Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores (II/2018)

Tarea 9

Fecha de entrega: miércoles 14 de noviembre de 2018 a las 11:59 PM

Parte programación

En esta tarea, deberán simular la ejecución de un programa en el computador básico con *pipeline* visto en clases, con el objetivo de:

- Ver cómo cambia la cantidad de ciclos de ejecución con respecto a programas en el computador básico.
- Identificar y dar solución a *hazards* de datos y de control en tiempo de ejecución.

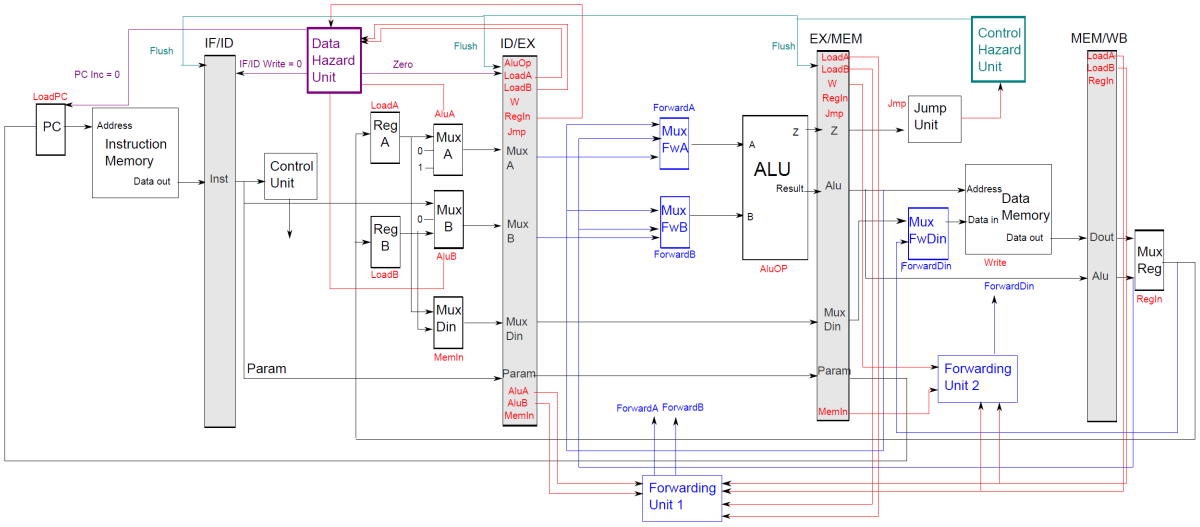

Figura 1: Arquitectura del computador básico con *pipeline*.

Al trabajar con esta arquitectura, se tiene una ISA reducida en comparación a la del computador básico dadas las limitantes generadas a favor del paralelismo. Las instrucciones a implementar, entonces, se encuentran plasmadas en la tabla 1.

Instrucción	Operandos	Operación	Condiciones	Ejemplo de uso
MOV	A,B	A=B		-
	B,A	B=A		-
	A,Lit	A=Lit		MOV A,15
	B,Lit	B=Lit		MOV B,15
	A,(Dir)	A=Mem[Dir]		MOV A,(var1)
	B,(Dir)	B=Mem[Dir]		MOV B,(var2)
	(Dir),A	Mem[Dir]=A		MOV (var1),A
	(Dir),B	Mem[Dir]=B		MOV (var2),B
	A,(B)	A=Mem[B]		-
	B,(B)	B=Mem[B]		-
	(B),A	Mem[B]=A		-
ADD	A,B	A=A+B		-
	B,A	B=A+B		-
	A,Lit	A=A+Lit		ADD A,5
SUB	A,B	A=A-B		-
	B,A	B=A-B		-
	A,Lit	A=A-Lit		SUB A,2
AND	A,B	A=A and B		-
	B,A	B=A and B		-
	A,Lit	A=A and Lit		AND A,15
OR	A,B	A=A or B		-
	B,A	B=A or B		-
	A,Lit	A=A or Lit		OR A,5
XOR	A,B	A=A xor B		-
	B,A	B=A xor B		-
	A,Lit	A=A xor Lit		XOR A,15
NOT	A,A	A=notA		-
	B,A	B=notA		-
SHL	A,A	A=shift left A		-
	B,A	B=shift left A		-
SHR	A,A	A=shift right A		-
	B,A	B=shift right A		-
INC	B	B=B+1		-
JMP	Dir	PC = Dir		JMP end
JEQ	Dir	PC = Dir	Z=1	JEQ label
JNE	Dir	PC = Dir	Z=0	JNE label
NOP	-	-		NOP

Tabla 1: ISA del computador básico con *pipeline*.

