Arquitectura y Tecnología de Computadores. Curso 10/11

Arquitecturas Paralelas. **Practica IV**

RIESGOS EN PROCESADORES SEGMENTADOS (I)

Simulación de la Arquitectura del MIPS R3000

1. Objetivo:

Evaluar la incidencia de los **riesgos estructurales**, los **riesgos por dependencias de datos** y los **riesgos de control** sobre el rendimiento de un procesador segmentado real mediante la simulación de la arquitectura de dicho procesador.

2. Análisis a realizar:

La práctica se va llevar a cabo mediante el empleo de dos simuladores de la arquitectura DLX, arquitectura que se corresponde con la del procesador segmentado visto en clase y muy similar a la del procesador comercial **MIPS R3000** de 32 bits.

Los Ingenieros de la firma MIPS utilizaron un diseño segmentado de cinco etapas para el MIPS R3000. Esta estructura segmentada permite la ejecución concurrente de hasta un máximo de cinco instrucciones, cada un de ellas en una etapa diferente de su cauce, siendo posible obtener un **ritmo ideal de ejecución de una instrucción por ciclo**. Las etapas del cauce de ejecución son: **Búsqueda de Instrucción** (IF), **Decodificación de Instrucción y Lectura de Operandos** (ID), **Ejecución** (EX), **Acceso a Datos en Memoria** (MEM) y **Escritura de Registro** (WB). Otras características importantes del R3000 son la existencia de memorias CACHE de Instrucciones y Datos independientes en el propio chip y la avanzada **optimización de código** que permiten sus compiladores.

Los análisis a realizar con objeto de poner de manifiesto la influencia de los riesgos aludidos sobre el rendimiento del procesador segmentado son los siguientes:

1. Analizar la **influencia de los problemas estructurales** sobre el ritmo de ejecución en el cauce, tanto los debidos a la escasez de recursos (vías de acceso a memoria), como los debidos a la falta de segmentación en alguna de sus unidades funcionales (unidades de punto flotante).

2. Analizar la **influencia de las dependencias de datos** sobre el ritmo de ejecución en el cauce (una de las instrucciones establece un nuevo valor del contenido de un registro y alguna de las posteriores utiliza el mismo como uno de sus operandos).

3. Analizar la **influencia de las instrucciones de control** sobre el ritmo de ejecución en el cauce (recordar que la instrucción de salto genera un hueco de retardo de salto que se tratará de rellenar de acuerdo con una determinada estrategia de planificación).

Los análisis se efectuarán sobre programas de prueba sintéticos.

3. Pasos a seguir en la práctica:

- Descarga el programa *Windlx* desde la página web de la asignatura e instálalo en tu PC de trabajo. Ejecuta el simulador y examina sus diferentes menús y ventanas. Configura únicamente el simulador para que cuente ciclos de reloj de forma absoluta.

- Antes de cargar ningún programa, ejecuta las instrucciones del tipo NOP (*No Operation*, que no hacen nada) que están cargadas en memoria por defecto, y examina las diferentes vistas de la ejecución que permite el simulador. De todas ellas, utilizaremos fundamentalmente la *"Clock Cycle Diagram"*, que se corresponde con los cronogramas vistos en clase.

Análisis de Riesgos Estructurales

- Edita y compara los programas **est1-1.s** y **est1-2.s** (figura 1) correspondientes al análisis de la influencia de los problemas estructurales del primer tipo (escasez de recursos). ¿En cuál de ellos crees que puede haber problemas estructurales derivados del acceso a memoria?. Ejecuta cada uno de ellos paso a paso en el simulador *Windlx* y analiza si hay diferencias en su comportamiento ¿a que conclusión llegas después de las simulaciones? (lee de nuevo el punto 2 para encontrar una justificación al comportamiento observado). Anota los ciclos consumidos por cada programa.

