Cachés

Introducción

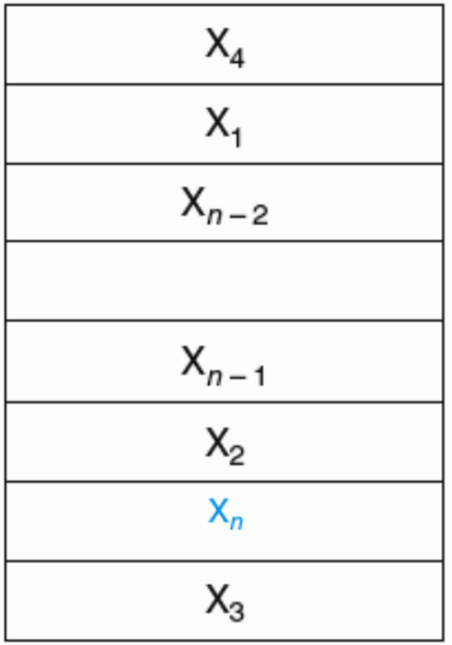
- La memoria caché es el nivel de memoria situada entre el procesador y la memoria principal.
- Se comenzaron a usar a fines de los años 60s.
- Hoy en día, todas la computadoras incluyen cachés.

Usos de las memorias cachés

- Cachés de datos. Guardan los últimos datos referenciados.
- Cachés de instrucciones. Guardan las últimas instrucciones ejecutadas.
- Cachés de trazas (trace caches). Guardan secuencias de instrucciones para ejecutar que no son necesariamente adyacentes.

Ejemplo

- El tamaño del bloque es 1 palabra (4 bytes).
- Las peticiones de la CPU son de 1 palabra.
- La memoria caché ya tiene los siguientes datos:
- Se pide X_n
- Se produce una falla.
- Se trae X_n de la memoria.
- Se inserta en el caché.



2 preguntas 2

1. ¿Cómo se sabe si un dato está en el caché?
2. Si está, ¿Cómo se encuentra?

- Estrategias:
  a) Caché de mapeo directo (direct mapped cache).
  b) Caché asociativo total (fully associative cache).
  c) Caché asociativo por conjunto (n-way set associative cache).

Caché de mapeo directo

- A cada bloque en la memoria se le asigna un bloque (línea) en el caché.
- Problema: dada una dirección de un bloque en la memoria hay que encontrar en que bloque del caché se va a guardar.
- Hay 2 soluciones (al menos):

1. Usar fórmulas.
2. Usar la dirección del bloque en la memoria.

Método 1

- Método 1: Se usan las siguientes fórmulas.
- Datos de entrada:
  - *a* – dirección del dato en la memoria.
  - *k* – tamaño de bloque en bytes.
  - *n* – número de bloques que tiene el caché.
- $d = a \ \mathbf{div} \ k$

  a es la dirección del dato en la memoria.

  k es el tamaño de bloque en bytes.

  d es la dirección de bloque.

Método 1

- $b = d$ **mod** n

  d es la dirección de bloque.

  n es el número de bloques del caché.

  b es el bloque que le corresponde en el caché.
- Conclusión: el dato que tiene dirección a en la memoria se guarda en bloque b del caché.

Método 2

- Método 2: la dirección, *a*, del dato se convierte a binario y se divide en tres partes.

1. El offset. Ocupa lo bits más bajos. Indica en que byte dentro del bloque se almacena el dato. Su tamaño es $\log_2(k)$, donde k es el tamaño del bloque en bytes.
2. El índice. Ocupa los bits intermedios. En mapeo directo indica en que bloque está guardado el dato. Su tamaño es $\log_2(n)$, donde n es el número de bloques del caché.

Método 2

3. La etiqueta. Ocupa los bits altos. Indica de que dirección en la memoria viene el dato (esto se verá a continuación). Su tamaño es lo que no se usa para el offset ni para el índice.

Ejemplo 1

- Dados los siguientes datos:
  - $k = 1$ (el caché tiene bloques de 1 byte).
  - $n = 8$ (el caché tiene 8 bloques).
- ¿Qué bloque dentro del caché le toca a un dato con dirección 57 (i.e. $a = 57$)?
- Suponer direcciones de 8 bits.
- Respuesta:

Ejemplo 1

- Método 1.
- Datos: $a = 57$, $k = 1$, $n = 8$.
  - $d = 57 \text{ div } 1 = 57$
  - $b = 57 \bmod 8 = 1$
  - Le toca el bloque 1 en el caché.

Ejemplo 1

- Método 2.
- Datos $a = 57 = 00111001$, $k = 1$, $n = 8$.
  - Tamaño del offset = $\log2(k) = \log2(1) = 0$
  - Tamaño del index = $\log2(n) = \log2(8) = 3$
  - Tamaño de la etiqueta = $8 - 3 = 5$
- Respuesta: la dirección se separa en:
  - $a = 00111\ 001$
  - Index = 001
  - Etiqueta = 00111
  - Le toca el bloque 1 en el caché.

Ejemplo 2

- Dados los siguientes datos:
  - $k = 4$ (el caché tiene bloques de 4 bytes).
  - $n = 8$ (el caché tiene 8 bloques).
- ¿Qué bloque dentro del caché le toca a un dato con dirección 57 (i.e. $a = 57$)?
- Suponer direcciones de 8 bits.

Ejemplo 2

- Método 1.
- Datos: $a = 57$, $k = 4$, $n = 8$,
- Respuesta:
  - $d = 57 \text{ div } 4 = 14$
  - $b = 14 \bmod 8 = 6$
  - Le toca el bloque 6 en el caché.

Ejemplo 2

- Método 2.
- Datos $a = 57 = 00111001$, $k = 4$, $n = 8$.
  - Tamaño del offset = $\log2(k) = \log2(4) = 2$
  - Tamaño del index = $\log2(n) = \log2(8) = 3$
  - Tamaño de la etiqueta = $8 - 3 - 2 = 3$

Ejemplo 2

- Respuesta: la dirección se separa en:
  - *a* = 001 110 01
  - Offset = 01
  - Index = 110
  - Etiqueta = 001
  - Le toca el bloque 6 en el caché.

Etiqueta y bit válido

- Se necesita saber si el dato en el caché corresponde con el dato buscado.
- El problema es que varias direcciones en la memoria pueden corresponder a un mismo bloque en el caché.
- Cada bloque en el caché tiene una *etiqueta*.
- La etiqueta tiene la información necesaria para identificar si el dato en el caché es el dato buscado.
- La etiqueta tiene los bits altos de la dirección del dato (la parte que no se usa para el offset ni para el índice).

Etiqueta y bit válido

- Además se necesita saber si el bloque tiene información válida o no.
- Cada bloque tiene un bit llamado *bit válido*.

Ejemplo

- Direcciones de 8 bits.
- Tamaño de la memoria caché: 8 bloques.
- Tamaño del bloque: 1 byte.
- Offset = $\log_2(1) = 0$ bits.
- Index = $\log_2(8) = 3$ bits.
- Etiqueta = $8 - 3 - 0 = 5$ bits.
- Inicialmente la memoria caché está vacía (para todas las entradas bit válido = falso).

Ejemplo

- Se reciben las siguientes peticiones de direcciones: 22, 26, 22, 26, 16, 3, 16 y 18

Estado inicial

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

Se pide la dirección 22_{10}

- $22_{10} = 10110_2$.
- Se busca en el bloque 110 y se produce una falla.
- Se carga el dato en el bloque 110. Tasa de éxito: 0/1.

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

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		

Se pide la dirección 26_{10}

- $26_{10} = 11010_2$.
- Se busca en el bloque 010 y se produce una falla.
- Se carga el dato en el bloque 010. Tasa de éxito 0/2.

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		

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		

Se pide la dirección 22_{10}

- $22_{10} = 10110_2$.
- Se busca en el bloque 110.
- Se compara la etiqueta con 10 (la parte alta de la dirección).
- Se produce un éxito. Tasa de éxito: 1/3.

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		

Se pide la dirección 26_{10}

- $26_{10} = 11010_2$.
- Se busca en el bloque 010.
- Se compara la etiqueta con 11 (la parte alta de la dirección).
- Se produce un éxito. Tasa de éxito: 2/4.

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		

Se pide la dirección 16_{10}

- $16_{10} = 10000_2$.
- Se busca en el bloque 000 y se produce una falla.
- Se carga el dato en el bloque 000. Tasa de éxito 2/5.

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		

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		

Se pide la dirección 3_{10}

- $3_{10} = 00011_2$.
- Se busca en el bloque 011 y se produce una falla.
- Se carga el dato en el bloque 011. Tasa de éxito 2/6.

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		

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		

Se pide la dirección 16_{10}

- $16_{10} = 10000_2$.
- Se busca en el bloque 000.
- Se compara la etiqueta con 10 (la parte alta de la dirección).
- Se produce un éxito. Tasa de éxito: 3/7.

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		

Se pide la dirección 18_{10}

- $18_{10} = 10010_2$.
- Se busca en el bloque 010 y se produce una falla.
- Se carga el dato en el bloque 010. Tasa de éxito 3/8.

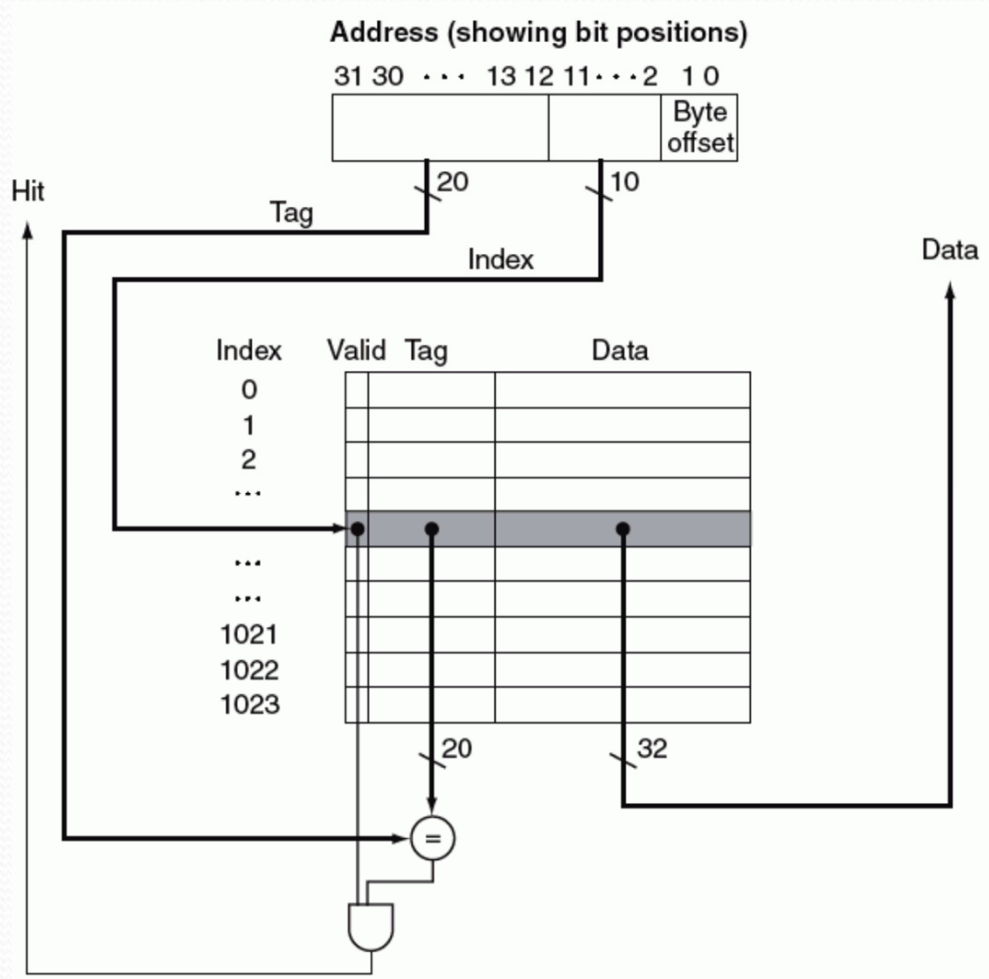
Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

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		

Nomenclatura

- El nombre del caché indica el espacio disponible para guardar datos.
- No se toma en cuenta el espacio extra dedicado a las etiquetas y a los bits válidos.
- Un caché de 8KB tiene 8 kilobytes para guardar datos.
- Un cache de 64KB tiene 64 kilobytes para guardar datos.

Caché de 4KB de mapeo directo



Explicación

- 1K = 1024 = 2^{10} palabras.
- Direcciones de 32 bits.
- La dirección se divide en:
  - Índice del cache (bits 11:2) selecciona el bloque.
  - Etiqueta (bits 31:12) se compara con la etiqueta del bloque del caché.
- Éxito = válido AND (etiqueta == dirección[31:12])

Overhead de un caché

- Overhead es el espacio extra requerido para guardar los datos en un caché.
- En un caché de mapeo directo, el overhead incluye las etiquetas y los bits válidos.
- Hay al menos dos métodos para calcular el overhead de un caché:

Overhead de un caché

- Método 1:
- Calcular el número de bits que ocupan los datos en el caché y llamarle a este número *a*.
- Calcular el número de bits que ocupa todo el caché (incluyendo etiquetas y bits válidos) y llamarle a este número *b*.
- Overhead = ((*b* / *a*) – 1) x 100.

Overhead de un caché

- Método 2:
- Obtener el tamaño de bloque en bits y llamarle a este número *a*.
- Calcular *b* = tamaño de etiqueta en bits + 1 (por el bit válido).
- Overhead = (*b* / *a*) x 100.

Ejemplo

- ¿Cuál es el overhead de un caché de 16 KB de datos en bloques de 4 palabras? Las direcciones son de 32 bits.

Ejemplo método 1

- Calcular el número de bits que ocupan los datos en el caché y llamarle a este número *a*.
- El caché es de 16KB.
- *a* = 16 x 1024 x 8 = 131072 bits.
- Calcular el número de bits que ocupa todo el caché (incluyendo etiquetas y bits válidos) y llamarle a este número *b*.
- *b* = tamaño total del bloque en bits (datos + etiqueta + bit válido) x número de bloques.

Ejemplo método 1

- Tamaño del bloque de datos = 4 palabras.
- Tamaño del bloque de datos en bits = 4 x 4 x 8 = 128 bits.
- Tamaño de etiqueta = Tamaño de la dirección – (tamaño del index + tamaño del offset).
- Tamaño del index = $\log_2$ (número de bloques).
- Número de bloques = tamaño del cache / tamaño del bloque de datos.

Ejemplo método 1

- Número de bloques = (16 x 1024) / (4 x 4) = 1024 bloques.
- Tamaño del index = $\log_2$ (1024) = 10 bits.
- Tamaño del offset = $\log_2$ (tamaño de bloque en bytes).
- Tamaño del offset = $\log_2$ (4 x 4) = 4 bits.
- Tamaño de la etiqueta = 32 – (10 + 4) = 18 bits.
- Tamaño total del bloque = 128 + 18 + 1 = 147 bits.

Ejemplo método 1

- El número de bits que ocupa todo el caché (incluyendo etiquetas y bits válidos) es:
- b = 147 x 1024 = 150528 bits.
- Son 1024 bloques y cada bloque ocupa en total 147 bits.
- Overhead = (150528 / 131072) – 1 x 100.
- Overhead = 14.84
- Conclusión: el caché ocupa un espacio extra de casi 15% dedicado a las etiquetas y bits válidos.

Ejemplo método 2

- Calcular el tamaño de bloque en bits y llamarle a este número *a*.
- Tamaño de bloque es de 4 palabras.
- *a* = 4 x 4 x 8 = 128 bits.

Ejemplo método 2

- Calcular *b* = tamaño de etiqueta en bits + 1 (por el bit válido).
- Tamaño de etiqueta = Tamaño de la dirección – (tamaño del index + tamaño del offset).
- Tamaño del index = $\log_2$ (número de bloques).
- Número de bloques = tamaño del cache / tamaño del bloque de datos.

Ejemplo método 2

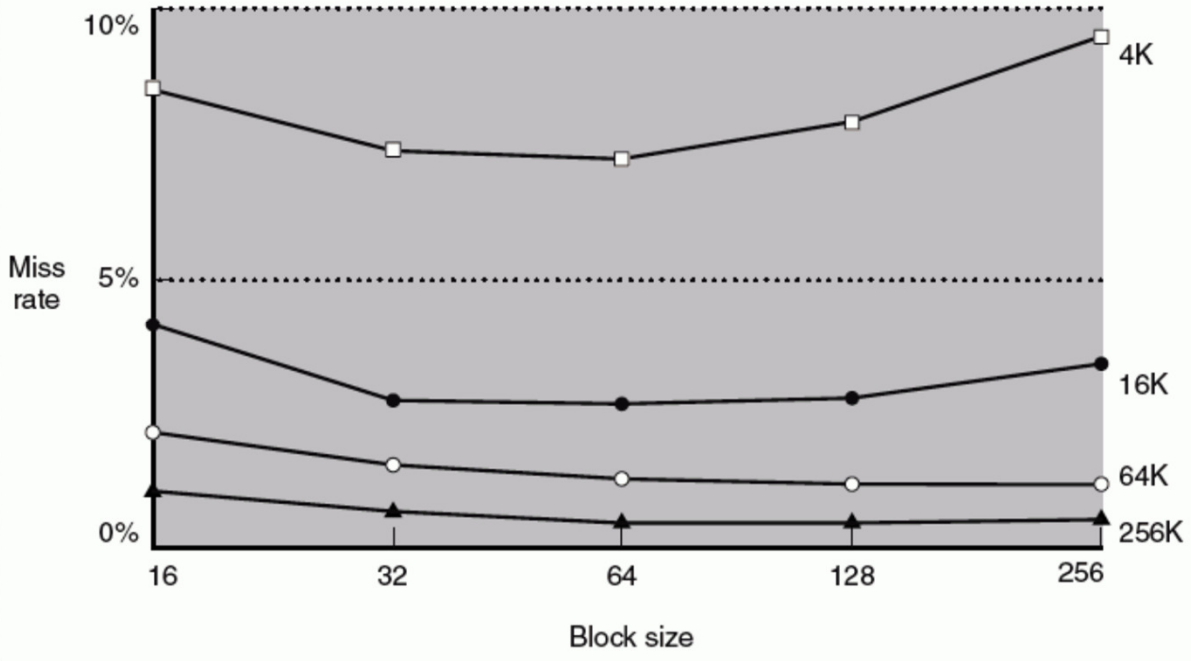
- Número de bloques = (16 x 1024) / (4 x 4) = 1024 bloques.
- Tamaño del index = $\log_2$ (1024) = 10 bits.
- Tamaño del offset = $\log_2$ (tamaño de bloque en bytes).
- Tamaño del offset = $\log_2$ (4 x 4) = 4 bits.
- Tamaño de la etiqueta = 32 – (10 + 4) = 18 bits.
- *b* = 18 + 1 = 19 bits.

Ejemplo método 2

- Overhead = (*b* / *a*) x 100.
- Overhead = (19 / 128) x 100 = 14.84.
- El overhead es de casi el 15%.

Impacto del tamaño del bloque

- Incrementar el tamaño de bloque generalmente reduce la tasa de fallas.
- La tasa de fallas aumenta si el bloque es muy grande comparado con el tamaño del cache.



Impacto del tamaño del bloque

- Incrementar el tamaño de bloque aumenta el castigo por falla.
- El castigo por falla es el tiempo para obtener el bloque del siguiente nivel y cargarlo en el caché.
- El tiempo para obtener el bloque consta de:
  - La latencia por la primera palabra.
  - Tiempo de transferencia por el resto del bloque.
- Incrementar el tamaño de bloque aumenta el tiempo de transferencia y por lo tanto, el castigo por falla (a menos que se rediseñe la transferencia de bloques).

Manejo de escrituras

- Una escritura a la memoria es el resultado de una instrucción ***sw***.
- Hay dos escenarios:
  - Escritura con éxito. El dato está en el caché (y en la memoria).
  - Escritura con falla. El dato está en la memoria y no en el caché.

Manejo de escrituras

- Si el dato está en el caché (escritura con éxito), surge la pregunta si hay que actualizar el dato en el caché y en la memoria (de otro modo el caché y la memoria serían *inconsistentes*.)
- Hay dos políticas de escritura con éxito: write-through y write-back.

Manejo de escrituras

- Si el dato no está en el caché (escritura con falla), surge la pregunta si después de actualizar el dato en la memoria hay que subirlo o no al caché.
- Hay dos políticas de escritura con falla: write allocate y no write allocate.

Write-through

- Escribir cada vez en el caché y en la memoria.
- Escribir en la memoria es lento.
- Una escritura puede tomar 100 ciclos de reloj.
- Según SPECInt2000, 10% de las instrucciones son escrituras.
- Si el CPI es 1, gastar 100 ciclos en cada escritura aumenta el CPI a 1 + 100 * 0.1 = 11.
- Escribir en el caché y la memoria puede reducir el rendimiento en un factor de 10.

Write-through

- Una solución es usar un buffer de escritura.
- El buffer de escritura es una cola que guarda datos que esperan ser escritos en la memoria principal.
- El programa escribe en el caché y en el buffer y continúa ejecutando instrucciones.
- Si el buffer está lleno, el siguiente store se detiene.
- Un procesador superescalar puede continuar ejecutando alguna otra instrucción.

Write-through

- Ningún buffer es suficiente si el procesador produce escrituras mas rápido de lo que la memoria puede aceptarlas.

Write-back

- Escribir solo en el caché y copiar el dato a la memoria cuando la entrada en el caché va a ser reemplazada.
- El bloque del caché se escribe en la memoria solo si es necesario.
- El bloque tiene un bit llamado *bit sucio*.
- Si el bit sucio está apagado, el bloque no fue modificado y puede ser reemplazado sin peligro.

Write-back

- Si el bit sucio está prendido, hay una inconsistencia entre el caché y la memoria. Antes de ser reemplazado, el bloque debe escribirse en la memoria.

Comparación – write-through

- Ventajas:
- Las fallas de lectura nunca producen escrituras a la memoria principal.
- Fácil de implementar.
- La memoria principal siempre está actualizada (consistente).
- Desventajas:
- La escritura es más lenta.
- Cada escritura necesita acceso a la memoria principal.
- Por lo tanto, requiere más ancho de banda.

Comparación – write-back

- Ventajas:
- Las escrituras ocurren a la velocidad del caché.
- Múltiples escrituras dentro de un bloque requieren una sola escritura en la memoria principal.
- Por lo tanto, requiere menos ancho de banda.
- Desventajas:
- Más difícil de implementar.
- La memoria principal no siempre es consistente con el caché.

Comparación – write-back

- Las lecturas que resultan en un reemplazo pueden resultar en escrituras de bloques sucios a la memoria principal.

Conclusión

- No hay un claro ganador entre write-through y write-back.

Políticas de escritura con falla

- Write allocate: el bloque se carga en el caché y luego se aplica la política de escritura con éxito (write-through o write-back).
- No write allocate: el bloque se escribe (actualiza) en la memoria y **no** se carga en el caché.

Combinaciones

- En teoría, cualquier política de escritura con éxito (write-back o write-through) se puede combinar con cualquier política de escritura con falla (write allocate o no write allocate).
- En la práctica, lo más común es usar una política de las siguientes dos combinaciones:
- Write-through con no write allocate.
- Write-back con write allocate.

Write-through con no write allocate

- Si hay éxito, escribe en el caché y en la memoria principal.
- Si hay falla, actualiza el bloque en la memoria sin cargar el bloque en el caché.
- Las siguientes escrituras al mismo bloque actualizan el bloque en la memoria.
- Se ahorra tiempo al no cargar el bloque en el caché al producirse una falla.

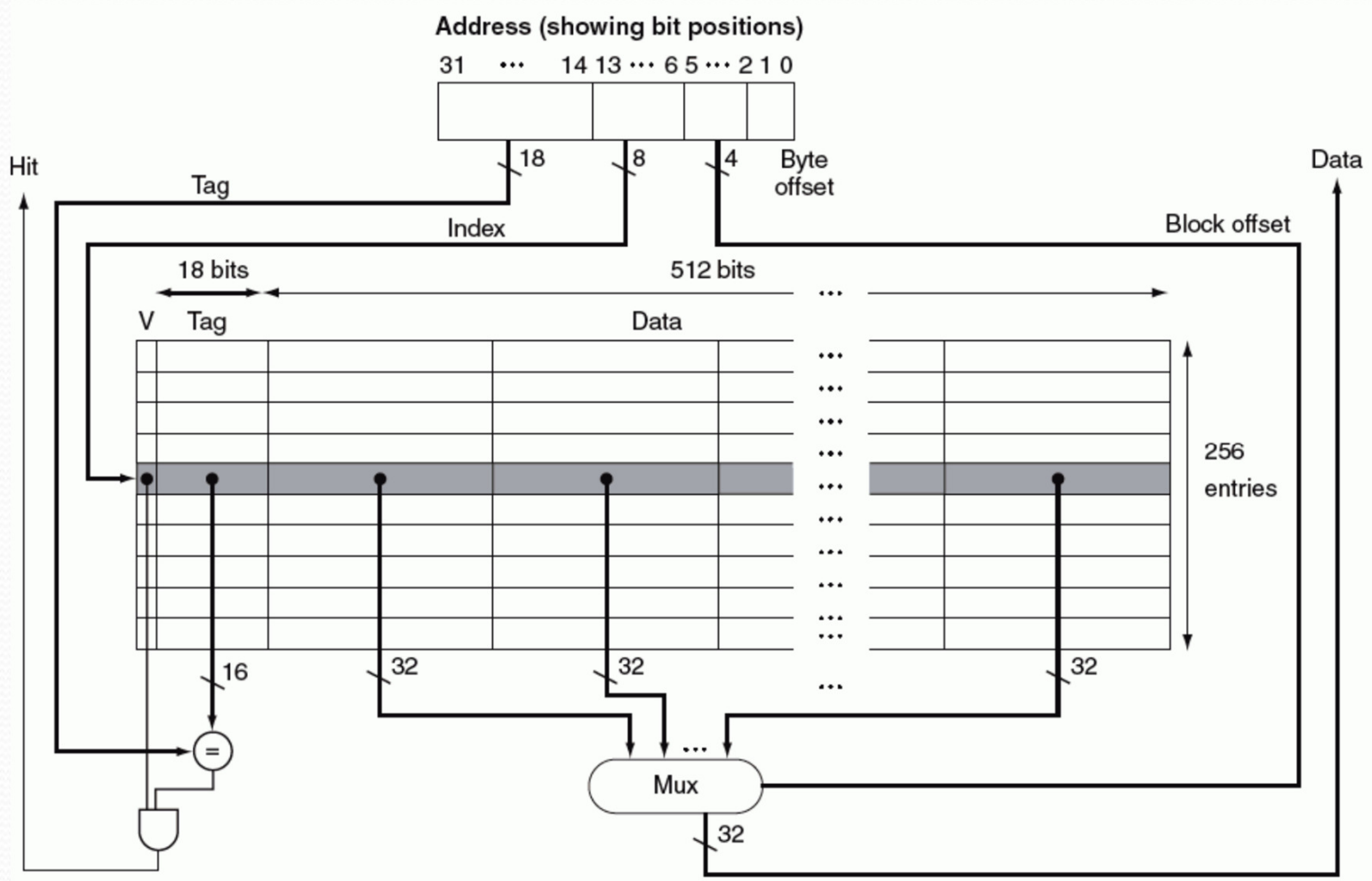
Write-back con write allocate

- Si hay éxito, escribe en el caché y prende el bit sucio del bloque. La memoria no se actualiza.
- Si hay falla, actualiza el bloque en el memoria y lo carga en el caché.
- Las siguientes escrituras al mismo bloque van a producir éxitos y se ahorran accesos a la memoria.

FastMATH

- Intrinsity FastMATH es un procesador con un pipeline de 12 etapas y arquitectura MIPS.
- FastMATH tiene un caché dividido (split cache): dos cachés independientes que operan en paralelo, uno para instrucciones y otro para datos.
- La capacidad del caché es de 32 KB en total (16 KB por cada caché).
- Cada caché tiene 256 bloques con 16 palabras (64 bytes) por bloque.

FastMATH


Address (showing bit positions)
31 ••• 14 13 ••• 6 5 ••• 2 1 0
18
8
4
Byte offset
Hit
Tag
Index
Block offset
Data
18 bits
512 bits
V
Tag
Data
256 entries
16
32
32
32
=
Mux
32

FastMATH (Explicación)

- La palabra correcta se selecciona con un MUX controlado por los bits 2 a 5 de la dirección.
- Para escribir, FastMATH ofrece write-through y write-back, dejando al sistema operativo decidir que estrategia usar para una aplicación.
- Además, tiene un buffer de escritura de una entrada.

FastMATH

- Tasas de falla en SPEC2000.

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

- Por lo general, un caché unificado tiene tasa de falla ligeramente menor que un caché dividido: 3.18% vs. 3.24%.
- Un caché dividido tiene mas *ancho de banda*, puede accesar instrucciones y datos a la vez.

Organización de la memoria

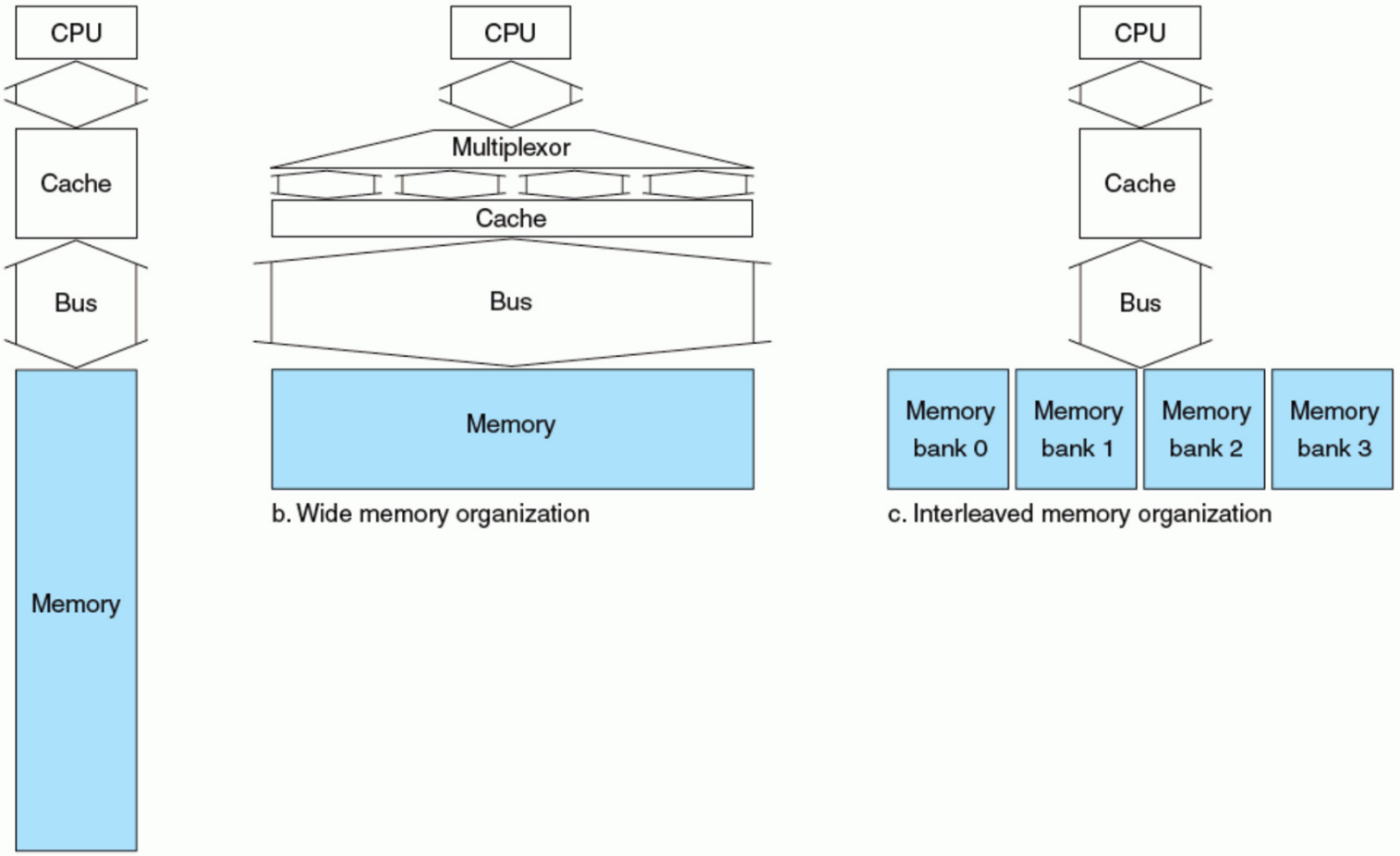
- La memoria se puede organizar para reducir el castigo por falla.
- Hay 3 formas de organizar la memoria:

1. Memoria delgada: el tamaño de bloque del caché, el ancho del bus que conecta el caché con la memoria y el ancho de la memoria es típicamente de 1 palabra.
2. Memoria ancha: el tamaño de bloque del caché, el ancho del bus que conecta el caché con la memoria y el ancho de la memoria es típicamente de 4 palabras o más.

Organización de la memoria

3. Memoria interleaved: el tamaño de bloque del caché y el ancho del bus que conecta el caché con la memoria es típicamente de 1 palabras. La memoria está organizada en bancos que se pueden leer en paralelo.

Organización de la memoria



Ejemplo

- Suponer los siguientes tiempos:
  - 1 ciclo de reloj del bus para enviar la dirección.
  - 15 ciclos por cada acceso a memoria.
  - 1 ciclo para enviar una palabra.
- Suponer un tamaño de bloque de 4 palabras.

Memoria delgada (1 palabra)

- Ancho del bus y de la memoria es de 1 palabra.
- El acceso a la memoria es secuencial.
- Castigo por falla: 1 + 4 x 15 + 4 x 1 = 65 ciclos de reloj del bus.
- Número de bytes transferidos por ciclo de reloj del bus por cada falla: (4 x 4) / 65 = 0.25

Memoria ancha (4 palabras)

- Ancho del bus y de la memoria es de 4 palabras.
- El acceso a la memoria es en paralelo.
- Castigo por falla: 1 + 1 x 15 + 1 x 1 = 17 ciclos de reloj del bus.
- Número de bytes transferidos por ciclo de reloj del bus por cada falla: (4 x 4) / 17 = 0.94

Memoria interleaved

- Ancho del bus es de 1 palabra.
- Ancho de la memoria es de 4 palabras repartidas en 4 bancos de memoria independientes.
- El acceso a la memoria es en paralelo.
- La transferencia es secuencial.
- Castigo por falla: 1 + 1 x 15 + 1 x 4 = 20 ciclos de reloj del bus.
- Número de bytes transferidos por ciclo de reloj del bus por cada falla: (4 x 4) / 20 = 0.8

Organización de la memoria

- La organización interleaved es la preferida:
  - El castigo por falla es aceptable.
  - Un bus y memoria de 4 palabras es mas caro que un bus y 4 bancos de 1 palabra cada uno.
  - Cada banco puede escribir en forma independiente, cuadruplicando el ancho de banda al escribir y provocando menos detenciones (stalls) en un caché write-through.

Resumen

- Caché de mapeo directo.
- Una palabra puede ir en un solo bloque y hay una etiqueta para cada bloque.
- Estrategias para mantener el caché y la memoria consistentes: write-through y write-back.
- Para tomar ventaja de la locality espacial el bloque del caché debe ser mayor a un byte.
- Bloques grandes reducen la tasa de fallas y mejora la eficiencia al requerir menos espacio para las etiquetas.

Resumen

- Bloques grandes aumentan el castigo por falla.
- El castigo por falla se puede reducir incrementando el ancho de banda de la memoria para transferir bloques de forma eficiente.
- Dos métodos comunes para aumentar el ancho de banda son memoria ancha y memoria interleaved.
- Interleaving tiene ventajas adicionales.