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).

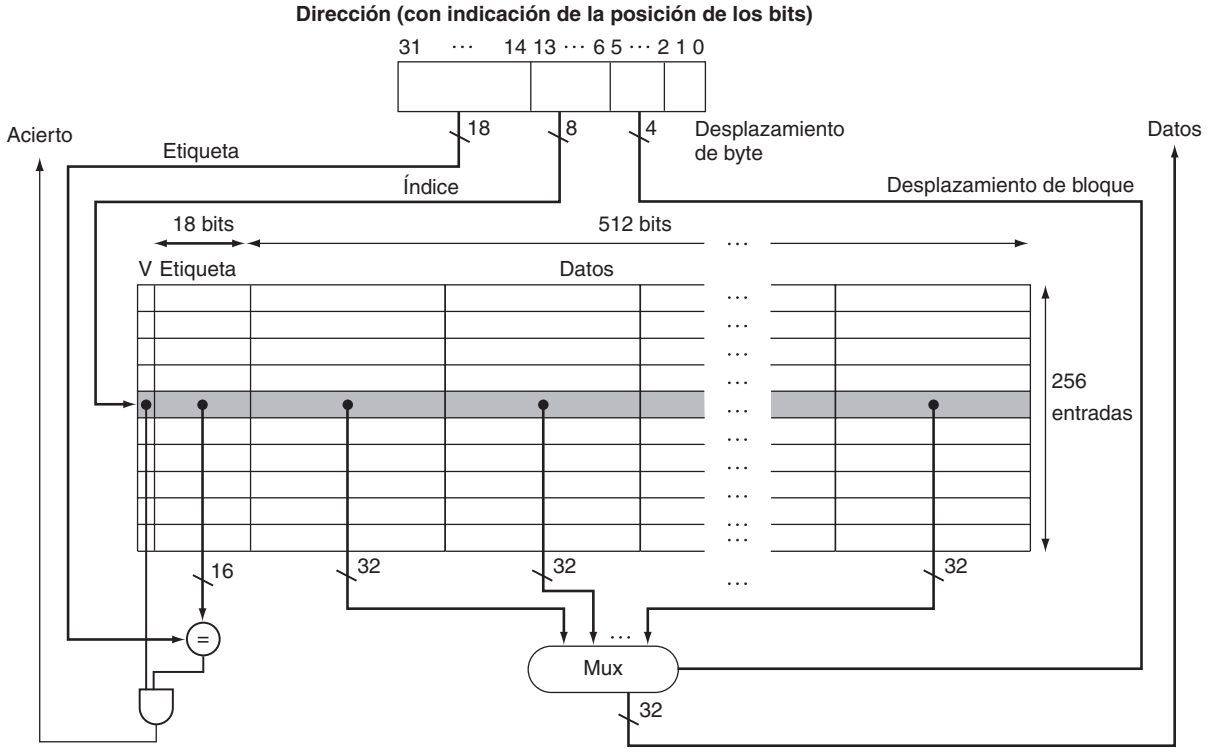

FIGURA 5.9 Las caches de 16 KB en el FastMATH de Intrinsity, cada una de ellas contiene 256 bloques con 16 palabras por bloque. El campo etiqueta ocupa 18 bits y el campo índice ocupa 8 bits, mientras que un campo de 4 bits (bits 5-2) se usa para indexar el bloque y seleccionar la palabra del bloque 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.

La ventaja de doblar el ancho de banda de la cache a través del acceso simultáneo a una instrucción y a un dato, supera fácilmente la desventaja de una frecuencia de fallos ligeramente mayor. Esta observación nos recuerda que no podemos utilizar la frecuencia de fallos como única medida de las prestaciones de la cache, como se muestra en la sección 5.3.

Diseño del sistema de memoria para soportar caches

Los fallos de cache se resuelven con los datos de la memoria principal, la cual se fabrica con DRAMs. En la sección 5.1 vimos que el diseño de las DRAMs persigue optimizar la densidad del chip en vez del tiempo de acceso. Aunque es difícil reducir la latencia para recibir la primera palabra desde la memoria principal, podemos reducir la penalización por fallo si aumentamos el ancho de banda desde la memoria a la cache. Esta reducción permite usar tamaños más grandes de bloque manteniendo una baja penalización por fallo, similar a aquella que caracteriza a bloques más pequeños.

Por regla general, el procesador está conectado a la memoria a través de un bus. (Como se verá en el capítulo 6, esta forma de conexión está cambiando, pero en este capítulo nos tiene sin cuidado la tecnología actual de interconexión, por lo que usaremos el término bus.) La frecuencia de reloj del bus normalmente es muy inferior a la del procesador. La frecuencia de este bus afecta a la penalización por fallo.

Para comprender el impacto de distintas organizaciones de memoria, definamos un conjunto de tiempos de acceso a memoria hipotéticos. Tomemos

- 1 ciclo de reloj del bus de memoria para enviar la dirección
- 15 ciclos de reloj del bus de memoria para cada acceso iniciado en la DRAM
- 1 ciclo de reloj del bus de memoria para enviar una palabra de datos

Si tenemos un bloque de cache de cuatro palabras y un banco de DRAM de una palabra de ancho, la penalización por fallo sería $1 + 4 \times 15 + 4 \times 1 = 65$ ciclos de reloj del bus de memoria. De este modo, el número de bytes transferidos por ciclo de reloj en un fallo individual sería

$$\frac{4 \times 4}{65} = 0.25$$

La figura 5.11 muestra tres opciones para diseñar el sistema de memoria. La primera opción obedece a lo que hemos estado suponiendo: una memoria de una palabra de ancho, y todos los accesos se realizan de forma secuencial. La segunda opción aumenta el ancho de banda de la memoria utilizando una memoria y un bus procesador-memoria más anchos; esto permite realizar accesos paralelos a varias palabras del bloque. La tercera opción aumenta el ancho de banda por medio de una memoria más ancha pero sin aumentar el ancho del bus de interconexión. De este modo, todavía se requiere un tiempo para transmitir cada palabra, pero se evita esperar una latencia que sea superior a la de un solo acceso. Observemos en cuánto estas dos opciones mejoran la penalización por fallo de 65 ciclos que calculamos para la primera de las opciones (figura 5.11a).

Aumentando el ancho de la memoria y el ancho del bus se aumenta proporcionalmente el ancho de banda, y se reducen las componentes de la penalización por fallo que corresponden tanto al tiempo de acceso como al tiempo de transferencia. Con un ancho de memoria principal de dos palabras, la penalización por fallo se

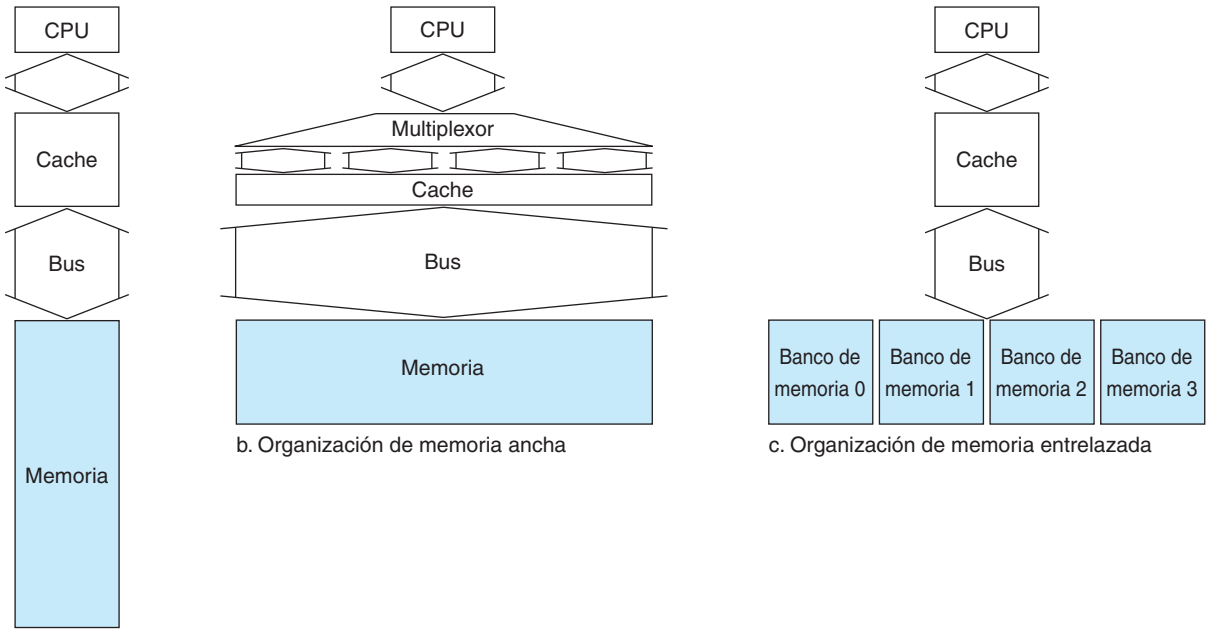

FIGURA 5.11 El principal método para conseguir un mayor ancho de banda de memoria consiste en incrementar el ancho físico y lógico del sistema de memoria. En esta figura, el ancho de banda de memoria se mejora de dos maneras. El diseño más simple, (a), utiliza una memoria donde todos los componentes son de una palabra de ancho; (b) muestra una memoria, bus, y cache más anchas; mientras que (c) muestra un bus y una cache más estrechos con una memoria entrelazada. En (b), la lógica entre la cache y el procesador está formada por un multiplexor que se usa en las lecturas y por una lógica de control que actualiza las palabras apropiadas de la cache en las operaciones de escritura.

reduce de los iniciales 65 ciclos de reloj del bus de memoria a $1 + 2 \times 15 + 2 \times 1 = 33$ ciclos de reloj del bus de memoria. El ancho de banda para un único fallo es de 0.48 (casi el doble) bytes por ciclo de reloj del bus considerando una memoria de dos palabras de ancho. Los costes más importantes de esta mejora corresponden a un bus más ancho y al posible incremento del tiempo de acceso a la cache debido al multiplexor y a la lógica de control que aparecen entre el procesador y la cache.

