Pipelining

Peligros (hazards)

Peligros (hazards)

- Eventos que evitan que la siguiente instrucción se ejecute en el siguiente ciclo de reloj.

★ Estructurales.
  - Motivo: conflictos de recursos.
  - Solución: duplicación de recursos.

★ Datos.
  - Motivo: dependencias entre instrucciones.
  - Soluciones: detener (stall) el pipeline, forwarding (bypassing) y reordenamiento de instrucciones.

Peligros (hazards)

- Control (brincos).
  - 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

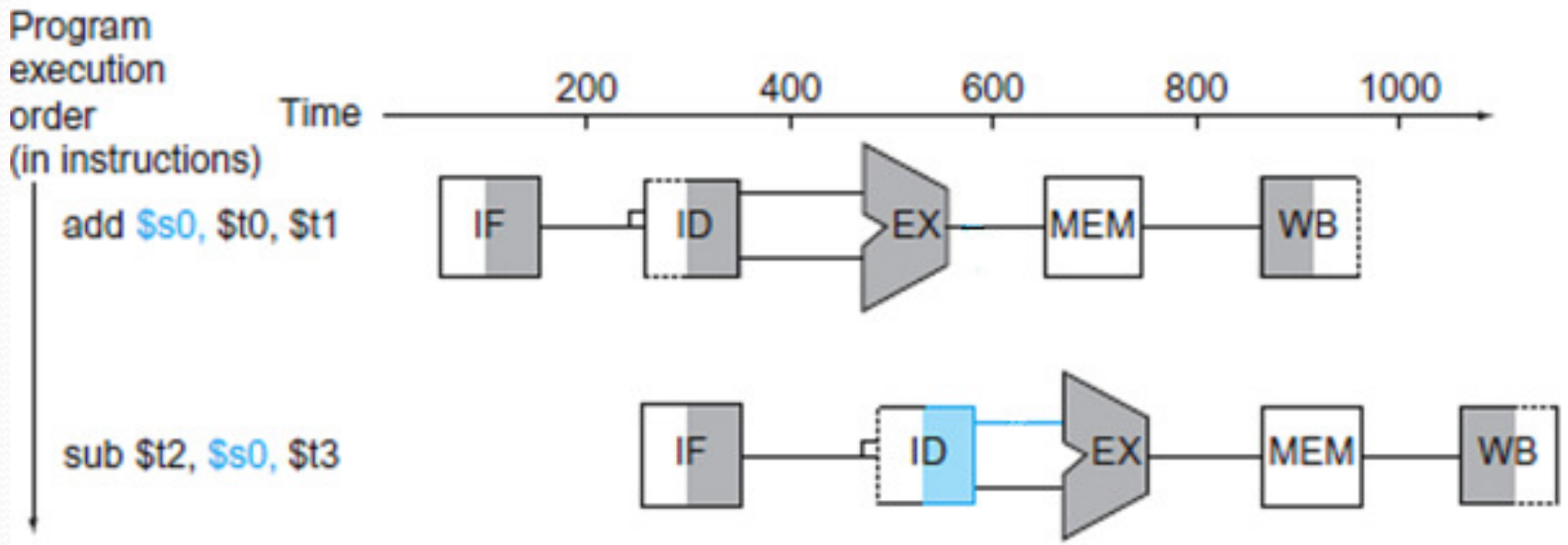
- Motivo: Hay una dependencia entre dos instrucciones *A* y *B*.
- Consecuencia: *B* no puede entrar al pipeline al siguiente ciclo en que *A* entró.
