ESTRUCTURA DE COMPUTADORES
Segunda evaluación (ejercicios)

APELLIDOS: _____ NOMBRE: _____

[1.4p] 1. Un computador tiene una memoria principal de **1 MiB**. Tiene una caché de **512 Bytes**, asociativa por conjuntos de **2 vías** con algoritmo de reemplazo LRU, con un tamaño de línea de **64 Bytes** y política de **post-escritura** (write-back). Estando la caché inicialmente vacía, se ejecuta el siguiente código. La matriz m se encuentra almacenada por filas a partir de la dirección de memoria 0xA0000, y el vector b a partir de la dirección 0xA0200. El resto de variables no provocan accesos a memoria caché y cada elemento de tipo "int" ocupa exactamente 4 Bytes.

```
int m[4][32], b[4];
for (i = 0; i < 4; i++)
  m[0][i] = m[i][0] + b[i];
```

a) **[0.3p]** Determina las direcciones de memoria a las que accede este código.

b) **[0.2p]** Indica razonadamente cómo se divide una dirección física desde el punto de vista de la caché.

sustituimos la i en el apartado a) y calculamos

c) **[0.4p]** Rellena los campos de la siguiente tabla para los accesos a memoria caché: dirección, etiqueta e índice/conjunto (en **hexadecimal**); si el acceso es un acierto o un fallo; en este último caso, qué tipo de fallo; y si se produce una escritura en memoria principal (MP WR).

	Dirección	Etiqueta	Índice	Acierto/Fallo	Tipo de fallo	MP WR
$m_{0,0}$	0xA0000	A00	0	fallo	forzoso	NO
b_0	0xA0200	A02	0	fallo	forzoso	NO
* $m_{0,0}$	0xA0000	A00	0	acierto	—	NO
$m_{1,0}$	0xA0080	A00	2	fallo	forzoso	NO
b_1	0xA0204	A02	0	acierto	—	NO
* $m_{0,1}$	0xA0004	A00	0	acierto	—	NO
$m_{2,0}$	0xA0100	A01	0	fallo	reemplazo, forzoso	NO
b_2	0xA0208	A02	0	fallo	conflicto	SÍ xq reemplaza a $m_{0,2}$
* $m_{0,2}$	0xA0008	A00	0	fallo	conflicto	NO
$m_{3,0}$	0xA0180	A01	2	fallo	forzoso	NO
b_3	0xA020C	A02	0	acierto	—	NO
* $m_{0,3}$	0xA000C	A00	0	acierto	—	NO

d) **[0.3p]** Indica cómo quedaría el directorio caché en las líneas ocupadas tras la ejecución de este código. BV = Bit de validez, BM = Bit de modificación

Conjunto	Vía	BV	BM	Etiqueta
0	0	1	0	A02
0	1	1	1 * $m_{0,3}$	A00
2	0	1	0	A00
2	1	1	0	A01

e) **[0.2p]** Sabiendo que el tiempo de acierto caché es de 2 ciclos y el tiempo de acceso a memoria principal es de 48 ciclos, calcula el tiempo medio de acceso a memoria.

$THD = t_{acierto} + TF \times PF = 2 + \frac{7}{12} \times 48 =$ **34 ciclos**

$t_{acierto} = 2$ ciclos

$PF = 48$ ciclos

$TF = 7/12 \rightarrow$ tasa fallos = fallos/accesos totales

2. [0.6p] Discute la **veracidad** de las siguientes afirmaciones

a) La política de **ubicar en escritura** es mejor tanto para cachés de post-escritura como de escritura directa.

En las cachés de post-escritura podemos realizar escrituras en líneas de la caché sin necesidad de escribir en la memoria principal hasta que esa línea se expulse de la caché, por lo que reaprovechan si se cumplen los principios de localidad.

En la escritura directa las escrituras se hacen siempre en memoria principal, por lo que el hecho de tener la línea en la caché no supone ningún beneficio en el tiempo empleado en una escritura.

b) En el sistema descrito en el ejercicio 1, incorporando una **caché de segundo nivel** de mayor tamaño se reduciría el tiempo medio de acceso a memoria

Los reemplazos en fallos de conflictos se resuelven en la caché de nivel 2, porque aunque ese bloque haya sido expulsado antes de la caché de nivel 1, es muy probable que siga estando en la de nivel 2 y por lo tanto no sea necesario bajar de nuevo a la memoria principal.

Esto no ocurre con los fallos forzosos porque al ser la primera vez que se acceden no estará en ninguna de las cachés.

c) En el sistema descrito en el ejercicio 1, se podrían reducir los fallos de conflicto **aumentando la asociatividad** de la caché a 4 vías

No es necesario hacer la tabla de nuevo, pensemos:

Si duplicamos la asociatividad, la longitud del índice se reduce en un bit y por lo tanto todos los accesos del ejercicio caerán en el conjunto 0.

Se accede a 5 bloques de memoria diferentes, así que no cabrían todos en el mismo conjunto y por lo tanto habrá remplazo en algún momento.

Sin embargo, la línea que se reemplazará será la A00, índice 2, que no se reutiliza, y por lo tanto no habrá fallos de conflicto (todos los fallos son forzosos).

