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

IIC2343 - Arquitectura de Computadores (II/2025)

Ayudantía

Ayudantes: Daniela Ríos (danielaarp@uc.cl), Alberto Maturana (alberto.maturana@uc.cl), Tomás López (tomás.lopezm20@uc.cl)

Pregunta 1: Preguntas Conceptuales

(a) **[P4.2 | EX-2014-1]** ¿Por qué pueden haber problemas de coherencia de caché en una CPU de 1 solo *core*?

> **Solución:** Pueden haber problemas si un dispositivo de IO escribe en memoria usando DMA. Al escribir con DMA la información no pasa por la CPU, por lo cual se podrá escribir en memoria un dato que existe en caché sin que la CPU sepa sobre el cambio, produciéndose problemas de coherencia de caché.

(b) **[P2 | EX-2017-2]** En un multiprocesador (varias CPUs que comparten una memoria común a través de un bus), cada CPU puede tener su propia memoria caché para así evitar tener que recurrir frecuentemente a la memoria principal a través del bus. Sin embargo, esto normalmente da origen al **problema de coherencia de cachés**. Para solucionar estos problemas, los controladores de cachés son diseñados de manera que "observen" (*snoop*) las solicitudes que pasan por el bus (y que fueron hechas por alguna otra caché) y que hagan algo en ciertos casos. El conjunto de reglas que define qué hacer y cuándo se llama **protocolo de consistencia de cachés**.

I. Considera el protocolo *write through* estudiado en clase, cuya esencia es que todas las operaciones de escritura resultan en que la palabra que está siendo escrita en la caché también es escrita en la memoria para mantener la memoria actualizada todo el tiempo. Si el controlador de caché solo pudiera observar las líneas de dirección del bus, y no las de datos, ¿se vería el protocolo *write through* afectado por esta situación? Explica.

II. Mencione y explique cada estado del protocolo *write-back* MESI.

III. En los protocolos de tipo *write-back* no todas las escrituras van directamente a la memoria: cuando una línea de la caché es modificada, se pone un bit de la caché en 1 indicando que la línea de la caché está correcta pero la memoria no; finalmente, la línea es escrita en la memoria, pero posiblemente después de sufrir varias escrituras. El protocolo MESI estudiado en clase define cuatro estados para cada línea de la caché: *modified*, *exclusive*,

shared (compartido), *invalid*. Si solo pudiéramos tener tres estados, ¿cuáles estados podrían ser eliminados (solo uno a la vez) y cuáles serían las consecuencias en cada caso?

Solución:

I. No. En el protocolo *write-through* básico, cuando un controlador de caché observa que una dirección de memoria está siendo escrita por otra CPU, chequea su propio caché para ver si tiene esa misma dirección, y, si la tiene, entonces invalida la línea correspondiente, pero no la actualiza (hasta que su propia CPU lea esa misma dirección más adelante, en cuyo caso el controlador va a buscar la línea a la memoria). Por lo tanto, para que el protocolo funcione correctamente, lo que importa es que el controlador de caché sepa cuál dirección está siendo escrita, pero no es necesario que sepa con qué valor.

Nota: La respuesta podría ser sí, si se estuviera hablando de la versión del protocolo en que el controlador actualiza la línea de su caché en lugar de invalidarla; en este caso, el protocolo se vería afectado, ya que tendría que ser modificado para funcionar como en la versión básica.

II. El protocolo *write-back* MESI busca mejorar el desempeño de la consistencia de caché evitando actualizar en cada solicitud de escritura la memoria principal. Para ello, maneja un total de cuatro estados para cada línea de la caché:

- *Modified*: Indica que la línea de la caché corresponde a un bloque de la memoria principal que ha sido modficado (siendo el procesador el que lo modicó desde su caché).
- *Exclusive*: Indica que dicha línea de la caché corresponde a un bloque de la memoria principal que solo ha sido accedido por el procesador correspondiente, haciéndola exclusiva.
- *Shared*: Indica que dicha línea de la caché corresponde a un bloque de la memoria principal que ha sido accedido por más de un procesador, haciéndola compartida.
- *Invalid*: Indica que dicha línea de la caché es inválida (es decir, está disponible para su uso pues lo contenido en ella no tiene validez alguna, ya sea por ser recientemente inicializada o por no estar actualizada).

III. Eliminar el estado *invalid* es una mala idea: exigiría mantener todas las líneas de todas las cachés permanentemente actualizadas.

En cambio, los estados *exclusive* y *shared* pueden combinarse en uno, con comportamiento *shared*: la línea con el valor más reciente está en una o más cachés y la memoria está actualizada; si cualquiera de las CPUs involucradas escribe (cambia un valor de) la línea en su caché, entonces le avisa a las otras (aunque no haya "otras") que invaliden sus líneas, y pasa a estado *modified*, es decir, solo mi línea es

válida e incluso la memoria es inválida.

El estado *modified* puede eliminarse —haciendo que la memoria esté siempre actualizada— pero el costo es que el protocolo deja de ser *write-back* y pasa a ser *write-through*.

Pregunta 2: Protocolo MESI (P4 | I3-2024-2)

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	`var1`	255
0x01	`var2`	253
...	...	...
0xFE		0
0xFF		0

```
// CPU0              // CPU1          // CPU2
MOV A, (var1)        MOV A, 0       MOV B, 0
MOV B, (var2)        NOP            PUSH B
AND B,A              PUSH A         INC B
INC B                POP B          PUSH B
MOV B, (B)                          POP A
MOV (B), B           MOV B,(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**	**0xFE**	**0xFF**	**0x00**	**0x01**	**0xFE**	**0xFF**	**0x00**	**0x01**	**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**	**0x01**	**0xFE**	**0xFF**	**0x00**	**0x01**	**0xFE**	**0xFF**	**0x00**	**0x01**	**0xFE**	**0xFF**
0	I	I	I	I	I	I	I	I	I	I	I	I
1	E	I	I	I	I	I	I	I	I	I	I	I
2	E	E	I	I	I	I	I	I	I	I	I	M
3	E	E	I	I	I	I	I	M	I	I	I	I
4	E	E	I	I	I	I	I	M	I	I	M	I
5	E	E	S	I	I	I	I	M	I	I	S	I
6	S	M	S	I	S	I	I	M	I	I	S	I

- **Ciclo 1: CPU0** realiza una lectura (Emite `BusRd`) de un dato que no estaba en memoria, al confirmar que no está en ninguna otra caché, pasa al estado *Exclusive* de la dirección **0x00**. **CPU1** y **CPU2** ejecutan operaciones `MOV Reg, Lit`, por lo que no lee memoria.

- **Ciclo 2: CPU0** realiza una nueva lectura (Emite `BusRd`) de un dato que no estaba en memoria, al confirmar que no está en ninguna otra caché, pasa a tener otro estado *Exclusive* de la dirección **0x01**. **CPU1** sigue con todas sus líneas inválidas y **CPU2** realiza una escritura (Emite `BusRdX`), leyendo y copiando el bloque de memoria directamente en caché, pasa al estado *Modified* de la dirección **0xFF**.

- **Ciclo 3: CPU0** no lee y ni escribe en memoria, por lo que se mantiene como estaba. **CPU1** realiza una escritura (Emite `BusRdX`), almacena en su línea el estado *Modified* de la dirección **0xFF** e invalida el contenido de la caché de **CPU2**, que por su parte no ejecuta ninguna operación de lectura o escritura.

- **Ciclo 4: CPU0** no lee y ni escribe en memoria nuevamente, por lo que se mantiene como estaba. **CPU1** ejecuta el primer ciclo de `POP B`, por lo que aún no ha escrito en memoria, manteniendo sus estados como antes. **CPU2** realiza una escritura (Emite `BusRdX`), leyendo y copiando el bloque de memoria directamente en caché, pasa al estado *Modified* de la dirección **0xFE**.

- **Ciclo 5: CPU0** realiza una lectura (Emite `BusRd`) de la dirección **0xFE**, pero el contenido ya existía en otra caché (**CPU2**), por lo que su estado pasa a *Shared*. **CPU1** realiza una solicitud de lectura (Emite `BusRd`) de una dirección con un estado ya asociado a *Modified*, por lo que se mantiene como esta. **CPU2** ejecuta el primer ciclo de la operación `POP A` por lo que debería mantener el estado de la dirección **0xFE** en *Modified*, pero como **CPU1** ahora también tiene acceso al contenido, su estado cambia a *Shared*.

- **Ciclo 6: CPU0** realiza una escritura de la dirección **0x00**, por lo que su estado asociado debería pasar a *Modified*. Sin embargo, **CPU1** realiza una lectura de esa misma dirección, por lo que el estado de ambos pasa a ser *Shared*. Finalmente, **CPU2** ejecuta el segundo ciclo de la operación `POP A`, leyendo el contenido de la dirección **0xFE**, pero como ya estaba en estado *Shared*, las lecturas hacen que mantenga este estado.

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

- Si se realiza la operación `AND B,A` con 8 bits para cada registro, se llega al resultado `B` = 253, por lo que la CPU0 luego accede a la dirección de memoria 254 = 0xFE considerando la ejecución de la instrucción `INC B`.
- 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 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.

Pregunta 3: Otros Protocolos (P4.b - Sec. 3 | I3-2024-1)

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 otra segunda caché 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.

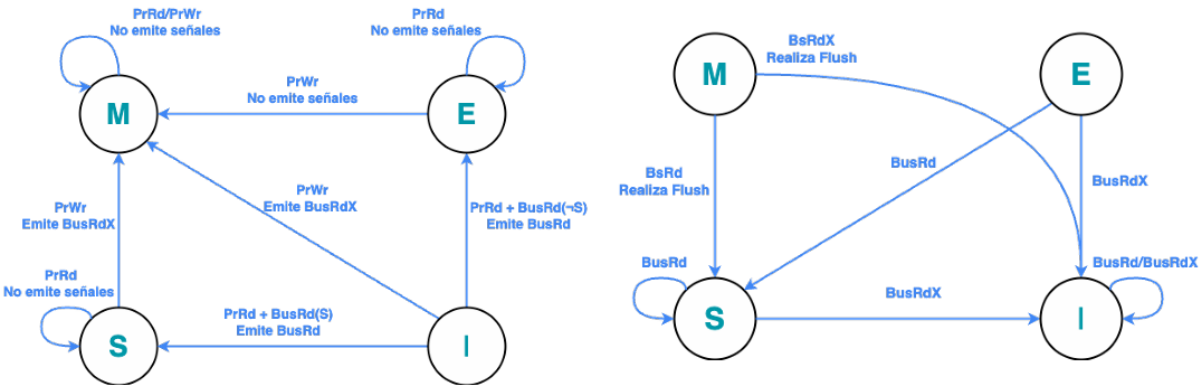


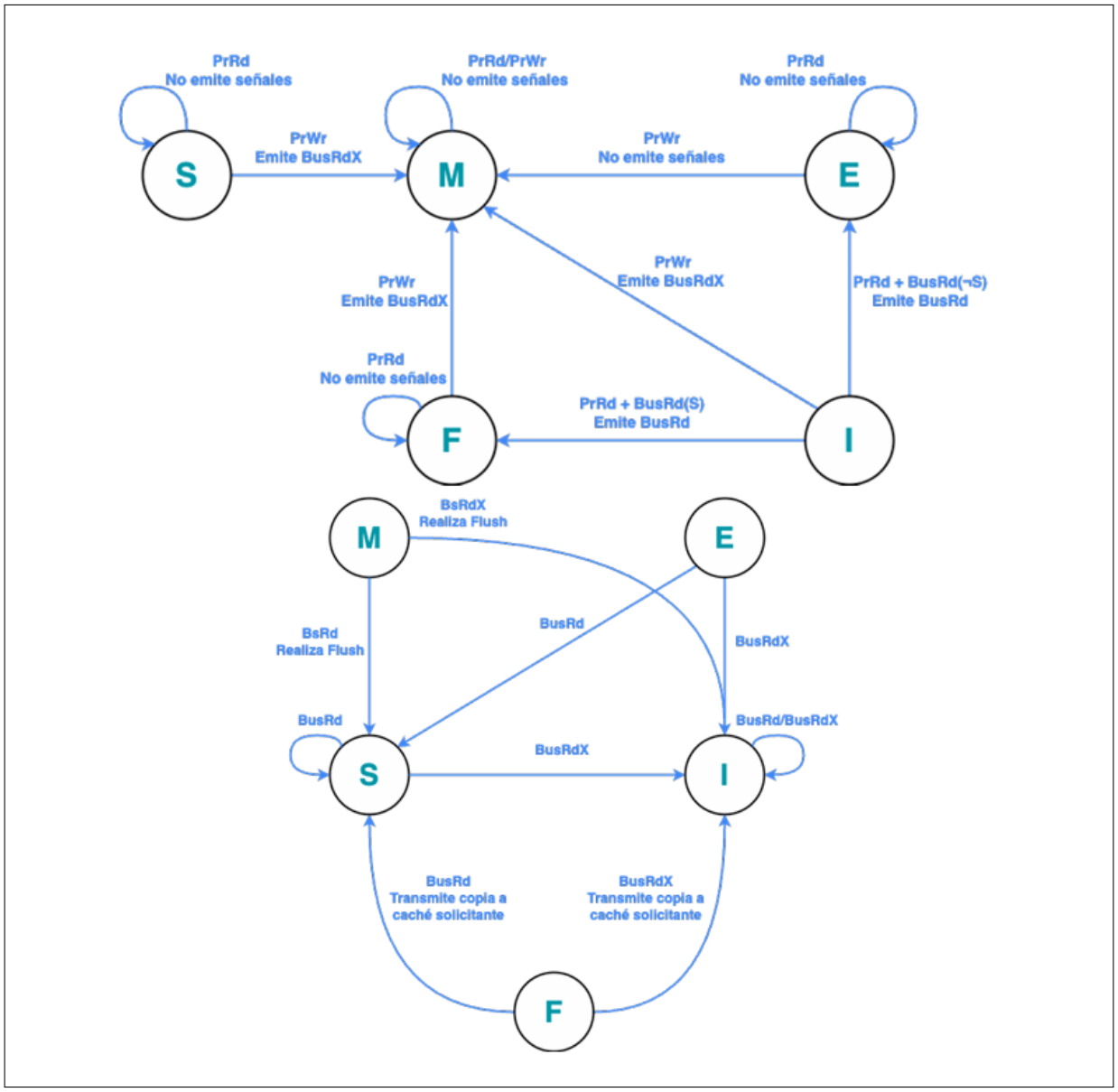
Figura 1: Diagramas MESI

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 las eñal `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:


PrRd
No emite señales
PrRd/PrWr
No emite señales
PrRd
No emite señales
S
PrWr
Emite BusRdX
M
PrWr
No emite señales
E
PrWr
Emite BusRdX
PrWr
Emite BusRdX
PrRd + BusRd(¬S)
Emite BusRd
PrRd
No emite señales
F
PrRd + BusRd(S)
Emite BusRd
I
M
BsRdX
Realiza Flush
E
BsRd
Realiza Flush
BusRd
BusRdX
BusRd
BusRd/BusRdX
S
BusRdX
I
BusRd
Transmite copia a
caché solicitante
BusRdX
Transmite copia a
caché solicitante
F

Feedback ayudantía

Escanee el QR para entregar *feedback* sobre la ayudantía.