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

IIC2343 - Arquitectura de Computadores (I/2024)

Interrogación 3

Pauta de evaluación

Pregunta 1: Jerarquía de memoria (6 ptos.)

(a) (1 pto.) En la jerarquía de memoria de un computador, cuenta con una memoria caché de 64KiB, líneas de 16 palabras y tiempo de acceso de 10ns. Por otra parte, cuenta con un *miss penalty* de 100ns. ¿Qué *hit-rate* debe tener la caché para obtener un tiempo de acceso promedio de 20ns en el computador?

Solución: Para obtener el tiempo de acceso promedio en una memoria, se utiliza la siguiente fórmula:

$$TP = HR \times HT + (1 - HR) \times (HT + MP)$$

Siendo TP el tiempo de acceso promedio; HT el *hit-time* promedio; HR el *hit-rate* promedio; y MP el *miss penalty*. Luego, reemplazando valores:

$$20\text{ns} = HR \times 10\text{ns} + (1 - HR) \times (10 + 100)$$

$$20\text{ns} = 110 - 100HR$$

$$HR = \frac{90}{100}$$

$$HR = 0{,}9$$

Se otorgan **0.5 ptos.** por la aplicación correcta de la fórmula y **0.5 ptos.** por el cálculo correcto del *hit-rate* promedio.

(b) (1 pto.) Suponga que posee dos computadores con arquitecturas similares, salvo por sus memorias caché: poseen las mismas dimensiones, pero una es 16-*way associative* y la otra es 8-*way associative*. ¿Cuál presenta un *hit-time* promedio menor? Justifique su respuesta.

Solución: Considerando que ambas memorias caché poseen las mismas dimensiones, la que presenta un menor *hit-time* promedio corresponde a la 8-*way associative*. Esto se debe a que la búsqueda en memorias caché N-*way associative* se realiza solo sobre las líneas del conjunto al que se mapea el bloque de la palabra buscada. Como N equivale a la cantidad de líneas por conjunto, en una caché 8-*way associative* la búsqueda se realiza sobre la mitad de las líneas de una 16-*way associative*, lo que implica un menor tiempo para determinar si hay *hit* o no. Se otorgan **0.5 ptos.** por indicar la respuesta correcta y **0.5 ptos.** por justificación.

(c) (4 ptos.) Suponga que, para una memoria principal de 256 palabras de 1 byte, posee una caché de 16 líneas y 4 palabras por línea con el siguiente estado intermedio:

Línea	Validez	*Tag*	*Timestamp*
0	1	20	1
1	1	5	11
2	1	10	4
3	1	25	9
4	1	31	8
5	1	16	3
6	1	1	6
7	1	8	2

Línea	Validez	*Tag*	*Timestamp*
8	0	10	-
9	0	20	-
10	0	30	-
11	0	5	-
12	0	15	-
13	0	2	-
14	0	1	-
15	0	0	-

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**. A partir de este estado, se realiza el acceso a memoria de las siguientes direcciones: 0x54, 0xE4, 0xE3, 0xCA, 0xCC, 0xA1. Indique, para cada acceso, si existe un *hit* o *miss*; si corresponde, la línea de la caché que es modificada; y el *hit-rate* 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 256 palabras, se necesitan $\log_2(256) = 8$ bits para cada dirección. Por otra parte, al ser las líneas de 4 palabras, se requiere $\log_2(4) = 2$ 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 $\frac{16}{8} = 2$ conjuntos y, de esta forma, $\log_2(2) = 1$ bit de índice de conjunto. Esto, finalmente, resulta en $8 - 1 - 2 = 5$ bits de *tag*. En este caso, el conjunto 0 corresponde a las líneas 0-7 y el conjunto 1 a las líneas 8-15.

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

1. Acceso a dirección 0x54 = 84 = 01010|1|00b. Resulta en *miss* al no existir línea válida con *tag* 01010b = 10 en el conjunto 1. Se copia el bloque sobre la línea 8.
2. Acceso a dirección 0xE4 = 228 = 11100|1|00b. Resulta en *miss* al no existir línea válida con *tag* 11100b = 28 en el conjunto 1. Se copia el bloque sobre la línea 9.
3. Acceso a dirección 0xE3 = 227 = 11100|0|11b. Resulta en *miss* al no existir línea válida con *tag* 11100b = 28 en el conjunto 0. Se copia el bloque sobre la línea 1, al ser esta la de mayor *timestamp*. Ahora, la línea 3 es la que posee mayor *timestamp*, en caso de requerir un reemplazo.
4. Acceso a dirección 0xCA = 202 = 11001|0|10b. Resulta en *hit*, ya que la línea 3 posee *tag* igual a 11001b = 25 en el conjunto 0. Ahora, la línea 4 es la que posee mayor *timestamp*, en caso de requerir un reemplazo.
5. Acceso a dirección 0xCC = 204 = 11001|1|00b. Resulta en *miss* al no existir línea válida con *tag* 11001b = 25 en el conjunto 1. Se copia el bloque sobre la línea 10.
6. Acceso a dirección 0xA1 = 161 = 10100|0|01b. Resulta en *hit*, ya que la línea 0 posee *tag* igual a 10100b = 20 en el conjunto 0.

