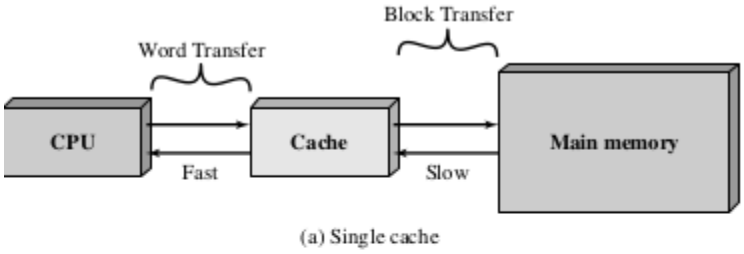
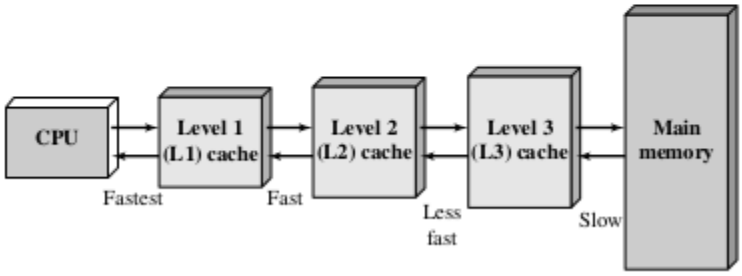
Capítulo 5. Memoria Cache

5.1 Principios de la memoria caché

El objetivo de la memoria caché es proveer un tipo de memoria que se acerque a la velocidad de las memorias más rápidas disponibles y al mismo tiempo proveer una mayor capacidad, todo con el precio/costo de memorias semiconductoras más baratas.



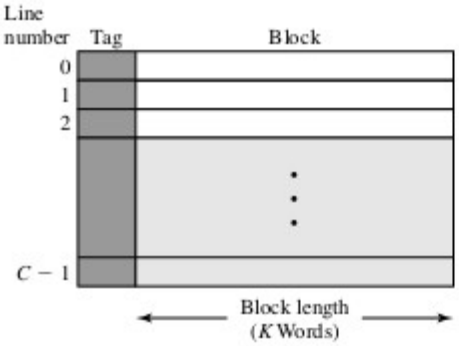
(a) Single cache



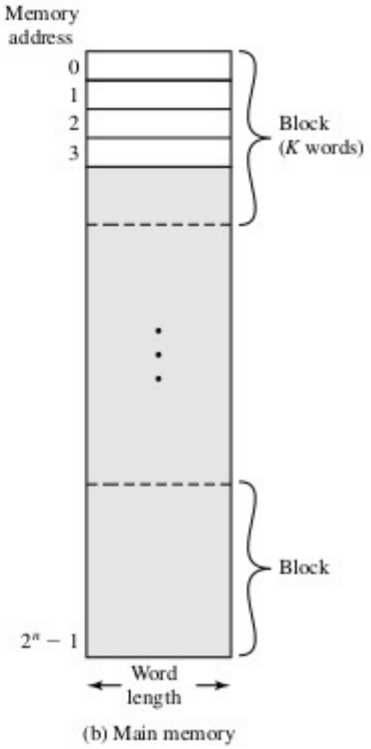
(b) Three-level cache organization

La caché contiene una copia de porciones de la memoria principal. Cuando el procesador intenta leer una palabra de memoria, se revisa si la palabra se encuentra en caché. En caso afirmativo, la palabra se entrega al procesador. De otra forma, un bloque de memoria principal, que consiste de un número fijo de palabras, se guarda en caché y entonces la palabra es entregada al procesador. Debido al fenómeno de localidad de referencia, cuando un bloque de datos se trae al caché para satisfacer una referencia a memoria única, es probable que existan futuras referencias a esa misma localidad de memoria o a otras localidades en el mismo bloque.

En la siguiente figura se aprecia la estrucutra de un sistema caché/memoria principal. La memoria principal consiste de hasta *2^n* localidades direccionables, donde cada localidad tiene una dirección única de *n* bits. La memoria principal consiste de un número fijo de bloques con *K* localidades cada uno. Existen, *M=(2^n)/K*. El caché consiste de *m* bloques, llamados **líneas**. Cada línea contiene *K* unidades direccionables (bytes o palabras), además de una etiqueta *(tag)* de unos cuantos bits. Cada línea además contiene algunos bits de control que se utilizan en casos como el de indicar si una línea ha sido modificada desde que se guardó en caché. La longitud de una línea, sin incluír la etiqueta y los bits de control, se conoce como **tamaño de línea**. El número de líneas en caché es considerablemente menor al número de bloques de la memoria principal (*m << M*). En todo momento, un subconjunto de los bloques de la memoria reside en las



(a) Cache

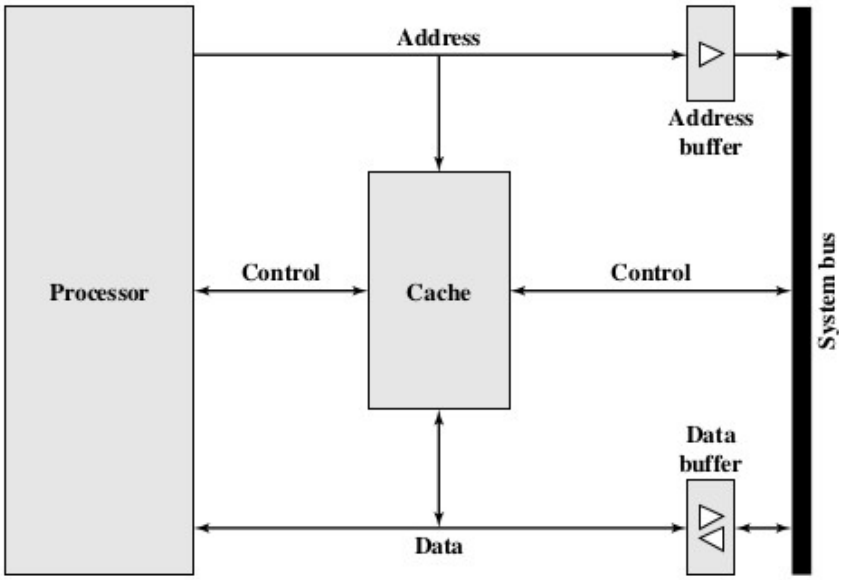


(b) Main memory

líneas del caché. Si se lee una localidad de un bloque de memoria, dicho bloque se transfiere a una de las líneas del caché.

Debido a que existe muchos más bloques que líneas, una línea individual no puede ser dedicada de manera única y exclusiva a un bloque en particular. Así, la **etiqueta** de cada línea se utiliza para identificar qué bloque en particular se encuentra guardado en ella. La etiqueta es usualmente una porción de la dirección en memoria principal.