En vez de hacer más ancha toda la ruta entre la memoria y la cache, los chips de memoria pueden ser organizados en bancos para leer o escribir muchas palabras en un solo tiempo de acceso en vez de leer o escribir una única palabra cada vez. Cada banco podría ser de una palabra de ancho de forma que el ancho del bus y la cache no necesiten cambiar, pero el envío de una dirección a varios bancos permite leerlos todos simultáneamente. Este esquema que se denomina *entrelazado*, tiene la ventaja de que se necesita esperar sólo la latencia de un único acceso a memoria. Por ejemplo, con cuatro bancos, el tiempo para obtener un bloque de cuatro palabras se calcularía sumando 1 ciclo para transmitir la dirección y la petición de lectura a los bancos, 15 ciclos para que todos los bancos accedan a memoria, y 4 ciclos para enviar las cuatro palabras a la cache. Esto conlleva una penalización por fallo de $1 + 1 \times 15 + 4 \times 1 = 20$ ciclos de reloj del bus de

memoria. Esto implica que el ancho de banda por fallo es de 0.80 bytes por ciclo, o unas tres veces el ancho de banda de una memoria y un bus de una palabra de ancho. Los bancos son también valiosos para las escrituras. Cada banco puede escribir independientemente, cuadriplicando el ancho de banda de las escrituras y proporcionando menos paradas en una cache de escritura directa. Como veremos, una estrategia alternativa para las escrituras hace que el entrelazado sea una técnica todavía más atractiva.

Debido a la gran implantación de las caches y al deseo de disponer de tamaños de bloque mayores, los fabricantes de DRAM mantienen un acceso rápido a datos (*burst acces*) situados en una serie de posiciones consecutivas en la DRAM. El desarrollo más actual es la *DRAM de doble frecuencia de datos (Double data rate DRAM),* o DDR DRAM. En esta DRAM los datos se transfieren tanto en el flanco de subida del reloj como en el de bajada, doblando así el ancho de banda respecto al que se podría esperar viendo el ciclo de reloj y el ancho de los datos. Para alcanzar este ancho de banda, la DRAM se organiza internamente como una memoria entrelazada.

La ventaja de esta optimización es que se obtiene una mejora significativa en el ancho de banda utilizando mayoritariamente la circuitería ya disponible en la DRAM, con un pequeño coste añadido. En la sección C.9 en el **apéndice C** se describe la arquitectura interna de la DRAM y la implementación de estas optimizaciones.

Extensión: Los chips de memoria están organizados para proporcionar varios bits a su salida, normalmente entre 4 y 32, siendo 16 el valor más común en 2008. Describimos la organización de una RAM de $d \times w$, donde d es el número de posiciones de memoria direccionables (el rango de direccionamiento) y w es la salida (o ancho de cada posición de memoria). Las DRAMs están organizadas de forma lógica en matrices rectangulares, y su tiempo de acceso se divide en una parte que cuantifica el acceso a las filas y otra parte que cuantifica el acceso a las columnas. Las DRAMs utilizan un búfer para almacenar temporalmente una fila de bits en el interior de la DRAM para el posterior acceso a las columnas. También disponen de señales opcionales de temporización que permiten accesos sucesivos al búfer sin que se tenga que incurrir de nuevo en un tiempo de acceso a la fila. El búfer actúa como una SRAM; cambiando la dirección de la columna, se puede acceder a bits aleatorios hasta que se produzca el siguiente acceso a una fila. Esta posibilidad modifica significativamente el tiempo de acceso, ya que el tiempo de acceso a los bits de la fila es mucho más corto. La figura 5.12 muestra cómo la densidad, el coste económico y el tiempo de acceso de las DRAMs han cambiado a lo largo de los años.

Para mejorar la interfaz con los procesadores, se han añadido señales de reloj a las DRAM y se denominan de forma más apropiada DRAM Síncronas, o SDRAM. La ventaja de las SDRAM es que la presencia de una señal de reloj elimina el tiempo necesario para la sincronización del procesador y la memoria.

Extensión: Una manera de medir las prestaciones del sistema de memoria detrás de la cache son los programas de prueba Stream [McCalpin, 1995]. Miden las prestaciones de operaciones vectoriales largas. No tienen localidad temporal y acceden a vectores y matrices que son mayores que la cache del computador que se está probando.

Extensión: La estructura mejorada que se ha descrito para la memoria DRAM, *burst access*, se utiliza también en los buses de memoria, por ejemplo, en el bus Intel Duo Core Front Side.

Año de aparición	Capacidad del chip	Dólares por GB	Tiempo total de acceso a una nueva fila/columna	Tiempo de acceso a una fila existente
1980	64 Kbit	$1 500 000	250 ns	150 ns
1983	256 Kbit	$500 000	185 ns	100 ns
1985	1 Mbit	$200 000	135 ns	40 ns
1989	4 Mbit	$50 000	110 ns	40 ns
1992	16 Mbit	$15 000	90 ns	30 ns
1996	64 Mbit	$10 000	60 ns	12 ns
1998	128 Mbit	$4 000	60 ns	10 ns
2000	256 Mbit	$1 000	55 ns	7 ns
2004	512 Mbit	$250	50 ns	5 ns
2007	1 Gbit	$50	40 ns	1.25 ns

FIGURA 5.12 Hasta 1996, la capacidad de las DRAMs se cuadruplicaba cada tres años aproximadamente, y desde entonces el aumento es mucho menor. Las mejoras en el tiempo de acceso han sido más lentas pero continuas, y el coste económico casi sigue la trayectoria marcada por las mejoras en la densidad del chip, aunque al coste a menudo le afectan otros aspectos como la disponibilidad y la demanda. En el coste por megabyte no se ha considerado la inflación.

Resumen

Comenzamos la sección anterior examinando las caches más sencillas: una cache de correspondencia directa con un bloque de una palabra. En esta cache, tanto los aciertos como los fallos son sencillos, ya que una palabra puede alojarse sólo en una posición y existe una etiqueta distinta para cada palabra. Para mantener la coherencia entre la cache y la memoria principal, se puede usar el método de escritura directa, tal que cada escritura en la cache obliga a actualizar la memoria principal. La alternativa a la escritura directa es método de escritura retardada que guarda un bloque en memoria principal cuando es reemplazado en la cache; este método se abordará con mayor detalle en las siguientes secciones.

Para aprovechar la localidad espacial, una cache debe tener un tamaño de bloque superior a una palabra. El uso de bloques más grandes disminuye la frecuencia de fallos y mejora la eficiencia de la cache al reducir el almacenamiento de etiquetas en relación con la cantidad de almacenamiento de datos en la cache. Aunque un mayor tamaño de bloque disminuye la frecuencia de fallos, la penalización por fallo también se puede incrementar. Si la penalización por fallo aumenta linealmente con el tamaño del bloque, bloques más grandes podrían fácilmente ocasionar peores prestaciones.

Para evitar esto, el ancho de banda de la memoria principal se aumenta para transferir bloques de cache más eficientemente. Los métodos que se utilizan normalmente para incrementar el ancho de banda de la DRAM consisten en aumentar el ancho de la memoria y en utilizar el entrelazamiento. Los diseñadores de DRAM han mejorado constantemente la interfaz entre el procesador y la memoria para aumentar el ancho de banda del modo de transferencia rápido (*burst mode*) y reducir el coste de tamaños de bloque de cache más grandes.

Autoevaluación

La frecuencia del sistema de memoria influye en la decisión que debe tomar el diseñador sobre el tamaño del bloque de cache. ¿Cuáles de las siguientes directrices del diseñador de caches son generalmente válidas?

1. A medida que el tamaño del bloque de cache se hace más pequeño, la latencia de memoria disminuye.
2. A medida que el tamaño del bloque de cache se hace más grande, la latencia de memoria disminuye.
3. A medida que el tamaño del bloque de cache se hace más pequeño, el ancho de banda de la memoria es mayor.
4. A medida que el tamaño del bloque de cache se hace más grande, el ancho de banda de la memoria es mayor.

