
DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343

# Arquitectura de Computadores

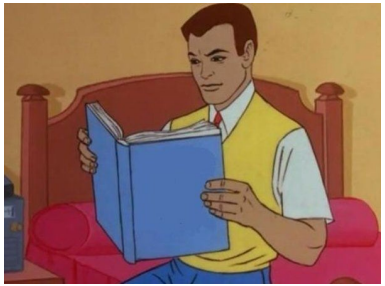
## Clase 11 - Coherencia de Caché

Profesor: Germán Leandro Contreras Sagredo

# Bibliografía


- **Apuntes históricos. Hans Löbel, Alejandro Echeverría**
  - 12 - Paralelismo Avanzado

- **D. Patterson, Computer Organization and Design RISC-V Edition: The Hardware Software Interface. Morgan Kaufmann, 2020.**
  - Capítulo 5.12. Página 459, 586 en PDF.*

* Estos se basan en la implementación de una arquitectura que implementa RISC-V. Tomar solo como referencia.

## Objetivos de la clase

- Conocer los problemas que surgen en arquitecturas con múltiples procesadores con memoria compartida.

- Conocer los mecanismos de coherencia de caché en arquitecturas de memoria compartida.

## Hasta ahora...

- Ya construimos un computador que tiene todo lo necesario para que pueda ser operado y programado, además de implementar lo necesario para que se comunique con otros dispositivos.

- Además, vimos cómo optimizar los accesos a memoria mediante cachés.

Ahora, veremos un caso particular: el manejo de caché en un sistema con más de un procesador. No obstante, primero debemos saber en qué consisten estos sistemas.

# Sistemas multiprocesador con memoria compartida

En estos sistemas, los procesadores comparten en *algún nivel* su memoria, lo que facilita la transferencia de datos entre ellos. Cada uno podría tener su propia jerarquía de memoria (*i.e.* su propia caché) y compartirla en niveles más altos. Si todos los procesadores se incorporan en un único chip, hablamos de un procesador ***multicore***.

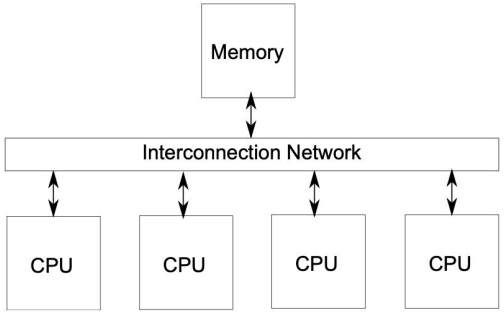
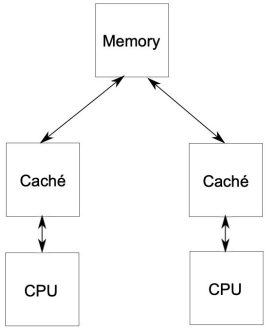
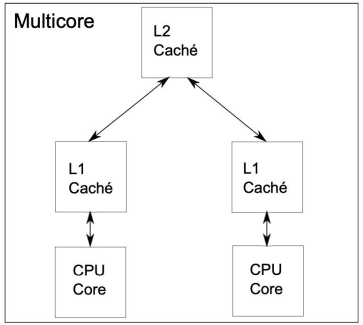



Diagrama general de un sistema multiprocesador con memoria compartida.




Memoria compartida en el segundo nivel, independiente en el primer nivel.

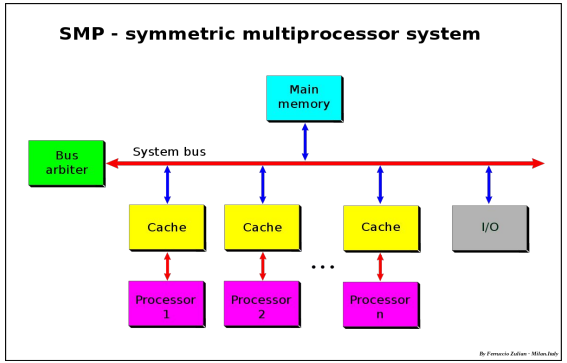



Procesador *multicore* con memoria compartida.

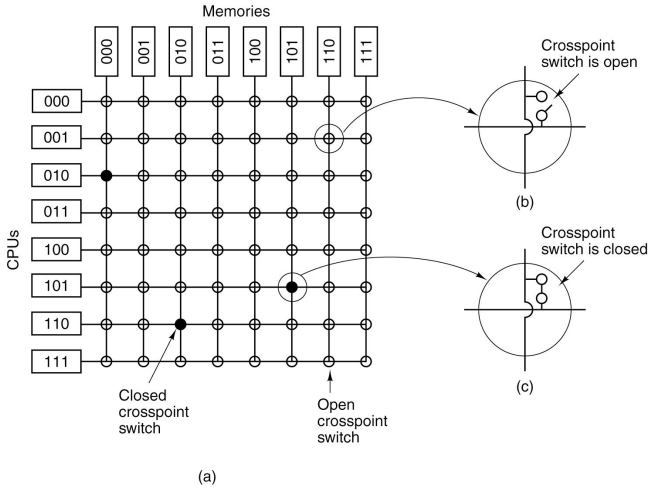
# Sistemas multiprocesador con memoria compartida

## Esquemas de memoria compartida según tipo de acceso

***Uniform Memory Access* (UMA):** Todos los procesadores acceden **de la misma forma** a la memoria. Puede ser a través de un único bus o un ***crossbar switch***.