La organización contemporanea de la caché se muestra en la siguiente figura. El caché se conecta con el procesador a través de lineas de datos, control y dirección. Las líneas de datos y direcciones también se conectan a un buffer, los cuales a su vez, se conectan con el bus del sistema para accesar a la memoria principal. Cuando la palabra buscada se encuentra en el caché *(cache hit),* se des activan los buffers de dirección y de datos y la comunicación ocurre entre el procesador y el caché unicamente. Cuando no se encuentra la palabra buscada en el caché *(cache miss)*, la dirección deseada se carga en el bus del sistema y los datos se obtienen a través del buffer de datos tanto para el caché como para el procesador.



5.3 Elementos de diseño del caché

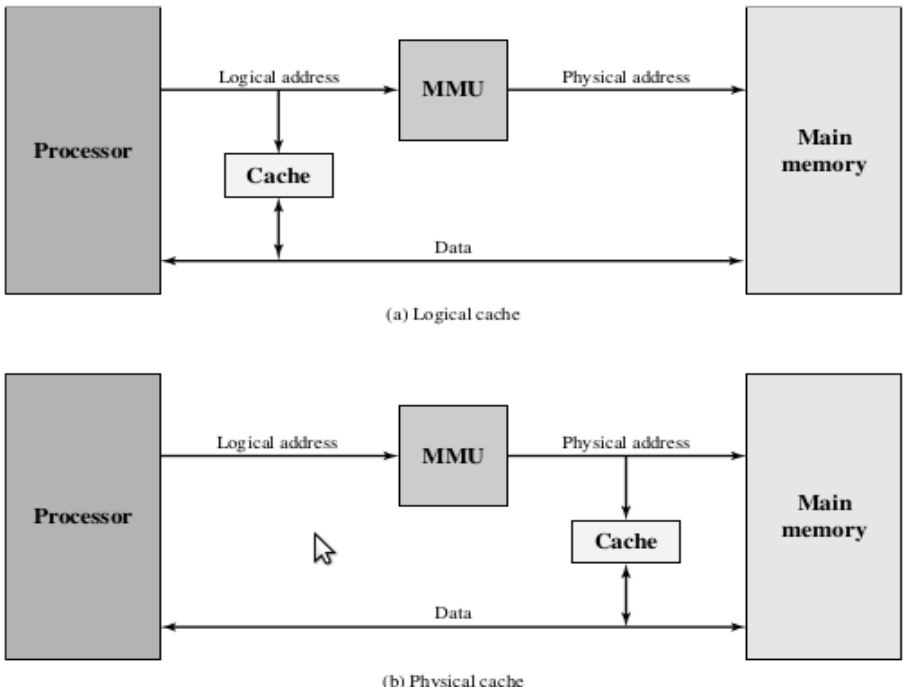
Aunque existen un gran número de implementaciones distintas de caché, existen algunos elementos básicos de diseño que sirven para clasificar y diferenciar las arquitecturas:

- **Direcciones caché**
  - Lógicas
  - Físicas
- **Tamaño del caché**
- **Función de mapeo**
  - Directo
  - Asociativo
  - Asociativo por conjuntos
- **Algoritmo de reemplazo**
  - *Least recently used (LRU)*
  - *Firs in, first out*
  - *Least frecuently used (LFU)*
  - *Aleatorio*
- **Política de escritura**
  - *Write through*
  - *Write back*
  - *Write once*
- **Tamaño de línea**
- **Número de cachés**
  - Uno o dós niveles
  - Unificado o dividido

Direcciones caché

Casi todos los procesadores no embebidos, y varios embebidos, soportan el uso de memoria virtual. La memoria virtual permite a los programas direccionar memoria desde un punto de vista lógico, sin importar la cantidad de memoria principal disponible físicamente. Cuando se utiliza memoria virtual, los campos de las instrucciones contienen direcciones virtuales. Para poder escribir y leer de la memoria principal, se necesita de una unidad de manejo de memoria (*MMU, Memory Management Unit*) que traduce direcciones virtuales a direcciones físicas en la memoria.

El diseñador de sistema puede elegir ubicar el caché entre el procesador y el MMU o entre el MMU y la memoria principal. Un **caché lógico**, también conocido como **caché virtual**, guarda los datos utilizando **direcciones virtuales**. El procesador accesa al caché directamente, sin utilizar el MMU. Un **caché físico** guarda los datos utilizando **direcciones físicas** de la memoria principal. Una ventaja del caché lógico es la velocidad de respuesta, debido a que el caché puede responder sin que el MMU realice traducciones. Una desventaja es el hecho de que la mayoría de los sistemas de memoria virtual proveen a cada aplicación con el mismo espacio de direcciones virtuales. Es decir, cada aplicación ve una memoria virtual que comienza con la dirección 0. Así, la misma dirección virtual para dos aplicaciones diferentes se refiere a dos direcciones físicas distintas. De esta maner, la memoria caché debe ser vaciada completamente con cada cambio de contexto de aplicación o se deben de agregar bits extra para identificar a cuál espacio de direcciones virtual se refiere la dirección.



Tamaño del caché

Lo ideal sería tener un tamaño de caché lo suficientemente pequeño para que el costo promedio por bit sea cercano al de la memoria principal por sí sola; y lo suficientemente grande para que el tiempo de acceso promedio sea cercano al del caché por si solo. Adicionalmente, mientras más grande es el caché, más grande es el número de compuertas utilizadas para direccionarlo. El resultado es que las memorias caché grandes tienden a ser ligeramente más lentas que las pequeñas, incluso si son construidas con la misma tecnología semiconductora. El área disponible en el chip y la tarjeta también es un factor a considerar al elegir el tamaño del caché.

Función de mapeo

Debido a que existen muchas menos líneas de caché que bloques de memoria principal, es necesario un algoritmo para mapear o relacionar los bloques de memoria a las líneas de caché. Además, es necesario un meodo para determinal cuál bloque de memoria ocupa cada línea en determinado momento. Tres técnicas se utilizan: mapeo directo, asociativo y asociativo por conjuntos. Se examinarán los tres tipos a continuación, para todos los ejemplos se utilizará la estructura general siguiente:

- El caché puede almacenar 64 KBytes
- Los datos se transfieren entre la memoria y el caché en bloques de 4 bytes. Ésto significa que el caché está organizado como *16K = 2^14* líneas de 4 bytes cada una.
- La memoria principal consiste de 16 Mbytes, donde cada byte es directamente direccionable mediante una dirección de 24 bits (*2^24 = 16M*). Así, la memoria principal consiste de 4M bloques de 4 bytes cada uno.

