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

IIC2343 - Arquitectura de Computadores (I/2023)

Tarea 4

Respuestas sin desarrollo o justificación no tendrán puntaje.

Publicación: Miércoles 7 de Junio 19:00
Entrega: Viernes 30 de Junio 22:00

Instrucciones

- Lea atentamente el enunciado. Responda cada pregunta en un PDF por separado. Ponga su nombre y número de alumno/a en cada pregunta. Adicionalmente, debe incluir archivos de código con extensión `.py`, asegurando que el nombre de los archivos incluya el número y letra de la pregunta.
- La entrega de la tarea será a través de **canvas**.
- La tarea debe ser en formato **digital** y no manuscrito. De no cumplir esto, **no** se revisará dicha pregunta.
- Cualquier duda que tengan respecto a la tarea, preguntar en el foro de clases.
- **Cualquier uso de material externo al curso debe ser citado, incluyendo material de semestres pasados no incluido en canvas**.
- Para el uso de cupones de atraso, debe responder el siguiente formulario.
- Siga el código de honor.

Código de Honor de la UC

"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."

Pregunta 1: Memoria caché (30 ptos.)

Para esta parte, deberá implementar en Python 3.x un **simulador de accesos a memoria caché**. Este programa recibirá un archivo `.txt` como *input* a través de la línea de comandos y tendrá el siguiente formato:

```
CACHE_SIZE
LINE_SIZE
MEM_SIZE
CORR_FUNCTION
ACCESS_SEQUENCE
```

Donde cada variable indica lo siguiente:

- `CACHE_SIZE`: Tamaño **en bytes** de la memoria caché.
- `LINE_SIZE`: Tamaño **en bytes** de una línea de la caché.
- `MEM_SIZE`: Tamaño **en bytes** de la memoria principal.
- `CORR_FUNCTION`: Función de correspondencia a aplicar. Puede tener uno de los siguientes valores: `DM` (*Directly Mapped*), `FA` (*Fully Associative*), `2W` (2-*Way associative*).
- `ACCESS_SEQUENCE`: Secuencia de direcciones de memoria accedidas. Corresponde a una secuencia de números separados por una coma.

Un ejemplo concreto de *input* es el siguiente:

```
8
2
16
2W
0, 1, 5, 7, 10, 13, 4, 6
```

Este ejemplo representa una memoria caché de 8 bytes con líneas de 2 bytes (por ende, 4 líneas); una memoria principal de 16 bytes, una función de correspondencia 2-*way associative* y una secuencia de accesos a las direcciones 0, 1, 5, 7, 10, 13, 4, 6.

En respuesta, su programa deberá imprimir:

- La secuencia de accesos, con una letra al lado de cada número: una `H` si hubo *hit* y una `M` si hubo *miss*.
- *Hit rate* total de la memoria caché.
- Estado final de la memoria caché, incluyendo para cada línea: Índice de conjunto en bits (si corresponde), Índice de línea en bits, *tag* del contenido en bits y el bit de validez.

La impresión debe seguir el formato señalado a continuación:

```
0M, 1H, 5M, 7M, 10M, 13M, 4H, 6H
HR=0.375
ÍNDICE CONJUNTO | ÍNDICE LÍNEA | TAG | BIT VALIDEZ
0 | 00 | 11 | 1
0 | 01 | 01 | 1
1 | 10 | 01 | 1
1 | 11 | 10 | 1
```

Este es el *output* esperado para el ejemplo anterior. Ahora, se muestra el *input* y *output* esperado para el mismo ejemplo, pero con funciones de correspondencia *directly mapped* y *fully associative*:

```
# Input Directly Mapped
8
2
16
DM
0, 1, 5, 7, 10, 13, 4, 6
# Output
0M, 1H, 5M, 7M, 10M, 13M, 4M, 6H
HR=0.25
ÍNDICE LÍNEA | TAG | BIT VALIDEZ
00 | 0 | 1
01 | 1 | 1
10 | 0 | 1
11 | 0 | 1
```

```
# Input Fully Associative
8
2
16
FA
0, 1, 5, 7, 10, 13, 4, 6
# Output
0M, 1H, 5M, 7M, 10M, 13M, 4H, 6H
HR=0.375
ÍNDICE LÍNEA | TAG | BIT VALIDEZ
00 | 110 | 1
01 | 010 | 1
10 | 011 | 1
11 | 101 | 1
```

