UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

ORGANIZACIÓN DE COMPUTADORES

LABORATORIO 2

Profesores: Alfonso Guzmán & Daniel Wladdimiro
Ayudantes: Pablo Ulloa & Ariel Undurraga

Santiago - Chile
7 de abril de 2017

CAPÍTULO 1. CONTEXTO

Como lo confirma la teoría vista en cátedra, en la organización de computadores lo esencial es el camino de datos (*datapath* en inglés), el cual consiste en un conjunto de unidades funcionales (unidad de control, ALU, multiplexores, buses, registros, etc), donde sin la existencia de estos elementos, no sería posible ejecutar las instrucciones que se les dan a un computador, es decir, no existirían los computadores como los conocemos hoy en día.

Este camino de datos, como su nombre lo sugiere, representa el camino que recorren las señales eléctricas a través de las distintas unidades funcionales, con el fin de realizar una instrucción específica. Es por esto que cada instrucción básica, como las del lenguaje MIPS, utiliza un conjunto acotado de unidades del camino de datos, que se pueden evidenciar en las señales activas de la unidad de control. Ante esto surge la siguiente interrogante: ¿Qué unidades funcionales se activan al ejecutar un determinado conjunto de instrucciones?

CAPÍTULO 2. INSTRUCCIONES

El programa a construir debe leer dos archivos de entrada, que contendrán instrucciones de un código en MIPS y los valores de cada uno de los registros. Además debe ser capaz de generar dos archivos de salida, el primero entrega la traza de los registros de las instrucciones de entrada por cada ciclo de reloj, y el segundo corresponde a los valores en los buffer en cada ciclo de reloj. Para efectos de este laboratorio, debe considerar que el camino de datos está siendo ejecutado en un procesador optimizado con *forwarding*, con una predicción del branch `taken` (siempre se considera el branch como tomado) y que utiliza los componentes presentes en la imagen del Anexo.

De ser así, el primer archivo de salida debe contener todos los registros de la arquitectura MIPS, donde estos valores van a ir cambiando en cada ciclo de reloj según las instrucciones vayan entrando. En la Tabla 2.1 se observa un ejemplo de salida, donde se poseen todos los registros y cómo estos van cambiando en cada ciclo de reloj. Pues bien, recordando la teórica, nos podemos acordar que en la primera mitad del ciclo de reloj se escribe y en la segunda mitad se lee, por lo tanto, en la segunda mitad vamos a poseer un nuevo valor del registro en el caso que este se modifique, como sucede en la tabla con $v0, y $ra en el ciclo C_2 o $fp en el ciclo C_4. No es necesario colocar siempre los valores de la primera mitad y la segunda mitad, dado que si ambos son iguales, como sucede con los valores de $v0 en el ciclo C_3, no es necesario repetirlo dos veces.

Ciclo del reloj	PC	$zero	$at	$v0	...	$fp	$ra
C_1	0	0	0	3	...	`0x550400`	`0xafa234`
C_2	4	0	0	3 / 5	...	`0x550400`	`0xafa234` / `0x02555ae`
C_3	8	0	0	5	...	`0x550400`	`0x02555ae`
C_4	12	0	0	5	...	`0x550400` / `0x550500`	`0x02555ae`
C_5	30	0	0	5	...	`0x550500`	`0x02555ae`

Cuadro 2.1: Ejemplo del primer archivo de salida.

Por otra parte, el segundo archivo de salida debe contener los valores de los buffers del camino de datos en cada ciclo de reloj, los cuales están detallados en el Anexo. En la Tabla 2.2 se muestra un ejemplo de los valores de los buffer, el cual es referencial, por lo tanto, en caso de requerir más datos en los buffers, estos pueden ser agregados. Como observamos, se muestra la referencia a la instrucción que hace alusión a los datos guardos en el buffer, de tal manera de tener una guía para la visualización de la traza de estos.

