

Universidad de Murcia
Facultad de Informática

TÍTULO DE GRADO EN
INGENIERÍA INFORMÁTICA

Estructura y Tecnología de Computadores

Tema 5: Jerarquía de memoria — Caché

Apuntes

CURSO 2010 / 11

VERSIÓN 2.0

Departamento de Ingeniería y Tecnología de Computadores

Área de Arquitectura y Tecnología de Computadores



Índice general

5.1. Introducción	1
5.1.1. Principio de localidad	2
5.1.2. Conceptos generales	3
5.2. Memoria caché	5
5.2.1. Memoria caché de correspondencia directa	5
5.2.2. Rendimiento de la caché	12
5.2.3. Memoria caché asociativa por conjuntos	14
5.3. Tratamientos de los fallos de caché	18
5.3.1. Tratamiento de los fallos de lectura	18
5.3.2. Tratamiento de los fallos de escritura	19
5.4. Memorias caché multinivel	20
5.5. Características de la memoria caché en algunos sistemas actuales	20
A5. Apéndices	23
A5.1. Diseño del sistema de memoria para soportar cachés	23
B5. Boletines de prácticas	25
B5.1. Simulación de cachés	25
B5.1.1. Objetivos	25
B5.1.2. Prerequisitos	25
B5.1.3. Plan de trabajo	25
B5.1.4. El simulador de cachés de MARS	26
B5.1.5. Ejercicios	27
B5.2. Organización de programas	28
B5.2.1. Objetivos	28
B5.2.2. Prerequisitos	28
B5.2.3. Plan de trabajo	28
B5.2.4. Acceso a datos y rendimiento de la caché	29
B5.2.5. Ejercicios	29
E5. Ejercicios	32
E5.1. Jerarquía de memoria — Caché	32
E5.2. Solución a ejercicios seleccionados	37

5.1. Introducción

Desde los primeros días de la informática, los programadores han querido contar con una memoria rápida e infinita para que sus programas se ejecuten más rápidamente y puedan manejar grandes cantidades de código y datos de una forma eficiente. El problema es que hoy en día no existe un único tipo de memoria que sea a la vez rápida y grande a un coste razonable. Por ello, como veremos a lo largo de este tema, los sistemas actuales combinan distintos tipos de memoria, con distintas velocidades y tamaños, para *crear la ilusión* de que realmente existe una memoria con las características deseadas. Esta combinación dará lugar a una *jerarquía de memoria*.

La forma en la que se combinan distintos tipos de memoria se basa en ciertos principios y mecanismos que es importante entender. Para ello, los vamos a ilustrar de una forma sencilla mediante la siguiente analogía¹.

Un estudiante está sentado en una mesa en la biblioteca realizando un trabajo sobre un tema específico. En la mesa tiene algunos libros que ha cogido de las estanterías que versan sobre el tema del trabajo. Si algún elemento de información necesario para el trabajo no se encuentra en los libros que tiene sobre la mesa, coge uno de los libros de la mesa que cree que no va a necesitar más, va a la estantería, coge un nuevo libro y deja el que ya no necesita. Una vez que el estudiante ha hecho una buena selección de libros, hay una gran probabilidad de que gran parte de la información que necesita para el trabajo se encuentre en alguno de los libros que tiene sobre su mesa, por lo que no tendrá que perder más tiempo en acceder de nuevo a las estanterías.

El mismo principio que acabamos de ilustrar permite crear la ilusión de una memoria grande (biblioteca) a la que se pueda acceder a la misma velocidad que a una pequeña (mesa de trabajo). De la misma manera que el estudiante no necesita acceder a todos los libros de la biblioteca con la misma probabilidad, un programa no accede a todo su código o datos con la misma probabilidad. Por tanto, un primer objetivo será detectar las zonas de código y datos de un programa que tienen más probabilidad de ser accedidos y así intentar tenerlos disponibles en la memoria más rápida.

5.1.1. Principio de localidad

El principio de localidad explica tanto el trabajo del estudiante en la biblioteca como la manera de funcionar de los programas. El principio de localidad afirma que **en un momento concreto, los programas acceden a una parte relativamente pequeña de su espacio de direcciones**, de la misma manera que el estudiante sólo accede a una pequeña parte de los libros de la biblioteca. Existen dos tipos de localidad:

- **Localidad temporal:** si se consulta un dato, seguramente será consultado próximamente. En el ejemplo anterior, si el estudiante lleva un libro a su mesa y lo utiliza, seguramente pronto volverá a utilizarlo otra vez.
- **Localidad espacial:** si se consulta un dato, seguramente otros datos cercanos a él serán consultados próximamente. Si en el ejemplo, el estudiante está buscando información sobre “Álgebra de Boole y Sistemas Digitales” y ha ido a la estantería y coge un libro concreto de “Circuitos Combinacionales”, seguramente los libros que están próximos en el estante versarán sobre el mismo tema y probablemente los consultará también más tarde; incluso puede que le interese coger alguno de estos libros para usarlo posteriormente sin tener que volver a levantarse de su mesa. Los libros con la misma temática están en la misma estantería de la biblioteca para incrementar la localidad espacial.

Tal como los accesos a los libros de una misma temática muestran localidad, la localidad de los programas surge de su estructura de una forma simple y natural:

- **Localidad temporal en un programa:** en los programas aparecen multitud de bucles, por lo que se accederá repetidamente a instrucciones y datos, mostrando mucha localidad temporal. Por ejemplo, en el código:

```
for i:=1 to 1000 do
begin
  a := a + 2;
  b := b * 3;
  c := c div 2;
end
```

¹Algunos de los conceptos que vamos a ver ya se introdujeron en la asignatura “Fundamentos de Computadores”. Aquí se incluyen por completitud y como repaso de lo ya visto.

Tabla 1: Tiempos de acceso y precios de las distintas tecnologías de memoria más comunes (año 2009).

Tecnología	Tiempo de acceso típico	€ por MB
SRAM	1 ns	20 €
SDRAM	5 ns	0,01 €
Disco magnético	8.500.000 ns	0,0001 €

las 3 instrucciones del interior del bucle son ejecutadas 1000 veces sucesivamente. Las variables a, b y c son referenciadas 1000 veces sucesivamente.

- Localidad espacial en un programa:** como normalmente se accede a las instrucciones secuencialmente, los programas muestran mucha localidad espacial en el acceso a su código. Los accesos a los datos también muestran mucha localidad espacial natural, como son, por ejemplo, los accesos a los elementos de una tabla (array). Así, en el código:

```
for i := 1 to 1000 do
begin
  a[i] := 1;
  b[i] := 2;
end
```

a los elementos de los arrays a y b se accede siguiendo su distribución espacial (secuencialmente).

5.1.2. Conceptos generales

El objetivo de combinar los distintos tipos de memoria de un ordenador en una jerarquía de memorias es aprovechar el principio de localidad. Una **jerarquía de memoria** consiste en múltiples niveles de memoria con diferentes velocidades y capacidades, donde las memorias más rápidas suelen ser las más caras por bit y, por tanto, las más pequeñas. Hoy en día, son tres las tecnologías que se utilizan habitualmente para construir la jerarquía de memoria. La memoria principal se construye con SDRAM (*Synchronous Dynamic RAM*). Los niveles más cercanos a la CPU, como la memoria caché, se construyen con tecnología SRAM. La SDRAM es más barata por bit que la SRAM pero es más lenta, como ya se vio en el tema 2. Por último, la tecnología para construir el nivel más lento y barato por bit es el disco magnético. En la tabla 1 tenemos valores de tiempo de acceso y precios de las tres tecnologías en el año 2009.

Debido a estas diferencias de coste y tiempo de acceso, es ventajoso construir memorias como una jerarquía de niveles, con las memorias más rápidas cerca del procesador y las más lentas lejos de él (figura 1).

El objetivo general que se persigue es **proporcionar al usuario la máxima capacidad de memoria que se pueda conseguir usando la tecnología más barata, pero con un tiempo de acceso similar al de la memoria más rápida**. Esto se puede lograr debido a que, como ya hemos comentado anteriormente, los programas muestran tanto localidad temporal (tendencia a volver a utilizar en breve datos a los que ya se ha accedido) como localidad espacial (tendencia a referenciar datos que están cerca de otros recientemente accedidos). Las jerarquías de memoria sacan provecho de la localidad temporal manteniendo los datos accedidos más recientemente cerca del procesador (en los niveles de memoria más rápidos). También sacan provecho de la localidad espacial moviendo bloques de datos contiguos a los niveles de la jerarquía más cercanos al procesador.

Una jerarquía de memoria suele consistir en múltiples niveles, pero los datos sólo se transfieren entre dos niveles adyacentes, por lo que podemos realizar el estudio centrándonos sólo en dos niveles. Llamaremos nivel superior al nivel más cercano al procesador (que será pequeño y rápido), y nivel inferior al más lejano (que

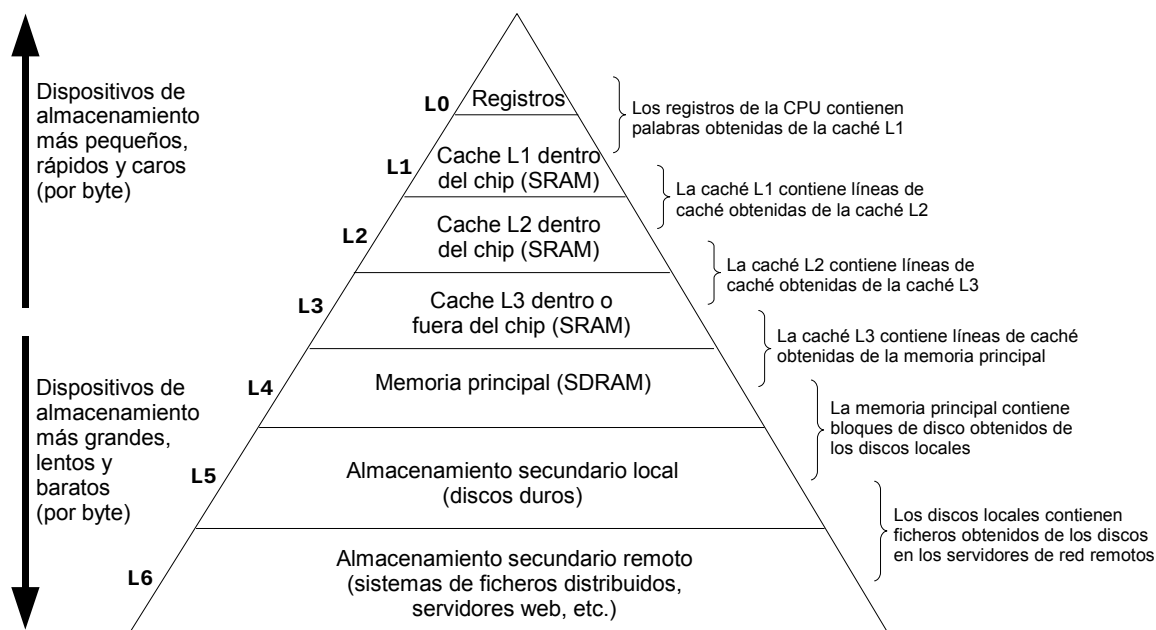


Figura 1: Estructura básica de la jerarquía de memoria.

será grande y lento). A continuación, se definen algunos conceptos básicos que vamos a utilizar durante este estudio:

Bloque: unidad mínima de información con la que se trabaja en el sistema de jerarquía de memoria. En el ejemplo de la biblioteca, un bloque de información sería un libro.

Acierto (*hit*): cuando los datos que pide el procesador aparecen en algún bloque del nivel superior. Esto es análogo a encontrar la información deseada en alguno de los libros de encima de la mesa.

Fallo (*miss*): cuando los datos pedidos por el procesador no se encuentran en el nivel superior, teniendo que acceder al nivel inferior para leer el bloque que contiene los datos deseados. En el ejemplo de la biblioteca, es cuando el estudiante no encuentra lo que necesita en los libros de la mesa y se tiene que levantar e ir a las estanterías a buscar un nuevo libro.

Tasa de aciertos (*hit rate*): es el porcentaje de accesos a memoria encontrados en el nivel superior. Este dato se usa como medida del rendimiento de una jerarquía de memoria.

Tasa de fallos (*miss rate*): es igual a $(1 - \text{tasa de aciertos})$ y nos informa del porcentaje de accesos a memoria en los que los datos no están en el nivel superior.

Tiempo de acierto: es el tiempo necesario para acceder al nivel superior de la memoria, incluyendo el tiempo para determinar si el acceso es un acierto o un fallo. En nuestro ejemplo de la biblioteca, sería el tiempo necesario para ojear los libros de encima de la mesa para saber si tenemos la información que buscamos en alguno de ellos o bien tenemos que levantarnos a por otro.

Penalización por fallo: es el tiempo necesario para reemplazar un bloque del nivel superior por otro bloque del nivel inferior. En el ejemplo sería el tiempo de levantarnos de la mesa, coger otro libro de las estanterías, dejarlo encima de la mesa y volvernos a sentar.

Como el nivel superior es más pequeño y está construido con componentes más rápidos, el tiempo de acierto será mucho menor que la penalización por fallo.

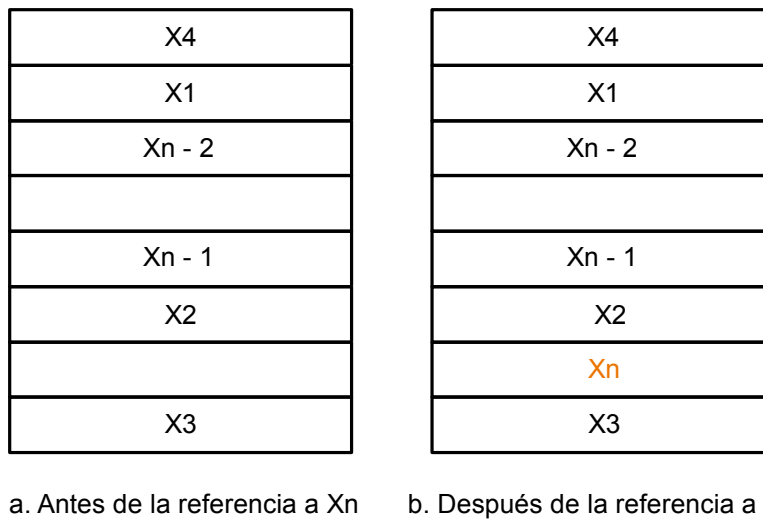


Figura 2: Una memoria caché antes y después de la referencia a la palabra X_n .

Todos los programas pasan mucho tiempo accediendo a memoria ya que, para ejecutar cada instrucción del programa, lo primero que se hace es llevar la codificación de esa instrucción desde memoria al procesador. Además, si la instrucción lleva algún operando que sea un dato de memoria, habrá otro acceso a memoria para leer/escribir ese dato. Por esta razón, el sistema de memoria es un factor de gran importancia para determinar el rendimiento completo de un computador.

Por otro lado, también es importante resaltar que las decisiones tomadas a la hora de diseñar los sistemas de memoria afectan a otros aspectos del ordenador, incluyendo cómo gestiona el sistema operativo la memoria y la E/S, cómo generan código los compiladores, etc.

5.2. Memoria caché

En el ejemplo de la biblioteca, la mesa del estudiante actuaba de caché: un espacio pequeño donde guardar cosas (libros) que se necesitan examinar continuamente. **Caché** fue el término escogido en un principio para representar el nivel de la jerarquía de memoria entre la CPU y la memoria principal. Las memorias cachés aparecieron primero en máquinas de investigación a principios de los años sesenta y más tarde, a finales de la década, en máquinas de propósito general. En la actualidad casi todas las máquinas de propósito general tienen algún tipo de memoria caché.

Supongamos una caché muy simple, un procesador cuyas peticiones son de una palabra y bloques con un tamaño también de una palabra. En la figura 2 se muestra esta caché antes y después de la petición de un dato que no está en la caché. Antes de la petición, la caché contiene un conjunto de palabras $\{X_1, X_2, \dots, X_{n-1}\}$ que han sido referenciadas por el procesador recientemente. Si el procesador pide ahora la palabra X_n , se produce un *fallo de caché*, con lo que se busca esa palabra en la memoria principal, se trae a la caché y se coloca en uno de los huecos existentes.