Notar que en estos ejemplos no se incluye el índice de conjunto al no ser utilizado en estas funciones. Adicionalmente, debe realizar los siguientes supuestos:

- Todas las líneas de la caché parten siendo inválidas (*i.e.* su bit de validez parte en 0).
- Todos los tamaños en bytes serán potencias de dos.
- Si corresponde, las cachés utilizan una política de reemplazo `LRU`.
- Todas las direcciones de la secuencia son válidas para la configuración de memoria entregada.
- El formato del archivo de *input* siempre será válido.
- No es necesario redondear ni truncar el *hit-rate*.
- Si una línea de la caché termina inválida, puede imprimir el símbolo "-" en lugar del *tag*.
- Solo se debe simular los accesos de memoria y copia de bloques de datos a través del almacenamiento del *tag*, no se debe simular el contenido "real" de la caché a partir de una memoria "real".
- No se utilizarán rutas de archivos que no existan para probar su tarea.

Distribución de puntaje: Para evaluar su programa, se utilizarán 2 configuraciones de memoria y secuencias de acceso distintas para probar cada función de correspondencia y el funcionamiento general de su simulación, es decir, se aplicarán 6 *tests* en total. Por cada *test* se entregará puntaje de la siguiente forma:

- **5 puntos** si el *test* pasa correctamente.
- **3 puntos** si el *test* presenta un error en la secuencia de accesos, *hit rate* o en el estado final de la caché.
- **1 punto** si presenta dos errores.
- **0 puntos** si presenta más de dos errores o si no corre.

Pregunta 2: Multiprogramación (15 ptos.)

Considere un sistema con páginas de 16KB, direcciones virtuales de 48 bits y direcciones físicas de 30 bits.

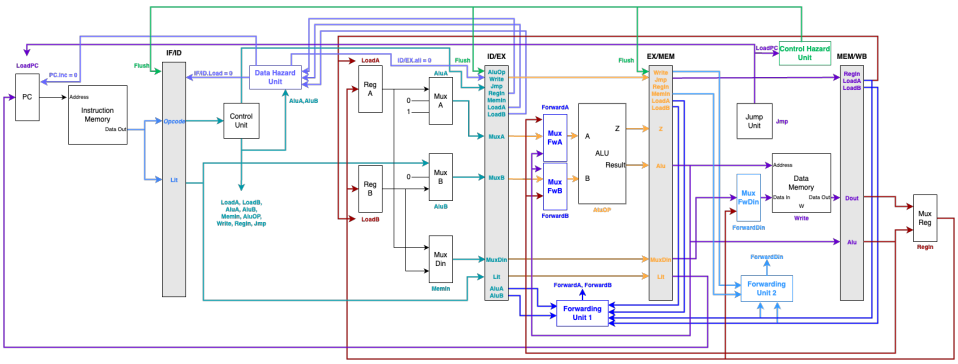
- Indique, en bytes, la cantidad máxima de memoria que puede direccionar cada proceso. **(1 punto)**
- Indique, en bytes, la cantidad máxima de memoria física que puede direccionar el sistema. **(1 punto)**
- Calcule el tamaño, en bytes, de una tabla de páginas de 1 nivel para este sistema, considerando 4 bits para *metadata* **(2 puntos)**.
- Proponga una modificación en el esquema de paginación que asegure que una tabla de páginas **quepa** en una página. **(6 puntos)**. Se otorgarán **4 puntos** si su propuesta presenta como máximo un error de implementación; **2 puntos** si posee al menos dos errores de implementación; y **0 puntos** si presenta más de dos errores o si no responde.
- Asuma que ahora se añade al esquema una TLB (*Translation Lookaside Buffer*) de 256 entradas. Indique la cantidad mínima de bits que requiere cada entrada para poder ser utilizada para realizar traducciones de memoria virtual a física **(3 puntos)** y el tamaño mínimo de esta memoria caché **(2 puntos)**. Se otorgarán **2 puntos** en total si presenta valores errados pero **cercanos** al esperado.

