Problema 13. Continuación anticipada, Transferencia en desorden

Un procesador tiene una cache de primer nivel (que llamaremos L1) con bloques de 32 bytes y está conectado a un segundo nivel de jerarquía de memoria (que llamaremos L2) mediante un bus de 8 bytes de ancho. El primer nivel (L1) tiene un tiempo de acceso de 1 ciclo cuando se produce un acierto. Cuando fallamos en el primer nivel accedemos al segundo nivel (L2) para leer un bloque de datos. Suponemos que solo se realizan lecturas y que nunca hay fallos en L2. El siguiente cronograma ilustra un fallo en L1: se necesitan 5 ciclos de latencia y 4 para transferir los datos (T0-T3). Los datos se cargan en L1 mientras se transfieren (car L1), y una vez tenemos todo el bloque en L1, hay que realizar una lectura en L1 (Lect) para enviar el dato a la CPU (DATO), cosa que ocurre dentro del mismo ciclo.

CLK																
Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CPU											DATO					
L1	MISS						car L1	car L1	car L1	car L1	Lect					
L2		Latencia					T0	T1	T2	T3						

Al ejecutar un programa en un simulador, hemos obtenido que, a una frecuencia de 2GHz el programa tardaría 2 segundos en el caso ideal de que no haya fallos de cache, que se han realizado 10^9 accesos a memoria y que el 20% de los accesos provocarían un fallo en L1.

a) **Calculad** el tiempo de ciclo y los ciclos que tarda el programa en el caso ideal.

b) **Calculad** los ciclos de penalización de un fallo de L1 y el tiempo de ejecución del programa teniendo en cuenta la penalización debida a los fallos de L1.

Para mejorar el rendimiento, permitimos que el procesador pueda captar el dato en el mismo ciclo que se transfiere de L2 a L1 (continuación anticipada o *early restart*). Mediante simulación sabemos que cuando se produce un fallo en L1 en nuestro programa, en el 70% de los casos el dato deseado se corresponde al que se transfiere en el ciclo T0,

mientras que hay una probabilidad del 10% de que se transfiera en cada uno de los ciclos restantes (T1-T3). Sabemos además que nunca se produce un fallo mientras se acaba de transferir el resto del bloque.

c) **Completad** el siguiente cronograma en donde se ilustran las acciones a realizar en caso de fallo en L1, suponiendo que tenemos continuación anticipada y que el dato solicitado se corresponde al byte 12 del bloque.

CLK																
Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CPU																
L1	MISS															
L2																

d) **Calculad** el tiempo medio de penalización (en ciclos) de un fallo en L1 y el tiempo de ejecución del programa para el procesador con continuación anticipada.

Para mejorar más el rendimiento hemos modificado también L2 de forma que se transfiera el dato solicitado en el primer ciclo (T0) de la transferencia (transferencia en desorden). Sabemos además que nunca se produce un fallo mientras se acaba de transferir el resto del bloque.

e) **Completad** el siguiente cronograma en donde se ilustran las acciones a realizar en caso de fallo en L1, suponiendo que tenemos transferencia en desorden y que el dato solicitado se corresponde al byte 12 del bloque.

CLK																
Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CPU																
L1	MISS															
L2																

f) **Calculad** el tiempo medio de penalización (en ciclos) de un fallo en L1 y el tiempo de ejecución del programa para el procesador con continuación anticipada.

g) **Calculad** la ganancia (speedup) del sistema con continuación anticipada y del sistema con transferencia en desorden respecto el sistema sin ninguna mejora.

Problema 14.

Prefetch

Disponemos de un procesador de 16 bits con direcciones de 16 bits que tiene una memoria cache de datos con las siguientes características: 3-asociativa, con algoritmo de reemplazo LRU, 12 bloques y 64 bytes por bloque, política de escritura: *copy back + write allocate*

El contenido inicial de la memoria de etiquetas (tags) es el siguiente:

conjunto 0	DB	conjunto 1	DB	conjunto 2	DB	conjunto 3	DB
13	1	13	1	13	0	13	0
43	1	43	1	43	0	43	0
AC	0	AC	0	AC	1	AC	1

El DB=1 indica que el bloque correspondiente ha sido modificado. La información de reemplazo está implícita en la posición. Las posiciones inferiores corresponden a los bloques que llevan más tiempo sin utilizarse. Las posiciones superiores corresponden a los últimos bloques utilizados. Por ejemplo, en el conjunto 3, el bloque con tag 13 es el último utilizado, y el bloque con tag AC el que lleva más tiempo sin ser utilizado.

a) **Rellenad** la siguiente tabla, indicando para cada referencia, el número de bloque de memoria que le corresponde, la etiqueta (TAG), a qué conjunto de MC va a parar, si es acierto o fallo (A/F), el bloque reemplazado cuando proceda, el número de bytes leídos de MP (si se lee de MP) y el número de bytes escritos en MP (si se escribe en MP)

tipo	dirección (hex)	bloque de memoria (hex)	TAG (hex)	conjunto MC	¿acierto o fallo? (A/F)	bloque reemplazado	bytes escritura MP	bytes lectura MP
LECT	B12B							
LECT	B145							
LECT	B1AF							
LECT	B1C4							
ESCR	4387							
LECT	1108							
ESCR	1199							
LECT	11AA							