Ciclo del reloj	IF/ID	ID/EX	EX/MEM	MEM/WB
C_1	add $t0, $t1, $t2	sub $v0, $a1, $a2	lw $t3, 40($t4)	and $v1, $s3, $s4
	Add_PC=0x800 add $t0, $t1, $t2	Add_PC=0x796 ALUSrc=0 ALUOp=10 RegDst=1 Branch=0 MemWrite=0 MemRead=0 RegWrite=1 MemToReg=0 Read_data_1=8 Read_data_2=8 Sign-extend=0 Rs=$a1 Rt=$a2 Rd=$v0	Branch=0 MemWrite=0 MemRead=1 RegWrite=1 MemToReg=1 Zero=0 ALU_Result=0x2328af Read_data_2=0 Add_result=0 Mux_RegDst=$t3	RegWrite=1 MemToReg=0 Read_data=0 ALU_Result=9 Mux_RegDst=$v1
C_2	lw $s3, 0($s4)	add $t0, $t1, $t2	sub $v0, $a1, $a2	lw $t3, 40($t4)
	Add_PC=0x804 lw $s3, 0($s4)	Add_PC=0x800 ALUSrc=0 ALUOp=10 RegDst=1 Branch=0 MemWrite=0 MemRead=0 RegWrite=1 MemToReg=0 Read_data_1=7 Read_data_2=8 Sign-extend=0 Rs=$t1 Rt=$t2 Rd=$t0	Branch=0 MemWrite=0 MemRead=0 RegWrite=1 MemToReg=0 Zero=1 ALU_Result=0 Read_data_2=8 Add_result=0 Mux_RegDst=$v0	RegWrite=1 MemToReg=1 Read_data=16 ALU_Result=0x2328af Mux_RegDst=$t3
C_3	beq $t7, $t8, LABEL	lw $s3, 0($s4)	add $t0, $t1, $t2	sub $v0, $a1, $a2
	Add_PC=0x808 beq $t7, $t8, LABEL	Add_PC=804 ALUSrc=1 ALUOp=00 RegDst=0 Branch=0 MemWrite=0 MemRead=1 RegWrite=1 MemToReg=1 Read_data_1=0x40032a Read_data_2=0 Sign-extend=0 Rs=$s4 Rt=$s3 Rd=0	Branch=0 MemWrite=0 MemRead=0 RegWrite=1 MemToReg=0 Zero=0 ALU_Result=15 Read_data_2=8 Add_result=0 Mux_RegDst=$t0	RegWrite=1 MemToReg=0 Read_data=0 ALU_Result=0 Mux_RegDst=$v0

Cuadro 2.2: Ejemplo del segundo archivo de salida.

Cabe destacar que el ejemplo anterior no se utilizó la técnica de *forwarding*, por lo que se explicará a continuación los pasos a seguir para su uso.

En el pipeline de 5 etapas que se ha utilizado, en la etapa EX es donde debe aplicarse la técnica de *forwarding*, de tal manera de modificar los valores de los registros leídos. Las condiciones que deben cumplirse para que haya un hazard de dato son:

```
if (EX/MEM.RegWrite=1)
and (EX/MEM.Mux_RegDst != 0)
and (EX/MEM.Mux_RegDst = ID/EX.Rs)

if (EX/MEM.RegWrite=1)
and (EX/MEM.Mux_RegDst != 0)
and (EX/MEM.Mux_RegDst = ID/EX.Rt)

if (MEM/WB.RegWrite=1)
and (MEM/WB.Mux_RegDst != 0)
and (MEM/WB.Mux_RegDst = ID/EX.Rs)

if (MEM/WB.RegWrite=1)
and (MEM/WB.Mux_RegDst != 0)
and (MEM/WB.Mux_RegDst = ID/EX.Rt)
```

En la Tabla 2.3 se observa un ejemplo, donde se cumple una de las condiciones mencionadas, ya que el registro `$t3` es el mismo en el valor de `Mux_RegDst` del buffer `EX/MEM` y en `Rs` del buffer `ID/EX`. De tal manera, que el hazard de datos es detectado y el valor debe ser actualizado al momento de realizar la operación aritmética. Es decir, el valor que se guardará en `$t3` en C_1 para la instrucción `add $t3, $t2, $t1` y que está almacenados en el buffer `EX/MEM`, específicamente en el registro `ALU.Result`, debe ser actualizado para realizar la operación de `sub $v0, $t3, $t4` correctamente. En caso de que no se aplicara *forwarding*, el resultado almacenado en el ciclo C_2 dentro de `ALU_Result` en el buffer de `EX/MEM` no sería 5, sino 0, dado que los valores no serían modificados.

