Arquitectura pipeline

Definición

- Técnica de implementación.
- Consiste en ejecutar las instrucciones traslapadas.
- La siguiente instrucción puede comenzar antes de que la instrucción actual termine de ejecutarse.

Ejemplo

- Considerar la tarea de lavar y secar ropa.
- Si hay más de una carga de ropa, hay dos formas de lavar:
  1. Versión secuencial: lavar y secar una carga de ropa a la vez.
  2. Versión con pipeline: separar la tarea en pasos y hacer distintos pasos en paralelo.

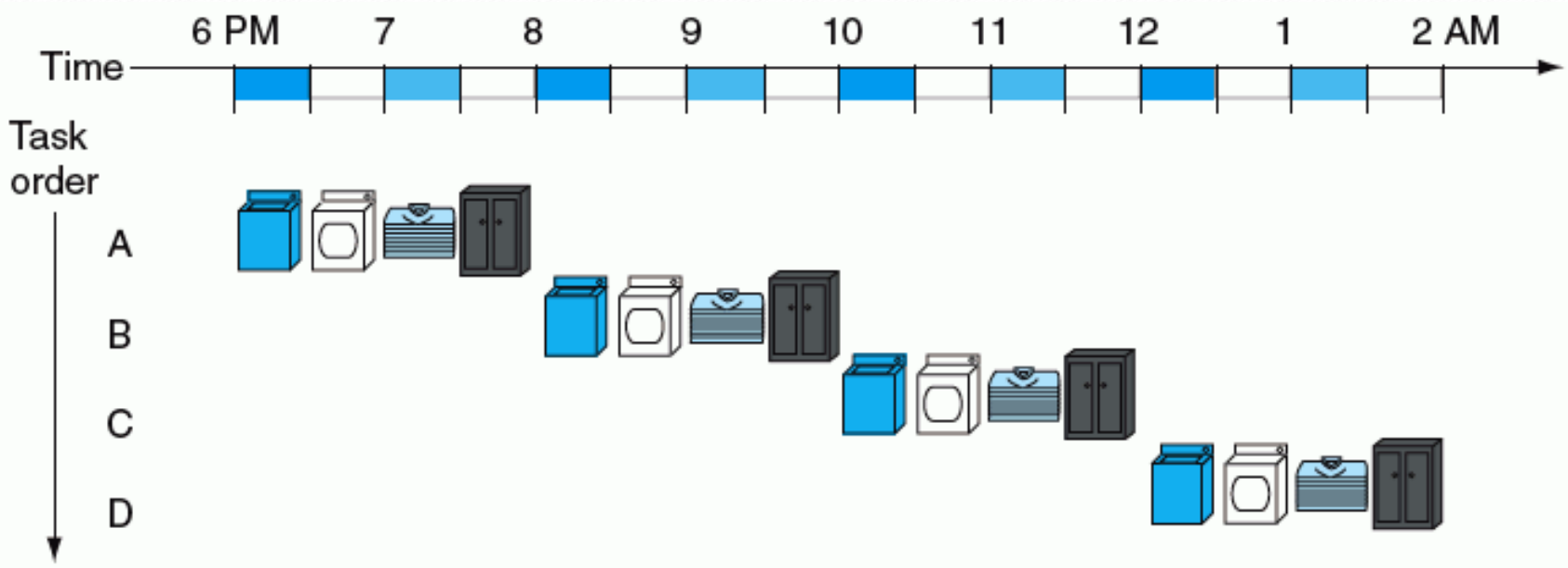
Ejemplo

- Pasos para lavar y secar una carga de ropa:

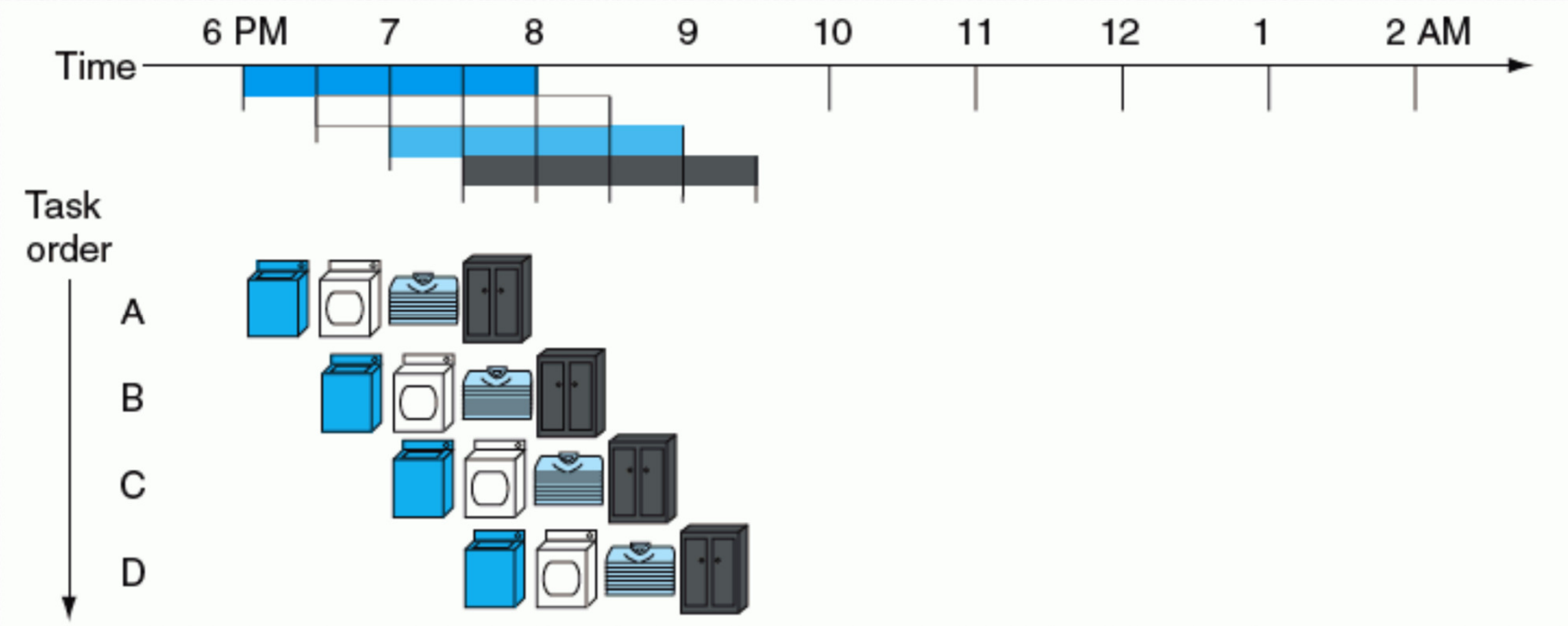
1. Poner una carga de ropa sucia en la lavadora.
2. Al terminar, poner la ropa húmeda en la secadora.
3. Al terminar, poner la ropa seca en una mesa y doblarla.
4. Al terminar, guardar la ropa doblada.

- Suponer que hay cuatro cargas de ropa y que cada paso tarda media hora (30 minutos).

Versión secuencial



Versión con pipeline



Conclusión

- Para 1 carga de ropa:
  - Versión secuencial: 2 horas.
  - Versión con pipeline: 2 horas.
  - Speedup: 2 / 2 = 1.0.
- Para 4 cargas de ropa:
  - Versión secuencial: 8 horas.
  - Versión con pipeline: 3.5 horas.
  - Speed-up: 8 / 3.5 = 2.29.

Conclusión

- Para 20 cargas de ropa:
  - Versión secuencial: 40 horas.
  - Versión con pipeline: 11.5 horas.
  - Speed-up: 40 / 11.5 = 3.48.

Conclusión

- Conclusiones:

a) El speed-up está limitado por el número de etapas del pipeline.

b) El speed-up depende del factor de utilización del pipeline.

c) Para tener un pipeline se necesitan recursos para cada etapa.

Pipelining en MIPS

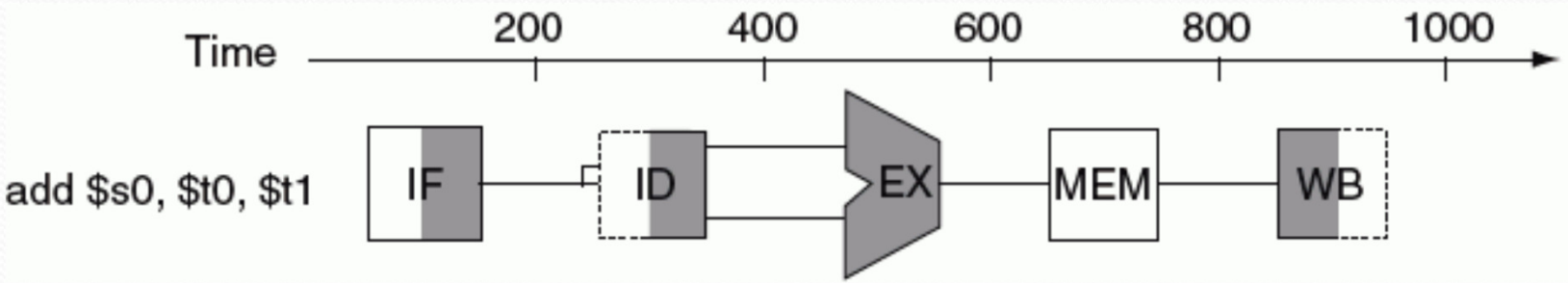
- Las instrucciones de MIPS tienen 5 etapas:

1. Obtener (fetch) una instrucción de la memoria.
2. Leer los registros mientras se decodifica la instrucción.
3. Ejecutar la operación o calcular una dirección.
4. Accesar un operando en la memoria de datos.
5. Escribir el resultado en un registro.

Pipelining en MIPS

- IF (instruction fetch): obtiene la instrucción de la memoria.
- ID (instruction decoding): decodifica la instrucción y lee los operandos.
- EX (execute): ejecuta la instrucción, o si es lw/sw calcula la dirección de la memoria.
- MEM (memory access): lee/escribe la memoria. Es una no-op para instrucciones tipo-R.
- WB (write back): escribe el resultado en el registro.

