5

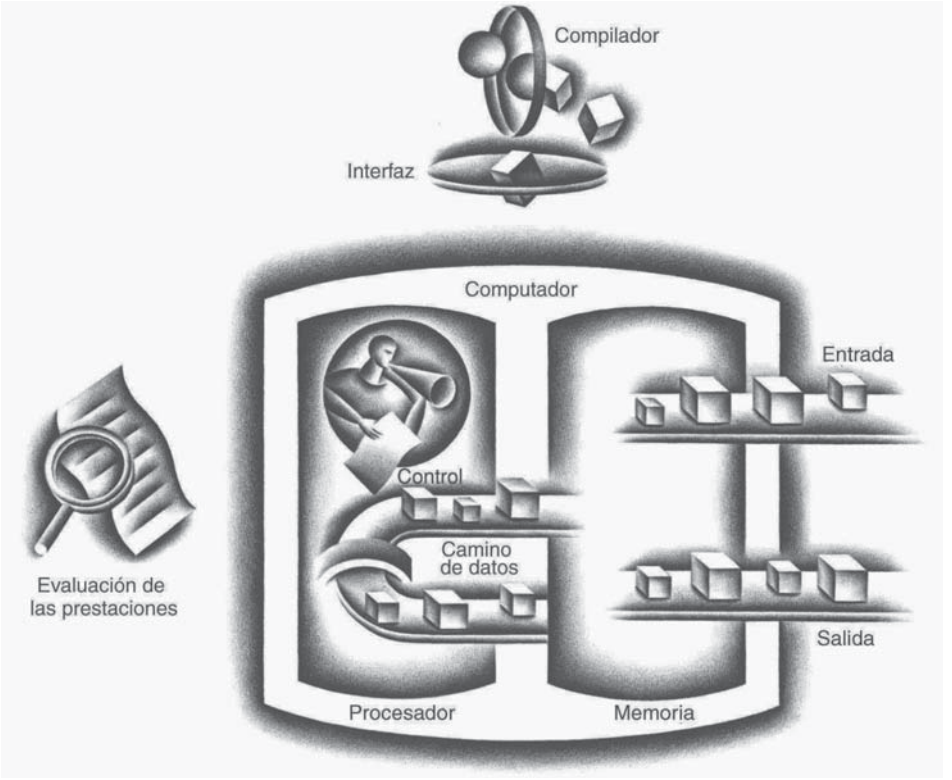
Idealmente, uno desearía una capacidad de memoria muy grande, de forma tal que cualquier… palabra estuviera disponible inmediatamente… Nos vemos…forzados a admitir la posibilidad de construir una jerarquía de niveles de memorias, cada uno de los cuales tiene mayor capacidad que el precedente, pero al que se accede menos rápidamente.

A. W. Burks, H. H. Goldstine y J. von Neumann
Discusión preliminar del diseño lógico de un instrumento de computación electrónico, 1946

Grande y rápida: aprovechamiento de la jerarquía de memoria

Nota importante: En la presente edición en castellano, los contenidos del CD incluido en la edición original (en inglés) son accesibles a través de la página web www.reverte.com/microsites/pattersonhennessy. Aunque en la presente edición no se proporciona un CD-ROM físico, a lo largo de todo el texto se menciona el CD y se utiliza el icono que lo representa para hacer referencia a su contenido.

Los cinco componentes clásicos del computador



5.1 Introducción

Desde los primeros días de la Informática, los programadores han deseado cantidades ilimitadas de memoria rápida. Los temas que abordaremos en este capítulo ayudan a los programadores a tener la impresión de que disponen de una memoria rápida e ilimitada. Antes de ver cómo se genera realmente esta impresión, consideremos una simple analogía que ilustra los principios y mecanismos básicos que utilizaremos.

Suponga que usted está escribiendo un informe sobre los acontecimientos históricos más importantes en la evolución del hardware del computador. Usted se encuentra sentado en la mesa de una biblioteca junto a varios libros que ha sacado de las estanterías y los está hojeando. Encuentra que varios de los computadores más importantes sobre los que es necesario escribir están en los libros que dispone, pero no encuentra nada sobre el EDSAC. Por lo tanto, usted vuelve a las estanterías a buscar otro libro. Por fin encuentra uno sobre los primeros computadores británicos que describe el EDSAC. Una vez que dispone de una buena selección de libros sobre su mesa, existe una alta probabilidad de que en ellos encuentre muchos de los temas que necesita cubrir, y por ello usted puede estar la mayor parte del tiempo utilizando los libros de su mesa sin tener que volver a las estanterías. Tener varios libros en su mesa permite ahorrar tiempo, en comparación con la situación en la que se consulta un solo libro, se vuelve a las estanterías para colocarlo en su sitio y se escoge otro libro de consulta.

El mismo principio permite crear la impresión de que se dispone de una memoria grande a la que se puede acceder tan rápidamente como a una memoria muy pequeña. De la misma forma que usted no necesitaba acceder al mismo tiempo a todos los libros de la biblioteca con la misma probabilidad, un programa no accede a la vez a todo su código o datos con la misma probabilidad. Si no fuera así, sería imposible realizar en un computador de forma rápida la mayoría de los accesos a una memoria que fuera grande, de la misma forma que sería imposible para usted poner todos los libros de la biblioteca sobre su mesa y además encontrar rápidamente lo que busca.

Localidad temporal: principio que establece que si se accede a la dirección de memoria de un dato, pronto se accederá de nuevo a esa dirección.

Localidad espacial: principio de localidad que establece que si se accede a la dirección de memoria de un dato es accedida, pronto se accederá a las direcciones de memoria que se encuentran próximas a ella.

Este *principio de localidad* subyace tanto la manera en que usted hace una búsqueda en la biblioteca como en la forma en que funcionan los programas. El principio de localidad establece que los programas acceden a una parte relativamente pequeña del espacio de direcciones en un determinado instante, de la misma forma que usted accede a una cantidad muy pequeña de libros del total de libros de la biblioteca. Existen dos tipos distintos de localidad:

- **Localidad temporal** (localidad en el tiempo): Si se accede a una dirección de memoria, esta dirección será utilizada de nuevo en un corto intervalo de tiempo. Si usted ha llevado recientemente un libro a su mesa para consultarlo, probablemente pronto necesitará consultarlo de nuevo.
- **Localidad espacial** (localidad en el espacio): Si se accede a una dirección de memoria, las direcciones de memoria que se encuentran próximas a ella serán utilizadas en un corto intervalo de tiempo. Por ejemplo, cuando usted sacó de

la estantería el libro sobre los primeros computadores ingleses para encontrar información sobre EDSAC, quizás vio cerca de él otro libro que trataba sobre los primeros computadores mecánicos; así que también se lo llevó, y posteriormente encontró en ese libro alguna información que le fue útil. Los libros que tratan un mismo tema se colocan juntos en las estanterías de las bibliotecas para incrementar la localidad espacial. Más adelante, en este capítulo, veremos cómo se utiliza la localidad espacial en las jerarquías de memoria.

Así como los accesos a los libros que están encima de la mesa tienen localidad de forma natural, la localidad en los programas surge de sus estructuras simples y naturales. Por ejemplo, la mayoría de los programa contienen lazos, por lo que probablemente se accederá repetitivamente a instrucciones y datos, mostrando una gran cantidad de localidad temporal. Puesto que el acceso a las instrucciones es secuencial, los programas muestran también una alta localidad espacial. Los accesos a datos también tienen una localidad espacial natural. Por ejemplo, los accesos a elementos de una matriz o una tupla experimentarán por naturaleza un alto grado de localidad espacial.

Nosotros aprovechamos el principio de localidad implementando la memoria de un computador como una **jerarquía de memoria**, que está formada por varios niveles de memoria con diferentes tiempos de accesos y capacidades. Las memorias más rápidas son de mayor coste económico por bit que las memorias más lentas, y por ello, son más pequeñas.

Jerarquía de memoria: estructura que utiliza varios niveles de memoria; a medida que aumenta la distancia desde la CPU, el tamaño de las memorias y el tiempo aumentan.

Hoy en día, en la construcción de jerarquías de memorias se usan principalmente tres tecnologías. La memoria principal es implementada con DRAM (memoria dinámica de acceso aleatorio), mientras que los niveles más cercanos al procesador (caches) utilizan SRAM (memoria estática de acceso aleatorio). La DRAM es menos costosa por bit que la SRAM, aunque es considerablemente más lenta. La diferencia en precio se debe a que la DRAM utiliza bastante menos área de chip por bit de memoria, y por ello la DRAM dispone de mayor capacidad de memoria para la misma cantidad de silicio; la diferencia en tiempo de acceso se debe a distintos factores que se describen en la sección C.9 del apéndice C. La tercera tecnología que se usa para implementar el mayor y más lento nivel de la jerarquía, es el disco magnético. (Las memorias Flash se utilizan en muchos dispositivos empotrados en lugar de los discos; véase sección 6.4). El tiempo de acceso y precio por bit de estas tecnologías varían extensamente, como se puede ver en la siguiente tabla, en la que se han utilizado valores típicos de 2008.

