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

IIC2343 - Arquitectura de Computadores (II/2025)

Interrogación 3

Pauta de evaluación

Pregunta 1: Jerarquía de memoria (6 ptos.)

(a) (1 pto.) El tamaño de línea en una caché incide en el tiempo de transferencia de datos desde la memoria principal. A partir de este hecho, indique cómo afecta este tamaño al *hit-rate* y al *miss penalty*.

Solución: Al tener un mayor tamaño de línea, se podrá tener un mayor *hit-rate* dado que cada acceso involucrará a un espacio adyacente mayor (*i.e.* mayor cantidad de direcciones transferidas a la caché en caso de *miss*), por lo que se explotará aún más el principio de localidad espacial. El *miss penalty*, en cambio, también aumentará, ya que naturalmente aumenta la cantidad de datos a transferir de la memoria principal a la caché en caso de un *miss*. Se otorgan **0.5 ptos.** por explicar correctamente la incidencia sobre el *hit-rate*, y **0.5 ptos.** por explicar correctamente la incidencia sobre el *miss penalty*.

(b) (1 pto.) Su supervisor le solicita que, con el fin de usar de forma óptima la memoria caché en el computador que está diseñando en su empresa, implemente la política de reemplazo Bélády. Según lo estudiado en el curso, ¿cómo respondería ante esta solicitud?

Solución: La solicitud se debe responder de forma **negativa**. La política de reemplazo Bélády es un ideal inalcanzable, ya que implica saber cuál será la línea a ser accedida de forma más lejana en el futuro **antes de que ocurra**. Como no tenemos la capacidad de ver el futuro, no es posible cumplir con las expectativas del supervisor. Se otorgan **0.5 ptos.** por señalar correctamente que no se puede implementar la política solicitada, y **0.5 ptos.** por justificación.

(c) (4 ptos.) Suponga que, para una memoria principal de 4096 palabras de 1 byte, posee una caché de 32 líneas y 16 palabras por línea con el siguiente estado intermedio:

Línea	*Valid*	*Tag*	*Time*
0	1	18	14
1	1	50	13
2	1	57	12
3	1	24	11
4	1	51	10
5	1	4	9
6	1	52	8
7	0	6	-

Línea	*Valid*	*Tag*	*Time*
8	1	20	20
9	1	26	19
10	1	50	18
11	1	39	17
12	1	45	16
13	1	63	15
14	0	1	-
15	0	15	-

Línea	*Valid*	*Tag*	*Time*
16	1	0	24
17	1	43	23
18	1	58	22
19	1	7	21
20	0	10	-
21	0	15	-
22	0	2	-
23	0	3	-

Línea	*Valid*	*Tag*	*Time*
24	1	25	7
25	1	31	6
26	1	12	5
27	1	14	4
28	1	23	3
29	1	18	2
30	1	60	1
31	0	63	-

La columna "Línea" representa el índice de línea; "*Valid*" representa el *valid bit*; "*Tag*" representa el valor decimal del *tag* que indica el bloque de memoria almacenado en la línea; y "*Time*" representa el tiempo **desde el último acceso a la línea**. A partir de este estado, se realiza el acceso a memoria de las siguientes direcciones: 0x000, 0xC9A, 0x03F, 0x07F, 0x1E1. Indique, para cada acceso, si existe un *hit* o *miss* y, si corresponde, la línea de la caché que es modificada para la función de correspondencia *8-way associative*. Asuma una la 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 considerar su valor en caso de reemplazos.