Viendo la situación planteada en la figura 2, surgen dos preguntas: ¿cómo se sabe si un dato está en la caché? Y si está, ¿cómo se localiza?. Las respuestas a estas dos preguntas están relacionadas tal como se verá en la siguiente subsección, donde se describe la estructura y funcionamiento de un primer modelo de memoria caché.

5.2.1. Memoria caché de correspondencia directa

En una memoria caché de correspondencia directa cada dirección de memoria se corresponde con una sola entrada de la caché. Es decir, si el procesador referencia la palabra de dirección X , para saber si esa palabra se

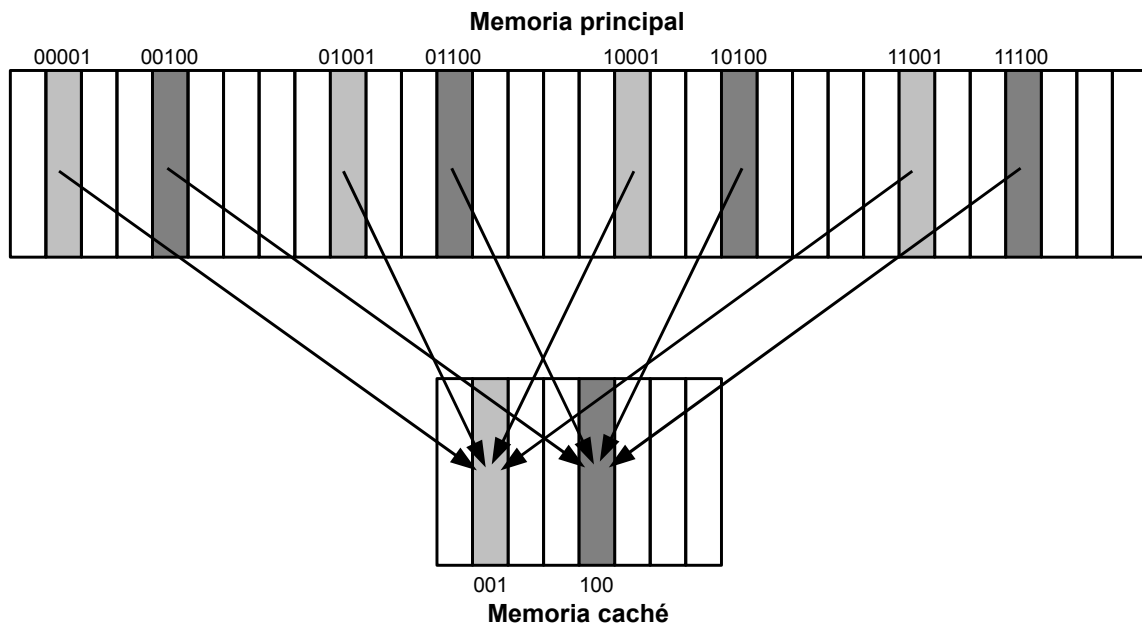


Figura 3: Una memoria caché de 8 entradas y una memoria principal de 32 palabras. Correspondencias de direcciones de memoria principal con posiciones de caché usando bloques monopalabra.

encuentra en caché únicamente será necesario mirar en la posición **Y** de la caché, existiendo normalmente una relación muy simple entre **X** e **Y**. Esta relación en casi todas las memorias caché de correspondencia directa sería:

$$\text{posición en caché} = (\text{dirección de bloque}) \text{ MOD } (\text{número posiciones caché})$$

siendo “MOD” el resto de la división entera.

Bloques monopalabra

Si partimos de un caso simple donde los bloques contienen una sola palabra, la fórmula anterior quedaría:

$$\text{posición en caché} = (\text{dirección de palabra}) \text{ MOD } (\text{número posiciones caché})$$

Esta técnica es interesante ya que, si el número de posiciones de la caché es una potencia de dos, el módulo se puede calcular simplemente tomando los $\log_2(\text{número de posiciones de la caché})$ bits de menor peso de la dirección y, en consecuencia, se puede acceder a la caché usando estos bits de menor peso. Por ejemplo, en la figura 3 se muestra una caché de correspondencia directa con 8 posiciones para 8 bloques monopalabra y una memoria principal de 32 palabras. Como hay 8 posiciones en caché, si una palabra de memoria principal de dirección **X** debe ser llevada a caché, se le asignará la posición $X \text{ MOD } 8$. Por lo tanto, a las palabras de las direcciones 1, 9, 17 y 25 les corresponde la posición de caché 1, ya que $(1 \text{ MOD } 8)$, $(9 \text{ MOD } 8)$, $(17 \text{ MOD } 8)$ y $(25 \text{ MOD } 8)$ valen 1. O visto de otra manera, los 3 bits de menor peso servirán de índice de la caché (ya que $\log_2 8 = 3$), es decir, a las direcciones 00001_2 , 01001_2 , 10001_2 y 11001_2 les corresponderá la posición de caché 001_2 .

Usando esta técnica descrita, si el procesador solicita, por ejemplo, la palabra de dirección 9 habrá que ir a buscarla a la posición de caché 1. Pero entonces, surge otra pregunta: puesto que cada posición de la caché puede contener los datos de unas cuantas direcciones de memoria, ¿cómo se sabe si una palabra solicitada por el procesador es la que está en caché en esa posición o es otra (la 1, la 17 o la 25) la que ocupa su puesto? Este problema se puede resolver añadiendo un conjunto de etiquetas a la caché. Estas **etiquetas** (*tags*) contendrán

Tabla 2: Cálculo de la posición en caché y la etiqueta para cuatro direcciones de memoria.

Dirección de palabra	Posición en caché	Etiqueta en caché
1 = 00 001) ₂	001	00
9 = 01 001) ₂	001	01
17 = 10 001) ₂	001	10
25 = 11 001) ₂	001	11

la información necesaria para distinguir cuál de las posibles palabras que pueden ocupar una posición de caché es la que la ocupa en un momento dado.

La etiqueta más simple está formada por aquellos bits de la dirección en los que difieren las diferentes palabras que pueden ocupar una posición. Así, si en el ejemplo anterior decíamos que a las palabras de dirección 00001)₂, 01001)₂, 10001)₂ y 11001)₂ les corresponderá la posición de caché 001)₂ tomando el valor de sus 3 bits menos significativos, si usamos como etiqueta los restantes 2 bits más significativos podremos distinguir cuál de ellas ocupa en cada momento la posición 001)₂ de la caché. La tabla 2 muestra la posición de caché y la etiqueta asociada a cada una de las cuatro direcciones anteriores.

También se necesita una forma de reconocer que un bloque de caché no tiene información válida. Por ejemplo, cuando un procesador se inicia, la caché estará vacía y las etiquetas no tendrán significado real, como ocurría en la figura 2. En consecuencia, se necesita saber que la etiqueta tiene que ser ignorada en estos casos. El método más común es añadir un *bit de validez* que indica si una posición de caché contiene un bloque con datos válidos.

En la tabla 3 se muestra un ejemplo de la ejecución de un programa que genera 8 peticiones de palabras de memoria y el efecto que ello tendría usando una memoria caché de 8 posiciones como en el ejemplo anterior, pero a la que se ha añadido un bit de validez y una etiqueta para cada posición. En la tabla 4 se muestra en detalle la evolución del contenido de la misma caché. Como se puede ver en la última referencia, cuando la palabra de dirección 10010)₂ es traída a la posición de caché 010)₂, la información que anteriormente estaba en esta posición es reemplazada por la nueva información pedida. Este comportamiento permite que la caché se aproveche de la localidad temporal: palabras accedidas recientemente reemplazan palabras referenciadas menos recientemente. Esta situación es análoga a la del estudiante que necesita un libro de las estanterías y no tiene más espacio sobre la mesa: algún libro de la mesa tendrá que devolverse a las estanterías.

Como se puede ver, se sabe dónde buscar en la caché para cada posible dirección: la parte baja de la dirección se usa para encontrar la posición de la caché que se le asigna a esa dirección, mientras que la parte alta de la dirección se compara con la etiqueta de la caché para determinar si lo que hay en la posición de la caché corresponde a la dirección solicitada. Si, además, el bit de validez indica con un valor de 1 que la información es válida, se tiene un *acierto de caché* y la palabra se suministra al procesador. En caso contrario es un *fallo de caché*. La figura 4 muestra cómo se divide una dirección referenciada en una máquina con palabras de 32 bits y equipada con una caché monopalabra, de correspondencia directa, y 1024 entradas (es decir, una caché para 4 KBytes de datos).

- Un índice para la caché, que se usa para seleccionar la posición. Como la caché tiene 1024 (2^{10}) posiciones, este índice lo formarán los 10 bits menos significativos de la dirección de palabra.
- Una etiqueta, que se compara con el valor de la etiqueta de la posición de la caché seleccionada por el índice. La etiqueta estará formada por los restantes bits más significativos no usados para el índice y que, por tanto, sirven para distinguir entre las distintas direcciones de memoria a las que les corresponde la misma posición de caché.
- Los 2 bits menos significativos se desprecian pues únicamente indican el desplazamiento del byte dentro de la palabra buscada (palabras de 32 bits).

Tabla 3: Secuencia de ocho peticiones de acceso a memoria.

Peticiones	Dir) ₁₀	Dir) ₂	Posición caché	Etiqueta caché	Acierto o Fallo
1º	22	10110	110	10	Fallo
2º	26	11010	010	11	Fallo
3º	22	10110	110	10	Acierto
4º	26	11010	010	11	Acierto
5º	16	10000	000	10	Fallo
6º	4	00100	100	00	Fallo
7º	16	10000	000	10	Acierto
8º	18	10010	010	10	Fallo

El número total de bits necesarios para una caché es función del número de posiciones de la caché, del tamaño de las direcciones de memoria, del número de palabras por bloque y del número de bits por palabra:

$$N^{\circ} \text{ total bits} = n^{\circ} \text{ posiciones} \times (n^{\circ} \text{ bits control} + \text{tamaño etiqueta} + \text{tamaño bloque})$$

donde:

- Tamaño bloque = (nº de palabras por bloque) × (nº de bits por palabra)
- Tamaño etiqueta = (tamaño dirección) – (nº bits de índice) – (nº bits del desplazamiento del byte)
- Nº bits del índice = \log_2 (nº de posiciones)
- Nº bits del desplazamiento del byte (o *byte offset*) = \log_2 (nº de bytes por palabra)

Si, como en el ejemplo anterior, tenemos una caché de 2^{10} posiciones, cada una con su bit de validez, direcciones de memoria de 32 bits (donde los 2 bits menos significativos sirven para apuntar a un byte dentro de la palabra), bloques monopalabra y palabras de 32 bits, entonces:

- Nº bits para el desplazamiento del byte = $\log_2(4) = 2$ bits
- Nº bits del índice = $\log_2(2^{10}) = 10$ bits
- Tamaño etiqueta = $(32 - 10 - 2) = 20$ bits
- Tamaño bloque de datos = 1 palabra × 32 bits/palabra = 32 bits

por lo tanto:

$$N^{\circ} \text{ total de bits} = 2^{10} \times (1 + 20 + 32) = 54272 \text{ bits} = 6784 \text{ bytes}$$

Bloques multipalabra

Con el diseño de la caché descrita hasta ahora no se hace nada para sacar partido de la localidad espacial, ya que cada palabra está en su propio bloque. Para aprovechar esta localidad (que, como se vio en la sección 5.1.1, se da de forma natural en los programas) es necesario manejar bloques de más de una palabra de manera que, cuando se produzca un fallo, se traiga a caché no sólo la palabra buscada sino un grupo de palabras de direcciones adyacentes a ella y que, por tanto, tendrán una alta probabilidad de ser referenciadas en breve.

En la figura 5 se muestra una caché de 64 KBytes de datos pero con bloques de 4 palabras (16 bytes). Comparada con la figura 4, aparece un campo *desplazamiento de la palabra* (*word offset*) dentro del bloque

Tabla 4: Detalles de los contenidos de una caché durante ocho peticiones de palabras después de cada petición que provoca fallo de caché.

Índice	V	Etiqueta	Dato	Índice	V	Etiqueta	Dato
000	N			000	N		
001	N			001	N		
010	N			010	N		
011	N			011	N		
100	N			100	N		
101	N			101	N		
110	N			110	S	10	M(10110)
111	N			111	N		
a) Estado inicial de la caché después de arrancar				b) Después de tratar el fallo de la dirección de memoria (10110)			
Índice	V	Etiqueta	Dato	Índice	V	Etiqueta	Dato
000	N			000	S	10	M(10000)
001	N			001	N		
010	S	11	M(11010)	010	S	11	M(11010)
011	N			011	N		
100	N			100	N		
101	N			101	N		
110	S	10	M(10110)	110	S	10	M(10110)
111	N			111	N		
c) Después de tratar el fallo de la dirección de memoria (11010)				d) Después de tratar el fallo de la dirección de memoria (10000)			
Índice	V	Etiqueta	Dato	Índice	V	Etiqueta	Dato
000	S	10	M(10000)	000	S	10	M(10000)
001	N			001	N		
010	S	11	M(11010)	010	S	10	M(10010)
011	N			011	N		
100	S	00	M(00100)	100	S	00	M(00100)
101	N			101	N		
110	S	10	M(10110)	110	S	10	M(10110)
111	N			111	N		
e) Después de tratar el fallo de la dirección de memoria (00100)				f) Después de tratar el fallo de la dirección de memoria (10010)			

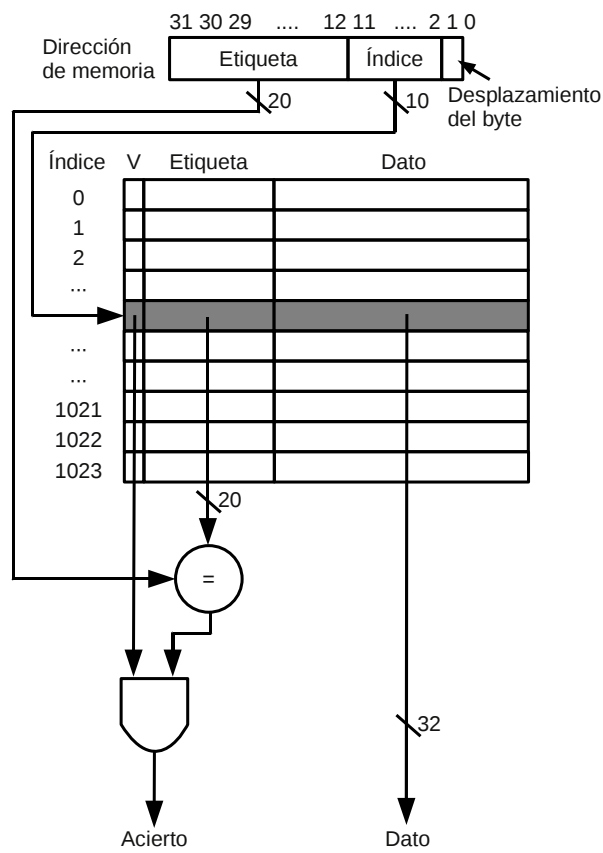


Figura 4: División de una dirección de memoria para el correcto acceso a la caché.

