Apunte TP 5:

Segmentación

UNPSJB – Sede Puerto Madryn

Prof. Lic. Paula Kölln

Apunte TP 5: Segmentación

Contenido

Práctico 5: Segmentación

Objetivo General

Comprender cómo la segmentación (pipelining) mejora el rendimiento de los procesadores RISC, identificar los problemas que surgen al segmentar (dependencias, atascos, predicción de saltos), y aplicar soluciones prácticas (adelantamiento, reordenamiento, predicción) a través del análisis teórico y la simulación.

Objetivo del Práctico

- Comprender qué es la segmentación (pipelining) y cómo mejora el rendimiento de un procesador RISC.
- Identificar y clasificar los distintos tipos de riesgos: de datos, control y estructurales.
- Aplicar técnicas de resolución: adelantamiento (forwarding), inserción de burbujas (stalls), predicción de saltos.
- Analizar secuencias de instrucciones y optimizar su ejecución.
- Utilizar simuladores (como RARS + BHT) para visualizar y cuantificar estos efectos.

1. Segmentación (Pipelining)

¿Qué es?

La **segmentación** divide la ejecución de una instrucción en varias etapas que se ejecutan en paralelo con partes de otras instrucciones. El objetivo es mejorar el **rendimiento** del procesador al aprovechar mejor los recursos internos.

Etapas Clásicas del Pipeline (RISC de 5 etapas)

Etapa	Abrev.	Descripción
Instruction Fetch	IF	Se busca la instrucción en memoria.
Instruction Decode	ID	Se decodifica y se accede a registros.
Execute	EX	ALU realiza la operación o dirección efectiva.
Memory Access	MEM	Se accede a la memoria de datos.
Write Back	WB	Se escribe el resultado en registros.

Cada etapa dura aproximadamente el mismo tiempo (≈2 ns en este práctico), por lo tanto, el **ciclo del pipeline** está determinado por la etapa más lenta.

2. Comparación: Procesador Uniciclo vs Segmentado

Procesador Uniciclo

- Ejecuta **una instrucción por ciclo completo** (todas las etapas en un solo ciclo largo).
- **Tiempo por instrucción:** suma de todos los tiempos (≈10 ns).
- Poco eficiente: no aprovecha la superposición de instrucciones.

Apunte TP 5: Segmentación

Procesador Segmentado

- Divide la ejecución en etapas que se superponen entre instrucciones.
- Cada instrucción avanza una etapa por ciclo (ciclo de 2 ns).
- **A partir del ciclo 5**, se completa **una instrucción por ciclo** (idealmente pipeline lleno).

Ejemplo

Con instrucciones: lw, sw, add.
Uniciclo:
3 instrucciones × 10 ns = **30 ns**
Segmentado:
5 ciclos para completar la 1ª instrucción

- 1 ciclo por cada instrucción adicional → Total: 7 ciclos × 2 ns = **14 ns**

Modo	Tiempo por instrucción	Características
Uniciclo	5 × 2 ns = 10 ns	Ejecuta una instrucción completa por ciclo
Segmentado	1 instrucción cada 2 ns (pipeline lleno)	Etapas paralelas

Speed-Up (Aceleración)

$$Speed - Up = \frac{T\_Uniciclo}{T\_Segmentado}$$

$Speed - Up = \frac{30}{14} \approx 2.14$

Escalabilidad

A medida que aumenta el número de instrucciones, **el rendimiento del pipeline mejora y se acerca a una instrucción por ciclo.**

Para muchas instrucciones, el tiempo en pipeline tiende a:

$$T_{pipeline} \approx (n + 4)\ x\ Ciclo$$

La n representa el número de **instrucciones** que se ejecutan.

¿Por qué se suma 4?: En un pipeline clásico de 5 etapas, se necesitan 4 ciclos adicionales para llenar el pipeline (tubería) antes de que se empiecen a completar instrucciones "una por ciclo".

- Durante los primeros 4 ciclos, las instrucciones todavía están entrando al pipeline.
- A partir del ciclo 5 (si no hay riesgos), se empieza a completar una instrucción por ciclo.