Solución: Al tener una memoria de 4096 palabras, se necesitan $\log_2(4096) = 12$ bits para cada dirección. Por otra parte, al ser las líneas de 16 palabras, se requieren $\log_2(16) = 4$ bits de *offset* para ubicar una palabra dentro de una línea. Por otra parte, en una caché *8-way associative* se tienen conjuntos de 8 líneas, lo que deriva en un total de $\frac{32}{8} = 4$ conjuntos y, de esta forma, $\log_2(4) = 2$ bits de índice de conjunto. Esto, finalmente, resulta en $12 - 4 - 2 = 6$ bits de *tag*. Con esto en consideración, se obtiene el siguiente resultado de la secuencia de accesos a memoria:

1. Acceso a dirección 0x000 = 0 = 000000|00|0000b. Resulta en *miss* al no existir línea válida con *tag* 000000b = 0 en el conjunto 00 (líneas 0-7). Se copia el bloque sobre la línea 7 al estar desocupada. La línea 0 sigue siendo la que posee mayor *timestamp* dentro del conjunto, en caso de requerir un reemplazo.

2. Acceso a dirección 0xC9A = 3226 = 110010|01|1010b. Resulta en *hit*, ya que la línea 10 posee *tag* igual a 110010b = 50 en el conjunto 01 (líneas 8-15). La línea 8 sigue siendo la que posee mayor *timestamp* dentro del conjunto en caso de que este se llene y se requiera un reemplazo.

3. Acceso a dirección 0x03F = 63 = 000000|11|1111b. Resulta en *miss* al no existir línea válida con *tag* 000000b = 0 en el conjunto 11 (líneas 24-31). Se copia el bloque sobre la línea 31 al estar desocupada. La línea 24 sigue siendo la que posee mayor *timestamp* dentro del conjunto en caso de requerir un reemplazo.

4. Acceso a dirección 0x07F = 127 = 000001|11|1111b. Resulta en *miss* al no existir línea válida con *tag* 000001b = 1 en el conjunto 11 (líneas 24-31). Se copia el bloque sobre la línea 24, al ser esta la de mayor *timestamp*. Ahora, la línea 25 es la que posee mayor *timestamp* dentro del conjunto en caso de requerir un reemplazo.

5. Acceso a dirección 0x1E1 = 481 = 000111|10|0001b. Resulta en *hit*, ya que la línea 19 posee *tag* igual a 000111b = 7 en el conjunto 10 (líneas 16-23). La línea 16 sigue siendo la que posee mayor *timestamp* dentro del conjunto en caso de requerir un reemplazo.

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

Línea	*Valid*	*Tag*
0	1	18
1	1	50
2	1	57
3	1	24
4	1	51
5	1	4
6	1	52
7	1	0

Línea	*Valid*	*Tag*
8	1	20
9	1	26
10	1	50
11	1	39
12	1	45
13	1	63
14	0	-
15	0	-

Línea	*Valid*	*Tag*
16	1	0
17	1	43
18	1	58
19	1	7
20	0	-
21	0	-
22	0	-
23	0	-

Línea	*Valid*	*Tag*
24	1	1
25	1	31
26	1	12
27	1	14
28	1	23
29	1	18
30	1	60
31	1	0

Se otorgan **0.8 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 **0.4 ptos.** en el acceso correspondiente.

Excepcionalmente, se otorgan **3 ptos.** del total si se realiza **todo** el procedimiento de forma correcta, pero con error de arrastre en el cálculo de bits de *offset*, índice de conjunto o *tag*; o bien si se parte contando del conjunto 1 y no del conjunto 0 al momento de seleccionar las líneas.

Pregunta 2: Coherencia de Caché (6 ptos.)

Suponga que posee una arquitectura MIMD de memoria compartida con 3 CPU que poseen su propia caché y que siguen el protocolo MESI para asegurar consistencia. La memoria contiene los siguientes datos de variables; mientras que CPU0, CPU1 y CPU2 ejecutan los siguientes programas:

Dirección	*Label*	Valor
0x00	`var`	0
...	...	...
0xFD		0
0xFE		0
0xFF		0

```
// CPU0              // CPU1              // CPU2
MOV B, 1             MOV B, 253          MOV A, 1
PUSH B               MOV (var), B        MOV B, 2
MOV A, (var)         PUSH B              POP A
ADD A, 1             POP B
MOV B, A                                 PUSH B
MOV (B), B           MOV B,(B)           PUSH B
```

