Arquitectura de Computadoras para Ingeniería

(Cód. 7526)
1° Cuatrimestre 2016

Dra. Dana K. Urribarri
DCIC - UNS

Memoria cache (continuación)

Memoria caché

- 4 preguntas para diseñar una caché:
  1) ¿Cuándo llevar el contenido de MP a caché?
  2) ¿Dónde ponerlo?
  3) ¿Cómo sabemos si está en caché?
  4) ¿Qué pasa si hay que llevar nuevo contenido a caché y ésta está llena (o el lugar donde iría el contenido está ocupado)?

Memoria caché

- En una organización básica de caché:
  1) En principio, el contenido se lleva a la caché bajo demanda (cuando hay un faltante)
  2) La caché se divide en entradas. La organización de la caché determina el mapeo de posiciones de memoria a entradas en la caché.
  3) Cada entrada de la caché contiene el nombre (o tag) y el contenido.

  4) Cuando hay un faltante en la caché y la nueva entrada colisiona con alguna existente, un algoritmo de reemplazo resuelve qué entrada reemplazar.

Algoritmos de reemplazo

¿Qué pasa si hay que llevar nuevo contenido a caché y ésta está llena (o el lugar donde iría el contenido está ocupado)?

- *Direct-mapped* caché:
  - A cada dirección de memoria le corresponde una única línea de caché.
  - Si hay que cargar un nuevo bloque y la línea que le corresponde está ocupada, se reemplaza con la línea nueva.
- *Set associative* y *fully associative* cachés:
  - A cada bloque de memoria le corresponde cualquier línea de caché dentro de un conjunto.
  - Si en el conjunto no hay ninguna línea libre, hay que elegir un bloque para reemplazar.
    - Random
    - Elegir inteligentemente

Por la localidad temporal, la mejor opción sería elegir la línea que hace más tiempo que no se usa.

Least recently used (LRU)

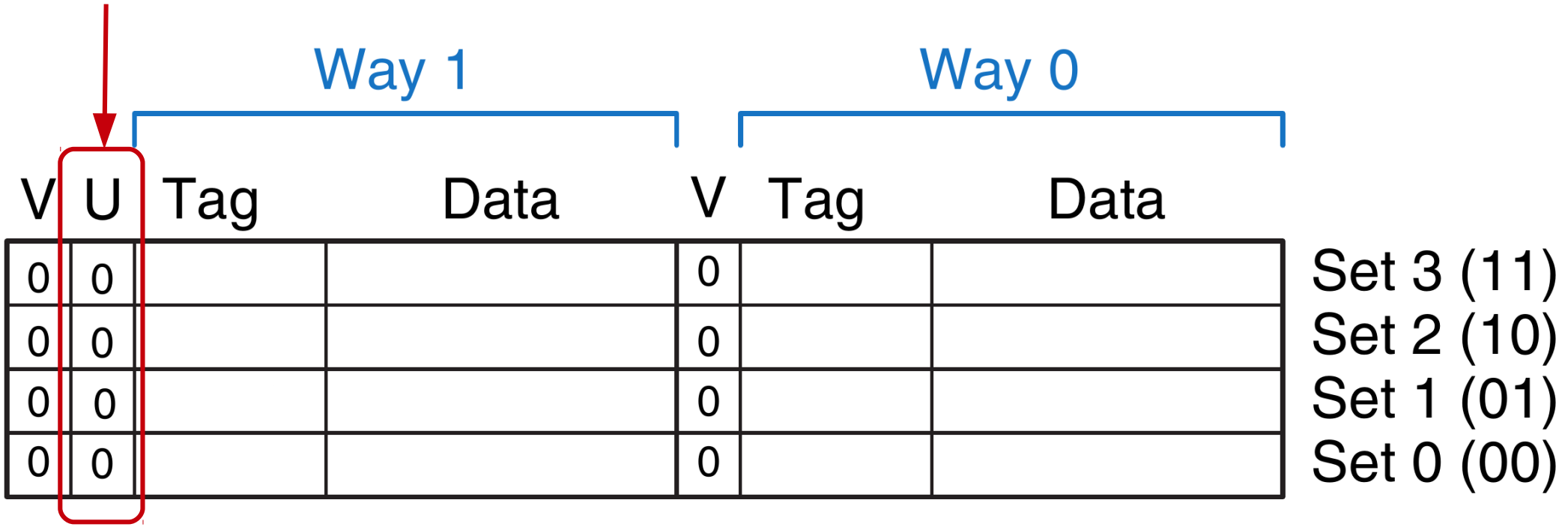
- Se lleva el control de cuál es la línea del conjunto que hace más tiempo que no se usa.
- Es simple de implementar en cachés con poco grado de asociatividad ($m \leq 4$).
- Para grados mayores de asociatividad es más costoso.
- Se utilizan esquemas alternativos.

