Juan Menduiña

Trabajo Práctico N° 4:
Segmentación de Cauce en Procesador RISC.

Ejercicio 1.

Muchas instrucciones comunes en procesadores con arquitectura RISC 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.*

La tarea que realiza esta instrucción es sumar r2 y r0 (0) y guardar el resultado en r1. Su equivalente en lenguaje Assembly del x86 es *mov r1, r2*.

(b) *daddi r3, r0, 5.*

La tarea que realiza esta instrucción es sumar r0 (0) y 5 y guardar el resultado en r3. Su equivalente en lenguaje Assembly del x86 es *mov r3, 5*.

(c) *dsub r4, r4, r4.*

La tarea que realiza esta instrucción es restar r4 menos r4 y guardar el resultado en r4. Su equivalente en lenguaje Assembly del x86 es *mov r4, 0*.

(d) *daddi r5, r5, -1.*

La tarea que realiza esta instrucción es sumar r5 más -1 y guardar el resultado en r5. Su equivalente en lenguaje Assembly del x86 es *dec r5*.

(e) *xori r6, r6, 0xffffffffffffffff.*

La tarea que realiza esta instrucción es hacer un XOR entre r6 y ffffffffffffffff y guardar el resultado en r6. Su equivalente en lenguaje Assembly del x86 es *XOR r6, 0FFFFh*.

Ejercicio 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)
        ld r2, B(r0)
        sd r2, A(r0)
        sd r1, B(r0)
        halt
```

(a) *Ejecutar 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 (o pipeline) y por qué?*
- *¿Qué tipo de 'stall' es el que aparece?*
- *¿Cuál es el promedio de Ciclos Por Instrucción (CPI) en la ejecución de este programa bajo esta configuración?*

La instrucción que está generando atascos (*stalls*) en el cauce (o *pipeline*) es *sd r2, A(r0)* (en su etapa ID) porque necesita que la instrucción *ld r2, B(r0)* finalice su etapa WB.

El tipo de '*stall*' que aparece es RAW (*Read After Write*).

El promedio de Ciclos Por Instrucción (CPI) en la ejecución de este programa bajo esta configuración es 2,2.

(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.*

En este caso, no se presenta ningún atasco porque el dato contenido en el registro R2 podrá ser leído por la instrucción *sd r2, A(r0)* (en su etapa MEM) cuando la instrucción *ld r2, B(r0)* se encuentra finalizando su etapa MEM, es decir, la instrucción *sd r2, A(r0)*

(en su etapa ID) no tiene que esperar a que la instrucción *ld r2, B(r0)* finalice su etapa WB, por lo que no aparecen atascos del tipo RAW.

El color de los registros en la ventana Register durante la ejecución indica que el dato (registro R1) está disponible en etapa MEM para adelantamiento. Además, los registros pueden tener color rojo, indicando que el resultado está disponible en EX y puede ser adelantado. Si el color es gris, el valor no está disponible en este ciclo para adelantamiento.

El promedio de Ciclos Por Instrucción (CPI) en la ejecución de este programa, bajo esta configuración, es 1,8, menor que el anterior.

Ejercicio 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.*

Se presentan atascos tipo RAW porque la instrucción *bnez r2, loop* necesita, en su etapa ID, del contenido del registro R2, que está siendo utilizado por la instrucción *daddi r2, r2, -1* (en su etapa EX).

Branch Taken significa que se produjo una incorrecta ejecución de la instrucción siguiente a una instrucción condicional y se produce porque la condición a evaluar tarda algunos ciclos en ser ejecutada, mientras que, durante esos ciclos, siguen entrando nuevas instrucciones al *pipeline*. Luego de evaluada la condición, si la instrucción posterior a ésta que se ejecutó no es la que debía ser ejecutada, su ejecución se trunca y se ejecuta la que está en el lugar de memoria indicada por la etiqueta en la instrucción condicional.

La ejecución de este programa tiene 1,75 CPI (21 ciclos y 12 instrucciones).

(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.*

Las instrucciones que generan los atascos tipo RAW son *dsll r1, r1, 1* (en su etapa ID) y *bnez r2, loop* (en su etapa ID). La primera porque necesita que la instrucción *ld r1, A(r0)* finalice su etapa WB y la segunda porque necesita que la instrucción *daddi r2, r2, -1* finalice su etapa WB. En el primer caso, el atasco se produce durante 1 ciclo y, en el segundo caso, durante 2 ciclos.

La cantidad de ciclos que dura el atasco *Branch Taken Stalls* en cada vuelta del lazo *"loop"* es 2. La diferencia con la ejecución con *Forwarding* se debe a que, en ese caso, el dato contenido en el registro R2 podrá ser leído por la instrucción *bnez r2, loop* (en su etapa ID) cuando la instrucción *daddi r2, r2, -1* se encuentra finalizando su etapa MEM y no su etapa WB (como sucede sin *Forwarding*).

En este caso, la ejecución del programa tiene 2,083 CPI (25 ciclos y 12 instrucciones).

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

```
        .data
A:      .word 1
B:      .word 3

        .code
        ld r2, B(r0)
        ld r1, A(r0)
LOOP:   daddi r2, r2, -1
        dsll r1, r1, 1
        bnez r2, LOOP
        halt
```

(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?*