5.3 Evaluación y mejora de las prestaciones de la cache

En esta sección comenzamos describiendo cómo evaluar y analizar las prestaciones de la cache. Luego estudiaremos dos técnicas distintas para mejorar las prestaciones de la cache. Una de ellas se centra en reducir la frecuencia de fallos disminuyendo la probabilidad de que dos bloques de memoria distintos compitan por la misma posición de cache. La segunda técnica reduce la penalización por fallo añadiendo un nivel más a la jerarquía. Esta técnica, denominada *cache multinivel*, apareció por primera vez en computadores de altas prestaciones que se vendían en 1990 por unos 100 000 dólares. Desde entonces, su uso se ha extendido a computadores de sobremesa que se venden por menos de 500 dólares.

El tiempo de CPU se puede separar en los ciclos de reloj que la CPU utiliza para ejecutar el programa y en los ciclos de reloj que se requieren cuando la CPU espera a la memoria principal. Normalmente se supone que los tiempos de acceso que corresponden a aciertos forman parte de los ciclos normales de ejecución de la CPU. De este modo,

$$\begin{aligned}\text{Tiempo CPU} = (&\text{Ciclos de reloj de la ejecución de la CPU} + \\ &\text{Ciclos de reloj de parada debido a la memoria}) \times \\ &\text{Tiempo del ciclo de reloj}\end{aligned}$$

Los ciclos de reloj de parada debido a la memoria se deben principalmente a los fallos de cache, y así lo suponemos de aquí en adelante. También restringimos la exposición a un modelo simplificado del sistema de memoria. En procesadores reales, las paradas originadas por lecturas y escrituras pueden ser bastante complejas, y la predicción precisa de las prestaciones normalmente requiere simulaciones muy detalladas del procesador y del sistema de memoria.

Los ciclos de reloj de parada debido a la memoria pueden ser definidos como la suma de los ciclos de parada que se originan por las lecturas más los que se producen por las escrituras:

$$\begin{aligned}\text{Ciclos de reloj de parada debido a la memoria} = &\text{Ciclos de reloj de parada por} \\ &\text{lecturas} + \text{Ciclos de reloj de parada por escrituras}\end{aligned}$$

Los ciclos de reloj de parada por lecturas pueden ser definidos en términos del número de accesos de lectura por programa, de la penalización por fallos en ciclos de reloj para una lectura y de la frecuencia de fallos de las lecturas:

$$\text{Ciclos reloj de parada debido a lecturas} = \frac{\text{Número lecturas}}{\text{Programa}} \times \text{Frecuencia de fallos de lecturas} \times \text{Penalización por fallo lectura}$$

Las escrituras son más complicadas. Para la técnica de escritura directa, tenemos dos fuentes que originan paradas: los fallos de escritura, los cuales normalmente requieren que se busque el bloque antes de continuar la escritura (véase la Extensión de la página 467 para conocer más detalles sobre el manejo de las escrituras), y las paradas debidas al búfer de escritura, las cuales se producen cuando el búfer de escritura está lleno y aparece una nueva escritura. De este modo, los ciclos de parada para las escrituras se calculan con la siguiente expresión:

$$\text{Ciclos de bloqueo por escritura} = \left(\frac{\text{Escrituras}}{\text{Programa}} \times \text{Frecuencia fallos de escritura} \times \text{Penalización por fallo de escritura}\right) + \text{Bloqueo por búfer de escritura}$$

Debido a que las paradas del búfer de escritura dependen de la concentración de escrituras a lo largo del tiempo, y no precisamente de la frecuencia, no es posible proporcionar una ecuación sencilla que calcule tales penalizaciones. Afortunadamente, en los sistemas que disponen de un búfer de escritura de tamaño razonable (p. ej., cuatro o más palabras) y una memoria capaz de aceptar un ritmo de operaciones de escritura que exceda significativamente al ritmo promedio de ejecución de operaciones de escritura en los programas (p. ej., el doble), las penalizaciones originadas por el búfer de escritura serán pequeñas, y se pueden ignorar sin peligro alguno. Si un sistema no cumple con estos criterios, no estaría bien diseñado. En lugar de ello, el diseñador debe haber usado, o bien un búfer de escritura más grande, o bien una organización basada en escrituras retardadas.

Las técnicas de escritura retardada también pueden sufrir penalizaciones adicionales originadas por la necesidad de escribir un bloque de cache en la memoria cuando éste es reemplazado. Desarrollaremos esto en la sección 5.5.

En la mayoría de las organizaciones de cache de escritura directa, las penalizaciones por fallos de lectura y escritura son iguales (el tiempo necesario para buscar un bloque de la memoria). Si suponemos que las paradas debidas al búfer de escritura son insignificantes, podemos combinar las lecturas y escrituras utilizando una única frecuencia de fallos y la penalización por fallo:

$$\text{Ciclos de reloj de parada debidos a la memoria} = \frac{\text{Acceso a memoria}}{\text{Programa}} \times \text{Frecuencia de fallos} \times \text{Penalización por fallo}$$

Esta ecuación se puede transformar en esta otra:

$$\text{Ciclos de reloj de parada debidos a la memoria} = \frac{\text{Instrucciones}}{\text{Programa}} \times \frac{\text{Fallos}}{\text{Instrucciones}} \times \text{Penalización por fallo}$$

Consideraremos un ejemplo sencillo para ayudarnos a entender el impacto de las prestaciones de la cache sobre las prestaciones del procesador.

Cálculo de las prestaciones de la cache

EJEMPLO

Suponga que en un programa la frecuencia de fallos de la cache de instrucciones es el 2%, y la frecuencia de fallos de la cache de datos es el 4%. Si un procesador tiene un CPI igual a 2 sin incluir paradas debidas a la memoria, y la penalización por fallo es de 100 ciclos para todo tipo de fallo, determinar cuánto más rápido es un procesador cuya cache es perfecta, es decir, que nunca falla. Suponga que la frecuencia de las instrucciones de carga y almacenamiento es el 36%.

RESPUESTA

El número de ciclos por fallo de memoria debidos a los accesos a instrucciones en términos del número de instrucciones (I) es

$$\text{Ciclos debidos a fallos por accesos a instrucciones} = I \times 2\% \times 100 = 2.00 \times I$$

Como la frecuencia de instrucciones de carga y almacenamiento es el 36%, podemos calcular el número de ciclos por fallo de memoria cuando se acceden a los datos:

$$\text{Ciclos debidos a fallos por accesos a datos} = I \times 36\% \times 4\% \times 100 = 1.44 \times I$$

El número total de ciclos de parada por accesos a memoria es 2.00 I + 1.44 I = 3.44 I. Este valor indica que se producen más de 3 ciclos de parada por instrucción. Por consiguiente, el CPI con paradas por accesos a memoria es 2 + 3.44 = 5.44. Ya que no se modifica el número de instrucciones o la frecuencia de reloj, la relación de tiempos de CPU es

$$\frac{\text{Tiempo CPU con parada}}{\text{Tiempo CPU con cache perfecta}} = \frac{I \times \text{CPI}_{\text{paradas}} \times \text{Ciclo de reloj}}{I \times \text{CPI}_{\text{perfecta}} \times \text{Ciclo de reloj}}$$

$$= \frac{\text{CPI}_{\text{parada}}}{\text{CPI}_{\text{perfecta}}} = \frac{5.44}{2}$$

Las prestaciones con la cache perfecta son mejores en un factor $\frac{5.44}{2} = 2.72$.

¿Qué ocurre si el procesador fuera más rápido, pero el sistema de memoria no? La cantidad de tiempo que se consume en paradas por memoria constituirá un porcentaje del tiempo de ejecución que será cada vez mayor; la ley de Amdahl, que se estudió en el capítulo 1, nos justifica este hecho. Unos pocos ejemplos sencillos muestran la gravedad de este problema. Suponga que aumentamos las prestaciones del computador del ejemplo anterior mediante la reducción de su CPI de 2 a 1 sin modificar la frecuencia de reloj, lo cual se podría conseguir mejorando el camino de datos segmentado. El sistema con fallos de cache obtendría un CPI de 1 + 3.44 = 4.44, y el sistema con cache perfecta sería

$$\frac{4.44}{1} = 4.44 \text{ veces más rápido}$$

El tiempo de ejecución que se origina en las paradas de memoria ha aumentado del

$$\frac{3.44}{5.44} = 63\%$$

al

$$\frac{3.44}{5.44} = 77\%$$

De manera semejante, el aumento de la frecuencia de reloj sin cambiar el sistema de memoria también incrementan las prestaciones que se pierden debido a los fallos de cache.

Los ejemplos y ecuaciones anteriores suponen que el tiempo de acierto no es un factor que se use para determinar las prestaciones de la cache. Obviamente, si el tiempo de acierto aumenta, el tiempo total para acceder a una palabra del sistema de memoria aumentará, causando posiblemente un aumento del tiempo de ciclo del procesador. Aunque dentro de poco veremos otros ejemplos de factores que aumentan el tiempo de acierto, un ejemplo es aumentar la capacidad de la cache. Una cache de mayor capacidad podría tener un tiempo de acceso mayor, ya que, al igual que si su mesa en la biblioteca fuera muy grande (de 3 metros cuadrados, por ejemplo), se necesitaría más tiempo para localizar uno de los libros de la mesa. Un aumento del tiempo de acierto probablemente requiere añadir otra etapa de segmentación, ya que un acierto de cache puede tardar varios ciclos. Aunque es más complejo calcular el impacto sobre las prestaciones en un procesador con un mayor número de etapas, en algún momento el aumento del tiempo de ciclo en una cache más grande podría dominar sobre la mejora en la frecuencia de aciertos, ocasionando que las prestaciones del procesador disminuyan.

