PRACTICA 4

Segmentación de cauce en procesador RISC

Objetivos: Comprender el funcionamiento de la segmentación de cauce del procesador MIPS de 64 bits. Analizar las ventajas e inconvenientes de este tipo de arquitectura. Familiarizarse con el desarrollo de programas para procesadores con sets reducidos de instrucciones (RISC). Resolver problemas y verificarlos a través de simulaciones (winmips64)

Los ejercicios con * tienen solución propuesta total o parcial

1) Muchas instrucciones comunes en procesadores con arquitectura CISC no forman parte del repertorio de instrucciones del MIPS64, pero pueden implementarse haciendo uso de una única instrucción. Evaluar las siguientes instrucciones, indicar qué tarea realizan y cuál sería su equivalente en lenguaje assembly del x86.

a) `dadd r1, r2, r0` Suma r2 con r0, dejando el resultado en r1 (valores con signo). Su equivalente es ADD

b) `daddi r3, r0, 5` Suma r0 con el valor inmediato 5, dejando el resultado en r3 (valores con signo). Su equivalente es ADD.

c) `dsub r4, r4, r4` Resta r4 a r4, dejando el resultado en r4 (valores con signo). Su equivalente es SUB

d) `daddi r5, r5, -1` Suma r5 con el valor inmediato -1, dejando el resultado en r5 (valores con signo). Su equivalente es ADD

e) `xori r6, r6, 0xffffffffffffffff` Realiza un XOR entre r6 y el valor inmediato 0xffffffffffffffff (bit a bit), dejando el resultado en r6. Su equivalente es XOR.

2) * El siguiente programa intercambia el contenido de dos palabras de la memoria de datos, etiquetadas A y B.

```
    .data
A:  .word  1
B:  .word  2
    .code
    ld     r1, A(r0)      Copia en r1 un double word (64 bits) desde la dirección (A+r0)
    ld     r2, B(r0)      Copia en r2 un double word (64 bits) desde la dirección (B+r0)
    sd     r2, A(r0)      Guarda r2 a partir de la dirección (A+r0)
    sd     r1, B(r0)      Guarda r1 a partir de la dirección (B+r0)
    halt
```

a) Ejecutarlo en el simulador con la opción Configure/Enable Forwarding deshabilitada. Analizar paso a paso su funcionamiento, examinar las distintas ventanas que se muestran en el simulador y responder:
- ¿Qué instrucción está generando atascos (stalls) en el cauce (ó pipeline) y por qué?
- ¿Qué tipo de 'stall' es el que aparece? Raw
- ¿Cuál es el promedio de Ciclos Por Instrucción (CPI) en la ejecución de este programa bajo esta configuración? 3 CPI

b) Una forma de solucionar los atascos por dependencia de datos es utilizando el Adelantamiento de Operandos o Forwarding. Ejecutar nuevamente el programa anterior con la opción Enable Forwarding habilitada y responder:
- ¿Por qué no se presenta ningún atasco en este caso? Explicar la mejora.
- ¿Qué indica el color de los registros en la ventana Register durante la ejecución?
- ¿Cuál es el promedio de Ciclos Por Instrucción (CPI) en este caso? Comparar con el anterior.

Porque los resultados de una etapa del pipeline se reenvíen directamente a una etapa anterior para ser utilizados por instrucciones posteriores. Esto ayuda a evitar que las instrucciones tengan que esperar a que los datos estén disponibles en la memoria o en el registro, mejorando así el rendimiento general del pipeline.

El color verde en el registro R1 significa que el dato está disponible en etapa MEM para adelantamiento. 1.8 CPI

3) * Analizar el siguiente programa con el simulador MIPS64:

```
        .data
A:      .word 1
B:      .word 3
        .code
        ld     r1, A(r0)
        ld     r2, B(r0)
loop:   dsll   r1, r1, 1
        daddi  r2, r2, -1
        bnez   r2, loop
        halt
```

a) Ejecutar el programa con Forwarding habilitado y responder:
- ¿Por qué se presentan atascos tipo RAW?
- Branch Taken es otro tipo de atasco que aparece. ¿Qué significa? ¿Por qué se produce?
- ¿Cuántos CPI tiene la ejecución de este programa? Tomar nota del número de ciclos, cantidad de instrucciones y CPI.

b) Ejecutar ahora el programa deshabilitando el Forwarding y responder:
- ¿Qué instrucciones generan los atascos tipo RAW y por qué? ¿En qué etapa del cauce se produce el atasco en cada caso y durante cuántos ciclos?
- Los Branch Taken Stalls se siguen generando. ¿Qué cantidad de ciclos dura este atasco en cada vuelta del lazo 'loop'? Comparar con la ejecución con Forwarding y explicar la diferencia.
- ¿Cuántos CPI tiene la ejecución del programa en este caso? Comparar número de ciclos, cantidad de instrucciones y CPI con el caso con Forwarding.

c) Reordenar las instrucciones para que la cantidad de RAW sea '0' en la ejecución del programa (Forwarding habilitado)

d) Modificar el programa para que almacene en un arreglo en memoria de datos los contenidos parciales del registro r1 ¿Qué significado tienen los elementos de la tabla que se genera?

