Universidad de Córdoba

Arquitectura y Tecnología de Computadores

Práctica 5

MIPSim y los riesgos de datos

Arquitecturas Avanzadas de Procesadores

D. Miguel Ángel Montijano Vizcaíno (el1movim@uco.es)
D. Héctor Martínez Pérez (el2mapeh@uco.es)

Curso 2025/2026

1 Introducción

Esta práctica continuará con los fundamentos aprendidos sobre MIPSim en la práctica anterior y abordará los distintos tipos de riesgos de datos que pueden surgir en esta clase de arquitecturas segmentadas.

Los objetivos principales de esta práctica son:

- Profundizar en las simulaciones con MIPSim.
- Detectar los distintos tipos de riesgos de datos que puede ocasionar un determinado código ensamblador/máquina.
- Conocer y saber aplicar diferentes técnicas para solventar esta problemática.

2 Riesgos de datos (hazards)

En la práctica anterior se pudo observar cómo en ciertos casos el resultado que acababan tomando ciertos registros o direcciones de memoria no era el esperado. Esto ocurre cuando existen instrucciones en nuestro código que tienen dependencia de datos entre ellas. En arquitecturas segmentadas, esta dependencia de datos puede llegar a ser un gran conflicto. En el caso que nos ocupa, la arquitectura MIPS segmentada que emula MIPSim no dispone de anticipación de datos; por este motivo, a la hora de desarrollar el código a ejecutar hay que tener en cuenta este factor.

Existen distintos tipos de riesgos de datos. A continuación se presenta una lista con algunos de los tipos de riesgos más comunes:

- **RAW (Read After Write):** Se da cuando una instrucción lee un dato que otra instrucción previa debe actualizar o escribir primero. En este caso, si la lectura se lleva a cabo antes que la escritura, la ubicación no contendrá el valor correcto.
- **WAR (Write After Read):** Este riesgo de datos se da cuando una instrucción escribe en una ubicación donde otra instrucción anterior lee su contenido. Que se adelante la escritura de la instrucción que es posterior, pueda dar lugar a que la instrucción previa no lea el dato correcto.
- **WAW (Write After Write):** Cuando dos instrucciones escriben en una misma posición de memoria. Si hay adelantamiento de la instrucción posterior, puede ser que el dato almacenado finalmente no sea el correcto.

El Código 1 muestra un ejemplo de riesgo RAW. Nótese cómo la instrucción 2, `SW`, escribe en la posición de memoria 0x00000000 el dato leído desde el registro `$1`. Este registro se actualiza en la instrucción 1, `ADDI`. Si el dato no está disponible en la etapa donde se ejecuta la instrucción 2, el valor leído desde el banco de registro no será el correcto.

Código 1: Riesgo de tipo RAW.

```
1 ADDI $1, $0, 4
2 SW   $1, 0($0)
3 LW   $3, 4($0)
4 ADDI $4, $3, 8
```

Como ejercicio de entrenamiento, introduzca el siguiente código en el simulador MIPSim y observe el comportamiento de la ejecución.

Una manera de solventar estos tipos de riesgos de datos es insertando instrucciones de NO-Operación (`NOP`) entre las instrucciones donde ocurre el conflicto. Esto insertará instrucciones que no llevan a cabo ninguna operación pero ocupan ciclos de reloj, los que ayudan a que el dato se almacene a tiempo.

Dicho esto, inserte las instrucciones `NOP` necesarias entre la primera y la segunda instrucción para eliminar este riesgo de datos.

Otra forma de solventar este tipo de riesgo de datos es reordenando el código; esta solución es generalmente la más óptima debido a que no se pierden ciclos de reloj.

Ejercicio 1

1.1 Empezando con el primero de los ejercicios, se deberán detectar todas las dependencias de datos existentes en el anterior código y solventarlas (ya sea bien reordenando el código o insertando instrucciones `NOP`).

1.2 ¿En cuántos ciclos de reloj totales se consigue ejecutar todas las instrucciones en el pipeline segmentado?

Ejercicio 2

Para llevar a cabo este ejercicio, inicialmente almacene en las posiciones de memoria 0x00000000 y 0x000000004, los valores 8 y 4, respectivamente.

Código 2: Código ejercicio 2.

```
1 LW   $1,0($0)
2 LW   $2,4($0)
3 ADD  $3,$1,$2
4 SUB  $4,$1,$2
5 SW   $3,8($0)
6 SW   $4,c($0)
```

2.1 ¿Qué resultado se obtiene al final de la ejecución?

2.2 Si no es el esperado, indique qué tipo de riesgo de segmentación ha aparecido y cómo se podría resolver.

Ejercicio 3

Inicialice el banco de datos con los mismos valores que el ejercicio anterior. En este caso, además, inicialice el registro `$1` a 0 y el `$4` a 5.

Código 3: Código ejercicio 3.

```
LW   $2,0($1)
LW   $3,4($1)
ADDI $4,$4,1
SW   $3,0($1)
SW   $2,4($1)
```

3.1 ¿Qué resultado se obtiene?

3.2 Si no es el esperado, indique qué tipo de riesgo de segmentación ha aparecido y cómo se podría resolver.

Ejercicio 4

Código 4: Código ejercicio 4.

```
SUB $4, $4, $4
LW  $1, 0($0)
LW  $2, 4($0)
AND $2, $2, $1
LW  $3, 8($0)
AND $3, $3, $1
ADD $4, $3, $2
SW  $2, 4($0)
SW  $3, 8($0)
```

4.1 Resuelva las dependencias de datos mediante la inserción de instrucciones `NOP`.

4.2 Resuelva las dependencias de datos mediante reordenamiento del código junto a la inserción de instrucciones `NOP`.

Ejercicio 5

Código 5: Código ejercicio 5.

```
LW  $1,0($0)
LW  $2,4($0)
NOP
NOP
ADD $3,$1,$2
LW  $4,8($0)
LW  $5,c($0)
NOP
NOP
SUB $6,$4,$5
SW  $3,10($0)
```

5.1 ¿Cómo se podría modificar el código para eliminar algunas de las instrucciones nop?

Ejercicio 6

Inicialice todas las posiciones de memoria a 0. Asimismo, los registros `$1, $2, $3 y $6` deben contener el valor 7, 7, 5, y 0 respectivamente.

Código 6: Código ejercicio 6.

```
1         BEQ   $1,$2,salto
2         ADD   $3,$1,$2
3         SUB   $4,$1,$2
4         ADDI  $5,$5,1
5 salto:  SW    $3,0($6)
```

6.1 ¿Qué resultado se obtiene?

6.2 Si no es el esperado, indique qué tipo de riesgo de segmentación ha aparecido y cómo se podría resolver.

6.3 Ejecute el mismo programa cambiando el valor del registro $3 a 6. ¿Qué resultado se ha obtenido ahora? Si no es el esperado, indique qué tipo de riesgo de segmentación ha aparecido y cómo se podría resolver.