Ejemplo de arquitectura UMA:
*Symmetric Multiprocessor System* (SMP)




Con múltiples procesadores, el problema de múltiples accesos a memoria se vuelve relevante. En este caso, la memoria se puede dividir en distintos módulos y los *crossbar switches* habilitan el acceso de memoria al módulo si ningún procesador está accediendo a él.

# Sistemas multiprocesador con memoria compartida

## Esquemas de memoria compartida según tipo de acceso

***Non-Uniform Memory Access* (NUMA):** Los procesadores se reparten en nodos donde cada uno puede acceder a un segmento específico de la memoria (memoria local). Pueden acceder a espacios de memoria asignados a otros nodos pero con una latencia de acceso mayor (memoria remota).

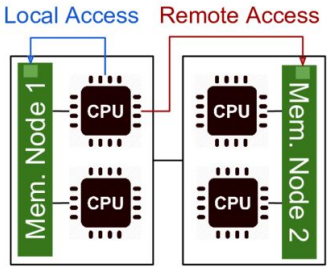



Diagrama general de un esquema NUMA.

## Tipos de arquitectura paralela - Memoria compartida

En ambos esquemas de memoria compartida, se pueden agilizar los accesos a través de memorias caché (con uno o más niveles).

Si las caché siguen una política de escritura ***write-through***, en general no habrá problemas ya que la memoria será consistente para todos los procesadores, pero cada escritura implica un *overhead* significativo que aumenta el tiempo de ejecución de los programas.

Por lo tanto, se opta por seguir el protocolo de escritura ***write-back***, pero de este nace un nuevo problema: la **coherencia de caché**.

## Coherencia de caché

Entenderemos como **coherencia de caché** a la consistencia en la memoria caché de dos o más procesadores distintos que leen o escriben el dato de una misma dirección de memoria.

Si un procesador escribe en una dirección de memoria bajo el protocolo *write-back*, debemos asegurar que los otros procesadores que accedan al mismo dato lean la versión **más actualizada** para asegurar consistencia en la ejecución de programas.

Para asegurarla, implementaremos protocolos que serán adoptados por el **controlador de caché**.

## Coherencia de caché - *Snooping y Snarfing*

Para adoptar los protocolos señalados, los controladores de caché deben revisar las solicitudes de lectura y escritura que se realizan a sus contrapartes a través de un bus compartido. Este concepto se conoce como ***bus snooping***.

También es posible implementar protocolos con ***snarfing***: cuando un controlador de caché detecta una solicitud de escritura a una dirección de memoria que contiene, actualiza **inmediatamente** el contenido de su caché con los datos de la solicitud. No obstante, esto se utiliza menos por el aumento de latencia en los accesos a memoria y el aumento de tráfico en el bus compartido de datos.

# Coherencia de caché - Protocolo MSI

Este protocolo se basa en el uso de estados adicionales al de validez para las líneas de caché, lo que permite detectar errores de coherencia. Su nombre viene de los tres estados posibles:

- ***Modified***: Indica que el contenido de la línea fue modificado y **no es consistente** con el bloque de la memoria principal. Este puede existir **solo en una caché** para un mismo bloque compartido.
- ***Shared***: Indica que el contenido de la línea se encuentra **en al menos una memoria caché**, pero no se ha modificado.
- ***Invalid***: Indica que el contenido de la línea es inválido, que puede ser por dato no existente o por solicitud desde el bus compartido.

# Coherencia de caché - Protocolo MSI

Para entender el protocolo, se definen los siguientes **eventos**:

- **Eventos del procesador actual**
  - `PrRd`: Solicitud de lectura (*Read* - `Rd`).
  - `PrWr`:. Solicitud de escritura (*Write* - `Wr`).
- **Eventos del bus compartido**
  - `BusRd`: Solicitud de un bloque de datos **sin** intención de modificarlo.
  - `BusRdX`:. Solicitud de un bloque de datos **con** intención de modificarlo.
  - `Flush`: Transferencia de línea de caché a la memoria principal.

# Coherencia de caché - Protocolo MSI

## Cambios de estado de línea de caché por eventos del procesador

- Si se hace la lectura de un dato que no está en memoria o que se invalidó por una modificación (estado ***Invalid***), entonces se lee el bloque de memoria y se copia; el estado de la línea de caché pasa a estado ***Shared*** y se emite la señal `BusRd` al bus compartido.

- Si se hace una lectura en estado ***Shared*** o lectura/escritura en estado ***Modified***, no se emite ninguna señal en el bus compartido y se realiza la operación directamente sobre la caché.

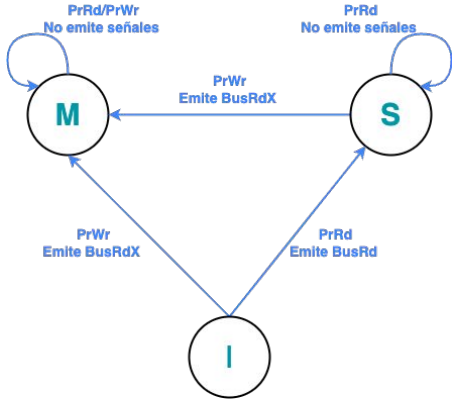



Diagrama de estado de una línea de caché para el protocolo MSI y eventos del procesador.

# Coherencia de caché - Protocolo MSI

## Cambios de estado de línea de caché por eventos del procesador

- Si se hace una escritura en estado ***Invalid***, se lee el bloque de memoria y se copia; se modifica el contenido directamente en caché; el estado de la línea pasa a estado ***Modified*** y se emite la señal `BusRdX` al bus compartido.

- Si se hace una escritura en estado ***Shared***, se realizan las mismas acciones pero se evita la copia del bloque de memoria al ya encontrarse en caché.