Mapeo Directo

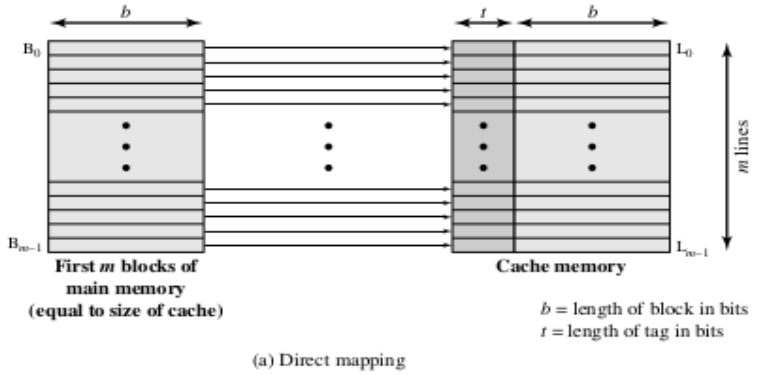
Esta técnica mapea cada bloque de memoria con una posible línea del caché únicamente. El mapeo se expresa como:

i = j módulo m

donde

i = número de línea
j = número del bloque de memoria
m = número de líneas en el caché

Recuerde que la cantidad total de bloques de memoria ***M*** es mucho mayor a la cantidad de líneas del caché ***m***. Cada bloque de memoria puede ser mapeado a una línea única del caché. Por ejemplo, si el caché se compone de 10 líneas y la memoria principal se compone de 100 bloques, sólo los bloques 1, 11, 21, 31, ... ,91 pueden ser mapeados a la línea 1; los bloques 2, 12, 22, 32, ... , 92 pueden ser mapeados a la línea 2; y así sucesivamente (Nota: las líneas y bloques usualmente se numeran comenzando desde cero).



(a) Direct mapping

La función de mapeo es fácilmente implementada utilizando la dirección de memoria principal. Para realizar el acceso a caché, cada dirección de memoria principal puede ser vista como si estuviera compuesta de 3 campos. Los ***w*** bits menos significativos identifican una unidad direccionable (byte o palabra) dentro del bloque de memoria principal. Los ***s*** bits restantes especifican uno de los ***2^s*** bloques de memoria. La lógica del caché interpreta estos ***s*** bits como 2 campos: una etiqueta de ***s – r*** bits (bits más significativos) y un campo identificador de línea de ***r*** bits. Este último campo identifica una de las ***m = 2^r*** líneas del caché. Para sumarizar:

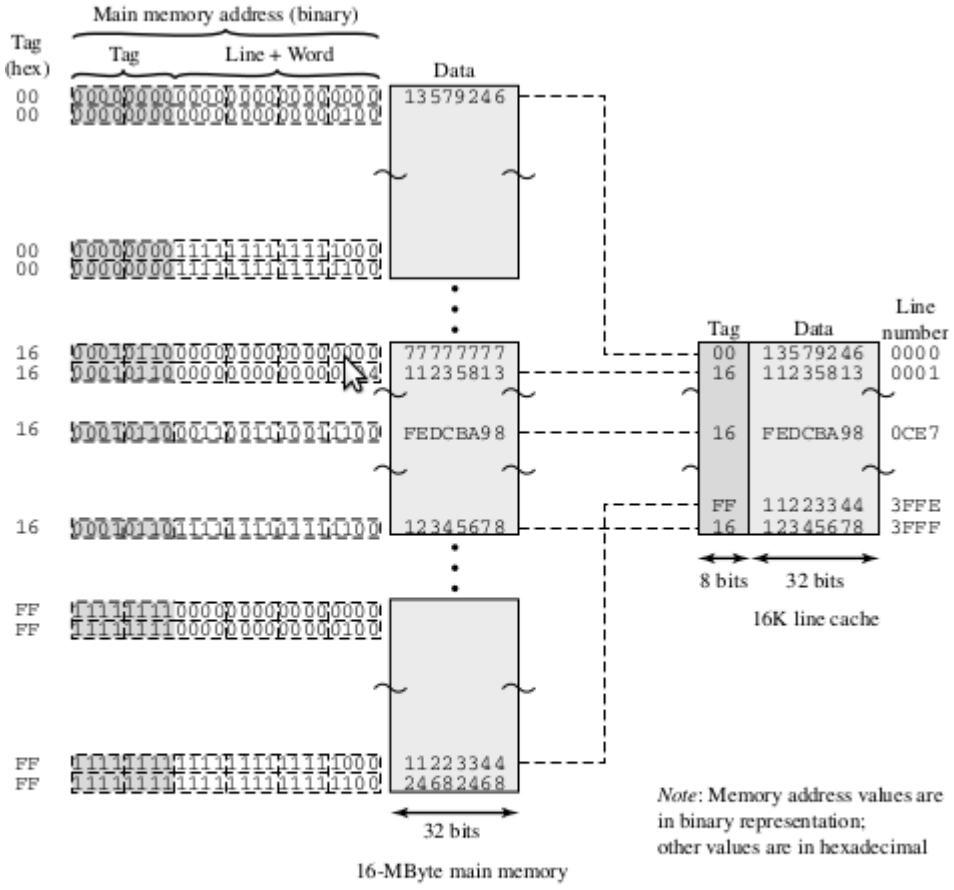
- Longitud de la dirección de memoria de cada localidad = *s + w* bits = 24 bits para nuestros ejemplos.
- Número de unidades direccionables = *2 ^ (s + w)* palabras o bytes, bytes para nuestros ejemplos.
- Tamaño de bloque = tamaño de línea = *2 ^ w ,* 4 bytes para nuestros ejemplos, por lo tanto *w = 2.*
- Número de bloques totales en memoria principal = *[2 ^ (s + w)] / 2 ^ w = 2 ^ s*
- Número de lineas en el caché = *m = 2 ^ r* , para nuestros ejempos *r = 14, m = 16K*
- Tamaño del caché = *2 ^ (r + w)* palabras o bytes, para nuestros ejemplos = *2^(14+2) = 2^16 = 64K*.
- Tamaño de la etiqueta = *(s – r)* bits, para nuestros ejemplos *s = 22, r = 14, s – r = 8*
- Función de mapeo: *i = j módulo m,* para nuestro ejemplo ***i = j módulo 2 ^ 14***

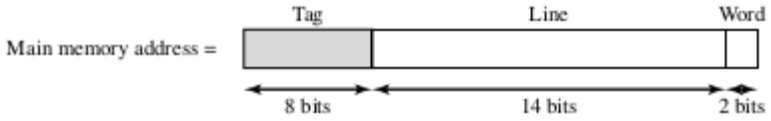
Como cada línea/bloque contiene 4 bytes se utilizan 2 bits para especificar uno cuál de los cuatro se desea accesar *(w = 2).* Así las direcciones de memoria iniciales de cada bloque (en hexadecimal) son: 000000, 000004, 000008, 00000C, 000010, 000014, ..., etc. El mapeo resultante es el siguiente:

Línea	**Direcciones de memoria iniciales de los bloques**
0	000000, 010000, 020000, ... , FF0000
1	000004, 010004, 020004, ... , FF0004
2^14 - 1	00FFFC, 01FFFFC, 02FFFC, ... , FFFFFC

Nótese que no existen dos bloques que se mapeen a la misma línea y que tengan el mismo número de etiqueta. Así, la etiqueta del bloque 000000 es 00, la etiqueta del bloque 010000 es 01, la etiqueta del bloque FF0000 es FF y así sucesivamente.