Entonces:

Si se ejecutan **n instrucciones**, el tiempo total en un pipeline ideal es:

$$T_{pipeline} \approx (n + 4)\ x\ Ciclo$$

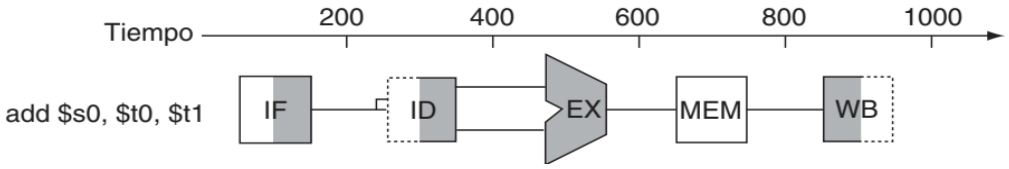
Esto refleja que al inicio hay una **latencia de arranque** de 4 ciclos hasta que el pipeline empieza a rendir a pleno.

3. Riesgos y Dependencias

Tipos de Dependencias

Tipo	Nombre	Descripción
RAW	Read After Write	Una instrucción quiere leer un dato que otra aún no escribió
WAR	Write After Read	Una instrucción escribe antes de que otra lea el mismo dato
WAW	Write After Write	Dos instrucciones escriben el mismo registro

En arquitecturas RISC, la más común es **RAW**.
Los **riesgos de datos** surgen porque una instrucción necesita un dato que aún no está listo.



El sombreado en la mitad derecha indica lectura; en la izquierda, escritura.Por tanto, la mitad derecha de ID está sombreada en la segunda etapa porque se lee el banco de registros, y la mitad izquierda de WB está sombreada en la quinta etapa porque se escribe en el banco de registros.

Resolución de Riesgos

Adelantamiento de Resultados (Forwarding)

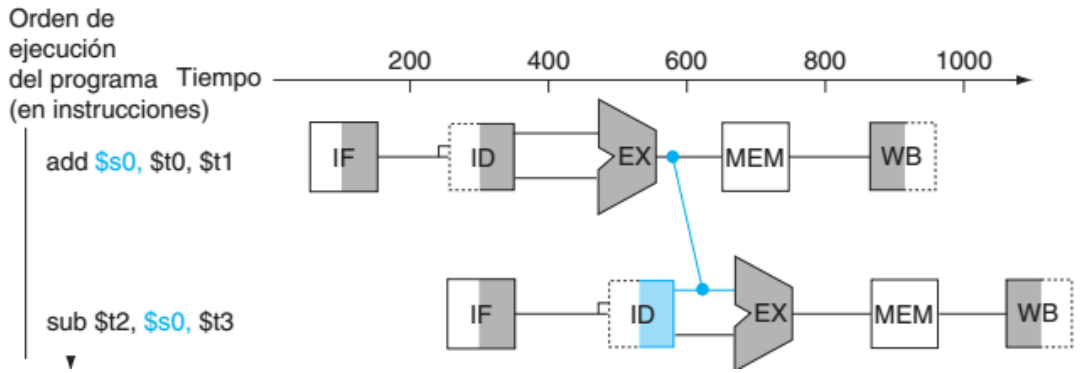
- Se **toma el valor directamente desde una etapa intermedia** (EX, MEM, WB), antes de que se escriba en el banco de registros.
- Esto evita esperas innecesarias si el dato ya está disponible.

Bloqueos del Pipeline (Stalls o Burbujas)

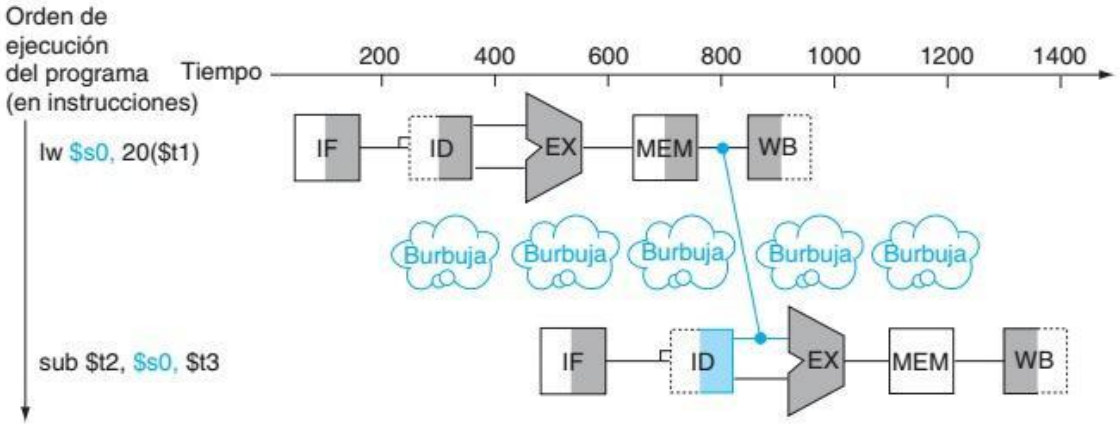
- Se **insertan instrucciones NOP** para dar tiempo a que los datos estén listos.
- Se usa cuando no se puede aplicar forwarding, como en una dependencia inmediata después de una carga (lw).

