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

IIC2343 - Arquitectura de Computadores (I/2025)

Interrogación 3

Pauta de evaluación

Pregunta 1: Jerarquía de memoria (6 ptos.)

(a) (1 pto.) Indique y justifique qué principio de localidad es explotado con la lógica de bloques de memoria y líneas de caché en una jerarquía de memoria.

> **Solución:** La lógica de bloques de memoria y líneas de caché explota el **principio de localidad espacial**. Este principio señala que es "probable que datos cercanos al accedido también sean utilizados", por lo que se almacena en caché el contenido del bloque contiguo al que pertenece un dato y no solo este de forma individual. Se otorgan **0.5 ptos.** por indicar el principio correcto y **0.5 ptos.** por justificación. Si se señala el principio sin explicación alguna, **no se otorga puntaje**.

(b) (2 ptos.) Para estudiar el comportamiento de una caché *N-way associative* de L líneas a partir del valor de N, evalúa dos casos: (1) $N = 1$; (2) $N = L$. Para cada caso, explique cómo se comporta el mapeo de un bloque de memoria a una línea de caché. Si se asemeja al comportamiento de otra función de correspondencia conocida, indíquela y justifique por qué.

> **Solución:**
>
> (1) Si $N = 1$, cada conjunto es de una línea. Esto implica que existen tantos conjuntos como líneas haya en la caché. Como cada bloque se puede mapear a un solo conjunto, esto es equivalente a que se pueda mapear a una sola línea. Por lo tanto, su comportamiento se asemeja al de la función de correspondencia *directly mapped*.
>
> (2) Si $N = L$, cada conjunto es de L líneas. Esto implica que existe un solo conjunto que alberga todas las líneas de la caché. Como cada bloque se puede mapear a cualquier línea dentro de un conjunto, esto es equivalente a que se pueda mapear a cualquier línea de la caché. Por lo tanto, su comportamiento se asemeja al de la función de correspondencia *fully associative*.
>
> Por cada caso, se otorgan **0.5 ptos.** por explicar correctamente el comportamiento, y **0.5 ptos.** por indicar la función de correspondencia a la que se asemeja. Si solo se indica la función de correspondencia similar sin ninguna justificación, **no se otorga puntaje**.

(c) (3 ptos.) Suponga que, para una memoria principal de 1024 palabras de 1 byte, posee una caché de 8 líneas y 16 palabras por línea con el siguiente estado intermedio:

Línea	Validez	*Tag*	*Timestamp*
0	1	57	5
1	1	27	10
2	1	40	2
3	1	34	8

Línea	Validez	*Tag*	*Timestamp*
4	1	58	7
5	1	9	4
6	1	62	3
7	1	0	1

La columna "Línea" representa el índice de línea; "Validez" representa el *valid bit*; "*Tag*" representa el valor decimal del *tag* que indica el bloque de memoria almacenado en la línea; y "*Timestamp*" representa el tiempo desde el último acceso a la línea (*i.e.* **un valor menor representa un acceso más reciente**). A partir de este estado, se realiza el acceso a memoria de las siguientes direcciones: 0x3AA, 0x1B0, 0x00F, 0x1FF, 0x01A, 0x2BB. Indique, para cada acceso, si hay *hit* o *miss*; la línea de caché accedida o modificada; y el *hit-rate* para la función de correspondencia ***fully associative*** **con política de reemplazo LRU**. Para responder, complete las tablas adjuntas al enunciado. El *tag* de cada línea lo puede escribir en base binaria o decimal. No necesita indicar el *timestamp* final por línea, pero sí debe usarlo en caso de reemplazos. **Considere que, si bien las direcciones se dan en hexadecimal, su representación binaria no necesariamente representa la cantidad de bits de cada dirección de memoria. Esta cantidad se debe deducir de la información otorgada.**

Solución: Al tener una memoria de 1024 palabras, se necesitan $\log_2(1024) = 10$ bits para cada dirección. Por otra parte, al ser las líneas de 16 palabras, se requiere $\log_2(16) = 4$ bits de *offset* para ubicar una palabra dentro de una línea. Por otra parte, al ser una caché *fully associative* se tendrán $10 - 4 = 6$ bits de *tag*. Dados estos valores, se puede observar que el dígito hexadecimal menos significativo de cada dirección será el *offset*; mientras que el resto será el *tag*, recordando que son 6 bits y no 8.

Con esto en consideración, se obtiene el siguiente resultado de la secuencia de accesos a memoria:

1. Acceso a dirección 0x3AA = 938 = 111010|1010b. Resulta en *hit*, ya que la línea 4 posee *tag* igual a 111010b = 58 = 0x3A. La línea 1 sigue siendo la que posee mayor *timestamp*, en caso de requerir un reemplazo.