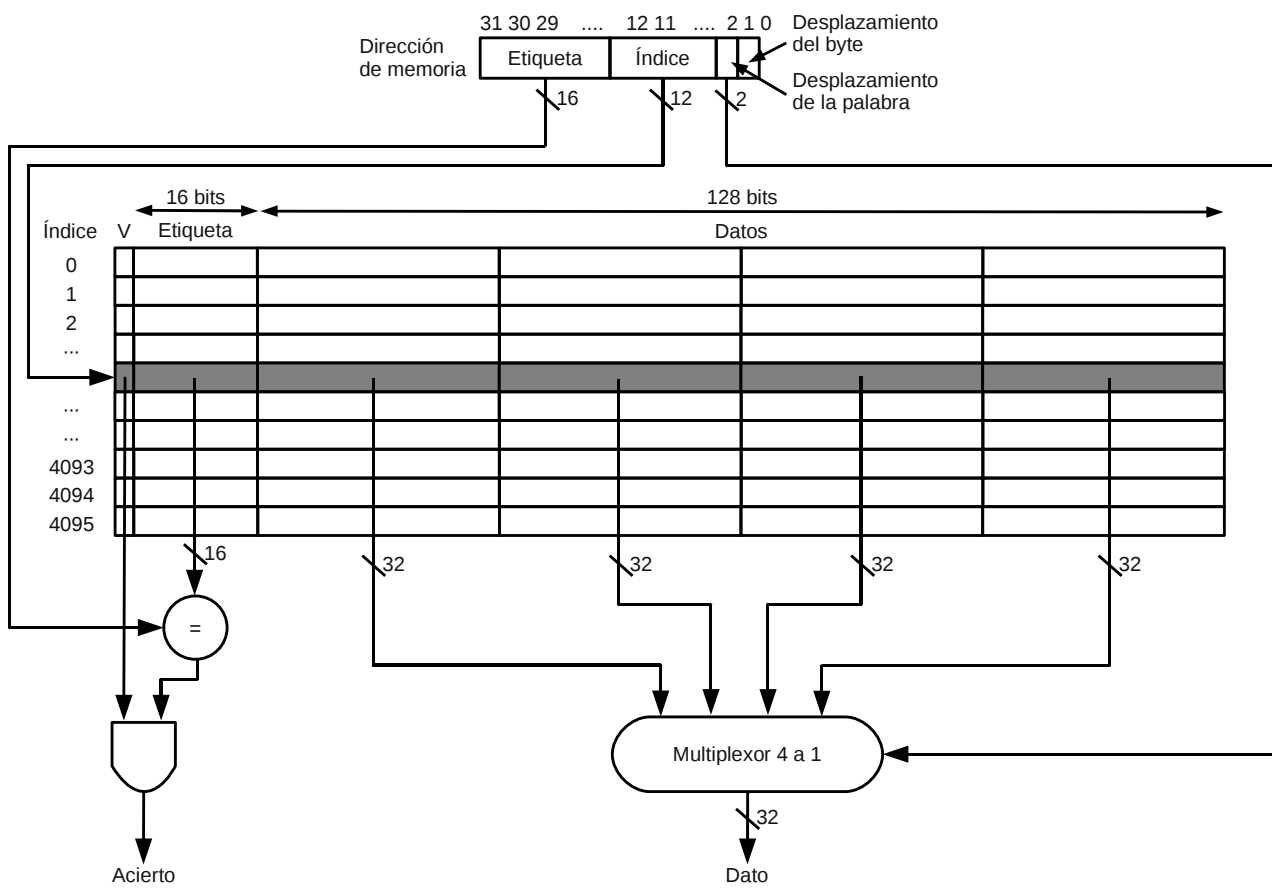


Figura 5: Memoria caché de 64 KB con bloques de cuatro palabras.

en la dirección de memoria. Este campo se usa para controlar el multiplexor (mostrado en la parte inferior de la figura) que selecciona la palabra deseada entre las cuatro que forman cada bloque.

Como se puede observar, el número total de etiquetas y bits de validez en la caché con bloques multipalabra es menor porque cada etiqueta y bit de validez se usa para cada bloque (cuatro palabras) y no para cada palabra.

En una caché de bloques multipalabra, para determinar la posición de caché que corresponde a una dirección particular, se usa el mismo esquema que en las de bloques monopalabra:

$$\text{posición en caché} = (\text{dirección de bloque}) \text{ MOD } (\text{número posiciones caché})$$

$$\text{dirección de bloque} = (\text{dirección de palabra}) \text{ DIV } (\text{número palabras por bloque})$$

Con lo que, volviendo al ejemplo de la figura 5, la dirección de un dato solicitado por el procesador se descompone en: los 2 bits menos significativos, que indican el **desplazamiento del byte** (*byte offset*) dentro de la palabra (como siempre, el número de bits de este campo será igual a $\log_2(\text{número de bytes de la palabra})$) y los restantes 30 bits más significativos, que forman la **dirección de palabra** (véase la tabla 5). Además, como el dato solicitado es una palabra completa, los 2 bits del campo *desplazamiento del bytes* se desprecian directamente.

Dentro de la dirección de palabra, los 2 bits menos significativos indican el **desplazamiento de la palabra** dentro del bloque (el número de bits de este campo será igual a $\log_2(\text{número de palabras del bloque})$), siendo los 28 bits restantes los que forman la **dirección de bloque**.

La dirección de bloque es la que se utiliza para acceder a la caché, tomando sus 12 bits menos significativos como **posición en caché** (*index*) a la que acceder (la posición en caché la marcarán los $\log_2(\text{número de posiciones de caché})$ bits menos significativos de la dirección de bloque) y los restantes 16 bits como **eti-**

Tabla 5: Uso de las direcciones de memoria para el acceso a la caché.

Dirección del dato solicitado por el procesador			
32 bits			
Dirección de palabra			Desplazamiento de byte
30 bits			2 bits
Dirección de bloque		Desplazamiento de palabra	Desplazamiento de byte
28 bits		2bits	2 bits
Etiqueta	Posición en caché	Desplazamiento de palabra	Desplazamiento de byte
16 bits	12 bits	2bits	2 bits

queta (*tag*) que se deben comparar con la etiqueta existente en la posición accedida para saber si el dato que buscamos está o no en caché.

Si el dato está en caché (acierto de caché, *hit*), tendremos que acceder a la palabra correcta dentro del bloque seleccionado, por lo que se utiliza el *desplazamiento de la palabra* dentro del bloque para seleccionar esta palabra en el multiplexor de salida.

El motivo del incremento del número de palabras por bloque es sacar partido a la localidad espacial de manera que la tasa de fallos disminuya. Pero, si aumentamos demasiado el tamaño del bloque, esta tasa de fallos podría crecer, porque el número de bloques que podrían estar en caché sería muy pequeño y, por lo tanto, habría una fuerte competencia por las posiciones de caché. Como resultado, los bloques serían expulsados de la caché antes de que muchas de las palabras que contienen fueran referenciadas. Como se muestra en la figura 6, incrementar el tamaño de bloque disminuye normalmente la tasa de fallos, pero la localidad espacial entre las palabras de un bloque desciende en un bloque muy grande. Consecuentemente, las mejoras en la tasa de fallo son menores e, incluso, puede aumentar.

Un problema más serio asociado con incrementar el tamaño de bloque es que la penalización por fallo aumenta. La penalización por fallo se determina como el tiempo requerido para ir a buscar el bloque a la memoria principal y cargarlo en caché. El tiempo de búsqueda de un bloque tiene dos partes: la latencia de la primera palabra y el tiempo de transferencia para el resto del bloque. Claramente, a menos que se cambie el sistema de memoria (esto lo veremos en una próxima sección), el tiempo de transferencia aumenta al crecer el tamaño de bloque.

5.2.2. Rendimiento de la caché

Los ciclos de una CPU se pueden dividir entre los ciclos en los que la CPU ejecuta un programa y los ciclos en los que la CPU espera a la memoria. Normalmente, se supone que el coste de los accesos a la caché que son aciertos son parte de los ciclos de ejecución normal de la CPU. Por lo tanto,

$$\text{Tiempo total de ejecución de un programa} = (\text{ciclos de ejecución de la CPU} + \text{ciclos de bloqueo por memoria}) \times \text{Tiempo de ciclo}$$

En general, podemos decir que los ciclos de bloqueo causados por la memoria son principalmente debidos a los fallos de caché. Estos ciclos, con un enfoque simplificado del sistema de memoria, se pueden definir como:

$$\text{Ciclos de bloqueo por Memoria} = \text{N}^\circ \text{ accesos a memoria en el programa} \times \text{Tasa de fallos} \times \text{Penalización por fallo}$$

donde la *penalización por fallos* viene dada en ciclos.

Ejemplo:

Tenemos los siguientes datos para un programa P:

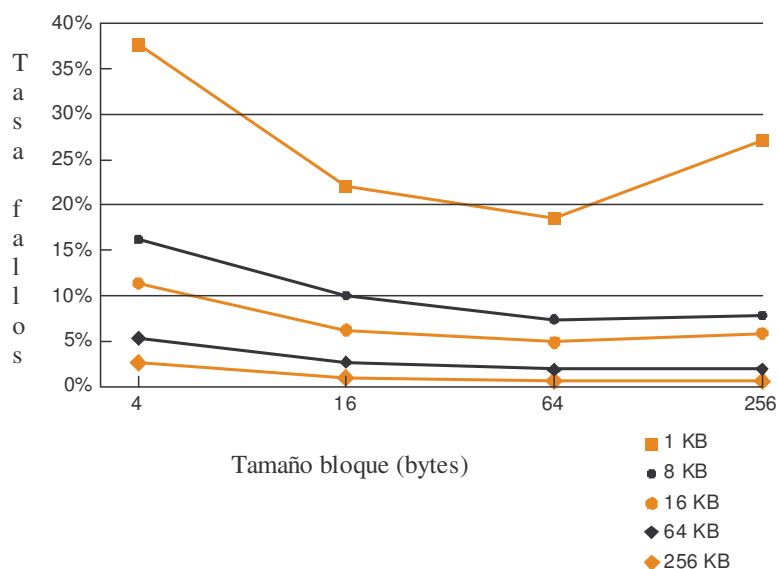


Figura 6: Tasa de fallos respecto al tamaño de bloque en cachés de diferentes tamaños. Con una caché de 1 KB y bloques de 256 bytes sólo hay 4 bloques en caché.

- La tasa de fallos de instrucciones (TF_{inst}) es del 2% (cuando se va a buscar el código de la siguiente instrucción a ejecutar, un 2% de las veces dicho código no se encuentra en caché).
- La tasa de datos por instrucción o D/I es del 36% (es decir, el 36% de las instrucciones acceden a algún dato de memoria).
- La tasa de fallos de datos (TF_{dat}) es del 4% (cuando se va a buscar un dato, el 4% de las veces dicho dato no se encuentra en caché).

Si el programa se ejecuta en una máquina M con una frecuencia de reloj de 50 MHz, un CPI_{ideal} (número de ciclos, en promedio, para ejecutar una instrucción sin bloqueos de memoria) de 2 ciclos y una penalización por fallo (PF) de 40 ciclos, determine cuánto más rápida sería una máquina con una caché perfecta.

Respuesta:

Para hallar la solución, vamos a calcular el CPI_{real} , es decir, el número de ciclos que se necesitan por término medio para ejecutar una instrucción, incluyendo los bloqueos de memoria:

- $CPI_{real} = CPI_{ideal} + \text{ciclos bloqueo por fallos instrucciones} + \text{ciclos bloqueo por fallos datos}.$
- Ciclos bloqueo por fallos de instrucción = $TF_{inst} \times PF = 2\% \times 40 = 0,80$ ciclos.
- Ciclos bloqueo por fallos de datos = $D/I \times TF_{dat} \times PF = 36\% \times 4\% \times 40 = 0,56$ ciclos.
- $CPI_{real} = 2 + 0,80 + 0,56 = 3,36$ ciclos.

Por otro lado, en una máquina con una caché perfecta, el CPI_{real} coincidiría con el CPI_{ideal} y dicha máquina sería:

- $CPI_{real} / CPI_{ideal} = 3,36 / 2 = 1,68$ veces más rápida.

Ejemplo:

Con los datos del ejemplo anterior, ¿qué pasaría si el procesador es más rápido (al reducir su CPI_{ideal} a 1), pero la memoria es la misma?

Respuesta:

Pues que la cantidad de tiempo destinada a los bloqueos de memoria tendrá un porcentaje mayor en el tiempo de ejecución. Ahora, con un $CPI_{ideal} = 1$, el CPI_{real} de la máquina es 2,36 ciclos, por lo que una máquina con una caché perfecta sería $2,36/1 = 2,36$ veces más rápida.

El porcentaje de tiempo de ejecución empleado en bloqueos de memoria habría crecido respecto de lo que teníamos en el ejemplo anterior desde el $1,36/3,36 = 41\%$ hasta el $1,36/2,36 = 58\%$.

Ejemplo:

¿Y si se incrementa el rendimiento de la máquina del primer ejemplo doblando la frecuencia de reloj del procesador a 100 MHz (con lo que el tiempo por ciclo de reloj es la mitad) manteniendo CPI_{ideal} en 2 ciclos?

Respuesta:

La penalización por fallo depende principalmente del tiempo de acceso a memoria para traer el bloque solicitado. Como en la nueva máquina el sistema de memoria es el mismo, este tiempo de acceso no varía (medido en segundos) pero medido en los ciclos de reloj de su CPU sería el doble que anteriormente, o sea, 80 ciclos. Por tanto:

- Ciclos bloqueo por fallos de instrucciones = $TF_{inst} \times PF = 2\% \times 80 = 1,60$ ciclos.
- Ciclos bloqueo por fallos de datos = $DI \times TF_{dat} \times PF = 36\% \times 4\% \times 80 = 1,12$ ciclos.
- $CPI_{real} = 2 + 1,60 + 1,12 = 4,76$ ciclos.

Con lo que el rendimiento relativo de esta nueva máquina respecto a la antigua sería:

$$\frac{T_{ejec \text{ a } 50 \text{ MHz}}}{T_{ejec \text{ a } 100 \text{ MHz}}} = \frac{CPI_{real} \times T_{ciclo}}{CPI_{real} \times T_{ciclo}} = \frac{3,36 \times \frac{1}{50 \times 10^6}}{4,76 \times \frac{1}{100 \times 10^6}} = 1,41$$

Por lo tanto, la máquina con el reloj más rápido es 1,4 veces más rápida en lugar de 2 veces más, como sería lo ideal. Esto es debido a que las penalizaciones relativas de la caché se incrementan conforme la máquina es más rápida.

En consecuencia, la importancia del rendimiento de la caché para una CPU con un CPI bajo y/o una frecuencia de reloj alta es mayor y, consecuentemente, el peligro de no tener en cuenta el comportamiento de la caché cuando se mide el rendimiento de estas máquinas también es mayor.

5.2.3. Memoria caché asociativa por conjuntos

En una memoria caché de correspondencia directa, cuando se ponía un bloque en la caché se utilizaba un esquema de asignación sencillo: cada bloque podía ir sólo a una posición de la caché. El otro extremo es un esquema donde un bloque puede ser asignado a cualquier posición de la caché. Este esquema se llama *totalmente asociativo* porque un bloque de la memoria puede ser asociado con cualquier entrada de la caché y, por tanto, para encontrar un bloque en este tipo de cachés se debe buscar en todas las entradas porque puede estar en cualquiera. El término medio entre los diseños de correspondencia directa y el totalmente asociativo es el *asociativo por conjuntos*.

En una caché *asociativa por conjuntos* hay m posiciones, donde en cada posición, a diferencia de la caché de correspondencia directa, hay espacio para almacenar n bloques de datos, diciéndose entonces que es una caché *n-asociativa* o *asociativa de n-vías*. A cada bloque que se trae de la memoria principal le corresponde una posición concreta de caché, pero, dentro de esa posición, puede ocupar cualquiera de los n huecos disponibles. Cada posición de la caché se convierte, por tanto, en un conjunto de n elementos.