```
        .data
A:      .word 1
B:      .word 3
C:      .word 0, 0, 0

        .code
        ld r2, B(r0)
        ld r1, A(r0)
        daddi r3, r0, 0
LOOP:   sd r1, C(r3)
        dsll r1, r1, 1
        daddi r2, r2, -1
        daddi r3, r3, 8
```

```
        bnez r2, LOOP
        halt
```

Los elementos de la tabla que se genera hacen referencia al número decimal que representa R1 previo a cada uno de los tres corrimientos hacia la izquierda.

Ejercicio 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?*

La tarea que realiza este programa es buscar el número 7 en la tabla con 10 números y, al encontrarlo, poner el valor 1 en el registro R10.

(b) *Re-ejecutar el programa con la opción Configure/Enable Branch Target Buffer habilitada. Explicar la ventaja de usar este método y cómo trabaja.*

La ventaja de utilizar la opción *Branch Target Buffer* es reducir a 4 los atascos tipo BTS (*Branch Taken Stall*). Esta opción carga la dirección del último salto. Es un algoritmo de predicción para cargar la próxima instrucción. Si nunca se ejecutó, carga la siguiente instrucción, sino carga instrucción de la tabla, la cual se actualiza cuando sucede un atasco tipo BTS. Tener en cuenta que esta opción es útil cuando aumenta la cantidad de iteraciones de un lazo.

(c) *Confeccionar una tabla que compare número de ciclos, CPI, RAWs y Branch Taken Stalls para los dos casos anteriores.*

	Inciso (a)	**Inciso (b)**
Número de ciclos	71	67
CPI	1,651	1,558
RAWs	16	16
Branch Taken Stalls	8	4

Ejercicio 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)?*

El efecto que tiene habilitar la opción *Delay Slot* (salto retardado) es saltar un ciclo después, por lo que ejecuta siempre la instrucción siguiente al salto y hay 0 atascos tipo BTS siempre.

(b) *¿Con qué fin se incluye la instrucción NOP? ¿Qué sucedería si no estuviera?*

La instrucción NOP se incluye con el fin de no modificar el funcionamiento del programa, como solución simple al *Delay Slot*. Si no estuviera, el programa finalizaría a causa del HLT.

(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.*

```
            .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)
        bnez r2, LOOP
        daddi r1, r1, 8
        halt
```

	Inciso (a)	**Inciso (d)**
Número de ciclos	63	55
Instrucciones	59	51
CPI	1,068	1,078

Ejercicio 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.

```
            .data
A:          .word 1
B:          .word 2
C:          .word 3
D:          .word 0

            .code
            ld r1, A(r0)
            ld r2, B(r0)
            ld r3, C(r0)
            dadd r4, r0, r0
            bne r1, r2, NOIGUAL1
            daddi r4, r4, 1
NOIGUAL1:   bne r1, r3, NOIGUAL2
            daddi r4, r4, 1
            j fin2
NOIGUAL2:   bnez r4, FIN2
            bne r2, r3, FIN3
            daddi r4, r4, 2
            j FIN3
FIN1:       beqz r4, FIN2
FIN2:       daddi r4, r4, 1
FIN3:       sd r4, D(r0)
            halt
```

Ejercicio 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.

```
            .data
TABLA:      .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
RES:        .word 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
X:          .word 5
TAM:        .word 10
CANT:       .word 0

            .code
            ld r1, TAM(r0)
            ld r2, X(r0)
            dadd r3, r0, r0
            dadd r4, r0, r0
            daddi r5, r0, 1
            daddi r2, r2, 1
LAZO:       ld r6, TABLA(r3)
            slt r7, r6, r2
            bnez r7, MENOR
            sd r5, RES(r3)
            daddi r4, r4, 1
MENOR:      daddi r1, r1, -1
            daddi r3, r3, 8
            bnez r1, LAZO
            sd r4, CANT(r0)
            halt
```

Ejercicio 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.

```
        .data
NUM1:   .word 1
NUM2:   .word 2
RES:    .word 0

        .code
        ld r1, NUM1(r0)
        ld r2, NUM2(r0)
        dadd r3, r0, r0
LAZO:   daddi r2, r2, -1
        dadd r3, r3, r1
        bnez r2, LAZO
        sd r3, RES(r0)
        halt
```

Ejercicio 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.

```
            .data
A:          .word 5
X:          .word 0
Y:          .word 5

            .code
            ld r1, A(r0)
            ld r2, X(r0)
            ld r3, Y(r0)
LAZO:       beqz r1, FIN
            daddi r1, r1, -1
            dadd r2, r2, r3
            j LAZO
FIN:        sd r2, X(r0)
            halt
```

Ejercicio 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
```

```
            .data
CADENA:     .asciiz  "adbdcdedfdgdhdid"
CAR:        .asciiz  "d"
CANT:       .word 0

            .code
            dadd r1, r0, r0
            dadd r2, r0, r0
            lbu r3, CAR(r0)
LAZO:       lbu r4, CADENA(r1)
            beqz r4, FIN
            bne r3, r4, NOIGUAL
            daddi r2, r2, 1
NOIGUAL:    daddi r1, r1, 1
            j LAZO
FIN:        sd r2, CANT(r0)
            halt
```