Input

Su programa recibirá como entrada un archivo `.txt` con el código escrito en el `Assembly` del computador básico con *pipeline*, siguiendo exclusivamente la `ISA` señalada anteriormente. A continuación, un ejemplo de un código válido:

```
;Segmento de datos.
DATA:
n    2
i    0
;Segmento de codigo.
CODE:
MOV B,(n)
MOV A,(i)
start:
ADD A,1
JNE start
MOV A,5
ADD A,B
MOV B,3
```

Debe tomar las siguientes consideraciones:

- Los comentarios ocupan solo una línea y parten con el caracter "`;`".
- Todo código partirá inicialmente con un segmento `DATA` para definir variables y un segmento `CODE` para el programa. Si no se definen variables, seguirá existiendo el segmento `DATA`, pero estará vacío.
- No se entregan los *opcodes* de la ISA dado que no los necesita para desarrollar su tarea.

Ejecución

Su programa debe tener como nombre `pipeline_simulator.py` y debe ser ejecutado por línea de comando de la siguiente forma:

```
C:\User\IIC2343\>python pipeline_simulator.py program.txt stats.txt
```

En este caso, `program.txt` corresponde a la ruta del archivo de entrada y `stats.txt` a la ruta del nuevo archivo de salida a generar.

Output

En tiempo de ejecución

Durante la ejecución, su simulación deberá **imprimir en consola** la instrucción que está siendo analizada y la acción a realizar, si corresponde. Debe seguir el siguiente formato:

`INSTRUCTION | ACTION`
`INSTRUCTION | ACTION 1 | ACTION 2` **(Si corresponde)**

IMPORTANTE: En esta simulación se debe considerar que el computador, frente a una instrucción de salto condicional, **decide no saltar.** Debe tener en cuenta la incidencia que esto puede tener en los ciclos de ejecución, considerando el envío de la señal de *flush*. Asuma, **en caso de necesitarlo**, que después de la última instrucción del programa ingresado sigue una cantidad infinita de instrucciones `NOP`. Por último, por simplicidad, puede suponer que los **saltos incondicionales se manejan con tres *stalling* consecutivos**.

Para el ejemplo anterior, entonces, su programa debe imprimir:

```
MOV B,(n) | NONE
MOV A,(i) | NONE
NOP       | STALLING
ADD A,1   | FORWARDING - MOV A,(i) - FORWARDING UNIT 1
JNE start | FORWARDING - ADD A,1 - FORWARDING UNIT 1
MOV A,5   | NONE
ADD A,B   | FORWARDING - MOV A,5 - FORWARDING UNIT 1
MOV B,3   | FLUSHING - MOV A,5; ADD A,B; MOV B,3
ADD A,1   | NONE
JNE start | FORWARDING - ADD A,1 - FORWARDING UNIT 1
MOV A,5   | NONE
ADD A,B   | FORWARDING - MOV A,5 - FORWARDING UNIT 1
MOV B,3   | NONE
```

Al término de la ejecución

Al terminar la ejecución de la simulación, su programa deberá escribir en `stats.txt` (archivo de salida entregado por línea de comando) la acción realizada en cada instrucción, en conjunto con las siguientes estadísticas:

- Número de ciclos de ejecución.
- Cantidad realizada de *forwardings* por unidad.
- Cantidad realizada de *stallings*.
- Cantidad realizada de *flushings*.

Para el ejemplo anterior, `stats.txt` debe ser:

```
===== ACTIONS =====
MOV B,(n) | NONE
MOV A,(i) | NONE
NOP       | STALLING
ADD A,1   | FORWARDING - MOV A,(i) - FORWARDING UNIT 1
JNE start | FORWARDING - ADD A,1 - FORWARDING UNIT 1
MOV A,5   | NONE
ADD A,B   | FORWARDING - MOV A,5 - FORWARDING UNIT 1
MOV B,3   | FLUSHING - MOV A,5; ADD A,B; MOV B,3
ADD A,1   | NONE
JNE start | FORWARDING - ADD A,1 - FORWARDING UNIT 1
MOV A,5   | NONE
ADD A,B   | FORWARDING - MOV A,5 - FORWARDING UNIT 1
MOV B,3   | NONE
===== STATS =====
cycles: 17
forwarding unit 1: 5
forwarding unit 2: 0
stalling: 1
flushing: 1
```

IMPORTANTE: Tanto para la impresión de resultados como para la escritura del archivo de salida, debe respetar el formato utilizado en el ejemplo anterior.

