LO BÁSICO DE LAS CACHES

ARQUITECTURA DEL PROCESADOR II

Desde los primeros días de la computación[1] , los programadores han deseado disponer de memorias de máxima velocidad en cantidades ilimitadas. En este documento se consideran algunos aspectos introductorios que ayudan a los programadores a comprender cómo es posible concretar estos deseos. Antes de analizar de que manera se crea esta ilusión, veremos una analogía sencilla que ilustra los aspectos fundamentales y los mecanismos involucrados.

Supongamos que está preparando un informe sobre los principales desarrollos en la historia del hardware de computadoras. Estando sentado en un escritorio de la biblioteca, comienza a examinar algunos libros que ha seleccionado de las estanterías. Descubre que varios de los modelos de computadoras más importantes sobre los que necesita escribir están descriptos en los libros que ha seleccionado; pero no hay nada sobre la EDSAC. Entonces vuelve a buscar en las estanterías para traer otro libro. Encuentra uno sobre las primeras computadoras Británicas en el que se habla sobre la EDSAC. Una vez que Ud. cuenta con una buena selección de libros sobre su escritorio, existe una alta probabilidad que los temas que necesita se encuentren en ellos y, de esa manera, puede aprovechar todo su tiempo usando únicamente esos libros sin tener que volver a las estanterías. Tener varios libros sobre su escritorio le ahorra tiempo comparado con tener un solo libro a mano y tener que llevarlo de vuelta a las estanterías cada vez que necesita traer uno nuevo.

En este ejemplo de la biblioteca, el escritorio actúa como una *cache* – un lugar seguro para almacenar objetos, como ser los libros que se están usando. Las memorias que aparecen en los caminos de datos que hemos estudiado, deben ser reemplazadas por caches. *Cache* fue el nombre elegido para representar el nivel de la jerarquía de memoria entre el procesador y la memoria principal en la primer computadora comercial que contó con este nivel adicional. Si bien actualmente, ese es el principal uso la palabra cache, el término también designa todo tipo de almacenamiento que es administrado tomando ventaja de la localidad de acceso. Las caches primero aparecieron a comienzos de los años 60 en computadores de investigación y en las computadoras comerciales a fines de la misma década; todas las computadoras de propósito general que se construyen hoy día, desde los servidores hasta procesadores embebidos de bajo consumo, incluyen caches.

Este documento comienza mostrando una cache muy simple en la que los requerimientos del procesador son de una palabra y los bloques también consisten de una única palabra. La Figura 1 muestra esta cache simple, antes y después del requerimiento de un dato que inicialmente no está en la cache. Antes del requerimiento la cache contiene una serie de referencias recientes $X_1,\ X_2, \ldots,\ X_{n-1}$, y el procesador requiere una palabra X_n que no se

[1]Este material de estudio es la traducción de la sección 5.3 del libro de Patterson y Hennessy [1]

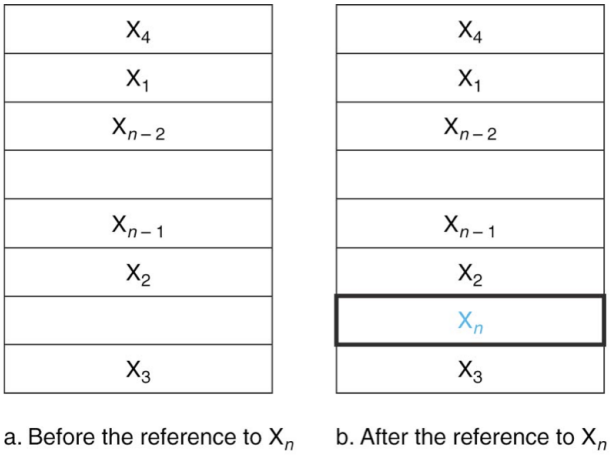

FIGURE 1. Antes y después de una referencia a X_n

encuentra en la cache. Este requerimiento resulta en un fallo (en inglés *miss*), y la palabra X_n es traída desde la memoria a la cache.

Al observar el escenario de la Figura 1, surgen dos preguntas: ¿Cómo saber si un ítem de datos está en la cache? Además, si está ¿cómo se lo encuentra? Las respuestas están relacionadas. Si cada palabra puede ir exactamente en un único lugar en la cache, entonces es fácil encontrar la palabra, si es que está en la cache. La forma más fácil de asignar una posición en la cache para cada palabra de la memoria es asignar una posición en la cache en base a la *dirección* de la palabra en la memoria. Esta estructura es llamada *correspondencia directa* (en inglés *direct-mapped*), debido a que cada posición de memoria se corresponde directamente de manera exacta con una posición en la cache. En esta correspondencia es usualmente simple encontrar la vinculación entre las direcciones y su posición en la cache. Por ejemplo casi todas las cache de correspondencia directa usan la siguiente transformación para encontrar un bloque en cache:

$$(\textit{Dirección del bloque})\ \textit{módulo}\ (\textit{Número de bloques en la cache})$$

Si el número de entradas en la cache es una potencia de 2, entonces el módulo puede ser calculado simplemente usando los $log_2(\textit{tamaño en bloques de la cache})$ bits de menor orden de la dirección. Así, una cache de 8 bloques usa los tres ($8 = 2^3$) bits de menos peso de la dirección del bloque. Por ejemplo, la Figura 2 muestra cómo las direcciones de memoria entre 1_{diez} (00001_{dos}) y 29_{diez} (11101_{dos}) se corresponden con las posiciones 1_{diez} (001_{dos}) y 5_{diez} (101_{dos}) en una cache de correspondencia directa de ocho palabras.