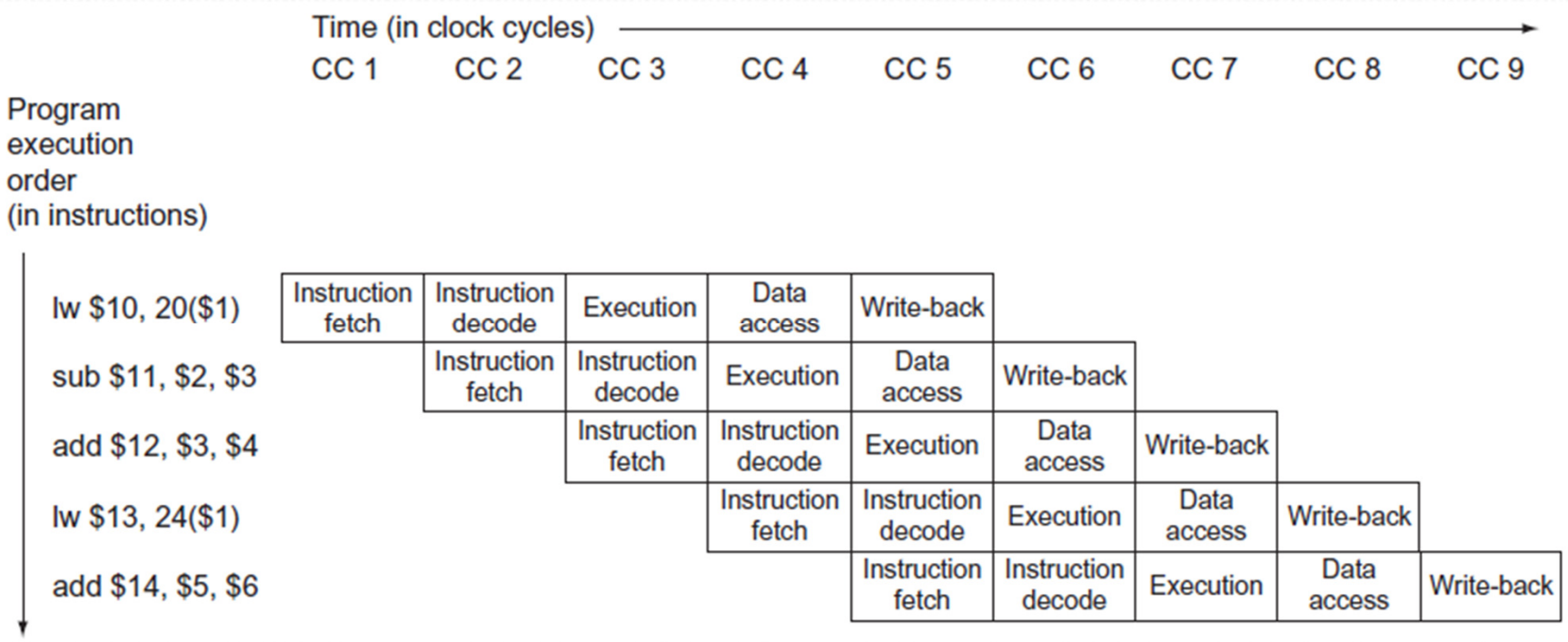
Diagrama de pipeline



- Cada etapa se hace en un ciclo.
- El cuadro completo representa la memoria de instrucciones.
- El cuadro punteado representa el banco de registros.
- EX tiene el símbolo de la ALU.
- MEM está en blanco porque *add* no usa la memoria de datos.

Diagrama de pipeline

- Idealmente en cada ciclo entra una instrucción.



Fuente: COD:HSI, p. 299.

Diagrama de tabla

- Para cada instrucción se indica el ciclo de entrada y el de salida.

Instrucción	Entra	Sale
lw $10, 20($1)	1	5
sub $11, $2, $3	2	6
add $12, $3, $4	3	7
lw $13, 24($1)	4	8
add $14, $5, $6	5	9

Comparación

- Comparar la implementación MIPS de un ciclo contra la implementación de un pipeline.
- Se van a comparar una secuencia de 3 instrucciones lw.
- Versión de un ciclo: todas las instrucciones tardan un ciclo de reloj.
- Versión con pipeline: cada etapa tarda un ciclo.

Comparación

- Suponer los siguientes tiempos:
  - 200 ps por acceso a memoria.
  - 200 ps por operación de la ALU.
  - 100 ps por leer o escribir en el banco de registros.