Para incluir el hecho de que el tanto el tiempo de acceso a los datos en caso de acierto como en caso de fallo afecta a las prestaciones, los diseñadores utilizan a veces el *tiempo medio de acceso a memoria* (*average memory acces time*, *AMAT*) como una forma de analizar diseños alternativos de cache. El tiempo medio de acceso a memoria es el tiempo medio que se obtiene considerando tanto los fallos como los aciertos y la frecuencia de los diferentes accesos; es igual a

$$\text{AMAT} = \text{tiempo de un acierto} + \text{frecuencia de fallos} \times \text{penalización por fallo}$$

EJEMPLO

Cálculo del tiempo medio de acceso a memoria

Determinar el AMAT para un procesador con un ciclo de reloj de 1 ns, una penalización por fallo de 20 ciclos, una frecuencia de fallos de 0.05 fallos por instrucción y un tiempo de acceso a cache (incluyendo detección de acierto) de 1 ciclo. Suponga que las penalizaciones por fallo de lectura y escritura son iguales e ignorar otras paradas de escritura.

El tiempo medio de acceso a memoria por instrucción es

RESPUESTA

AMAT = tiempo de un acierto + frecuencia de fallos × penalización por fallo
= 1 0.005 × 20
= 2 ciclos

o 2 ns.

La siguiente subsección describe organizaciones alternativas de cache que disminuyen la frecuencia de fallos, pero a veces pueden aumentar el tiempo de acierto; otros ejemplos aparecen en las Falacias y errores habituales de la sección 5.11.

Reducción de los fallos de cache mediante un emplazamiento más flexible de los bloques

Hasta ahora hemos utilizado un método sencillo para emplazar los bloques: un bloque puede alojarse exactamente en un solo lugar de la cache. Como se mencionó previamente, se denomina *correspondencia directa* porque existe una asignación entre la dirección de memoria de un bloque y una única posición del nivel superior de la jerarquía. Existe realmente una amplia variedad de técnicas para realizar el emplazamiento de bloques. En un extremo está la correspondencia directa, donde un bloque puede ser emplazado en una única posición.

En el otro extremo se encuentra un método en el que un bloque puede ser emplazado en cualquier posición de la cache. Tal método se denomina **completamente asociativo** porque un bloque de memoria puede estar asociado a cualquiera de las entradas de la cache. Para encontrar un bloque determinado en una cache completamente asociativa, todas las entradas de la cache deben ser inspeccionadas, ya que un bloque puede encontrarse en cualquiera de ellas. Para que la búsqueda sea factible, se realiza en paralelo a través del comparador asociado a cada entrada de la cache. Estos comparadores incrementan significativamente el coste del hardware, haciendo que el emplazamiento completamente asociativo sea factible sólo para caches con un pequeño número de bloques.

Cache completamente asociativa: estructura de cache en la cual un bloque puede ser emplazado en cualquier posición de la cache.

La variedad de diseños que se encuentra entre la de correspondencia directa y la completamente asociativa se denomina **asociativa por conjuntos**. En una cache asociativa por conjuntos, existe un número establecido de posiciones donde cada bloque puede ser emplazado. Una cache asociativa por conjuntos con *n* posiciones posibles para un bloque se denomina cache *asociativa por conjuntos de n vías*. Una cache asociativa por conjuntos de *n* vías está formada por un número de conjuntos, cada uno de los cuales está constituido por *n* bloques. Cada bloque de la memoria se corresponde con un único *conjunto* de la cache que viene dado por el campo índice, y un bloque puede ser emplazado en cualquiera de los elementos de ese conjunto. De este modo, un emplazamiento aso-

Cache asociativa por conjuntos: cache que tiene un número establecido de posiciones (al menos dos) donde cada bloque puede ser emplazado.

ciativo por conjuntos combina emplazamiento de correspondencia directa con emplazamiento completamente asociativo: un bloque se corresponde con un conjunto, y todos los bloques del conjunto son inspeccionados para encontrar una posible coincidencia. Por ejemplo, la figura 5.13 muestra donde se emplazaría el bloque 12 en una cache con 8 bloques, según cada una de las tres estrategias de emplazamiento.

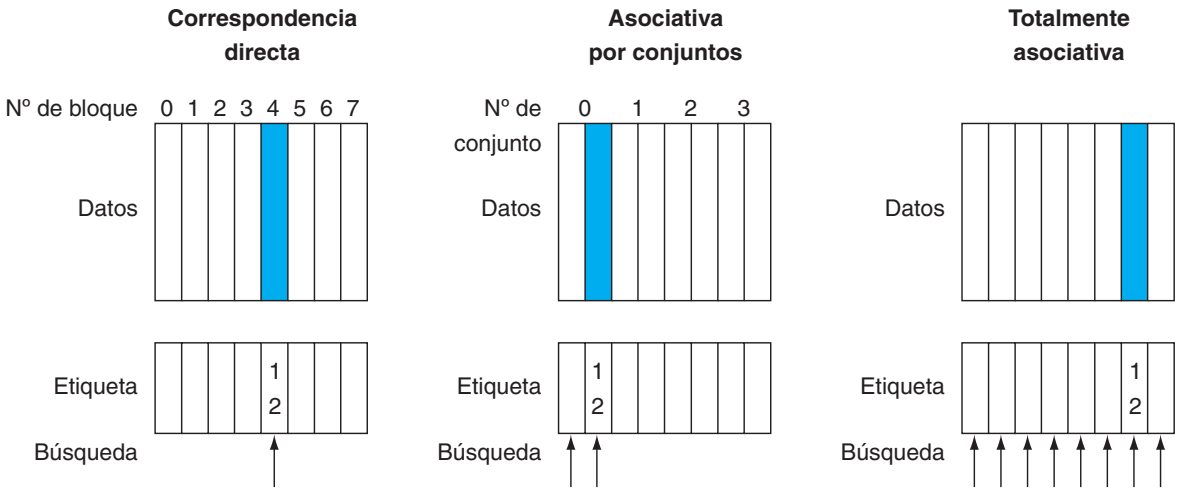

FIGURA 5.13 La posición de un bloque de memoria cuya dirección es 12 en una cache con 8 bloques varía según sea el emplazamiento: de correspondencia directa, asociativo por conjuntos o completamente asociativo. En el emplazamiento de correspondencia directa sólo existe un bloque de cache donde el bloque 12 de memoria puede ser encontrado, y ese bloque viene dado por (12 módulo 8) = 4. En una cache asociativa por conjuntos de dos vías, hay cuatro conjuntos, y el bloque 12 de memoria debe estar en el conjunto (12 mod 4) = 0; el bloque de memoria podría estar en cualquier elemento de ese conjunto. En un emplazamiento completamente asociativo, el bloque de memoria para la dirección 12 de bloque puede aparecer en cualquiera de los ocho bloques de cache.

Recuerde que en una cache de correspondencia directa la posición de un bloque de memoria viene dada por

(Número de bloque) módulo (Número de *bloques* de cache)

En una cache asociativa por conjuntos, el conjunto que contiene a un bloque de memoria viene dado por

(Número de bloque) módulo (Número de *conjuntos* de la cache)

Ya que el bloque puede ser emplazado en cualquier elemento del conjunto, *todas las etiquetas de todos los elementos del conjunto* deben ser inspeccionadas. En una cache completamente asociativa, el bloque puede estar en cualquier parte, y se deben analizar *todas las etiquetas de todos los bloques de la cache*.

Podemos considerar que cada estrategia de emplazamiento de bloques es una variación de la asociatividad por conjuntos. La figura 5.14 muestra las posibles estructuras asociativas para una cache de ocho bloques. Una cache de correspondencia directa es simplemente una cache asociativa por conjuntos de una vía: cada entrada de la cache guarda un bloque y cada conjunto tiene un único elemento. Una cache completamente asociativa con *m* entradas es simplemente una cache asociativa por conjuntos de *m* vías; tiene un único conjunto de *m* elementos, y un bloque puede alojarse en cualquiera de los elementos de ese único conjunto.

La ventaja de aumentar el grado de asociatividad consiste en que normalmente se produce una disminución de la frecuencia de fallos, como se muestra con el siguiente ejemplo. La principal desventaja, que discutiremos posteriormente con más detalle, consiste en que se incrementa el tiempo de acierto.