4) * Dado el siguiente programa:

```
        .data
tabla: .word 20, 1, 14, 3, 2, 58, 18, 7, 12, 11
num:   .word 7
```

```
long:  .word 10
       .code
       ld     r1, long(r0)
       ld     r2, num(r0)
       dadd   r3, r0, r0
       dadd   r10, r0, r0
loop:  ld     r4, tabla(r3)
       beq    r4, r2, listo
       daddi  r1, r1, -1
       daddi  r3, r3, 8
       bnez   r1, loop
       j      fin
listo: daddi  r10, r0, 1
fin:   halt
```

a) Ejecutar en simulador con Forwarding habilitado. ¿Qué tarea realiza? ¿Cuál es el resultado y dónde queda indicado?
b) Re-Ejecutar el programa con la opción Configure/Enable Branch Target Buffer habilitada. Explicar la ventaja de usar este método y cómo trabaja.
c) Confeccionar una tabla que compare número de ciclos, CPI, RAWs y Branch Taken Stalls para los dos casos anteriores.

5) El siguiente programa multiplica por 2 los elementos de un arreglo llamado datos y genera un nuevo arreglo llamado res. Ejecutar el programa en el simulador winmips64 con la opción Delay Slot habilitada.

```
       .data
cant:  .word 8
datos: .word 1, 2, 3, 4, 5, 6, 7, 8
res:   .word 0
       .code
       dadd   r1, r0, r0
       ld     r2, cant(r0)
loop:  ld     r3, datos(r1)
       daddi  r2, r2, -1
       dsll   r3, r3, 1
       sd     r3, res(r1)
       daddi  r1, r1, 8
       bnez   r2, loop
       nop
       halt
```

a) ¿Qué efecto tiene habilitar la opción Delay Slot (salto retardado)?.
b) ¿Con qué fin se incluye la instrucción NOP? ¿Qué sucedería si no estuviera?.
c) Tomar nota de la cantidad de ciclos, la cantidad de instrucciones y los CPI luego de ejecutar el programa.
d) Modificar el programa para aprovechar el 'Delay Slot' ejecutando una instrucción útil. Simular y comparar número de ciclos, instrucciones y CPI obtenidos con los de la versión anterior.

6) Escribir un programa que lea tres números enteros A, B y C de la memoria de datos y determine cuántos de ellos son iguales entre sí (0, 2 o 3). El resultado debe quedar almacenado en la dirección de memoria D.

7) * Escribir un programa que recorra una TABLA de diez números enteros y determine cuántos elementos son mayores que X. El resultado debe almacenarse en una dirección etiquetada CANT. El programa debe generar además otro arreglo llamado RES cuyos elementos sean ceros y unos. Un '1' indicará que el entero correspondiente en el arreglo TABLA es mayor que X, mientras que un '0' indicará que es menor o igual.

8) * Escribir un programa que multiplique dos números enteros utilizando sumas repetidas (similar a Ejercicio 6 o 7 de la Práctica 1). El programa debe estar optimizado para su ejecución con la opción Delay Slot habilitada.

9) Escribir un programa que implemente el siguiente fragmento escrito en un lenguaje de alto nivel:

```
while a > 0 do
begin
       x := x + y;
       a := a - 1;
end;
```

Ejecutar con la opción Delay Slot habilitada.

10) Escribir un programa que cuente la cantidad de veces que un determinado caracter aparece en una cadena de texto. Observar cómo se almacenan en memoria los códigos ASCII de los caracteres (código de la letra "a" es 61H). Utilizar la instrucción lbu (load byte unsigned) para cargar códigos en registros. La inicialización de los datos es la siguiente:

```
       .data
cadena: .asciiz "adbdcdedfdgdhdid"   ; cadena a analizar
car:    .asciiz "d"                  ; caracter buscado
cant:   .word 0                      ; cantidad de veces que se repite el caracter car en cadena.
```