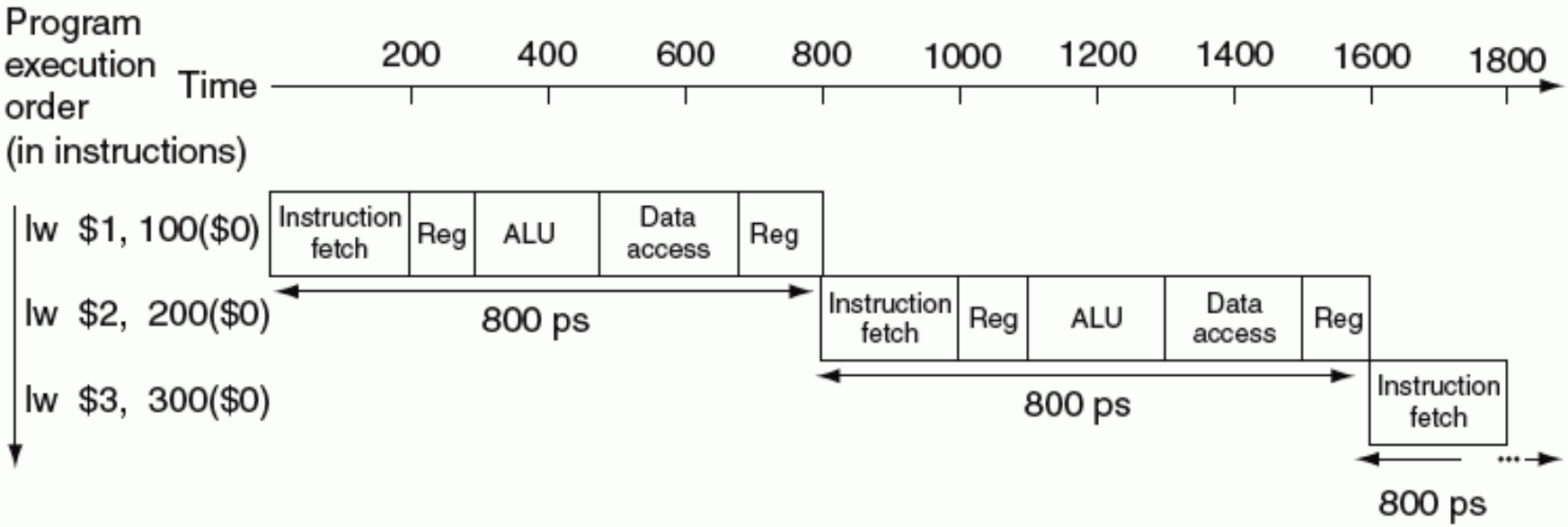
Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add`, `sub`, `and`, `or`, `slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps

Comparación

- En la versión de un ciclo, la duración del ciclo de reloj se acopla a la instrucción mas lenta.
- El periodo de reloj es de 800 ps, aunque haya instrucciones que tarden 500 ps.

Versión de un ciclo

- Las 3 instrucciones tardan 800 x 3 = 2,400 ps.

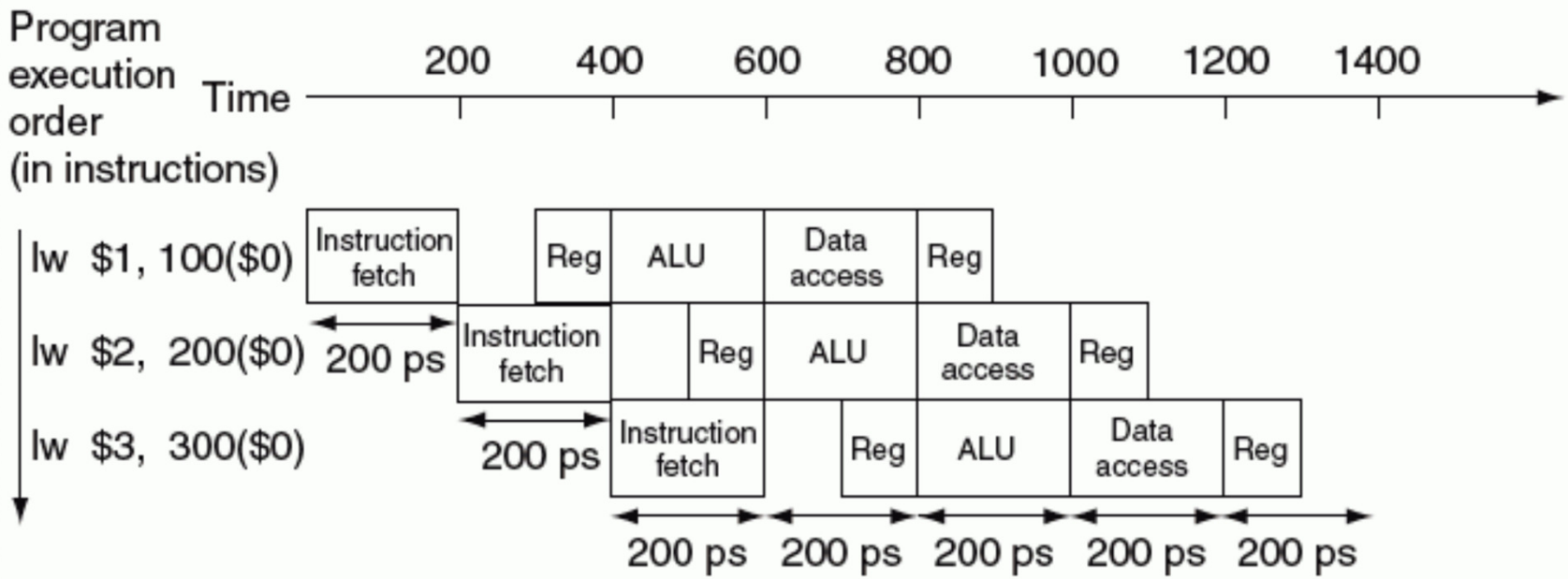


Versión con pipeline

- Cada etapa se realiza en un ciclo.
- La duración del ciclo de reloj se acopla a la etapa más lenta.
- El periodo de reloj es de 200 ps, aunque haya etapas que tarden 100 ps.