A la cache anterior le añadimos un ***buffer de prefetch*** de una entrada. En este *buffer* se hace prebúsqueda hardware del bloque **i+1** cuando se accede (tanto en acierto como en fallo) al bloque **i**, siempre que el **i+1** no esté ya en la cache o en el *buffer*. En este último caso, no se realiza *prefetch*.

b) **Rellenad** la siguiente tabla (mismas referencias que la anterior) indicando, para cada referencia, el número de bloque de memoria que le corresponde, la etiqueta (TAG), a qué conjunto de MC va a parar, si se produce acierto o fallo en la cache (A/F), el número de bytes leídos de MP (si se lee de MP), el número de bytes escritos en MP (si se escribe en MP), el bloque de MP que se encuentra en el *buffer* (si procede), si se produce acierto o fallo (A/F) en el *buffer* y el bloque que se prebusca de MP (si procede).

tipo	dirección (hex)	bloque de memoria (hex)	TAG (hex)	conjunto MC	Cache ¿acierto o fallo?	bytes escritura MP	bytes lectura MP	bloque actual buffer	Buffer ¿acierto o fallo?	bloque prefetch buffer
LECT	B12B									
LECT	B145									
LECT	B1AF									
LECT	B1C4									
ESCR	4387									
LECT	1108									
ESCR	1199									
LECT	11AA									

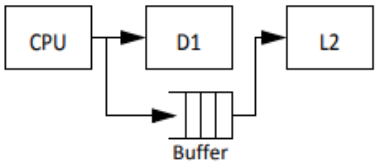
Problema 16. Buffers de escritura

En un procesador que interpreta el lenguaje máquina x86 ejecutamos el siguiente código:

```
        movl $0, %esi
A:      movl %eax, a(,%esi,4) ; store a[i]
B:      movl %eax, b(,%esi,4) ; store b[i]
i:      incl %esi             ; i++
c:      cmpl $N, %esi         ; i<N ?
j:      jl A
```

En este código, que inicializa 2 vectores a un valor, se han etiquetado las instrucciones del bucle. Las instrucciones A y B realizan *stores* en los vectores a y b cuyas direcciones de inicio son 0x10000000 y 0x20000000, respectivamente.

Este procesador tiene una cache de datos de primer nivel (D1) con escritura a inmediata (Write Trough) y sin asignación en caso de fallo en escritura (Write NO Allocate). El tiempo de escribir una palabra en el siguiente nivel de la jerarquía (L2) es de 5 ciclos, por lo que se ha incorporado un buffer de escritura (write buffer), como se muestra en la figura de la derecha, para que el procesador no necesite esperar estos cinco ciclos en cada *store* que realiza. Un **buffer** de escritura funciona como una **cola FIFO**, los datos entran en la cola y van avanzando hasta la cabeza, el dato en la cabeza (en el ejercicio se le llama entrada 0 del buffer o Buffer[0]) es el que se escribe en memoria.



Todas las instrucciones se ejecutan en 1 ciclo (suponemos que nunca tendremos fallos en la cache de instrucciones). Las instrucciones de *store* (A y B en nuestro bucle) también se ejecutan en un ciclo si hay espacio en el buffer para almacenar el dato a escribir en L2. En caso que el buffer este lleno, el procesador se espera los ciclos necesarios, hasta que tenemos espacio en el buffer para ejecutar una instrucción de *store*.

En el **Cronograma 1:** (ver apendice al final de tema 3) se muestra un cronograma de la ejecución de las 2 primeras iteraciones del bucle en donde podemos ver el funcionamiento con un buffer de 1 sola entrada. En este cronograma se ha etiquetado el ciclo en que se ejecuta cada una de las instrucciones con la etiqueta usada en el código anterior. La fila "# Buffer" indica el numero de entradas ocupadas del buffer y la fila "Buffer[0]" indica el contenido de la entrada 0 del buffer. En este caso solo hay una entrada, en los cronogramas 2 y 3 se usan buffers con 2 y 3 entradas respectivamente. En nuestros cronogramas, dado que no hay lecturas de memoria que puedan interferir con las escrituras del buffer, la entrada Buffer[0] siempre se corresponde con los datos que se están escribiendo en L2 (ocupación bus).

En el ciclo 01 se ha ejecutado el store A, por lo que la entrada Buffer[0] estará ocupada por el dato a[0] durante los siguientes 5 ciclos. En el ciclo 02 el procesador intentaría ejecutar el store B, pero debe esperarse al ciclo 07 a que el buffer tenga entradas disponibles (Buffer[0] estará ocupado con el dato b[0] durante los siguientes 5 ciclos). En los ciclos 08, 09 y 10 se ejecutan las siguientes 3 instrucciones (que no realizan accesos a datos). En el ciclo 11 se inicia la iteración 1, pero dado que el buffer está ocupado, el procesador no puede ejecutar la instrucción A hasta el ciclo 13 (en donde escribe el dato a[1] en el buffer). Igualmente, B debe esperarse al ciclo 19 (en que escribe el dato b[1]).

a) **Completa** el **Cronograma 1:** hasta el ciclo 44.
Calcula el CPI para N=1.000.000 iteraciones.
Calcula el ancho de banda (en bytes por ciclo) entre el buffer y L2 para N=1.000.000 iteraciones.