Tecnología de memoria	Tiempo de acceso típico	Dólares por GB en 2004
SRAM	0.5–2.5 ns	$2000–$5000
DRAM	50–70 ns	$20–$75
Disco magnético	5 000 000–20 000 000 ns	$0.20–$2

Debido a estas diferencias en coste económico y tiempo de acceso, es beneficioso construir memorias como una jerarquía de niveles. La figura 5.1 muestra que la memoria más rápida está cerca del procesador y la más lenta y menos cara está debajo de ella. El objetivo es presentar al usuario tanta memoria como esté disponible en la tecnología más barata, a la vez que se pueda acceder a la velocidad de la memoria más rápida.

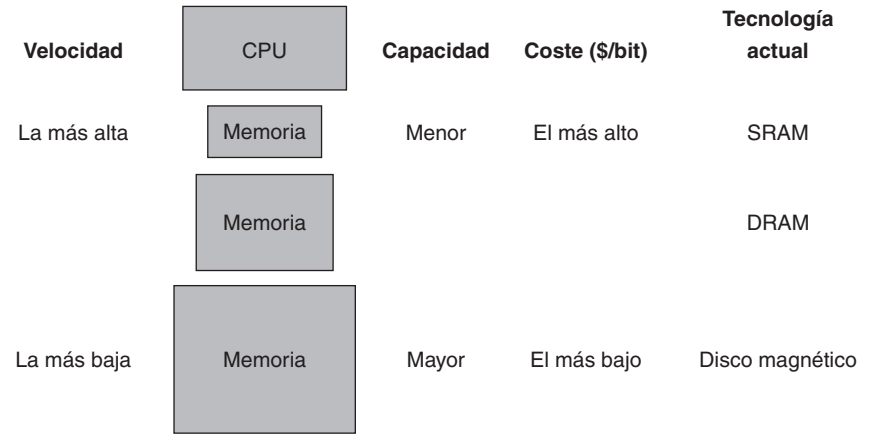


FIGURA 5.1 Estructura básica de una jerarquía de memoria. Al implementar el sistema de memoria como una jerarquía, el usuario tiene la impresión de que existe una memoria que es tan grande como la del nivel más grande de la jerarquía, pero que puede ser accedida como si estuviera construida con la memoria más rápida. Las memorias Flash han reemplazado a los discos en muchos dispositivos empotrados, y pueden llevar a un nuevo nivel en la jerarquía de almacenamiento en servidores y computadores de sobremesa; véase sección 6.4.

El sistema de memoria está organizado como una jerarquía: un nivel más cercano al procesador corresponde generalmente a un subconjunto de cualquiera de los niveles más alejados, y todos los datos están almacenados en el nivel más bajo. Siguiendo nuestra analogía, los libros con los que está trabajando y que están encima de la mesa forman un subconjunto de los libros de la biblioteca, que a su vez forman un subconjunto de todas las bibliotecas del campus. Además, a medida que nos alejamos del procesador, los niveles requieren progresivamente mayor tiempo de acceso, al igual que lo que nos podríamos encontrar en una jerarquía de bibliotecas del campus.

Bloque: unidad mínima de información que puede estar o no estar presente en una jerarquía de dos niveles.

Frecuencia de aciertos: fracción de accesos a memoria que se encontraron en un nivel determinado de la jerarquía de memoria.

Frecuencia de fallos: fracción de accesos a memoria que no se encontraron en un determinado nivel de la jerarquía de memoria.

Una jerarquía de memoria puede constar de varios niveles, pero en un instante los datos sólo se transfieren entre dos niveles adyacentes, de forma tal que podemos centrar nuestra atención en solamente dos niveles. El nivel superior —el más cercano al procesador— es más pequeño y rápido que el nivel inferior ya que utiliza la tecnología más cara. La figura 5.2 muestra que la unidad mínima de información que puede estar o no estar presente en una jerarquía de dos niveles se llama **bloque** o *línea*; en nuestra analogía de la biblioteca, un bloque de información es un libro.

Si los datos que requiere el procesador se encuentran en algún bloque del nivel superior, se dice que esto es un *acierto* (análogo a encontrar la información en uno de los libros de la mesa). Si los datos no se encuentran en el nivel superior, la petición se denomina *fallo*. En este caso, se accede al nivel inferior de la jerarquía para recuperar el bloque que contiene los datos requeridos. (Continuando con nuestra analogía, usted se dirige desde su mesa a las estanterías para encontrar el libro deseado). La **frecuencia de aciertos** o *tasa de aciertos* es la fracción de accesos a memoria cuyos datos se encuentran en el nivel superior; a menudo se usa como medida de prestaciones de la jerarquía de memoria. La **frecuencia de fallos** (1 – frecuencia de aciertos) es la fracción de los accesos a memoria no encontrados en el nivel superior.

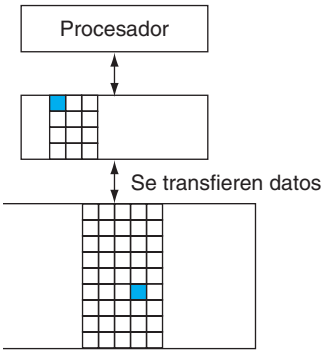


FIGURA 5.2 Se puede considerar que cada par de niveles de la jerarquía de memoria tiene un nivel superior y un nivel inferior. Dentro de cada nivel, la unidad de información que está o no presente se denomina *bloque*. Cuando algo se transmite entre niveles, normalmente se transfiere un bloque completo.

Como las prestaciones son el motivo más importante para establecer una jerarquía de memoria, es importante considerar el tiempo dedicado a aciertos y fallos. El **tiempo de acierto** es el tiempo necesario para acceder al nivel superior de la jerarquía de memoria, que incluye el tiempo requerido para determinar si el acceso corresponde a un acierto o a un fallo (esto es, el tiempo necesario para ver si un libro está encima de la mesa). La **penalización por fallo** es el tiempo necesario para reemplazar un bloque en el nivel superior por el correspondiente bloque del nivel inferior, al que hay que añadir el tiempo que se tarda en suministrar este bloque al procesador (o, el tiempo para escoger otro libro de las estanterías y depositarlo sobre la mesa). Como el nivel superior es más pequeño y está construido con componentes de memoria más rápidos, el tiempo de acierto será mucho menor que el tiempo de acceso al siguiente nivel de la jerarquía, el cual es la componente temporal más importante de la penalización por fallo. (Se tarda mucho menos en inspeccionar los libros de la mesa que en levantarse de la silla y escoger un nuevo libro de las estanterías).

Tiempo de acierto: tiempo necesario para acceder a un determinado nivel de la jerarquía de memoria, incluido el tiempo necesario para determinar si el acceso corresponde a un acierto o a un fallo.

Penalización por fallo: tiempo necesario para ir a buscar un bloque a un determinado nivel de la jerarquía de memoria partiendo desde otro nivel inferior, incluido el tiempo necesario para acceder al bloque, transmitirlo de un nivel al otro y guardarlo en el nivel que experimentó el fallo.

Como veremos en este capítulo, los conceptos utilizados para construir sistemas de memoria afectan a muchos otros aspectos de un computador, entre ellos la gestión de la memoria y la E/S por parte del sistema operativo, la generación de código por parte de los compiladores, e incluso la forma en que las aplicaciones utilizan los recursos del computador. Por supuesto, como todos los programas dedican mucho tiempo a los accesos a memoria, el sistema de memoria es necesariamente un factor de gran importancia en la determinación de las prestaciones. La dependencia de las jerarquías de memoria para conseguir buenas prestaciones ha supuesto que los programadores, quienes solían pensar que la memoria es un dispositivo de acceso aleatorio de un solo nivel, ahora necesiten entender cómo funcionan las jerarquías de memoria para obtener buenas prestaciones. Posteriormente, utilizaremos dos ejemplos, figura 5.18 en la página 490, para mostrar la importancia de esta compresión.