Al igual que en la caché de correspondencia directa, la posición de caché que le corresponde a un bloque de memoria principal se calcula como:

$$\text{posición en caché} = (\text{dirección de bloque}) \text{ MOD } (\text{número posiciones caché})$$

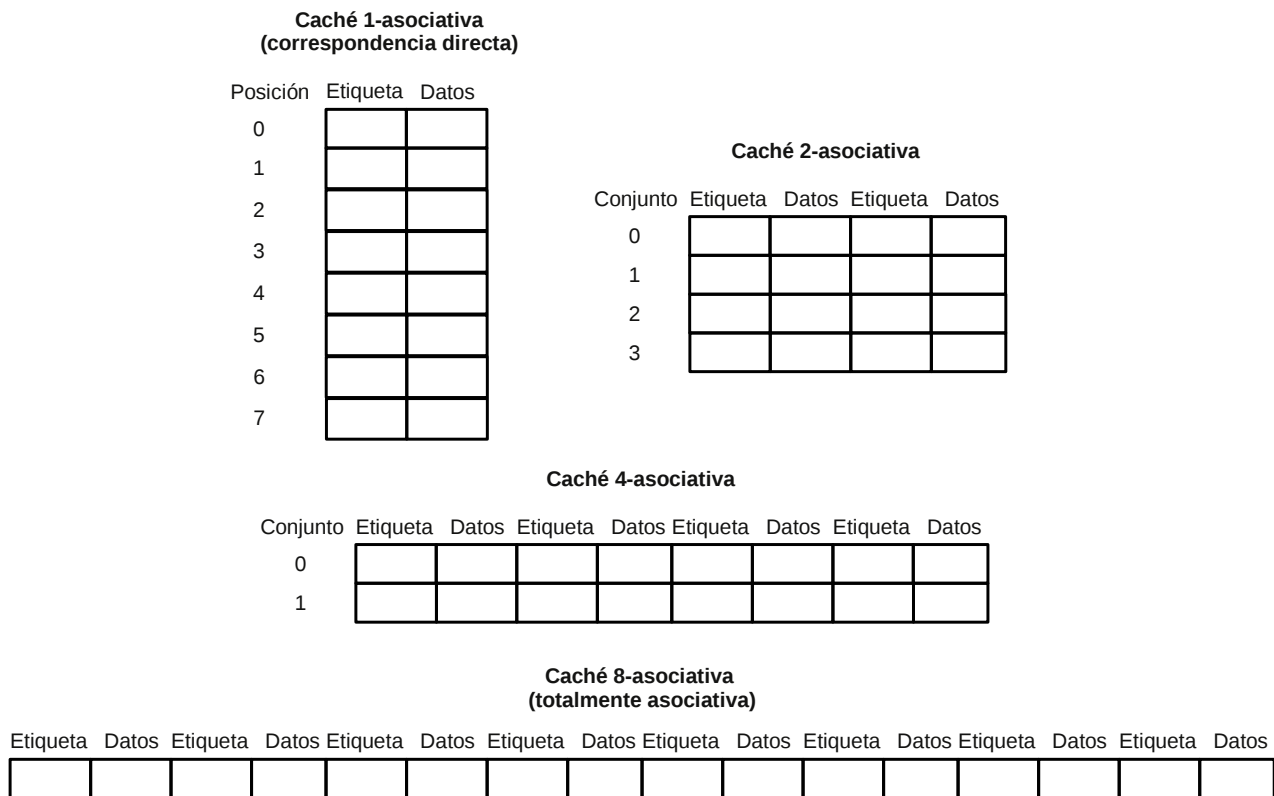


Figura 7: Cuatro posibilidades para una caché con capacidad para 8 bloques: de correspondencia directa, 2–asociativa, 4–asociativa y totalmente asociativa.

La diferencia es que ahora, una vez conocida la posición de caché, habrá que buscar entre todos los diferentes bloques que forman el conjunto de esa posición, comparando con todas las correspondiente etiquetas (en la caché de correspondencia directa había que comparar con la etiqueta del único bloque existente en una posición).

Se puede pensar en cada tipo de memoria caché visto hasta ahora como en una variación de la asociativa por conjuntos. Una caché de correspondencia directa es simplemente una caché 1–asociativa (cada posición de la caché es un conjunto formado por un solo bloque), mientras que una caché totalmente asociativa con m entradas es simplemente una caché m –asociativa (figura 7).

La ventaja de incrementar el grado de asociatividad es que normalmente hace descender la tasa de fallos gracias a una asignación más flexible del espacio disponible en caché a los bloques llegados desde la memoria principal, como se mostrará en el siguiente ejemplo. La desventaja, como se explicará más tarde, es un incremento en el tiempo de acierto.

Ejemplo:

- La tabla 6 muestra la evolución del contenido de una memoria caché de correspondencia directa de 4 posiciones según una secuencia de 5 referencias a bloques. Recordemos que, en este caso, la posición en caché se calcula como:

$$\text{posición en caché} = (\text{dirección de bloque}) \text{ MOD } 4$$

(= los dos bits menos significativos de la dirección de bloque)

- La tabla 7, por su parte, muestra la evolución del contenido de una memoria caché asociativa con 2 conjuntos de 2 bloques. En este caso, la posición en caché se calcula como:

Tabla 6: Evolución del contenido de una memoria caché de correspondencia directa de 4 posiciones.

Dir. bloque) ₁₀	Dir. bloque) ₂	Pos. caché	Acierto o Fallo	Contenido de la caché			
				Pos. 0	Pos. 1	Pos. 2	Pos. 3
0	0000	00	Fallo	M[0]			
8	1000	00	Fallo	M[8]			
0	0000	00	Fallo	M[0]			
6	0110	10	Fallo	M[0]		M[6]	
8	1000	00	Fallo	M[8]		M[6]	

Tabla 7: Evolución del contenido de una memoria caché asociativa con 2 conjuntos de 2 bloques.

Dir. bloque) ₁₀	Dir. bloque) ₂	Pos. caché	Acierto o Fallo	Contenido de la caché			
				Conjunto 0		Conjunto 1	
0	0000	0	Fallo	M[0]			
8	1000	0	Fallo	M[0]	M[8]		
0	0000	0	Acierto	M[0]	M[8]		
6	0110	0	Fallo	M[0]	M[6]		
8	1000	0	Fallo	M[8]	M[6]		

$$\text{posición en caché} = \text{conjunto de caché} = (\text{dirección de bloque}) \text{ MOD } 2$$

(= el bit menos significativo de la dirección de bloque)

Observa que esta caché tiene la misma capacidad que la caché anterior: 4 bloques.

- Finalmente, la tabla 8 muestra la evolución del contenido de una memoria caché totalmente asociativa. Esta caché también tiene la misma capacidad que las caché anteriores (4 bloques), pero es la que produce el menor número de fallos.

En una caché de correspondencia directa, como la de la figura 4, sólo se necesita un comparador porque el bloque buscado puede estar en un único hueco dentro de la posición que le corresponde y, simplemente, se accede a la caché indexando. En una caché 4–asociativa, como la mostrada en la figura 8, se necesitan cuatro comparadores, junto con un multiplexor 4–a–1 para escoger entre los cuatro miembros potenciales del conjunto seleccionado. El acceso a la caché consiste en indexar el conjunto apropiado y después buscar entre los elementos del conjunto. Como la velocidad es importante, todas las etiquetas de los elementos de un conjunto se comparan en paralelo.

Tabla 8: Evolución del contenido de una memoria caché totalmente asociativa.

Dir. bloque) ₁₀	Dir. bloque) ₂	Acierto o Fallo	Contenido de la caché			
			Conjunto único			
0	0000	Fallo	M[0]			
8	1000	Fallo	M[0]	M[8]		
0	0000	Acierto	M[0]	M[8]		
6	0110	Fallo	M[0]	M[8]	M[6]	
8	1000	Acierto	M[0]	M[8]	M[6]	

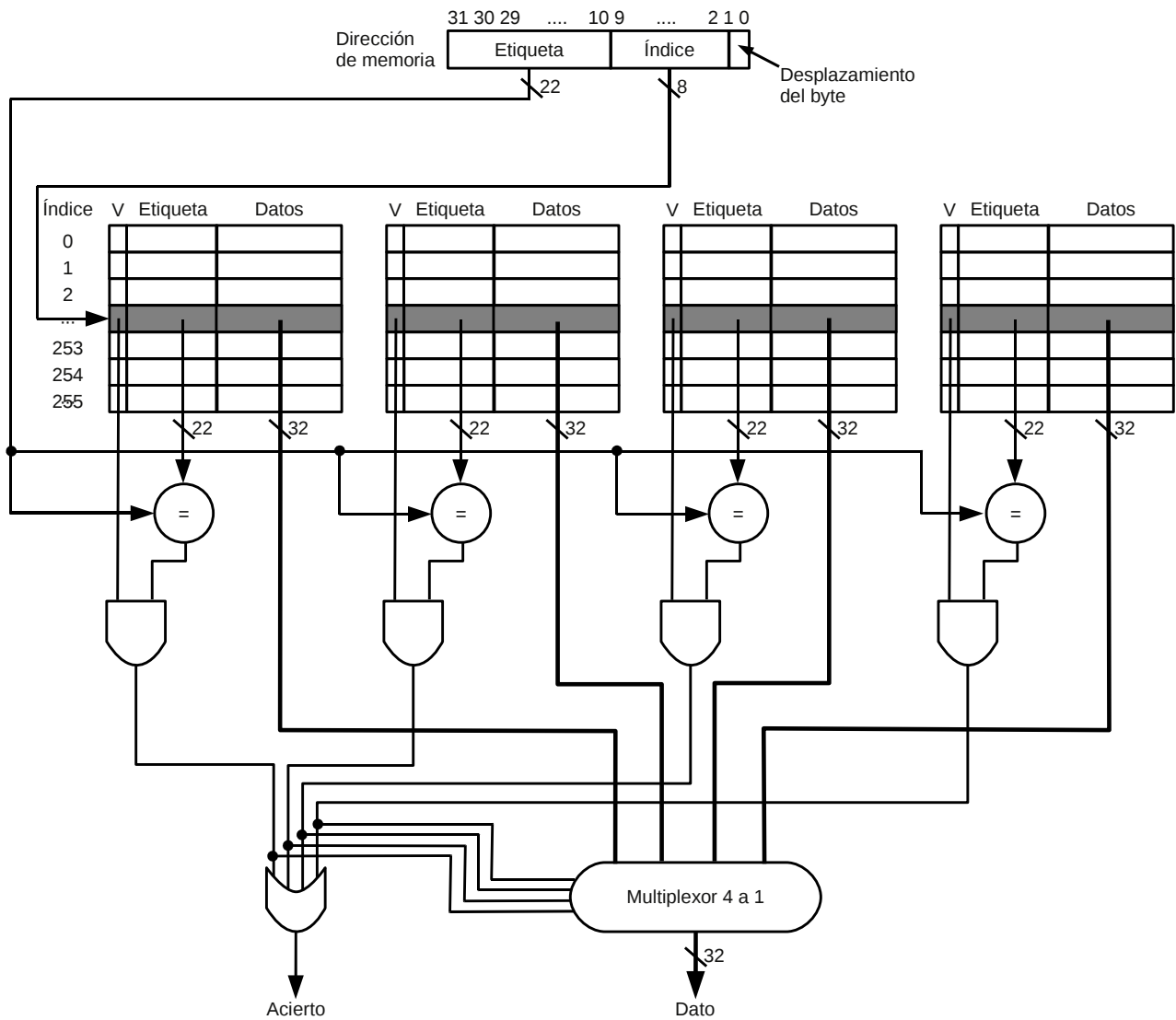


Figura 8: Construcción de una memoria caché 4-asociativa.

Ejemplo:

Aumentar la asociatividad requiere más comparadores, así como más bits para las etiquetas de los bloques de caché. Suponiendo una caché de 4096 (2^{12}) bloques, bloques monopalabra y direcciones de 32 bits, encontrar el número total de conjuntos y el número total de bits para las etiquetas para cachés de correspondencia directa, 2–asociativas, 4–asociativas y totalmente asociativas.

Respuesta:

La dirección de palabra es de $(32-2) = 30$ bits. Como los bloques son monopalabra, entonces la dirección de bloque es también de 30 bits. Veamos cada caso.

- La caché de correspondencia directa tiene el mismo número de conjuntos que de bloques (4096), y en consecuencia 12 bits de índice, ya que $\log_2(4096) = 12$; por lo tanto, el número total de bits para cada etiqueta de la caché será de $(30-12) = 18$ bits. Como hay un total de 4096 bloques, entonces el número total de bits dedicados para todas las etiquetas de caché será $4096 \times 18 = \mathbf{72\ Kbits}$.
- Para una caché 2–asociativa, hay 2048 conjuntos de 2 bloques y, en consecuencia, 11 bits de índice, ya que $\log_2(2048) = 11$; por lo tanto, el número total de bits para cada etiqueta de la caché será de $(30-11) = 19$ bits. Como hay un total de 4096 bloques, entonces el número total de bits dedicados para todas las etiquetas de caché será $4096 \times 19 = \mathbf{76\ Kbits}$.
- Para una caché 4–asociativa, hay 1024 conjuntos de 4 bloques y, en consecuencia, 10 bits de índice, ya que $\log_2(1024) = 10$; por lo tanto, el número total de bits para cada etiqueta de la caché será de $(30-10) = 20$ bits. Como hay un total de 4096 bloques, entonces el número total de bits dedicados para todas las etiquetas de caché será $4096 \times 20 = \mathbf{80\ Kbits}$.
- Finalmente, para una caché totalmente asociativa, hay un único conjunto de 4096 bloques, y en consecuencia 0 bits de índice, ya que $\log_2(1) = 0$; por lo tanto, el número total de bits para cada etiqueta de la caché será de $(30-0) = 30$ bits (todos los bits de la dirección de bloque son usados para la etiqueta). Como hay un total de 4096 bloques, entonces el número total de bits dedicados para todas las etiquetas de caché será $4096 \times 30 = \mathbf{120\ Kbits}$.

La selección entre una caché de correspondencia directa, asociativa por conjuntos o totalmente asociativa, en cualquier jerarquía de memorias, dependerá del coste de cada fallo frente al coste de la asociatividad, tanto en tiempo como en circuitería extra.

5.3. Tratamientos de los fallos de caché

5.3.1. Tratamiento de los fallos de lectura

Cuando el procesador requiere la lectura de una palabra de memoria, si esta palabra está en caché, se envía al procesador; en caso contrario, se habrá producido un fallo de lectura y, entonces, habrá que traer desde memoria principal a caché el bloque que contiene esa palabra y colocarlo en la posición de caché correspondiente a su dirección de bloque. En una caché de correspondencia directa, si esa posición está ocupada por otro bloque, éste se sustituye por el nuevo. Sin embargo, si la caché es asociativa y el conjunto de la posición de caché que le corresponde al nuevo bloque está lleno, necesitamos un algoritmo de sustitución para decidir qué bloque de ese conjunto será sustituido. Las dos principales estrategias son:

- **Aleatoria:** se elige un bloque al azar.
- **Usado Menos Recientemente** (*Last Recently Used, LRU*): el bloque reemplazado es el que ha estado más tiempo sin utilizar.

El algoritmo de reemplazo se implementa en circuitería para mayor rapidez, lo que significa que el esquema tiene que ser fácil de realizar. El esquema aleatorio es fácil de construir en circuitería y, para una caché 2–asociativa, la tasa de fallos es sólo 1,1 veces mayor que usando LRU.

En la práctica, el algoritmo LRU es demasiado costoso de realizar cuando aumentamos la asociatividad, pues aumenta demasiado la información que se necesita mantener. En ocasiones se utiliza algún esquema pseudo–LRU como, por ejemplo, el que se podría hacer con un solo bit por bloque, llamado *bit de uso o referencia*, que inicialmente está a cero, y que se pone a 1 si alguna palabra del bloque es referenciada. A la hora de sustituir una página, se elegirá alguna que tenga a 0 el bit de referencia (significa que recientemente no se ha utilizado). Nótese que, para que este esquema pseudo–LRU funcione, es necesario que el bit de uso o referencia se ponga a 0 periódicamente en todos los bloques.