Una operación lectura se efectúa de la siguiente manera: se presenta al caché con una dirección de 24 bits. El número de línea de 14 bits se utiliza como un índice dentro del caché para acceder a una línea en particular. Si los 8 bits más significativos de la dirección presentada coinciden con la etiqueta de 8 bits almacenada para esa línea, entonces se utilizan los últimos dos bits (w) para seleccionar uno de los 4 bytes en la línea. De otra manera, los 22 bits de la etiqueta más el número de línea se utilizan para traer un bloque desde la memoria principal. La dirección utilizada para la obtención del bloque son los 22 bits mencionados más dos bits 0 concatenados al final, de manera que los 4 bytes se obtienen a partir de la dirección de inicio de un bloque.



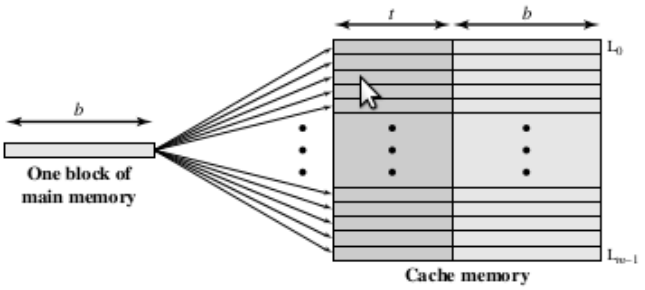


El mapeo directo es simple y barato de implementar. Su principal desventaja es que existe una ubicación fija en el caché para cada bloque. Así, si un programa referencia palabras repetidamente de dos bloques diferentes que se mapean a la misma línea del caché, los bloques se intercambiarán continuamente y el porcentaje de aciertos *(hit ratio)* será bajo.

Una manera de disminuir los fallos en el caché es recordar el bloque descartado en caso de que sea necesitado de nuevo. Como los bloques descartados ya han sido traídos de la memoria, pueden ser usados de nuevo con un costo bajo. Este tipo de esquemas se implementan utilizando un nivel adicional de caché conocido como **caché de víctimas.** El caché de víctimas es un caché asociativo, de tamaño entre 4 y 16 líneas, que reside entre un L1 de caché mapeado directametne y el siguiente nivel de memoria.

Mapeo Asociativo

El mapeo asociativo supera la desventaja del mapeo directo permitiendo que cada bloque de memoria sea mapeado/cargado a cualquier línea de la caché. En este caso, la lógica de control del caché interpreta una dirección de memoria simplemente como una etiqueta y un campo para la palabra. La etiqueta identifica de manera única un bloque de memoria principal. Para determinar si un bloque se encuentra en caché, la lógica de control debe examinar de manera simultánea cada etiqueta. Note que no hay ningún campo en la dirección que identifique el número de línea así que el número de líneas en el caché no está determinado por el formato de la dirección. Para sumarizar:

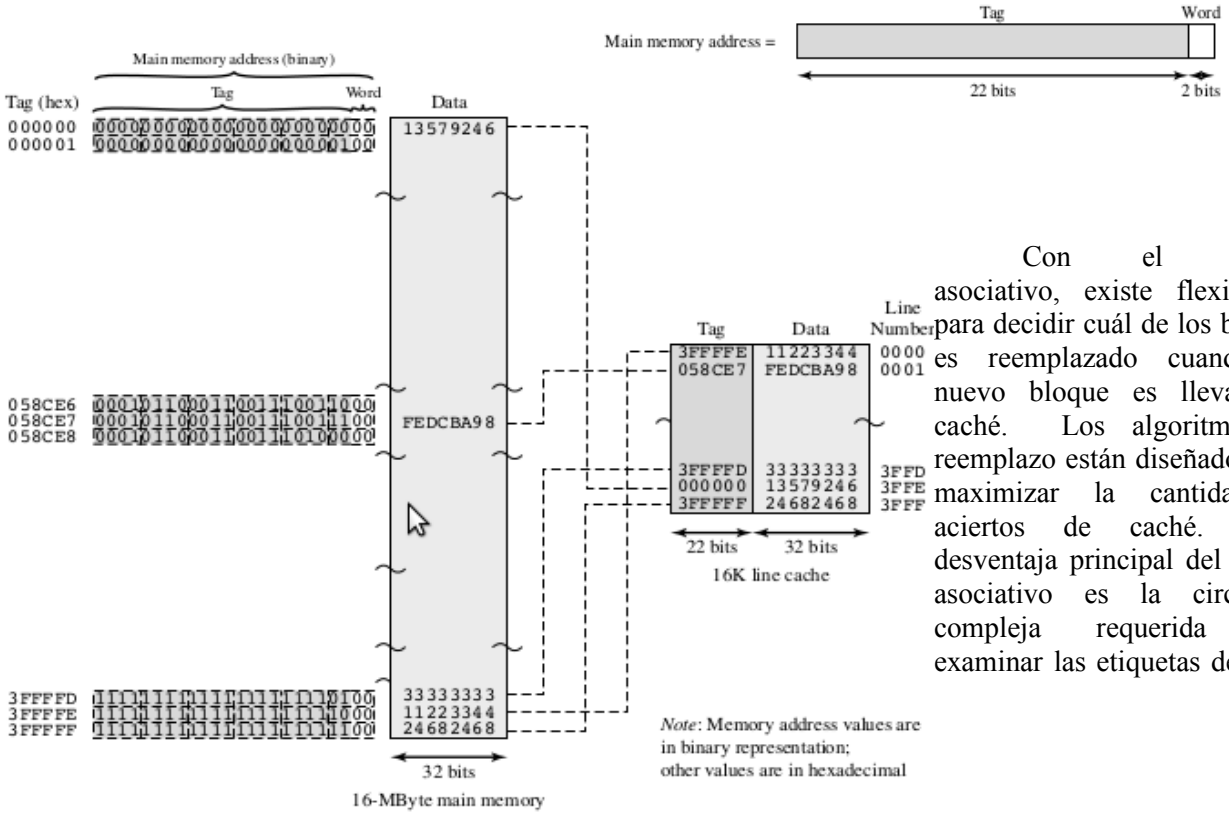


(b) Associative mapping

- Longitud de la dirección = *s + w* bits = 24 para nuestros ejemplos.
- Número de unidades direccionables = *2 ^ (s + w)* palabra o bytes.
- Tamaño de bloque = tamaño de línea = *2 ^ w* palabras o bytes = 4 para nuestros ejemplos, *w = 2*
- Número de bloques totales en memoria principal = *[2 ^ (s + w)] / 2 ^ w = 2 ^ s*
- Número de líneas en el caché = indeterminado = *16K* para nuestros ejemplos.
- Tamaño de la etiqueta = *s* bits = 22 para nuestros ejemplos.