Puesto que los sistemas de memoria son tan importantes para obtener buenas prestaciones, los diseñadores de computadores dedicaron muchos esfuerzos a estos sistemas y desarrollaron sofisticados mecanismos para mejorar las prestaciones del sistema de memoria. En este capítulo veremos las ideas conceptuales más impor-

tantes, aunque hemos utilizado muchas simplificaciones y abstracciones para que la materia sea manejable tanto en longitud como complejidad.

IDEA clave

Los programas muestran tanto localidad temporal, que es la tendencia a reutilizar los datos que previamente han sido accedidos, como localidad espacial, que es la tendencia a acceder datos que se localizan en posiciones de memoria cercanas a otras que han sido referenciadas recientemente. Las jerarquías de memoria aprovechan la localidad temporal guardando los datos recientemente accedidos en un lugar próximo al procesador. Las jerarquías de memoria aprovechan la localidad espacial transmitiendo hacia niveles superiores de la jerarquía bloques de varias palabras que se almacenan en posiciones contiguas de memoria.

La figura 5.3 muestra que una jerarquía de memoria utiliza cerca del procesador tecnologías de memoria más pequeñas y más rápidas. De este modo, los datos que se encuentran en el nivel más elevado de la jerarquía pueden ser procesados rápidamente. Los datos que no se encuentran en el nivel más alto, deben ser buscados en niveles inferiores de la jerarquía, los cuales son más grandes y más lentos. Si la frecuencia de aciertos en el nivel más alto es suficientemente elevada, la jerarquía de memoria se caracteriza por un tiempo de acceso que es próximo al tiempo de acceso del nivel más alto (y más rápido), y tiene una capacidad igual a la del nivel más bajo (y más grande).

En la mayoría de los sistemas, la memoria es una verdadera jerarquía, lo que significa que los datos no pueden estar presentes en el nivel *i* a menos que se encuentren también presentes en el nivel *i*+1.

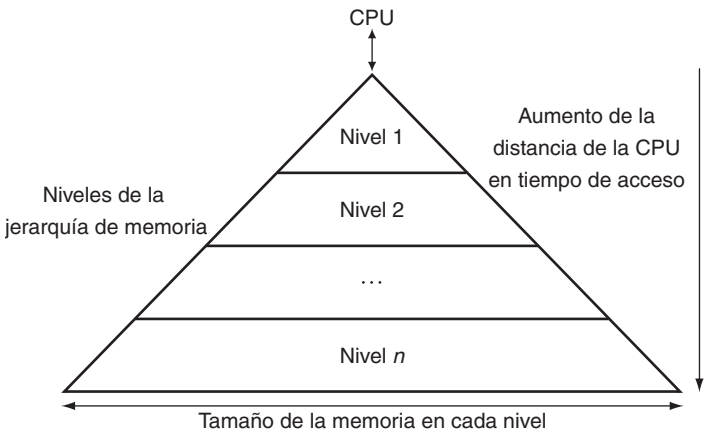


FIGURA 5.3 Este diagrama muestra la estructura de una jerarquía de memoria: a medida que aumenta la distancia desde el procesador, también aumenta la capacidad de almacenamiento. Esta organización, junto con sus condiciones de funcionamiento, permite al procesador tener un tiempo de acceso determinado principalmente por el nivel 1 de la jerarquía y además disponer de una memoria tan grande como la del nivel *n*. Inculcar esta forma de ver la jerarquía de memoria es el objetivo de este capítulo. Aunque el disco local es generalmente el nivel más bajo de la jerarquía, algunos sistemas utilizan periféricos de cinta o sistemas de ficheros en una red de área local como los siguientes niveles de la jerarquía.

Autoevaluación

¿Cuáles de las siguientes afirmaciones son generalmente ciertas?

1. Las memorias caches aprovechan la localidad temporal.
2. En una lectura a memoria, el valor devuelto depende de qué bloques se encuentran en la cache.
3. La mayor parte del coste de la jerarquía de memoria se corresponde con el del nivel más alto.
4. La mayor parte de la capacidad de la jerarquía de memoria se encuentra en el nivel más bajo.

5.2 Principios básicos de las caches

Cache: un lugar seguro para esconder o guardar cosas.

Nuevo Diccionario Mundial del Lenguaje Americano de Webster, Tercera Edición Académica (1988)

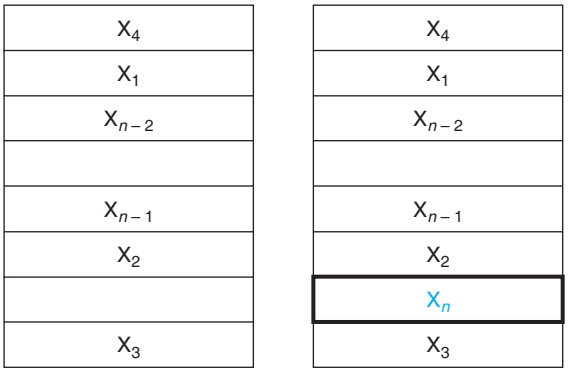
En nuestro ejemplo de la biblioteca, la mesa equivalía a la cache: un lugar seguro para almacenar cosas (libros) que necesitábamos consultar. *Cache* fue el nombre escogido para representar el nivel de la jerarquía que se sitúa entre el procesador y la memoria principal en el primer computador comercial que tuvo este nivel extra de memoria. Las memorias en el camino de datos del capítulo 4 sencillamente se reemplazan por caches. Hoy en día, aunque éste siga siendo el ámbito dominante donde se usa la palabra cache, el término se utiliza también para referirse a cualquier tipo de almacenamiento que aproveche la localidad de los accesos. Las cache aparecieron a principios de la década de 1960 en máquinas de investigación, y posteriormente, durante la misma década, en máquinas comerciales; cada computador de propósito general que se construye hoy en día, desde servidores hasta procesadores empotrados de bajo consumo, incluyen caches.

Empezaremos esta sección considerando una cache muy simple, en la cual tanto las solicitudes del procesador como los bloques tienen el tamaño de una palabra. (Los lectores que ya estén familiarizados con los fundamentos de las caches podrían saltarse esta sección e ir directamente a la sección 5.3). La figura 5.4 muestra esta sencilla cache, antes y después de solicitar un dato que inicialmente no se encuentra en la cache. Antes de la solicitud, la cache contiene un conjunto de datos recientemente solicitados X_1, X_2, ..., X_{n-1}, y el procesador solicita una palabra X_n que no se encuentra en la cache. Esta solicitud genera un fallo, y la palabra X_n es llevada desde la memoria principal a la cache.

Observando la situación de la figura 5.4, surgen dos preguntas a responder: ¿cómo sabemos si un dato está en la cache? Y además, si está, ¿cómo lo encontramos? Las respuestas a estas dos preguntas están relacionadas entre sí. Si cada palabra puede estar alojada en un único lugar de la cache, entonces es muy sencillo encontrar la palabra si ésta se encuentra en la cache. La forma más sencilla de asignar una posición de la cache a cada palabra de la memoria principal consiste en asignar una posición de la cache basándose en la *dirección* de la palabra en memoria principal. Esta estructura de la cache se denomina de **correspondencia directa**, puesto que cada posición de memoria principal se corresponde directamente con una única posición de la cache. La correspondencia típica entre direcciones y posiciones de la cache para caches de este tipo es normalmente muy sencilla. Por ejemplo, casi todas las caches de correspondencia directa utilizan la siguiente asignación

(Dirección de bloque) módulo (Número de bloques de la cache)

Cache de correspondencia directa: organización de cache en la que cada posición de la memoria principal se corresponde con una única posición de la cache.



a. Antes de acceder a X_n b. Después de acceder a X_n

FIGURA 5.4 Estado de la cache antes y después de que la palabra X_n haya sido solicitada, la cual no se encuentra inicialmente en la cache. Esta solicitud ocasiona un fallo que fuerza a la cache a traer X_n desde la memoria principal e insertarla dentro de la cache.

Esta asignación es atractiva ya que si el número de entradas de la cache es una potencia de dos, entonces la operación módulo puede ser realizada simplemente utilizando los $\log_2$ (capacidad de la cache en bloques) bits menos significativos de la dirección de memoria principal. De este modo, en una cache de 8 bloques se utilizan los tres bits menos significativos de la dirección del bloque ($8 = 2^3$). Por ejemplo, la figura 5.5 muestra cómo las direcciones de memoria principal entre 1_{diez} (00001_{dos}) y 29_{diez} (11101_{dos}) se corresponde con las localizaciones 1_{diez} (001_{dos}) y 5_{diez} (101_{dos}) en una cache de correspondencia directa de ocho palabras.