5.3.2. Tratamiento de los fallos de escritura

Cuando el procesador quiere escribir una palabra en memoria, según esté o no la palabra en caché y según la política de escritura que tenga la caché, se pueden dar estas situaciones:

- La palabra está en caché:
 - **Política de escritura directa (*write through*)**: la palabra se escribe tanto en el bloque de caché como en el bloque de memoria principal.
 - **Política de postescritura (*write back*)**: la palabra se escribe sólo en el bloque de caché. Se activa un bit de control llamado *bit de modificación* que indicará que el bloque de caché modificado se escribirá completamente en la memoria principal cuando sea sustituido por otro.
- La palabra NO está en caché:
 - **Política de escritura directa (*write through*)**: la solución más común, de cara a mantener la simplicidad de la circuitería asociada a esta política, consiste en escribir directamente la palabra en memoria principal, sin traer su bloque a caché (*no-write allocate*).
 - **Política de postescritura (*write back*)**: en las de bloques multipalabra, es necesario traer a caché el bloque al que pertenece la palabra que se quiere modificar (aunque no se necesita saber el contenido de la palabra que se va a modificar, sí necesitamos tener el bloque completo con los valores del resto de palabras de dicho bloque) y, tras ello, se escribe la palabra en el bloque traído a caché (*write allocate*). Nótese que, en las cachés con bloques monopalabra, no es necesaria ninguna lectura de la memoria principal previa a escribir la palabra en caché.

Las ventajas más importantes del esquema de postescritura son:

- Las palabras se pueden escribir muy rápidamente, pues sólo se escriben en caché y no en memoria principal.
- Múltiples escrituras dentro de un bloque requieren sólo una escritura en la memoria principal (cuando el bloque es sustituido por otro).
- Las escrituras en memoria principal son siempre de bloques completos, con lo que el sistema puede hacer un uso efectivo de un alto ancho de banda en esa comunicación.

Las ventajas más importantes del esquema de escritura directa son:

- Los fallos son menos costosos ya que nunca requieren que un bloque completo se escriba en memoria principal. En los fallos de lectura, si un bloque es expulsado de caché para ser sustituido por otro no es necesario reescribirlo en memoria principal. En los fallos de escritura sólo se escribe un palabra en memoria principal.

- Este esquema es más fácil de realizar desde el punto de vista del hardware necesario, aunque, para ser práctico en un sistema muy rápido, necesita de un *buffer de escrituras* que almacene los datos mientras están esperando para ser escritos en memoria. Después de escribir los datos en la caché y en el buffer de escrituras, el procesador puede continuar la ejecución. Cuando una escritura a memoria finaliza, se libera la entrada en el buffer de escrituras. Si el buffer está lleno cuando el procesador llega a una escritura, el procesador tiene que pararse hasta que haya una entrada libre en el buffer.

5.4. Memorias caché multinivel

Para reducir la diferencia entre las grandes frecuencias de reloj de los procesadores de hoy en día y el gran tiempo para acceder a la memoria principal (tecnología DRAM), la mayoría de microprocesadores actuales incorporan, al menos, un nivel adicional de caché. A este segundo nivel de caché, que en la actualidad está integrado en el propio chip del procesador, se accede cuando ocurre un fallo en la caché primaria. Si el segundo nivel de caché contiene los datos deseados, la penalización por fallo del primer nivel será el tiempo de acceso al segundo nivel, que será mucho menor que el tiempo de acceder a memoria principal. Si ni el nivel primario ni el secundario de caché contienen los datos, se requiere acceder a la memoria principal, lo que produce una penalización por fallo mayor.

Ejemplo:

Veamos con un ejemplo cómo se produce la mejora del rendimiento al usar una caché secundaria. Supóngase un procesador con un CPI_{ideal} de 1 ciclo y un reloj a 500MHz. Si se supone un tiempo de acceso a la memoria principal de 200 ns, incluida la gestión del fallo, y una tasa de fallos por instrucciones en la caché primaria de un 5 %, ¿cuánto más rápida será la máquina al añadir una segunda caché que tiene un tiempo de acceso de 20 ns y una tasa de fallos del 2 %?

Respuesta:

En la máquina original tenemos que la penalización por fallo es de:

$$\frac{200 \text{ ns}}{2 \text{ ns/ciclo}} = 100 \text{ ciclos}$$

Con lo que:

$$CPI_{real} = CPI_{ideal} + \text{ciclos de bloqueo de memoria} = 1 + 5 \% \times 100 = 6 \text{ ciclos}$$

En la máquina con 2 niveles de caché tenemos que el número de ciclos necesarios para acceder a la caché secundaria es:

$$\frac{20 \text{ ns}}{2 \text{ ns/ciclo}} = 10 \text{ ciclos}$$

Con lo que:

$$CPI_{real} = CPI_{ideal} + \text{ciclos bloqueo memoria} = 1 + 5 \% \times (10 + 2 \% \times 100) = 1,6 \text{ ciclos}$$

O lo que es lo mismo: 1 ciclo ideal, más el 5 % de las veces que falla la caché primaria que conlleva acceder a la caché secundaria (con un tiempo de acceso de 10 ciclos en caso de acierto). De esos accesos a la caché secundaria, un 2 % de las veces se produce un nuevo fallo que conlleva un acceso a memoria principal (100 ciclos). Por lo tanto, la máquina con 2 niveles de caché es $(6/1,6) = 3,75$ veces más rápida que la original.

5.5. Características de la memoria caché en algunos sistemas actuales

Las tablas 9, 10 y 11 muestran, respectivamente, las características más relevantes de las cachés de nivel 1, 2 y 3 de un procesador AMD Phenom II y un procesador IBM Power6. Las cachés de estos procesadores

Tabla 9: Características de las cachés L1 del AMD Phenom II y el IBM Power6. Hay una caché de datos y otra de instrucciones por cada núcleo.

Característica	AMD Phenom II	IBM Power6
Organización de la caché	Cachés I+D separadas	Cachés I+D separadas
Tamaño de la caché	64 KB cada una	64 KB cada una
Asociatividad	2–asociativa	4–asociativa (I) / 8–asociativa (D)
Reemplazo	ND	ND
Tamaño de bloque	ND	128 bytes
Política de escritura	ND	Escritura directa

Tabla 10: Características de las cachés L2 del AMD Phenom II y el IBM Power6. Hay una caché por núcleo y es compartida por datos e instrucciones.

Característica	AMD Phenom II	IBM Power6
Organización de la caché	Caché compartida	Caché compartida
Tamaño de la caché	512 KB	4 MB
Asociatividad	16–asociativa	8–asociativa
Reemplazo	ND	Pseudo–LRU
Tamaño de bloque	ND	128 bytes
Política de escritura	ND	Postescritura

Tabla 11: Características de las cachés L3 del AMD Phenom II (integrada en el chip del procesador) y el IBM Power6 (fuera del chip del procesador). En ambos casos, la caché es compartida por todos los núcleos del procesador.

Característica	AMD Phenom II	IBM Power6
Organización de la caché	Caché compartida	Caché compartida
Tamaño de la caché	6 MB	32MB
Asociatividad	64–asociativa	16–asociativa
Reemplazo	ND	Pseudo–LRU
Tamaño de bloque	ND	128 bytes
Política de escritura	ND	Caché de víctimas

utilizan códigos de detección (y, a veces, de corrección) de errores para mejorar la fiabilidad del sistema de memoria dentro del procesador (y, por ende, del propio procesador).

Ya que tanto el Phenom II como el Power6 son procesadores multinúcleo (o *multicore*), puede haber cachés de uso exclusivo para cada núcleo y cachés que sean compartidas por todos los núcleos del procesador. En estos procesadores, hay dos cachés de nivel 1 (una para datos y otra para instrucciones) y una caché de nivel 2 por cada núcleo. La caché de nivel 3, en cambio, es compartida por todos los núcleos.

Podemos observar que la caché de tercer nivel del IBM Power6 es una caché que se encuentra fuera del procesador y es, por tanto, opcional (para una determinada configuración del hardware, podría no existir). Ésta es una *caché de víctimas*. Esta caché sólo almacenan bloques expulsados (víctimas) de la caché de nivel 2 con el fin de darles una segunda oportunidad. Dicho de otro modo, si un bloque no se encuentra ni en la caché de nivel 2 ni en la caché de nivel 3, el bloque se solicita a memoria y se copia directamente a la caché de nivel 2, sin pasar por la caché de nivel 3. Sólo si el bloque se expulsa del nivel 2, se almacenará en la caché de nivel 3 de donde puede pasar de nuevo a la caché de nivel 2 si es solicitado de nuevo.

Apéndices

A5.1. Diseño del sistema de memoria para soportar cachés

La memoria principal, que está construida con DRAMs, suministra los datos cuando se producen fallos de caché. Aunque es difícil reducir la latencia de ir a buscar la primera palabra a memoria, se puede reducir la penalización por fallo si se incrementa el ancho de banda entre la memoria principal y la caché. Este incremento de ancho de banda permite leer o escribir bloques grandes en un tiempo sólo ligeramente superior al de leer o escribir una única palabra, lo que permite reducir, en promedio, la penalización por fallo por palabra.

Para entender el impacto de diferentes organizaciones de memoria, vamos a considerar una serie de tiempos de acceso a memoria hipotéticos para una operación de lectura:

- 1 ciclo para enviar la dirección a memoria.
- 15 ciclos de latencia para cada acceso a la DRAM iniciado (este es el tiempo que la DRAM necesita para preparar los datos a enviar).
- 1 ciclo para enviar una palabra de datos.

Con los tiempos de acceso a memoria hipotéticos mostrados, el tiempo necesario (en ciclos) para leer un bloque de 4 palabras en cada uno de los tres posibles diseños del sistema de memoria que aparecen en la figura A5.1 es el siguiente:

- En el primer diseño del sistema de memoria, se tiene un único banco de DRAMs de una palabra de ancho. La penalización por fallo, es decir, el tiempo necesario para leer un bloque de 4 palabras, será: 1 (para enviar a memoria la dirección de la primera palabra del bloque) + 4×15 (tiempo que necesita la memoria para preparar las 4 palabras) + 4×1 (tiempo para enviar las 4 palabras a caché) = 65 ciclos. Este tiempo es muy superior al necesario para leer una única palabra de memoria que es de 17 ciclos ($=1+15+1$).
- En el segundo diseño del sistema de memoria, se incrementa el ancho del bus, con lo que se incrementa proporcionalmente el ancho de banda de memoria. Con una memoria de 4 palabras de ancho, la penalización por fallo baja a: 1 (para enviar a memoria la dirección de la primera palabra del bloque) + 1×15 (tiempo que necesita la memoria para preparar las 4 palabras; recordemos que la memoria tiene un ancho de 4 palabras) + 1×1 (tiempo para enviar las 4 palabras a la vez a caché) = 17 ciclos. Los costes de esta mejora son un bus más ancho y el incremento potencial del tiempo de acceso a la caché debido al multiplexor y a la lógica de control entre la CPU y la caché. Además, con este diseño no es posible leer una única palabra de memoria ya que tanto la caché, como el bus, como la propia memoria trabajan con bloques de 4 palabras.
- En este tercer y último diseño del sistema de memoria, los chips de la memoria pueden estar organizados en bancos para leer o escribir múltiples palabras en el tiempo de un solo acceso, más que leer o escribir una sola palabra cada vez. Cada banco puede tener un ancho de palabra y, por tanto, el ancho del bus y de la caché no hay que cambiarlos, como ocurría en el caso anterior. Enviar una dirección a los diferentes bancos permite que lean todos simultáneamente, con lo que la latencia de memoria sólo se cuenta una vez. La penalización por fallo sería: 1 (para enviar a memoria la dirección de la primera palabra del bloque) + 1×15 (tiempo que necesitan los cuatro módulos de memoria para preparar las 4 palabras) + 4×1 (tiempo para enviar las 4 palabras a caché) = 20 ciclos. Este tiempo es sólo ligeramente superior al necesario para leer una única palabra de memoria que es de 17 ciclos ($=1+15+1$).

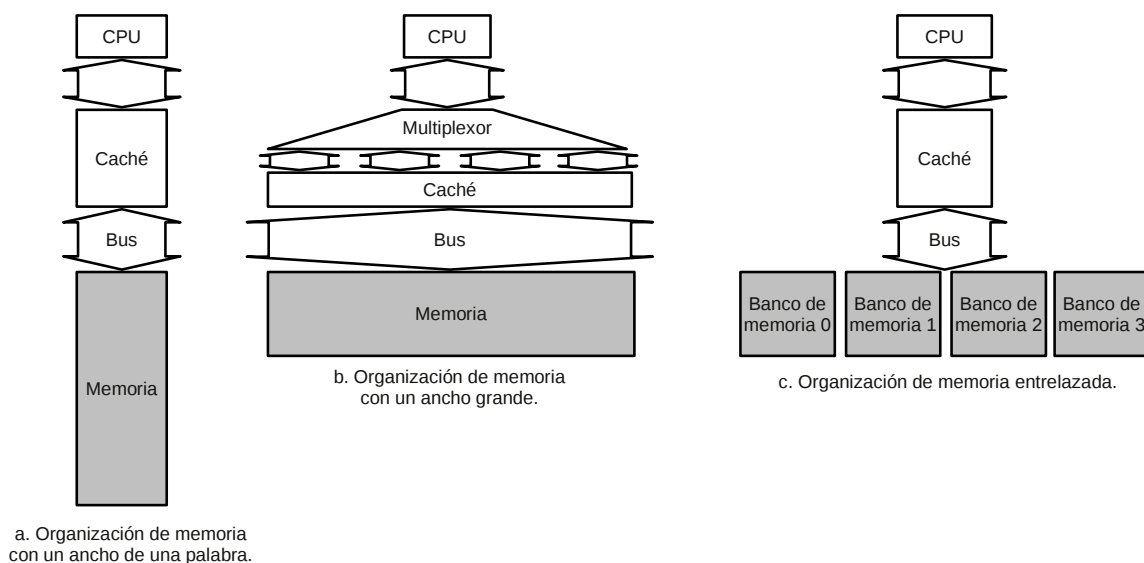


Figura A5.1: Tres posibles diseños del sistema de memoria para soportar cachés.

Como podemos observar, los diseños a) y c) son prácticamente iguales, sólo se diferencian en cómo se organiza la propia memoria DRAM. Al utilizar bancos de memoria separados y entrelazados, el tercer diseño permite reducir sustancialmente el tiempo necesario para transferir un bloque de 4 palabras lo que, como ya hemos dicho, le permite también reducir considerablemente la penalización por fallo que en media se produce por palabra.

Boletines de prácticas

Normas sobre la entrega de prácticas

Será necesario seguir las siguientes reglas para entregar las prácticas de todos los boletines, además de cualquier otra regla específica que se indique en un boletín particular:

- Las prácticas se entregarán únicamente mediante la opción de contenidos del alumno en SUMA.
- Se entregará un único archivo comprimido en formato *.tar.gz* o *.zip* que contendrá la memoria en formato PDF, los circuitos y programas que se hayan generado (código fuente) y cualquier otro fichero que se considere oportuno. El nombre del archivo será *prácticas-DNI-BOLETIN.FORMATO* (por ejemplo: *prácticas-12345678-B2.3.tar.gz*).
- La memoria incluirá, al menos, la siguiente información en un solo documento PDF:
 - Nombre y DNI del autor o autores de la práctica.
 - Descripción de los ficheros y directorios contenidos en el archivo entregado.
 - Contestación a las preguntas planteadas en los boletines. La respuesta a cada pregunta debe ser independiente, y debe estar claramente identificada.
 - Explicación de las pruebas realizadas para comprobar la corrección de la práctica entregada e instrucciones para su reproducción. Cuando sea posible, se incluirán los ficheros utilizados en dichas pruebas.
 - Explicación del trabajo realizado y cualquier aclaración que el alumno considere pertinente.
 - Lista de bibliografía y otras fuentes de información consultadas.

No se corregirá ninguna práctica que no se ciña estrictamente a los formatos especificados anteriormente.

B5.1. Simulación de cachés

B5.1.1. Objetivos

Los objetivos de esta sesión son que el alumno se familiarice con el simulador de cachés que proporciona MARS y que, a través de dicho simulador y de programas sencillos, el alumno entienda mejor el funcionamiento de los diferentes tipos de cachés vistos en teoría.

B5.1.2. Prerequisitos

- Lectura de los apuntes de teoría.

B5.1.3. Plan de trabajo

El plan de trabajo de esta sesión será el siguiente:

1. Lectura por parte del alumno de la sección **B5.1.4**.
2. Realización, de forma individual, de los ejercicios propuestos en el boletín (con supervisión del profesor).