Asumiendo que cada dirección se almacena en una línea distinta, y que el contador `SP` posee un valor inicial igual a 255 en **todas** las CPU, indique el estado de cada línea de las caché (`M`/`E`/`S`/`I`) para cada ciclo completando la tabla adjunta, asumiendo que todas las líneas parten en estado `I` (ciclo 0):

	CPU0				CPU1				CPU2			
Ciclo	**0x00**	**0xFD**	**0xFE**	**0xFF**	**0x00**	**0xFD**	**0xFE**	**0xFF**	**0x00**	**0xFD**	**0xFE**	**0xFF**
0	I	I	I	I	I	I	I	I	I	I	I	I
1												
2												
3												
4												
5												
6												

Solución: A continuación se muestra la tabla final esperada por cada iteración:

	CPU0				CPU1				CPU2			
Ciclo	**0x00**	**0xFD**	**0xFE**	**0xFF**	**0x00**	**0xFD**	**0xFE**	**0xFF**	**0x00**	**0xFD**	**0xFE**	**0xFF**
0	I	I	I	I	I	I	I	I	I	I	I	I
1	I	I	I	I	I	I	I	I	I	I	I	I
2	I	I	I	M	M	I	I	I	I	I	I	I
3	S	I	I	I	S	I	I	M	I	I	I	I
4	S	I	I	I	S	I	I	M	S	I	I	I
5	I	I	I	I	I	I	I	M	M	I	I	I
6	I	I	M	I	I	E	I	I	M	I	I	M

Para llegar al estado final correcto, es importante haber tomado en cuenta lo siguiente:

- Al ser `SP` parte de una CPU, su valor es **independiente** de lo que ejecuten otras. Es decir, que una CPU realice un `PUSH` o un `POP` no afecta el valor de `SP` de otra.

- La ejecución de `POP A` y `POP B` toma dos ciclos. La lectura del valor de memoria se realiza en el **segundo ciclo**, dado que el primero lo único que hace es incrementar el valor de `SP`.
- Al hacer `POP` sin haber hecho un `PUSH` antes, la operación generará un *overflow* sobre el contador `SP`, causando que su valor cambie de 255 a 0. Esto implica que, en la CPU2, al ejecutar `POP` realizará una lectura de memoria sobre la dirección 0 en el cuarto ciclo, *i.e.* la variable `var`.
- Si una CPU hace una lectura sobre una línea ya modificada por ella misma, esta sigue estando en dicho estado y solo cambia en dos escenarios: (1) si otra CPU realiza una lectura sobre ella, cambiando a estado *Shared*; o (2) si otra CPU realiza una escritura sobre ella, cambiando a estado *Invalid*.

Se otorga **1 pto.** por cada iteración (fila) con estado final correcto para todas las CPU. Se otorgan **0.7 ptos.** por iteración (fila) si se tiene como máximo una celda con estado erróneo; y **0.3 ptos.** si se tienen como máximo dos celdas con estado erróneo. En otro caso, no se otorga puntaje en la iteración.

Para efectos de corrección, toda celda vacía se considerará en estado inválido, a excepción del caso en el que no se haya contestado la pregunta.

Si una celda se modifica a un estado erróneo en una iteración y se mantiene en la siguiente, solo se descuenta en la fila donde ocurrió el error. En cambio, si debía cambiar su estado en la siguiente iteración y se mantiene en el estado erróneo, se descuenta en ambas filas.

Por último, si un error de interpretación de código genera un estado erróneo en múltiples celdas, se aplica el descuento solo sobre una de ellas si los cambios de todas las celdas son coherentes con la interpretación.