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

Línea	Validez	*Tag*
0	1	20
1	1	28
2	1	10
3	1	25
4	1	31
5	1	16
6	1	1
7	1	8

Línea	Validez	*Tag*
8	1	10
9	1	28
10	1	25
11	0	-
12	0	-
13	0	-
14	0	-
15	0	-

Hit-rate: $\frac{2}{6} = 0.\overline{3} = 33,\overline{3}\,\%$

Se asignan **0.4 ptos.** por obtener el *hit-rate* correcto y **0.6 ptos** por acceso correctamente descrito (*i.e.* obtención de *tag*; determinación de *hit* o *miss* y selección de línea a reemplazar). Si existen errores de arraste, se otorgan como máximo **0.3 ptos.** por acceso si en el desarrollo se evidencia una correcta obtención de bits de *offset*, índice y *tag* por dirección.

Pregunta 2: Multiprogramación (6 ptos.)

(a) (1 pto.) En un computador moderno, varios procesos comparten los recursos del procesador para llevar a cabo su ejecución. Esto incluye a la TLB; memoria caché utilizada para agilizar la traducción de direcciones virtuales a físicas. A partir de este contexto, suponga que cada entrada de una TLB posee: un bit de validez, bits de número de página y bits de marco físico, siendo esta información la que permite realizar la traducción de la dirección virtual solicitada por el proceso en ejecución. ¿Qué ocurre con el contenido de la TLB si cambia el proceso que se encuentra ejecutando en la CPU? Justifique su respuesta.

Solución: Al haber un cambio de contexto (*i.e.* cambio de proceso en ejecución), por las características descritas de la TLB, **no es posible** reutilizar su contenido. Esto se debe a que los marcos a los que se traducen las páginas del proceso que comenzará su ejecución son distintos a los del que se detiene. Por lo tanto, se deben **invalidar** las entradas de la TLB configurando en 0 el bit de validez. **No se "borra" el contenido**, solo se cambia el bit de validez. Se otorgan **0.5 ptos.** por indicar correctamente lo que ocurre y **0.5 ptos.** por justificación.

(b) (1 pto.) Suponga que posee una memoria virtual de 4GiB, una memoria física de 1GiB, un tamaño de página de 32KiB y un tamaño de tabla de páginas de 256KiB. ¿Es posible determinar la cantidad de bits de *metadata* por entrada de la tabla de páginas? Si es posible, explique cómo; en otro caso, indique por qué no lo es.

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; mientras que `sizeof(PTE)` corresponde a la suma entre la cantidad de bits de marco físico y la cantidad de bits de *metadata*. Con los datos obtenidos, es posible reemplazar y despejar:

$$\begin{aligned}
256\text{KiB} &= \frac{4\text{GiB}}{32\text{KiB}} \times \left(\log_2\left(\frac{1\text{GiB}}{32\text{KiB}}\right) + metadata\right)\text{b} \\
2^{18}\text{B} &= 2^{17} \times (\log_2(2^{15}) + metadata)\text{b} \\
2^{18}\text{B} &= 2^{17} \times (15 + metadata)\text{b} \\
2\text{B} &= (15 + metadata)\text{b} \\
2\text{B} &= \frac{15 + metadata}{8}\text{B} \\
16 &= 15 + metadata \\
metadata &= 1
\end{aligned}$$

Se otorgan **0.5 ptos.** por indicar correctamente que es posible y **0.5 ptos** por señalar cómo obtener la cantidad. No es necesario que se realice el cálculo, pero sí que se despeje la fórmula para evidenciar la factibilidad del cómputo.

(c) (4 ptos.) Suponga que posee un espacio de direcciones virtuales de 32 bits, direcciones físicas de 31 bits y un tamaño de páginas de 256MiB. 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	0
1	0	0
2	1	5
3	0	7
4	0	3
5	1	4
6	0	6
7	1	6

Entrada	Presente	Marco
8	0	3
9	0	4
10	0	5
11	1	1
12	1	7
13	1	2
14	0	2
15	1	3

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: 0xDEADC0DE, 0xBADDCAFE, 0x0A0A0A0A, 0xF0CACC1A. 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 resultante. Para responder, se solicita que complete la tabla adjunta al enunciado. Puede escribir las direcciones físicas obtenidas en base binaria, decimal o hexadecimal.