Versión con pipeline

- Las 3 instrucciones tardan 1,400 ps.



Conclusiones

- Speedup = 2400 / 1400 = 1.71.
- La versión con pipeline es 1.71 veces más rápida que la versión secuencial de un ciclo.

Speedup

- El speedup depende de la utilización del pipeline.
- En el ejemplo, con 3 instrucciones el speedup es de 1.71.
- Si se ejecutan un millón de instrucciones:

$$\text{Speedup} = \frac{800{,}000{,}000}{200{,}000,\ 800} = 3.99998$$

Speedup

- El speedup máximo que se obtiene al usar un pipeline es:

$$\text{Speedup}_{\max} = \frac{\text{Ciclo}_{\text{secuencial}}}{\text{Ciclo}_{\text{pipeline}}}$$

- En el ejemplo:

$$\text{Speedup}_{\max} = \frac{800}{200} = 4$$

Speedup

- **Pipeline balanceado**. Es un pipeline donde todas las etapas duran lo mismo.
- En un pipeline balanceado el speedup máximo es:

$$\text{Speedup}_{\max} = \text{Num. de etapas}$$

- Un pipeline de 5 etapas puede ser hasta 5 veces más rápido que la versión sin pipeline (secuencial).
- El pipeline del ejemplo es no balanceado (unas etapas duran 100 ps y otras 200 ps).
- Por eso el speedup está limitado a 4.

Conclusión

- El uso de un pipeline mejora el rendimiento incrementando el throughput sin decrementar el tiempo de ejecución de una instrucción individual.

Peligros (hazards)

- Eventos que evitan que la siguiente instrucción se ejecute en el siguiente ciclo de reloj.
- Hay 3 tipos de peligros en un pipeline.

1. Estructurales.
   - Motivo: conflictos de recursos.
   - Solución: duplicación de recursos.
2. Datos.
   - Motivo: dependencias entre instrucciones.
   - Soluciones: detener (stall) el pipeline, bypassing (forwarding) y reordenamiento de instrucciones.

Peligros (hazards)

3. Control.
   - Motivo: en un brinco no se conoce la siguiente instrucción hasta que la instrucción de brinco sale del pipeline.
   - Soluciones: detener (stall) el pipeline, especular (adivinar) la siguiente instrucción y decisión retrasada (delayed decision).

Peligros estructurales

- Motivo: conflicto de recursos.
- Solución: duplicar recursos.
- Ejemplo: IF y MEM necesitan acceso a la memoria.
- Solución: separar la memoria de instrucciones y la memoria de datos.
- Ejemplo: IF e ID necesitan acceso al PC.
- Solución: tener 2 copias del PC. Un PC apunta a la siguiente instrucción y otro a la instrucción que está siendo ejecutada.

Peligros de datos

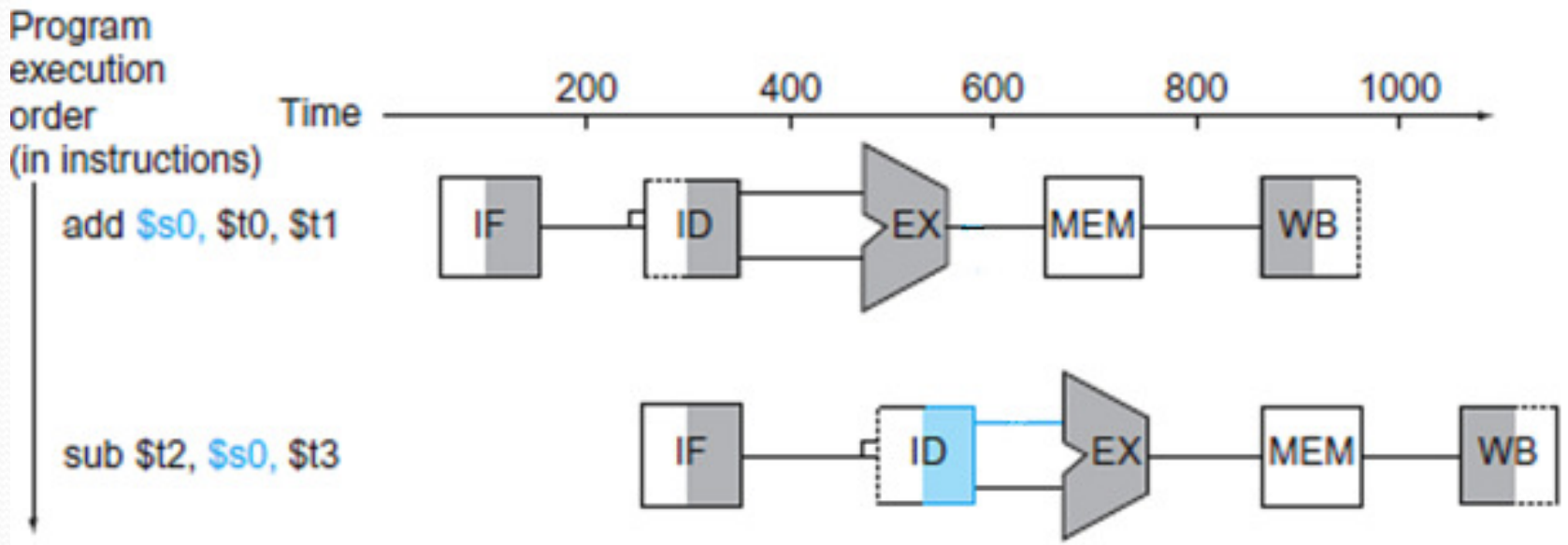
- Motivo: Hay una dependencia entre dos instrucciones A y B.
- Consecuencia: B no puede entrar al pipeline al siguiente ciclo en que A entró.
- Conocido como peligro RAW (read-after-write).
- Ejemplo:

```
A: add $s0, $t0, $t1      # s0 = t0 + t1
B: sub $t2, $s0, $t3      # t2 = s0 – t3
```