Ejemplo LRU

2-way associative cache

Indica cuál de los dos conjuntos es que menos recientemente se usó.

```
ld $t0, 0x04($0)
ld $t1, 0x24($0)
ld $t2, 0x54($0)
```



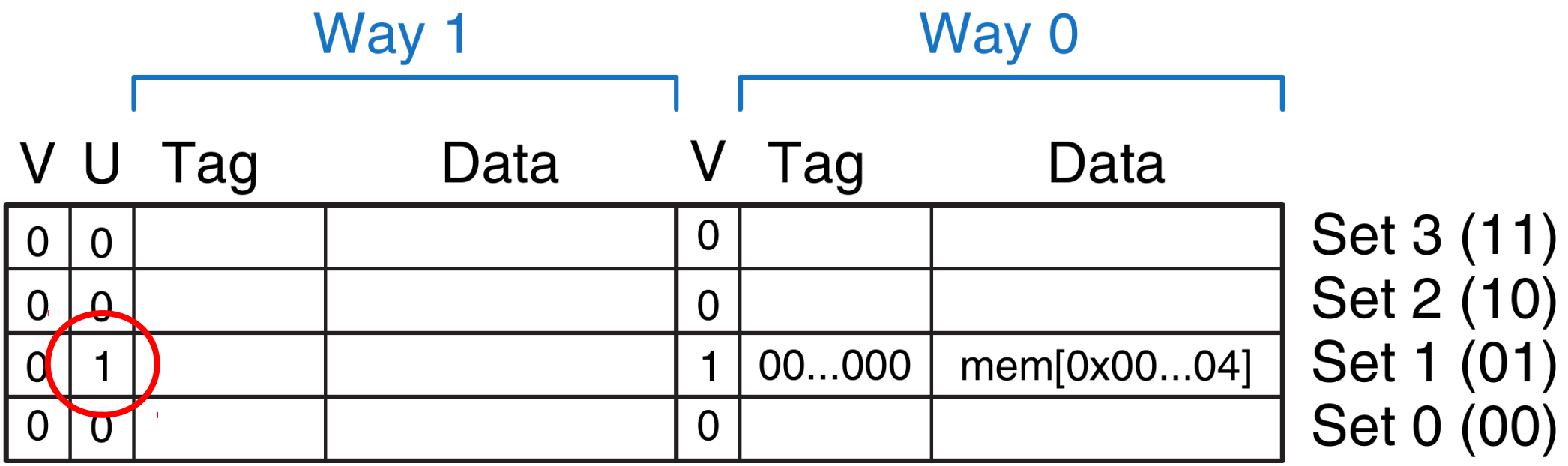
Ejemplo LRU

2-way associative cache

ld $t0, 0x04($0)

ld $t1, 0x24($0)

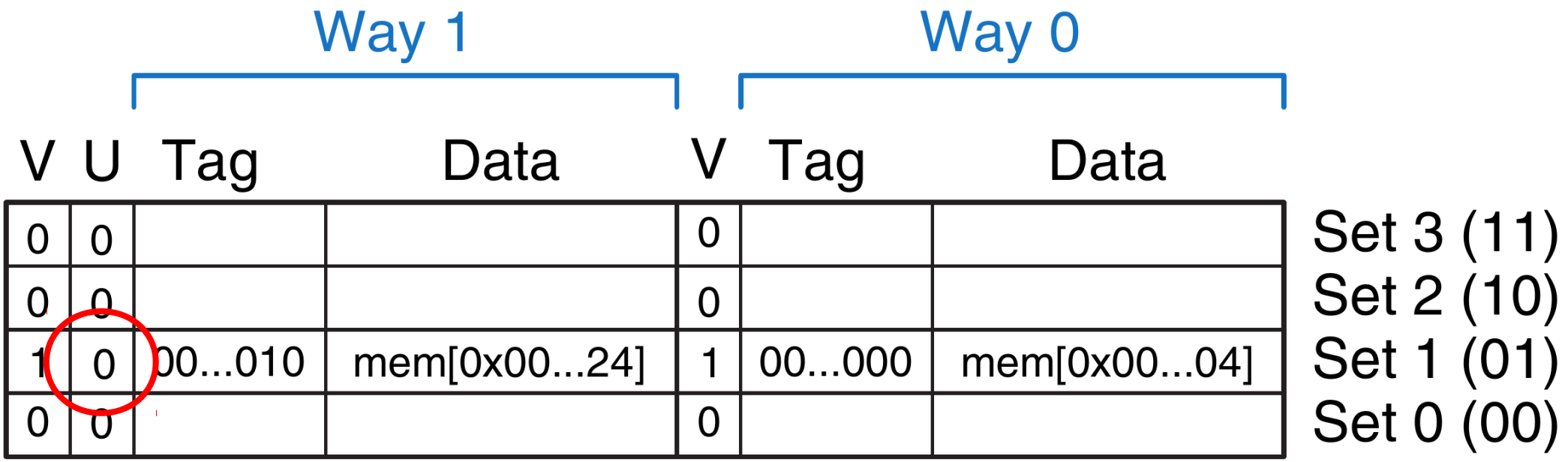
ld $t2, 0x54($0)