2. Acceso a dirección 0x1B0 = 432 = 011011|0000b. Resulta en *hit*, ya que la línea 1 posee *tag* igual a 011011b = 27 = 0x1B. Ahora, la línea 3 es la que posee mayor *timestamp*, en caso de requerir un reemplazo.

3. Acceso a dirección 0x00F = 15 = 000000|1111b. Resulta en *hit*, ya que la línea 7 posee *tag* igual a 000000b = 0 = 0x00. La línea 3 sigue siendo la que posee mayor *timestamp*, en caso de requerir un reemplazo.

4. Acceso a dirección 0x1FF = 511 = 011111|1111b. Resulta en *miss* al no existir línea válida con *tag* 011111b = 31. Se copia el bloque sobre la línea 3, al ser esta la de mayor *timestamp*. Ahora, la línea 0 es la que posee mayor *timestamp*, en caso de requerir un reemplazo.

5. Acceso a dirección 0x01A = 26 = 000001|1010b. Resulta en *miss* al no existir línea válida con *tag* 000001b = 1. Se copia el bloque sobre la línea 0, al ser esta la de mayor *timestamp*. Ahora, la línea 5 es la que posee mayor *timestamp*, en caso de requerir un reemplazo.

6. Acceso a dirección 0x2BB = 699 = 101011|1011b. Resulta en *miss* al no existir línea válida con *tag* 101011b = 43. Se copia el bloque sobre la línea 5, al ser esta la de mayor *timestamp*. Ahora, la línea 6 es la que posee mayor *timestamp*, en caso de requerir un reemplazo.

A continuación, se muestra el estado final de la caché:

Línea	*Valid*	*Tag*
0	1	1
1	1	27
2	1	40
3	1	31

Línea	*Valid*	*Tag*
4	1	58
5	1	43
6	1	62
7	1	0

Hit-rate: $\frac{3}{6} = 0{,}5 = 50\,\%$

Se asignan **0.6 ptos.** por obtener el *hit-rate* correcto (ya sea explicitado u obtenido correctamente por cantidad de *hits* y *misses*) y **0.4 ptos** por acceso correctamente descrito (*i.e.* obtención de *tag*; determinación de *hit* o *miss* y selección de línea a reemplazar). Si se reemplaza la línea incorrecta por *timestamp*, se otorgan **0.2 ptos.** en el acceso correspondiente. Si se realiza todo de forma correcta, pero tomando un *tag* de 8 bits y no de 6, se descuentan **0.5 ptos.** del total.

Pregunta 2: Multiprogramación (6 ptos.)

(a) (1 pto.) Explique, en el contexto de la traducción de una dirección virtual a una dirección física, en qué consiste y cómo se utiliza el componente `PTBR`.

Solución: El `PTBR` o *Page Table Base Register* consiste en un registro que almacena la dirección de memoria base de la tabla de páginas del proceso en ejecución. Se utiliza, en conjunto con el número de página como índice, para obtener la dirección de memoria de la `PTE` o *Page Table Entry* que contiene los bits necesarios para llevar a cabo la traducción de la dirección virtual accedida a su dirección física correspondiente. Se otorgan **0.5 ptos.** por explicar en qué consiste el componente, y **0.5 ptos.** por explicar su uso en la traducción de una dirección virtual.

(b) (2 ptos.) Suponga que posee una memoria virtual dividida en 2^{22} páginas; una memoria física dividida en 2^{20} marcos físicos; y un total de 4 bits de *metadata* por entrada de la tabla de páginas. ¿Es posible determinar el tamaño de la tabla de páginas en este esquema de paginación? Si no es posible, indique por qué; en otro caso, explique cómo se determina y entregue el tamaño en KiB, MiB o GiB, según la mayor unidad de medida que pueda utilizar, asegurando que el valor calculado sea mayor o igual a 1.

Solución: Sí, es posible. Hacemos uso de la siguiente fórmula:

$$\text{Tamaño tabla} = \#\texttt{PTE} \times \texttt{sizeof(PTE)}$$

En este caso, `#PTE` corresponde a la cantidad de entradas, que es equivalente a la cantidad de páginas distintas en el esquema -correspondiente a 2^{22}-; mientras que `sizeof(PTE)` corresponde a la suma entre la cantidad de bits de marco físico y la cantidad de bits de *metadata*. Si bien la cantidad de bits de marco físico no es dada de forma explícita, se puede deducir a partir de la cantidad de marcos físicos disponible:

$$\#\text{ Bits marco físico} = \log_2(\#\text{Marcos}) = \log_2(2^{20}) = 20\text{b}$$

. Con los datos obtenidos, es posible reemplazar y despejar:

$$\text{Tamaño tabla} = 2^{22} \times (20+4)\text{b} = 2^{22} \times 3\text{B} = 12\text{MiB}$$

Se otorgan **0.5 ptos.** por indicar correctamente que es posible; y **1 pto.** por señalar cómo obtener la cantidad y realizar el cálculo; y **0.5 ptos.** por expresar la medida en MiB.

