Arquitectura y Organización de Computadoras II

Cache

MSc. Ing. Ticiano J. Torres Peralta
Ing. Pablo Toledo

2023 – Segundo Cuatrimestre

Universidad Nacional de Tucumán
Facultad de Ciencias Exactas y Tecnología

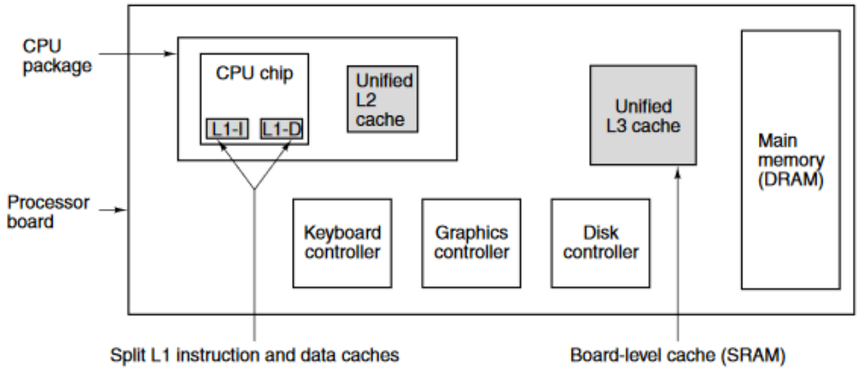
Cache: Lo Básico

Uno de los aspectos más desafiantes del diseño de computadoras a lo largo de la historia ha sido tratar de diseñar un sistema de memoria lo suficientemente rápido como la capacidad del procesador para procesar instrucciones y datos. La alta tasa de crecimiento del rendimiento de los procesadores no ha ido acompañada de una correspondiente aceleración de las memorias; En relación con la CPU, las memorias se han vuelto considerablemente más lentas.

Los procesadores modernos imponen una demanda abrumadora al sistema de memoria, tanto en términos de latencia como de ancho de banda. A medida que las velocidades de reloj del procesador aumentan, resulta más difícil proporcionar un sistema de memoria capaz de suministrar datos en uno o dos ciclos de reloj. Una forma de atacar este problema es proporcionando cachés.

Cache: Lo Básico

Las cachés aparecieron por primera vez en computadoras de investigación a principios de la década de 1960 y más tarde en esa misma década en computadoras de producción. Hoy en día, todas las computadoras de uso general, desde servidores hasta procesadores integrados de bajo consumo, incluyen cachés. En las primeras computadoras comerciales, se eligió las caché para representar el nivel de jerarquía de memoria entre el procesador y la memoria principal. Aunque este sigue siendo el uso principal de un caché, el término también se utiliza para referirse a cualquier almacenamiento temporario de accesso rápido.