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

(a) (2 ptos.) Suponga que en el computador básico con *pipeline* corre un programa de 100 instrucciones. Si un 10 % de ellas son de salto, y en un 75 % de ellos el salto **no se realiza**, ¿cuál es el número aproximado de ciclos perdidos por *flushing*? Justifique a través de cálculos.

Solución: En el computador básico con *pipeline*, la unidad predictora de saltos asume que estos nunca se realizan. Por lo tanto, cada vez que se realiza un salto, se pierden tres ciclos por *flushing*. Luego, considerando la información otorgada, se tiene el siguiente número aproximado de ciclos perdidos:

$$3_{\text{Ciclos}} \times (100_{\text{Instrucciones}} \times 0{,}1_{\text{Saltos}} \times 0{,}25_{\text{Saltos que se realizan}}) = 7{,}5_{\text{Ciclos}}$$

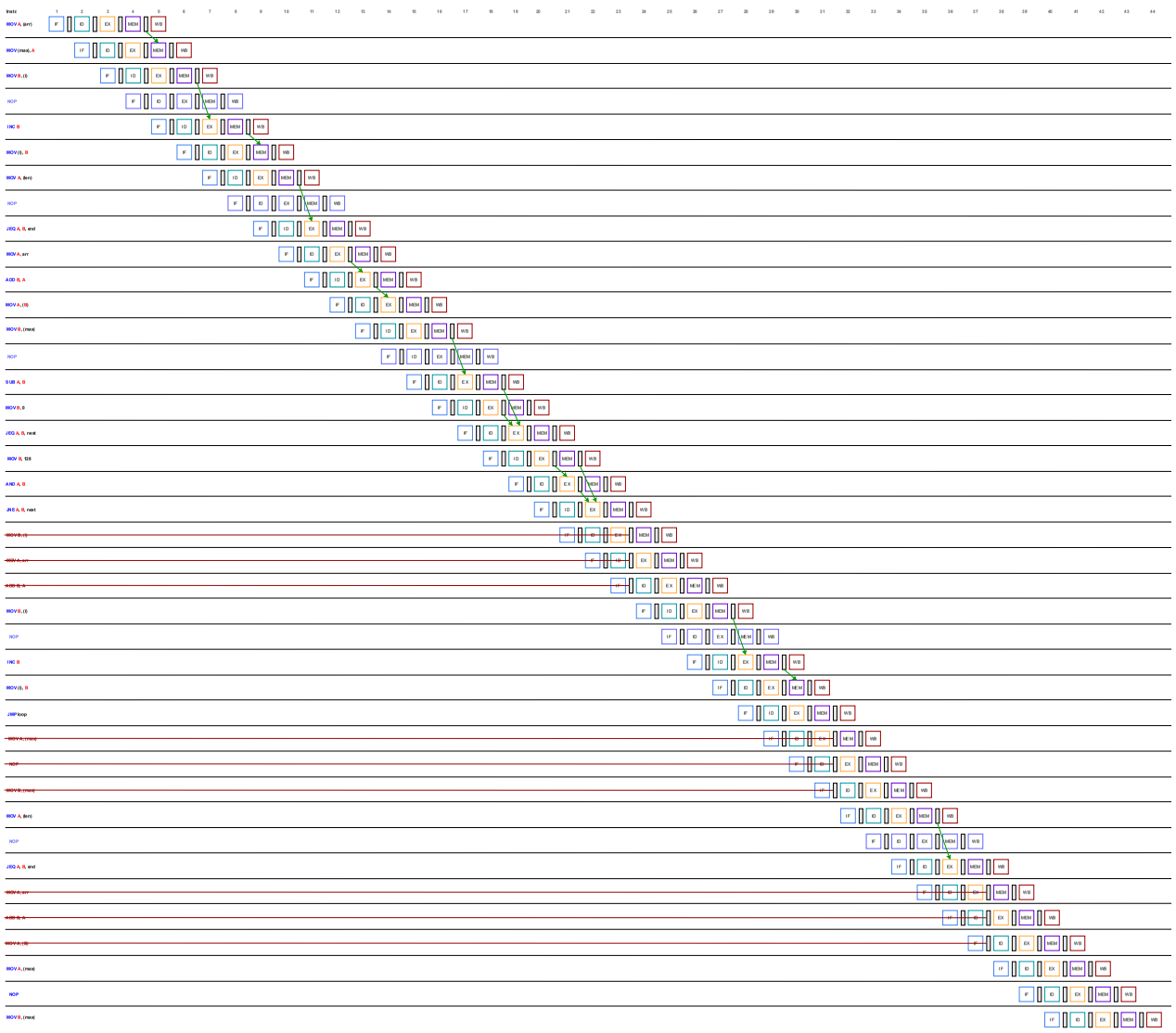
Se otorga **1 pto.** por señalar la cantidad correcta de ciclos perdidos por salto, y **1 pto.** por calcular correctamente el número perdido de ciclos en total con los datos entregados, siempre que se respalden con cálculos.