Ya que cada posición de la cache puede alojar el contenido de varias posiciones diferentes de memoria principal, ¿cómo sabemos si los datos de la cache se corresponden con la palabra solicitada? Es decir, ¿cómo sabemos si la palabra solicitada se encuentra o no en la cache? Respondemos a esta pregunta añadiendo un conjunto de **etiquetas** a la cache. Las etiquetas contienen la información de la dirección que se necesita para identificar si una palabra de la cache se corresponde con la palabra solicitada. La etiqueta sólo necesita incluir la parte más significativa de la dirección, lo cual se corresponde con los bits que no son utilizados por la cache como índice. Por ejemplo, en la figura 5.5 es necesario que la etiqueta sólo incluya los 2 bits más significativos de los 5 bits que forman la dirección, ya que el campo índice de la dirección, formado por sus 3 bits menos significativos, selecciona el bloque. Excluimos de la etiqueta los bits del índice debido a que son redundantes, ya que por definición, el campo índice de cualquier dirección de un bloque de cache es siempre el número de bloque.

Etiqueta: campo de cada una de las entradas de la tabla utilizada por la jerarquía de memoria que contiene información de la dirección que es necesaria para identificar si el bloque de información asociado se corresponde con la palabra solicitada.

Bit de validez: campo de cada una de las entradas de la tabla utilizada por la jerarquía de memoria que indica que el bloque asociado en la jerarquía contiene datos válidos.

Necesitamos también una forma de reconocer que la información contenida en un bloque de cache no tiene validez. Por ejemplo, cuando un procesador empieza a ejecutar un programa, la cache puede contener información que no tiene significado para el programa y por eso las etiquetas no tienen sentido. Incluso después de ejecutar muchas instrucciones, algunas de las entradas de la cache pueden estar vacías, como se observa en la figura 5.4. Por ello, se necesita saber que las correspondientes etiquetas deben ser ignoradas. El método más común consiste en añadir un **bit de validez** que indique si

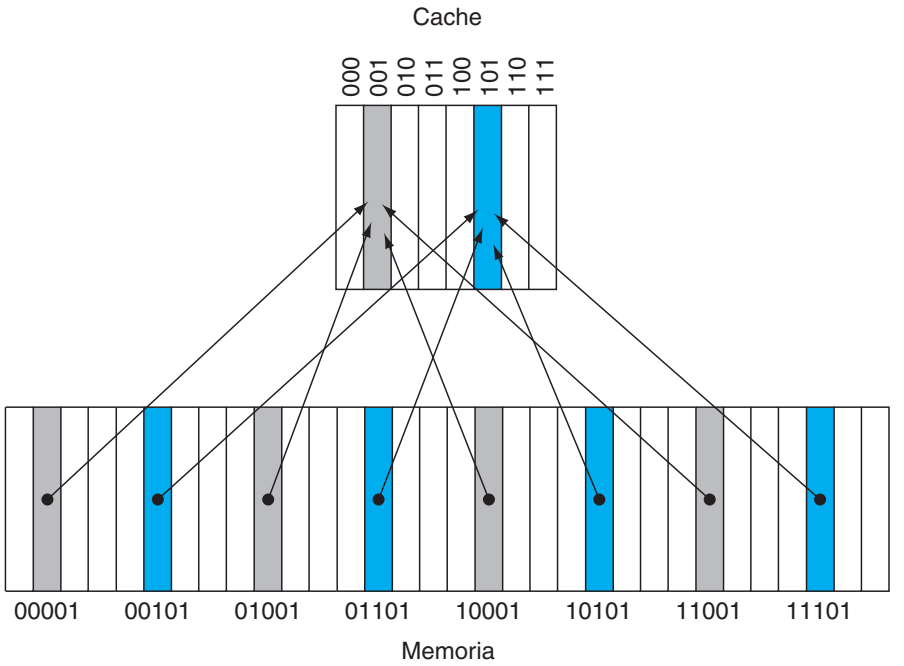


FIGURA 5.5 Una cache de correspondencia directa con ocho entradas donde se muestran las direcciones de palabra de memoria principal que van desde la 0 a la 31 y que se corresponden con las mismas entradas de la cache. Como existen ocho palabras en la cache, una dirección X se corresponde con la palabra de la cache X módulo 8. Es decir, los 3 bits menos significativos ($\log_2(8)$) se utilizan como índice la cache. De este modo, las direcciones 00001_{dos}, 01001_{dos}, 10001_{dos} y 11001_{dos} se corresponden todas con la entrada 001_{dos} de la cache, mientras que las direcciones 00101_{dos}, 01101_{dos}, 10101_{dos} y 11101_{dos} se corresponden todas con la entrada 101_{dos} de la cache.

una entrada contiene una dirección con información válida o no. Si el bit no está activado, el correspondiente bloque de datos no puede ser utilizado.

En el resto de esta sección nos centraremos en explicar cómo funcionan las lecturas a la cache. En general, el manejo de las operaciones de lectura a la cache es una tarea un poco más sencilla que el manejo de las operaciones de escritura, ya que las lecturas no necesitan cambiar el contenido de la cache. Después de describir cómo funcionan las lecturas y cómo se manejan los fallos de cache, examinaremos diseños de caches en computadores reales y detallaremos cómo estas caches manejan las escrituras.

Acceso a la cache

A continuación se muestra una secuencia de nueve referencias a memoria a una cache de 8 bloques inicialmente vacía, incluyendo la acción para cada referencia. La figura 5.6 muestra el estado de la cache y los cambios en cada fallo. Ya que la cache dispone de ocho bloques, los 3 bits menos significativos de la dirección proporcionan el número de bloque:

Dirección decimal del acceso	Dirección binaria del acceso	Acierto o fallo en la cache	Bloque de la cache asignado (donde se encuentra o se almacena)
22	10110_{dos}	fallo (7.6b)	$(10110_{dos} \bmod 8) = 110_{dos}$
26	11010_{dos}	fallo (7.6c)	$(11010_{dos} \bmod 8) = 010_{dos}$
22	10110_{dos}	acierto	$(10110_{dos} \bmod 8) = 110_{dos}$
26	11010_{dos}	acierto	$(11010_{dos} \bmod 8) = 010_{dos}$
16	10000_{dos}	fallo (7.6d)	$(10000_{dos} \bmod 8) = 000_{dos}$
3	00011_{dos}	fallo (7.6e)	$(00011_{dos} \bmod 8) = 011_{dos}$
16	10000_{dos}	acierto	$(10000_{dos} \bmod 8) = 000_{dos}$
18	10010_{dos}	fallo (7.6f)	$(10010_{dos} \bmod 8) = 010_{dos}$

Puesto que la cache está vacía, varías de las primeras referencias son fallos; el pie de la figura 5.6 describe las acciones para cada referencia a memoria. En la octava referencia se produce un conflicto en el acceso a un bloque. La palabra almacenada en la dirección 18 (10010_{dos}) de la memoria principal se lleva al bloque 2 de la cache (010_{dos}), reemplazando a la palabra almacenada en la dirección 26 (11010_{dos}) que ya estaba en el bloque 2 de la cache (010_{dos}). Este comportamiento permite a una cache aprovecharse de la localidad temporal: las palabras utilizadas recientemente reemplazan a las palabras que han sido menos utilizadas recientemente.

Esta situación es análoga a necesitar un libro de las estanterías y no disponer de más espacio sobre la mesa; alguno de los libros de la mesa debe volver a las estanterías. En una cache de correspondencia directa sólo existe un lugar donde alojar los elementos recientemente solicitados y de ahí que sólo exista una única opción para decidir qué reemplazar.

Sabemos dónde buscar en la cache cada una de las posibles direcciones: los bits menos significativos de una dirección pueden ser usados para encontrar la única entrada de la cache que se corresponde con la dirección. La figura 5.7 muestra cómo una dirección solicitada se divide en

- una etiqueta que se usa para compararla con el valor de la etiqueta almacenado en la cache
- un índice de la cache, que se utiliza para seleccionar el bloque de datos

El índice de un bloque de cache junto con el contenido de la etiqueta para este bloque, determina con precisión la dirección de memoria de la palabra almacenada en el bloque de cache. Ya que el campo índice es utilizado como una dirección para acceder a la cache y dado que un campo de n bits codifica hasta 2^n valores distintos, el número total de entradas de una cache de correspondencia directa debe ser potencia de dos. En la arquitectura MIPS, dado que las palabras están alineadas en múltiplos de 4 bytes, los 2 bits menos significativos de cada dirección determinan uno de los bytes que constituyen una palabra, y por ello se ignoran cuando se accede a la palabra de un bloque.