En el **Cronograma 2:** suponemos que el buffer dispone de 2 entradas (nótese que ahora es posible escribir en el buffer siempre que haya menos de 2 entradas ocupadas).

b) **Rellena** el **Cronograma 2:** hasta el ciclo 44.
Calcula el CPI para N=1.000.000 iteraciones.
Calcula el ancho de banda (en bytes por ciclo) entre el buffer y L2 para N=1.000.000 iteraciones.

En el **Cronograma 3:** suponemos que el buffer dispone de 3 entradas (nótese que ahora es posible escribir en el buffer siempre que haya menos de 3 entradas ocupadas).

c) **Rellena** el **Cronograma 3:** hasta el ciclo 44.
Calcula el CPI para N=1.000.000 iteraciones.
Calcula el ancho de banda (en bytes por ciclo) entre el buffer y L2 para N=1.000.000 iteraciones.

Como se puede observar, el tener 3 entradas en el buffer no representa una mejora sustancial para este código.

d) **Razona** ¿A que crees qué es debido?

Supongamos ahora que disponemos de un Merge Buffer de 3 entradas, en donde cada entrada del buffer puede almacenar un bloque de datos de 8 bytes debidamente alineados. En nuestro ejemplo se correspondería a una pareja de palabras consecutivas debidamente alineadas, es decir una entrada podría almacenar las parejas de datos a[0:1] o b[8:9] pero no las a[1:2] o b[3:4] dado que no se corresponden a un bloque de 8 bytes alineado.

Con el Merge Buffer, cuando se ejecuta una instrucción de *store*, primero se comprueba si el dato se puede combinar con una entrada existente, y si no es así se usa una nueva entrada. Sólo en caso que se necesite una nueva entrada y el buffer esté lleno será necesario bloquear el procesador, pero no si la nueva escritura se puede combinar con una entrada existente. Si la entrada 0 del buffer ya ha iniciado su escritura en L2, entonces no será posible la combinación con futuras escritura. Nótese que en nuestros cronogramas, dado que no hay lecturas de memoria que puedan interferir con las escrituras del buffer, la entrada Buffer[0] siempre se corresponde con datos que se están escribiendo en L2. Escribir un bloque de 8 bytes sigue tardando 5 ciclos. Para indicar que una entrada del buffer tiene una pareja de datos lo podemos escribir como: a[0:1].

e) **Rellena** el **Cronograma 4:** hasta el ciclo 44 en donde suponemos que tenemos un merge buffer de 3 entradas y 2 palabras (8 bytes) por entrada.
Calcula el CPI para N=1.000.000 iteraciones.
Calcula el ancho de banda (en bytes por ciclo) entre el buffer y L2 para N=1.000.000 iteraciones.

Cronograma 1: Buffer de 1 entrada.

Iteración	<----------Iteración 0 ---------->										<----------Iteración 1 ---------->																																		
Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
movl %eax, a(,%esi,4)	A										-	-	A																																	
movl %eax, b(,%esi,4)		-	-	-	-	-	B							-	-	-	-	-	B																											
incl %esi								i												i																										
cmpl $N, %esi									c												c																									
jl A										j												j																								
Ocupación bus		a[0]						b[0]						a[1]						b[1]																											
# Buffer	0	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1																						
Buffer[0]		a[0]						b[0]						a[1]						b[1]																											

CPI = ... Ancho de banda = ...

Cronograma 2: Buffer de 2 entradas

Iteración																																													
Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
movl %eax, a(,%esi,4)																																													
movl %eax, b(,%esi,4)																																													
incl %esi																																													
cmpl $N, %esi																																													
jl A																																													
Ocupación bus																																													
# Buffer																																													
Buffer[0]																																													
Buffer[1]																																													

CPI = ... Ancho de banda = ...

Cronograma 3: Buffer de 3 entradas

Iteración																																													
Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
movl %eax, a(,%esi,4)																																													
movl %eax, b(,%esi,4)																																													
incl %esi																																													
cmpl $N, %esi																																													
jl A																																													
Ocupación bus																																													
# Buffer																																													
Buffer[0]																																													
Buffer[1]																																													
Buffer[2]																																													

CPI = ... Ancho de banda = ...

Cronograma 4: Merge buffer de 3 entradas

Iteración																																													
Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
movl %eax, a(,%esi,4)																																													
movl %eax, b(,%esi,4)																																													
incl %esi																																													
cmpl $N, %esi																																													
jl A																																													
Ocupación bus																																													
# Buffer																																													
Buffer[0]																																													
Buffer[1]																																													
Buffer[2]																																													

CPI = ... Ancho de banda = ...