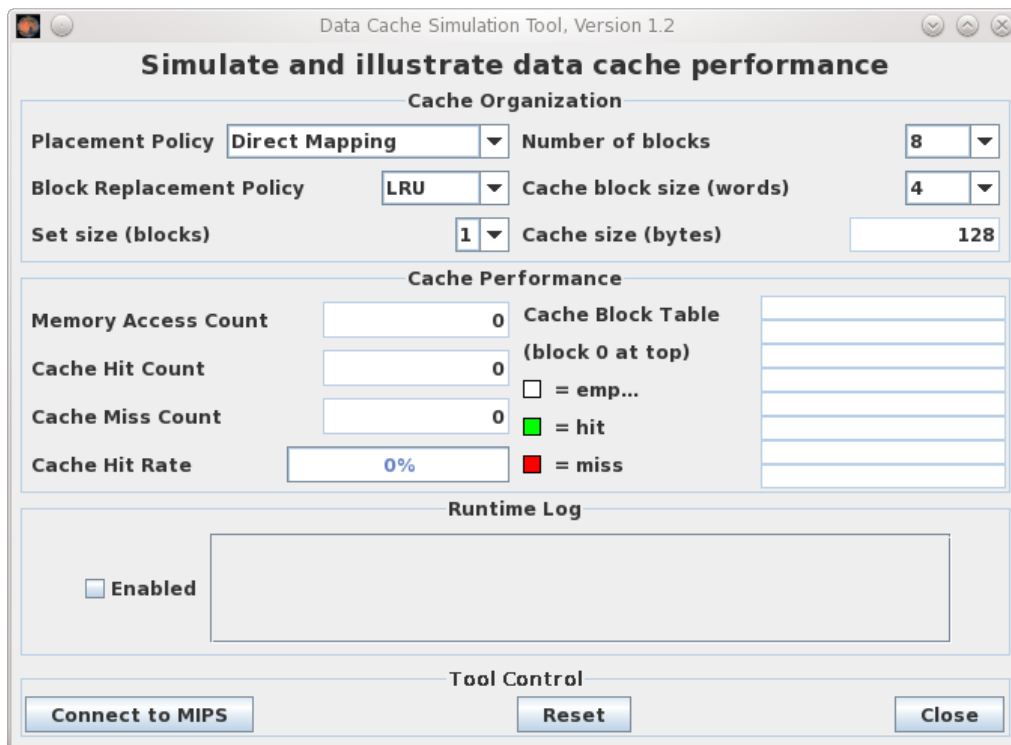


Figura B5.1: Ventana del simulador de cachés de MARS.

B5.1.4. El simulador de cachés de MARS

Bajo la opción del menú `Tools`, MARS proporciona algunas herramientas que pueden ser útiles para comprender mejor el funcionamiento de algunos de los elementos que constituyen la arquitectura de un ordenador. Entre estas herramientas se encuentra un simulador de cachés, al que se accede a través de `Tools` → `Data Cache Simulator`.

La ventana del simulador de cachés de MARS (ver figura B5.1) se encuentra dividida en dos partes. La parte superior, `Cache Organization`, determina el tipo de caché a simular, mientras que la parte inferior, `Cache Performance`, muestra el rendimiento de la caché durante la ejecución de un determinado programa.

El simulador es capaz de simular los tres tipos de cachés que hemos visto en teoría. El tipo de caché a simular se puede seleccionar mediante la opción `Placement Policy` y éste puede ser: `Direct Mapping` (para las cachés de correspondencia directa), `N-way Set Associative` (para las cachés asociativas de N-vías) y `Fully Associative` (para las cachés totalmente asociativas). En todos los casos se pueden simular tanto bloques monopalabra como multipalabra. El número de palabras de un bloque se puede fijar mediante la opción `Cache block size (words)`.

En el caso de las cachés asociativas, el algoritmo de reemplazo se selecciona mediante la opción `Block Replacement Policy`. El simulador nos proporciona dos tipos de algoritmo de reemplazo: `LRU` y `Random` (éste último, para un reemplazo aleatorio). Por otro lado, el número de vías, en el caso de una caché asociativa de N-vías, se puede fijar mediante la opción `Set size (blocks)`. Como vemos, el número de vías es el número de bloques que almacena cada conjunto (o posición) de la caché.

Dese cuenta que el tamaño total de la caché viene determinado por el número de bloques de ésta (`Number of blocks` y el tamaño de cada bloque (la opción `Cache block size (words)` que ya hemos comentado). En función del tipo de caché, el número de bloques determina:

- El número de posiciones, en una caché de correspondencia directa.

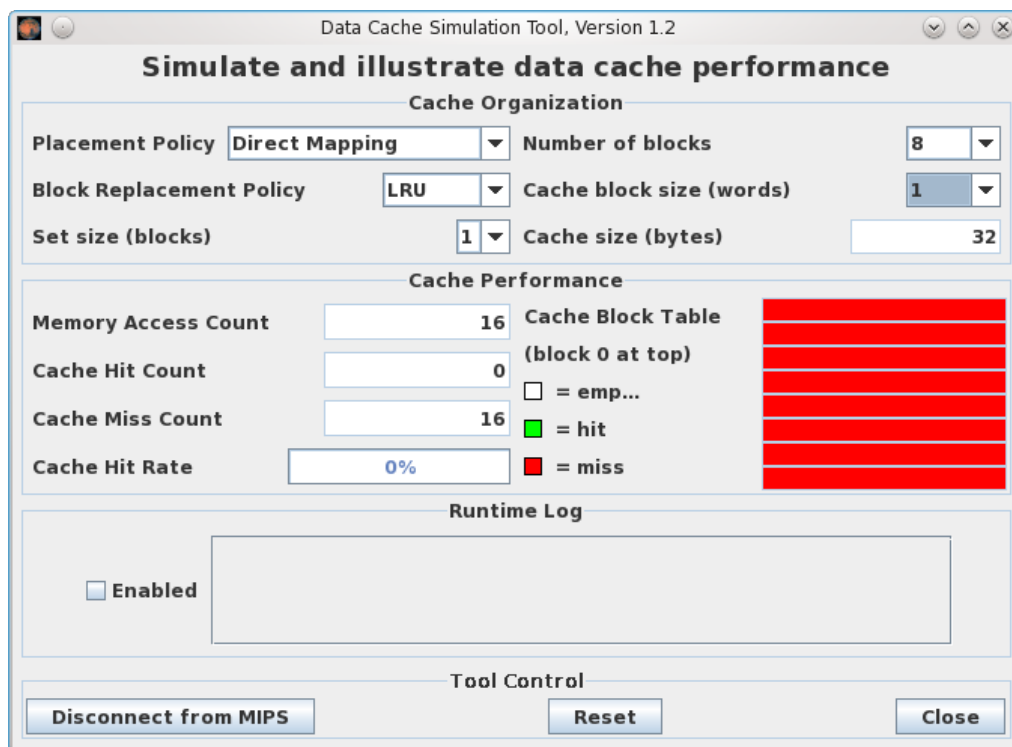


Figura B5.2: Estado de la caché tras la ejecución del programa de la figura B5.3.

- El número de conjuntos, en una caché asociativa de N -vías. En este caso, hay que dividir el número de bloques entre el tamaño del conjunto (fijando mediante la opción *Set size (blocks)*, ya comentada) para obtener el número de conjuntos.
- El tamaño del único conjunto disponible, en una caché totalmente asociativa.

Como ya hemos comentado, la parte inferior de la ventana del simulador de cachés muestra el rendimiento de la caché durante la ejecución de un determinado programa. Para ello, antes de la ejecución del programa, tendremos que “conectar” el simulador de cachés a la máquina MIPS pulsando el botón *Connect to MIPS*. La figura B5.2 muestra el estado final de una caché de correspondencia directa de 8 bloques monopolpalabra tras la ejecución del programa de la figura B5.3.

Como podemos observar, el simulador nos muestra que se han realizado 16 accesos a memoria (*Memory Access Count*) de los cuales 0 han sido aciertos de caché (*Cache Hit Count*) y 16 han sido fallos (*Cache Miss Count*). Estos resultados nos dan una tasa de aciertos (*Cache Hit Rate*) del 0%.

Hay dos aspectos del simulador que merece la pena destacar. El primero es que el simulador sólo simula una caché de datos. Por tanto, la ejecución de las propias instrucciones no produce ni aciertos ni fallos de caché. El segundo es que la caché utiliza escritura directa, por lo que cualquier dato que se escribe en caché también se escribe en memoria principal.

B5.1.5. Ejercicios

Para cada ejercicio, el portafolios deberá incluir una explicación de cómo se ha resuelto y, en caso de que se realice o modifique algún programa, el código realizado y, al menos, un ejemplo de su ejecución.

1. Identifique en el programa `suma-arrays.s` todas las instrucciones que acceden a datos *durante la ejecución del mismo* y que, por tanto, hacen uso de la caché. Suponiendo una caché 2-asociativa, con un tamaño total de 8 bloques monopolpalabra y un algoritmo de reemplazo LRU, para cada instrucción que

```

1      .data
2 array_a: .word 0, 1, 2, 3, 4, 5, 6, 7
3 array_b: .word 8, 9, 10, 11, 12, 13, 14, 15
4
5      .text
6 main:  la      $t0, array_a
7        la      $t1, array_b
8        li      $t2, 8
9        add     $t3, $zero, $zero
10 loop:
11       lw      $t4, 0($t0)
12       lw      $t5, 0($t1)
13       add     $t3, $t3, $t4
14       add     $t3, $t3, $t5
15       addi    $t0, $t0, 4
16       addi    $t1, $t1, 4
17       addi    $t2, $t2, -1
18       bne    $t2, $zero, loop
19
20       move    $a0, $t3
21       li     $v0, 1
22       syscall
23       li     $v0, 10
24       syscall
25
26       jr     $ra

```

Figura B5.3: suma-arrays.s

accede a datos indique: la dirección (o direcciones) de memoria a la que accede, el conjunto (posición) y bloque dentro del mismo (vía) en donde se almacena el dato leído o escrito, la etiqueta asignada al bloque, y el tipo de acceso (acierto o fallo de caché).

2. Modifique la caché como crea conveniente (es decir, cambie el tipo, tamaño, etc.) para obtener una tasa de aciertos distinta de 0 durante la ejecución del programa. Incluya una imagen con el estado final de la caché y explique por qué las modificaciones realizadas han incrementado la tasa de aciertos.
3. Repita el primer ejercicio, pero esta vez usando la configuración de la caché dada como solución en el ejercicio anterior.

B5.2. Organización de programas

B5.2.1. Objetivos

El objetivo de esta sesión es entender cómo la forma en la que se accede a los datos puede determinar el rendimiento de un programa, según una determinada organización de la caché.

B5.2.2. Prerequisitos

- Lectura de los apuntes de teoría.

B5.2.3. Plan de trabajo

El plan de trabajo de esta sesión será el siguiente:

1. Lectura por parte del alumno de la sección **B5.2.4**.
2. Realización, de forma individual, de los ejercicios propuestos en el boletín (con supervisión del profesor).

B5.2.4. Acceso a datos y rendimiento de la caché

El rendimiento que nos proporciona una caché viene determinado no sólo por la forma en la que se organiza la misma sino también por la forma en la que los programas hacen uso de ella. Generalmente, los procesadores incluyen una organización de caché determinada que no es posible cambiar. Esto hace que sea importante prestar atención al modo en el que un programa accede a la caché.

Cuando se programa utilizando un lenguaje de alto nivel, el compilador puede realizar, de manera transparente al programador, ciertas optimizaciones que permiten generar un código más eficiente para su ejecución en un determinado procesador o familia de procesadores. Sin embargo, cuando se programa en lenguaje ensamblador, dichas optimizaciones deben ser realizadas por el propio programador.

Para ilustrar esto, veamos un pequeño ejemplo. El programa de la figura **B5.3**, que aparece en el boletín anterior, suma todos los elementos del array «a» (`array_a`) y todos los elementos del array «b» (`array_b`) y muestra por pantalla el resultado de dicha suma. Utilizando MARS y su simulador de cachés, podremos ver que la ejecución de dicho programa con una caché de correspondencia directa de 4 bloques y bloques de 2 palabras producirá 16 accesos a memoria, todos ellos fallos de caché, lo que nos da un porcentaje de aciertos del 0 %.

Si analizamos la ejecución del programa, observaremos que, a pesar de que a los arrays se accede de manera secuencial y de que utilizamos bloques multipalabra, nunca se produce un acierto de caché en la segunda palabra de cada bloque porque cada acceso al array «b» provoca la expulsión de la caché del bloque del array «a» que se ha leído previamente. El problema es que cada elemento `b[i]` del array «b» ocupa en la caché la misma posición que el correspondiente elemento `a[i]` del array «a».

Para evitar el problema anterior, la solución es bien sencilla y consiste en sumar primero todos los elementos del array «a» y después todos los elementos del array «b». De esta manera, aprovechamos en el acceso secuencial que se hace a los arrays los bloques multipalabra de la caché. El código quedaría como se muestra en la figura **B5.4** (evidentemente, el código se puede mejorar, por ejemplo, reutilizando registros).

Utilizando de nuevo MARS y su simulador de cachés, podremos ver que la ejecución de este código produce un total de 16 accesos a datos, de los cuales 8 son fallos de caché y 8 son aciertos, lo que nos da ahora un porcentaje de aciertos del 50 %.

B5.2.5. Ejercicios

Para cada ejercicio, el portafolios deberá incluir una explicación de cómo se ha resuelto y, en caso de que se realice o modifique algún programa, el código realizado y al menos un ejemplo de su ejecución.

1. El programa de la figura **B5.5** suma los vectores «a» y «b», deja el resultado en un tercer vector «c» y, finalmente, multiplica por un escalar (4 en este caso) cada elemento del array «c». Suponiendo una caché de correspondencia directa de 4 bloques monopalabra, modifica dicho programa para reducir la tasa de fallos de caché sin aumentar el número de acceso a memoria.

```
1      .data
2 array_a: .word 0, 1, 2, 3, 4, 5, 6, 7
3 array_b: .word 8, 9, 10, 11, 12, 13, 14, 15
4
5      .text
6 main:  la      $t0, array_a
7        la      $t1, array_b
8        li      $t2, 8
9        add     $t3, $zero, $zero
10 loop1:
11        lw      $t4, 0($t0)
12        add     $t3, $t3, $t4
13        addi   $t0, $t0, 4
14        addi   $t2, $t2, -1
15        bne   $t2, $zero, loop1
16
17        li      $t2, 8
18 loop2:
19        lw      $t5, 0($t1)
20        add     $t3, $t3, $t5
21        addi   $t1, $t1, 4
22        addi   $t2, $t2, -1
23        bne   $t2, $zero, loop2
24
25        move   $a0, $t3
26        li    $v0, 1
27        syscall
28        li    $v0, 10
29        syscall
30
31        jr    $ra
```

Figura B5.4: suma-arrays-mejorada.s

```
1      .data
2 array_a: .word 0, 1, 2, 3, 4, 5, 6, 7
3 array_b: .word 8, 9, 10, 11, 12, 13, 14, 15
4 array_c: .space 32
5
6      .text
7 main:  la      $t0, array_a
8        la      $t1, array_b
9        la      $t2, array_c
10       li      $t3, 8
11 loop1:
12       lw      $t4, 0($t0)
13       lw      $t5, 0($t1)
14       add     $t4, $t4, $t5
15       sw      $t4, 0($t2)
16       addi   $t0, $t0, 4
17       addi   $t1, $t1, 4
18       addi   $t2, $t2, 4
19       addi   $t3, $t3, -1
20       bne    $t3, $zero, loop1
21
22       la      $t2, array_c
23       li      $t3, 8
24 loop2:
25       lw      $t4, 0($t2)
26       sll    $t4, $t4, 2
27       sw      $t4, 0($t2)
28       addi   $t2, $t2, 4
29       addi   $t3, $t3, -1
30       bne    $t3, $zero, loop2
31
32       li      $v0, 10
33       syscall
34
35       jr      $ra
```

Figura B5.5: arrays-escalar.s

Ejercicios

E5.1. Jerarquía de memoria — Caché

1. Dada la secuencia de direcciones de memoria de la tabla E5.1, rotular cada referencia como un acierto o un fallo y mostrar el contenido final de la caché para las siguientes organizaciones de caché:
 - a) Correspondencia directa con 16 bloques de una palabra.
 - b) Correspondencia directa con bloques de cuatro palabras y un tamaño total de 16 palabras.
 - c) Asociativa por conjuntos de dos bloques, bloques de una palabra y un tamaño total de 16 palabras. Se empleará el algoritmo de reemplazo LRU.
 - d) Totalmente asociativa con bloques de una palabra y un tamaño total de 16 palabras. Se empleará el algoritmo de reemplazo LRU.
 - e) Totalmente asociativa con bloques de cuatro palabras y un tamaño total de 16 palabras. Se empleará el algoritmo de reemplazo LRU.