COMPUTER STRUCTURE
Second test (exercises)

LAST NAME: _____ FIRST NAME: _____

$2^{10} \cdot 2^4 = 2^{14}$ direcciones de 14 bits 2^{10}

[1.4p] 1. A computer has **16 KiB** of main memory and a **2-way** set-associative **1 KiB** cache memory with **16-byte** lines. The cache uses a **FIFO** replacement algorithm. The write policies are **write-back** and **write-allocate**. Starting with an empty cache, we execute the following code. The matrix A is stored by rows starting on address 0x3000, and each float element takes up exactly 4 bytes. Scalar variables are stored in CPU registers, and do not generate memory accesses.

```
float A[8][128];
for (i=0; i<4; i++)
  A[0][i] = -1 * A[i][0];
```

a) **[0.3p]** Calculate the memory addresses accessed by this code.

&A[i][0] = 0x3000 + i x 128 x 4 = 0x3000 + i 0x200 (lectura)

$2^7 \cdot 2^2 = 2^9 = 2 \cdot 00$

&A[0][i] = 0x3000 + (i x 4) (escritura)

i=0 & A[0][0] = 0x3000

i=1 & A[1][0] = 0x3000 + 0x200 = 0x3200

& A[0][1] = 0x3000 + 4 = 0x3004

i=2 & A[2][0] = 0x3000 + 0x400 = 0x3400

& A[0][2] = 0x3000 + 0x8 = 0x3008

i=3 & A[3][0] = 0x3000 + 0x600 = 0x3600

& A[0][3] = 0x3000 + 0xC = 0x300C

b) **[0.2p]** Reason the breakdown of a physical address from the cache point of view.

ETIQUETA	INDICE	DESP
13	5	8 5 2 4 0

→ depende del tamaño de línea $2^{(4)}$ 4 bits

→ depende del nº de conjuntos (TA no hay índice) (AC 2 vías)

nº líneas = $\frac{2^{10}}{2^4} = 2^6$ B

14 - 5 - 4 = 5 bits

nº conj = $\frac{2^6}{2}$ = 32 conjuntos = $2^{(5)}$ conj

c) **[0.4p]** Fill all fields on the following table for memory accesses: address, label and index/set (in **hex** format); whether the access results in a hit or a miss; in the latter case, what type of miss is it; and whether the access triggers a write to Main Memory (MM) or not.

	Address	Label	Index	Hit/Miss	Miss type	MM WR	BM	C/V
$a_{0,0}$	0x3000	11000	00000	F	forzoso	NO	0	COVO
* $a_{0,0}$	0x3000	11000	00000	A	—	NO	1	COVO
$a_{1,0}$	0x3200	11001	00000	F	forzoso	NO	0	COVI
* $a_{0,1}$	0x3004	11000	00000	A	—	NO	1	COVO
reemplazo ← $a_{2,0}$	0x3400	11010	00000	F	forzoso	SÍ	0	COVO
reemplazo * $a_{0,2}$	0x3008	11000	00000	F	conflicto	NO	1	COVI
reemplazo ← $a_{3,0}$	0x3600	11011	00000	F	forzoso	NO	0	COVO
* $a_{0,3}$	0x300C	11000	00000	A	—	NO	1	COVI

ya estuvo en la caché y no es de capacidad xq la caché está casi vacía

reemplazamos algo modificado

d) **[0.3p]** Write down the final state of the cache directory in the used lines after running this code.

Set	Way	Valid bit	Dirty bit	Label
0	0	1	0	11011
0	1	1	1	11000

e) **[0.2p]** The average memory access time during the execution of the code was 46 cycles, and the cache hit time is 1 cycle. Compute the time to access main memory.

$T_{acceso} = T_{acierto} + T_{fallo} \times PF$

$46 ciclos = 1 ciclo + 5/8 \times TMP \qquad TMP = \frac{45 \times 8}{5} = 72 ciclos$

2. **[0.6p]** Discuss the truthfulness of the following statements:

a) In general, the average memory access time improves when increasing the cache associativity because there are fewer capacity misses.

La asociatividad reduce los fallos de conflicto, no de capacidad. Podría reducir el tiempo medio de acceso por eliminar fallos de conflicto, pero no los otros

b) In a multi-level cache, it is possible to observe a low global miss rate while the lower level miss rate is close to 100 %.

En una caché de múltiples niveles es posible observar la tasa de fallos global baja mientras la tasa de fallos de la de último nivel es del 100% V o F

Verdadero, la caché de nivel más bajo en la jerarquía normalmente llegan pocos accesos y la mayoría pueden producir fallos que tienen que ser resueltos con un acceso a la memoria principal.

c) In the system desribed in exercise 1, if the replacement algorithm was LRU instead of FIFO, misses would not trigger a write to Main Memory.

Sí que cambiaría, porque al usar el LRU necesitamos modificar la vía 2 que no tiene BM, entonces no se escribe en memoria y hay menos accesos.

Con LRU ya no se retira la línea modificada, sino la otra y por lo tanto ya no hay que escribir en memoria.