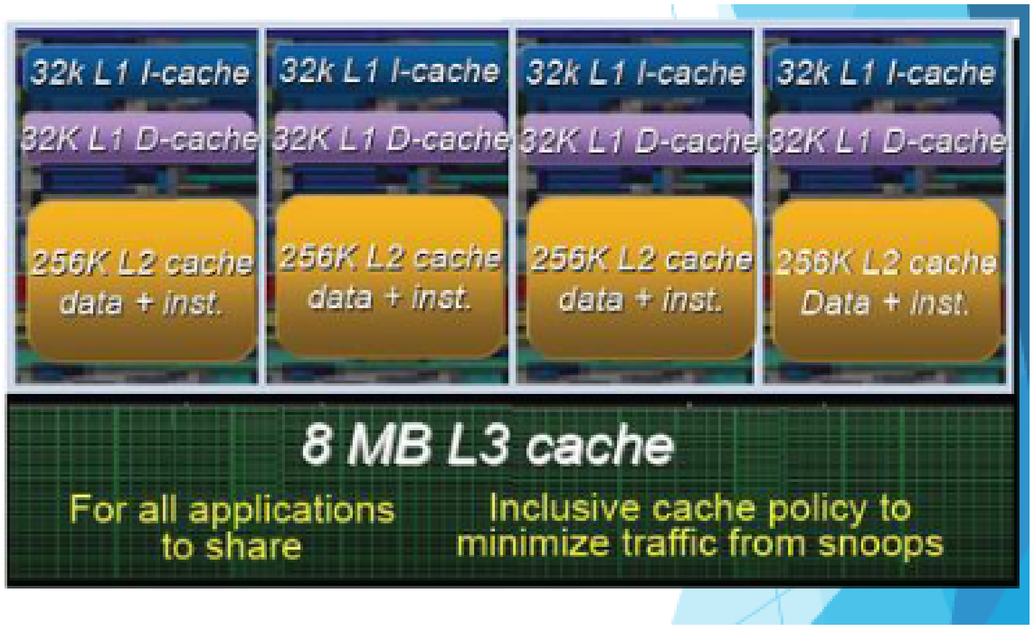
Cache: Lo Básico

Las cachés guardan las palabras utilizadas más recientemente en una memoria pequeña y rápida, lo que acelera el acceso a ellas. Una de las formas más efectivas de mejorar tanto el ancho de banda como la latencia es proporcionar múltiples cachés. Por ejemplo, una técnica que funciona de manera muy efectiva es introducir un caché separado para instrucciones y datos. Esto se llama **caché dividida**. Esto significa que las operaciones de memoria se pueden iniciar de forma independiente en cada caché, duplicando efectivamente el ancho de banda. Además cada caché tiene acceso independiente a la memoria principal.

Hoy en día, la mayoría de los sistemas de memoria tienen varios niveles de caché, algunos con tres o más niveles.

Cache: Lo Básico

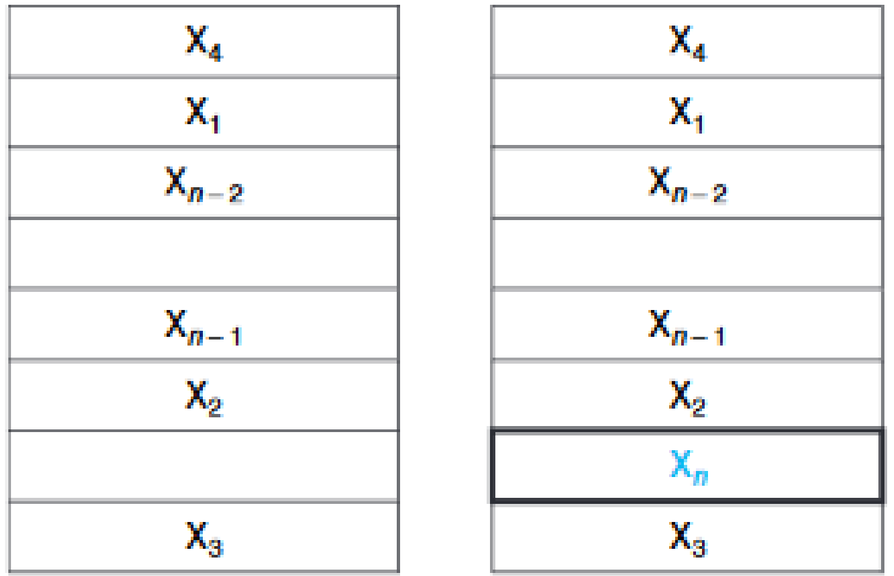
En la siguiente figura vemos un sistema con tres niveles de caché. Los tamaños típicos para la caché L1 son entre 16 KiB a 64 KiB, para la caché L2 entre 512 KiB a 1 MiB y L3 puede tener varios MiB. Las cachés son generalmente **inclusivas**, que significa que el contenido completo de L1 se encunetra en L2 y el contenido completo de L2 en L3.



Cache: Lo Básico

Todas las cachés están divididas en bloques de tamaño fijo llamados **líneas de caché**. Una línea de caché normalmente consta entre 4 a 64 Bytes y se numeran consecutivamente comenzando en 0.

Consideremos un ejemplo simple donde las solicitudes del procesador son de una palabra y los bloques (en la caché) también. La siguiente figura muestra tal caché, antes y después de solicitar un objeto que no está inicialmente en ella. Podemos ver que al no estar la palabra solicitada se produce una **falla** y la palabra Xn se la trae de memoria principal.



a. Before the reference to X_n b. After the reference to X_n

Cache: Mapeo Directo

Observando este escenario, hay dos preguntas que deben responderse:

1. ¿Cómo sabemos si un elemento de datos está en la caché?

2. Si está en la caché, ¿cómo sabemos adonde encontrarlo?

Las respuestas en realidad están relacionadas. Supongamos que, si cada palabra puede ir exactamente en una línea especifica de la caché, entonces sería sencillo encontrara. Entonces podríamos decir que, la forma más sencilla de asignar esta ubicación es asignarla en función de la dirección que dicha palabra tiene en memoria principal. Este tipo de caché se denomina **caché de mapeo directo**. Aquí, cada dirección en memoria principal se asigna directamente a exactamente una línea en la caché.

Cache: Mapeo Directo

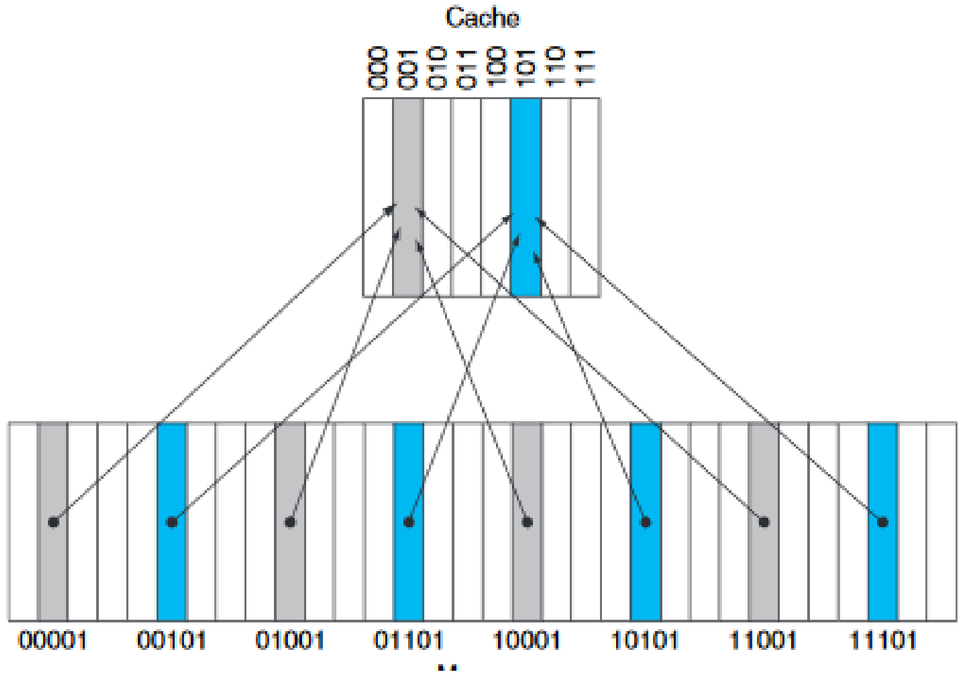
Por ejemplo, casi todas las cachés de mapeo directo utilizan la siguientes formulas para encontrar un bloque:

$$Block\ Address = \frac{Byte\ address}{Bytes\ per\ block}$$

$$Block\ Address\ \%\ Number\ of\ blocks \in the\ cache$$

Cache: Mapeo Directo

Las formulas anteriores tienen la particularidad que, si el número de entradas en la caché es una potencia de 2, entonces el módulo puede ser computado simplemente usando los log2(numero de bloques en la caché) bits menos significante la dirección en memoria principal. La siguiente figura muestra cómo las direcciones de memoria entre 00001 y 11101 se asigna a las ubicaciones 001 y 101 en una caché de mapeo directo de ocho bloques.



Cache: Mapeo Directo

La segunda pregunta ya la tenemos resuelta; ahora consideremos cómo responder la primera, pero formulada de manera un poco diferente. Si cada ubicación de la caché puede contener contenidos de varias direcciones diferente de memoria principal,

¿cómo sabemos si los datos de la caché corresponden a la palabra solicitada?

Podemos responder a esta pregunta agregando un campo de **etiqueta** al bloque de la caché. La etiqueta tendrá la información necesaria para identificar si la palabra en caché corresponde a la palabra solicitada. Siguiendo nuestro ejemplo, la etiqueta sólo necesita tener la parte superior de la dirección, en otras palabras, los bits mas significativos que no se utilizaron para producir el índice de la línea. Entonces, solo necesitamos tener los 2 bits mas significativos de los 5 bits de dirección para la etiqueta. (La etiqueta puede técnicamente, si se desea, contener todos los 5 bits de dirección, pero los arquitectos omiten los bits de índice porque son redundantes y solo tomarían espacio.)

Cache: Mapeo Directo

Otra cosa que debemos tener en cuenta es que necesitamos una forma de reconocer si un bloque de caché tiene información válida. Por ejemplo, cuando se inicia el procesador, la caché no tiene datos válidos y los campos de etiquetas no tendrán sentido. Incluso después de ejecutar muchas instrucciones, es posible que algunos de los bloques de caché sigan vacíos. Por lo tanto, necesitamos saber que se debe ignorar la etiqueta para dichas entradas. El método más común para tratar este tema es agregar **un bit de validación** para indicar si una entrada contiene una dirección válida.

Cache: Acceso

Observemos un ejemplo más concreto de una secuencia de referencias de memoria a un caché vacío de ocho bloques, incluida la acción para cada referencia.

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.6b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (5.6c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (5.6d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (5.6e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (5.6f)	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

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	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	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	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	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		

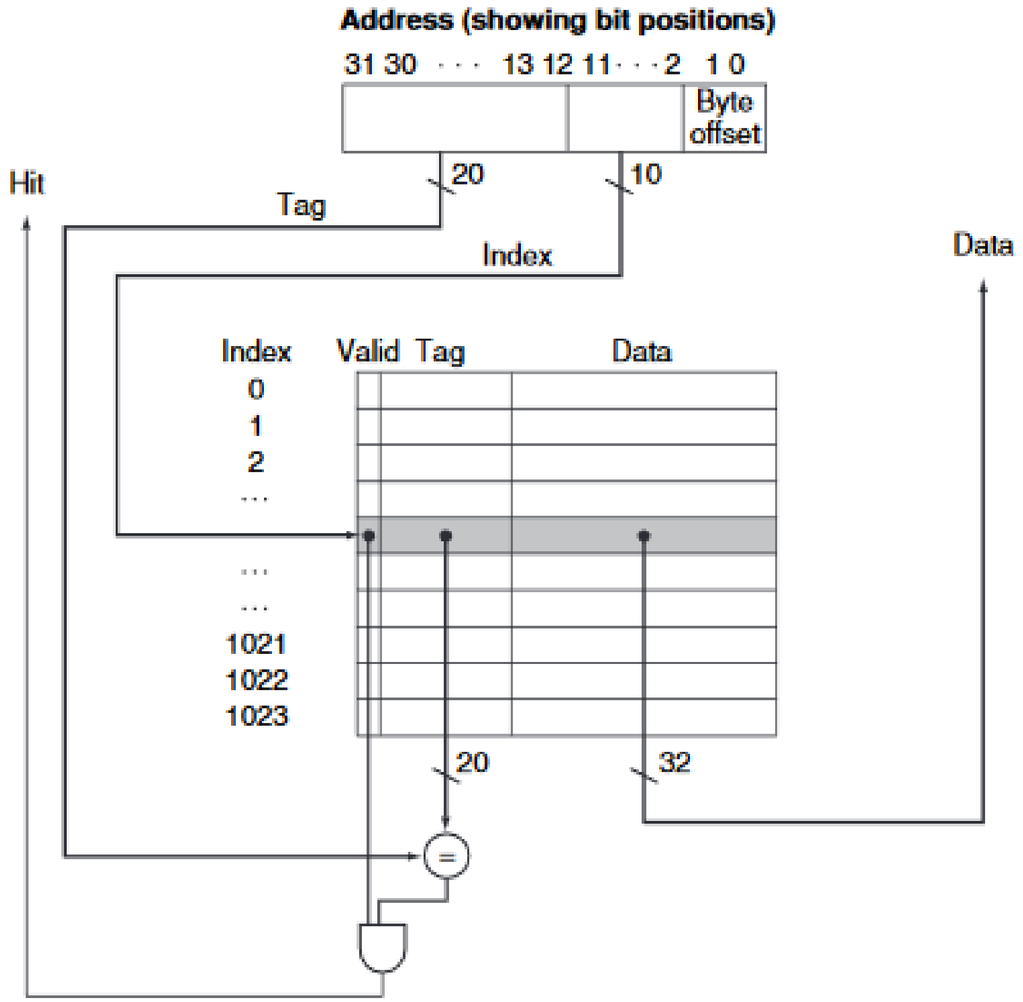
d. After handling a miss of address (10000_{two})

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	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})

Dado que la caché está vacía, varias de las primeras referencias terminan en fallas. Con respecto a la octava referencia, tenemos una colisión en la demanda de un bloque. La palabra con dirección 18 debe llevarse al bloque #2 y reemplaza la palabra con dirección 26 que ya existía en ese bloque.

Cache: Mapeo Directo



Conocemos ya la estructura de un caché y su funcionamiento básico. La siguiente figura muestra cómo se divide una dirección de referencia en:

- El campo de etiqueta.
- El índice de caché.

Cache: Mapeo Directo

El número total de bits necesarios para una caché esta en función del tamaño de los bloques y del tamaño de la dirección de memoria principal, porque la caché incluye tanto el almacenamiento de los datos como las etiquetas. Es algo que se puede calcular fácilmente.

Supongamos que tenemos la siguiente situación:

- Direcciones de 32 bits
- Un caché de mapeo directo
- El tamaño de la caché es de 2^n bloques, por lo que se utilizan n bits para el índice.
- El tamaño del bloque es de 2^m palabras (o 2^(m+2) Bytes).

Cache: Mapeo Directo

Por lo tanto el tamaño en bits de la etiqueta es de:

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

El número total de bits en una caché de mapeo directo es de:

$$2^n * (\text{block size} + \text{tag size} + \text{valid field size})$$

Dado que el tamaño del bloque es de 2^m de palabras (o 2^(m+5) bits) y necesitamos 1 bit para el campo de validación, el número de bits total es:

$$2^n * (2^m * 32 + (32 - n - m - 2) + 1) = 2^n * (2^m * 32 + 31 - n - m)$$

(Aunque este es el tamaño real en bits, la convención de nomenclatura es excluir el tamaño de la etiqueta y el campo de validación y contar sólo el tamaño de los datos. Por tanto, la caché del ejemplo anterior es una caché de 4 KiB.)

Cache

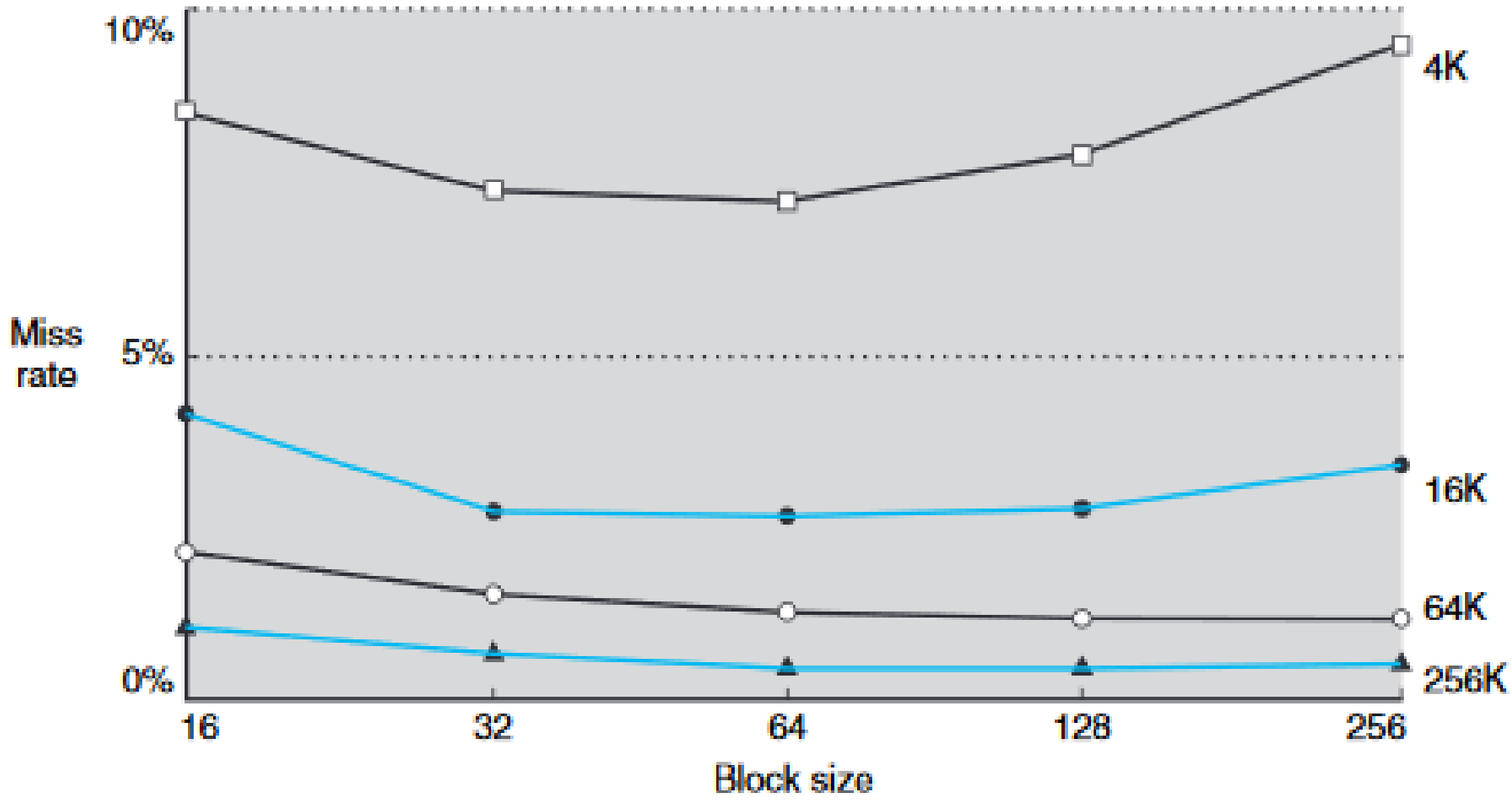
Dado que el tamaño de los bloques suele variar entre una y muchas palabras:

¿qué tamaño debe tener?