- A esta dependencia de datos se le llama **dependencia de datos verdadera**.

Peligros de datos

- B necesita el valor de $s0 en la etapa EX.
- A escribe $s0 en la etapa WB.



Peligros de datos

- Soluciones:

1. Detener (stall) el pipeline.
2. Agregar un bypass.
3. Reordenar las instrucciones.

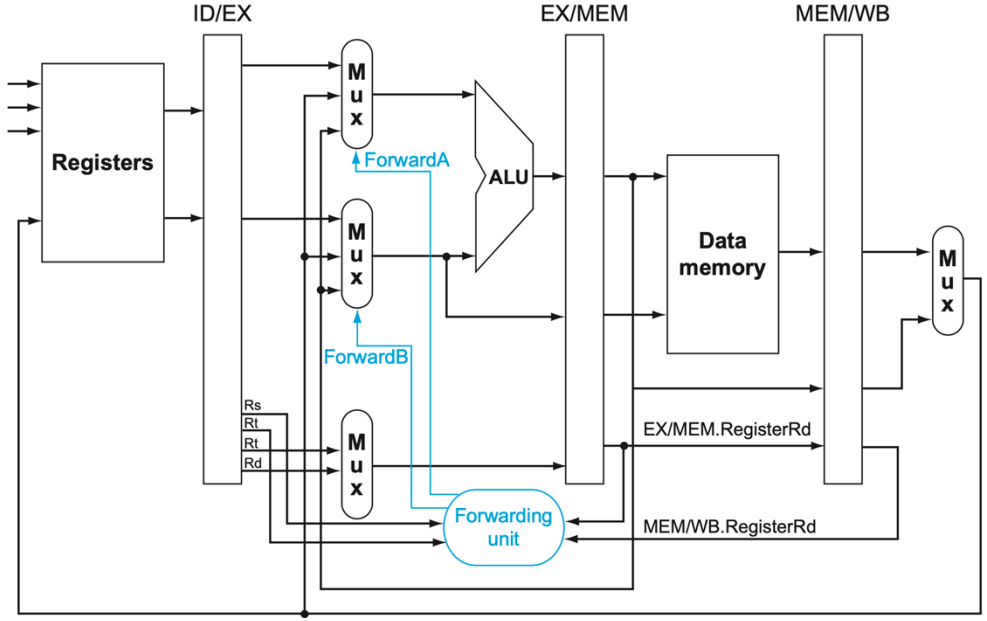
Peligros de datos

1. Detener (stall) el pipeline. La segunda instrucción no entra al pipeline al siguiente ciclo. Se pierden dos ciclos.

	200	400	600	800	1000	1200	1400	1600
add $s0, $t0, $t1	IF	ID	EX	M	WB			
nop		IF	ID	EX	M	WB		
nop			IF	ID	EX	M	WB	
sub $t2, $s0, $t3				IF	ID	EX	M	WB

Peligros de datos

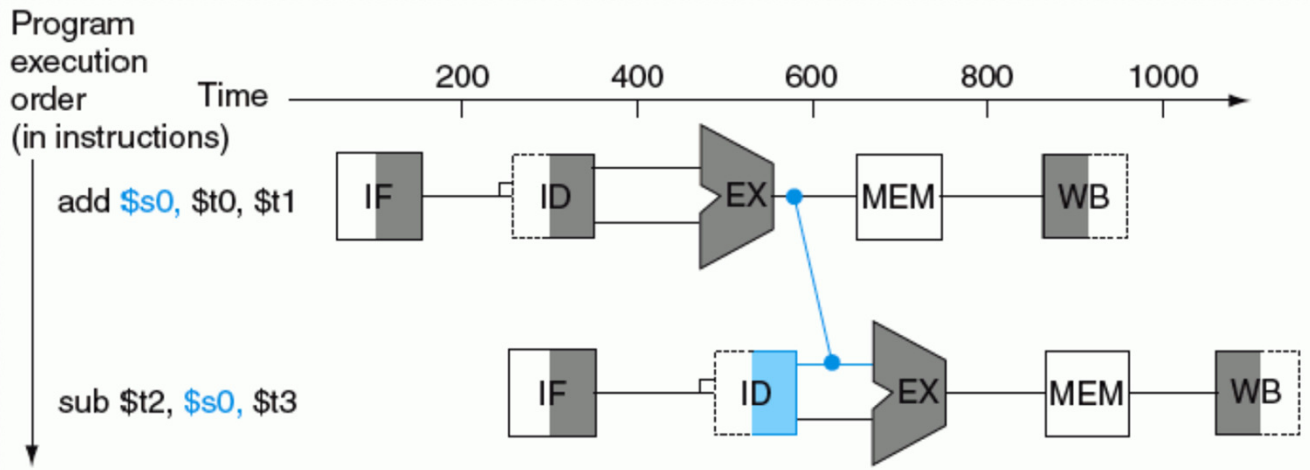
2. Bypassing (forwarding). Conectar la salida de las etapas EX y MEM con la entrada de la ALU.