Anticipación (Bypassing)

- El resultado se **anticipa desde una instrucción anterior a una posterior**, sin pasar por el banco de registros.
- Solo es válida si la etapa destino ocurre más adelante en el tiempo que la etapa origen.
- Conexión para anticipar el valor de $s0 después de la etapa de ejecución de la instrucción add a la entrada de la etapa de ejecución de la instrucción sub.



Las líneas de anticipación de datos sólo son válidas si la etapa destino está más adelante en el tiempo que la etapa origen



El dato deseado sólo estaría disponible después de la cuarta etapa de la primera instrucción, lo cual es demasiado tarde para la entrada de la etapa EX de la instrucción sub. Por lo tanto, incluso con la anticipación de resultados, habría que bloquear durante una etapa en el caso del riesgo del dato de una carga (load-use data Hazard).

4. Diagramas de Pipeline

Visualizar en qué ciclo está cada instrucción ayuda a detectar:

- Atascos
- Oportunidades de adelantamiento
- Cuellos de botella

Ejemplo sin adelantamiento

```
Ciclo:   1  2  3   4    5    6    7   8    9
Instr1:  IF ID EX  MEM  WB
Instr2:     IF ID  -    EX   MEM  WB
Instr3:        IF  -    ID   -    EX  MEM  WB
```

Cada "-" representa un bloqueo (*stall)*. La cantidad de ciclos perdidos por dependencias es clave para analizar el rendimiento.

- Las instrucciones están parcialmente superpuestas en el tiempo.

- Las líneas de dependencia van de etapas más adelantadas a más atrasadas si hay un **riesgo de datos**.

5. Adelantamiento

Es una técnica para **evitar riesgos de dependencia de datos** ("data hazards") pasando los datos directamente entre etapas del pipeline **sin esperar a que se escriban en el banco de registros**.

Tipos comunes de instrucciones y su adelantamiento

Tipo de instrucción	**¿Desde dónde puede hacerse el forwarding?**	**¿Hacia qué etapa se hace?**
R tipo (ALU) **como ADD, SUB, etc.**	Desde **EX**, **MEM**, o **WB**	Hacia **EX** de la siguiente instrucción
Carga (LW)	Desde **MEM** o **WB**	Hacia **EX** o **ID** (según el caso)
Almacenamiento (SW)	Forward del dato a almacenar desde **EX**, **MEM** o **WB**	Hacia **MEM** (porque es donde ocurre la escritura a memoria)
Branch (BEQ, BNE)	Forward para comparar operandos desde **EX**, **MEM** o **WB**	Hacia **ID** (porque en ID se hace la comparación)

Ejemplos comunes

1. **Instrucciones tipo R (ej. ADD)**
   - Si una instrucción en EX depende del resultado anterior en EX o MEM, se puede hacer *adelantamiento* desde EX/MEM o MEM/WB.
2. **LW seguido de instrucción que usa ese valor**
   - No se puede hacer adelantamiento desde EX, porque el dato recién estará disponible en MEM.
   - Esto puede generar un **riesgo de carga (load hazard)** → a veces se necesita *stall*.
3. **SW con dato a guardar calculado por una ALU antes**
   - El dato a almacenar se puede "adelantar" desde EX o MEM, según cuándo se haya producido.

En resumen:

- Adelantamiento ocurre **desde EX, MEM o WB**.
- Suele dirigirse hacia **la etapa EX** de una instrucción que necesita el operando.
- **LW es el caso especial**: el resultado no está disponible hasta MEM → puede requerir *stal (bloqueo)l*.

6. Casos de Dependencias Típicas

Supongamos:

```
sub x2, x1, x3
and x12, x2, x5
or  x13, x6, x2
```

- and necesita el resultado de sub → riesgo **RAW**.
- El valor de x2 aún no está escrito cuando and lo necesita en EX.
- Si el valor está disponible en la etapa EX/MEM, se puede usar **forwarding**.
- Si no, se inserta una **burbuja**.

Soluciones:

- Si el resultado está en EX/MEM o MEM/WB, se puede **adelantar**.
- Si viene de una instrucción load, puede no estar disponible a tiempo → se **inserta burbuja**.