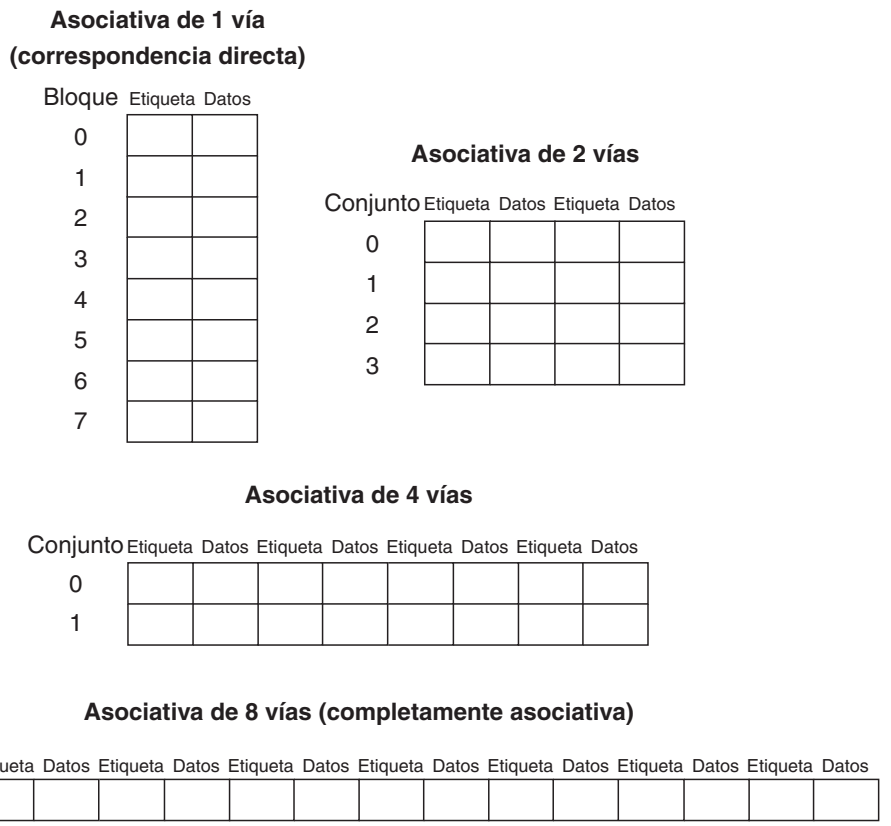

FIGURA 5.14 Una cache de ocho bloques configurada bien en correspondencia directa, asociativa por conjuntos de dos vías, asociativa por conjuntos de cuatro vías, o completamente asociativa. La capacidad total de la cache en bloques es igual al número de conjuntos multiplicado por la asociatividad. De esta forma, dado un tamaño de cache, el aumento de la asociatividad disminuye el número de conjuntos, mientras se aumenta el número de elementos en cada conjunto. Con ocho bloques, una cache asociativa por conjuntos de ocho vías es igual que una cache completamente asociativa.

EJEMPLO

Fallos y asociatividad en las caches

Suponga que se existen tres pequeñas caches, cada una de ellas formada por cuatro bloques de una palabra. La primera de las caches es completamente asociativa, la segunda es asociativa por conjuntos de dos vías, y la tercera es de correspondencia directa. Encuentre el número de fallos en cada organización de cache para la siguiente secuencia de direcciones de bloque: 0, 8, 0, 6, 8.

RESPUESTA

El caso de la cache de correspondencia directa es el más sencillo. Primero, determinemos a qué bloque de cache se corresponde cada dirección de bloque:

Dirección de bloque	Bloque de cache
0	(0 módulo 4) = 0
6	(6 módulo 4) = 2
8	(8 módulo 4) = 0

Ahora se puede completar el contenido de la cache después de que cada acceso se haya realizado. Utilizaremos el convenio que indica que una entrada en blanco significa que el bloque no es válido; si el texto está coloreado, indica que una nueva entrada de cache que se ha asociado a un acceso se ha actualizado; y si el texto está en negro, indica que la entrada de la cache es antigua.

Dirección del bloque de memoria accedido	Acierto o fallo	Contenido de los bloque de cache después de cada acceso			
		0	1	2	3
0	fallo	Memorial[0]			
8	fallo	Memorial[8]			
0	fallo	Memorial[0]			
6	fallo	Memorial[0]		Memorial[6]	
8	fallo	Memorial[8]		Memorial[6]	

La cache de correspondencia directa ocasiona cinco fallos en los cinco accesos.

La cache asociativa por conjuntos tiene dos conjuntos (con índice 0 y 1) con dos elementos por conjunto. Sepamos primero a qué conjunto corresponde cada dirección de bloque:

Dirección de bloque	Conjunto de la cache
0	(0 módulo 2) = 0
6	(6 módulo 2) = 0
8	(8 módulo 2) = 0

Como tenemos que elegir el elemento del conjunto que se va a sustituir cuando se produce un fallo, necesitamos establecer una política de reemplazos. Las caches asociativas por conjuntos normalmente reemplazan el bloque del conjunto que se ha usado menos recientemente; esto es, se reemplaza el bloque que fue utilizado por última vez hace más tiempo. (Seguidamente discutiremos las políticas de reemplazos con más detalle). Utilizando esta política para los reemplazos, el contenido de la cache asociativa por conjuntos después de cada acceso es el siguiente:

Dirección del bloque de memoria accedido	Acierto o fallo	Contenido de los bloque de cache después de cada acceso			
		Conjunto 0	Conjunto 0	Conjunto 1	Conjunto 1
0	fallo	Memorial[0]			
8	fallo	Memorial[0]	Memorial[8]		
0	acierto	Memorial[0]	Memorial[8]		
6	fallo	Memorial[0]	Memorial[6]		
8	fallo	Memorial[8]	Memorial[6]		

Observe que cuando el bloque 6 es accedido, reemplaza al bloque 8, ya que el bloque 8 ha sido accedido menos recientemente que el bloque 0. La cache asociativa por conjuntos de dos vías experimenta cuatro fallos, uno menos que la cache de correspondencia directa.

La cache completamente asociativa tiene cuatro bloques de cache (en un solo conjunto); cualquier bloque de memoria puede ser almacenado en cualquier bloque de cache. La cache completamente asociativa tiene las mejores prestaciones, con sólo tres fallos:

Dirección del bloque de memoria accedido	Acierto o fallo	Contenido de los bloque de cache después de cada acceso			
		Bloque 0	Bloque 1	Bloque 2	Bloque 3
0	fallo	Memorial[0]			
8	fallo	Memorial[0]	Memorial[8]		
0	acierto	Memorial[0]	Memorial[8]		
6	fallo	Memorial[0]	Memorial[8]	Memorial[6]	
8	acierto	Memorial[0]	Memorial[8]	Memorial[6]	

Para este conjunto de accesos, tres fallos es lo menos que podemos conseguir porque se accede a tres direcciones distintas de bloques. Observe que si tuviéramos ocho bloques en la cache, no habría reemplazos en la cache asociativa por conjuntos de dos vías (compruebe esto usted mismo), y se experimentaría el mismo número de fallos que con la cache completamente asociativa. De manera parecida, si tuviéramos 16 bloques, las tres caches experimentarían el mismo número de fallos. Este cambio en frecuencia de fallos nos indica que la capacidad y la asociatividad de la cache influyen sobre las prestaciones de la cache.

¿Qué parte de la reducción de la frecuencia de fallos se debe a la asociatividad? La figura 5.15 muestra las mejoras obtenidas en una cache de datos de 64 KB, bloques de 16 palabras y una asociatividad que va desde la correspondencia directa hasta las ocho vías. Pasando de una asociatividad de una vía a dos vías, la frecuencia de fallos disminuye en un 15% aproximadamente, pero existe un poco más de mejora cuando se pasa a una mayor asociatividad.

Localización de un bloque en la cache

Consideremos ahora la tarea de encontrar un bloque en una cache asociativa por conjuntos. Igual que en una cache de correspondencia directa, cada bloque de una cache asociativa por conjuntos incluye una etiqueta de dirección que viene determinada por la dirección del bloque. La etiqueta de cada bloque de cache dentro del conjunto apropiado es inspeccionada para comprobar si se corresponde con la dirección del bloque que envía el procesador. La figura 5.16 muestra cómo se descompone la dirección. El valor del índice se usa para seleccionar el conjunto que contiene la dirección de interés, y las etiquetas de todos los bloques del conjunto son inspeccionadas. Como el tiempo de ejecución es primordial, todas las etiquetas de todos los bloques del conjunto se inspeccionan en paralelo. Al igual que en una cache completamente asociativa, una búsqueda secuencial ocasionaría que el tiempo de acierto de una cache asociativa por conjuntos fuera muy largo.

Si la capacidad total de la cache no cambia, el aumento de la asociatividad aumenta el número de bloques por conjunto, el cual coincide con el número de comparaciones simultáneas que son necesarias para realizar la búsqueda en paralelo: cada incremento de la asociatividad en un factor dos, dobla el número de bloques por conjunto y reduce a la mitad el número de conjuntos. Por consiguiente, cada incremento de la asociatividad en un factor dos, disminuye el tamaño del índice en 1 bit e incrementa el tamaño de la etiqueta en 1 bit. En una cache completamente asociativa sólo existe efectivamente un conjunto, y todos los bloques deben ser comprobados en paralelo. De este modo, no existe índice, y la dirección entera,

Asociatividad	Frecuencia de fallos de datos
1	10.3%
2	8.6%
4	8.3%
8	8.1%

FIGURA 5.15 Frecuencias de fallos de la cache de datos organizada como la del procesador Fast-MATH de Intrinsity y probada con programas SPEC2000 con asociatividades que varían desde una vía a ocho vías. Los resultados de los 10 programas SPEC200 fueron extraídos del libro Hennessy y Patterson [2003].

