DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (I/2020)

Tarea 3

Caché y pipelining.
Fecha de entrega: 24 de junio, 2020. 18:00 horas.

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.

Objetivos

- Aprender a usar el tipo de dato `Decimal` en python.
- Aprender de la implementación de distintos protocolos de reemplazo en la *caché* por medio de programar una simulación.
- Aprender sobre la detección de hazards automática en la ejecución de un programa en el computador básico con pipeline.

Entrega y evaluación

Esta tarea es de carácter estrictamente **INDIVIDUAL**. Tiene dos tipos de preguntas evaluadas: programadas en *python* y teóricas de alternativas o similar disponibles en *canvas*. Para la pregunta del código de honor y las preguntas programadas, deberás enviar mediante un formulario de google:[1]

- Una imagen o PDF con tu respuesta a la pregunta del código de honor.
- Tus códigos en archivos `.py` independientes. **Sólo uno por pregunta.**

Debes referenciar correctamente[2] en caso de utilizar material externo. Tus programas serán corregidos vía *tests*, los *inputs* se darán por STDIN y debes entregar tus *outputs* por STDOUT, tal como en la tarea 1.

Importante: para evitar que se bloquée la corrección, se utilizará un *timeout* de 2 segundos de ejecución.

Código de honor

Escribe **a mano en un papel**[3] el siguiente texto completo y fírmalo. A continuación tómale una foto o escanéalo y súbelo al formulario. **No hacerlo equivale a un 1 en todas las preguntas de la tarea.**

> Tarea 3 - IIC2343
>
> Yo, <tu nombre>, afirmo que respetaré el código de honor de la universidad y no cometeré ninguna falta a la ética ni a la política de integridad académica.
>
> <número de alumno>
> <firma>

[1] Link completo:
https://docs.google.com/forms/d/e/1FAIpQLSe2m6Tqi2bfNlOk_O1BmT2rlq01hkxdOTnBmNpxPdn2jfRTHQ/viewform?usp=sf_link

[2] Si tienes dudas, usa el formato APA.

[3] Si te preocupas mucho por el medio ambiente, puedes escribirlo a mano en digital.

1. Pregunta recuperativa de la Tarea 1

1.1. Contexto

¿Sabías que la convención IEEE754 en realidad detalla **5** tipos de datos distintos? Estos son: binary32 (*float*), binary64 (*double*), binary128, decimal64 y decimal128. Los primeros 3 utilizan la notación científica en base 2 mientras que los dos últimos la usan en base 10. En esta pregunta utilizarás la implementación de python del tipo de dato `Decimal` en un caso aplicado.

1.2. Lectura recomendada

1. Implementación de python. Contiene la documentación de la librería Decimal de python.

1.3. Caso aplicado:

La ley de redondeo establece que si el monto (en pesos chilenos) termina en 1, 2, 3, 4 o 5, se redondea hacia la decena inferior, mientras que si termina en 6, 7, 8 o 9, se redondea hacia la decena superior.

Eres el encargado de programar el software contable de las tiendas DCComercio y tu jefe te pide que le calcules: ¿Cómo y cuánto cambiarían los ingresos mensuales si el redondeo hacia arriba fuera a partir de 5, como nos enseñaron en el colegio?

Además, el DCComercio pidió un crédito hace algunos años para abrir sus nuevos locales, por lo que deberás calcular (utilizando la fórmula del interés compuesto descrita más abajo), en cuánto tiempo más (en meses) conseguirían pagar el crédito (sólo con el ingreso mensual calculado redondeando 5 hacia abajo).

Considera que destinarán el 33 % de sus ingresos mensuales de manera fija para ello. Utiliza el ingreso mensual que calcules y asume que se mantendrá estable en el tiempo.

Los montos se trabajarán en **decenas de pesos**, eso quiere decir que la cantidad `Decimal("421.5")` representa el monto "cuatro mil doscientos quince". Utiliza las 28 cifras de precisión que vienen por defecto.

1.3.1. Cálculo del interés compuesto

Supongamos que el monto del crédito que queda pendiente es x, la tasa de interés es i y el ingreso mensual destinado al pago del crédito es m.

Entonces, en cada mes, lo que se hará es pagar m pesos descontándolo del monto x y luego el monto resultante se multiplicará por $1 + i$ y esa será la cantidad que quede para el mes siguiente. Expresado matemáticamente:

$$\text{queda para el mes siguiente} = (x - m) \cdot (1 + i)$$

Como consideración adicional, existe un punto de equilibrio tal que pagar esa cantidad mantendrá el monto que queda para el pago y, si se paga menos, el monto que queda aumentará. El punto de equilibrio es:

$$m = \frac{x \cdot i}{1 + i}$$

1.4. *Inputs:*

Recibirás a través de STDIN:

1. El número de compras N que han realizado los clientes ese mes.
2. N líneas, cada una con el detalle de una compra. Se utilizará el formato:

   `k cantidad producto precio_unitario cantidad producto precio_unitario ....`

   Donde `k` indica el número de productos distintos que ha comprado ese cliente.
3. El monto que queda del crédito.
4. La tasa de interés.