Así pues, una dirección de memoria principal consiste de una etiqueta de 22-bits y un número de byte de 2-bits. Los 22 bits de la etiqueta deben de ser almacenados en la línea del caché junto con los 32 bits de datos (4 bytes). Nótese que son los 22 bits más significativos de la dirección los que forman la etiqueta. Entonces, una dirección hexadecimal de **24 bits 16339C** tiene una etiqueta de **22 bits 058CE7**:

Dirección de memoria	0001	0110	0011	0011	1001	1100
	1	6	3	3	9	C
Etiqueta	00	0101	1000	1100	1110	0111
	0	5	8	C	E	7



Con el mapeo asociativo, existe flexibilidad para decidir cuál de los bloques es reemplazado cuando un nuevo bloque es llevado al caché. Los algoritmos de reemplazo están diseñados para maximizar la cantidad de aciertos de caché. La desventaja principal del mapeo asociativo es la circuitería compleja requerida para examinar las etiquetas de todas

las líneas en paralelo.

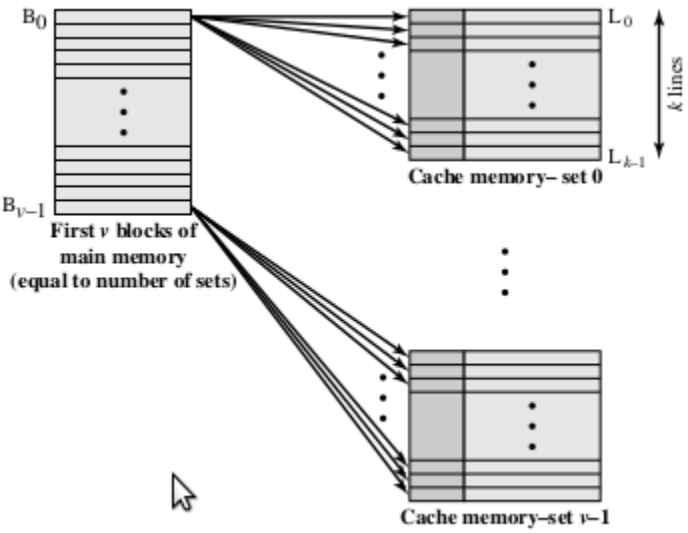
Mapeo asociativo por conjuntos

Este esquema es una combinación que exhibe las fortalezas de los enfoques directo y asociativo, reduciendo sus desventajas. En este caso el caché consiste de un número de conjuntos, cada uno de los cuales consiste de un número de líneas. Las relaciones son:

$$m = v \times k$$
$$i = j \text{ módulo } v$$

donde

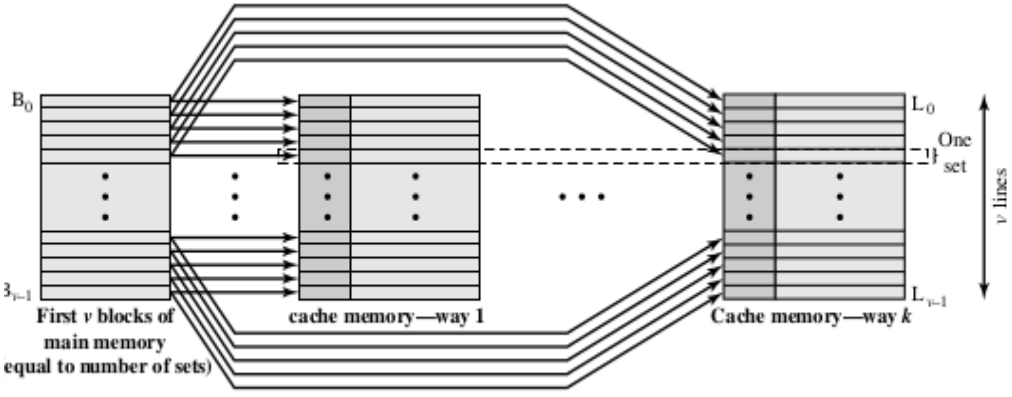
i = número del conjunto
j = número del bloque de memoria
m = número de líneas en el caché
v = número de conjuntos totales
k = número de líneas en cada set



(a) v Associative–mapped caches

A este esquema se le conoce como, mapeo asociativo por conjuntos de *k-maneras*. El bloque *B_j* puede ser mapeado a cualquiera de las líneas del conjunto *j*. Al igual que en el mapeo asociativo, un bloque puede ser mapeado a diferentes líneas del caché. Sin embargo, los bloques sólo pueden ser mapeados a las líneas de un conjunto en particular, de ahí asociativo por conjuntos. Por ejemplo, el bloque B_0 puede ser mapeado a cualquier línea del conjunto de líneas 0. Así, un caché asociativo por conjuntos puede ser implementado físicamente como *v* cachés asociativos.

También es posible implementar un caché asociativo por conjuntos utilizando *k* cachés con mapeo directo. Cada caché mapeado directamente se conoce como una *manera* y consiste de *v* líneas. Los primeros *v* bloques de memoria principal se mapean directamente a las *v* líneas de cada *manera*; el siguiente grupo de *v* bloques se mapean de manera similar y así sucesivamente. La implementación con cachés directametne mapeados se utiliza típicamente para grados pequeños de asociatividad (valores pequeños de *k*) mientras que la implementación con cachés asociativos se utiliza para grados mayores de asociatividad.