(c) (3 ptos.) Suponga que posee un espacio de direcciones virtuales de 21 bits, direcciones físicas de 20 bits y un tamaño de páginas de 64KiB. En este esquema, un proceso en ejecución posee el siguiente estado para su tabla de páginas en un instante dado:

Entrada	Presente	Marco
0	1	15
1	1	1
2	1	5
3	1	7
4	0	9
5	0	1
6	0	0
7	0	4

Entrada	Presente	Marco
8	0	2
9	1	6
10	0	7
11	1	8
12	0	5
13	1	10
14	0	11
15	1	12

Entrada	Presente	Marco
16	0	10
17	1	13
18	1	2
19	0	13
20	1	4
21	1	3
22	0	3
23	0	12

Entrada	Presente	Marco
24	1	9
25	1	14
26	1	0
27	0	14
28	0	8
29	0	15
30	0	6
31	1	11

La columna "Entrada" representa el número de entrada de la tabla; la columna "Presente" representa si el contenido del marco físico se encuentra en la memoria principal (1) o en el *swap file* (0); y "Marco" representa el valor, en base decimal, del marco físico asociado a la página ligada a la entrada. Con este estado de tabla de páginas, se realiza el acceso a memoria a las siguientes direcciones virtuales: 0x1FE505, 0x04B1DD, 0x11ECAF, 0x0FFFEE. Indique, para cada acceso, si existe un *page fault* o no. Solo en caso de no existir *page fault* (*i.e.* que la página de la dirección virtual posea un marco físico asociado válido), indique el número de marco y la dirección física a la que se traduce la dirección virtual. Para responder, se solicita que complete la tabla adjunta al enunciado. Debe escribir las direcciones físicas obtenidas en base **hexadecimal**. **Considere que, si bien las direcciones virtuales se dan en hexadecimal y se pueden escribir con 24 bits, debe respetar las dimensiones descritas y usar los 21 bits menos significativos para representar cada dirección.**

Solución: Al tener un tamaño de páginas de 64KiB, se tienen $2^6 \times 2^{10} = 2^{16}$ palabras de memoria por página, lo que implican $\log_2(2^{16}) = 16$ bits de *offset*. De esto se deduce que el número de marco físico es de $20 - 16 = 4$ bits y el número de página es de $21 - 16 = 5$ bits. Al tener 5 bits de número de página, los 2 dígitos hexadecimales más significativos de cada dirección virtual nos entregan el número de página. De esta forma, se deben reemplazar los dígitos del número de página por el valor hexadecimal obtenido de la tabla, que se puede representar con un solo dígito al tener $2^4 = 16$ valores distintos. Así, se tiene el siguiente resultado de la secuencia de traducciones:

1. Dirección 0x1FE505 → página 0x1F = 31. Se tiene un marco físico **presente en memoria** con valor igual a 11 = 1011b = 0xB. Dirección física = 0xBE505.
2. Dirección 0x04B1DD → página 0x04 = 4. Se tiene el marco físico **en el *swap file***, por lo que existe un ***page fault***.
3. Dirección 0x11ECAF → página 0x11 = 17. Se tiene un marco físico **presente en memoria** con valor igual a 13 = 1101b = 0xD. Dirección física = 0xDECAF.
4. Dirección 0x0FFFEE → página 0x0F = 15. Se tiene un marco físico **presente en memoria** con valor igual a 12 = 1100b = 0xC. Dirección física = 0xCFFEE.

Se asigna **0.75 ptos.** por acceso correctamente realizado (*i.e.* determinación de *page fault* o traducción de dirección virtual a física). Si existen errores de arrastre, se otorgan como máximo **0.375 ptos.** por traducción si en el desarrollo se evidencia una correcta obtención de bits de *offset*, número de página y número de marco físico para cada dirección virtual.

Pregunta 3: Paralelismo a nivel de instrucción (ILP) (6 ptos.)

(a) (1 pto.) Suponga que se implementa *stalling* en el computador básico con *pipeline* vía *hardware*. Indique desde qué etapa se puede llevar a cabo la detección del *hazard* de datos que requiere de *stalling* para su resolución y qué señales son necesarias para detectarlo.