est1-1.s

```
        .data  0
dato:   .word  1

        .text
main:
        addi   r20, r0, dato
        addi   r30, r0, 10

loop:   add    r1, r1, r1
        xor    r2, r2, r2
        and    r3, r4, r5
        subui  r30, r30, 1
        add    r6, r7, r8
        add    r9, r10, r11
        bnez   r30, loop
        nop
        trap   #0
```

est1-2.s

```
        .data  0
dato:   .word  1

        .text
main:
        addi   r20, r0, dato
        addi   r30, r0, 10

loop:   add    r1, r1, r1
        lw     r2, (r20)
        and    r3, r4, r5
        subui  r30, r30, 1
        add    r6, r7, r8
        add    r9, r10, r11
        bnez   r30, loop
        nop
        trap   #0
```

Fig. 1. Influencia de los problemas estructurales (I)

- Edita y compara los programas **est2-1.s** y **est2-2.s** (figura 2) correspondientes al análisis de la influencia de los problemas estructurales del segundo tipo (unidades funcionales no totalmente segmentadas). Examina la configuración de las unidades funcionales del procesador mediante la correspondiente opción del menú. Observa sus latencias y ten en cuenta que no están segmentadas. En vista de lo anterior ¿en cuál de ellos crees que puede haber problemas estructurales?. Ejecuta cada uno de ellos en el simulador *Windlx* y analiza si hay diferencias en su comportamiento ¿a que conclusión llegas después de las simulaciones?. Anota los ciclos consumidos por cada programa.

est2-1.s

```
        .data  0
dato:   .word  1

        .text
main:
        addi   r20, r0, dato
        addi   r30, r0, 10

loop:   xor    r2, r2, r2
        and    r3, r4, r5
        subui  r30, r30, 1
        add    r6, r7, r8
        addd   f0, f2, f4
        subd   f6, f8, f10
        bnez   r30, loop
        nop
        trap   #0
```

est2-2.s

```
        .data  0
dato:   .word  1

        .text
main:
        addi   r20, r0, dato
        addi   r30, r0, 10

loop:   xor    r2, r2, r2
        and    r3, r4, r5
        subui  r30, r30, 1
        addd   f0, f2, f4
        add    r6, r7, r8
        subd   f6, f8, f10
        bnez   r30, loop
        nop
        trap   #0
```

Fig. 2. Influencia de los problemas estructurales (II)

- Sigue las indicaciones de la página web de la asignatura para abrir una sesión X en *centauro*. Copia a tu cuenta los programas de prueba vía ftp (modo ascii). Ejecuta el simulador *Dlxview* mediante el comando del mismo nombre.

- Repite las mismas simulaciones, observando además como evoluciona el hardware del camino de datos enteros del procesador. Observa que el número total de ciclos consumidos cambia ¿por qué?

- El simulador *Dlxview* permite además de configurar la latencia de las unidades funcionales, configurar así mismo si dichas unidades funcionales están o no segmentadas. Configura la unidad flotante de suma como NO segmentada y repite la simulación del programa que planteaba problemas estructurales (**est2-1.s)**. Analiza su comportamiento y determina el número total de ciclos consumidos ¿a que conclusión llegas después de la simulación? ¿por qué sigue sin coincidir el tiempo con el resultante en *Windlx*?

Análisis de Riesgos por Dependencias de Datos

- Edita y compara los programas **dep1-1.s** y **dep1-2.s** (figura 3) correspondientes al análisis de la influencia de las dependencias de datos NO provocadas por instrucciones de carga ¿en cuál de ellos crees que se da realmente una dependencia de datos?. Ejecuta cada uno de ellos en el simulador *Windlx* con el *forwarding* deshabilitado (menú de configuración) y analiza si hay diferencias en su comportamiento ¿a que conclusión llegas después de las simulaciones?. Anota los ciclos consumidos por cada programa.

- Edita el programa **dep1-3.s** y compáralo con el **dep1-2.s** ¿cuál es la solución que propone este programa al problema de la dependencia de datos?. Simula este último programa y compara los ciclos consumidos con los anteriores ¿a que conclusión llegas después de la simulación?