Nota: las palabras son de 32 bits; todas las cachés están inicialmente vacías.

2. Dado un computador con las siguientes características:
 - El CPI ideal es de 3 ciclos.
 - El 40 % de las instrucciones contienen un acceso a datos.
 - La penalización de fallos es de 9 ciclos de reloj.
 - La tasa de fallos de instrucciones es del 5 % y la tasa de fallos de datos es del 10 %.
 - La duración del ciclo de reloj es de 1,5 ns.

Determinar el tiempo de CPU para un programa con 10^6 instrucciones.

3. Sea una caché con una tasa de fallos de instrucciones del 5 % y una tasa de fallos de datos del 10 %. Si la máquina tiene un CPI de 4 sin ninguna detención de memoria y la penalización de fallos es de 12 ciclos de reloj para todos los fallos, determinar el número de veces que es más rápida una máquina con una caché perfecta que nunca falle. La tercera parte de las instrucciones contienen una referencia a datos.
4. Supongamos que incrementamos el rendimiento de la máquina del ejemplo anterior duplicando su frecuencia de reloj. Supongamos también que la velocidad de la memoria principal no cambia, es decir, el tiempo absoluto para resolver un fallo de caché no varía. ¿Cuántas veces es más rápida la máquina que tiene el reloj más rápido con respecto a la que tiene el reloj más lento? El resto de los datos necesarios se toman del ejercicio anterior.
5. Dado un computador con las siguientes características:
 - El CPI ideal es de 3 ciclos.
 - El 20 % de las instrucciones contienen un acceso a datos.
 - La penalización por fallos es de 9 ciclos de reloj.
 - La tasa de fallos de instrucciones es el doble que la tasa de fallos de datos.

Tabla E5.1: Secuencia de referencias a memoria para el ejercicio 1.

Ref.	Dir. de byte	Dir. de palabra	Dir. de bloque	Acierto/fallo
1 ^a	6 = 0000 0110	1 = 0000 01		
2 ^a	17 = 0001 0001	4 = 0001 00		
3 ^a	32			
4 ^a	22			
5 ^a	82			
6 ^a	70			
7 ^a	224			
8 ^a	39			
9 ^a	45			
10 ^a	18			
11 ^a	174			
12 ^a	21			
13 ^a	25			
14 ^a	38			
15 ^a	68			

¿Qué valor de la tasa de fallos de instrucciones hace que el CPI real sea el doble que el CPI ideal?

6. Consideremos tres máquinas con diferentes configuraciones de caché:

- M1: correspondencia directa con bloques de una palabra.
- M2: correspondencia directa con bloques de cuatro palabras.
- M3: asociativa por conjuntos de dos bloques, con bloques de cuatro palabras.

Se han realizado las siguientes medidas de la tasa de fallos:

- M1: la tasa de fallos de instrucciones es de 4 % y la tasa de fallos de datos es del 8 %.
- M2: la tasa de fallos de instrucciones es del 2 % y la tasa de fallos de datos es del 5 %.
- M3: la tasa de fallos de instrucciones es del 2 % y la tasa de fallos de datos es del 4 %.

Para estas máquinas, la mitad de las instrucciones contienen una referencia a datos. Suponer que la penalización por fallo, en ciclos, es 6 + tamaño del bloque en palabras. El CPI para las tres máquinas es de 2 ciclos de reloj.

Determinar qué máquina emplea más ciclos de reloj por instrucción.

7. Las duraciones del ciclo de reloj para las máquinas del ejercicio anterior son de 10 ns para la primera y segunda máquinas, y 12 ns para la tercera. Determinar qué máquina es la más rápida y cuál la más lenta.
8. El programa MIPS de la figura E5.1 se ejecuta en una máquina con una caché (inicialmente vacía) asociativa por conjuntos. La caché tiene 2 conjuntos de 2 bloques cada uno, cada bloque es de 2 palabras (palabras de 4 bytes). La política de reemplazo es LRU. La memoria principal es de 1 KByte.

Téngase en cuenta que:

- a) Los únicos accesos a caché son las referencias a `numeros`, que está en la dirección 0. Mostrar como evoluciona el contenido de la caché al ejecutar este programa.

```

##### SEGMENTO DE DATOS #####
        .data
        ...
numeros: .space 400
        ...
##### SEGMENTO DE TEXTO #####
        .text
        ...
        li    $t0,10           # Contador
        la    $t1,numeros      # Dirección de comienzo
        li    $t2,0            # Índice
        li    $s0,0            # Acumulador
ini:     add   $t3,$t1,$t2
        lb   $t3,0($t3)
        add  $s0,$s0,$t3
        addi $t2,$t2,4
        addi $t0,$t0,-1
        bne  $t0,$zero,ini
        ...

```

Figura E5.1: Programa MIPS para el ejercicio 8.

- b) Las operaciones inmediatas duran 4 ciclos (*li*, *addi* y también *la* suponiendo que la dirección se puede cargar con una sola instrucción real); el resto: *lb* 5 ciclos, *add* 4 ciclos y *bne* 3 ciclos.
- c) La penalización por fallos es de 10 ciclos.

Si sabemos que nunca se produce un fallo de instrucción y que el procesador funciona a 200 MHz, ¿cuál es tiempo de ejecución de este programa?

9. El siguiente programa en ensamblador MIPS (ver figura E5.2) se ejecuta en una máquina M con un procesador P. El procesador P posee una memoria caché de instrucciones de primer nivel (L1) de 32 bytes con bloques de 2 palabras (8 bytes por bloque), una memoria caché de datos de primer nivel (L1) de 128 bytes con bloques de 4 palabras (16 bytes), y una memoria caché combinada para datos e instrucciones de segundo nivel (L2) con una tiempo de acceso de 20 ciclos de reloj para un bloque de 2 palabras y de 30 ciclos para un bloque de 4 palabras.

Las etiquetas fuente y destino se corresponden con las direcciones 0×10010000 y 0×10020000 , respectivamente. Se pide:

- a) Si suponemos que la memoria caché de instrucciones está inicialmente vacía, calcular la tasa de fallos de instrucciones (recordar que cada instrucción ocupa una palabra de 32 bits).
- b) Si suponemos que la memoria caché de datos está inicialmente vacía, calcular la tasa de fallos de datos.
- c) Calcular el CPI ideal para el programa.
- d) Calcular el promedio de acceso a datos por instrucciones.
- e) Calcular el tiempo de ejecución del programa si el procesador tiene una frecuencia de reloj de 450 MHz.

```
##### SEGMENTO DE TEXTO #####
      .text
      ...
      la      $t0, fuente
      la      $t1, destino
      li      $t2, 100          # Contador
bucle:  lw      $s0, 0($t0)
        addi   $s0, $s0, 1
        add    $s1, $s1, $s0
        addi   $t0, $t0, 4
        addi   $t1, $t1, 4
        addi   $t2, $t2, -1
        bne   $t2, $zero, bucle
      ...
```

Figura E5.2: Programa MIPS para el ejercicio 9.

10. La máquina M posee una memoria caché asociativa por conjuntos de n vías con 2^c conjuntos, 2^p palabras por bloque y 2^b bytes por palabra. El tamaño de la dirección es de d bits. La memoria caché sigue una estrategia de sustitución aleatoria y una política de postescritura. Determinar:
- Tamaño del bloque, tamaño de la etiqueta, tamaño del conjunto, tamaño útil de la memoria caché y tamaño total de la memoria caché en función de d , c , p y b (en bits).
 - Valor de n si la memoria caché fuese de correspondencia directa.
 - Valor de c si la memoria caché fuese totalmente asociativa.
 - Valor de p si la memoria caché tuviese bloques monopalabra.
 - Valor de c si la memoria caché tiene un tamaño útil de T_c bytes.

Nota: el tamaño útil hace referencia al tamaño ocupado por los datos.

11. Supóngase un procesador con un CPI ideal igual a 1,0 y una frecuencia de reloj de 500 MHz. El sistema dispone de una cache con un tiempo de acceso despreciable y una memoria principal con un tiempo de acceso de 200 ns, incluida la gestión de los fallos de cache. Si la tasa de fallos por instrucciones en la cache es el 5%:
- ¿Cuánto más rápida será la máquina al añadir una segunda cache que tiene un tiempo de acceso de 20 ns tanto para aciertos como para fallos y una tasa de fallos de instrucciones del 40%? Se supone que una petición a memoria principal se lanza solamente cuando se detecta un fallo en la cache de segundo nivel.
 - ¿Cuánto más rápida será la máquina si, en lugar de añadir una segunda cache, duplicamos la frecuencia de reloj del procesador manteniendo el tiempo de acceso a memoria principal en 200 ns?

Nota: las referencias a datos no producen fallos.

12. El siguiente programa en ensamblador (ver figura E5.3) se ejecuta en una máquina M con un procesador P. La máquina M posee una memoria caché con las siguientes características:

```

##### SEGMENTO DE DATOS #####
                .data
                ...
c:              .space 2
v:              .space 6
                ...
##### SEGMENTO DE TEXTO #####
                .text
                ...
I0              la      $s0,v
I1              li      $t0,3
I2  etiq:      lb      $s1,5($s0)
I3              addi   $s0,$s0,-1
I4              addi   $t0,$t0,-1
I5              bne    $t0,$zero,etiq
                ...

```

Figura E5.3: Programa MIPS para el ejercicio 12.

- Memoria caché de correspondencia directa.
- Memoria caché combinada para datos e instrucciones.
- Tamaño útil de 16 bytes.
- Bloques de 1 palabra y palabras de 4 bytes.

La máquina M posee una memoria principal de 256 bytes que tiene un tiempo de acceso de 20 ciclos de reloj para bloques de 1 palabra. Además, téngase en cuenta que:

- Cada instrucción ocupa 1 palabra.
- Los datos del programa están almacenados en memoria principal justo a partir de la dirección absoluta 0 y el código justo a continuación de los datos, sin dejar ni un sólo byte de hueco.
- El tiempo de ejecución ideal de cada instrucción es la vista en la implementación multiciclo.
- La memoria caché está inicialmente vacía.

Se pide:

- a) Realizar un seguimiento exhaustivo del contenido de la memoria caché a lo largo de la ejecución de este programa.
- b) Calcular la tasa de fallos de instrucciones y la tasa de fallos de datos para este programa.
- c) Calcular el CPI ideal para este programa.
- d) Calcular el tiempo de ejecución de este programa si el procesador tiene una frecuencia de reloj de 1GHz.

Tabla E5.2: Tabla con la solución para el ejercicio 1(a).

Ref.	Dir. de byte	Dir. de palabra	Dir. de bloque	Pos. Caché	Acierto/fallo
1 ^a	6 = 0000 0110	1 = 0000 01	1	1	F
2 ^a	17 = 0001 0001	4 = 0001 00	4	4	F
3 ^a	32	8	8	8	F
4 ^a	22	5	5	5	F
5 ^a	82	20	20	4	F+R
6 ^a	70	17	17	1	F+R
7 ^a	224	56	56	8	F+R
8 ^a	39	9	9	9	F
9 ^a	45	11	11	11	F
10 ^a	18	4	4	4	F+R
11 ^a	174	43	43	11	F+R
12 ^a	21	5	5	5	A
13 ^a	25	6	6	6	F
14 ^a	38	9	9	9	A
15 ^a	68	17	17	1	A

E5.2. Solución a ejercicios seleccionados

1. Dada la secuencia de direcciones de memoria de la tabla E5.1, rotular cada referencia como un acierto o un fallo y mostrar el contenido final de la caché para las siguientes organizaciones de caché:

- a) Correspondencia directa con 16 bloques de una palabra.

La solución a este apartado se puede ver en la tabla E5.2. Como podemos ver, a la tabla que nos pide el ejercicio que rellenemos, se le ha añadido una columna más para reflejar la posición de caché correspondiente a cada referencia. En el caso de los aciertos y los fallos, se indica con una «A» un acierto, con una «F» un fallo y, en este último caso, con una «R» adicional aquellos fallos que suponen expulsar de caché lo que hay en la posición correspondiente.

- b) Correspondencia directa con bloques de cuatro palabras y un tamaño total de 16 palabras.

La tabla E5.3 muestra la solución a este apartado.

- c) Asociativa por conjuntos de dos bloques, bloques de una palabra y un tamaño total de 16 palabras. Se empleará el algoritmo de reemplazo LRU.

La solución se puede ver en la tabla E5.4. En este caso, hay un fallo de caché que produce un reemplazo, el cual hemos indicado con una «R» a la que le sigue, entre paréntesis, el número del bloque que se expulsa de caché.

- d) Totalmente asociativa con bloques de una palabra y un tamaño total de 16 palabras. Se empleará el algoritmo de reemplazo LRU.

La solución viene dada por la tabla E5.5.

- e) Totalmente asociativa con bloques de cuatro palabras y un tamaño total de 16 palabras. Se empleará el algoritmo de reemplazo LRU.

La solución se puede ver en la tabla E5.6. Al igual que antes, las R's indican aquellos casos en los que se producen reemplazos y entre paréntesis se indica el bloque que se reemplaza.

Nota: las palabras son de 32 bits; todas las cachés están inicialmente vacías.

5. Dado un computador con las siguientes características:

Tabla E5.3: Tabla con la solución para el ejercicio 1(b).

Ref.	Dir. de byte	Dir. de palabra	Dir. de bloque	Pos. Caché	Acierto/fallo
1 ^a	6	1	0	0	F
2 ^a	16	4	1	1	F
3 ^a	32	8	2	2	F
4 ^a	22	5	1	1	A
5 ^a	82	20	5	1	F+R
6 ^a	70	17	4	0	F+R
7 ^a	224	56	14	2	F+R
8 ^a	39	9	2	2	F+R
9 ^a	45	11	2	2	A
10 ^a	18	4	1	1	F+R
11 ^a	174	43	10	2	F+R
12 ^a	21	5	1	1	A
13 ^a	25	6	1	1	A
14 ^a	38	9	2	2	F+R
15 ^a	68	17	4	0	A

Tabla E5.4: Tabla con la solución para el ejercicio 1(c).

Ref.	Dir. de byte	Dir. de palabra	Dir. de bloque	Pos. Caché	Acierto/fallo
1 ^a	6	1	1	1	F
2 ^a	16	4	4	4	F
3 ^a	32	8	8	0	F
4 ^a	22	5	5	5	F
5 ^a	82	20	20	4	F
6 ^a	70	17	17	1	F
7 ^a	224	56	56	0	F
8 ^a	39	9	9	1	F+R(1)
9 ^a	45	11	11	3	F
10 ^a	18	4	4	4	A
11 ^a	174	43	43	3	F
12 ^a	21	5	5	5	A
13 ^a	25	6	6	6	F
14 ^a	38	9	9	1	A
15 ^a	68	17	17	1	A

Tabla E5.5: Tabla con la solución para el ejercicio 1(d).