Ejemplo:[4]

```
3
2 1 tomate 100.4 6 cerveza 659.9
4 8 tomate 803.2 6 huevo 435.0 2 salchicha 113.2 4 salmón 1298.0
1 1 toallitas_de_bebés 100.7
60000.0         // seiscientos mil pesos
0.02            // esto indica que es un 2% mensual
```

1.5. *Outputs:*

Deberás entregar a través de STDOUT:

1. El ingreso mensual redondeando el 5 hacia abajo.
2. El número de meses que faltan para terminar de pagar el crédito, redondeando el 5 hacia abajo. (Ver fórmula descrita más arriba)
3. El ingreso mensual redondeando el 5 hacia arriba.

Si con el ingreso mensual ves que no es posible terminar de pagar el crédito, debes imprimir el texto `"NUNCA"` en lugar de la cantidad de meses.

Ejemplo:

```
18614   // redondeando el 5 hacia abajo
11      // total de meses
18615   // redondeando el 5 hacia arriba
```

[4] Lo que está con `//` son comentarios para la explicación, no forman parte del *input*.

2. Preguntas de la Tarea 3

Información general

Recibirás la ejecución de un programa por STDIN con un formato como el siguiente:

```
13
 N° | PC    Instrucción [RegA|RegB]
---------------------------------
0001|0000   MOV B,120  [0000|0078]
0002|0001   MOV (0),B  [0000|0078]
0003|0002   MOV B,34   [0000|0022]
0004|0003   MOV (1),B  [0000|0022]
0005|0004   MOV B,0    [0000|0000]
0006|0005   MOV (2),B  [0000|0000]
0007|0006   MOV B,255  [0000|00FF]
0008|0007   MOV (3),B  [0000|00FF]
0009|0008   MOV B,0    [0000|0000]
0010|0009   MOV (4),B  [0000|0000]
0011|0010   MOV B,0    [0000|0000]
```

La primera linea significa la cantidad de lineas siguientes que tiene el STDIN. Ten en cuenta que esto incluye a la linea del *header* y la linea divisoria.

Librería del *parser*

Les entregaremos un código para que puedan hacer *parse* de los inputs de las preguntas 1.1, 1.2 y 1.3. este sera una librería en formato **.py**. Este archivo se encontrará en la carpeta de la Tarea 3, junto con este enunciado.
Nota: Si desean revisar la ayuda de la libreria, pueden utilizar el siguiente comando en su codigo de python:

```python
import nombre_libreria


help(nombre_libreria)
```

2.1. Pipeline I

En base a la información obtenida del STDIN, debes analizar los datos e imprimir los **hazards de datos** ocurridos, indicando el número del clock de la ejecución que te pasamos, con qué instrucción existe el conflicto (identificándola según su número de clock) y qué *forwarding unit* se usa para resolverlo (en caso de necesitar hacer stalling, también debes indicarlo). Para esto, también puedes asumir lo siguiente:

- No hubo hazards de control.
- No se encuentran presentes los NOPs por hacer stalling.

Formato de output

Deberás imprimir en una linea, los valores que pedimos al inicio de la pregunta. Todos los valores deben ir separados por comas.

```
nro_clock,nro_clock_instruccion,forw_unit
```

Variables:

- **nro_clock (int)**: el número del clock de la ejecución.
- **nro_clock_instruccion (int)**: Instrucción en la que existe el conflicto.
- **forw_unit (str)**: La *forwarding unit* que se usa para resolver el conflicto. Puede ser "fw1" o "fw2"

IMPORTANTE! El número de clock es **distinto** al número del Program Counter.

2.2. Pipeline II

Con la misma información del SDTIN, ahora deberás imprimir los **hazards de control** ocurridos, indicando el número del clock en que se detecta cada hazard, además de imprimir el número de instrucciones perdidas en total. Para esto, toma las siguientes consideraciones:

- Solo están presentes los ciclos **no borrados** al hacer *flush*.
- No hubo hazards de control "anidados".
- El contador de hazards aumenta **siempre** que haya un salto incondicional (es decir, la ejecución especulativa siempre se equivoca).

Formato de output

Deberás imprimir el valor del clock al momento de detectar el hazard, por cada hazard que encuentres. Luego, después de detectar todos los hazards, deberás imprimir la cantidad de instrucciones que se perdieron.

```
nro_clock               # Primer hazard detectado
.
.
.
nro_clock               # Último hazard detectado
instrucciones_perdidas  #Cantidad de instrucciones perdidas
```

Variables:

- **nro_clock (int)**: el número del clock de la ejecución.
- **instrucciones_perdidas (int)**: Número de instrucciones perdidas en total.