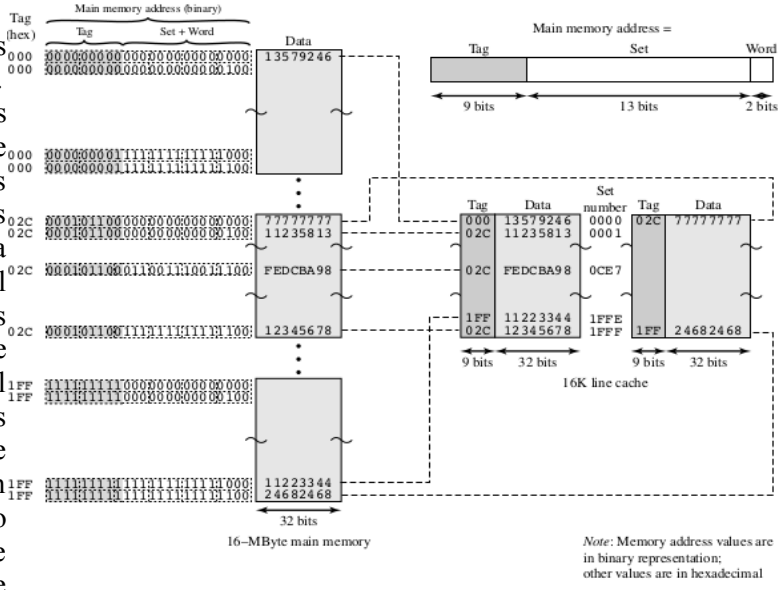


(b) k Direct–mapped caches

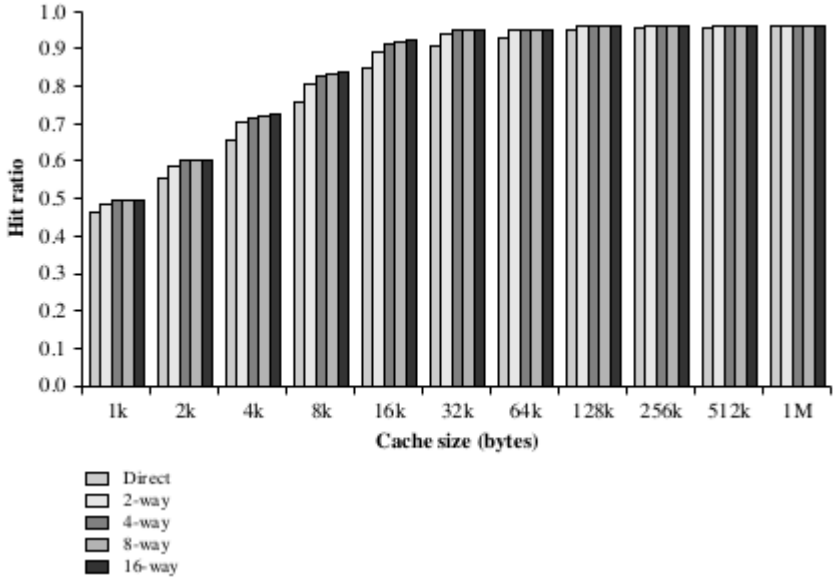
Para el mapeo asociativo por conjuntos, la lógica de control del caché interpreta una dirección de memoria como 3 campos: etiqueta, conjunto y palabra. Los *d* bits de conjunto especifican uno de los *v=2^d* conjuntos. Los *s* bits de etiqueta y conjunto especifican uno de los *2^s*

bloques de memoria principal. Con mapeo completamente asociativo, la etiqueta en la dirección de memoria es bastante grande y debe ser comparada con la etiqueta de cada línea de caché. Con un mapeo asociativo por conjuntos de *k-maneras*, la etiqueta en una dirección de memoria es mucho más pequeña y únicamente se compara con las *k* etiquetas de un sólo conjunto. Para sumarizar:

- Longitud de la dirección = *s + w* bits = 24 para nuestros ejemplos.
- Número de unidades direccionables = *2 ^ (s + w)* palabra o bytes.
- Tamaño de bloque = tamaño de línea = *2 ^ w* palabras o bytes = 4 para nuestros ejemplos, *w = 2*
- Número de bloques totales en memoria principal = *[2 ^ (s + w)] / 2 ^ w = 2 ^ s*
- Número de líneas en un conjunto = *k,* para nuestros ejemplos *k = 2*
- Número de conjuntos = *v = 2^d,* para nuestros ejemplos *d = 13*
- Número de líneas en el caché = *m = kv = k* X *2^d = 16K* para nuestros ejemplos.
- Tamaño del caché = *k* X *2^(d + w)* palabras o bytes = *64K* para nuestros ejemplos.
- Tamaño de la etiqueta = *s - d* bits = *22 – 13 = 9*



Para el ejemplo, utilizamos conjuntos de dos líneas por lo tanto *k = 2*. Así, necesitaremos *2^13 (d = 13)* conjuntos distintos para completar nuestro caché de *16K*. El número de conjunto de 13 bits identifica un conjunto único y también nos indica el número de bloque en memoria principal, módulo *2^13*. Esto determina el mapeo de los bloques en las líneas. Así, los bloques 000000, 008000, ... , FF8000 de memoria principal se mapean en el conjunto 0 del caché. Cualquiera de esos bloques se puede mapear a cualquiera de las dos líneas del set. Note que no existen dos bloques que se mapeen en el mismo conjunto y contengan el mismo número de etiqueta. En el caso de una operación de lectura, se utiliza el número de conjunto para determinar cuál conjunto de dos líneas será examinado. Se busca una coincidencia examinando ambas líneas del set utilizando el número de etiqueta de la dirección a ser accesada.



En el caso extremo en que *v = m, k = 1,* la técnica asociativa por conjuntos se reduce a mapeo directo; para *v = 1, k = m*, se reduce a mapeo asociativo. El uso de dos líneas por set *v = m/2, k = 2* es la organización más común asociativa por conjuntos. Ésta asociación de *2-maneras* mejora de manera significativa la cantidad de aciertos en el caché por sobre el mapeo directo. La asociación por conjuntos de *4-maneras (v = m/4, k = 4)* provoca una modesta mejora por un consto adicional relativametne pequeño. Incrementos mayores en el número de líneas por conjunto tienen muy poco efecto.

Algoritmos de reemplazo

Una vez que el caché se ha llenado, cuando un nuevo bloque se desea guardar en el caché, uno de los bloques que ya se encuentran en él debe ser reemplazado. Cuando se utiliza mapeo directo, únicamente existe una posible línea para cada bloque en particular y, por lo tanto, no es necesario tomar una decisión. Para las técnicas asociativas y asociativas por conjuntos, es necesario un algoritmo de reemplazo. Para lograr velocidades altas, dicho algoritmo deberá estar implementado en hardware. Muchos algoritmos se han considerado, a continuación se presentan 4 de los más comunes.

Probablemente el más efectivo sea el algoritmo de *menos recientemente usado (LRU, least recently used)*. Consiste en reemplazar el bloque en el conjunto que haya estado en el caché por el mayor tiempo sin que haya habido referencias a él. En un caché asociativo de 2-maneras se implementa fácilmente utilizando un bit de USO. Cuando se hace referencia a una línea del conjunto, su bit de USO se activa y se desactiva el bit de la otra línea. Cuando se desea cargar un nuevo bloque a ese conjunto, la linea cuyo bit de USO es 0 se reemplaza. LRU también es relativamente fáicl de implementar para un caché completamente asociativo. El mecanismo de caché mantiene una lista de índices a todas las líneas en el caché. Cuando una línea es referenciada, se mueve al inicio de la lista. Se reemplaza la línea al final de la lista. Por su simplicidad de implementación y su buen *hit ratio,* LRU es el algoritmo de reemplazo más popular.

Otra posibilidad es *primero en entrar, primero en salir (FIFO, first-in first-out)*: reemplazar el bloque en el conjunto que haya estado en el caché por más tiempo. FIFO se implementa con un buffer circular. Otro algoritmo es el *menos frecuentemente utilizado (LFU, least frequently used)*: reemplazar el bloque en el conjunto que haya tenido la menor cantidad de referencias. LFU puede ser implementado asociando un contador a cada línea. Una última posibilidad, que no se basa en el uso, es seleccionar un bloque de manera aleatoria. Simulaciones han mostrado que el reemplazo aleatorio tiene un desempeño ligeramente inferior a un algoritmo basado en el uso.

Política de escritura

Cuando se va a reemplazar una línea del caché hay dos casos a considerar. Si el bloque no ha sido alterado, entonces puede ser sobreescrito por el nuevo bloque sin primero guardarlo. Si al menos una operación de escritura se ha efectuado en una de las unidades direccionables del bloque, entonces la memoria principal debe ser actualizada escribiendo la línea del caché en el bloque de memoria correspondiente antes de sobreescribirla.

Es posible una variedad de políticas de escritura con ventajas y desventajas. Hay dos problemas con los cuales lidear. Primero, más de un dispositivo puede accesar a la memoria principal. Por ejemplo, un módulo I/O puede ser capaz de escribir y leer directamente de la memoria. Si una palabra se ha alterado únicamente en el caché, entonces la palabra correspondiente en memoria es inválida. Adicionalente, si el dispositivo I/O ha alterado la memoria, la palabra almacenada en el caché es inválida. Un problema más complejo ocurre cuando múltiples procesadores se encuentran conectados al mismo bus y cada procesador cuenta con su propio caché. Entonces, si una palabra es alterada en un caché, podría invalidar a otros cachés.

La técnica más simple se conoce como ***write through*****.** Utilizando esta técnica, todas las operaciones de escritura se hacen tanto en memoria principal como en el caché, asegurándose de que la memoria siempre sea válida. Cualquier módulo procesador-caché monitorea el tráfico hacia la memoria principal para mantener la consistencia en su propio caché. La mayor desventaja de esta técnica es la gran cantidad de trafico de memoria que puede ocasionar un cuello de botella.

Una técnica alternativa conocida como ***write back*** minimiza el número de operaciones de escritura. Las actualizaciones se hacen únicamente en caché. Cuando ocurre una actualización, se activa un **bit de uso** o **bit de modificación** asociado a esa línea. Entonces, cuando el bloque es reemplazado, se escribe únicamente si dicho bit está activo. El problema con esta técnica es que, por cierto tiempo, porciones de la memoria principal son inválidas y por ende, los accesos de los módulos I/O deben permitirse únicamente a través del caché. Lo anterior hace necesaria circuitería más compleja.

La experiencia ha mostrado que del número total de referencias a memoria, aproximadamente el 15% son operaciones de escritura. **Para que un caché write-back sea más eficiente que un caché write-through,**

el número de operaciones promedio para una línea cualquiera debe ser mayor al tamaño de la línea del caché.

En una organización en la que más de un dispositivo tiene un caché y la memoria principal es compartida se produce un nuevo problema. Si los datos en un caché son alterados, se invalida no únicamente la palabra correspondiente en memoria, sino también la misma palabra si se encuentra en otros cachés. Incluso si se utiliza una política write-through, los otros cachés pueden contener información inválida. Posibles enfoques para mantener la coherencia del caché son:

- **Monitoreo del bus con write-through:** cada controlador de caché monitorea las lineas de direcciones para detectar operaciones de escritura. Si otro maestro escribe a una localidad de memoria que también reside en el caché, el controlador del caché invalida la línea correspondiente. Esta estrategia depende de que todos los controladores caché utilicen una política write-through.
- **Por hardware:** se utiliza hardware adicional para asegurarse que todas las actualizaciones a memoria se vean reflejadas en todos los cachés.
- **Mediante memoria no cacheable:** sólo una porción de la memoria principal es compartida por más de un procesador, y se designa como **no-cacheable**. En un sistema así, todos los accesos a la memoria compartida se registran como fallos de caché, porque la memoria compartida jamás se copia en el caché.

Tamaño de línea

Cuando una palabra es requerida por el procesador, no sólo se obtiene la palabra correspondiente sino también un número de palabras adyacentes. A este conjunto de palabras adyacentes se le denomina bloque y se almacena en una de las líneas del caché. Conforme aumenta el tamaño del bloque desde tamaños muy pequeños a tamaños más grandes, el porcentaje de aciertos del caché aumentará en un inicio por el principio de localidad: más datos útiles son traídos al caché. Sin embargo, el porcentaje de aciertos comenzará a disminuir conforme el bloque se hace cada vez más grande y la probabilidad de utilizar la nueva información se vuelve menor a la probabilidad de utilizar la información que necesita ser reemplazada. Dos efectos específicos toman parte:

- Bloques grandes reducen el número de bloques que caben en el caché. Como cada vez que se trae un bloque se reescribe alguno de los ya existentes, un número pequeño de bloques ocasiona que los datos sean sobreescritos poco tiempo después de que fueron obtenidos.
- Conforme un bloque se hace maś grande, cada palabra adicional se encuentra más lejana de la palabra requerida y, por lo tanto, es menos probable que sea requerida en el futuro.

La relación entre tamaño de bloque y porcentaje de aciertos es compleja, no existe un valor óptimo definitivo. Un tamaño de entre 8 a 64 bytes se considera apropiado.

Número de cachés

Cuando se introdujo el uso del caché, un sistema típico tenía solamente uno. Más recientemente, la norma se ha convertido en el uso de múltiples cachés. Dos aspectos a considerar es la cantidad de cachés a utilizar y el uso de cachés unificados vs cachés separados.

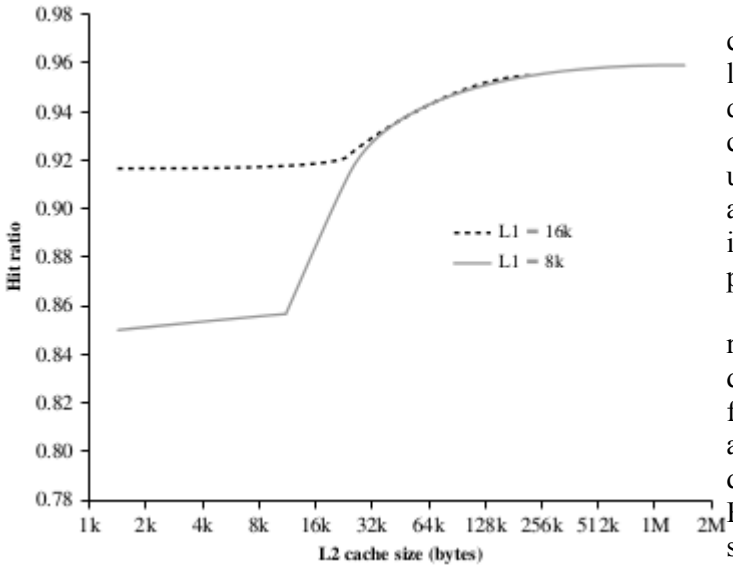
Cachés multinivel

Conforme ha aumentado la densidad de componentes, se ha vuelto posible tener un caché en el mismo chip que el procesador llamado ***on-chip caché***. Comparado con un caché alcanzable mediante un bus externo, el caché on-chip aumenta la velocidad de ejecución y el desempeño global del sistema. Cuando la instrucción o los datos requeridos se encuentran en el caché on-chip, se elimina por completo el acceso al bus. Además, durante este periodo de tiempo el bus se encuentra libre para atender otras transferencias.

La inclusión del caché en el chip del procesador deja abierta la pregunta de si será conveniente además tener un caché externo. La mayoría de los diseños contemporáneos incluyen tanto un caché on-chip como uno externo. La organización multinivel más simple se conoce como caché de 2 niveles, con el caché interno denominado como el nivel 1 (L1) y el caché externo como nivel 2 (L2). La razón para incluir L2 es la siguiente: si no existe un caché L2 y el procesador hace una petición de una localidad de memoria que no existe en L1,

entonces el procesador debe accesar DRAM o ROM a través del bus. Debido a las velocidades típicamente lentas del bus y al lento tiempo de acceso a memoria principal, ésto resulta en un desempeño pobre. Por otra parte, si se utiliza un caché L2 SRAM, entonces la información que frecuentemente no se encuentra en L1 puede ser accesada rápidamente. Si la SRAM es lo suficientemente rápida para igualar la velocidad del bus, entonces los datos pueden ser accesados en una transacción de **cero espera** *(zero-wait)*, el tipo de transferencia por bus más rápido.

Muchos diseños no utilizan el bus del sistema para hacer las transferencias entre L2 y el procesador, sino un bus independiente para reducir la carga en el bus del sistema. Adicionalmente, con el contínuo encogimiento de los componentes, un número de procesadores ahora incorporan L2 en el chip del procesador, mejorando así el desempeño.



Los ahorros potenciales por el uso de un caché L2 dependen del porcentaje de aciertos de los cachés L1 y L2. Varios estudios han mostrado que, en general, el uso de un segundo nivel de caché si mejora el desempeño. Sin embargo, el uso de un caché multinivel complica todos los aspectos de diseño relacionados con el caché incluyendo: tamaño, algoritmo de reemplazo y política de escritura.

La figura de la izquierda muestra los resultados de un estudio de simulación del desempeño de un caché de dos niveles como función del tamaño de los cachés. La figura asume que ambos cachés tienen el mismo tamaño de línea y muestra el porcentaje total de aciertos. Es decir, un acierto se cuenta si los datos deseados se encuentran en L1 o L2.

Se puede observar que L2 tiene un efecto pequeño en el número de aciertos hasta que es al menos del doble de tamaño que L1. Note que la sección más pronunciada de la pendiente para un caché L1 de 8KBytes es para un caché L2 de 16 KBytes. De igual manera para un caché L1 de 16 KBytes, la sección más pronunciada de la curva es para un caché L2 de 32 KBytes. La necesidad de que L2 sea más grande que L1 para afectar el desempeño tiene sentido. Si L2 tiene el mismo tamaño de línea y capacidad que L1, sus contenidos básicamente serían los mismo que los de L1.

Con la inclusión de L2 en el chip del procesador, muchos procesadores actuales han incluído un caché L3, originalmente accesible a través de un bus externo, aunque recientemente el caché L3 también es on-chip.

Cachés unificados vs cachés separados

Recientemente, se ha vuelto común dividir el caché en dos: uno dedicado a las instrucciones y otro dedicado a los datos. Estos dos cachés existen al mismo nivel, típicamente como dos cachés L1. Cuando el procesador intenta obtener una instrucción de la memoria, primero consulta el caché L1 de instrucciones, y cuando el procesador intenta obtener datos de la memoria principal, primero consulta el caché L1 de datos.

Por otro lado, existen dos ventajas potenciales para utilizar un caché unificado:

- Para un tamaño de caché dado, un caché unificado tiene un mayor porcentaje de aciertos que cachés separados porque balancea la carga entre instrucciones y datos de manera automática. Por ejemplo, si el patrón de ejecución requiere muchas más instrucciones que datos, entonces el caché tenderá a llenarse con instrucciones y viceversa.
- Sólamente se requiere diseñar e implementar un caché.

A pesar de estas ventajas, la tendencia es utilizar cachés divididos principalmente en máquinas que enfatizan la ejecución paralela de instrucciones y el *prefetching* (obtención por adelantado) de instrucciones futuras. La ventaja clave de un caché separado es que elemina la pelea por el caché entre la unidad de obtención/decodificación de instrucciones y la unidad de ejecución. Tipicamente, un procesador obtendrá instrucciones por adelantado y llenara con ellas un buffer de instrucciones a ejecutar. Ahora suponga que se

tiene un caché unificado para datos e instrucciones. Cuando la unidad de ejecución realice un acceso a memoria para obtener datos, esta petición se hace al caché unificado. Si al mismo tiempo, la unidad de obtención de instrucciones solicita una operación de lectura para obtener una instrucción del caché, esta petición se bloqueará de manera temporal mientras que el caché atiende a la unidad de ejecución. Esta pelea por el caché puede ocasionar que el desempeño se degrade e interfiera con el uso eficiente de la segmentación de instrucciones. Al utilizar cachés separados se supera dicha dificultad.