- Figura 4.54b p. 309 (COD 5)

Peligros de datos

- Al dibujar el pipeline, el bypass se ve como una flecha hacia adelante en el tiempo.



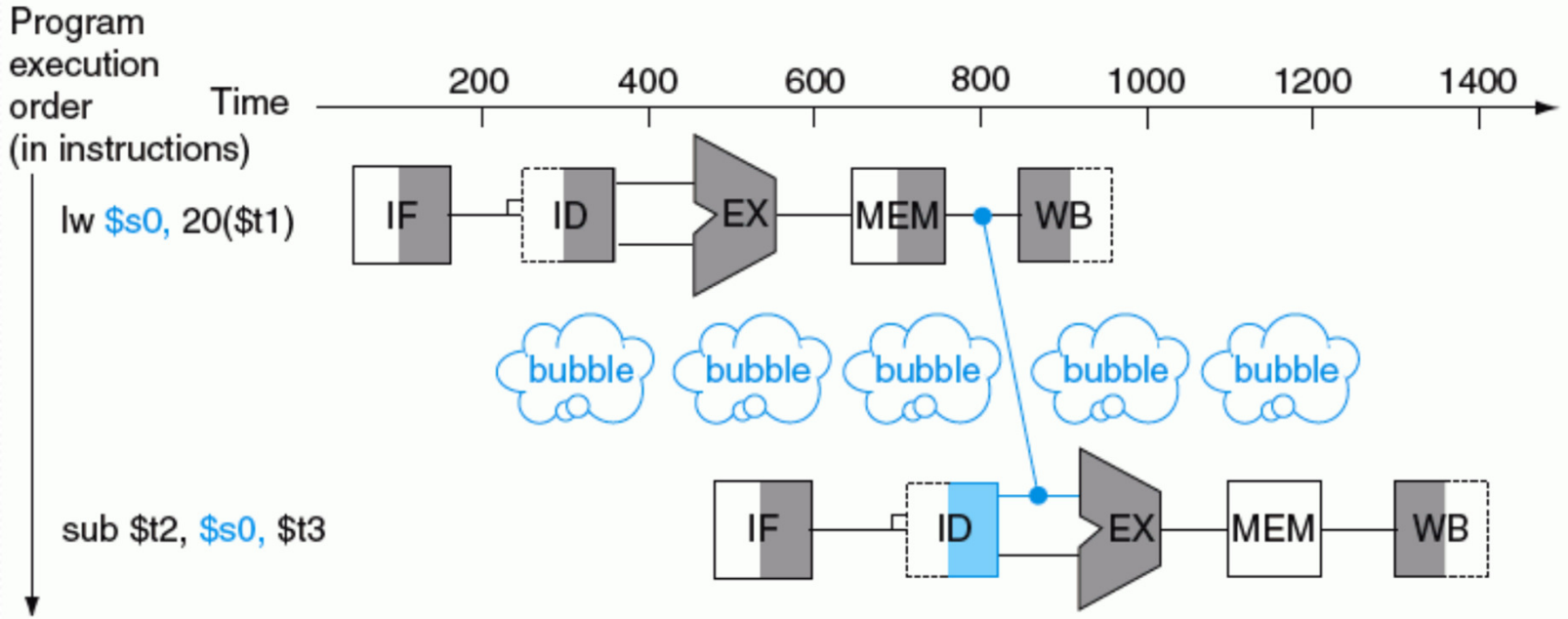
Peligros de datos

- El bypass no soluciona todos los peligros RAW.
- Ejemplo:

```
A: lw $s0, 20($t1)        # s0 = Mem[t1 + 20]
B: sub $t2, $s0, $t3      # t2 = s0 – t3
```

- $s0 se calcula en la etapa MEM de A.
- Ya es tarde para hacer un bypass a la etapa EX de B.
- El pipeline se detiene (stall) un ciclo.

Peligros de datos



- Bypassing no resuelve los peligros RAW cuando una instrucción tiene una dependencia con una instrucción lw.