- Ahora vamos a comprobar el efecto de la técnica hardware de adelantamiento (*forwarding*) sobre el comportamiento del procesador. Para ello habilita el *forwarding* en el menú de configuración y vuelve a simular los programas que planteaban problemas por dependencias de datos. Analiza su comportamiento y determina el número total de ciclos consumidos ¿a que conclusión llegas después de la simulación?

dep1-1.s

```
        .data  0
dato:   .word  1

        .text
main:
        addi   r20, r0, dato
        addi   r30, r0, 10

loop:   add    r1, r1, r1
        xor    r2, r2, r2
        and    r3, r4, r5
        subui  r30, r30, 1
        add    r6, r7, r8
        add    r9, r10, r11
        bnez   r30, loop
        nop
        trap   #0
```

dep1-2.s

```
        .data  0
dato:   .word  1

        .text
main:
        addi   r20, r0, dato
        addi   r30, r0, 10

loop:   add    r1, r1, r1
        xor    r2, r2, r2
        and    r3, r2, r4
        subui  r30, r30, 1
        add    r5, r6, r7
        add    r8, r9, r10
        bnez   r30, loop
        nop
        trap   #0
```

dep1-3.s

```
        .data  0
dato:   .word  1

        .text
main:
        addi   r20, r0, dato
        addi   r30, r0, 10

loop:   add    r1, r1, r1
        xor    r2, r2, r2
        subui  r30, r30, 1
        and    r3, r2, r4
        add    r5, r6, r7
        add    r8, r9, r10
        bnez   r30, loop
        nop
        trap   #0
```

Fig. 3. Influencia de las dependencias de datos (I)

- Deshabilita de nuevo el *forwarding* en *Windlx.* Edita y compara los programas **dep2-1.s** y **dep2-2.s** (figura 4) correspondientes al análisis de la influencia de las dependencias de datos provocadas por instrucciones de carga ¿en cuál de ellos crees que se da realmente una dependencia de datos?. Ejecuta cada uno en *Windlx* y analiza si hay diferencias en su comportamiento ¿a que conclusión llegas después de las simulaciones?. Anota los ciclos consumidos por cada programa.

- Edita el programa **dep2-3.s** y compáralo con el **dep2-2.s** ¿cuál es la solución que propone este programa al problema de la dependencia de datos?. Simula este último programa y compara los ciclos consumidos con los anteriores ¿a que conclusión llegas después de la simulación?

- Ahora vamos a comprobar de nuevo el efecto de la técnica hardware de adelantamiento (*forwarding*) sobre el comportamiento del procesador. Para ello habilita el forwarding en el menú de configuración y vuelve a simular los programas que planteaban problemas por dependencias de datos. Analiza su comportamiento y determina el número total de ciclos consumidos ¿a que conclusión llegas después de la simulación? ¿que número de ciclos de procesador supone el *"hueco de retardo de carga"*?

dep2-1.s

```
      .data  0
dato: .word  1

      .text
main:
      addi   r20, r0, dato
      addi   r30, r0, 10

loop: add    r1, r1, r1
      lw     r2, (r20)
      and    r3, r4, r5
      subui  r30, r30, 1
      add    r6, r7, r8
      add    r9, r10, r11
      bnez   r30, loop
      nop
      trap   #0
```

dep2-2.s

```
      .data  0
dato: .word  1

      .text
main:
      addi   r20, r0, dato
      addi   r30, r0, 10

loop: add    r1, r1, r1
      lw     r2, (r20)
      and    r3, r2, r4
      subui  r30, r30, 1
      add    r5, r6, r7
      add    r8, r9, r10
      bnez   r30, loop
      nop
      trap   #0
```

dep2-3.s

```
      .data  0
dato: .word  1

      .text
main:
      addi   r20, r0, dato
      addi   r30, r0, 10

loop: add    r1, r1, r1
      lw     r2, (r20)
      subui  r30, r30, 1
      and    r3, r2, r4
      add    r5, r6, r7
      add    r8, r9, r10
      bnez   r30, loop
      nop
      trap   #0
```

Fig. 4. Influencia de las dependencias de datos (II)

- Copia a tu cuenta en *centauro* los programas de prueba vía ftp (modo ascii). Repite las mismas simulaciones con *Dlxview*, observando además como evoluciona el hardware del camino de datos enteros del procesador ¿qué técnica hardware te parece que esta considerando el simulador en este caso?