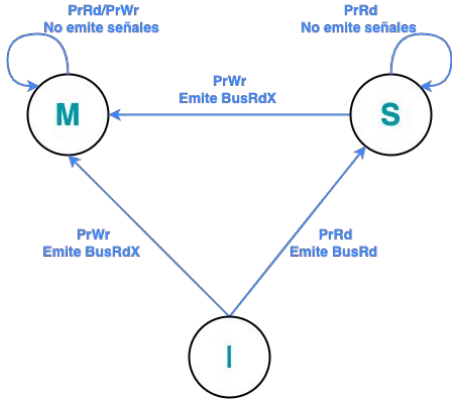



Diagrama de estado de una línea de caché para el protocolo MSI y eventos del procesador.

# Coherencia de caché - Protocolo MSI

## Cambios de estado de línea de caché por eventos del bus

- Si un controlador recibe la señal `BusRd` o `BusRdX` para una de sus líneas en estado ***Invalid***, mantiene su estado sin hacer nada. Lo mismo ocurre para la señal `BusRd` y una línea en estado ***Shared***.

- Si un controlador recibe la señal `BusRdX` para una de sus líneas en estado ***Shared***, significa que su contenido dejará de ser válido porque el contenido se modificará, por lo que se actualiza a estado ***Invalid***.

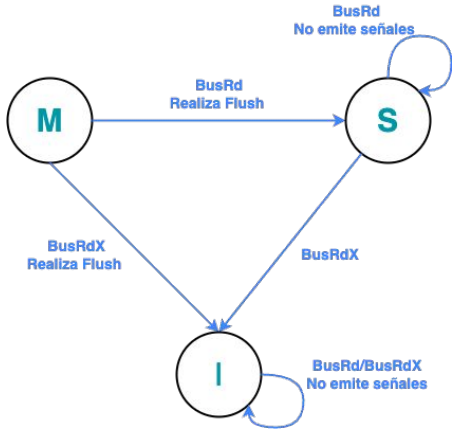



Diagrama de estado de una línea de caché para el protocolo MSI y eventos del bus compartido.

# Coherencia de caché - Protocolo MSI

## Cambios de estado de línea de caché por eventos del bus

- Si un controlador recibe la señal `BusRd` para una de sus líneas en estado ***Modified***, actualiza su estado a ***Shared*** y realiza `Flush` (escritura de la línea en la memoria principal) para que el controlador que emitió la señal pueda hacer la lectura correcta de memoria.

- Si un controlador recibe la señal `BusRdX` para una de sus líneas en estado ***Modified***, también realiza `Flush` pero pasa a estado ***Invalid*** ya que el controlador que emitió la señal será el único habilitado para poseer la línea modificada.

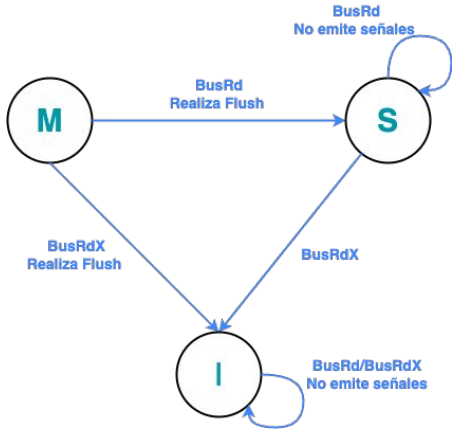



Diagrama de estado de una línea de caché para el protocolo MSI y eventos del bus compartido.

# Coherencia de caché - Protocolo MSI

## Diagrama de estado general

- Incluyendo ambos tipos de cambio, así se ve el diagrama de estado de una línea de caché para el protocolo MSI.

- Algunas implementaciones permiten que un bloque que ya se encuentre compartido pueda ser transferido por el bus para evitar una lectura de la memoria principal (evento `BusUpgr`).

- **Problema del protocolo:** Usamos el estado ***Shared*** aunque solo una caché posea el bloque.

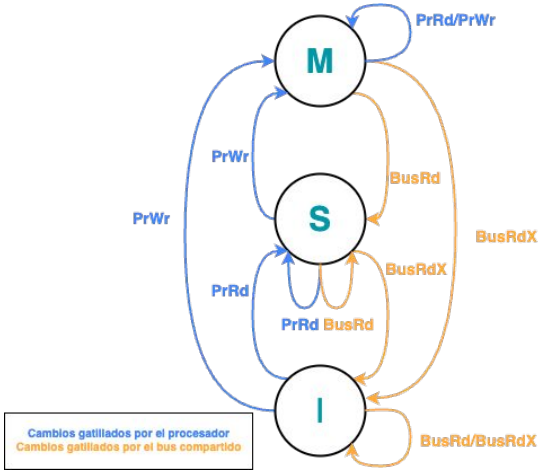



Diagrama de estado de una línea de caché para el protocolo MSI.

# Coherencia de caché - Protocolo MESI

Este protocolo es una mejora sobre MSI, desarrollado en la Universidad de Illinois (y, por ende, también conocido como "Protocolo Illinois"). En este protocolo, el estado ***Shared*** cambia su uso y se agrega otro nuevo:

- ***S**hared*: Indica que el contenido de la línea se encuentra en **más de una caché** (al menos dos).

- ***E**xclusive*: Indica que el contenido de la línea se encuentra **exclusivamente en dicha caché**, sin modificaciones.

## Coherencia de caché - Protocolo MESI

Además de los eventos anteriores, se añade la confirmación (o no) de que el contenido de una línea de caché exista o no en otra:

- `BusRd(S)`: Indica que se realiza una lectura y se obtuvo una confirmación **afirmativa** de la existencia del contenido en otra caché.

- `BusRd(¬S)`: Indica que se realiza una lectura y se obtuvo una confirmación **negativa** de la existencia del contenido en otra caché (por lo que sabemos que solo existirá en la caché actual).

# Coherencia de caché - Protocolo MESI

## Cambios de estado de línea de caché por eventos del procesador

- La mayoría de las transiciones queda igual, así que veremos las que involucran al nuevo estado.

- Si se hace la lectura de un dato que no está en memoria y que se confirma que no está en ninguna otra caché, el estado de la línea pasa a estado ***Exclusive***. Si se realiza una lectura en este estado, se mantiene igual y no se emiten señales. Si se realiza una escritura, pasa a estado ***Modified*** pero no envía ninguna señal al bus compartido.

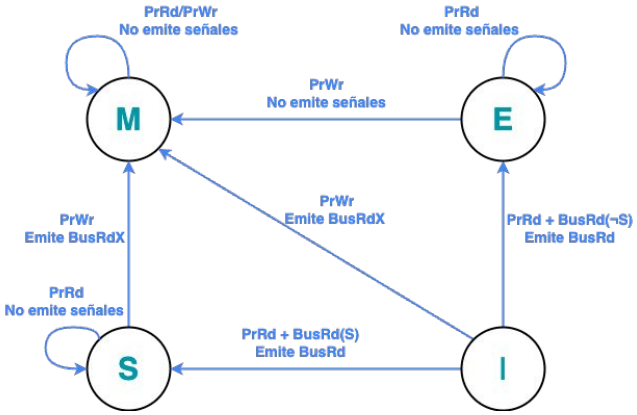



Diagrama de estado de una línea de caché para el protocolo MESI y eventos del procesador.

# Coherencia de caché - Protocolo MESI

## Cambios de estado de línea de caché por eventos del bus

- La mayoría de las transiciones queda igual, así que veremos las que involucran al nuevo estado.

- Si un controlador recibe la señal `BusRd` para una de sus líneas en estado ***Exclusive***, entonces esta ya no es la única y por ende cambia a estado ***Shared***. Si en cambio recibe la señal `BusRdX`, pasa a estado ***Invalid*** ya que no solo no será la única caché que tiene el contenido, sino que no tendrá la última versión de este.

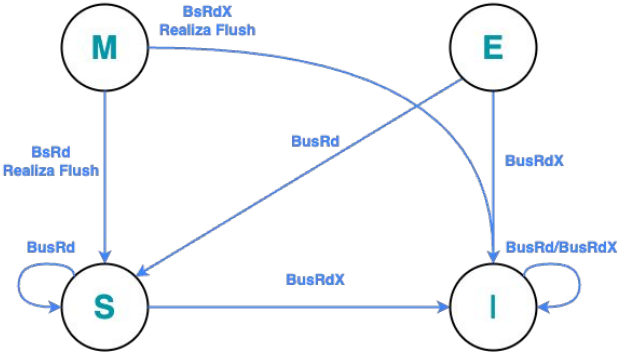



Diagrama de estado de una línea de caché para el protocolo MESI y eventos del bus compartido.

# Coherencia de caché - Protocolo MESI

## Diagrama de estado general

- Incluyendo ambos tipos de cambio, así se ve el diagrama de estado de una línea de caché para el protocolo MESI.

- El hecho de no gatillar la señal `BusRdX` para todo caso de escritura implica una ganancia importante de eficiencia respecto al tráfico de bus compartido y revisiones que deben hacer los controladores. Esta ganancia es aún mayor si se implementa *snarfing*.

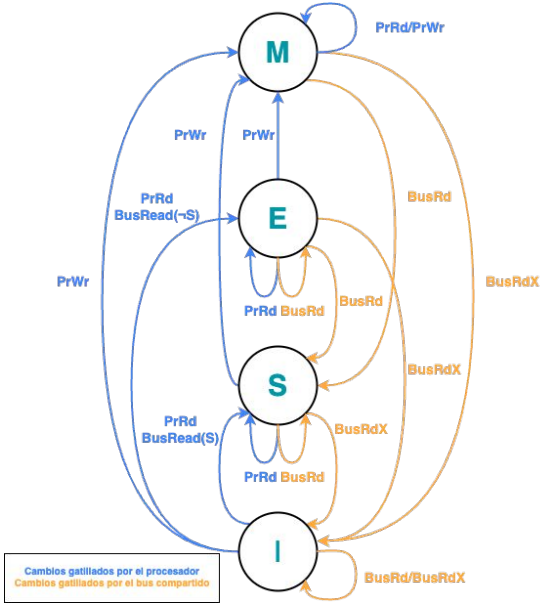



Diagrama de estado de una línea de caché para el protocolo MESI.

# Coherencia de caché - Protocolo MESI

## Ejercicio en clases

Considere la siguiente memoria compartida con los datos señalados en la tabla.

| Dirección | *Label* | Valor |
|---|---|---|
| 0xCD | i | 5 |
| 0xCE | temp | -45 |
| 0xCF | var1 | 2 |
| 0xD0 | var2 | 9 |
| 0xD1 | arr | 20 |
| 0xD2 | | -10 |
| 0xD3 | | 3 |
| 0xD4 | | 9 |

Esta es compartida por tres CPUs que ejecutan simultáneamente los siguientes programas:

**CPU0**

```
MOV A,(var1)
MOV B,(var2)
ADD A,B
MOV (arr),A
ADD A,(i)
MOV (temp),A
```

**CPU1**

```
MOV A,arr
ADD A,1
MOV (temp),A
MOV B,(temp)
MOV B,(B)
MOV (var1),B
```

**CPU2**

```
MOV B,(i)
MOV A,(arr)
ADD A,B
MOV (i),A
INC B
MOV (var2),B
```

# Coherencia de caché - Protocolo MESI

## Ejercicio en clases

Indique los valores finales de las variables para cada caché privada y complete las siguientes tablas según cómo se actualiza cada celda a partir del protocolo MESI (indicando con una letra qué bit está activo). Puede asumir que cada variable se almacena en una línea de la caché.

| | CPU0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

| | CPU1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

| | CPU2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | | | | | | | | |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

# Coherencia de caché - Protocolo MESI

## Ejercicio en clases

### Ejecución de instrucción 1

- CPU0: `MOV A,(var1)`
- CPU1: `MOV A,arr`
- CPU2: `MOV B,(i)`

**CPU0**

```
i = -
temp = -
var1 = 2
var2 = -
arr[0] = -
arr[1] = -
A = 2
B = -
```

**CPU1**

```
i = -
temp = -
var1 = -
var2 = -
arr[0] = -
arr[1] = -
A = 0xD1
B = -
```

**CPU2**

```
i = 5
temp = -
var1 = -
var2 = -
arr[0] = -
arr[1] = -
A = -
B = 5
```

| | CPU0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | I | I | E | I | I | I | I | I |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

| | CPU1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | I | I | I | I | I | I | I | I |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

| | CPU2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | E | I | I | I | I | I | I | I |
| 2 | | | | | | | | |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

CPU0 y CPU2 parten con una línea exclusiva ya que son los únicos que poseen el dato. CPU1 ejecuta un `MOV A,Lit`, por lo que no lee de memoria.

# Coherencia de caché - Protocolo MESI

## Ejercicio en clases

### Ejecución de instrucción 2

- CPU0: `MOV B,(var2)`
- CPU1: `ADD A,1`
- CPU2: `MOV A,(arr)`

**CPU0**

```
i = -
temp = -
var1 = 2
var2 = 9
arr[0] = -
arr[1] = -
A = 2
B = 9
```

**CPU1**

```
i = -
temp = -
var1 = -
var2 = -
arr[0] = -
arr[1] = -
A = 0xD2
B = -
```

**CPU2**

```
i = 5
temp = -
var1 = -
var2 = -
arr[0] = 20
arr[1] = -
A = 20
B = 5
```

| | CPU0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | I | I | E | I | I | I | I | I |
| 2 | I | I | E | E | I | I | I | I |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

| | CPU1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | I | I | I | I | I | I | I | I |
| 2 | I | I | I | I | I | I | I | I |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

| | CPU2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | E | I | I | I | I | I | I | I |
| 2 | E | I | I | I | E | I | I | I |
| 3 | | | | | | | | |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

CPU0 y CPU2 ahora poseen dos líneas exclusivas y CPU1 sigue con todas sus líneas inválidas.

# Coherencia de caché - Protocolo MESI

## Ejercicio en clases

### Ejecución de instrucción 3

- CPU0: `ADD A,B`
- CPU1: `MOV (temp),A`
- CPU2: `ADD A,B`

**CPU0**

```
i = -
temp = -
var1 = 2
var2 = 9
arr[0] = -
arr[1] = -
A = 11
B = 9
```

**CPU1**

```
i = -
temp = 0xD2
var1 = -
var2 = -
arr[0] = -
arr[1] = -
A = 0xD2
B = -
```

**CPU2**

```
i = 5
temp = -
var1 = -
var2 = -
arr[0] = 20
arr[1] = -
A = 25
B = 5
```

| | CPU0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | I | I | E | I | I | I | I | I |
| 2 | I | I | E | E | I | I | I | I |
| 3 | I | I | E | E | I | I | I | I |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

| | CPU1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | I | I | I | I | I | I | I | I |
| 2 | I | I | I | I | I | I | I | I |
| 3 | I | M | I | I | I | I | I | I |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

| | CPU2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | E | I | I | I | I | I | I | I |
| 2 | E | I | I | I | E | I | I | I |
| 3 | E | I | I | I | E | I | I | I |
| 4 | | | | | | | | |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

CPU1 almacena directamente la línea de temp en estado M por escribir sobre ella; no se invalidan en CPU0 ni CPU2 ya que no la poseen.

# Coherencia de caché - Protocolo MESI

## Ejercicio en clases

### Ejecución de instrucción 4

- CPU0: `MOV (arr),A`
- CPU1: `MOV B,(temp)`
- CPU2: `MOV (i),A`

**CPU0**

```
i = -
temp = -
var1 = 2
var2 = 9
arr[0] = 11
arr[1] = -
A = 11
B = 9
```

**CPU1**

```
i = -
temp = 0xD2
var1 = -
var2 = -
arr[0] = -
arr[1] = -
A = 0xD2
B = 0xD2
```

**CPU2**

```
i = 25
temp = -
var1 = -
var2 = -
arr[0] = 20
arr[1] = -
A = 25
B = 5
```

| | CPU0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | I | I | E | I | I | I | I | I |
| 2 | I | I | E | E | I | I | I | I |
| 3 | I | I | E | E | I | I | I | I |
| 4 | I | I | E | E | M | I | I | I |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

| | CPU1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | I | I | I | I | I | I | I | I |
| 2 | I | I | I | I | I | I | I | I |
| 3 | I | M | I | I | I | I | I | I |
| 4 | I | M | I | I | I | I | I | I |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

| | CPU2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | E | I | I | I | I | I | I | I |
| 2 | E | I | I | I | E | I | I | I |
| 3 | E | I | I | I | E | I | I | I |
| 4 | M | I | I | I | I | I | I | I |
| 5 | | | | | | | | |
| 6 | | | | | | | | |

CPU1 lee de su caché una línea que solo posee él, no invalida nada. CPU2 pasa a estado M para la línea de i, no invalida nada. CPU0 modifica arr[0] y almacena en su línea el estado M; invalida el contenido de la caché de CPU2.

# Coherencia de caché - Protocolo MESI

## Ejercicio en clases

### Ejecución de instrucción 5

- CPU0: `ADD A,(i)`
- CPU1: `MOV B,(B)`
- CPU2: `INC B`

**CPU0**

```
i = 25
temp = -
var1 = 2
var2 = 9
arr[0] = 11
arr[1] = -
A = 36
B = 9
```

**CPU1**

```
i = -
temp = 0xD2
var1 = -
var2 = -
arr[0] = -
arr[1] = -10
A = 0xD2
B = -10
```

**CPU2**

```
i = 25
temp = -
var1 = -
var2 = -
arr[0] = 20
arr[1] = -
A = 25
B = 6
```

| | CPU0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | I | I | E | I | I | I | I | I |
| 2 | I | I | E | E | I | I | I | I |
| 3 | I | I | E | E | I | I | I | I |
| 4 | I | I | E | E | M | I | I | I |
| 5 | S | I | E | E | M | I | I | I |
| 6 | | | | | | | | |

| | CPU1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | I | I | I | I | I | I | I | I |
| 2 | I | I | I | I | I | I | I | I |
| 3 | I | M | I | I | I | I | I | I |
| 4 | I | M | I | I | I | I | I | I |
| 5 | I | M | I | I | I | E | I | I |
| 6 | | | | | | | | |

| | CPU2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | E | I | I | I | I | I | I | I |
| 2 | E | I | I | I | E | I | I | I |
| 3 | E | I | I | I | E | I | I | I |
| 4 | M | I | I | I | I | I | I | I |
| 5 | S | I | I | I | I | I | I | I |
| 6 | | | | | | | | |

CPU1 queda con línea exclusiva de arr[1]. Por la lectura de i de CPU0, su línea pasa a estado S mientras que la línea de CPU2 también pasa a estado S, haciendo *flush* para que CPU0 pueda leer el contenido modificado.

# Coherencia de caché - Protocolo MESI

## Ejercicio en clases

### Ejecución de instrucción 6

- CPU0: `MOV (temp),A`
- CPU1: `MOV (var1),B`
- CPU2: `MOV (var2),B`

**CPU0**

```
i = 25
temp = 36
var1 = 2
var2 = 9
arr[0] = 11
arr[1] = -
A = 36
B = 9
```

**CPU1**

```
i = -
temp = 0xD2
var1 = -10
var2 = -
arr[0] = -
arr[1] = -10
A = 0xD2
B = -10
```

**CPU2**

```
i = 25
temp = -
var1 = -
var2 = 6
arr[0] = 20
arr[1] = -
A = 25
B = 6
```

| | CPU0 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | I | I | E | I | I | I | I | I |
| 2 | I | I | E | E | I | I | I | I |
| 3 | I | I | E | E | I | I | I | I |
| 4 | I | I | E | E | M | I | I | I |
| 5 | S | I | E | E | M | I | I | I |
| 6 | S | M | I | I | M | I | I | I |

| | CPU1 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | I | I | I | I | I | I | I | I |
| 2 | I | I | I | I | I | I | I | I |
| 3 | I | M | I | I | I | I | I | I |
| 4 | I | M | I | I | I | I | I | I |
| 5 | I | M | I | I | I | E | I | I |
| 6 | I | I | M | I | I | E | I | I |

| | CPU2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Instrucción | i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| 1 | E | I | I | I | I | I | I | I |
| 2 | E | I | I | I | E | I | I | I |
| 3 | E | I | I | I | E | I | I | I |
| 4 | M | I | I | I | I | I | I | I |
| 5 | S | I | I | I | I | I | I | I |
| 6 | S | I | I | M | I | I | I | I |

CPU0 queda con la línea temp modificada y se invalida en CPU1. CPU1 queda con la línea var1 modificada y se invalida en CPU0. CPU2 queda con la línea var2 modificada y se invalida en CPU0.

# Coherencia de caché - Protocolo MESI

## Ejercicio en clases

### Estado final

Se destaca que las variables **rojas** señalan que la línea quedó con el contenido de estas, pero fue invalidado por modificaciones de otras caché.

| CPU0 | CPU1 | CPU2 |
|---|---|---|
| i = 25 | i = - | i = 25 |
| temp = 36 | temp = 0xD2 | temp = - |
| var1 = 2 | var1 = -10 | var1 = - |
| var2 = 9 | var2 = - | var2 = 6 |
| arr[0] = 11 | arr[0] = - | arr[0] = 20 |
| arr[1] = - | arr[1] = -10 | arr[1] = - |
| A = 36 | A = 0xD2 | A = 25 |
| B = 9 | B = -10 | B = 6 |

| CPU0 | | | | | | | |
|---|---|---|---|---|---|---|---|
| i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| S | M | I | I | M | I | I | I |

| CPU1 | | | | | | | |
|---|---|---|---|---|---|---|---|
| i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| I | I | M | I | I | E | I | I |

| CPU2 | | | | | | | |
|---|---|---|---|---|---|---|---|
| i | temp | var1 | var2 | arr[0] | arr[1] | arr[2] | arr[3] |
| S | I | I | M | I | I | I | I |

Notar que no se incluyen `arr[2]` ni `arr[3]` porque nunca se leen ni modifican.

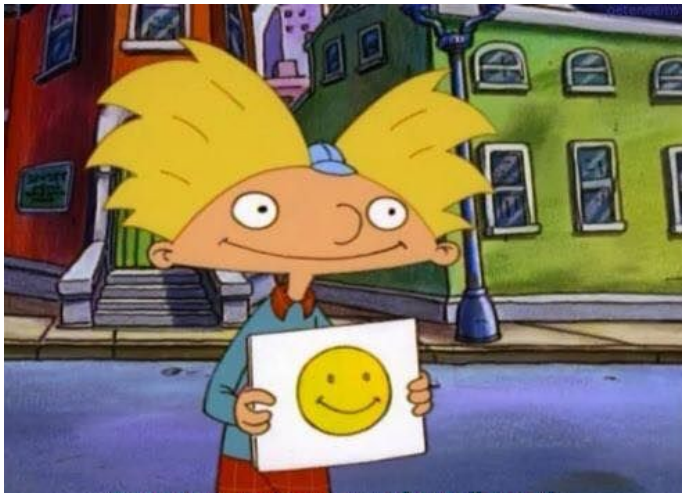
## Coherencia de caché - Transferencia entre cachés

Una mejora que se puede hacer en los protocolos es habilitar la transferencia de datos entre cachés. Así, si un controlador solicita un bloque que ya se encuentra contenido en una caché, su contenido se obtiene a través del bus compartido y no de la memoria principal.

Esto permite nuevos protocolos, por ejemplo, **MOESI**. En este, se agrega el estado ***Ownership***. Si un procesador modifica una línea de la caché y otro hace una lectura de este contenido, el procesador actual se vuelve "dueño" de la línea y, posteriormente, se encarga de transmitirlo a otras caché hasta realizar una nueva modificación; o bien hasta que un procesador distinto realice una escritura sobre ella.

# Coherencia de caché

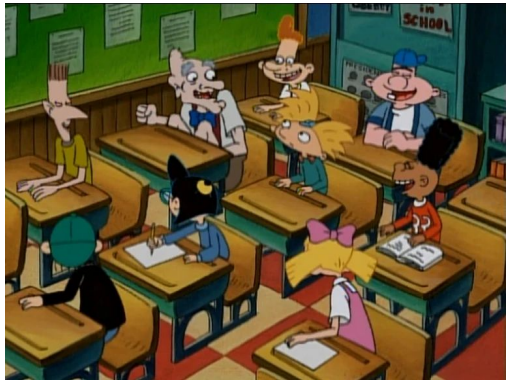
Existen más mejoras y variantes de protocolos que se pueden implementar, por ejemplo: **MOSI**, **MESIF**, **MOESIF**.



# Ejercicios

Ahora, veremos algunos ejercicios.

Estos se basan en preguntas de tareas y pruebas de semestres anteriores, por lo que nos servirán de preparación para las evaluaciones.



# Ejercicios

Suponga que posee una arquitectura MIMD de memoria compartida con 2 CPU que poseen su propia caché y que siguen el protocolo MESI para asegurar consistencia. La memoria contiene los siguientes datos de variables y las CPU0 y CPU1 ejecutan los siguientes programas:

| Dirección | *Label* | Valor |
|---|---|---|
| 0x00 | i | 1 |
| 0x01 | arr | 1 |
| 0x02 | | 10 |

```
// CPU0          // CPU1
MOV B,(i)        MOV A,(arr)
MOV A,(arr)      MOV B,arr
ADD B,A          ADD B,A
SHR A,A          MOV (B),A
MOV (B),A        NOP
```

Asumiendo que cada dirección se almacena en una línea distinta, indique el estado de cada línea de las caché (M/E/S/I) para cada iteración completando la tabla adjunta, asumiendo que todas las líneas parten en estado I:

| | CPU0 | | | CPU1 | | |
|---|---|---|---|---|---|---|
| **Instrucción** | **i** | **arr[0]** | **arr[1]** | **i** | **arr[0]** | **arr[1]** |
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |

***Examen*, 2023-1**

# Ejercicios

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 iteración 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 | | | | | | | | | | | | |

***Interrogación 3, 2024-2***

# Ejercicios

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.

## *Interrogación 3, 2024-1*

## Antes de terminar

¿Dudas?

¿Consultas?

¿Inquietudes?

¿Comentarios?

# Finalmente...

¡Damos por finalizado el contenido del curso!

# Anexo - Resolución de ejercicios

**¡Importante!**

Estos ejercicios pueden tener más de un desarrollo correcto. Las respuestas a continuación no son más que soluciones que **no excluyen** otras alternativas igual de correctas.

# Ejercicios - Respuesta

Suponga que posee una arquitectura MIMD de memoria compartida con 2 CPU que poseen su propia caché y que siguen el protocolo MESI para asegurar consistencia. La memoria contiene los siguientes datos de variables y las CPU0 y CPU1 ejecutan los siguientes programas:

| Dirección | *Label* | Valor |
|---|---|---|
| 0x00 | i | 1 |
| 0x01 | arr | 1 |
| 0x02 | | 10 |

```
// CPU0          // CPU1
MOV B,(i)        MOV A,(arr)
MOV A,(arr)      MOV B,arr
ADD B,A          ADD B,A
SHR A,A          MOV (B),A
MOV (B),A        NOP
```

Asumiendo que cada dirección se almacena en una línea distinta, indique el estado de cada línea de las caché (M/E/S/I) para cada iteración completando la tabla adjunta, asumiendo que todas las líneas parten en estado I:

| | CPU0 | | | CPU1 | | |
|---|---|---|---|---|---|---|
| **Instrucción** | **i** | **arr[0]** | **arr[1]** | **i** | **arr[0]** | **arr[1]** |
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |

**Respuesta en la siguiente diapositiva.**

# Ejercicios - Respuesta

A continuación, se muestra la tabla final esperada, que incluye columnas para los valores de los registros `A` y `B` y el valor en memoria de las variables en cada iteración:

| | CPU0 | | | | | CPU1 | | | | | Memoria | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Instrucción** | **i** | **arr[0]** | **arr[1]** | **A** | **B** | **i** | **arr[0]** | **arr[1]** | **A** | **B** | **i** | **arr[0]** | **arr[1]** |
| 1 | E | I | I | - | 1 | I | E | I | 1 | - | 1 | 1 | 10 |
| 2 | E | S | I | 1 | 1 | I | S | I | 1 | 1 | 1 | 1 | 10 |
| 3 | E | S | I | 1 | 2 | I | S | I | 1 | 2 | 1 | 1 | 10 |
| 4 | E | S | I | 0 | 2 | I | S | M | 1 | 2 | 1 | 1 | 1 |
| 5 | E | S | M | 0 | 2 | I | S | I | 1 | 2 | 1 | 1 | 0 |

# Ejercicios - Respuesta

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 iteración completando la tabla adjunta, asumiendo que todas las líneas parten en estado I (ciclo 0):

**Respuesta en la siguiente diapositiva.**

| | 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 | | | | | | | | | | | | |

# Ejercicios - Respuesta

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 |

# Ejercicios - Respuesta

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 v.
- 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.

# Ejercicios - Respuesta

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.

## Respuesta en la siguiente diapositiva.

# Ejercicios - Respuesta

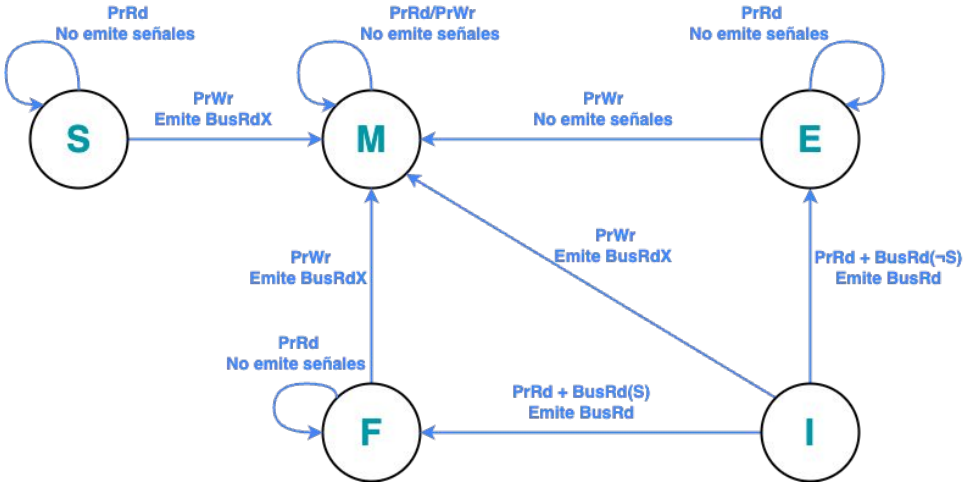
Se indicarán los cambios de estado de **F** según cada tipo de evento:

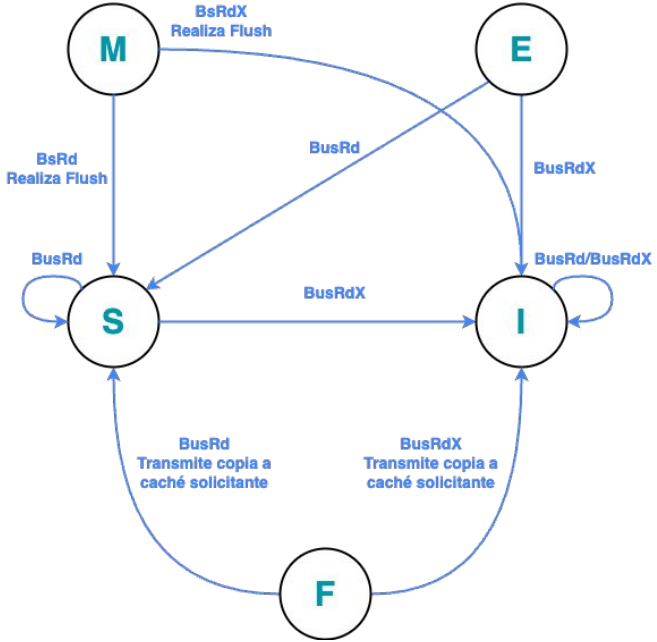
- **PrRd:** El procesador actual solicita una lectura sobre la línea en estado **F**. Esto no implica transmisión alguna de la copia a otra caché, por lo que se mantiene en este estado. No es necesario que emita señales.
- **PrWr:** El procesador actual solicita una escritura sobre la línea en estado **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 trasmitir 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**.

# Ejercicios - Respuesta

A continuación, los diagramas resultantes:




DCC

DEPARTAMENTO DE CIENCIA
DE LA COMPUTACIÓN

IIC2343

# Arquitectura de Computadores

## Clase 11 - Coherencia de Caché

Profesor: Germán Leandro Contreras Sagredo