Peligros de datos

3. Reordenar instrucciones.

- El sentido del programa no debe cambiar.
- Ejemplo:

```
A: lw $s0, 20($t1)      # s0 = Mem[t1 + 20]
B: sub $t2, $s0, $t3    # t2 = s0 – t3
```

- Solución:

```
A: lw $s0, 20($t1)      # s0 = Mem[t1 + 20]
C: lw $t4, 8($a0)       # t4 = Mem[a0 + 8]
B: sub $t2, $s0, $t3    # t2 = s0 – t3
```

Ejemplo

- Encontrar las dependencias en el siguiente código y reordenar las instrucciones para evitarlas.
- Código en C/Java

$$A = B + E$$

$$C = B + F$$

Ejemplo

- Código MIPS
- Asumiendo que B está apuntada por $t0, E por $t0+4, F por $t0+8, A por $t0+12 y C por $t0+16.

```
1. lw $t1, 0($t0)        # t1 = Mem[t0]
2. lw $t2, 4($t0)        # t2 = Mem[t0+4]
3. add $t3, $t1, $t2     # t3 = t1 + t2
4. sw $t3, 12($t0)       # Mem[t0+12] = t3
5. lw $t4, 8($t0)        # t4 = Mem[t0+8]
6. add $t5, $t1, $t4     # t5 = t1 + t4
7. sw $t5, 16($t0)       # Mem[t0+16] = t5
```

Ejemplo

- Dependencias de datos:

a) 1 y 3.
b) 2 y 3.
c) 3 y 4.
d) 5 y 6
e) 6 y 7

- Forwarding (bypassing) elimina todas las dependencias excepto b) y d).

Ejemplo

- Las dependencias b) y d) se resuelven moviendo la instrucción 5 hacia arriba:

```
1. lw $t1, 0($t0)        # t1 = Mem[t0]
2. lw $t2, 4($t0)        # t2 = Mem[t0+4]
5. lw $t4, 8($t0)        # t4 = Mem[t0+8]
3. add $t3, $t1, $t2     # t3 = t1 + t2
4. sw $t3, 12($t0)       # Mem[t0+12] = t3
6. add $t5, $t1, $t4     # t5 = t1 + t4
7. sw $t5, 16($t0)       # Mem[t0+16] = t5
```

Peligros de control

- También llamados peligros de brincos.
- No se sabe que instrucción sigue a un brinco.
- Ejemplo:

```
        beq $t0, $t1, etiqueta
        instrucción_1
        …
    etiqueta:
        instrucción_2
```

Peligros de control

- Tres soluciones típicas:

1. Detener (stall) el pipeline. La siguiente instrucción entra cuando se sepa el resultado del brinco.
2. Especular si se va a brincar o no.
3. Decisión retrasada.

Costo de detener el pipeline

- Según SPECint2006:
  - El 17% de las instrucciones son brincos.
  - El CPI de las otras instrucciones es 1.
- El resultado del brinco se conoce al final de la etapa EX. El pipeline se tendría que detener dos ciclos por cada brinco. El CPI se incrementaría a 1.34, es decir, la CPU sería 34% comparada con otra CPU que no se detenga después de cada brinco.

Costo de detener el pipeline

- Incluso si hubiera hardware extra para obtener el resultado al final de la etapa ID, se perdería un ciclo por cada brinco y el CPI se incrementaría a 1.17, es decir, 17% más lenta que una CPU que no se detenga después de cada brinco.
- Conclusión: detener el pipeline no es práctico.

Predicción de brincos

- Intentar adivinar el resultado de un brinco.
- Si se acierta, el pipeline continúa a toda velocidad.
- Si no se acierta, hay que deshacer la instrucciones del camino equivocado y empezar a sacar instrucciones del camino correcto.

Predicción de brincos

Fuente: COD 5, p. 283

Predicción de brincos

- Algunas estrategias básicas:

1. Predecir que los brincos nunca se toman.
2. Usar alguna heurística. Por ejemplo, predecir que los brincos hacia arriba son parte de un ciclo y predecir que se toman.
3. Predecir cada brinco en base a la historia del brinco.

Decisión retrasada

- Se usa en MIPS.
- Poner una instrucción independiente al brinco, después del brinco.
- Por ejemplo:

```
add $t0, $t1, $t2
beq $s0, $s1, Etiqueta
```

- Se convierte en:

```
beq $s0, $s1, Etiqueta
add $t0, $t1, $t2
```

Decisión retrasada

- La inclusión del add da tiempo a que la CPU sepa el resultado del beq.

Más información

- Smith, J. E. **A Study of Branch Prediction Techniques**. IEEE (1981)

  http://www.mat.uson.mx/~havillam/ca/Common/JSmith.pdf
- Michaud, P., Seznec, André. **A Comprehensive Study of Dynamic Global History Branch Prediction**. INRIA (2001)

  http://www.mat.uson.mx/~havillam/ca/Common/RR-4219.pdf

Resumen pipelining

- Incrementa el número de instrucciones que se ejecutan simultáneamente.
- Incrementa la tasa (rate) a la cual las instrucciones comienzan y son ejecutadas.
- No reduce la latencia, el tiempo que de ejecución de una instrucción individual.
- Mejora el throughput, la cantidad de trabajo hecha por unidad de tiempo.

Resumen pipelining

- Existen peligros estructurales, de datos y de control.
- Detener el pipeline, bypassing y la especulación de brincos ayudan a resolver los peligros.