El número total de bits que se requiere para construir una cache está en función de la capacidad de la cache y del tamaño de la dirección debido a que la cache incluye tanto el almacenamiento para los datos como para las etiquetas. El tamaño del bloque utilizado más arriba fue de una palabra, pero normalmente es de varias palabras. Con las siguientes condiciones:

Índice	V	Etiqueta	Datos
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. Estado inicial de la cache después de encender el computador

Índice	V	Etiqueta	Datos
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{dos}	Memoria(10110_{dos})
111	N		

b. Después de resolver el fallo de la dirección (10110_{dos})

Índice	V	Etiqueta	Datos
000	N		
001	N		
010	Y	11_{dos}	Memoria (11010_{dos})
011	N		
100	N		
101	N		
110	Y	10_{dos}	Memoria (10110_{dos})
111	N		

c. Después de resolver el fallo de la dirección (11010_{dos})

Índice	V	Etiqueta	Datos
000	Y	10_{dos}	Memoria (10000_{dos})
001	N		
010	Y	11_{dos}	Memoria (11010_{dos})
011	N		
100	N		
101	N		
110	Y	10_{dos}	Memoria (10110_{dos})
111	N		

d. Después de resolver el fallo de la dirección (10000_{dos})

Índice	V	Etiqueta	Datos
000	Y	10_{dos}	Memoria (10000_{dos})
001	N		
010	Y	11_{dos}	Memoria (11010_{dos})
011	Y	00_{dos}	Memoria (00011_{dos})
100	N		
101	N		
110	Y	10_{dos}	Memoria (10110_{dos})
111	N		

e. Después de resolver el fallo de la dirección (00011_{dos})

Índice	V	Etiqueta	Datos
000	Y	10_{dos}	Memoria (10000_{dos})
001	N		
010	Y	10_{dos}	Memoria (10010_{dos})
011	Y	00_{dos}	Memoria (00011_{dos})
100	N		
101	N		
110	Y	10_{dos}	Memoria (10110_{dos})
111	N		

f. Después de resolver el fallo de la dirección (10010_{dos})

FIGURA 5.6 Estados de la cache después de resolver las peticiones de memoria que fallan en la cache, mostrando en binario los campos del índice y la etiqueta para la secuencia de direcciones de la página 461. La cache está inicialmente vacía, con todos los bits de validez (campo V de la cache) desactivados (N). El procesador solicita las siguientes direcciones 10110_{dos} (fallo), 11010_{dos} (fallo), 10110_{dos} (acierto), 11010_{dos} (acierto), 10000_{dos} (fallo), 00011_{dos} (fallo), 10000_{dos} (acierto), y 10010_{dos} (fallo). Las tablas muestran el estado de la cache después de que cada fallo a la cache haya sido resuelto. Cuando se solicita la dirección 10010_{dos} (18), la entrada de la cache para la dirección 11010_{dos} (26) debe ser reemplazada, y una posterior referencia a 11010_{dos} causará un nuevo fallo en la cache. La etiqueta contendrá sólo la parte más significativa de la dirección. La dirección completa de una palabra almacenada en el bloque i de la cache cuya etiqueta contiene j es $j \times 8 + i$, o lo que es lo mismo, la concatenación de la etiqueta j y el índice i. Por ejemplo, en la situación f de la cache de arriba, el índice 010_{dos} contiene una etiqueta 10_{dos} que corresponde a la dirección 10010_{dos}.

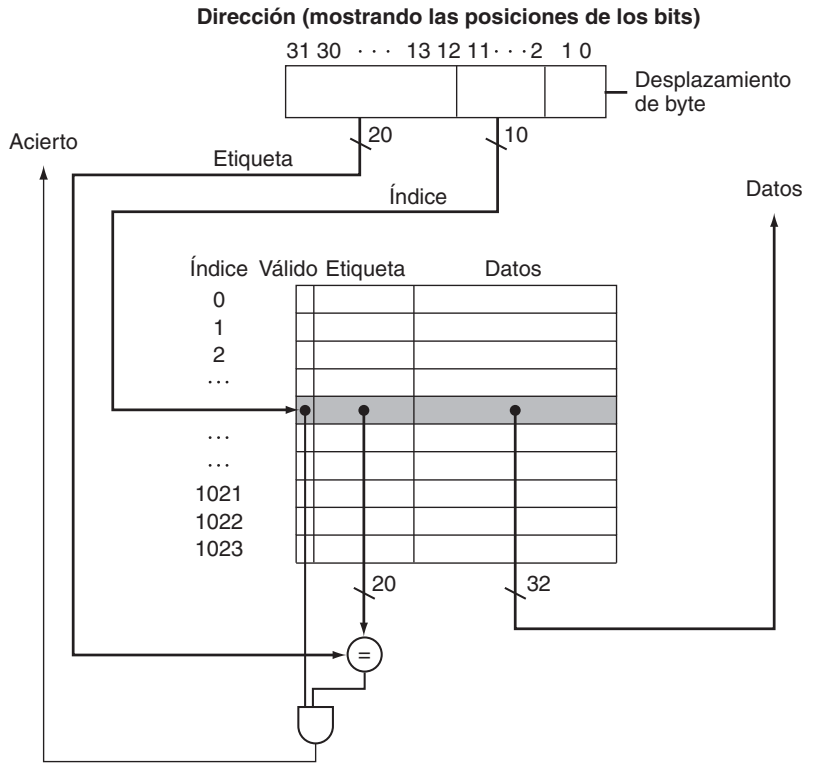


FIGURA 5.7 Para esta cache, la parte menos significativa de la dirección se utiliza para seleccionar una entrada de la cache que está formada por una palabra de datos y una etiqueta. Es una cache de 1024 palabras o 4 KB. En este capítulo supondremos que las direcciones son de 32 bits. La etiqueta almacenada en la cache se compara con la parte más significativa de la dirección para determinar si la entrada de la cache se corresponde con la dirección solicitada o no. Ya que la cache dispone de 2^{10} (o 1024) palabras y un tamaño de bloque de 1 palabra, se utilizan 10 bits para indexar la cache, dejando 32 – 10 – 2 = 20 bits para cada etiqueta. Si esta etiqueta y los 20 bits más significativos de la dirección coinciden y el bit de validez está activado, entonces la petición de memoria acierta en la cache, y la palabra es suministrada al procesador. En caso contrario, se produce un fallo.

- Direcciones de bytes de 32 bits
- Cache de correspondencia directa
- El tamaño de cache es 2^n bloques, por lo tanto se utilizan n bits para el índice
- El tamaño de bloque es 2^m palabras (2^{m+2} bytes), por lo tanto se utilizan m bits para identificar la palabra dentro del bloque y dos bits para identificar el byte de la dirección

el tamaño de la etiqueta, en bits, es

$$32 - (n + m + 2).$$

El número total de bits de una cache de correspondencia directa es

$2^n \times$ (tamaño de bloque + tamaño de la etiqueta + tamaño del campo de validez).

Puesto que el tamaño de bloque es 2^m palabras (2^{m+5} bits) y se necesita sólo 1 bit para el campo de validez, en número de bits de la cache es

$$2^n \times (2^m \times 32 + (32 - n - m - 2) + 1) = 2^n \times (2^m \times 32 + 31 - n - m).$$

Aunque este es el tamaño real en bits, la convención utilizada para referirse a la capacidad de la cache excluye el tamaño de la etiqueta y del campo de validez, para incluir sólo el tamaño de los datos. De este modo, la cache de la figura 5.7 se dice que es una cache de 4 KB.

Bits en una cache

EJEMPLO

¿Cuántos bits son necesarios para implementar una cache de correspondencia directa con 16 KB de datos y bloques de 4 palabras, suponiendo direcciones de 32 bits?

RESPUESTA

Sabemos que 16 KB se corresponden con 4K (2^{12}) palabras, y con un tamaño de bloque de 4 palabras (2^2), 1024 (2^{10}) bloques. Cada bloque tiene 4×32 ó 128 bits de datos además de una etiqueta, la cual está formada por 32 – 10 – 2 – 2 bits, a los que hay que añadir un bit de validez. De este modo, la capacidad total de la cache es

$$2^{10} \times (128 + (32 - 10 - 2 - 2) + 1) = 2^{10} \times 147 = 147 \text{ Kbits}$$

o 18.4 KB para una cache de 16 KB. Para esta cache, el número total de bits que forman la cache es aproximadamente 1.15 veces la capacidad necesaria para almacenar los datos.

Correspondencia de una dirección de memoria con un bloque multipalabra de cache

EJEMPLO

Suponga que existe de una cache con 64 bloques y un tamaño de bloque de 16 bytes. ¿Qué número de bloque se corresponde con la dirección de byte 1200?