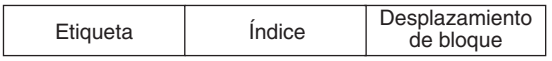

FIGURA 5.16 Las tres partes de una dirección de una cache asociativa por conjuntos o de correspondencia directa. El índice se usa para seleccionar el conjunto, y la etiqueta se usa para elegir el bloque después de compararla con las etiquetas de los bloques del conjunto seleccionado. El desplazamiento del bloque es la dirección deseada de los datos dentro del bloque.

excluyendo el desplazamiento del bloque, es comparada con la etiqueta de cada bloque. En otras palabras, se inspecciona toda la cache sin que exista indexación.

En una cache de correspondencia directa, se necesita un solo comparador, debido a que el dato solicitado puede encontrarse en un único bloque de la cache, al cual se accede indexándolo. La figura 5.17 muestra que en una cache asociativa por conjuntos de cuatro vías, se necesitan cuatro comparadores, además de un multiplexor 4 a 1 que realiza la elección entre los cuatro componentes del conjunto seleccionado. El acceso a la cache se realiza indexando el conjunto apropiado y después inspeccionando sus etiquetas. Los costes adicionales de una cache asociativa corresponden a los comparadores que se han añadido y a los retardos temporales impuestos por la necesidad de realizar la comparación y seleccionar entre los elementos del conjunto.

La elección entre correspondencia directa, asociativa por conjuntos o completamente asociativa en cualquier jerarquía de memoria dependerá del resultado de sopesar entre el coste asociado a un fallo y el coste de la implementación de la asociatividad, tanto en tiempo con en hardware adicional.

Extensión: Una *memoria direccionable por contenido (content addressable memory, CAM)* es un circuito que combina comparaciones y almacenamientos en un único dispositivo. En lugar enviar una dirección y leer la palabra almacenada en esa posición, como ocurre en las RAM, se envían el dato y la CAM comprueba si tiene una copia y devuelve el índice la fila que contiene la copia. Con las CAM, el diseñador de caches puede abordar la implementación de conjuntos de asociatividad mucho mayores que teniendo que construir el hardware a partir de SRAM y comparadores. En 2008, el mayor tamaño y consumo de potencia de las CAM, ha llevado a que generalmente los conjuntos de asociatividad de 2 y 4 vías se construyan a partir de SRAM y comparadores, y que los conjuntos de 8 o más vías se construyan con CAM.

Elección del bloque a reemplazar

Cuando se produce un fallo en una cache de correspondencia directa, el bloque solicitado sólo puede guardarse en una única posición, y por ello, el bloque que ocupaba esa posición debe ser reemplazado. En una cache asociativa tenemos la posibilidad de elegir dónde guardar el bloque solicitado, ya que podemos elegir qué bloque sustituir. En una cache asociativa por conjuntos, se puede elegir entre los bloques del conjunto seleccionado.

La política que normalmente se utiliza es la de **menos recientemente usado** (LRU), que es la que utilizamos en el ejemplo anterior. En una política LRU, el bloque reemplazado es el que no ha sido utilizado por el periodo de tiempo más largo. El ejemplo de conjunto de asociatividad de la página 482 utiliza la estrategia LRU, motivo por el que se reemplaza Memoria(0) en lugar de Memoria(6).

Menos recientemente usado (*Least recently used*, LRU): política de reemplazamientos en la cual el bloque reemplazado corresponde al que no ha sido usado por el periodo de tiempo más largo.

La implementación de reemplazos LRU consiste en realizar el seguimiento de cuándo se utiliza cada elemento del conjunto en relación con el resto de elementos de ese mismo conjunto. Para una cache asociativa por conjuntos de dos vías, podemos saber cuándo fueron utilizados los dos elementos del conjunto manteniendo un único bit en cada conjunto que indique el elemento referenciado en cada acceso. A medida que la asociatividad aumenta, la implementación de la LRU se complica. En la sección 5.5 describiremos una política de reemplazos alternativa.

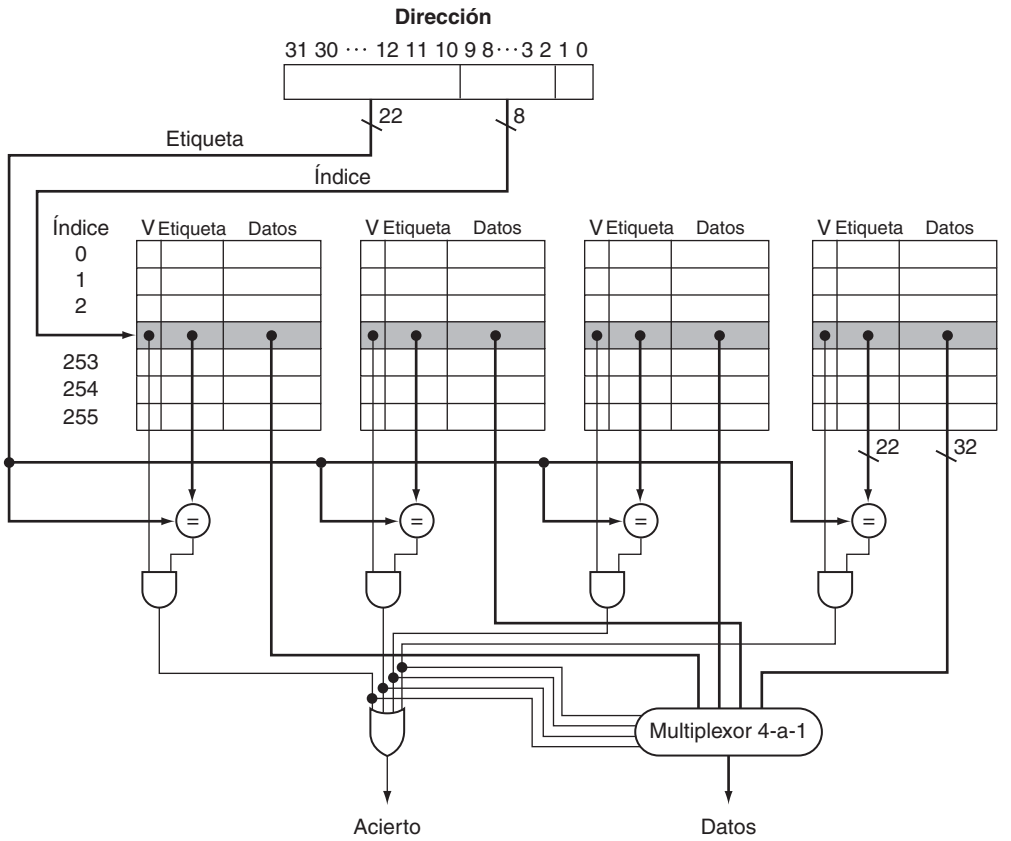

FIGURA 5.17 La implementación de una cache asociativa por conjuntos de cuatro vías requiere cuatro comparadores y un multiplexor 4 a 1. Los comparadores determinan qué elemento del conjunto seleccionado (si existe) tiene la misma etiqueta. La salida de los comparadores se utiliza para seleccionar el dato entre los cuatro bloques del conjunto indexado, haciendo uso de un multiplexor y de una señal de selección que se descodifica. En algunas implementaciones, las señales de habilitación de los datos de salida de las RAMs que constituyen la cache pueden ser usadas para seleccionar el elemento del conjunto y dirigirlo hacia la salida. La señal de habilitación de la salida se genera en los comparadores, ocasionando que el elemento seleccionado dirija el dato hacia la salida. Esta organización hace innecesaria la existencia del multiplexor.

EJEMPLO

Tamaño de las Etiquetas respecto la asociatividad por conjuntos

Incrementar la asociatividad significa que se necesitan más comparadores y más bits de etiqueta por bloque de cache. Suponiendo una cache de 4K bloques, un tamaño de bloque de cuatro palabras y una dirección de 32 bits, obtenga el número total de conjuntos y la cantidad de bits utilizados en las etiquetas de las caches de correspondencia directa, asociativas por conjunto de dos y cuatro vías, y completamente asociativa.

RESPUESTA

Ya que existen 16 ($= 2^4$) bytes en cada bloque, una dirección de 32 bits tiene asociada 32 – 4 = 28 bits para índice y etiqueta. La cache de correspondencia directa tiene el mismo número de conjuntos que de bloques, y de ahí que 12 bits se destinen al índice, ya que $\log_2(4K) = 12$. Por lo tanto, la cantidad total de bits dedicados a las etiquetas es (28 – 12) × 4K = 16 × 4K = 64 Kbits.

Cada nivel de asociatividad disminuye el número de conjuntos en un factor dos, y de este modo se disminuye en uno el número de bits dedicados al índice de la cache y se incrementa en uno el número de bits destinados a la etiqueta. En consecuencia, para una cache asociativa por conjuntos de dos vías existen 2K conjuntos, y el número de bits de etiquetas es (28 –11) × 2 × 2K = 34 × 2K = 68 Kbits. Para una cache asociativa por conjuntos de cuatro vías, el número total de conjuntos es 1K, y el número total de bits de etiquetas es (28 – 10) × 4 × 1K = 72 × 1K = 72 Kbits.