En el caso, que se necesite agregar un `NOP` (utilizando el forwarding), se deberá cumplir la siguiente condición:

```
if (ID/EX.MemRead=1)
and ((ID/EX.Rt=IF/ID.Rs)
or (ID/EX.Rt=IF/ID.Rt))
    stall the pipeline
```

De esta manera, cuando se realice la espera, todos los valores de la línea de control deberán valer 0, para que no se realice ninguna acción y se espere un ciclo para realizar la acción.

Ciclo del reloj	IF/ID	ID/EX	EX/MEM	MEM/WB
C_1	`add $t0, $t1, $t2`	`sub $v0, $t3, $t4`	`add $t3, $t2, $t1`	`and $v1, $s3, $s4`

	`Add_PC=0x800`
	`add $t0, $t1, $t2`
		`Add_PC=0x796`
		`ALUSrc=0`
		`ALUOp=10`
		`RegDst=1`
		`Branch=0`
		`MemWrite=0`
		`MemRead=0`
		`RegWrite=1`
		`MemToReg=0`
		`Read_data_1=8`
		`Read_data_2=8`
		`Sign-extend=0`
		`Rs=$t3`
		`Rt=$t4`
		`Rd=$v0`
			`Branch=0`
			`MemWrite=0`
			`MemRead=0`
			`RegWrite=1`
			`MemToReg=0`
			`Zero=0`
			`ALU_Result=13`
			`Read_data_2=2`
			`Add_result=0`
			`Mux_RegDst=$t3`
				`RegWrite=1`
				`MemToReg=0`
				`Read_data=0`
				`ALU_Result=9`
				`Mux_RegDst=$v1`

C_2	`lw $s3, 0($s4)`	`add $t0, $t1, $t2`	`sub $v0, $t3, $t4`	`add $t3, $t2, $t1`

	`Add_PC=0x804`
	`lw $s3, 0($s4)`
		`Add_PC=0x800`
		`ALUSrc=0`
		`ALUOp=10`
		`RegDst=1`
		`Branch=0`
		`MemWrite=0`
		`MemRead=0`
		`RegWrite=1`
		`MemToReg=0`
		`Read_data_1=7`
		`Read_data_2=8`
		`Sign-extend=0`
		`Rs=$t1`
		`Rt=$t2`
		`Rd=$t0`
			`Branch=0`
			`MemWrite=0`
			`MemRead=0`
			`RegWrite=1`
			`MemToReg=0`
			`Zero=0`
			`ALU_Result=5`
			`Read_data_2=8`
			`Add_result=0`
			`Mux_RegDst=$v0`
				`RegWrite=1`
				`MemToReg=0`
				`Read_data=0`
				`ALU_Result=13`
				`Mux_RegDst=$t3`

Cuadro 2.3: Ejemplo de hazard.

Supongamos que poseemos la siguientes dos instrucciones y realicemos la traza correspondiente:

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

En la Tabla 2.4 se verifica que se cumpla la condición mencionada anteriormente, es decir, se analiza que los registros sean iguales y de esta manera se establezcan las líneas de control en 0 para el siguiente ciclo.

Por otra parte, en el caso de existir un hazard de control, se debe realizar un `flush` a los buffer para aquellas instrucciones que no debieron ser planificadas, estableciendo todos sus valores en 0.

Cabe destacar que esto es una sugerencia y **no** la forma en como deben entregar el archivo de salida, el cual queda a criterio de cada persona. Y también no se han incluido **todos** los registros por motivos de espacio, lo que si **debe** hacerse en el primer archivo de salida.