RESPUESTA

Siguiendo la fórmula indicada en la página 457, el número de bloque viene dado por

(Dirección de bloque) módulo (Número de bloques de la cache)

donde la dirección del bloque es

$$\frac{\text{Dirección de byte}}{\text{Bytes por bloque}}$$

Observe que esta dirección de bloque es el bloque que contiene todas las direcciones que van desde

$$\left\lfloor \frac{\text{Dirección de byte}}{\text{Bytes por bloque}} \right\rfloor \times \text{Bytes por bloque}$$

y

$$\left\lfloor \frac{\text{Dirección de byte}}{\text{Bytes por bloque}} \right\rfloor \times \text{Bytes por bloque} + (\text{Bytes por bloque} - 1)$$

De esta manera, con 16 bytes por bloque, la dirección de byte 1200 se identifica con la siguiente dirección de bloque

$$\left\lfloor \frac{1200}{16} \right\rfloor = 75$$

la cual se corresponde con el bloque de cache número (75 módulo 64) = 11. De hecho, este bloque se corresponde con todas las direcciones que van desde 1200 hasta 1215.

Bloques más grandes aprovechan la localidad espacial para reducir la frecuencia de fallos. Como se puede observar en la figura 5.8, al aumentar el tamaño de bloque, la frecuencia de fallos normalmente se reduce. Sin embargo, la frecuencia de fallos puede aumentar si el tamaño de los bloques se hace demasiado grande, ya que el número total de bloques de la cache disminuye y habrá una gran competencia por ocupar esos bloques. Como resultado, el bloque será desalojado de la cache antes de que se acceda a muchas de sus palabras. Expresado de otra manera, la localidad espacial de las palabras de un bloque disminuye a medida que el bloque se hace muy grande; en consecuencia, los beneficios que proporciona la frecuencia de fallos se hacen cada vez más pequeños.

Un problema más serio asociado con el incremento del tamaño de bloque es que la penalización por fallo aumenta. La penalización por fallo viene determinada por el tiempo necesario para llevar el bloque desde el siguiente nivel inferior de la jerarquía hasta la cache. El tiempo para llevar el bloque tiene dos componentes: la latencia de la primera palabra y el tiempo de transferencia del resto del bloque. Obviamente, a no ser que cambiemos el sistema de memoria, el tiempo de transferencia (y por lo tanto, la penalización por fallo) aumentará a medida que el tamaño de bloque aumenta. Además, la mejora de la frecuencia de fallos comienza a disminuir a medida que el bloque es de tamaño mayor. Como resultado, el aumento de la penalización por fallo no es compensado por la disminución de la frecuencia de fallos para bloques grandes, y por eso las prestaciones de la cache disminuyen. Por supuesto, si se diseña el sistema de memoria de forma tal que los bloques más grandes son los que se transfieren más eficientemente, podemos aumentar el tamaño del bloque y mejorar las presentaciones de la cache. Discutiremos este tema en la siguiente sección.

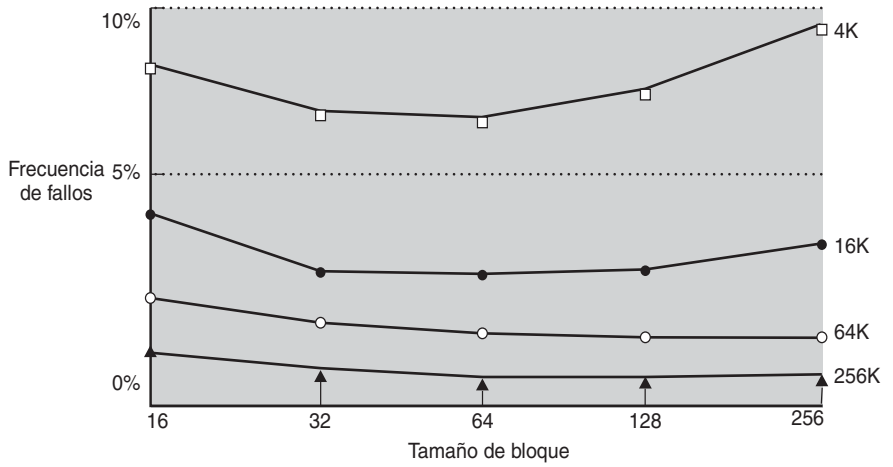


FIGURA 5.8 Frecuencia de fallos frente a tamaño de los bloques. Observe que la frecuencia de fallos aumenta realmente si el tamaño de los bloques es demasiado grande respecto a la capacidad de la cache. Cada curva representa una cache de distinta capacidad. (Esta figura es independiente de la asociatividad, la cual se discutirá posteriormente). Desafortunadamente, las simulaciones de los programas SPEC2000 tardarían demasiado tiempo si el tamaño de bloque fuera tomado en consideración, por eso, estos datos fueron obtenidos con programas SPEC92.

Extensión: Aunque es complicado mejorar la componente latencia de la penalización por fallo en bloques grandes, somos capaces de solapar parte del tiempo de transferencia de forma tal que el efecto de la penalización por fallo se reduce. La forma más sencilla de hacer esto, denominada *comienzo inmediato*, consiste simplemente en reanudar la ejecución tan pronto como la palabra solicitada del bloque es recibida, en vez de esperar a que termine de transferirse el bloque entero. Muchos procesadores usan esta técnica para acceder a las instrucciones, donde funciona mejor. Los accesos a instrucciones son en su mayor parte secuenciales, así que si el sistema de memoria puede proporcionar una palabra cada ciclo de reloj, el procesador puede ser capaz de comenzar a procesar la instrucción que ha producido el fallo cuando es recibida, y en los siguientes ciclos de reloj seguir recibiendo a tiempo las nuevas instrucciones. Esta técnica es normalmente menos efectiva para las caches de datos debido a que probablemente las palabras sean solicitadas de una forma menos predecible, y a la alta probabilidad de que el procesador necesite otra palabra de otro bloque antes que la transferencia finalice. Si el procesador no puede acceder a la cache de datos debido a que se está realizando una transferencia, entonces debe pararse.

Una técnica incluso más sofisticada consiste en organizar la memoria principal de forma tal que la palabra solicitada sea la primera en trasferirse desde la memoria principal a la cache. El resto del bloque se transfiere a continuación, comenzado con la dirección que se encuentra después de la palabra solicitada, y una vez llegado al final del bloque se sigue transfiriendo desde su comienzo. Esta técnica, denominada *palabra solicitada primero* o *palabra crítica primero*, puede ser ligeramente más rápida que la de comienzo inmediato, pero está limitada por las mismas propiedades que limitan el comienzo inmediato.

Manejo de los fallos de cache

Antes de analizar la cache de un sistema real, veamos cómo trata la unidad de control los **fallos de cache**. (El controlador de cache se describe en detalle en la sección 5.7.) La unidad de control debe detectar y procesar un fallo trayendo los datos solicitados

Fallo de cache: solicitud de acceso a datos de la cache que no puede ser atendida debido a que los datos no están presentes en la cache.

desde la memoria principal (o como veremos, desde un nivel inferior de cache). Si el acceso acierta en la cache, el computador continúa utilizando los datos como si lo que ha ocurrido es algo normal.

La modificación del control de un procesador para manejar aciertos es trivial; los fallos, sin embargo, requieren trabajo adicional. El manejo de los fallos de cache se realiza con la unidad de control del procesador y con un controlador independiente que pone en marcha el acceso a memoria y actualiza el contenido de la cache. El procesamiento de un fallo de cache genera una parada del procesador (capítulo 4) a diferencia de una interrupción, la cual requiere guardar el estado de todos los registros. Para un fallo de cache, podemos realizar la paralización de todo el procesador simplemente no actualizando los contenidos de los registros tanto temporales como los que son visibles al programador, mientras que se espera por la memoria principal. Procesadores más sofisticados con ejecución fuera de orden, permiten la ejecución de instrucciones durante la espera por un fallo de cache, pero en el resto de esta sección supondremos procesadores con ejecución en orden, que paran durante el fallo de cache.

Veamos con un poco más de detalle cómo se manejan los fallos de instrucciones; el mismo planteamiento puede aplicarse al manejo de los fallos de datos. Si un acceso a una instrucción produce un fallo, entonces el contenido del Registro de Instrucciones no es válido. Para obtener de la cache la instrucción correcta, debemos ser capaces de indicarle al nivel inferior de la jerarquía de memoria que realice una lectura. Ya que el contador de programa se incrementa en cada ciclo de reloj de la ejecución, la dirección de la instrucción que genera un fallo en la cache de instrucciones coincide con el valor del contador de programa restándole 4. Una vez que se conozca esta dirección, se indica a la memoria principal que realice una lectura. Esperamos a que la memoria principal responda (ya que el acceso tardará varios ciclos), y escribimos las palabras que contienen la instrucción solicitada en la cache.

