrementar el rendimiento
  - ▶ Los conflictos de datos hacen que incrementando la longitud se incremente el tiempo que se invierte en cada instrucción
  - ▶ Los conflictos de control provocan que un incremento en la longitud ralentice los saltos
  - ▶ La sobrecarga por los registros de segmentación puede limitar el aumento de la frecuencia de reloj

# Bibliografía

- David A. Patterson y John L. Hennessy. **Computer Organization and Design. The Hardware/Software Interface.** Morgan Kaufmann 2009
- Carl Hamacher. **Organización de Computadores** . McGraw Hill 2002