Para una cache completamente asociativa existe un solo conjunto con 4K bloques, y la etiqueta es de 28 bits, requiriéndose para las etiquetas un total de 28 × 4K × 1 = 112 Kbits.

Reducción de la penalización por fallo utilizando caches multinivel

Todos los computadores modernos hacen uso de caches. Para reducir aún más el hueco existente entre la alta frecuencia de reloj de los procesadores modernos y el largo tiempo que se requiere para acceder a las DRAMs, muchos microprocesadores se apoyan en un nivel adicional de cache. Este segundo nivel de cache se encuentra en el mismo chip y se accede cuando se produce un fallo en la cache principal. Si el segundo nivel de cache contiene el dato deseado, la penalización por fallo en el primer nivel de la cache será el tiempo de acceso al segundo nivel de la cache, que es mucho menor que el tiempo de acceso a la memoria principal. Si los datos no se encuentran ni en la cache principal ni en la secundaria, se realiza un acceso a la memoria principal, y se sufre una penalización más larga por fallo.

¿Cuál es la mejora de las prestaciones cuando se utiliza una cache secundaria? El próximo ejemplo nos lo muestra.

EJEMPLO

Prestaciones de las cache multinivel

Suponga que tenemos un procesador con un CPI base de 1.0, e imagine que todas las referencias aciertan en la cache principal y que la frecuencia de reloj es de 4 GHz. Tome un tiempo de acceso a la memoria principal de 100 ns, incluido todo el manejo de fallos. Suponga que la frecuencia de fallos por instrucción en la cache principal es del 2%. ¿Cuánto más rápido será el procesador si añadiéramos una cache secundaria que tiene un tiempo de acceso de 5 ns tanto para un acierto como para un fallo, y es lo suficientemente largo como para reducir la frecuencia de fallos de la memoria principal a 0.5%?

RESPUESTA

La penalización por fallo cuando se requiere acceder a la memoria principal es

$$\frac{100 \text{ ns}}{0.25 \frac{\text{ns}}{\text{ciclo de reloj}}} = 400 \text{ ciclos de reloj}$$

El CPI efectivo con un único nivel de cache viene dado por

CPI Total = CPI Base + Ciclos de parada por instrucción debido a memoria

Para el procesador con un nivel de cache,

CPI Total = 1.0 + Ciclos de parada por instrucción debido a la memoria = 1.0 + 2% × 400 = 9

Con dos niveles de cache, un fallo en la cache principal puede ser resuelto bien por la cache secundaria o por la memoria principal. La penalización por fallo para un acceso al segundo nivel de la cache es

$$\frac{5 \text{ ns}}{0.25 \frac{\text{ns}}{\text{ciclo de reloj}}} = 20 \text{ ciclos de reloj}$$

Si el fallo se resuelve en la cache secundaria, entonces ésta es toda la penalización por fallo. Si el fallo requiere que se acceda a la memoria principal, entonces la penalización total por fallo es la suma del tiempo de acceso a la cache secundaria y el tiempo de acceso a la memoria principal.

De este modo, para una cache de dos niveles, el CPI total es la suma de los ciclos de parada originados en ambos niveles de cache y el CPI base:

CPI Total = 1 + Ciclos de parada por instrucción en la cache principal
+ Ciclos de parada por instrucción en la cache secundaria
= 1 + 2% × 20 + 0.5% × 400 = 1 + 0.4 + 2.0 = 3.4

De este modo, el procesador con la cache secundaria es más rápido en un factor

$$\frac{9.0}{3.4} = 2.6$$

Alternativamente, podríamos haber calculado los ciclos de parada sumando los ciclos de paradas de aquellas referencias que aciertan en la cache secundaria ((2% – 0.5%) × 20 = 0.3) más los originados por los accesos a la memoria principal, los cuales deben incluir el tiempo de acceso a la cache secundaria así como el tiempo de acceso a la memoria principal (0.5% × (20 + 400) = 2.1). La suma, 1.0 + 0.3 + 2.1, de nuevo resulta ser 3.4.

Las decisiones de diseño que se toman para una cache primaria y una cache secundaria son significativamente diferentes, porque la presencia de la otra cache cambia la mejor elección respecto a una cache con un único nivel. En particular, una estructura de dos niveles permite que la cache principal se oriente a minimizar el tiempo de acierto para conseguir un ciclo de reloj lo más pequeño posible, mientras que la cache secundaria se orienta a reducir su frecuencia de fallos para reducir la penalización por fallo debida a los largos tiempos que se requieren para acceder a la memoria principal.

Comprender las prestaciones de los programas

Los métodos de ordenación se han analizado exhaustivamente para encontrar mejores algoritmos: Bubble Sort, Quicksort, Radix Sort, etc. La figura 5.18(a) muestra las instrucciones ejecutadas por elemento buscado tanto en Radix Sort como en Quicksort. De hecho, para matrices grandes, Radix Sort aventaja algorítmicamente a Quicksort en cuanto al número de operaciones. La figura 5.18(b) muestra el tiempo por clave en vez de instrucciones ejecutadas. Se observa que las líneas comienzan con la misma trayectoria que se refleja en la figura 5.18(a), pero en un momento determinado la línea de Radix Sort cambia de tendencia a medida que aumenta el volumen de datos a clasificar. ¿Qué ocurre? La figura 5.18(c) contesta a esta pregunta analizando los fallos de la cache por elemento clasificado: Quicksort tiene sistemáticamente menos cantidad de fallos por elemento que tiene que ser clasificado.

Desafortunadamente, el análisis algorítmico convencional ignora el impacto de la jerarquía de memoria. A medida que las frecuencias de reloj y la ley de Moore permiten a los arquitectos aprovechar la ejecución de instrucciones para aumentar las prestaciones, la utilización apropiada de la jerarquía de memoria es crítica para alcanzar altas prestaciones. Como expresamos en la introducción, la comprensión del comportamiento de la jerarquía de memoria es crítica para comprender las prestaciones de los programas en los computadores de hoy en día.

El efecto de estos cambios sobre los dos niveles de cache puede hacerse visible cuando se comparan con el diseño óptimo de un único nivel de cache. Cuando se compara con un único nivel de cache, la cache principal de una **cache multinivel** es a menudo más pequeña. Además, la cache principal utiliza a menudo un tamaño más pequeño de bloque, para acompañar a la reducción tanto del tamaño de cache como de la penalización por fallo. En comparación, la cache secundaria será a menudo de mayor capacidad que la cache de un solo nivel, ya que el tiempo de acceso de la cache secundaria es menos crítico. Con una capacidad total mayor, la cache secundaria a menudo usará un tamaño de bloque más grande que el apropiado para una cache de nivel único. Dado que el objetivo es reducir la frecuencia de fallos, a menudo utiliza una asociatividad mayor que la cache primaria.

Cache multinivel: jerarquía de memoria donde existen varios niveles de cache, en lugar de una sola y la memoria principal.

Extensión: Las caches multinivel originan varias complicaciones. En primer lugar, se producen distintos tipos de fallos con sus correspondientes frecuencias de fallos. En el ejemplo de la página 482 analizamos la frecuencia de fallos de la cache principal y la frecuencia de fallos global: el porcentaje de referencias a memoria que fallaron en todos los niveles de cache. Existe también una frecuencia de fallos para la cache secundaria, que se obtiene como el cociente entre todos los fallos en la cache secundaria y el número de sus accesos. Esta frecuencia de fallos se denomina frecuencia de fallos local de la cache secundaria. Puesto que la cache principal filtra los accesos, especialmente aquellos con buena localidad tanto espacial como temporal, la frecuencia de fallos local de la cache secundaria es mucho más alta que la frecuencia de fallos global. Para el ejemplo de la página 482, podemos calcular la frecuencia de fallos local de la cache secundaria como: 0.5% / 2% = 25%! Afortunadamente, la frecuencia de fallos global impone con qué frecuencia debemos acceder a la memoria principal.

Frecuencia de fallos global: fracción de accesos que fallan en todos los niveles de cache.

Frecuencia de fallos local: fracción de accesos a un nivel de cache que fallan; utilizada en caches multinivel.