- Conocido como peligro RAW (read-after-write).
- Ejemplo:

```
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. Diseñar un bypass.
3. Reordenar las instrucciones.

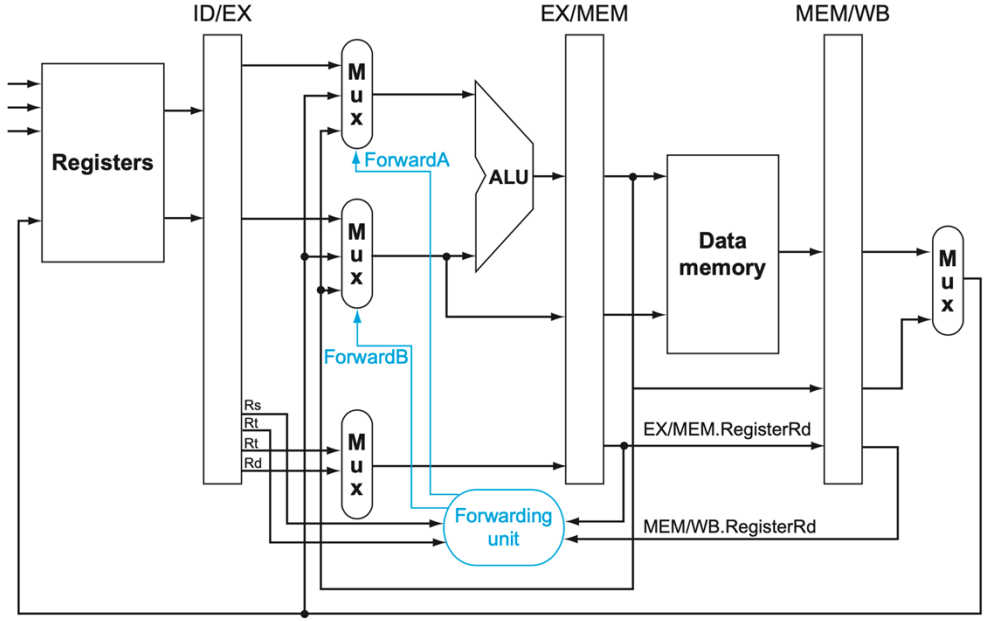
Peligros de datos

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

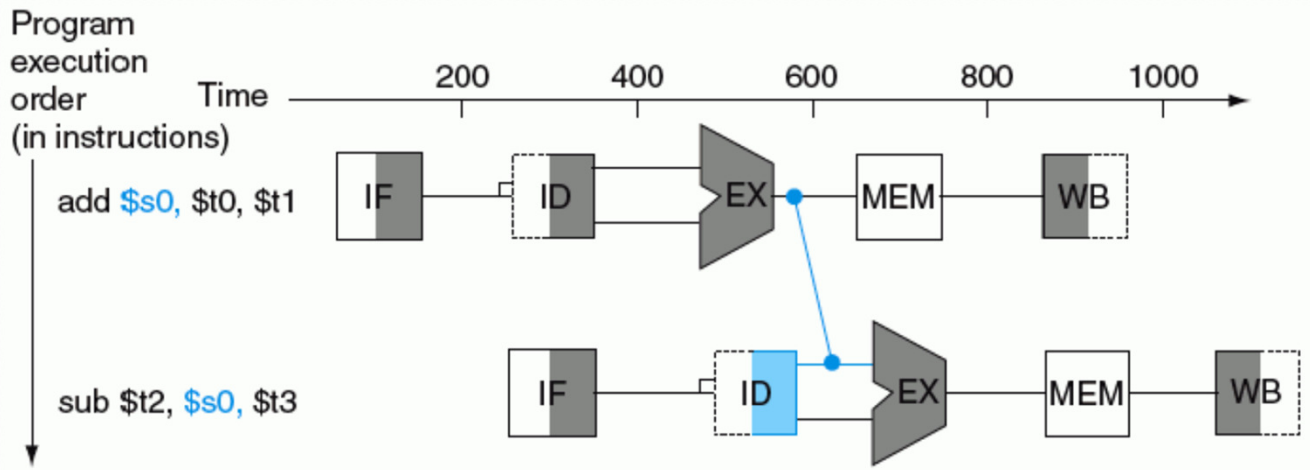
2. Bypassing (forwarding). Conectar la salida de las etapas EX y MEM con la entrada de la ALU.



- 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.



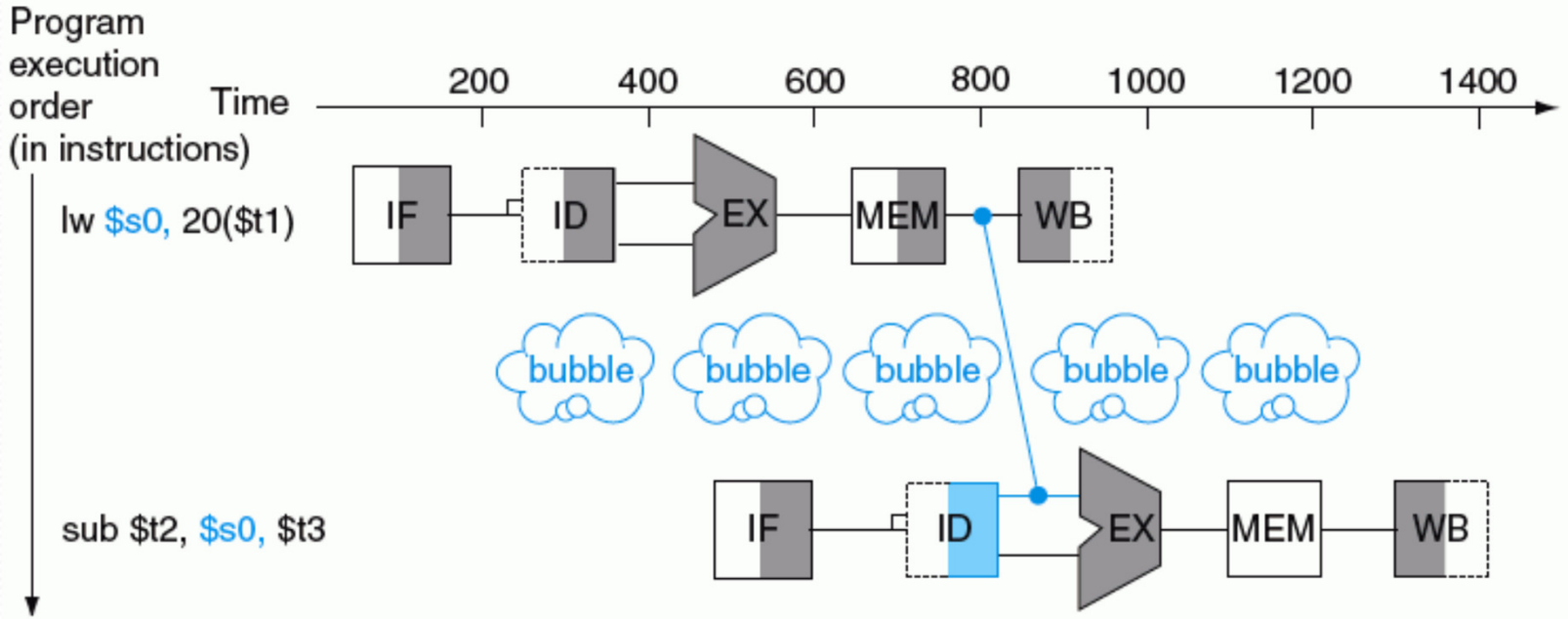
Peligros de datos

- 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 sigue a 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. Predecir (adivinar) 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.

Peligros de control

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

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

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

Resumen

Pipelining:

- Explota el paralelismo entre instrucciones en un flujo secuencial.
- 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.

Resumen

- Mejora el throughput, la cantidad de trabajo hecha por unidad de tiempo.
- Existen peligros (hazards) estructurales, de datos y de control.
- Parar (stall) el pipeline, predicción de brincos y bypassing (forwarding) ayudan a resolver los peligros.