Concepto 1. *Una etiqueta (en inglés tag) es un campo en la estructura de la jerarquía de memoria, que contiene información de la dirección y se usa para verificar si un bloque en la jerarquía se corresponde con la palabra requerida.*

Teniendo en cuenta que una posición de la cache puede mantener los contenidos de distintas posiciones de memoria: ¿cómo se sabe si la palabra requerida está o no en la

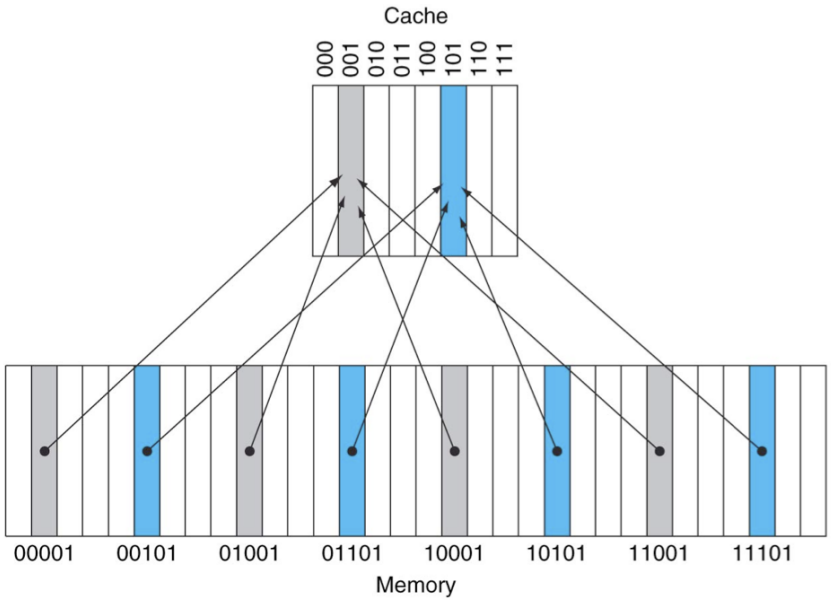

FIGURE 2. Una cache de correspondencia directa con ocho entradas

cache? La respuesta consiste en agregar un conjunto de *etiquetas* en la cache. Las etiquetas sacan de la dirección la información necesaria como para indicar si alguna de las palabras que están en la cache se corresponde con la palabra requerida. Las etiquetas sólo necesitan contener la porción superior de la dirección, es decir, aquellos bits que no son usados como un índice en la cache. Por ejemplo, en la Figura 2 una etiqueta consiste solamente de los dos bits más significativos de la dirección (de los cinco bits en total), porque el campo de índice se forma con los tres bits menos significativos de la dirección, los cuales sirven para seleccionar el bloque. Los diseñadores de computadoras omiten los bits de índice debido a que son redudundantes, ya que por definición, el campo de índice de cualquier dirección de bloque de cache, es precisamente, el número de bloque dentro de la cache.

También es necesario reconocer cuándo un bloque de cache no tiene información válida. Por ejemplo cuando un procesador arranca, la cache no tendrá datos buenos y el campo de etiqueta no tendrá ningún significado. Aun después de ejecutar varias instrucciones, algunas entradas de la cache podrán seguir vacías, como en la Figura 1. Por eso es necesario saber que para esas entradas, la etiqueta debe ser ignorada. El método más común es agregar un *bit de validez* para indicar si una entrada contiene una dirección válida. Si el bit no está puesto en uno, no puede haber coincidencia para ese bloque.

En esta parte del documento, la explicación se concentra en cómo trabajan las cache con las lecturas. En general, manejar las lecturas es un poco más simple que manejar las escrituras, porque las lecturas no tienen que cambiar el contenido de la cache. Después de revisar lo básico de cómo trabajan las cache con las lecturas y de cómo los fallos de cache

pueden ser manejados, se examina el diseño de las cache de las computadoras reales y los detalles de cómo estas caches manejan las escrituras.

Visión ampliada. El uso de las caches quizás sea el ejemplo más importante de la gran idea de **predicción**. Se basa en el principio de localidad, al tratar de encontrar los datos deseados en los niveles más altos de la jerarquía, además provee los mecanismos para asegurar que cuando la predicción falla, encuentra y usa los datos apropiados desde los niveles más bajos de la jerarquía de memoria. La tasa de éxito (en inglés *hit*) de la predicción de la cache en las computadoras modernas frecuentemente supera el 95%.

Accediendo a una cache. Debajo hay una secuencia de 9 referencias a memoria a través de una cache vacía de 8 bloques, cada referencia incluye el efecto que produce. La Figura 3 muestra cómo los contenidos de la cache cambian con cada fallo. Debido a que hay ocho bloques en la cache, los tres bits de menor peso de cada dirección dan el número de bloque:

Dirección decimal de referencia	Dirección binaria de referencia	Hit o miss en cache	Bloque asignado (donde encontrar o ubicar)
22	10110_{dos}	miss (3.b)	$(10110_{dos}\ mod\ 8) = 110_{dos}$
26	11010_{dos}	miss (3.c)	$(11010_{dos}\ mod\ 8) = 010_{dos}$
22	10110_{dos}	hit	$(10110_{dos}\ mod\ 8) = 110_{dos}$
26	11010_{dos}	hit	$(11010_{dos}\ mod\ 8) = 010_{dos}$
16	10000_{dos}	miss (3.d)	$(10000_{dos}\ mod\ 8) = 000_{dos}$
3	00011_{dos}	miss (3.e)	$(00011_{dos}\ mod\ 8) = 011_{dos}$
16	10000_{dos}	hit	$(10000_{dos}\ mod\ 8) = 000_{dos}$
18	10010_{dos}	miss (3.f)	$(10010_{dos}\ mod\ 8) = 010_{dos}$
16	10000_{dos}	hit	$(10000_{dos}\ mod\ 8) = 000_{dos}$

Debido a que la cache está vacía, varias de las primeras referencias son fallos; la Figura 3 describe el estado inicial y el posterior a cada fallo: inicialmente todos los bits de validez (entrada V de la cache) están deshabilitados (N). El procesador requiere las siguientes direcciones: 10110_{dos}(miss), 11010_{dos}(miss), 10110_{dos}(hit), 11010_{dos}(hit), 10000_{dos} (miss), 00011_{dos} (miss), 10000_{dos} (hit), 10010_{dos} (miss), 10000_{dos} (hit).

La figura muestra los contenidos de la cache después de que cada fallo de la secuencia ha sido manejado. Cuando la dirección 10010_{dos} (18) es referenciada, la entrada para la dirección 11010_{dos} (26) debe ser reemplazada, y una posterior referencia a 11010_{dos} causará un nuevo miss. El campo tag contiene solamente la porción superior de la dirección. La dirección completa de una palabra contenida en el bloque i de la cache con un campo de tag j para esta cache, es $j \times 8 + i$, o lo que es lo mismo la concatenación del campo j y el índice i. Por ejemplo, en la cache f de la Figura 3, el índice 010_{dos} tiene tag 10_{dos} y corresponde a la dirección 10010_{dos}.

En la octava referencia hay una demanda conflictiva a un bloque. La palabra en la dirección 18 (10010_{dos}) debe ser cargada en el bloque 2(010_{dos}) de la cache. Por eso, se debe reemplazar la palabra de la dirección 26(11010_{dos}) la cual ya estaba en el bloque 2(010_{dos}) de la cache. Esta característica le permite a la cache tomar ventaja de la localidad temporal: las palabras recientemente referenciadas reemplazan a las palabras menos recientemente referenciadas.

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. The initial state of the cache after power-on

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

b. After handling a miss of address (10110_{two})

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

c. After handling a miss of address (11010_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

d. After handling a miss of address (10000_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

e. After handling a miss of address (00011_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

f. After handling a miss of address (10010_{two})

FIGURE 3. Contenido de la cache después de cada referencia que falla, con los campos índice y tag en binario para la secuencia dada.

La situación es directamente análoga a necesitar un libro desde las estanterías y no tener más espacio en el escritorio – algunos libros que ya están en el escritorio deben ser retornados a las estanterías. En una cache de correspondencia directa, hay solamente un lugar para poner el ítem recientemente requerido y solamente una opción a cual reemplazar.

Ya se sabe donde mirar en la cache para cada posible dirección: los bits de menor peso de una dirección pueden ser usados para encontrar la única entrada a la cual la dirección podría corresponder. La Figura 4 muestra como se divide una referencia a memoria:

- Un campo tag, usado para comparar con el valor del campo tag de la cache.
- Un índice de cache, el cual es usado para seleccionar un bloque.

Esta cache mantiene 1024 palabras o 4 Kibibytes[2]. Se asume una dirección de 32 bits. El tag desde la cache es comparado con la parte superior de la dirección para determinar si la entrada en la cache corresponde con la dirección solicitada. Debido a que la cache tiene 2^{10} (o 1024) palabras y un tamaño de bloque de una palabra, 10 bits son usados para indexar

[2]Para resolver la ambigüedad entre la notación 2^x vs 10^y (como Kibibytes=2^{10} bytes vs. Kilobyte=10^3 bytes) es preferible usar las potencias de dos.

la cache, dejando $32 - 10 - 2 = 20$ bits para ser comparados con el tag. Si el tag y los 20 bits de la dirección son iguales y el bit de validez está en ON, entonces el requerimiento es exitoso en la cache y la palabra es provista al procesador. En otro caso ocurre un fallo (o *miss* en ingés).

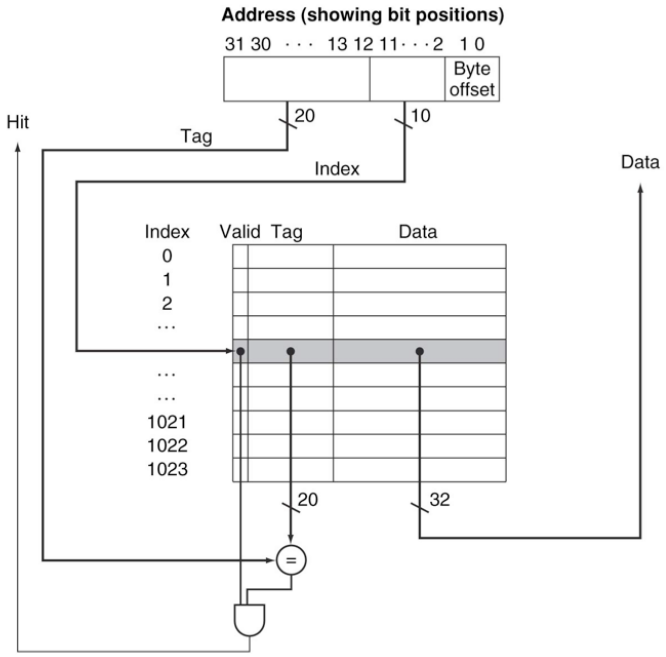

FIGURE 4. En esta cache la porción baja de la dirección se usa para seleccionar una entrada, la cual consiste de una palabra de dato y un tag.

El índice de un bloque de cache, junto con el tag del bloque, especifican de manera unívoca la dirección de la palabra contenida en ese bloque de cache. Debido a que el campo de índice es usado como una dirección para referenciar a la cache, y debido a que un campo de n-bits tiene 2^n valores, el número total de entradas en una cache de correspondencia directa debe ser una potencia de dos. En la arquitectura MIPS, debido a que las palabras están alineadas a múltiplos de cuatro bytes, los dos bits menos significativos de cada dirección especifican un byte dentro de una palabra. Por eso, los dos bits menos significativos son ignorados cuando se selecciona una palabra en el bloque.

El número total de bits necesarios para una cache es una función del tamaño de la cache y del tamaño de la dirección, porque la cache incluye a ambos, el almacenamiento para el dato y para el tag. El tamaño de bloque en el caso anterior fue de una palabra, pero normalmente tendrá varias palabras. Para la siguiente situación:

- Direcciones de 32 bits.
- Cache de correspondencia directa.
- El tamaño de cache es 2^n bloques, de esa manera se usan n bits para el índice.

- Un bloque de tamaño 2^m palabras (2^{m+2} bytes), de manera que m bits de la dirección son usados para identificar una palabra dentro del bloque, y otros dos bits más son usados para identificar un byte en el bloque.

el tamaño del campo tag es:

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

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

$$2^n \times (\textit{tamaño del bloque} + \textit{tamaño del tag} + \textit{tamaño del campo de validez}).$$

Debido a que el tamaño del bloque es 2^m palabras (2^{m+5} bits) y es necesario 1 bit para el campo de validez, el número total de bits en esta cache es:

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

Aunque este es el tamaño real en bits, la convención para dar nombres excluye el tamaño del tag y del campo de validez, se considera solamente el tamaño de los datos. De esa manera la cache en la Figura 4 es llamada una cache de 4 KiB (Kibibytes).

Ejemplo. Bits en la cache. ¿Cuántos bits en total son requeridos para una cache de correpondencia directa con 16 KiB de datos y bloques de 4 palabras, asumiendo una dirección de 32 bits?

Respuesta. Sabemos que 16 KiB son 4096 (2^{12}) palabras. Con un tamaño de bloque de 4 palabras (2^2), hay 1024 (2^{10}) bloques. Cada bloque tiene 4×32 ó 128 bits de datos más un tag, el cual es de $32 - 10 - 2 - 2$ bits, más un bit de validez. Así, el tamaño total de la cache es:

$$2^{10} \times (4 \times 32 + (32 - 10 - 2 - 2) + 1 = 2^{10} \times 147 = 147\ \textit{Kibibits}$$

ó $18,4$ KiB para una cache de 16 KiB. Para esta cache el número total de bits en la cache, es aproximadamente 1,15 veces lo necesario para el tamaño de los datos.□

Ejemplo. Correspondencia de una dirección en una cache de bloque multipalabra. Considere una cache con 64 bloques y un tamaño de bloque de 16 bytes. ¿A qué número de bloque corresponde la dirección de byte 1200?

Respuesta. Más arriba hemos visto la fórmula. La fórmula es:

$$(\textit{Dirección del bloque})\ \textit{módulo}\ (\textit{Número de bloques en la cache})$$

donde la dirección del bloque está dada por

$$\frac{\textit{Dirección a byte}}{\textit{Bytes por bloque}}$$

Notar que esta dirección de bloque corresponde al bloque que contiene todas las direcciones entre

$$\left[\frac{\textit{Dirección a byte}}{\textit{Bytes por bloque}}\right] \times \textit{Bytes por bloque}$$

y

$$\left[\frac{\textit{Dirección a byte}}{\textit{Bytes por bloque}}\right] \times \textit{Bytes por bloque} + (\textit{Bytes por bloque} - 1)$$

De esta manera, con 16 bytes por bloque, la dirección a byte 1200 se corresponde a la dirección de bloque

$$\left[\frac{1200}{16}\right] = 75$$

que está asociado con el bloque de cache (75 módulo 64)=11. En efecto, este bloque corresponde con todas las direcciones entre 1200 y 1215.□

Los bloques más grandes explotan la localidad espacial reduciendo la tasa de fallos. Como lo muestra la Figura 5, incrementar el tamaño de los bloques normalmente decrementa la tasa de fallos. La tasa de fallos puede crecer eventualmente si el tamaño del bloque se convierte en una fracción significativa del tamaño de la cache, porque el número de bloques que pueden ser mantenidos en la cache se hace pequeño, y habrá una gran competencia por esos bloques. Como resultado un bloque será expulsado de la cache antes que un alto porcentaje de sus palabras hayan sido accedidas. Lo que es lo mismo, la localidad espacial entre las palabras de un bloque decrementa cuando el bloque es muy grande; en consecuencia, los bebeficios en la tasa de fallos se hace menor.

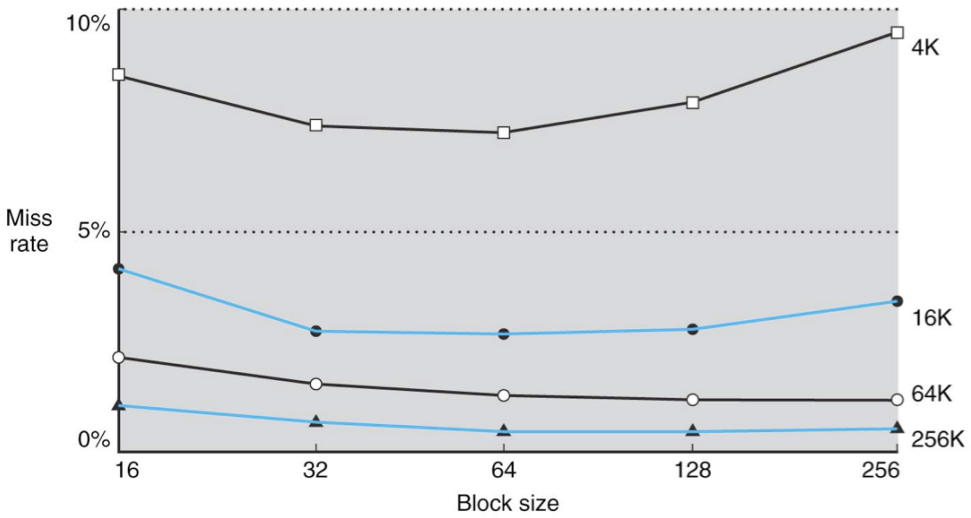

FIGURE 5. Tasa de fallo versus tamaño del bloque.

Un problema más serio asociado con simplemente incrementar el tamaño del bloque, es que se incrementa el costo de los fallos. La penalidad por miss es determinada por el tiempo requerido para recuperar el bloque desde el nivel inferior en la jerarquía y su correspondiente carga en la cache. El tiempo para recuperar el bloque tiene dos partes: la latencia para la primer palabra del bloque y el tiempo de tranferencia del resto del bloque. Claramente, a menos que cambiemos el sistema de memoria, el tiempo de transferencia – de aquí en más la penalidad por fallo – se incrementa a medida que el tamaño del bloque se incrementa. Por lo tanto, la mejora en la tasa de fallo comienza a decrementar en la medida que el bloque se hace más grande. El resultado es que el incremento en la penalidad

por fallo supera el decremento en la tasa de fallo para bloques que son demasiado grandes, y el desempeño de la cache, en definitiva, empeora. Desde luego, si se diseña una memoria capaz de transferir grandes bloques de manera eficiente, se puede incrementar el tamaño de bloque y obtener mejoras en el desempeño de las cache.

Manejo de fallos de cache. Antes de analizar la cache de un sistema real, veremos como trata la unidad de control los fallos de cache. Los detalles del controlador de cache se verán más adelante. La unidad de control debe detectar un miss y procesar el miss recuperando los datos requeridos desde la memoria (o una cache de nivel inferior). Si la cache reporta un hit, la computadora continúa usando los datos como si nada hubiese pasado.

Concepto 2. ***fallo de cache****: cuando el requerimiento de un dato a la cache no puede ser satisfecho debido a que el dato no está presente en la cache.*

Modificar el comportamiento del procesador para manejar un hit es trivial; los miss sin embargo requieren algún trabajo adicional. El manejo de los miss de cache es hecho en colaboración entre la unidad de control del procesador y un controlador separado que inicia el acceso a memoria y rellena la cache. El procesamiento de un miss de cache crea un atasco del pipe, en lugar de una interrupción, lo cual podría requerir guardar el estado de todos los registros. Para un miss de cache, se podría detener todo el procesador, esencialmente congelando el contenido de los registros temporarios y de los registros visibles al programador, mientras se espera por la memoria. Los procesadores más sofisticados, los llamados *fuera de orden*, permiten la ejecución de instrucciones mientras se espera por el miss de cache, por el momento se asumirá que los procesadores *en orden* se atascan cuando ocurre un miss de cache.

Comenzamos con el manejo de los miss de instrucciones, la misma aproximación se puede usar para manejar los miss de datos. Si el acceso a una instrucción resulta en un miss, entonces el contenido del *registro de instrucción* es inválido. Para conseguir que la instrucción apropiada se cargue en la cache, hay que indicarle al nivel inferior de la jerarquía de memoria que realice la lectura. Debido a que el contador de programa se incrementa en el primer ciclo de la ejecución, la dirección de la instrucción que genera el miss de cache por instrucción es igual al valor del PC menos 4. Una vez que se tiene la dirección de la instrucción, se debe instruir a la memoria para que realice la lectura. Se debe aguardar la respuesta de la memoria (debido a que el acceso tomará múltiples ciclos), y luego se procede a escribir en la cache las palabras que contienen la instrucción deseada.

Es posible definir los pasos a ser tomados por un miss de cache por instrucción:

(1) Enviar el valor del PC original (se envía PC-4) a la memoria.
(2) Instruir a la memoria principal a realizar la lectura y esperar que la memoria complete su acceso.
(3) Escribir en la entrada de la cache, poniendo el dato proveniente de la memoria en la porción de datos de la entrada, escribir los bits superiores de la dirección en el campo tag y colocar el bit de validez en uno.
(4) Recomenzar la ejecución de la instrucción desde el primer paso, para recuperar la instrucción, que esta vez se encontrará en la cache.

El control de la cache para el acceso a datos es esencialmente idéntico: en un miss de datos simplemente se atasca el procesador hasta que la memoria responda con el dato.
Manejo de las escrituras. Las escrituras funcionan de manera diferente a las lecturas. Considere una instrucción store, la escritura se hace solamente en la cache de datos (sin cambiar la memoria principal); entonces después de escribir en la cache, la memoria podría tener un valor diferente al de cache. En tal caso, se dice que la memoria y la cache son *inconsistentes*. La forma más simple de mantener la memoria y la cache consistentes es siempre escribir el dato en ambas: la memoria y la cache. Este esquema es llamado *escritura directa* (en inglés *write-through*).

Concepto 3. ***write-through****, es un esquema en el cual las escrituras siempre actualizan tanto la cache como el siguiente nivel inferior de la jerarquía de memoria, garantizando que los datos son siempre consistentes entre ambos.*

Otro aspecto clave de las escrituras es saber que pasa con los fallos de escritura. Primero se recuperan las palabras desde el bloque de memoria. Después que el bloque es recuperado y ubicado en la cache, se puede sobreescribir la palabra que causó el miss en el bloque de cache. También se escribe la palabra en la memoria principal usando la dirección completa.

Aunque este esquema maneja las escrituras de manera muy simple, podría no tener buen desempeño. Con un esquema write-through, cada escritura causa que el dato sea escrito en la memoria principal. Estas escrituras toman bastante tiempo, probablemente unos 100 ciclos del procesador y podría enlentecer el procesador considerablemente. Por ejemplo supongamos que el 10% de las instrucciones son stores. Si el CPI sin miss de cache era de 1, gastando 100 ciclos extras por cada escritura, podría llevar el CPI a $1+100\times10\% = 11$, reduciendo el desempeño por un factor mayor a 10.

Concepto 4. ***Buffer de escritura****, es una cola de espera que mantiene los datos mientras esperan ser escritos en memoria.*

Una solución al problema de las demoras por fallos en las escrituras, es usar un *buffer de escritura*. El buffer de escritura mantiene un dato mientras espera ser escrito en la memoria. Después de haber escrito el dato en la cache y en el buffer de escritura, el procesador puede continuar la ejecución. Cuando una escritura a la memoria se completa, se libera una entrada en el buffer de escritura. Si el buffer de escritura está lleno cuando el procesador alcanza una escritura, el procesador debe esperar hasta que haya una posición vacía en el buffer de escritura. Por supuesto, si la tasa en la cual la memoria puede completar las escrituras es menor que la tasa en la cual el procesador está generando escrituras, ningún tamaño de buffer puede ser de ayuda, porque las escrituras están siendo generadas más rápido de lo que el sistema de memoria puede aceptar.

La tasa a la cual las escrituras son generadas puede ser menor que la tasa a la cual la memoria puede aceptarlas y aún pueden ocurrir atascos. Esto puede suceder cuando las escrituras ocurren en ráfagas. Para reducir la ocurrencia de tales atascos, el procesador usualmente incrementa la profundidad del buffer de escritura más allá de una única entrada.

La alternativa a un esquema write-through es el esquema llamado *escritura demorada* (en inglés *write-back*). En un esquema write-back, cuando ocurre una escritura, el nuevo

valor sólo es escrito en el bloque de la cache. El bloque modificado es escrito en el nivel inferior de la jerarquía cuando el bloque debe ser reemplazado en la cache. El esquema write-back puede mejorar el desempeño, especialmente cuando el procesador puede generar escrituras tanto o más rápido que lo que puede manejar la memoria principal; sin embargo el esquema write-back es más complejo de implementar que el write-through.

Concepto 5. *Write-back, es un esquema que maneja las escrituras actualizando valores únicamente en la cache luego, cuando el bloque debe ser reemplazado, escribe el bloque modificado en el nivel inferior de la jerarquía.*

En lo que resta del documento, se describe la cache de un sistema real y se examina como maneja las lecturas y las escrituras.

Elaboración. Las escrituras introducen varias complicaciones en las cache que no se presentan en las lecturas. Aquí se discuten dos de ellos: la política sobre los miss de escritura y la implementación eficiente de las escrituras en las cache write-back.

Consideremos un fallo por escritura en una cache write-through. La estrategia más común, llamada *alojar para escribir* (en inglés *write allocate*), consiste en alojar el bloque referenciado en la cache. El bloque es recuperado de memoria y luego la porción apropiada del bloque se sobrescribe. Una estrategia alternativa, llamada *no alojar para escribir* (en inglés *no write allocate*), consiste en actualizar una porción del bloque en la memoria sin ubicar el bloque en la cache. La motivación es que aveces los programas escriben bloques enteros de palabras, como ser cuando un sistema operativo llena de ceros una página de memoria. En tales casos, el fetch asociado con el fallo de lectura inicial puede ser innecesario. Algunas computadoras permiten aplicar la política de *alojar para escribir* de acuerdo a la página considerada.