Ciclo del reloj	IF/ID	ID/EX	EX/MEM	MEM/WB
C_1	add $t3, $t0, $t2	lw $t0, 4($t1)	Empty	Empty
	Add_PC=0x404 add $t3, **$t0**, $t2	Add_PC=0x400 ALUSrc=1 ALUOp=00 RegDst=0 Branch=0 MemWrite=0 MemRead=1 RegWrite=1 MemToReg=1 Read_data_1=0x124000 Read_data_2=0 Sign-extend=4 Rs=$t1 **Rt=$t0**	Branch=0 MemWrite=0 MemRead=0 RegWrite=0 MemToReg=0 Zero=0 ALU_Result=0 Read_data_2=0 Add_result=0 Mux_RegDst=0	RegWrite=0 MemToReg=0 Read_data=0 ALU_Result=0 Mux_RegDst=0
C_1	add $t3, $t0, $t2	NOP	lw $t0, 4($t1)	Empty
	Add_PC=0x404 add $t3, $t0, $t2	Add_PC=0x800 ALUSrc=0 ALUOp=00 RegDst=0 Branch=0 MemWrite=0 MemRead=0 RegWrite=0 MemToReg=0 Read_data_1=7 Read_data_2=8 Sign-extend=0 Rs=$t1 Rt=$t2 Rd=$t0	Branch=0 MemWrite=0 MemRead=1 RegWrite=1 MemToReg=1 Zero=0 ALU_Result=0x124004 Read_data_2=0 Add_result=0 Mux_RegDst=$t0	RegWrite=1 MemToReg=0 Read_data=0 ALU_Result=13 Mux_RegDst=$t3

Cuadro 2.4: Ejemplo de hazard.

Para efectos de la evaluación, se entregarán 3 archivos de entrada distintos: un programa secuencial, un programa con una función recursiva y un programa con ciclos, los cuales estarán disponibles en Moodle.

Junto con el programa, usted debe entregar un informe que cumpla con el formato tesis del Departamento de Ingeniería Informática de la Universidad de Santiago de Chile. Para efectos prácticos, se dejará disponible una plantilla en LaTeX en el Moodle, la cual posee las secciones a evaluar en la entrega del laboratorio.

En el desarrollo del informe se evaluará la forma en cómo usted dio solución al enunciado de este laboratorio, es decir, cómo decodificó el archivo de entrada, cómo realizó el seguimiento de las trazas, y así mismo, cómo presenta los resultados en el archivo de salida.

2.1 EXIGENCIAS

- El programa debe estar escrito en C o C++ (estándar ANSI C). En caso de usar C++ se exigirá orien-

tación a objetos, de caso contrario no será revisado.

- El programa y el informe deben ser entregados como una carpeta comprimida en formato `zip` o `tar.gz`-
- El programa debe ser entregado en su código fuente junto a un archivo Makefile para su compilación.
- El programa debe tener usabilidad. El usuario debe ser capaz de ingresar el nombre del archivo a leer y los nombres de los archivos de salida por una interfaz.
- El programa debe cumplir con un mínimo de calidad de software (funciones separadas y nombres tanto de funciones como de variables, de fácil comprensión).
- El informe escrito no debe exceder 10 páginas de texto escrito, sin considerar portada ni índice, en caso contrario, por cada página extra, se descontará 5 décimas.
- El informe escrito debe ser entregado en formato PDF, por lo que puede ser desarrollado en LaTeX, Microsoft Word, OpenOffice Writer, etc.

2.2 RECOMENDACIONES

- Pueden usar estructuras de datos para almacenar la información de los registros y señales de control en caso de utilizar C, sino, debe utilizar un objeto para esto.
- Utilizar la plantilla de LaTeX disponible en el Moodle del curso.
- Consultar a los ayudantes y en Moodle del curso.

2.3 DESCUENTOS

- Por cada exigencia no cumplida, se descontarán dos décimas a la nota, a excepción las que ya mencionan su descuento.
- Por cada día de atraso, se descontará un punto a la nota.
- Por cada tres faltas ortográficas o gramaticales en el informe, se descontará una décima a la nota.
- Por cada falta de formato en el informe, se descontará una décima a la nota.

2.4 EVALUACIÓN

- La nota del laboratorio será el promedio aritmético del código fuente con el informe, y en caso de obtener una nota inferior a 4 en alguno de las dos calificaciones, se evaluará con la menor nota.
- En caso que no se entregue alguno de los dos, se evaluará con la nota mínima.

Este laboratorio debe ser entregado el día 28 de Abril del año 2017, hasta las 23:59 hrs. Los descuentos por atraso corren a contar de las 00:59 hrs del día 29 de Abril del año 2017

En caso de dudas o problemas en el desarrollo, comunicarse con su ayudante de laboratorio.

CAPÍTULO 3. ANEXOS

Camino de datos de ejemplo: