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

IIC2343 - Arquitectura de Computadores (II/2025)

Guía de ejercicios: Caché y Coherencia de Caché

Ayudantes: Daniela Ríos (danielaarp@uc.cl), Alberto Maturana (alberto.maturana@uc.cl), Tomás López Massaro (tomas.lopezm20@uc.cl), Ignacio Gajardo (gajardo.ignacio@uc.cl)

Pregunta 1: Preguntas Conceptuales

(a) **[P1b | I3-2013-1]** Describa las tres funciones de correspondencia vistas en clase.

Solución:

- ***Directly Mapped***: Cada bloque de la memoria principal sólo puede almacenarse en un bloque fijo de la memoria *caché*, definido de acuerdo al tamaño de la *caché* y el tamaño de los bloques de memoria.
- ***Fully Associative***: Cada bloque de la memoria principal puede ubicarse en vualquier bloque de la memoria *caché*.
- ***N-way Associative***: La memoria *caché* es dividida en conjuntos de N bloques. Luego, cada bloque de la memoria principal sólo puede ubicarse en un conjunto de la *caché*, pero en ese conjunto puede ubicarse en cualquier bloque.

(b) **[P1b | I3-2014-2]** El algoritmo de reemplazo MRU *(Most Recently Used)*, a diferencia de LRU *(Least Recently Used)*, descarta primero los elementos que han sido ocupados más recientemente. ¿En qué casos podría ser útil el uso de este esquema?

Solución: Un caso específico en el que puede ser útil es en la iteración constante de un arreglo. Supongamos que el arreglo completo no cabe en la *caché*, así que una vez que se nos agote el espacio, reemplazamos la última posición iterada por la siguiente a acceder. Hacemos esto hasta llegar al final y luego, al reiniciar, tendremos en memoria una porción considerable del mismo, dándonos un buen *hit-rate* (que concluirá al llegar a la posición donde se realizó la primera reescritura).

Notar que este caso convendría siempre y cuando la memoria *caché* pueda contener más de la mitad del arreglo (en otro caso, el *miss rate* sería mayor).

(c) **[P2b | I3-2015-2]** ¿Por qué es importante que algunos niveles de memoria *caché* sean exclusivos para cada núcleo y otros compartidos?

Solución: Es muy posible que cada núcleo ejecute programas distintos, por lo tanto es necesario que los datos e instrucciones que usan se mantengan almacenadas de manera independiente a las de otro núcleo. Así, estas no tienen conflicto por el espacio en la memoria *caché*.

Por otro lado, los programas necesitan generalmente compartir datos, como por ejemplo los parámetros para llamadas a subrutinas. Así, para evitar usar la memoria principal (más lenta), para compartir estos datos se utiliza una memoria *caché* ubicada en la parte más baja de la jerarquía de memoria.

(d) Dentro del contexto de múltiples procesadores, existen dos formas de que estos tengan acceso a una misma fuente de memoria: `UMA` (*Uniform Memory Access*) y `NUMA` (*Non-Uniform Memory Access*). ¿En qué consiste cada una de estas? Mencione, además, una ventaja y desventaja para cada una.

Solución:

- `UMA`: Consiste en la arquitectura de memoria compartida en la que todos los procesadores acceden a la memoria de forma uniforme. Existen varios tipos, pero la estudiada en el curso es la `SMP` (*Symmetric Multiprocessor System*), donde todos los procesadores interactúan con la memoria a partir de un único bus de datos. Su ventaja es que el tiempo de acceso a una sección de memoria es independiente del procesador que trate de acceder a ella, haciéndolo útil en aplicaciones de propósito general y tiempo compartido para múltiples usuarios. Su desventaja, no obstante, es que es costoso de implementar (tamaño del bus de datos considerando la cantidad de procesadores), además de tener una escalabilidad restringida debido a la saturación del bus compartido en el caso de muchos procesadores.
- `NUMA`: Consiste en la arquitectura de memoria compartida en la que esta se distribuye a partir de unidades de memoria locales a las que pueden acceder los distintos procesadores. Su ventaja radica en que es menos costoso de implementar que `UMA`, además de tener una escalabilidad mucho menos restringida al poder extenderse fácilmente (no dependen de un bus de acceso particular para todas las memorias locales disponibles). La desventaja es que los procesadores que busquen acceder a unidades de memoria local remotas tardarán más en poder concretar el acceso, debido a la red interconectada que se debe atravesar para la transferencia de datos.

(e) **[I3-2018-1]** Mencione una ventaja y una desventaja de actualizar las líneas de caché desactualizadas en vez de invalidarlas.

Solución:
Ventaja: Permite que el computador aproveche al máximo la ventaja de tener datos en la caché.
Desventaja: Es muchísimo más complejo implementar un protocolo de coherencia que cubra este caso en lugar de simplemente invalidar.

(f) **[EX-2018-1]** El protocolo `MOESI` corresponde a una variante del protocolo `MESI` estudiado en clases. Este posee el mismo comportamiento, salvo por la existencia de un nuevo estado `O` ("*Owned*"). Este es utilizado para indicar que uno y solo uno de los controladores de caché tiene el permiso para modificar el valor de una línea de la caché asociada a un bloque de la memoria y compartirlo con el resto de las cachés que la posean. Esto permite que, en caso de que se hagan lecturas de un recurso compartido, la transferencia de datos se haga entre cachés y, además, no es necesario que se escriban los cambios directamente en la memoria principal, solo cuando es estrictamente necesario.

I. ¿Cómo se podría asegurar que sea una sola caché la que posea el estado `O` para un recurso compartido? Comente considerando el procedimiento del protocolo `MESI` para recursos compartidos (en particular, para los estados `M`, `S` e `I`).
II. Suponga que un par de cachés comparten una línea con estados `O` y `S`, respectivamente. Si la segunda caché, en estado `S`, solicita realizar cambios en la línea compartida, ¿qué protocolo se puede seguir para mantener la consistencia?

Solución:

I. Cuando un recurso se comparte por primera vez se asigna el estado `E` para indicar que es exclusivo. Luego, cuando otro procesador solicita su lectura, tanto para dicho procesador como para el primero el estado se convierte en `S`, pues está compartido. Posteriormente, si una de las dos caché realiza una modificación, la caché que realiza la modificación cambia la línea a estado `M` y la otra invalida su copia. En el protocolo `MOESI` se puede establecer, en cambio, que la caché que escribe cambie a estado `O` en vez de `M` para indicar que será el encargado predeterminado para compartir el recurso entre las cachés, siempre que no existan solicitudes de escritura que hagan obsoleta su copia, mientras que la segunda mantiene su estado `S`, con la salvedad de que se le transfiere el nuevo valor de la línea. Naturalmente, el resto de los procesadores que soliciten la lectura del recurso para almacenarlo en sus cachés mantendrán un estado `S`, efectuando la transferencia desde la caché con la línea en estado `O`.

II. Suponga que un par de cachés comparten una línea con estados `O` y `S`, respectivamente. Si la segunda caché, en estado `S`, solicita realizar cambios en la línea compartida, ¿qué protocolo se puede seguir para mantener la consistencia? Se puede manejar de dos formas, pero en ambas la caché que solicita la escritura pasa a

estado `O`. En la primera forma, la caché que estaba encargada de la línea compartida puede simplemente invalidar su copia para no causar problemas de inconsistencia. En la segunda, en cambio, simplemente puede cambiar su estado a `S` y recibir a través de una transferencia el nuevo valor.

(g) **[P4d-Sec. 1 | I3-2024-1]** 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*?

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.

Pregunta 2: Evaluación de Jerarquía de Memoria [P1a | I3-2024-1]

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:

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

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

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

$$HR = 0{,}9$$

Pregunta 3: Accesos a Memoria *Caché* [P1c | I3R-2024-1]

Suponga que posee una memoria caché de 16 líneas, 4 palabras por línea y una memoria principal de 128 bytes. Para cada una de las siguientes funciones de correspondencia: *directly mapped*, *fully associative*, 2-*way associative* y 4-*way associative*, indique para ella el *hit-rate* y el estado final de la caché (bits de *tag* y bit de validez) para los siguientes accesos de memoria: 0, 2, 1, 3, 96, 98, 97, 99, 24, 26, 25, 27. Asuma que se implementa una política de reemplazo LRU, si es que la requiere.

Solución: Con las dimensiones entregadas, se tiene:

- **Bits por dirección de memoria:** $\log_2(128) = \log_2(2^7) = 7$
- **Bits de *offset*:** $\log_2(4) = \log_2(2^2) = 2$
- **Bits de índice de línea:** $\log_2(16) = \log_2(2^4) = 4$
- **Bits de índice de conjunto 2W:** $\log_2(\frac{16}{2}) = \log_2(8) = \log_2(2^3) = 3$
- **Bits de índice de conjunto 4W:** $\log_2(\frac{16}{4}) = \log_2(4) = \log_2(2^2) = 2$

De lo anterior se deduce que se tiene: 1 bit de *tag* para la caché *directly mapped*; 5 bits de *tag* para la caché *fully associative*; 2 bits de *tag* para la caché 2-*way associative*; y 3 bits de *tag* para la caché 4-*way associative*. A continuación, se muestra el *hit rate*, secuencia de *hits*/*misses* y estado final para cada caché:

```
# Directly mapped
0M, 2H, 1H, 3H, 96M, 98H, 97H, 99H, 24M, 26H, 25H, 27H
HR=0.75
ÍNDICE LÍNEA | TAG | BIT VALIDEZ
    0000     |  0  |      1
    0001     |  -  |      0
    0010     |  -  |      0
    0011     |  -  |      0
    0100     |  -  |      0
    0101     |  -  |      0
    0110     |  0  |      1
    0111     |  -  |      0
    1000     |  1  |      1
    1001     |  -  |      0
    1010     |  -  |      0
    1011     |  -  |      0
    1100     |  -  |      0
    1101     |  -  |      0
    1110     |  -  |      0
    1111     |  -  |      0
```

```
# Fully associative
0M, 2H, 1H, 3H, 96M, 98H, 97H, 99H, 24M, 26H, 25H, 27H
HR=0.75
ÍNDICE LÍNEA |  TAG  | BIT VALIDEZ
    0000     | 00000 |      1
    0001     | 11000 |      1
    0010     | 00110 |      1
    0011     | ----- |      0
    0100     | ----- |      0
    0101     | ----- |      0
    0110     | ----- |      0
    0111     | ----- |      0
    1000     | ----- |      0
    1001     | ----- |      0
    1010     | ----- |      0
    1011     | ----- |      0
    1100     | ----- |      0
    1101     | ----- |      0
    1110     | ----- |      0
    1111     | ----- |      0
```

```
# 2-way associative
0M, 2H, 1H, 3H, 96M, 98H, 97H, 99H, 24M, 26H, 25H, 27H
HR=0.75
ÍNDICE CONJUNTO | ÍNDICE LÍNEA | TAG | BIT VALIDEZ
      000       |     0000     | 00  |      1
      000       |     0001     | 11  |      1
      001       |     0010     | --  |      0
      001       |     0011     | --  |      0
      010       |     0100     | --  |      0
      010       |     0101     | --  |      0
      011       |     0110     | --  |      0
      011       |     0111     | --  |      0
      100       |     1000     | --  |      0
      100       |     1001     | --  |      0
      101       |     1010     | --  |      0
      101       |     1011     | --  |      0
      110       |     1100     | 00  |      1
      110       |     1101     | --  |      0
      111       |     1110     | --  |      0
      111       |     1111     | --  |      0
```

```
# 4-way associative
0M, 2H, 1H, 3H, 96M, 98H, 97H, 99H, 24M, 26H, 25H, 27H
HR=0.75
ÍNDICE CONJUNTO | ÍNDICE LÍNEA | TAG | BIT VALIDEZ
       00       |     0000     | 000 |      1
       00       |     0001     | 110 |      1
       00       |     0010     | --- |      0
       00       |     0011     | --- |      0
       01       |     0100     | --- |      0
       01       |     0101     | --- |      0
       01       |     0110     | --- |      0
       01       |     0111     | --- |      0
       10       |     1000     | 001 |      1
       10       |     1001     | --- |      0
       10       |     1010     | --- |      0
       10       |     1011     | --- |      0
       11       |     1100     | --- |      0
       11       |     1101     | --- |      0
       11       |     1110     | --- |      0
       11       |     1111     | --- |      0
```

Pregunta 4: *Streaming* y *Caché* [P2c | I3-2015-2]

Las aplicaciones que reproducen audio o video son parte de un tipo de aplicaciones que generan un trabajo en la memoria del tipo *streaming*, que consiste en la lectura de grandes cantidades de datos, pero una muy baja reutilización de estos. Para este ejercicio considere un computador que corre una aplicación que realiza *streaming* para acceder a 512KB de datos. El patrón de acceso a memoria es el siguiente: 0, 4, 8, 12, 16, 20, …

(a) Asuma que el computador tiene una memoria caché de 64KB con función de correspondencia *directly mapped* y líneas de 32 bytes. ¿Cuál es el *miss-rate* para el patrón de accesos a memoria descrito anteriormente? ¿Cuán sensible es este *miss-rate* con respecto al tamaño de la memoria *caché*? ¿Mejora el *hit-rate* si se utiliza una *caché* 4-*way associative*?

Solución: Dado que 32 es un múltiplo de 4, se realizan 8 lecturas por bloque/línea. Dado que el patrón de accesos es estrictamente ascendente, el *miss-rate* en un bloque es igual al de toda la caché, sin importar el tamaño de esta. Por lo tanto, el *miss-rate* es $\frac{1}{32/4} = 12{,}5\,\%$, independiente si la *caché* usa *directly mapped* o 4-*way-associative*.

(b) Recalcule el *miss-rate* tomando los siguientes tamaños de línea en una *caché* que usa *directly mapped*: 16 bytes, 64 bytes y 128 bytes. ¿Qué tipo de localidad aprovecha este tipo de aplicaciones?

Solución: $\frac{1}{16/4} = 25\,\%$, $\frac{1}{64/4} = 6,25\,\%$, $\frac{1}{128/4} = 3,125\,\%$. Dados estos resultados, queda claro que las aplicaciones de *streaming* aprovechan el principio de localidad espacial.

(c) El *prefetching* es una técnica que saca provecho de patrones de lectura predecibles para traer desde la memoria, de manera predictiva, bloques de memoria adicionales cuando una línea de la *caché* en particular es accedida.

Un ejemplo es el *stream buffering*, que consiste en copiar en un *buffer* separado bloques adyacentes a la línea de la *caché* accedida. Si el dato se encuentra en este *buffer*, se considera como *hit* y es copiado a la *caché*, mientras que el espacio que este dato usaba en el *buffer*, es llenado con un nuevo bloque adyacente.

Asuma que el computador cuenta con un *stream buffer* de dos entradas y que la latencia de la *caché* es tal que una línea puede ser cargada antes de que el cálculo realizado sobre la línea anterior ha sido completado. En base a esto y suponiendo una *caché* con líneas de 32 bytes, ¿cuál es el *hit-rate* para el patrón de acceso a memoria del apartado anterior?

Solución: El *hit-rate* será casi igual a 1, ya que sólo hay un *miss* en el primer acceso de memoria. Luego, todas las lecturas encontrarán el dato que necesitan ya sea en la memoria *caché* o en el *stream buffer*. Más específicamente, el *hit-rate* será

$$\frac{(\frac{512\times1024}{4}-1)}{(\frac{512\times1024}{4})} = 99,999\,\%$$

Pregunta 5: Protocolo MESI [P4-Sec. 3 | I3-2024-1]

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

Pregunta 6: Protocolo MESI [P6b | EXR-2023-1]

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		3
0x03		2

```
// CPU0        // CPU1
MOV B,(i)      MOV A,2
MOV A,(arr)    MOV B,arr
SUB B,A        ADD B,A
SHL A,A        MOV B,(B)
MOV (B),A      MOV (B),A
MOV B,A        MOV B,(B)
MOV B,(B)      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]**	**arr[2]**	**i**	**arr[0]**	**arr[1]**	**arr[2]**
1								
2								
3								
4								
5								
6								
7								

Solución: 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]**	**arr[2]**	**A**	**B**	**i**	**arr[0]**	**arr[1]**	**arr[2]**	**A**	**B**	**i**	**arr[0]**	**arr[1]**	**arr[2]**
1	E	I	I	I	-	1	I	I	I	I	2	-	1	1	3	2
2	E	E	I	I	1	1	I	I	I	I	2	1	1	1	3	2
3	E	E	I	I	1	0	I	I	I	I	2	3	1	1	3	2
4	E	E	I	I	2	0	I	I	I	E	2	2	1	1	3	2
5	M	E	I	I	2	0	I	I	M	E	2	2	2	1	2	2
6	M	E	I	I	2	2	I	I	M	E	2	2	2	1	2	2
7	M	E	S	I	2	2	I	I	S	E	2	2	2	1	2	2

Pregunta 7: Protocolo MESI [P4-Sec. 2,4 | I3-2024-1]

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.