En verdad, implementar las escrituras de manera eficiente en una cache que usa una estrategia write-back, es más complejo que en una cache write-through. Una cache write-through puede escribir los datos en la cache y leer el tag; si el tag no coincide, entonces ocurre el miss. Esto es así porque en la cache write-through la sobrescritura del bloque no es catastrófica, ya que la memoria mantiene el valor correcto. En la cache write-back si el dato en la cache tiene modificaciones, primero es necesario escribir el bloque de nuevo en la memoria y luego hay un miss de cache. Si directamente se sobrescribe el bloque en una instrucción store, antes de saber si la grabación tendrá éxito (como se hizo con la cache write-through), se podría destruir el contenido del bloque, que no está resguardado en el nivel inferior de la jerarquía de memoria.

En una cache write-back, debido a que no se puede sobrescribir el bloque, los stores requieren dos ciclos (un ciclo para controlar si es un hit, seguido de un ciclo que efectivamente haga la escritura), o bien, requieren de un buffer de escritura para mantener los datos – permitiendo efectivamente que el store tome solamente un ciclo del pipe. Cuando un buffer de almacenamiento es usado, el procesador recupera los datos de la cache y los ubica en el buffer de escritura durante el ciclo normal de acceso a cache. Asumiendo un hit de cache, el nuevo dato es escrito desde el buffer de escritura a la cache en el siguiente ciclo de acceso a la cache no usado.

Por su parte, en la cache write-through las escrituras siempre pueden ser hechas en un ciclo. Se lee el tag y se escribe la porción de datos del bloque seleccionado. Si el tag corresponde a la dirección del bloque a escribir, el procesador puede continuar normalmente, debido a que el bloque correcto ha sido actualizado. Si el tag no coincide, el procesador genera un fallo por escritura para recuperar el resto del bloque correspondiente a esa dirección.

Muchas cache write-back también incluyen buffers de escritura que son usados para reducir la penalidad por miss, cuando un miss reemplaza un bloque modificado. En tal caso el bloque modificado es movido a un buffer write-back asociado con la cache mientras el bloque requerido es leído desde memoria. Asumiendo que otro miss no ocurra inmediatamente, esta técnica alivia la penalidad por miss cuando un bloque modificado debe ser reemplazado.

Una cache como ejemplo: El procesador FastMATH de Intrinsity. El FastMATH de Intrisity es un micro-procesador embebido que usa la arquitectura MIPS y una implementación simple de cache. Luego veremos algunos diseños más complejos, pero ahora comenzamos con este ejemplo simple pero real. La Figura 6 muestra la organización de la cache de datos del FastMATH.

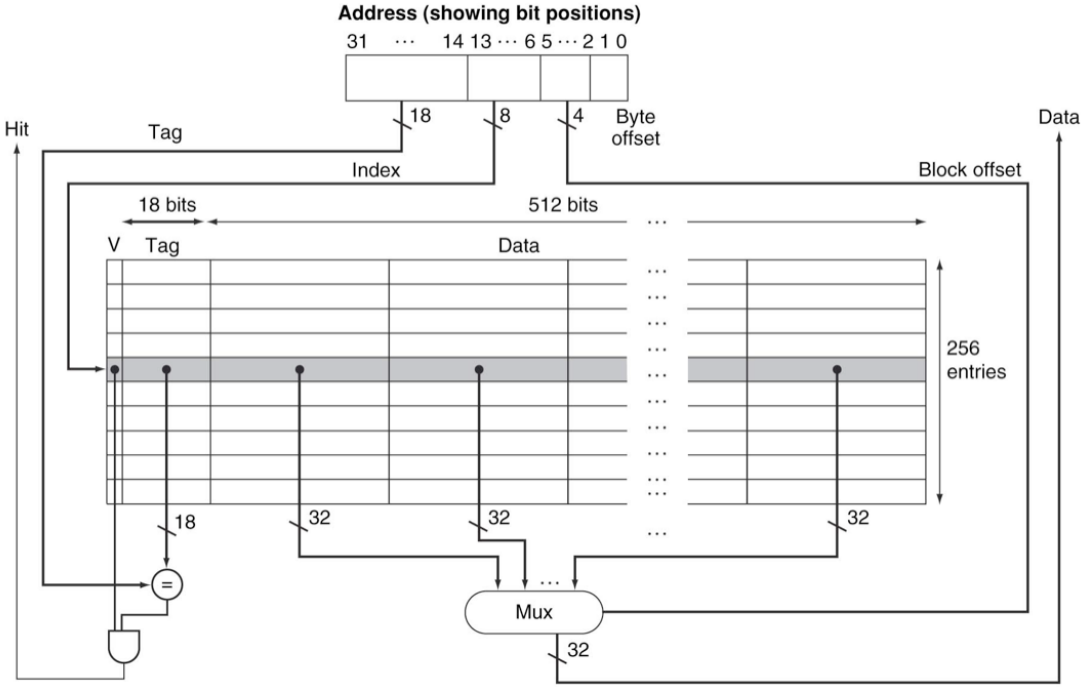

FIGURE 6. Las caches de 16KiB en el FastMATH, cada una contiene 256 bloques de 16 palabras.