Análisis de Riesgos de Control

- Edita y compara los programas **con1.s** y **con2.s** (figura 5) correspondientes al análisis de la influencia de los problemas de control. Observa que la única diferencia del segundo respecto al primero es la ausencia de la instrucción NOP (que de hecho aparece en todos los programas anteriores) y ten en cuenta que la instrucción *trap #0* es la que determina el final del programa. Simula ambos programas con *Windlx* y *Dlxview* y concluye, en función del comportamiento observado, por qué es necesaria la instrucción NOP con *Dlxview* y no con *Windlx*.

- En función del análisis del punto anterior ¿qué número de ciclos de procesador supone el *"hueco de retardo de salto"*?

- Edita el programa **con3.s** y compáralo con el **con1.s** ¿cuál es la solución que propone este programa para tratar de aprovechar el hueco de retardo de salto?. Simula este último programa con *Windlx* y compara los ciclos consumidos en ambos programas. Aunque el tiempo se reduce ¿se aprovecha realmente el hueco? ¿serán los resultados correctos?
  Nota: Al final de la ejecución, el registro r1 deberá estar cargado con el número de iteraciones que efectúa el programa. Puedes comprobarlo mediante la ventana "*Register*".

- Repite las mismas simulaciones con *Dlxview* ¿se aprovecha ahora el hueco de retardo? ¿serán los resultados correctos?

con1.s

```
      .data  0
dato: .word  1

      .text
main:
      addi  r20, r0, dato
      addi  r30, r0, 10
      xor   r1, r1, r1

loop: xor   r2, r2, r2
      and   r3, r4, r5
      subui r30, r30, 1
      add   r6, r7, r8
      add   r9, r10, r11
      addui r1, r1, 1
      bnez  r30, loop
      nop
      trap  #0
```

con2.s

```
      .data  0
dato: .word  1

      .text
main:
      addi  r20, r0, dato
      addi  r30, r0, 10
      xor   r1, r1, r1

loop: xor   r2, r2, r2
      and   r3, r4, r5
      subui r30, r30, 1
      add   r6, r7, r8
      add   r9, r10, r11
      addui r1, r1, 1
      bnez  r30, loop
      trap  #0
```

con3.s

```
      .data  0
dato: .word  1

      .text
main:
      addi  r20, r0, dato
      addi  r30, r0, 10
      xor   r1, r1, r1

loop: xor   r2, r2, r2
      and   r3, r4, r5
      subui r30, r30, 1
      add   r6, r7, r8
      add   r9, r10, r11
      bnez  r30, loop
      addui r1, r1, 1
      trap  #0
```

Fig. 5. Influencia de las instrucciones de control

4. Trabajo a entregar:

- Cronogramas correspondientes a la ejecución de los distintos programas sintéticos
- Tabla resumen de tiempos en ciclos según el modelo del Anexo I

Anexo I: *Tabla resumen de resultados (Excel)*

Programa	Tiempos (ciclos)		Comentarios (2 líneas máximo)	
	WinDLX	DLXView	**WinDLX**	**DLXView**
est1-1				
est1-2				
est2-1				
est2-2				
est2-1 (ss)	-			
dep1-1				
dep1-2				
dep1-3				
dep1-2 (fw)		-		
dep1-3 (fw)		-		
dep2-1				
dep2-2				
dep2-3				
dep2-2 (fw)		-		
dep2-3 (fw)		-		
con1.s				
con2.s				
con3.s				

Anexo II: *Los simuladores WinDLX y DLXView*