Bonus

Puede optar a un bonus de **hasta dos puntos** (a criterio del corrector) si elabora un *script* llamado `pipeline_optimizer.py` que reciba de *input* por línea de comando un código `Assembly` y retorne una versión equivalente reordenada de forma que tome menos ciclos de ejecución. Un ejemplo de uso a continuación:

```
C:\User\IIC2343\>python pipeline_optimizer.py program.txt new_program.txt
```

En este caso, `program.txt` corresponde al código original y `new_program.txt` será la salida correspondiente al código optimizado. Debe dejar este *script* dentro de una carpeta llamada **bonus** dentro de **la raíz del repositorio** de la tarea, además de dejar dentro de esta misma un archivo `.txt` de un programa que **pueda ser optimizado**, lo que se debe poder corroborar a partir de `pipeline_simulator.py`.

Si decide optar a esta bonificación, **debe** incluir detalles de su implementación en el `README` y comentar en este sus resultados.

Supuestos

Puede hacer uso de **supuestos** en casos límite o elementos que no hayan sido explicitados en el enunciado. No obstante, para que estos sean válidos durante la corrección, **debe** explicitarlo en su `README`.

No se aceptarán supuestos sobre criterios que hayan sido establecidos en el enunciado.

Entrega y evaluación

La tarea se debe realizar de **manera individual** y se entregará a través de GitHub. El formato de entrega corresponde a un *script* en Python 3.5 ó 3.6 llamado `pipeline_simulator.py`, que se debe encontrar en **la raíz del repositorio** junto con todos los archivos que sean necesarios para que su tarea funcione.

El repositorio subido debe contar con un archivo `README.md` escrito en *Markdown* que identifique sus datos y consideraciones que el corrector deba tomar en cuenta. Se espera **como mínimo** lo siguiente:

- Lo que sí se implementó en la tarea.
- Lo que no se implementó en la tarea.
- Supuestos de elementos no descritos en el enunciado.
- Explicación breve y concisa de su programa.

Si su `README` no posee todos los puntos indicados anteriormente, **su tarea no será considerada y deberá solicitar recorrección**. El `README` se puede subir hasta 24 horas después de la entrega. Si lo sube posterior al plazo de entrega establecido, debe crear una *issue* en el repositorio de su tarea indicándolo para que sea considerado en la corrección. Los archivos que no ejecuten o que no cumplan el formato de entrega establecido implicarán nota **1.0** en la tarea. En caso de atraso, se aplicará un descuento de **1.0** punto por cada 6 horas o fracción.

Política de Integridad Académica

Los alumnos de la Escuela de Ingeniería deben mantener un comportamiento acorde al Código de Honor de la Universidad:

"Como miembro de la comunidad de la Pontificia Universidad Católica de Chile me comprometo a respetar los principios y normativas que la rigen. Asimismo, prometo actuar con rectitud y honestidad en las relaciones con los demás integrantes de la comunidad y en la realización de todo trabajo, particularmente en aquellas actividades vinculadas a la docencia, el aprendizaje y la creación, difusión y transferencia del conocimiento. Además, velaré por la integridad de las personas y cuidaré los bienes de la Universidad."

En particular, se espera que mantengan altos estándares de honestidad académica. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un procedimiento sumario. Específicamente, para los cursos del Departamento de Ciencia de la Computación, rige obligatoriamente la siguiente política de integridad académica. Todo trabajo presentado por un alumno (grupo) para los efectos de la evaluación de un curso debe ser hecho individualmente por el alumno (grupo), sin apoyo en material de terceros. Por "trabajo" se entiende en general las interrogaciones escritas, las tareas de programación u otras, los trabajos de laboratorio, los proyectos, el examen, entre otros. Si un alumno (grupo) copia un trabajo, los antecedentes serán enviados a la Dirección de Docencia de la Escuela de Ingeniería para evaluar posteriores sanciones en conjunto con la Universidad, las que pueden incluir reprobación del curso y un procedimiento sumario. Por "copia" se entiende incluir en el trabajo presentado como propio partes hechas por otra persona. Está permitido usar material disponible públicamente, por ejemplo, libros o contenidos tomados de Internet, siempre y cuando se incluya la cita correspondiente.