Este procesador tiene un pipe de 12 etapas. Cuando opera a máxima velocidad, el procesador puede requerir en cada ciclo una palabra de instrucción y una palabra de dato. Para

satisfacer la demanda del procesador sin atascos, se usan cache separadas para instrucciones y datos. Cada cache tiene 16 KiB, o 4096 palabras, con bloques de 16 palabras.

Los requerimientos de lectura para la cache son sencillos. Debido a que hay caches separadas para instrucciones y datos, es necesario separar las señales de control de cada cache para leer y escribir. (Recordar que es necesario actualizar la cache de instrucciones cuando ocurre un miss) Así, los pasos para un requerimiento de lectura a cualquiera de las cache, son lo siguientes:

(1) Enviar la dirección a la cache apropiada. La dirección llega o bien desde el PC (para una instrucción) o desde la ALU (para datos).
(2) Si la cache indica un hit, la palbra requerida está disponible en las líneas de datos. Debido a que hay 16 palabras en el bloque apuntado, se debe seleccionar la correcta. Un campo hace de índice dentro del bloque y es usado para controlar el multiplexor (mostrado en la base de la figura), éste selecciona la palabra requerida entre las 16 palabras en el bloque indicado.
(3) Si la cache indica un miss, se envía la dirección a la memoria principal. Cuando la memoria devuelve el dato, se lo escribe en la cache y luego se lo lee para finalizar con el requerimiento.

El FastMATH ofrece dos alternativas para las escrituras: write-through y write-back, dejando al sistema operativo la decisión de que estartegia usar según la aplicación. Tiene un buffer de escritura de una entrada.

¿Qué tasa de fallo de cache se espera teniendo una estructura como la usada por el FastMATH?. La figura 7 muestra la tasa de fallo para las caches de instrucciones y de datos. La tasa de fallo combinada es la efectiva tasa de fallo por referencia para cada programa después de considerar las diferentes frecuencias de accesos a instrucciones y datos.

Aunque la tasa de fallos es una característica importante del diseño de la cache, la medida de mayor interés es el impacto del sistema de memoria sobre el tiempo de ejecución del programa; más adelante veremos cómo se relacionan la tasa de fallo y el tiempo de ejecución.

Instruction miss rate	Data miss rate	Effective combined miss rate
0.4%	11.4%	3.2%

FIGURE 7. Tasa de fallos aproximada para instrucciones y datos en el FastMATH con un grupo de programas.

Elaboración. Una cache combinada con un tamaño total igual a la suma de las dos caches divididas (en inglés *split*) usualmente tendrá una mejor tasa de hit. Esta mayor tasa ocurre debido a que una cache combinada no divide rígidamente el número de entradas que pueden ser usadas por las instrucciones de aquellas que pueden ser usadas por los datos. Pese a esto, casi todos los procesadores actuales usan cache de instrucciones y datos dividida para incrementar el ancho de banda (en inglés *bandwidth*) de la cache, y de ese modo, cumplir

con las expectativas de los procesadores segmentados. (También pueden haber menos fallos conflictivos: aquellos que dan origen a otros fallos). Aquí están las tasas de fallos para los tamaños de cache del procesador FastMATH, y para una cache combinada cuyo tamaño es igual a la suma de las dos caches:

- Tamaño total de cache: 32KiB
- Tasa de fallo efectiva de la cache dividida: 3,24%
- Tasa de fallo de la cache combinada: 3,18%

La tasa de fallo de la cache dividida es apenas peor.

La ventaja de duplicar el ancho de banda de la cache, permitiendo acceder simultaneamente a una instrucción y a un dato, claramente supera la desventaja de una tasa de fallos algo más grande. Esta observación nos advierte de que no se puede considerar a la tasa de fallos como la única medida del desempeño de la cache.

Resumen. Este documento comienza examinando las cache más simples: las cache de correspondencia-directa, con un tamaño de bloque de una palabra. En tales cache, tanto los hits como los misss son simples, porque una palabra puede ir en un único lugar y hay un tag separado para cada palabra. Para mantener la consistencia entre la cache y la memoria, se usa un esquema write-through. La alternativa a write-through es write-back que recién cuando un bloque debe ser reemplazado, copia el contenido actualizado en memoria.

Para tomar ventaja de la localidad espacial, la cache debe tener un bloque de tamaño mayor a una palabra. El uso de un bloque más grande decrementa la tasa de miss y mejora la eficiencia de la cache, ya que mejora la relación entre la cantidad de almacenamiento en la cache destinada al tag, respecto del almacenamiento de datos. Aunque un bloque grande decrementa la tasa de miss, incrementa la penalidad por miss. Si la penalidad por miss creció linealmente con el tamaño del bloque, los bloques grandes pueden fácilmente llevar a una baja del desempeño.

Para evitar pérdida de desempeño, se incrementa el ancho de banda de la memoria principal; esto permite transferir los bloques de cache de manera más eficiente. Para incrementar el ancho de banda por afuera del DRAM, comúnmente se usan dos métodos: aumentar el ancho de la memoria o interliving. Los diseñadores de las DRAM han mejorado de manera sostenida la interfaz entre el procesador y la memoria; el incremento del ancho de banda para la transferencia en modo ráfaga, reduce el costo de transferencia de los bloques grandes.

References

[1] Patterson, David A. and Hennessy, John L., Computer Organization and Design, Fifth Edition: The Hardware/Software Interface, Morgan Kaufmann Publishers Inc., 2014