(b) (4 ptos.) A partir del siguiente programa del computador básico con *pipeline*:

```
DATA:
  arr 2
      7
  len 2
  i   0
  max -1
CODE:
  MOV A, (arr)
  MOV (max), A
  MOV B, (i)
  INC B
  MOV (i), B
  loop:
    MOV A, (len)
    JEQ A, B, end
    MOV A, arr
    ADD B, A
    MOV A, (B)
    MOV B, (max)
    SUB A, B
    MOV B, 0
    JEQ A, B, next
    MOV B, 128
    AND A, B
    JNE A, B, next
    MOV B, (i)
    MOV A, arr
    ADD B, A
    MOV A, (B)
    MOV (max), A
    next:
      MOV B, (i)
      INC B
      MOV (i), B
      JMP loop
    end:
      MOV A, (max)
      NOP
      MOV B, (max)
```

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: En la siguiente plana, se muestra el *pipeline* resultante de la ejecución del programa.


Instr
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
MOV A, (arr)
MOV (mod), A
MOV B, (i)
NOP
INC B
MOV (i), B
MOV A, (arr)
NOP
JEQ A, B, end
MOV A, arr
ADD B, A
MOV A, (0)
MOV B, (mod)
NOP
SUB A, B
MOV B, 0
JEQ A, B, next
MOV B, 128
AND A, B
JNE A, B, next
MOV B, (i)
MOV A, arr
ADD B, A
MOV B, (i)
NOP
INC B
MOV (i), B
JMP loop
MOV A, (mod)
NOP
MOV B, (mod)
MOV A, (arr)
NOP
JEQ A, B, end
MOV A, arr
ADD B, A
MOV A, (0)
MOV A, (mod)
NOP
MOV B, (mod)
IF
ID
EX
MEM
WB

Del *pipeline* anterior se deduce que el programa termina de correr después de 44 ciclos. Se otorga puntaje de la siguiente forma: **0.25 ptos.** por cada *stalling* correctamente detectado (**1.25 ptos.** por todas las detecciones); **0.1 ptos.** por cada *forwarding* correctamente detectado (**1.5 ptos.** por todas las detecciones); **0.35 ptos.** por cada *flushing* correctamente detectado (**1.05** ptos. por todas las detecciones); y **0.2 ptos.** por la deducción correcta de la cantidad de ciclos en la que corre el programa.

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**. Excepcionalmente, se tomará también como correcta una cantidad de ciclos igual a 45 si agregan un *stalling* entre las primeras dos instrucciones, aunque no haya sido necesario.

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.**

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.

Nota: El objetivo original del código era obtener el máximo de `arr`. No obstante, para ello se debe modificar la instrucción `JNE A,B,next` posterior a `AND A,B` por `JEQ A,B,next`. Esto ocasionaría, no obstante, que la ejecución del *pipeline* se extendiera a 46 ciclos y reducido la cantidad de ciclos perdidos por *flushing* a 6.