Solución: Al tener un tamaño de páginas de 256MiB, se tiene un total de $2^8 \times 2^{20} = 2^{28}$ palabras de memoria por página, por lo que se tienen $\log_2(2^{28}) = 28$ bits de *offset*. De esto se deduce que el número de marco físico es de $31 - 28 = 3$ bits y el número de página es de $32 - 28 = 4$ bits. Este último coincide con las dimensiones de la tabla de páginas, dado que se tienen $2^4 = 16$ entradas. Al tener 4 bits de número de página, el dígito hexadecimal más significativo de cada dirección virtual nos entrega directamente el número de página. Así, se tiene el siguiente resultado de la secuencia de traducciones:

1. Dirección 0xDEADC0DE $\rightarrow$ página 0xD = 13. Se tiene un marco físico **presente en memoria** con valor igual a 2 = 010b. Dirección física traducida = 0x2EADC0DE.
2. Dirección 0xBADDCAFE $\rightarrow$ página 0xB = 11. Se tiene un marco físico **presente** con valor igual a 1 = 001b. Dirección física traducida = 0x1ADDCAFE.
3. Dirección 0x0A0A0A0A $\rightarrow$ página 0x0 = 0. Se tiene un marco físico **presente** con valor igual a 0 = 000b. Dirección física traducida = 0x0A0A0A0A.
4. Dirección 0xF0CACC1A $\rightarrow$ página 0xF = 15. Se tiene un marco físico **presente** con valor igual a 3. Dirección física traducida = 0x30CACC1A.

Se asigna **1 pto.** por acceso correctamente realizado (*i.e.* determinación de *page fault* o traducción de dirección virtual a física). Se acepta expresar la dirección física resultante como la concatenación de bits y dígitos hexadecimales. Si existen errores de arrastre, se otorgan como máximo **0.5 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.

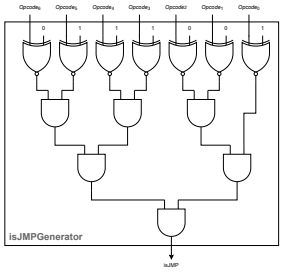
Pregunta 3: Paralelismo a nivel de instrucción (ILP) (6 ptos.)

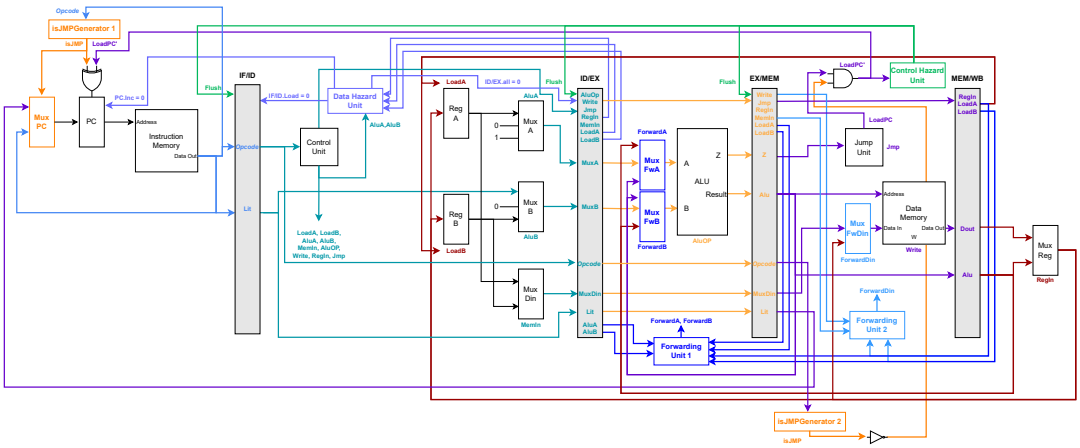
(a) (3 ptos.) La arquitectura del computador básico con *pipeline* no permite realizar saltos incondicionales inmediatamente después de la ejecución de la etapa `IF`. Estos, por construcción, se realizan en la etapa `MEM`, lo que conlleva al *flushing* de las instrucciones ejecutadas erróneamente. Explique, a partir del diagrama adjunto, por qué no es posible y proponga modificaciones sobre la microarquitectura para que los saltos **incondicionales** se ejecuten en la etapa `IF`. Puede realizar las modificaciones propuestas sobre el diagrama o describirlas.

Solución: El problema radica en que la etapa `IF` no cuenta con soporte de *hardware* para decodificar la señal que habilita el salto (`Jmp`, proveniente de `ID`) ni para cargar el literal en `PC` a través de las señal de carga `LoadPC` (provenientes de `MEM`). Por lo tanto, es necesario modificar esta etapa para cumplir lo solicitado. A continuación, una propuesta:

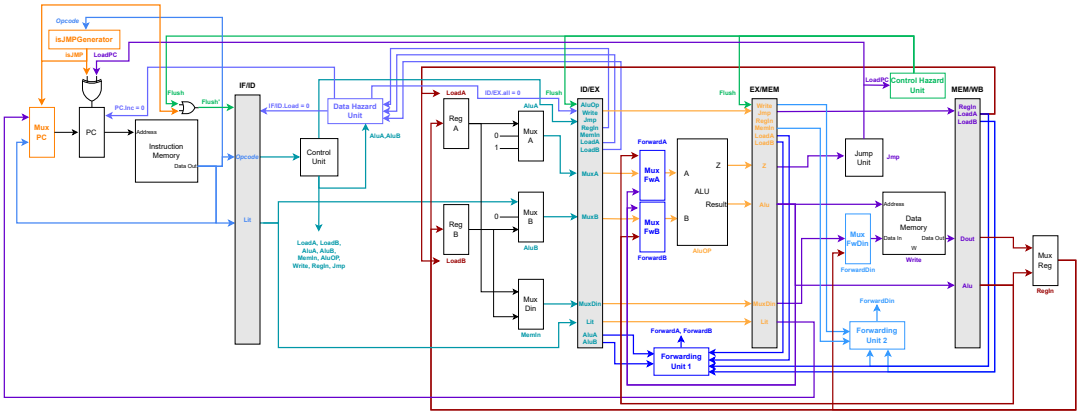
- Agregar un circuito que detecta si el *opcode* de la instrucción es exactamente igual al de `JMP`. Retorna 1 si lo es y 0 en otro caso. Para ejemplificar, se hará uso del *opcode* utilizado en la ISA adjunta. El componente con el circuito interno se llamará `isJMPGenerator`, mientras que su señal de salida se llamará `IF.isJMP`.
- Agregar un circuito exactamente igual al anterior, pero en `MEM`. La señal resultante se llamará `IF.isJMP`. Esto añade la necesidad de transmitir el *opcode* desde `IF` hasta `MEM`.
- Agregar un multiplexor `Mux PC`. Este recibirá de entrada el literal proveniente de la etapa `MEM` (`MEM.Lit`) y el de la etapa `IF` (`IF.Lit`). Su señal de selección será igual a `IF.isJMP`, seleccionando `IF.Lit` si su valor es 1 y `MEM.Lit` en otro caso.
- Agregar una compuerta lógica `AND` en `MEM`, cuyas dos entradas serán la señal `LoadPC` proveniente de la `Jump Unit` y `NOT isJMP`. La salida de esta compuerta se llamará `LoadPC'` y se enviará en reemplazo de la señal `LoadPC` original hacia la etapa `IF` y hacia la `Control Hazard Unit`. Esto evitará que se realice el salto incondicional dos veces y que se haga *flush* en este escenario.
- Agregar una compuerta lógica `XOR`, cuyas dos entradas serán `MEM.LoadPC'` e `IF.isJMP`. Su salida se conectará directamente con la entrada de carga de `PC`, lo que habilitará el salto instantáneo en `IF` y evitará que se realice de nuevo cuando se ejecuta la etapa `MEM` de la instrucción.

El siguiente diagrama permite visualizar esta propuesta:





Otra propuesta más óptima en términos de *hardware* es la siguiente: en vez de agregar una segunda unidad `isJMPGenerator` y evitar desde `MEM` la ejecución doble del salto incondicional, habilitar el *flush* del registro `IF/ID` para que no se propague la instrucción `JMP` al resto de etapas. Con eso bastaría para evitar la ejecución doble del salto. A continuación, el diagrama que muestra esta segunda propuesta:



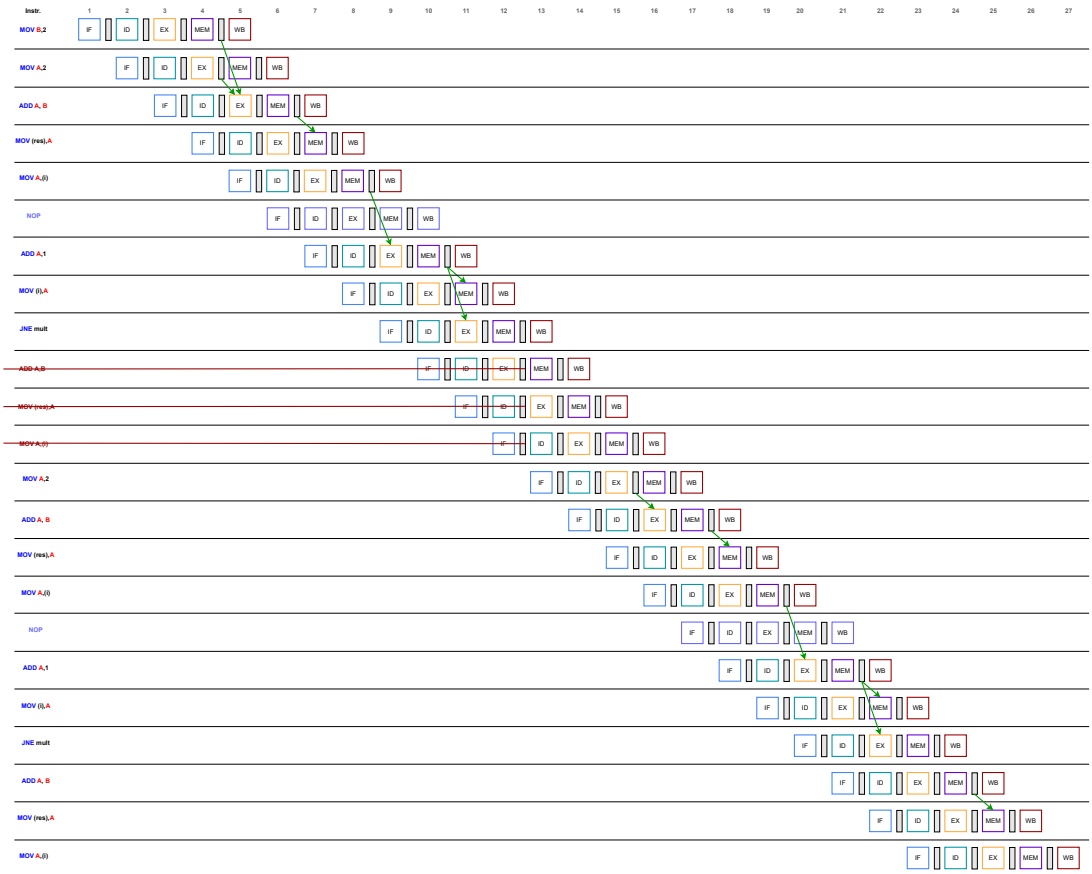
Se otorgan **1.5 ptos.** por explicar correctamente por qué no se pueden realizar los saltos incondicionales en la etapa `IF`; **0.5 ptos.** si se conecta correctamente el literal de `IF` con la entrada de `PC`; **0.5 ptos.** si se habilita correctamente la carga en `IF` solo para la instrucción `JMP`; y **0.5 ptos.** si se evita de forma correcta que se realice el salto dos veces. Por cada criterio, se otorga la mitad del puntaje si existe como máximo un error conceptual o de implementación.

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

```
DATA:
  res 0
  i   0
CODE:
  MOV B,2
  mult:
    MOV A,2
    ADD A,B
    MOV (res),A
    MOV A,(i)
    ADD A,1
    MOV (i),A
    JNE mult
    ADD A,B
    MOV (res),A
    MOV A,(i)
```

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 27 ciclos. Se otorga puntaje de la siguiente forma: **0.3 ptos.** por cada *stalling* correctamente detectado (**0.6 ptos.** por todas las detecciones); **0.4 ptos.** por la detección correcta del *flushing*; **0.15 ptos.** por cada *forwarding* correctamente detectado (**1.8 ptos.** por todas las detecciones); y **0.2 ptos.** por la deducción correcta de la cantidad de ciclos en la que corre el programa. 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.

Pregunta 4: Pregunta por sección (6 ptos.)

En este apartado, debe contestar **solo** la pregunta de la sección en la que se encuentra inscrito/a.

Sección 1

(a) (1.5 ptos.) En principio, si las líneas de la caché son de mayor tamaño (almacenan más bytes), entonces se aprovecha mejor la localidad espacial y se reduce el *miss rate*. Sin embargo, si aumentamos mucho el tamaño de las líneas de la caché, hasta que llegue a ser una fracción importante del tamaño de la misma, entonces se manifiestan dos problemas. Describe de manera precisa y breve uno de estos dos problemas.

Solución: A continuación, se describen los dos problemas:

a) Va a haber pocas líneas, por lo tanto va a haber mucha competencia por esas líneas, y por lo tanto cada línea va a ser reemplazada frecuentemente antes de que haya sido posible tener acceso a varias de sus palabras (el reemplazo frecuente de líneas es un problema porque deteriora el desempeño de la cache).

b) Aumenta el costo de un caché *miss*, ya que el tiempo que toma transferir una línea de la memoria a la caché depende del tamaño de la línea.

Se otorgan **1.5 ptos.** por la descripción correcta de uno de estos problemas. Se otorgan **0.75 ptos.** si es que la descripción presenta, como máximo, un error conceptual.

(b) (1.5 ptos.) ¿En qué sentido el mecanismo de paginación empleado en memoria virtual es **transparente**? Responde de manera precisa y breve.

Solución: La memoria virtual es la simulación de una gran memoria principal —que en realidad puede o no existir— mediante paginación desde la memoria secundaria: **esta simulación no es detectable por los programas**. Así, la memoria virtual/paginación permite a un programador ver al computador como si realmente tuviera una gran memoria principal —suficiente para todo el espacio virtual de direcciones— y escribir su programa de modo que este puede leer desde, o almacenar en, cualquier palabra en esta memoria, o saltar a cualquier instrucción ubicada en cualquier parte dentro de esta memoria. Se otorgan **1.5 ptos.** por una explicación correcta; y **0.75 ptos.** si la respuesta presenta, como máximo, un error conceptual.

(c) (1.5 ptos.) En el contexto de *pipelining*, explica de manera clara, precisa, y breve, la ventaja de la predicción dinámica de saltos usando dos bits frente a la predicción dinámica de saltos usando sólo un bit.

Solución: En un *loop*, normalmente la primera o segunda instrucción es un salto condicional hacia "afuera" del *loop* si es que la condición de este no se cumple. Consideremos la última evaluación de la condición: una vez que el *loop* en efecto ha terminado; debido a todas las iteraciones anteriores, la predicción es que el salto no se toma. Como en este caso el salto efectivamente se toma, hay que cambiar un bit del predictor. Si el predictor tiene un solo bit, entonces la próxima vez que el programa entre al mismo *loop*, la predicción va a ser que el salto no se toma y se va a equivocar. En cambio, si el predictor tiene dos bits, como solo se cambió uno, entonces la predicción va a ser que el salto no se toma y va a ser correcta. Así, en *loops* a los que el programa entre varias veces (por ejemplo, un *loop* anidado dentro de otro), un predictor de un bit se equivoca una vez al inicio del *loop* y otra vez al término del *loop*; en cambio, un predictor de dos bits solo se equivoca una vez al término del *loop*. Se otorgan **1.5 ptos.** por una explicación correcta; y **0.75 ptos.** si la respuesta presenta, como máximo, un error conceptual.

(d) (1.5 ptos.) Una variación del protocolo MESI de coherencia de caché es el protocolo MSI (sin el estado *Exclusive*). ¿Qué ventaja tiene tener el estado *Exclusive*? Y ¿cómo se hace cargo el protocolo MSI de no tener el estado *Exclusive*? Responde de manera clara, precisa y breve.

Solución: El estado *Exclusive* permite reducir el tráfico en el bus compartido por todas las memorias caché en caso de que el procesador realice una solicitud de escritura sobre una línea en este estado. Esto se debe a que como solo su caché posee la copia del bloque de memoria, no es necesario invalidar copias de otras caché. En el protocolo MSI, en cambio, se hace uso del estado `S` (*Shared*) aunque **una** sola caché posea una copia de un bloque de memoria. Por lo tanto, si se realizan solicitudes de escritura sobre ella, se emitirá la señal correspondiente para invalidar otras copias, a pesar de que no exista ninguna. Se otorgan **0.75 ptos.** por explicar la ventaja del estado *Exclusive*; y **0.75 ptos.** por señalar el manejo del estado MSI a falta de este estado. En cada caso, se otorgan **0.375 ptos.** si la respuesta presenta, como máximo, un error conceptual.

Sección 3

Considere la siguiente memoria. Esta es compartida por 3 CPUs con paralelismo UMA. Para mantener la coherencia entre sus caché, se usa el protocolo MESI.

Memoria compartida

Dir	Label	Valor
0	x	1
1	y	2
2	arr	3
3		4
4		5

CPU 0

```
MOV B, arr
INC B
MOV A, (B)
MOV (x), A
MOV A, 3
MOV B, (4)
ADD A, B
MOV (4), A
```

CPU 1

```
MOV A, (x)
MOV B, (y)
ADD A, B
MOV (3), A
MOV A, 2
MOV B, (x)
MOV (3), B
ADD A, B
```

CPU 2

```
MOV B, arr
MOV A, 0
ADD A, (B)
INC B
ADD A, (B)
INC B
MOV (y), A
MOV B, (3)
```

Puede asumir que las líneas de la caché son de una palabra y todas las líneas de las caché privadas parten en estado inválido. Complete las siguientes tablas a partir de la ejecución de los programas en las tres CPU. En cada entrada, indique si el valor fue **modificado** (`M`), es **exclusivo** (`E`), **compartido** (`S`) o **inválido** (`I`).

	Caché - CPU 0				
Inst	x	y	arr[0]	arr[1]	arr[2]
1					
2					
3					
4					
5					
6					
7					
8					

	Caché - CPU 1				
Inst	x	y	arr[0]	arr[1]	arr[2]
1					
2					
3					
4					
5					
6					
7					
8					

	Caché - CPU 2				
Inst	x	y	arr[0]	arr[1]	arr[2]
1					
2					
3					
4					
5					
6					
7					
8					

Solución:

A continuación, se muestra el estado de la caché de cada CPU posterior a la ejecución de cada una de las instrucciones de los programas:

	Caché - CPU 0				
`Inst`	`x`	`y`	`arr[0]`	`arr[1]`	`arr[2]`
1	I	I	I	I	I
2	I	I	I	I	I
3	I	I	I	E	I
4	M	I	I	I	I
5	M	I	I	I	I
6	S	I	I	I	E
7	S	I	I	I	E
8	S	I	I	I	M

	Caché - CPU 1				
`Inst`	`x`	`y`	`arr[0]`	`arr[1]`	`arr[2]`
1	E	I	I	I	I
2	E	E	I	I	I
3	E	E	I	I	I
4	I	E	I	M	I
5	I	E	I	S	I
6	S	E	I	S	I
7	S	I	I	M	I
8	S	I	I	S	I

	Caché - CPU 2				
`Inst`	`x`	`y`	`arr[0]`	`arr[1]`	`arr[2]`
1	I	I	I	I	I
2	I	I	I	I	I
3	I	I	E	I	I
4	I	I	E	I	I
5	I	I	E	S	I
6	I	I	E	S	I
7	I	M	E	I	I
8	I	M	E	S	I

Se otorgan **0.25 ptos.** por cada ciclo con estado final correcto por cada CPU. Se otorga la mitad del puntaje (**0.125 ptos.**) si el ciclo tiene como máximo una celda con estado erróneo. 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.

Sección 2, Sección 4

(a) (4 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	`arr`	2
0x01		0
0x02		1
...	...	...
0xFE		0
0xFF		0

```
// CPU0          // CPU1          // CPU2
MOV B,arr        MOV A,2          MOV A,5
INC B            PUSH A           MOV B,1
MOV B,(B)        POP A            PUSH A
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**	**0x01**	**0x02**	**0xFE**	**0xFF**	**0x00**	**0x01**	**0x02**	**0xFE**	**0xFF**	**0x00**	**0x01**	**0x02**	**0xFE**	**0xFF**
0	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
1															
2															
3															
4															

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

	CPU0					CPU1					CPU2				
Ciclo	**0x00**	**0x01**	**0x02**	**0xFE**	**0xFF**	**0x00**	**0x01**	**0x02**	**0xFE**	**0xFF**	**0x00**	**0x01**	**0x02**	**0xFE**	**0xFF**
0	I	I	I	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	I	I	I
2	I	I	I	I	I	I	I	I	I	M	I	I	I	I	I
3	I	E	I	I	I	I	I	I	I	I	I	I	I	I	M
4	M	E	I	I	I	I	I	I	I	S	I	I	I	M	S

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

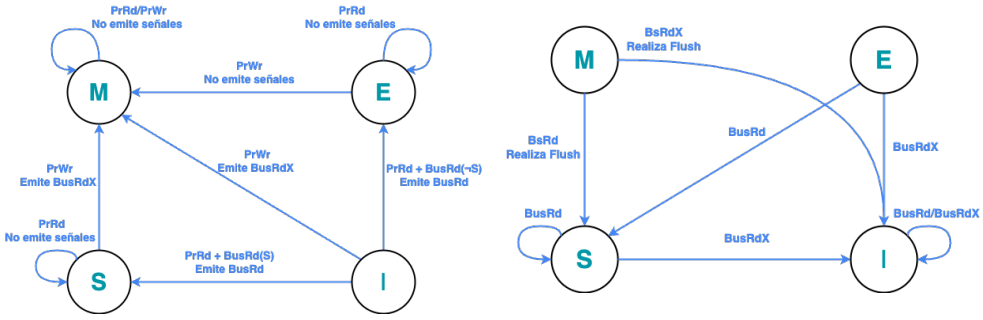
- `MOV B,arr` **no es una instrucción de lectura de memoria**. Solo guarda la dirección de `arr` en `B`.
- La ejecución de `POP A` toma dos ciclos. La lectura del valor de memoria se realiza en el **segundo ciclo**, dado que el primero lo único que hace es incrementar el valor de `SP`.
- Al ser `SP` 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.

Se otorga **1 pto.** por cada iteración con estado final correcto para todas las CPU. Se otorgan **0.5 ptos.** por iteración si se tiene como máximo una celda con estado erróneo. 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.

(b) (2 ptos.) El protocolo MESIF consiste en una extensión de MESI que incluye un nuevo estado `F` (*Forward*). Este protocolo presenta las siguientes diferencias respecto a MESI:

- Si dos o más caché poseen una misma copia de un bloque sin cambios de la memoria compartida, **solo una** de ellas se encarga de transmitir la línea a otras CPU que soliciten el contenido. Esta caché será la que poseerá la línea en estado `F`.
- La caché **con la copia más reciente** se encarga de realizar la transmisión. Una vez que la caché transmite un bloque, su línea pasa de estado `F` a `S`. La que recibe la copia, en cambio, pasa a estado `F` y se vuelve la nueva encargada de realizar las transmisiones.