Para todos los ítems, **debe incluir la fórmula o explicación de cómo obtuvo el valor numérico**. Si solo incluye el tamaño, no se le otorgará puntaje en la pregunta. Por otra parte, puede utilizar las siguientes transformaciones numéricas para simplificar sus respuestas:

- 8 bits = 8b = 1 byte = 1B
- 2^{10}B = 1KB
- 2^{20}B = 2^{10}KB = 1MB
- 2^{30}B = 2^{10}MB = 1GB
- 2^{40}B = 2^{10}GB = 1TB

Pregunta 3: Paralelismo a Nivel de Instrucción (ILP) (15 ptos.)

Desarrolle las preguntas de esta sección a partir del diagrama del computador básico con *pipeline*:



- Modifique la arquitectura del computador básico con *pipeline* para usar la memoria de *stack* a través de las instrucciones `PUSH A`, `PUSH B`, `POP A` y `POP B`, ejecutándose en solo 5 ciclos cada una y actualizando el valor del registro `SP` en la etapa *Writeback* **(8 puntos)**. Puede ignorar los *hazards* que podrían darse en su implementación. Se descontará **1 punto** por cada conexión, señal faltante o error en su implementación. No es necesario que asigne un *opcode* a cada instrucción, pero sí debe incluir una tabla con la combinación de señales para cada una de ellas. Se descontarán **2 puntos** si no la incluye.
- Indique, con ejemplos, los tipos de *hazard* que podrían surgir de las instrucciones añadidas **(2 puntos)**. No necesita de la implementación anterior para responder este ítem, pero sí debe señalar las etapas en las que ocurren.
- A partir del siguiente programa del computador básico con *pipeline* y memoria de *stack*:

```
DATA:
  n 2
  res 0
CODE:
  MOV A,(n)
  MOV B,0
  PUSH B
  main:
    POP B
    JEQ A,B,end    // If A == B -> Salto a end.
    ADD B,1
    PUSH B
    JMP main
  end:
    ADD A,B
    MOV (res),A
```

Determine el número de ciclos que demora su ejecución detallando en un diagrama los estados del *pipeline* por instrucción. Asuma que posee la arquitectura para hacer *forwarding* para el manejo de *stack* de la misma manera que se resuelven los *hazards* de lectura y escritura en la memoria de datos; que el manejo de *stalling* es por *software* a través de la instrucción `NOP`; y que la unidad predictora de saltos asume que estos nunca se realizan, es decir, se incrementa el registro `PC` y se realiza *flushing* en la etapa *Memory* si es que la predicción fue incorrecta. Indique en el diagrama cuando ocurre *forwarding* (flechas de registro a etapa), *stalling* (con instrucción `NOP`) y *flushing* (con tachado en las instrucciones *flusheadas*) **(5 puntos)**.
Se descontará un punto por cada error o indicación faltante en su diagrama.

- **BONUS:** Modifique la arquitectura que realizó en el primer ítem para resolver **todos los *hazards*** que haya detectado, independiente de su tipo **(5 puntos)**. Se otorgará este puntaje adicional **solo** si su implementación funciona y los *hazards* detectados son correctos. Puede añadir unidades nuevas o modificar las ya existentes para el funcionamiento correcto de su modificación. Si realiza modificaciones, cuide que no afecten el funcionamiento del resto de instrucciones y detecciones de *hazards*.

Anexo: Material adicional

Con el fin de facilitar el desarrollo de su tarea, en canvas se incluye el siguiente material adicional dentro de la carpeta "Tarea4-Anexo":

- `cache-input-example-*.txt`: Ejemplos de *input* del enunciado para probar su implementación de la pregunta 1. **No serán los utilizados para la evaluación**.
- `T4-p1-utils.py`: *Script* en Python 3.x con funciones de utilidad que pueden ser de ayuda para el desarrollo de la pregunta 1. Incluye adicionalmente el código que permite leer un archivo de texto como *input* de la línea de comando.
- `T4-p3-diagrams.drawio.xml`: Diagrama de `draw.io` que incluye dos pestañas: "`CPU pipeline`", que posee el diagrama del computador básico con *pipeline* para facilitar el desarrollo del primer ítem de la pregunta 3; y "`Pipeline workflow`" que incluye el diagrama de cajas de la ejecución de un programa, para facilitar la elaboración del último ítem de la pregunta 3.