Ejemplo Avanzado de Riesgos de Datos

```
sub x2, x1, x3
and x12, x2, x5
or  x13, x6, x2
```

- and necesita x2, pero sub recién lo escribe en WB.
- Riesgo de tipo **RAW**, detectado cuando and está en EX y sub en MEM → **Condición 1**: EX/MEM.RegisterRd = ID/EX.RegisterRs = x2

Tipos de detección de riesgo RAW:

1. EX/MEM.RegisterRd = ID/EX.RegisterRs o Rt
2. MEM/WB.RegisterRd = ID/EX.RegisterRs o Rt

Si no se puede anticipar el valor a tiempo, se genera un **bloqueo**.

7. Riesgo de Carga y Uso (Load-Use Hazard)

```
lw s0, 0(a0)
sub t0, s0, t1
```

- El valor de s0 recién está disponible **después del ciclo MEM**, pero sub lo necesita en EX.
- No puede adelantarse el resultado→ se inserta una **burbuja** en el ciclo intermedio.
- El pipeline se detiene insertando un nop en la etapa EX. Esto introduce un **retraso de 1 ciclo** para evitar leer un valor incorrecto.

8. Saltos Condicionales y Riesgos de Control

El problema

Las instrucciones de salto (beq, bne, blt, etc.) generan un **riesgo de control** porque el procesador no sabe si el salto se tomará hasta que evalúa la condición en la etapa EX. Mientras tanto, las instrucciones que siguen ya pueden estar entrando al pipeline. Si el salto se **toma**, esas instrucciones deben ser **descartadas (flush)**.

Comportamiento en RISC-V

- **No se ejecuta automáticamente la instrucción siguiente al salto.**

- Si no hay predicción, se introduce un **stall** (1 o 2 ciclos).
- Si se usa **predicción de saltos**, el procesador puede suponer:
    - Que el salto **se toma** o **no se toma**.
    - Si se falla en la predicción → **flush de las instrucciones en IF, ID y EX**.
- **No existe salto retardado** en RISC-V.
-

Técnicas de predicción

- **Estáticas:** Suponer que el salto no se toma.
- **Dinámicas:** Usar una Branch History Table o BHT (es una tabla indexada por bits del PC (Program Counter)), con:
    - **Predictor de 1 bit**
    - **Predictor de 2 bits**, más resistente a fluctuaciones

Esto evita pérdidas innecesarias de ciclos y mejora la eficiencia en bucles y bifurcaciones condicionales frecuentes.

Sin predicción

- El procesador debe esperar 2 o 3 ciclos (stalls).
- Se pierden ciclos por traer instrucciones equivocadas.

Predicción de saltos

- **BHT (Branch History Table):** Guarda el historial de saltos anteriores.
- **Predictor de 1 bit:** Recuerda si el salto se tomó la última vez.
- **Predictor de 2 bits:** Tolerante a cambios bruscos, más preciso.

Con predicción se reduce la cantidad de ciclos perdidos en los bucles.

9. Reordenamiento de Instrucciones

Reordenar instrucciones independientes puede:

- Evitar riesgos RAW
- Reducir el uso de burbujas
- Mejorar el rendimiento sin alterar la lógica del programa

Reglas:

- No cambiar la lógica del programa.
- Instrucciones reordenadas deben ser **independientes** de los resultados pendientes.

Ejemplo:

```
lw x2, B
lw x3, C
add x1, x2, x3
sw x1, A
```

→ Podemos intercalar instrucciones entre lw y add para **rellenar los ciclos vacíos**.

Esta no es una técnica aplicada por el hardware sino una práctica de optimización por el compilador o el programador.

10. Simulación con RARS + BHT

Se puede usar **RARS** con simulación de saltos para:

- Ver ciclos totales
- Medir precisión de predicción

- Comparar impacto de diferentes técnicas

Fórmulas:

- **CPI** = Total de ciclos / Instrucciones
- **Tiempo** = Total de ciclos / Frecuencia
- **Precisión predicción** = Saltos correctos / Total de saltos

11. Análisis de CPI y Tiempo de Ejecución

CPI: Ciclos por instrucción

$$CPI = \frac{Ciclos\ totales}{Número\ de\ instrucciones}$$

Tiempo de ejecución

$$Tiempo\ de\ ejecución = \frac{Ciclos\ totales}{Frecuencia}$$

12. Caso real: Programa con Saltos y BHT

```
loop:
  add t0, t0, t1
  addi t1, t1, 1
  blt t1, t2, loop
```

El salto (blt) se toma 9 veces y no se toma la décima.

- Con predictor de 1 bit: la primera predicción puede fallar (no se toma aún).
- El fallo final también es típico.
- **Predicción ideal:** 2-bit con saturación hacia el lado tomado.