IMPORTANTE! El número de clock es **distinto** al número del Program Counter.

2.3. Caché

En base a la información obtenida del STDIN, debes imprimir la secuencia de accesos a la memoria de datos y estadísticas de uso de la caché, comparando los protocolos **FIFO** y **LRU**, en una caché de las siguientes caracteristicas:

- 2-way associative (es decir, conjuntos de 2 bloques cada uno).
- Bloques de 4 palabras.
- 2 Conjuntos en total.
- Con un total de 16 palabras de memoria (*bytes*).

Como output de esta pregunta, debes utilizar la función `print_chache`, de la librería que te daremos, cada vez que intentes un acceso. Te recomendamos revisar con `help(print_cache)` los argumentos que necesita la función para poder utilizarla. Finalmente, al terminar todos los accesos, debes imprimir las estadísticas de uso de la caché (**hits y misses**).
Importante! Recuerda que debes imprimir esto, tanto para FIFO como para LRU.
A modo de ejemplo, haremos la siguiente secuencia de accesos:

$$0, 13, 15, 3, 0, 7, 6$$

```
[0, [0, 1, 2, 3, None, None, None, None, None, None, None, None, None, None, None, None], "M"]
[13, [0, 1, 2, 3, None, None, None, None, 12, 13, 14, 15, None, None, None, None], "M"]
[15, [0, 1, 2, 3, None, None, None, None, 12, 13, 14, 15, None, None, None, None], "H"]
[13, [0, 1, 2, 3, None, None, None, None, 12, 13, 14, 15, None, None, None, None], "H"]
[0, [0, 1, 2, 3, None, None, None, None, 12, 13, 14, 15, None, None, None, None], "H"]
[7, [0, 1, 2, 3, None, None, None, None, 12, 13, 14, 15, 4, 5, 6, 7], "M"]
[6, [0, 1, 2, 3, None, None, None, None, 12, 13, 14, 15, 4, 5, 6, 7], "H"]
4
3
```

Nota: En este ejemplo, primero se imprime la cantidad de **hits** y luego se imprime la cantidad de **misses**.

2.4. Bonus

Además de la ejecución del programa, recibirás el código fuente del programa. Deberás:

1. Extender la ejecución del programa añadiendo los NOP de hacer stalling y las instrucciones perdidas por hacer flush. Ante saltos condicionales siempre se equivoca al predecir los saltos.
2. Los hazards de control considerando la ejecución extendida.
3. Los hazards de datos, considerando la ejecución extendida.

2.4.1. Formato de salida:

Usando el mismo formato de STDIN, en la parte del número de *clock*, debes indicar si se hace *flush* (`flus`), *satlling* (stal) o *stalling* dentro de un *flush* (`flst`). A continuación, te mostramos un ejemplo con el formato de salida pedido.

```
 N° | PC    Instrucción [RegA|RegB]
---------------------------------
0036|0032   MOV B,128   [0079|0080]
0037|0033   AND A,B     [0000|0080]
0038|0034   JEQ 43      [0000|0080]
flus|0035   JMP 48      [0000|0080]
flus|0048   MOV B,(8)   [0000|----]  // no importa lo que escribamos en memoria, se va a borrar
flst|----   NOP         [0000|----]  // Aquí estamos haciendo staling dentro de un flush
0039|0043   MOV B,(8)   [0000|0021]
stal|----   NOP         [0000|0021]
0040|0044   MOV (5),B   [0000|0021]
0041|0045   MOV A,(9)   [0079|0021]
stal|----   NOP         [0079|0021]
0042|0046   MOV (4),A   [0079|0021]
```

Reglas para optar al bonus

1. Tener promedio 6 o superior entre las 3 preguntas de la tarea.

¿Qué hace el bonus?

Una vez se calculan las 7 notas que se quedan del promedio de tareas, reemplaza la peor de todas.

Anexo: paso de datos por STDIN

Para evitar tener que escribir constantemente o copiar y pegar los valores de entrada, los programadores ancestrales crearon una técnica milenaria para entregar a un proceso el contenido almacenado en un archivo por STDIN: el operador "<". A continuación les traspasaré los conocimientos de esta técnica arcana.

Supongamos que mi código está en el archivo `solucion_decimal.py` y tengo mi archivo `test.txt` con los valores de entrada del ejemplo que tengo más arriba.

Linux, MacOS y CMD:

Es tan simple como ejecutar en la consola:

```
$ python3 "./solucion_decimal.py" < "./test.txt"
```

El "$" representa el marcador de la línea.

PowerShell:

En la powershell tienen el operador reservado pero todavía no lo implementan (les hice una issue hace tiempo en el repositorio de GitHub). Sin embargo existe también una forma de hacerlo:

```
PS > get-content "./test.txt" | python3 "./solucion_decimal.py"
```

El "PS >" representa el marcador de la línea.