Ahora podemos establecer los pasos involucrados en el manejo de un fallo de la cache de instrucciones:

1. Enviar el valor del PC original (PC actual – 4) a la memoria.
2. Indicarle a la memoria principal que realice una lectura y que después espere a que la memoria termine este acceso.
3. Actualizar la entrada de la cache, guardando el dato traído desde la memoria en la parte de la entrada destinada a los datos, actualizando los bits más significativos de la dirección (proporcionados por la ALU) en la parte destinada a la etiqueta y modificando el bit de validez para indicar entrada válida.
4. Reanudación de la ejecución de la instrucción desde su comienzo, que buscará de nuevo la instrucción y esta vez la encontrará en la cache.

El control de la cache cuando se produce un acceso a datos es fundamentalmente idéntico: en un fallo, simplemente paralizamos el procesador hasta que la memoria devuelva los datos.

Manejo de las escrituras

Las escrituras funcionan de una forma un poco distinta. Suponga que aparece una instrucción de almacenamiento con la que sólo escribimos datos en la cache de datos (sin modificar la memoria principal); entonces, después de actualizar la cache, la

memoria guarda un valor distinto del que se guarda en la cache. En tal caso se dice que la cache y la memoria son *incoherentes*. La forma más simple de mantener la coherencia de la memoria principal y la cache consiste en escribir siempre los datos tanto en la memoria como en la cache. Esta técnica se denomina **escritura directa**.

Escritura directa: técnica en la cual las escrituras siempre actualizan tanto la cache como la memoria principal, asegurando que los datos siempre son coherentes en ambas memorias.

El otro aspecto clave de las escrituras consiste en explorar lo que ocurre en los fallos de escritura. Primero se van a buscar a memoria las palabras del bloque. Después de que el bloque se haya recibido y guardado en la cache, podemos sobrescribir la palabra que originó el fallo en un bloque de cache. La palabra también se escribe en memoria principal utilizando toda la dirección.

Aunque este diseño maneja las escrituras de una forma muy simple, no proporcionaría muy buenas prestaciones. Con un esquema de escritura directa, cada escritura obliga a que los datos sean escritos en la memoria principal. Estas escrituras tardarán mucho tiempo, probablemente al menos 100 ciclos del reloj del procesador, y podrían ralentizar considerablemente al procesador. Por ejemplo, supongamos que el 10% de las instrucciones son almacenamientos. Si el CPI sin considerar fallos de cache era 1.0, esperar 100 ciclos adicionales en cada escritura llevaría a que el CPI fuera $1.0 + 100 \times 10\% = 11$, reduciendo las prestaciones en un factor superior a 10.

Una solución a este problema consiste en usar un **búfer de escritura**. Un búfer de escritura almacena los datos mientras se están escribiendo en memoria. Después de escribir los datos tanto en la cache como en el búfer de escritura, el procesador puede continuar la ejecución. Cuando finaliza una escritura a memoria principal, la correspondiente entrada en el búfer de escritura se libera. Si el búfer de escritura se llena cuando el procesador ejecuta una escritura, el procesador debe pararse hasta que exista una entrada libre en el búfer de escritura. Por supuesto, si el ritmo al que la memoria principal puede completar las escrituras es inferior al ritmo con el que el procesador está generando las escrituras, ningún tamaño del búfer puede impedir las paradas debido a que las escrituras se generan a mayor ritmo del que el sistema de memoria puede aceptarlas.

Búfer de escritura: cola que almacena temporalmente los datos mientras que los datos esperan a que sean escritos en memoria principal.

El ritmo con el cual se generan las escrituras puede también ser inferior al ritmo que la memoria puede aceptarlas, y todavía producirse paradas. Esto puede ocurrir cuando las escrituras se producen a ráfagas. Para reducir la aparición de tales paradas, los procesadores normalmente aumentan el número de entradas del búfer de escritura.

La alternativa a la técnica de escritura directa corresponde a la técnica denominada **escritura retardada**. En un esquema de escritura retardada, cuando se produce una escritura, el nuevo valor se escribe sólo en el bloque de la cache. El bloque modificado se guarda en el nivel inferior de la jerarquía de memoria sólo cuando va a ser reemplazado. Los esquemas de escritura retardada pueden mejorar las prestaciones, especialmente en los casos en que los procesadores pueden generar escrituras con tanta o mayor frecuencia que la utilizada por la memoria principal para manejarlas. Sin embargo, un esquema de escritura retardada es más complejo de implementar que el de escritura directa.

Escritura retardada: técnica que maneja las escrituras primero actualizando sólo los valores del bloque de la cache y posteriormente guardando el bloque en el nivel inferior de la jerarquía cuando este bloque sea reemplazado.

En el resto de esta sección describiremos caches integradas en procesadores reales, y examinaremos cómo se manejan tanto las lecturas como las escrituras. En la sección 5.5 describiremos con más detalle el manejo de las escrituras.

Extensión: Las escrituras introducen en la cache varias complicaciones que no están presentes en las lecturas. Veamos ahora dos de ellas: la política de los fallos de escritura y la implementación eficiente de las escrituras en la cache de escritura retardada.

Consideremos un fallo en una cache de escritura directa. La estrategia más habitual consiste en reservar un bloque en la cache, llamado *reserva de escritura*. El bloque se lee de la memoria y la parte correspondiente del bloque se sobreescribe. Una estrategia alternativa, llamada *sin reserva de escritura*, consiste en actualizar la parte del bloque en la memoria, pero sin traerla a la cache. La motivación de esta técnica es que a veces los programas escriben bloques enteros de datos, por ejemplo cuando el sistema operativo escribe ceros en todas las palabras de una página de memoria. En estos casos, la búsqueda asociada con los primeros fallos de escritura puede ser innecesaria. En algunos computadores la estrategia de reserva de escritura se cambia por una estrategia basada en páginas de memoria.

Implementar eficientemente los almacenamientos en una cache que utiliza la estrategia de escritura retardada es más complejo que en una cache de escritura directa. Una cache de escritura directa puede escribir el dato en la cache y luego leer la etiqueta. Si la etiqueta no coincide se produce un fallo. Como la cache es de escritura directa, la sobreescritura del bloque no es catastrófica ya que la memoria dispone del valor correcto. En una cache de escritura retardada, el contenido del bloque debe guardarse en la memoria principal si el dato en la cache ha sido modificado y se produce un fallo de cache. Si en un almacenamiento simplemente se sobreescribe el bloque de cache antes de saber si el acceso acierta en la cache (como ocurre en la cache de escritura directa), se perdería el contenido del bloque, que no ha sido actualizado previamente en el siguiente nivel de la jerarquía de memoria.

En una cache de escritura retardada, como no se puede sobrescribir el bloque, los almacenamientos requieren o bien dos ciclos (un ciclo para comprobar que se acierta seguido de otro ciclo en el que se realiza la escritura) o bien un registro adicional denominado *búfer de almacenamiento*, para alojar los correspondientes datos (permitiendo de hecho que el almacenamiento tarde un solo ciclo si se utiliza segmentación). Cuando se utiliza un búfer de almacenamiento, el procesador realiza la inspección de la cache y guarda los datos en el búfer de almacenamiento a lo largo del ciclo que se usa normalmente para el acceso a la cache. Suponiendo que se acierta en la cache, los nuevos datos se transfieren desde el búfer de almacenamiento hasta la cache durante un siguiente ciclo de acceso a la cache que no se utilice.

Por comparación, en una cache de escritura directa, los almacenamientos siempre pueden ser realizados en un ciclo. Se lee la etiqueta y se escribe la parte de los datos del bloque seleccionado. Si la etiqueta se corresponde con la dirección del bloque que está siendo parcialmente escrito, el procesador puede continuar normalmente, ya que el bloque correcto ha sido actualizado. Si la etiqueta no se corresponde, el procesador genera un fallo de escritura para buscar el resto del bloque al que corresponde tal dirección.

Muchas caches de escritura retardada también incluyen búferes de escritura que se utilizan para reducir la penalización por fallo cuando un fallo reemplaza un bloque modificado. En tal caso, el bloque modificado se almacena temporalmente en el búfer de escritura retardada que está asociado a la cache mientras el bloque solicitado se lee desde la memoria principal. El contenido del búfer de escritura retardada es posteriormente actualizado en la memoria principal. Suponiendo que otro fallo no aparece inmediatamente, esta técnica reduce a la mitad la penalización cuando un bloque no coherente debe ser reemplazado.