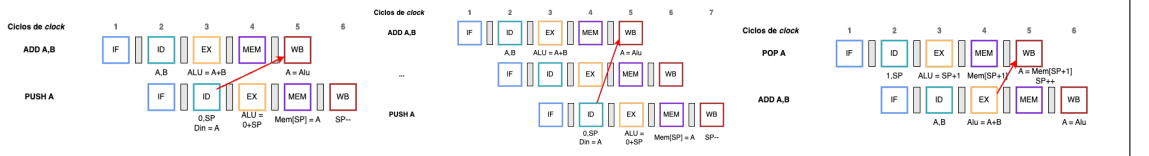
Solución: La detección se puede llevar a cabo desde la etapa ***Instruction Decode***, ya que en dicho punto se decodifica el *opcode* en las señales de control que permiten determinar la existencia del *hazard*. De esta forma, la detección se lleva a cabo con las siguientes combinaciones de señales:

- `ID.AluA == A && ID/EX.LoadA == 1 && ID/EX.RegIn == DOUT`. La instrucción actual hace uso del registro `A` como argumento en la ALU, mientras que la instrucción anterior actualizará su valor a través de una lectura de memoria.
- `ID.AluB == B && ID/EX.LoadB == 1 && ID/EX.RegIn == DOUT`. La instrucción actual hace uso del registro `A` como argumento en la ALU, mientras que la instrucción anterior actualizará su valor a través de una lectura de memoria.

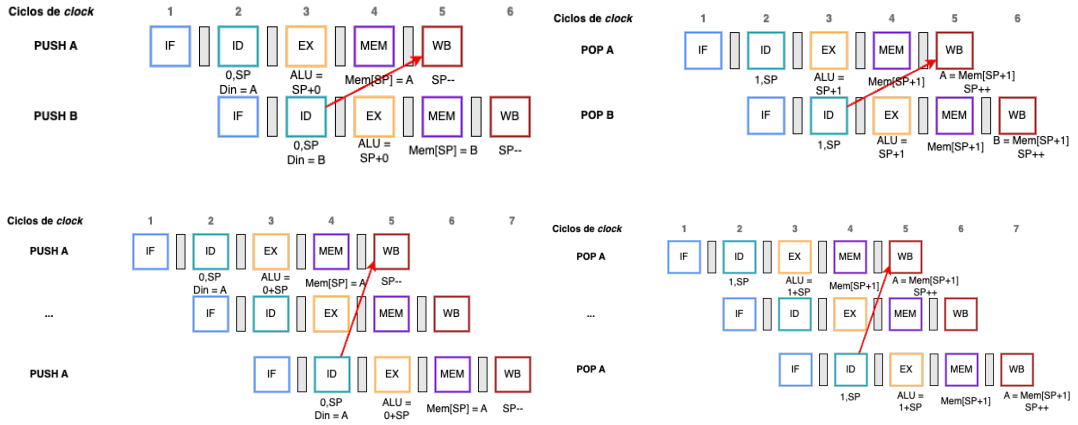
Se otorgan **0.5 ptos.** por señalar correctamente la etapa donde se detecta el *hazard*, y **0.5 ptos.** por indicar de forma correcta la combinación de señales que permite detectarlo. No es necesario que se incluyan ambas combinaciones, pero sí al menos una de ellas.

(b) (2 ptos.) Suponga que se modifica el computador básico con *pipeline* para reincorporar las instrucciones de manejo de *stack*: `PUSH A/B` y `POP A/B`. Indique dos casos **distintos** de *hazard* de datos que podrían surgir a raíz de esta modificación, explicando en cada ejemplo la etapa en la que surge la dependencia. Se considerarán "casos distintos" si se detectan con combinaciones de señales distintas. A continuación, se muestran tres casos distintos:

Solución: Como `PUSH A` y `PUSH B` corresponden a instrucciones de escritura de memoria, mientras que `POP A` y `POP B` son de lectura de memoria, entonces pueden generar los mismos tipos de *hazards* de estos tipos de instrucción. A continuación, se muestran tres casos distintos:



Por otra parte, si asumiéramos que el *stack pointer* se actualiza en la etapa ***Writeback***, podrían surgir otros cuatro casos distintos:



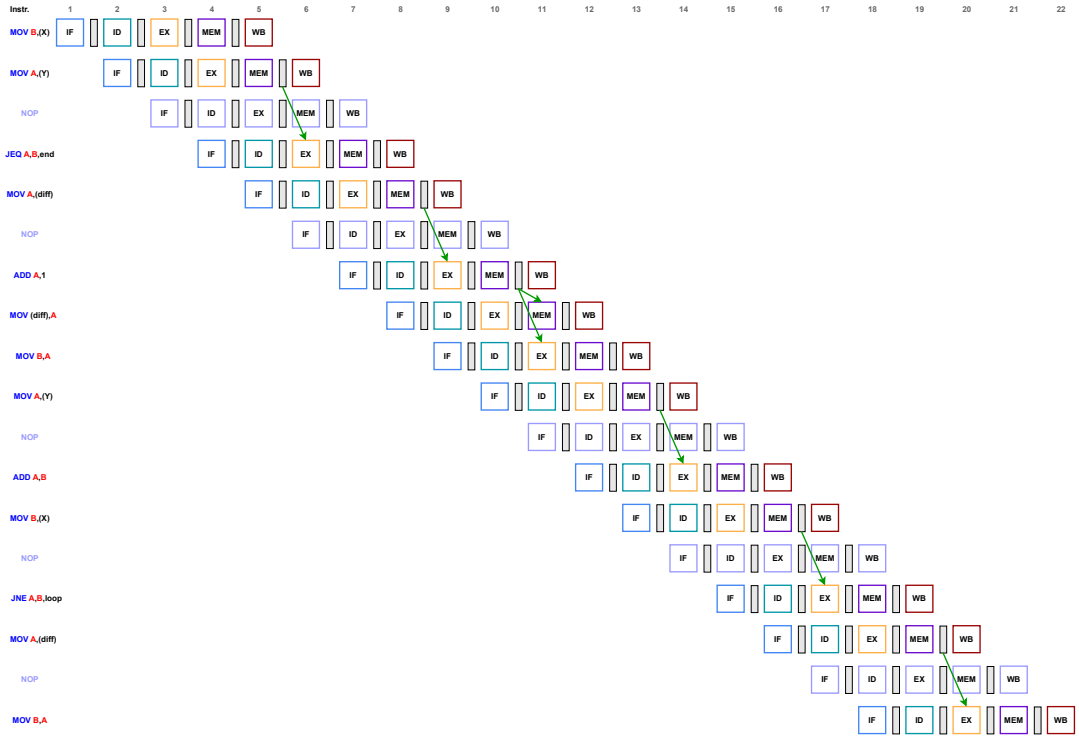
Por cada caso, se otorgan **0.5 ptos.** por señalar una combinación de instrucciones donde efectivamente exista un *hazard*, y **0.5 ptos.** por justificación. Si se presentan dos casos que son equivalentes en términos de detección, solo se otorga puntaje en uno de ellos.

(c) (3 ptos.) A partir del siguiente programa del computador básico con *pipeline*:

```
DATA:
    X    7
    Y    6
    diff 0
CODE:
    MOV B, (X)
    MOV A, (Y)
    JEQ A, B, end
    loop:
        MOV A, (diff)
        ADD A, 1
        MOV (diff), A
        MOV B, A
        MOV A, (Y)
        ADD A, B
        MOV B, (X)
        JNE A, B, loop
    end:
        MOV A,(diff)
        MOV B,A
```

Determine el número de ciclos que demora su ejecución detallando en un diagrama los estados del *pipeline* por instrucción. Asuma 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, independiente de que estos sean condicionales o incondicionales. Indique en el diagrama adjunto al enunciado cuando ocurre *forwarding* (con flechas desde el registro hacia la etapa que corresponda), *stalling* y *flushing* (con tachado en las instrucciones *flusheadas*).

Solución: A continuación, se muestra el *pipeline* resultante de la ejecución del programa.



Del *pipeline* anterior se deduce que el programa termina de correr después de 22 ciclos. Se otorga puntaje de la siguiente forma: **0.3 ptos.** por cada *stalling* correctamente detectado (**1.5 ptos.** por todas las detecciones) **0.2 ptos.** por cada *forwarding* correctamente detectado (**1.4 ptos.** por todas las detecciones); y **0.1 ptos.** por la deducción correcta de la cantidad de ciclos en la que corre el programa.

Si no se agrega correctamente un *stall*, pero se aplican bien los *forwardings*, se otorga puntaje en dicho criterio. No obstante, si existen uno o más *forwardings* mal aplicados, se descuentan **0.2 ptos.**

Se otorga el puntaje por cantidad de ciclos solo si se obtiene el valor correcto por la ejecución correcta del programa y **no por accidente por errores de ejecución**.

No hay descuento si se incluyen las instrucciones de escritura de memoria correspondientes al segmento `DATA`, pero solo si la ejecución de estas se describe de forma correcta.