13. Casos que No Se Pueden Resolver con Adelantamiento

Ejemplo típico:

```
lw $t1, 0($t0)
add $t2, $t1, $t3
```

- El dato se carga desde memoria en MEM.
- add lo necesita en EX → no está disponible aún.
- **Se debe insertar una burbuja.**

14. Condiciones que se pueden presentar

Son **comparaciones entre campos** de diferentes registros inter-etapas del pipeline.

EX/MEM.RegisterRd = ID/EX.RegisterRs
MEM/WB.RegisterRd = ID/EX.RegisterRs

¿Qué es un registro de segmentación?

En un pipeline, entre cada etapa hay registros que **guardan los datos que pasan de una etapa a otra**.

Por ejemplo:

- ID/EX: contiene la información de la instrucción que **acaba de salir de la etapa ID** y entra a la etapa EX.
- EX/MEM: contiene la instrucción que **salió de EX** y entra a MEM.
- MEM/WB: contiene la instrucción que **salió de MEM** y entra a WB.

¿Qué significan los campos?

- RegisterRd: es el **registro destino** de una instrucción (donde se escribirá el resultado).
- RegisterRs: es uno de los **registros fuente** de una instrucción (de donde se lee un operando).

Entonces:

EX/MEM.RegisterRd = ID/EX.RegisterRs

El registro destino de una instrucción que está en EX/MEM coincide con el registro fuente que necesita una instrucción que está en ID/EX.
• Traducción práctica:
  ○ Una instrucción anterior va a escribir en un registro.
  ○ Una instrucción siguiente lo quiere leer antes de que esté listo.
  ○ Hay un riesgo de datos → necesitamos adelantar (forwarding) o bloquear (stall).

MEM/WB.RegisterRd = ID/EX.RegisterRs
El registro que se escribirá en la etapa WB (MEM/WB) es el mismo que la instrucción en EX quiere leer. • Significa que una instrucción más antigua aún no terminó de escribir el dato, pero una más nueva ya lo necesita.

¿Por qué se usan estas condiciones?

Estas comparaciones se hacen en el **hardware** para detectar automáticamente si hay una **dependencia RAW** entre dos instrucciones que están "vivas" en el pipeline.
Si se cumple alguna de estas condiciones:

- El procesador debe **redireccionar los datos** (adelantamiento), o
- Insertar una **burbuja (NOP)** si no se puede adelantar.

Ejemplo

Supongamos estas instrucciones:

```
1: add x3, x1, x2      # x3 = x1 + x2
2: sub x4, x3, x5      # x4 = x3 - x5  ← usa el resultado anterior
```

- En el ciclo 3, la instrucción 1 está en EX/MEM (listo para escribir en x3).
- En ese mismo ciclo, la instrucción 2 está en ID/EX y **quiere leer x3** como operando.

→ Se cumple:

EX/MEM.RegisterRd = x3 = ID/EX.RegisterRs

Hay **riesgo RAW** y se necesita **forwarding** desde la instrucción 1 a la 2.

Detección de dependencias RAW mediante registros de segmentación

En un procesador segmentado, cada etapa del pipeline está conectada a través de **registros inter-etapas** que conservan la información necesaria para la siguiente fase. Estos registros tienen nombres como:

- ID/EX: entre las etapas de **decodificación (ID)** y **ejecución (EX)**.
- EX/MEM: entre **ejecución (EX)** y **acceso a memoria (MEM)**.
- MEM/WB: entre **memoria (MEM)** y **escritura final (WB)**.

Dentro de estos registros se guardan datos como:

- **RegisterRd**: el número del **registro destino**, donde se escribirá un resultado.
- **RegisterRs y RegisterRt**: los **registros fuente**, de donde se leen operandos.

Comparaciones para detectar riesgos RAW

El hardware puede detectar riesgos RAW comparando los campos de estos registros de segmentación.

Las condiciones son:

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs
1b. EX/MEM.RegisterRd = ID/EX.RegisterRt
2a. MEM/WB.RegisterRd = ID/EX.RegisterRs
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

¿Qué significa cada una?

- **1a y 1b:** La instrucción que está por entrar en EX **necesita leer un registro**, pero ese mismo registro **será escrito por una instrucción que está en MEM**.
  → Se puede aplicar **adelantamiento desde EX/MEM**.
- **2a y 2b:** La instrucción en EX **necesita leer un registro**, pero ese dato está por escribirse en WB.
  → Se puede aplicar **adelantamiento desde MEM/WB**.