Un ejemplo de cache: el procesador FastMATH de Intrinsity

El FastMATH de Intrinsity es un microprocesador empotrado rápido que utiliza la arquitectura MIPS y una implementación sencilla de la cache. Cerca del final del capítulo, examinaremos el diseño de cache más complejo del AMD Opteron X4 (Barcelona), pero por razones pedagógicas comenzaremos con este ejemplo que es simple pero real. La figura 5.9 muestra la organización de la cache de datos del FastMATH de Intrinsity.

Este procesador tiene un camino de datos segmentado de 12 etapas, similar al descrito en el capítulo 4. Cuando opera a velocidad pico, el procesador puede solicitar una palabra de instrucción y otra palabra de datos en cada ciclo de reloj. Para satisfacer las demandas del camino de datos segmentado sin que se produzcan paradas, las caches de instrucciones y de datos están separadas. Cada cache es de 16 KB, o 4K palabras, con bloques de 16 palabras.

Las solicitudes de lectura para la cache son simples. Debido a que existen caches de datos e instrucciones separadas, se necesitarán distintas señales de control para leer y escribir cada cache. (Recuerde que se necesita actualizar la cache de instrucciones cuando se produce un fallo). De este modo, los pasos que se siguen en cada cache para una solicitud de lectura son los siguientes:

1. Se envía la dirección a la cache apropiada. La dirección proviene, bien desde el PC (para una instrucción), bien de la ALU (para datos).

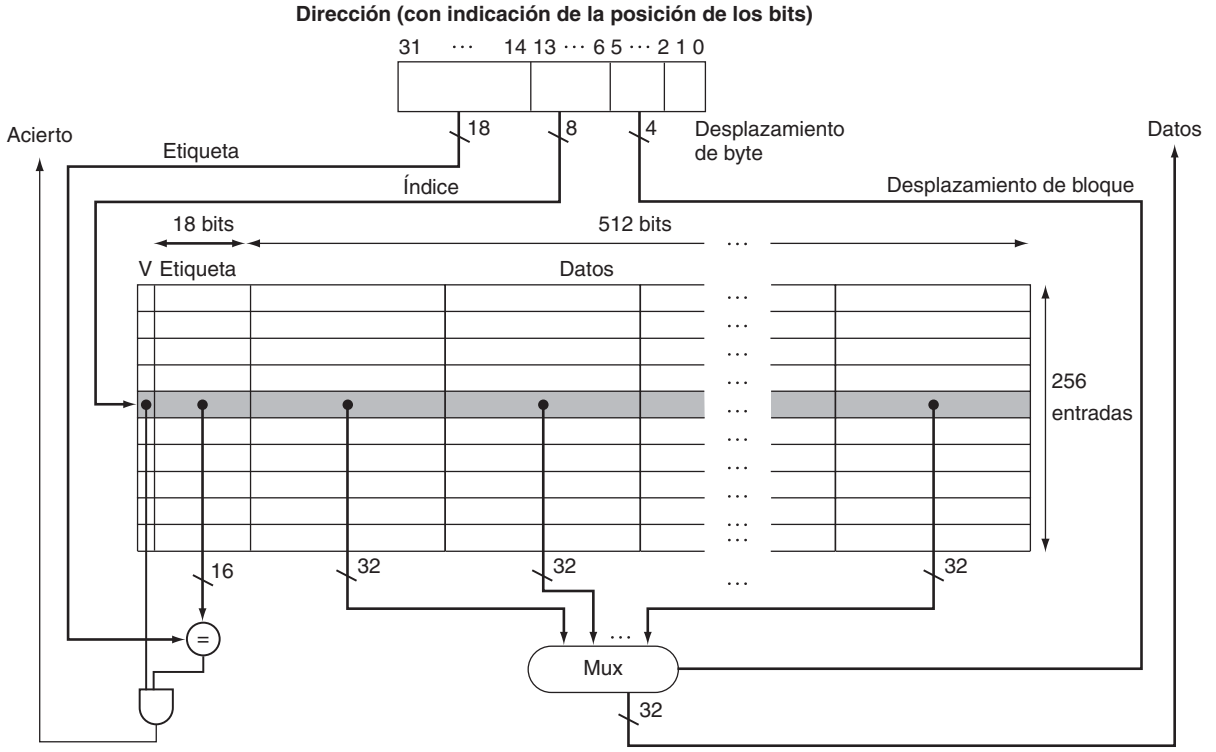


FIGURA 5.9 Las caches de 16 KB en el FastMATH de Intrinsity, cada una de ellas contiene 256 bloques con 16 palabras por bloque. El campo etiqueta ocupa 18 bits y el campo índice ocupa 8 bits, mientras que un campo de 4 bits (bits 5-2) se usa para indexar el bloque y seleccionar la palabra del bloque por medio de un multiplexor 16-a-1. En la práctica, para eliminar el multiplexor, las caches combinan una RAM con mayor rango de direccionamiento para los datos y una RAM con menor rango de direccionamiento para las etiquetas, con los bits adicionales de la dirección que determinan el desplazamiento dentro del bloque en la RAM de datos. En este caso, la RAM más grande tiene un ancho de palabra de 32 bits y debe disponer de un número de palabras que es 16 veces el número de bloques de la cache.

2. Si se acierta en la cache, la palabra solicitada se encuentra disponible en las líneas de datos. Ya que existen 16 palabras en el bloque deseado, necesitamos seleccionar la palabra que se pide. Un campo del índice del bloque se usa para controlar el multiplexor (que aparece en la parte inferior de la figura), el cual selecciona la palabra solicitada de entre las 16 palabras del bloque indexado.
3. Si se falla en la cache, la dirección se envía a la memoria principal. Cuando la memoria devuelve el dato, se escribe en la cache y luego éste se lee para completar la operación solicitada.

Para los almacenamientos, el FathMATH de Intrinsity ofrece tanto escrituras directas como escrituras retardadas, dejando al sistema operativo que decida qué estrategia usar para una aplicación. El microprocesador dispone de un búfer de escritura de una sola entrada.

¿Cuál es la frecuencia de fallos que se alcanza con una cache cuya estructura es como la que usa el FastMATH de Intrinsity? La figura 5.10 muestra las frecuencias de fallos de las caches de instrucciones y de datos. La frecuencia de fallos combinada es la frecuencia de fallos efectiva por acceso que muestra cada programa después de considerar las distintas frecuencias de accesos a instrucciones y datos.

Aunque la frecuencia de fallos es una característica importante del diseño de caches, la medida final reflejará el efecto del sistema de memoria sobre el tiempo de ejecución de los programas. Dentro de poco veremos cómo se relacionan la frecuencia de fallos y el tiempo de ejecución.

Cache separada: técnica en la cual un nivel de la jerarquía de memoria se compone de dos caches independientes que funcionan en paralelo, una de ellas destinada a manejar instrucciones y la otra a manejar datos.

Extensión: Una cache combinada con una capacidad total igual a la suma de dos **caches separadas** tendrá normalmente una mayor frecuencia de aciertos. Ellos se debe a que la cache combinada no diferencia estrictamente las entradas que pueden ser usadas por las instrucciones de las que son usadas por los datos. Aún así, muchos procesadores utilizan caches separadas para instrucciones y datos con el objetivo de aumentar el *ritmo de transferencia o ancho de banda* (*bandwidth*). (Puede haber, también, menos fallos de conflicto; véase sección 5.5.)

A continuación se dan las frecuencias de fallos para caches del tamaño que se encuentra en el procesador FastMATH de Intrinsity y para una cache combinada cuya capacidad es igual al total de las dos caches.

- Capacidad total de la cache: 32 KB
- Frecuencia de fallos efectiva de la cache separada: 3.24%
- Frecuencia de fallos de la cache combinada: 3.18%

La frecuencia de fallos de la cache separada resulta ser sólo ligeramente peor.

Frecuencia de fallos para las instrucciones	Frecuencia de fallos para los datos	Frecuencia de fallos combinada
0.4%	11.4%	3.2%

FIGURA 5.10 Frecuencias aproximadas de fallos para instrucciones y datos en el procesador FastMATH de Instrinsity que se obtienen con los programas de prueba SPEC2000. La frecuencia de fallos combinada es la frecuencia de fallos efectiva que experimenta una combinación formada por una cache de instrucciones de 16 KB y una cache datos de 16 KB. Se obtiene factorizando las frecuencias de fallos individuales por la frecuencia de accesos a instrucciones y datos.