Los bloques más grandes explotan lo que se llama **localidad espacial*** para reducir las tasas de fallas. La figura en la próxima diapositiva muestra que los tamaños de bloque más grandes generalmente disminuyen la tasa de fallas. Pero la misma puede aumentar eventualmente si el tamaño del bloque se convierte en una fracción significativa en comparación con el tamaño de la caché. Esto pasa porque la cantidad de bloques que se pueden guardar se reduce y habrá una gran competencia por esos bloques.

*La localidad espacial dice que si se utiliza una palabra de una dirección en particular, es muy probable que utilices otra palabra con dirección cercana a la palabra anterior (por ejemplo, en el uso de arreglos).

Cache



Cache

Otro problema, aún más serio, con aumentar el tamaño del bloque es que aumenta el costo de un fallo. La penalización por fallo está determinada por el tiempo que se necesita para traer al bloque del siguiente nivel inferior en la jerarquía y cargarlo en la caché (por ejemplo, traerlo desde L2 a L1 o desde memoria principal a L3, L2 y L1). El tiempo para buscar el bloque tiene dos partes:

1. La latencia para traer la primera palabra.
2. El tiempo de transferencia para el resto del bloque.

Claramente, a menos que cambiemos el sistema de memoria, el tiempo de transferencia (y por lo tanto la penalización) aumentará a medida que aumente el tamaño del bloque. La consecuencia es que el aumento en la penalización por una falla supera la disminución en la tasa de fallas para bloques que son demasiado grandes y, por lo tanto, el rendimiento de la caché disminuye.

Cache

Elaboración de lo anterior: Existen ciertas técnicas para mitigar la latencia más larga producida de una penalización por fallo en bloques grandes. El método más simple se llama **reinicio temprano**, donde el procesador retoma su ejecución tan pronto que llegue la palabra solicitada que está contenida en el bloque, en lugar de esperar hasta que se complete la llegada del bloque.

Esto se usa comúnmente en el acceso a instrucciones, donde se puede tomar ventaja ya que estos accesos son en gran medida muy secuenciales. En estos casos, si el sistema de memoria puede entregar una palabra en cada ciclo de reloj, el procesador puede reiniciar su operación en el momento que llegue la palabra solicitada, y con la memoria entregando nuevas palabras (instrucciones) justo a tiempo. Esta técnica suele ser menos eficaz para la caché de datos, ya que el acceso a los datos es menos predecible. Si el procesador no puede acceder a la caché de datos porque hay una transferencia en curso, entonces debe detenerse.

Una técnica aún más sofisticada consiste en organizar la memoria de modo que la palabra solicitada se transfiera primero. Luego se transfiere el resto del bloque, comenzando con la dirección después de la palabra solicitada y continuando de forma circular hasta completar el bloque. Esto se denomina **palabra solicitada primero** o **palabra crítica primero**.

Cache: Manejando Fallas

Ahora veamos cómo la unidad de control maneja las fallas y aciertos de la caché. La unidad de control debe detectar si ocurre una falla y procesarla recuperando los datos solicitados de la memoria (o de un caché de nivel inferior). En cambio, si la caché informa un acierto, la computadora continúa usando los datos como si nada hubiera pasado.

Modificar el control de un procesador para manejar un acierto es trivial, pero las fallas sin embargo, son más complejas. Resolver una se realiza en colaboración entre la unidad de control del procesador y un controlador independiente que inicia el acceso a la memoria para rellena el caché. Durante este proceso podemos detener todo el procesador, esencialmente congelando el contenido de los registros temporales y visibles para el programador, mientras esperamos a la memoria. Hay procesadores más sofisticados, que tienen una capacidad llamada **ejecución fuera de orden**, que le permiten la ejecución de otras instrucciones mientras se espera, pero vamos a trabajar con la suposición de un procesadore en orden se detienen ante un fallo de caché.

Cache: Manejando Fallas

Veamos cómo se manejan una falla con una instrucción; el mismo enfoque se puede ampliar fácilmente para manejar una con datos.

Si el acceso a una instrucción resulta fallida, entonces el contenido del registro de instrucciones no es válido. Para obtener la instrucción adecuada en el caché, debemos poder indicarle al nivel inferior en la jerarquía de memoria que realice una lectura. Dado que el contador del programa se incrementa en el primer ciclo de reloj de ejecución, la dirección de la instrucción que generó la falla es igual al valor de la PC menos 4. Una vez que tenemos la dirección, necesitamos indicarle a la memoria principal que realizar una lectura. Esperamos que la memoria responda (ya que el acceso requerirá varios ciclos de reloj) y escribimos las palabras del bloque que contienen la instrucción deseada en la caché.

Cache: Manejando Fallas

Teniendo esto en cuenta, podemos definir los pasos a seguir ante una falla en la caché de instrucciones:

1. Envíe la dirección del PC original (PC actual – 4) a la memoria.
2. Indique a la memoria principal que realice una lectura y espere a que la memoria complete su acceso.
3. Escriba el resultado colocando las palabras en el bloque correspondiente, escribiendo los bits superiores de la dirección en el campo de etiqueta y activando el bit válido.
4. Reinicie la ejecución de la instrucción desde el primer paso, lo que se buscará la instrucción y esta vez la encontrará en la caché.

Cache: Manejando Escrituras

Las escrituras funcionan de manera considerablemente diferente. Supongamos que en una instrucción de STORE, escribimos los resultados solo en la caché de datos (sin modificar la memoria principal). Al hacer esto, la memoria principal va a tener un valor diferente para la misma palabra en caché. En este caso, se dice que la **caché y la memoria principal son inconsistentes**.

La forma más sencilla de tratar este problema y mantener la **coherencia** entre la memoria principal y la caché es hacer que siempre se escribirán los datos tanto en la memoria como en la caché. Este esquema se llama **escritura directa (write-through)**.

(Otro aspecto clave en una escritura es lo que ocurriría si hay una **falla en escritura**. Primero se necesitaría recuperar las palabras de la memoria, luego colocarlas en el caché y finalmente sobrescribir la palabra.)

Cache: Manejando Escrituras

Las escrituras funcionan de manera considerablemente diferente. Supongamos que en una instrucción de STORE, escribimos los resultados solo en la caché de datos (sin modificar la memoria principal). Al hacer esto, la memoria principal va a tener un valor diferente para la misma palabra en caché. En este caso, se dice que la **caché y la memoria principal son inconsistentes**.

La forma más sencilla de tratar este problema y mantener la **coherencia** entre la memoria principal y la caché es hacer que siempre se escribirán los datos tanto en la memoria como en la caché. Este esquema se llama **escritura directa (write-through)**.

(Otro aspecto clave en una escritura es lo que ocurriría si hay una **falla en escritura**. Primero se necesitaría recuperar las palabras de la memoria, luego colocarlas en el caché y finalmente sobrescribir la palabra.)

Cache: Manejando Escrituras

Aunque se trata de un diseño sencillo, proporcionaría un rendimiento muy malo. Con este esquema, cada escritura obliga que los datos se escriban a memoria principal. Estas escrituras tardarán mucho tiempo, posiblemente al menos 100 ciclos de reloj del procesador y, en consecuencia, podrían ralentizar considerablemente el procesador. Por ejemplo, supongamos que el 10% de nuestro programa son instrucciones de STORE. Si el CPI sin fallos de caché fuera 1,0, gastar 100 ciclos adicionales en cada escritura llevaría a un CPI de 1,0 + 100 * 10% = 11, ¡lo que reduciría el rendimiento en un factor de 10!

Cache: Manejando Escrituras

Una posible solución a este problema es implementar un búfer de escritura. Un búfer de escritura almacena los datos mientras esperan ser escritos en la memoria. Después de escribir los datos en la caché y en el búfer de escritura, el procesador puede continuar la ejecución. Cuando se completa la escritura en la memoria principal, se libera el búfer. Ahora, si el búfer de escritura está lleno cuando el procesador realiza una escritura, el procesador debe detenerse hasta que haya espacio en el búfer. Por lo tanto, si la velocidad a la que la memoria puede completar las escrituras es menor que la velocidad a la que el procesador genera escrituras, ninguna cantidad de búfer resuelve el problema.

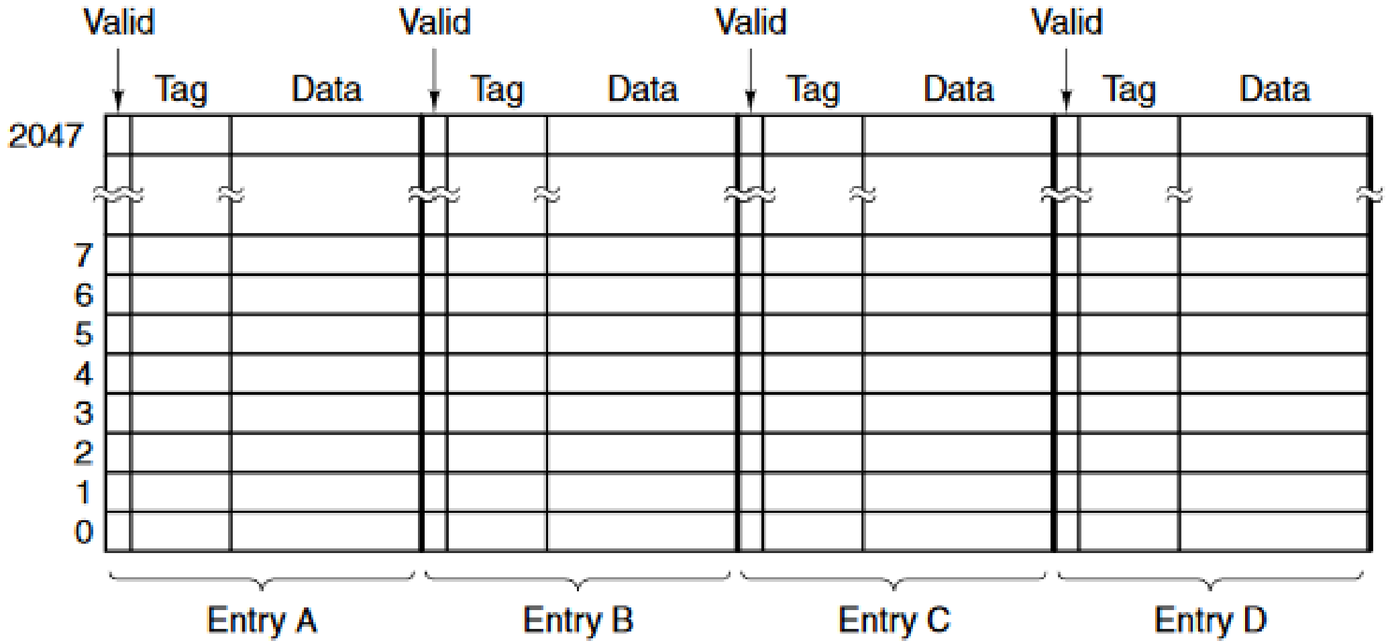
Incluso cuando la velocidad a la que se generan las escrituras es menor que la velocidad a la que la memoria puede aceptarlas, puede producirse un bloqueo. Esto puede suceder cuando las escrituras se producen en ráfagas. Para reducir la aparición de bloqueos debido a estas ráfagas, se puede aumentar la profundidad del búfer de escritura más allá de una sola palabra.

Cache: Manejando Escrituras

Un esquema alternativo a la escritura directa se llama **escritura diferida (write-back)**. En este esquema, cuando se produce una escritura, el nuevo valor se escribe sólo en el bloque de la caché. Luego, el bloque modificado se escribe a un nivel inferior de la jerarquía SOLO cuando se reemplaza. Esto mejora el rendimiento, especialmente cuando los procesadores pueden generar escrituras tan rápido como las que la memoria principal puede manejar. Sin embargo, este esquema es más complejo de implementar.

Cache: Set-Associative

Recordemos que para las cachés de mapeo directo, el procesador tiene el potencial de crear colisiones para direcciones que se asignan al mismo índice y podría potencialmente obstaculizar el rendimiento. Una solución es permitir dos o más líneas en cada índice de caché. Una caché con n entradas posibles para cada índice se denomina una caché set-associative de n vías. En la siguiente figura se puede ver un ejemplocon cuatro vías.



Cache: Set-Associative

Este tipo de caché presenta al diseñador con una opción. Cuando se va a introducir una nueva línea en la caché,

¿cuál de las líneas presentes se debe descartar?

La mejor solución es mirar hacia el futuro, lo cual no es práctico, pero un algoritmo bastante bueno para es LRU (Least Recently Used - Menos Utilizado Recientemente). Este algoritmo mantiene una lista ordenada de cada entrada. Cada vez que se accede a cualquiera de las líneas presentes, actualiza la lista, marcando esa entrada como la entrada a la que se accedió más recientemente. Cuando llega el momento de reemplazar una entrada, la que está al final de la lista (la a la que se accedió menos recientemente) es la que se descarta.