- Si la línea de una primera caché se encuentra en estado `E` y una segunda caché otra solicita una copia del bloque, entonces la línea de la primera pasa de estado `E` a `S`, mientras que la línea de la segunda pasa a estado `F`. En este caso, el bloque no es transmitido entre las dos caché, sino que se obtiene directamente de la memoria principal.

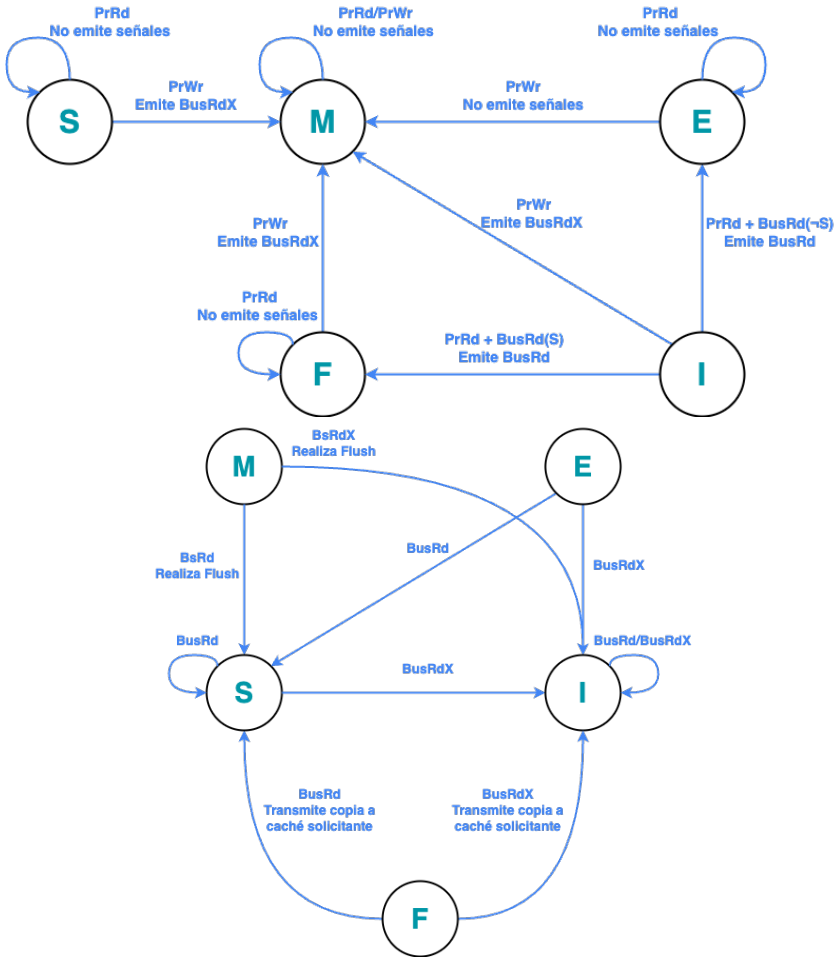
Modifique los diagramas de estados adjuntos para incluir el nuevo estado `F` y los cambios correspondientes según los eventos del procesador (`PrRd`, `PrWr`) y del bus compartido (`BusRd`, `BusRdX`). No es necesario que indique las señales que se emiten en cada cambio de estado, pero sí debe señalar el evento que gatilla el cambio de un estado a otro.



Solución: Se indicarán los cambios de estado de `F` según cada tipo de evento:

- `PrRd`: El procesador actual solicita una lectura sobre la línea en estado `F`. Esto no implica transmisión alguna de la copia a otra caché, por lo que se mantiene en este estado. No es necesario que emita señales.
- `PrWr`: El procesador actual solicita una escritura sobre la línea en estado `F`. Esto implica cambiar a estado `M` y, al mismo tiempo, invalidar las copias de otras caché, por lo que se emite la señal `BusRdX` con ese fin.
- `BusRd`: Un procesador distinto del actual solicita una lectura sobre la línea en estado `F`. La caché del procesador actual transmite su copia. Al hacerlo, cambia a estado `S` al no ser la que posee la copia más reciente. No emite señales ni realiza *flush*.
- `BusRdX`: Un procesador distinto del actual solicita una escritura sobre la línea en estado `F`. La caché del procesador actual se encarga de trasmite su copia. Al hacerlo, cambia a estado `I` al poseer ahora una copia que no posee las últimas modificaciones. No emite señales ni realiza *flush*.

El resto de los estados se mantiene sin cambios **salvo por `I`**: Si un procesador solicita la lectura (`PrRd`) de un bloque a ser almacenado en una línea en estado `I`, y esta copia existe en más memorias caché (`BusRd(S)`), entonces ya no pasará a estado `S`, sino que pasará a estado `F` al poseer la copia más reciente. Debe seguir emitiendo la señal `BusRd` ya que, a través de ella, la caché que se encuentre en estado `F` podrá transmitir su copia y pasar a estado `S`.
A continuación, se muestran los diagramas resultantes:



Se otorgan **0.4 ptos.** por la acción correcta sobre `F` a partir de cada tipo de evento (`PrRd`, `PrWr`, `BusRd`, `BusRdX`); y **0.4 ptos.** por actualizar correctamente el cambio de `I` a `F` para el evento `PrRd + BusRd(S)`. Solo se evalúa el cambio de estado, no se considera la emisión de señales. No es necesaria la modificación del diagrama siempre que el cambio esté descrito de forma clara y precisa en la respuesta.