Extensión: Con procesadores fuera de orden (véase capítulo 4), el análisis de las prestaciones es más complejo, ya que se ejecutan instrucciones durante el tiempo de

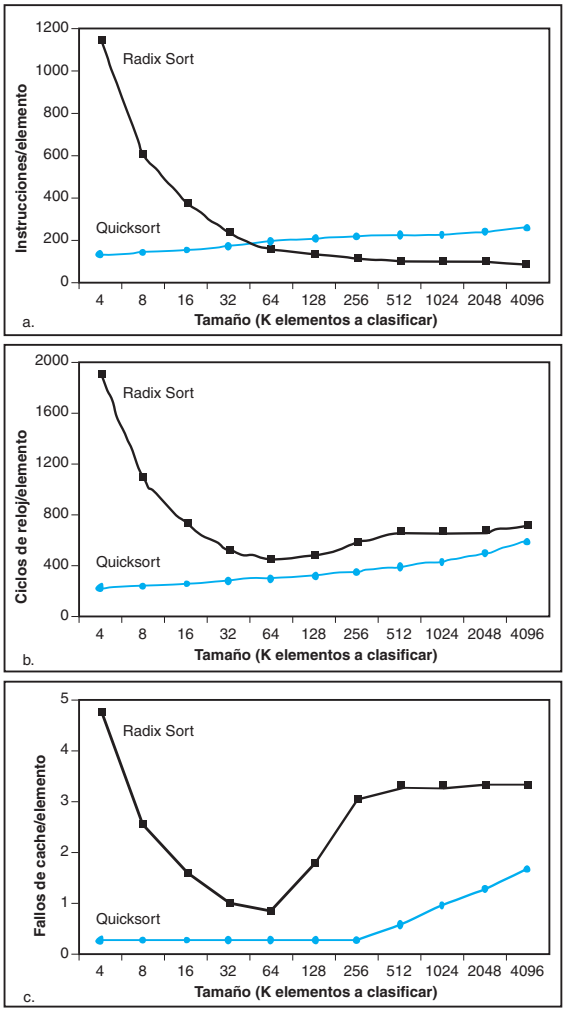

FIGURA 5.18 Comparación de Quicksort y Radix Sort utilizando (a) las instrucciones ejecutadas por elemento clasificado, (b) el tiempo por elemento clasificado, y (c) los fallos de cache por elemento clasificado. Estos datos fueron obtenidos del artículo de LaMarca y Ladner [1996]. Aunque los números pueden cambiar si se utilizan computadores más modernos, la idea sigue siendo válida. A causa de estos resultados, se inventaron nuevas versiones de Radix Sort que tienen en cuenta la jerarquía de memoria, con el objetivo de recuperar sus ventajas algorítmicas (véase la sección 5.11). La idea básica de las optimizaciones de la cache consiste en usar repetidamente los datos de un bloque antes de reemplazarlo cuando se produzca un fallo.

penalización de un fallo. En vez de frecuencia de fallos de instrucciones y frecuencia de fallo de datos, utilizamos fallos por instrucción, además de esta fórmula:

$$\frac{\text{Ciclos de parada debido a la memoria}}{\text{Instrucción}} =$$

$$\frac{\text{Fallos}}{\text{Instrucción}} \times \text{Latencia total por fallo} - \text{Latencia solapada por fallo}$$

No existe una manera generalizada de calcular la latencia solapada por fallo, por lo que el análisis de las jerarquías de memoria en procesadores fuera de orden requiere inevitablemente que el procesador y la jerarquía de memoria se simulen. Solamente observando la ejecución del procesador durante cada fallo podremos ver si el procesador se para esperando a que lleguen los datos o simplemente encuentra otra tarea que realizar. Frecuentemente, el procesador puede solapar la penalización ocasionada por un fallo en la cache L1 que acierta en la cache L2, pero rara vez solapa un fallo ocasionado en la cache L2.

Extensión: El reto de las prestaciones de los algoritmos consiste en que, según las distintas implementaciones de la misma arquitectura, la jerarquía de memoria varía en términos de capacidad, asociatividad y tamaño de bloque de la cache, así como del número de caches. Para acomodarse a esta variabilidad, algunas bibliotecas numéricas de reciente creación parametrizan sus algoritmos con el objetivo de realizar durante el tiempo de ejecución una sintonización de parámetros para aplicar la mejor combinación a un determinado computador. Esta alternativa se llama *autosintonización*.

Autoevaluación

¿Cuál de las siguientes afirmaciones sobre el diseño con varios niveles de caches es generalmente cierta?

1. A las caches de primer nivel les afectan más el tiempo de acierto, y a las caches de segundo nivel les afecta más la frecuencia de fallos.
2. A las caches de primer nivel les afectan más la frecuencia de fallos, y a las caches de segundo nivel les afecta más el tiempo de acierto.

Resumen

En esta sección nos hemos concentrado en tres temas: las prestaciones de las caches, la utilización de la asociatividad para reducir frecuencias de fallos y el uso de jerarquías multinivel de caches para reducir las penalizaciones por fallo.

El sistema de memoria afecta de forma significativa al tiempo de ejecución de los programas. El número de ciclos de parada debido a la memoria depende tanto de la frecuencia de fallos como de la penalización por fallo. El reto, como veremos en la sección 5.5, consiste en reducir uno de estos factores sin afectar significativamente a otros factores críticos de la jerarquía de memoria.

Para reducir la frecuencia de fallos, examinamos el uso de las políticas de emplazamiento asociativas. Tales políticas pueden reducir la frecuencia de fallos de una cache mediante la mayor flexibilidad que se permite al emplazamiento de bloques dentro de la cache. Las políticas completamente asociativas permiten emplazar los bloques en cualquier lugar, pero también requieren que cada bloque de la cache sea inspeccionado para resolver una solicitud de acceso a memoria. Los altos costes hacen impracticables las caches completamente asociativas de gran tamaño. Las caches asociativas por conjuntos son alternativas más factibles, ya que sólo necesitamos inspeccionar entre los elementos de un único conjunto que es seleccionado por indexación. Las caches asociativas por conjuntos tienen frecuencias de fallos más elevadas, pero son más rápidas de acceder. La cantidad de asociatividad que proporciona las mejores prestaciones depende tanto de la tecnología como de los detalles de la implementación.

Finalmente examinamos las caches multinivel como una técnica que reduce la penalización por fallo al permitir una cache secundaria más grande que maneja los fallos de la cache principal. Las caches de segundo nivel se han convertido en elementos habituales a medida que los diseñadores encuentran que las restricciones en área de

silicio, así como los objetivos marcados por las elevadas frecuencias de reloj, impiden que las caches principales puedan ser más grandes. La cache secundaria, que a menudo es 10 o más veces mayor que la cache principal, resuelve muchos accesos que fallan en la cache principal. En tales casos, la penalización por fallo viene dada por el tiempo de acceso a la cache secundaria (normalmente < 10 ciclos del procesador), en comparación con el tiempo de acceso a la memoria principal (normalmente > 100 ciclos del procesador). Como se hizo con la asociatividad, las compensaciones establecidas en el diseño entre el tamaño de la cache secundaria y su tiempo de acceso dependen de un conjunto de aspectos asociados a la implementación.

. . . se ha ideado un sistema para hacer que la combinación formada por la memoria principal y el tambor aparezca al programador como un único nivel de almacenamiento, en el que las transferencias solicitadas tienen lugar de forma automática.

Kilburn et al., "Sistema de almacenamiento de un nivel" 1962

5.4 Memoria virtual

En la sección anterior vimos cómo las caches proporcionaban rápidos accesos a los datos y al código de un programa que fueron utilizados recientemente. De un modo parecido, la memoria principal puede funcionar como una "cache" para el almacenamiento secundario, normalmente implementado con discos magnéticos. Esta técnica se denomina **memoria virtual**. Históricamente, existen dos motivaciones importantes que justifican la existencia de la memoria virtual: permitir que la memoria sea compartida de forma eficiente y segura por varios programas, y eliminar los inconvenientes de programación asociados a una memoria principal cuya capacidad sea pequeña y limitada. Cuatro décadas después de su invención, la justificación que reina hoy en día es la primera.

Memoria virtual: técnica que utiliza la memoria principal como una "cache" para el almacenamiento secundario.

Considere un conjunto de programas que se ejecutan a la vez en un computador. Evidentemente, para permitir que varios programas compartan la memoria principal, debemos ser capaces de proteger los programas de manera que no interfieran entre sí, asegurando que un programa sólo pueda leer y escribir en las porciones de memoria principal que se le han asignado. La memoria principal sólo necesita almacenar las partes activas de varios programas, del mismo modo que la cache almacena la parte activa de un solo programa. Así, el principio de localidad permite la existencia de la memoria virtual además de las caches, y la memoria virtual nos permite compartir eficientemente el procesador y la memoria principal.

No podemos saber qué programas compartirán la memoria cuando los compilemos. De hecho, los programas que comparten la memoria cambian dinámicamente mientras se están ejecutando. Debido a esta interacción dinámica, nos gustaría compilar cada programa en su propio *espacio de direcciones:* conjunto diferenciado de posiciones de memoria que sólo son accesibles por un programa. La memoria virtual implementa la traducción del espacio de direcciones de un programa a **direcciones físicas**. Este proceso de traducción impone la **protección** del espacio de direccionamiento de un programa frente al de los otros programas.

Dirección física: una de las direcciones de la memoria principal.

Protección: conjunto de mecanismos utilizados para asegurar que varios procesos que comparten el procesador, la memoria o los dispositivos de entrada/salida no interfieren entre sí realizando lecturas o escritura mutuas, de forma tanto intencionada como desintencionada. Estos mecanismos también aíslan al sistema operativo de los procesos de usuario.

La segunda motivación de la memoria virtual consiste en permitir a un único programa de usuario excederse en la capacidad de la memoria principal que utiliza. Antiguamente, si un programa era demasiado grande para la memoria, se le dejaba al programador que realizara el ajuste. Los programadores dividían los programas en trozos e identificaban los trozos que se excluían mutuamente. Estos trozos de pro-