Ref.	Dir. de byte	Dir. de palabra	Dir. de bloque	Pos. Caché	Acierto/fallo
1 ^a	6	1	1	0	F
2 ^a	16	4	4	0	F
3 ^a	32	8	8	0	F
4 ^a	22	5	5	0	F
5 ^a	82	20	20	0	F
6 ^a	70	17	17	0	F
7 ^a	224	56	56	0	F
8 ^a	39	9	9	0	F
9 ^a	45	11	11	0	F
10 ^a	18	4	4	0	A
11 ^a	174	43	43	0	F
12 ^a	21	5	5	0	A
13 ^a	25	6	6	0	F
14 ^a	38	9	9	0	A
15 ^a	68	17	17	0	A

Tabla E5.6: Tabla con la solución para el ejercicio 1(e).

Ref.	Dir. de byte	Dir. de palabra	Dir. de bloque	Pos. Caché	Acierto/fallo
	6	1	0	0	F
	16	4	1	0	F
	32	8	2	0	F
	22	5	1	0	A
	82	20	5	0	F
	70	17	4	0	F+R(0)
	224	56	14	0	F+R(2)
	39	9	2	0	F+R(1)
	45	11	2	0	A
	18	4	1	0	F+R(5)
	174	43	10	0	F+R(4)
	21	5	1	0	A
	25	6	1	0	A
	38	9	2	0	A
	68	17	4	0	F+R(14)

- El CPI ideal es de 3 ciclos.
- El 20 % de las instrucciones contienen un acceso a datos.
- La penalización por fallos es de 9 ciclos de reloj.
- La tasa de fallos de instrucciones es el doble que la tasa de fallos de datos.

¿Qué valor de la tasa de fallos de instrucciones hace que el CPI real sea el doble que el CPI ideal?

Solución:

Sabemos que el CPI_{real} viene dado por la siguiente expresión:

$$CPI_{real} = CPI_{ideal} + TF_{inst} \times PF + D/I \times TF_{dat} \times PF$$

Sustituyendo en esta expresión los términos cuyos datos aparecen en el enunciado del ejercicio, obtenemos lo siguiente:

$$2 \times 3 = 3 + TF_{inst} \times 9 + 0,20 \times \frac{TF_{inst}}{2} \times 9$$

Despejando TF_{inst} , obtenemos:

$$\begin{aligned} 6 &= 3 + TF_{inst} \times 9 + 1,8 \times \frac{TF_{inst}}{2} \\ 6 &= TF_{inst} \times 18 + 1,8 \times TF_{inst} \\ TF_{inst} &= \frac{6}{19,8} = 0,3030 \end{aligned}$$

Es decir, una tasa de fallos por instrucción del 30,3 % hace que el CPI real sea el doble del CPI ideal.

6. Consideremos tres máquinas con diferentes configuraciones de caché:

- M1: correspondencia directa con bloques de una palabra.
- M2: correspondencia directa con bloques de cuatro palabras.
- M3: asociativa por conjuntos de dos bloques, con bloques de cuatro palabras.

Se han realizado las siguientes medidas de la tasa de fallos:

- M1: la tasa de fallos de instrucciones es de 4 % y la tasa de fallos de datos es del 8 %.
- M2: la tasa de fallos de instrucciones es del 2 % y la tasa de fallos de datos es del 5 %.
- M3: la tasa de fallos de instrucciones es del 2 % y la tasa de fallos de datos es del 4 %.

Para estas máquinas, la mitad de las instrucciones contienen una referencia a datos. Suponer que la penalización por fallo, en ciclos, es 6 + tamaño del bloque en palabras. El CPI para las tres máquinas es de 2 ciclos de reloj.

Determinar qué máquina emplea más ciclos de reloj por instrucción.

Solución:

El CPI_{real} viene dado por la siguiente expresión:

$$CPI_{real}(Mx) = CPI_{ideal} + TF_{inst}(Mx) \times PF(Mx) + D/I \times TF_{dat}(Mx) \times PF(Mx)$$

El CPI_{ideal} y la tasa de acceso de datos (D/I) son iguales en las tres máquinas (2 ciclos y 50 %, respectivamente).

La penalización por fallo de cada máquina es la siguiente:

$$\begin{aligned} PF(M1) &= (6 + 1) \text{ ciclos} = 7 \text{ ciclos} \\ PF(M2) &= (6 + 4) \text{ ciclos} = 10 \text{ ciclos} \\ PF(M3) &= (6 + 4) \text{ ciclos} = 10 \text{ ciclos} \end{aligned}$$

Las tasas de fallo de instrucciones y datos de cada máquina ($TF_{inst}(M_x)$ y $TF_{datos}(M_x)$) aparecen directamente en el enunciado, con lo que:

$$\begin{aligned} CPI_{real}(M1) &= 2 + 0,04 \times 7 + 0,50 \times 0,08 \times 7 = 2,56 \text{ ciclos} \\ CPI_{real}(M2) &= 2 + 0,02 \times 10 + 0,50 \times 0,05 \times 10 = 2,45 \text{ ciclos} \\ CPI_{real}(M3) &= 2 + 0,02 \times 10 + 0,50 \times 0,04 \times 10 = 2,40 \text{ ciclos} \end{aligned}$$

Por lo tanto, la máquina que emplea más ciclos de reloj es la primera máquina.

8. El programa MIPS de la figura E5.1 se ejecuta en una máquina con una caché (inicialmente vacía) asociativa por conjuntos. La caché tiene 2 conjuntos de 2 bloques cada uno, cada bloque es de 2 palabras (palabras de 4 bytes). La política de reemplazo es LRU. La memoria principal es de 1 KByte.

Téngase en cuenta que:

- Los únicos accesos a caché son las referencias a `numeros`, que está en la dirección 0. Mostrar como evoluciona el contenido de la caché al ejecutar este programa.
- Las operaciones inmediatas duran 4 ciclos (`li`, `addi` y también `la` suponiendo que la dirección se puede cargar con una sola instrucción real); el resto: `lb` 5 ciclos, `add` 4 ciclos y `bne` 3 ciclos.
- La penalización por fallos es de 10 ciclos.

Si sabemos que nunca se produce un fallo de instrucción y que el procesador funciona a 200 MHz, ¿cuál es tiempo de ejecución de este programa?

Solución:

Este ejercicio se puede solucionar de varias maneras. Una es calcular el CPI_{real} y multiplicarlo por el número de instrucciones ejecutadas y el tiempo de ciclo. Otra, más sencilla, es simplemente calcular los ciclos de CPU que consumen las instrucciones y a dicha cantidad sumarles los ciclos de bloqueo de memoria que se pueden producir por los fallos de caché al ejecutar la instrucción `lb`.

Aplicando la segunda técnica, vemos que antes del bucle se ejecutan 3 instrucciones `li` y una instrucción `la` que consumen 4 ciclos cada una. El bucle se ejecuta 10 veces ya que inicialmente `$t0` vale 10, en cada pasada del bucle se reduce su valor en 1 y se sale del bucle cuando `$t0` es 0. En cada pasada del bucle se ejecuta dos instrucciones `add` y dos instrucciones `addi`, que consumen 4 ciclos cada una, una instrucción `lb` que necesita 5 ciclos de reloj y una instrucción `bne` que necesita 3 ciclos. Por tanto, el número de ciclos consumidos durante la ejecución de las instrucciones es:

$$\text{Ciclos por instrucciones} = 4 + 4 + 4 + 4 + 10 \times (4 + 5 + 4 + 4 + 4 + 3) = 256$$

Veamos ahora los ciclos que añaden los fallos de caché. La única instrucción que accede a memoria es `lb`. Esta instrucción accede a la dirección de memoria contenida en `$t3` cuyo valor se calcula como la suma de `$t1` y `$t2`. `$t1` siempre vale 0 (que es la dirección de memoria correspondiente a la etiqueta

numeros), mientras que \$t2 tiene inicialmente un valor 0 y se incrementa en 4 en cada pasada del bucle. Por lo tanto, la secuencia de referencias a memoria es: 0, 4, 8, 12, 16, 20, 24, 28, 32 y 36.

Veamos qué fallos y aciertos de caché produce la secuencia de accesos a memoria anterior. Según el enunciado, nuestra caché es asociativa con 2 conjuntos y 2 vías, es decir, hay 2 posiciones de caché y en cada una caben 2 bloques, teniendo cada bloque un tamaño de 2 palabras y siendo las palabras de 4 bytes. Por otro lado, tenemos que la memoria física es de 1 KB, lo que supone que las direcciones físicas son de 10 bits. Por lo tanto, cada dirección física se descompone en los siguientes campos:

6 bits	1 bit	1 bit	2 bits
Etiqueta	Índice	Desp. palabra	Desp. byte

donde los 7 bits más significativos nos dan el bloque en el que se encuentra un determinado byte o palabra. Nuestra caché, por su parte, tiene el siguiente aspecto:

		Vía 0					Vía 1				
		V	M	U	Etiqueta (6 bits)	Bloque (2 palabras)	V	M	U	Etiqueta (6 bits)	Bloque (2 palabras)
0		0	-	-	-	-	0	-	-	-	-
1		0	-	-	-	-	0	-	-	-	-

Aunque en este ejercicio la política de escritura no es importante porque en el programa no hay escrituras, hemos supuesto que la caché es de postescritura, de ahí el bit M. El bit de uso U se debe a que la política de reemplazo es LRU.

Veamos cómo evoluciona la caché:

- Dirección 0: los 10 bits de esta dirección son 0, por lo que la dirección pertenece al bloque 0 que se asigna al conjunto 0 de la caché, que queda de la siguiente manera:

		Vía 0					Vía 1				
		V	M	U	Etiqueta (6 bits)	Bloque (2 palabras)	V	M	U	Etiqueta (6 bits)	Bloque (2 palabras)
0		1	0	1	0	B[0]	0	-	-	-	-
1		0	-	-	-	-	0	-	-	-	-

Ya que la caché está inicialmente vacía, este acceso produce un fallo de caché.

- Dirección 4: los 10 bits de esta dirección son 000000 0 1 00 (hemos separado por espacios los bits de cada campo), que corresponde también al bloque 0. Como este bloque está ya en caché, el acceso a esta dirección produce un acierto
- Dirección 8: los 10 bits son ahora 000000 1 0 00 y la dirección pertenece al bloque 1, que no está en caché, por lo que este acceso produce un fallo. La caché queda así:

		Vía 0					Vía 1				
		V	M	U	Etiqueta (6 bits)	Bloque (2 palabras)	V	M	U	Etiqueta (6 bits)	Bloque (2 palabras)
0		1	0	1	0	B[0]	0	-	-	-	-
1		1	0	1	0	B[1]	0	-	-	-	-

- Dirección 12: sus 10 bits son 000000 1 1 00, por lo que esta dirección pertenece al bloque 1 que ya está en caché, produciendo un acierto.
- Y así sucesivamente. Al final en la caché quedan los últimos 4 bloques leídos, quedando la caché de la siguiente manera:

Vía 0				
	V	M	U	Bloque (2 palabras)
0	1	0	1	B[4]
1	1	0	0	B[1]

Vía 1				
	V	M	U	Bloque (2 palabras)
	1	0	0	B[2]
	1	0	1	B[3]

Observe que el bit de uso se pone a 0 o a 1 de en cada bloque de manera adecuada para reflejar en todo momento qué bloque se ha usado más recientemente dentro de un conjunto y así poder aplicar correctamente el algoritmo de reemplazo LRU si llega el caso.

Como vemos, cada 2 accesos a memoria se produce un acierto y un fallo. Por lo tanto, hay 5 fallos en total que añaden $5 \times 10 = 50$ ciclos a la ejecución del programa, ya que la penalización por fallo es de 10 ciclos.

En total, teniendo en cuenta los bloqueos de memoria, el programa tarda $256 + 50 = 306$ ciclos. Ya que el tiempo de ciclo son $TC = \frac{1}{200 \cdot 10^6}$ segundos o, lo que es lo mismo, 5 nanosegundos, el tiempo de ejecución del programa es:

$$T_{\text{ejecución}} = 306 \times 5 = 1530 \text{ nanosegundos.}$$

12. El siguiente programa en ensamblador (ver figura E5.3) se ejecuta en una máquina M con un procesador P. La máquina M posee una memoria caché con las siguientes características:

- Memoria caché de correspondencia directa.
- Memoria caché combinada para datos e instrucciones.
- Tamaño útil de 16 bytes.
- Bloques de 1 palabra y palabras de 4 bytes.

La máquina M posee una memoria principal de 256 bytes que tiene un tiempo de acceso de 20 ciclos de reloj para bloques de 1 palabra. Además, téngase en cuenta que:

- Cada instrucción ocupa 1 palabra.
- Los datos del programa están almacenados en memoria principal justo a partir de la dirección absoluta 0 y el código justo a continuación de los datos, sin dejar ni un sólo byte de hueco.
- El tiempo de ejecución ideal de cada instrucción es la vista en la implementación multiciclo.
- La memoria caché está inicialmente vacía.

Se pide:

a) Realizar un seguimiento exhaustivo del contenido de la memoria caché a lo largo de la ejecución de este programa.

Solución:

De los datos dados en el enunciado se deduce que las direcciones se dividirán de la siguiente forma:

4 bits	2 bits	2 bits
Etiqueta	Índice	Desp. byte

ya que las direcciones físicas son de 8 bits (la memoria tiene en total 256 bytes) y la caché tiene 4 posiciones de 1 palabra cada una (lo que hace que tenga un tamaño útil total de $4 \times 4 = 16$ bytes). Como nos dice el enunciado, los datos del programa se encuentran a partir de la dirección 0 de memoria, por lo que la etiqueta c representa a la dirección de memoria 0 y v a la dirección de

memoria 2. Como el código comienza justo a continuación, la primera instrucción se encuentra en la dirección 8 de memoria.

Por otro lado, tenemos dos pseudoinstrucciones, l_a y l_i . La dos tienen que cargar en el registro correspondiente un valor que cabe en 16 bits, por lo que las dos se traducen en una única instrucción real.

Los accesos realizados son, por tanto, los siguientes:

Línea	Tipo	Dirección	Índice	Etiqueta	Acierto
I0	Ifetch	0x08	2	0	Fallo
I1	Ifetch	0x0c	3	0	Fallo
I2	Ifetch	0x10	0	1	Fallo
I2	Load	0x07	1	0	Fallo
I3	Ifetch	0x14	1	1	Fallo
I4	Ifetch	0x18	2	1	Fallo
I5	Ifetch	0x1c	3	1	Fallo
I2	Ifetch	0x10	0	1	Acierto
I2	Load	0x06	1	0	Fallo
I3	Ifetch	0x14	1	1	Fallo
I4	Ifetch	0x18	2	1	Acierto
I5	Ifetch	0x1c	3	1	Acierto
I2	Ifetch	0x10	0	1	Acierto
I2	Load	0x05	1	0	Fallo
I3	Ifetch	0x14	1	1	Fallo
I4	Ifetch	0x18	2	1	Acierto
I5	Ifetch	0x1c	3	1	Acierto

- b) Calcular la tasa de fallos de instrucciones y la tasa de fallos de datos para este programa.

Solución:

La tasa de fallos de instrucciones será de $\frac{8}{14} = 57,14\%$, y la de datos será de $\frac{3}{3} = 100\%$.

- c) Calcular el CPI ideal para este programa.

Solución:

En este programa se ejecutan 14 instrucciones, de las que 3 son cargas, 3 son saltos y el resto son aritmético-lógicas. Por tanto:

$$CPI_{ideal} = \frac{3 \times 5 + 3 \times 3 + 8 \times 4}{14} = 4$$

- d) Calcular el tiempo de ejecución de este programa si el procesador tiene una frecuencia de reloj de 1GHz.

Solución:

El tiempo de ciclo es de $\frac{1}{10^9 \text{ Hz}} = 1 \text{ ns}$. La penalización por fallos es de 20 ciclos (tiempo de acceso a memoria para un bloque de 1 palabra). Dado que se producen 11 fallos de caché, el procesador estará parado esperando a la memoria durante $11 \times 20 = 220$ ciclos. Por tanto:

$$T = (CPI_{ideal} \times N_{inst} + \text{ciclos_detención_memoria}) \times TC = 4 \times 14 + 220 = 276 \text{ ns}$$