Ejemplo LRU

2-way associative cache

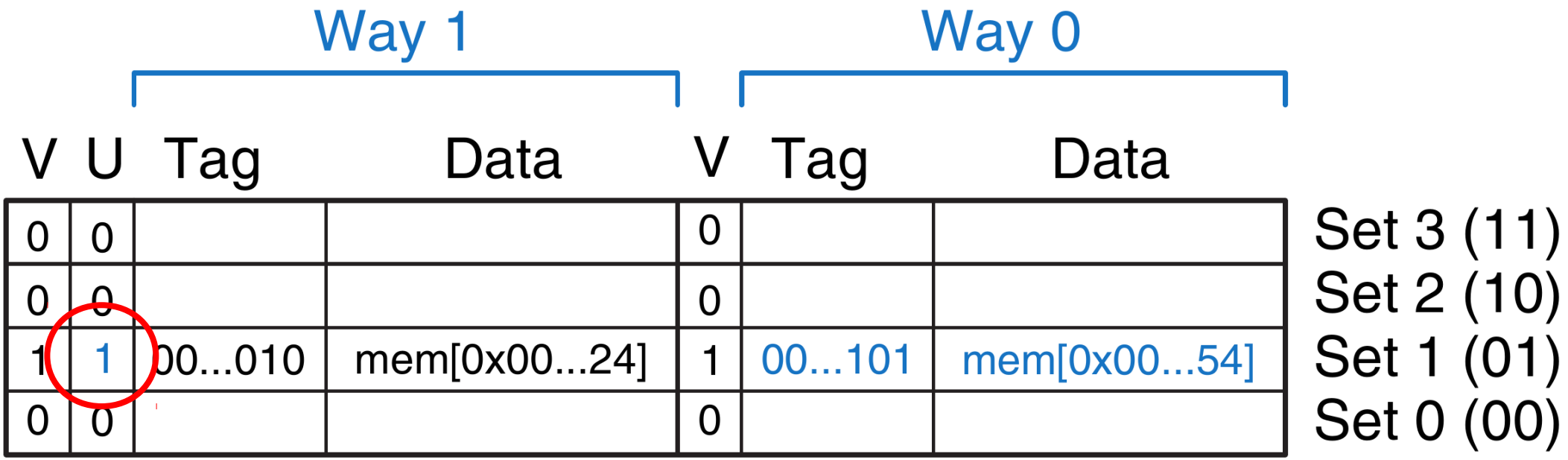
```
ld $t0, 0x04($0)
ld $t1, 0x24($0)
ld $t2, 0x54($0)
```



Ejemplo LRU

2-way associative cache

ld $t0, 0x04($0)
ld $t1, 0x24($0)
ld $t2, 0x54($0)



Pseudo-LRU

- Llevar la cuenta de cuál es la línea menos usada en cachés con grados de asociatividad altos es costoso.
- El problema se simplifica dividiendo los conjuntos en 2 grupos.
- Un bit indica cuál de los dos grupos es el menos recientemente usado.
  - Es decir, no contiene a la última línea accedida.
- Dentro de ese grupo se reemplaza uno al azar.

Estrategias de escritura

¿Qué ocurre con un store (escritura en memoria)?

Pueden ocurrir dos cosas

1) La dirección de memoria referenciada está en la caché
   1) *Writeback* caché
   2) *Write-through* caché

Hay que modificar la caché (las futuras referencias a esa dirección deben retornar el último valor).

2) La dirección de memoria referenciada no está en la caché
   1) *Write-allocate*
   2) *Write-around*

Políticas de escritura

Writeback caché

- Se escribe sólo en la caché.
- La información en la caché ya no es la misma que la del siguiente nivel (*de alguna forma hay que indicarlo*).
- A cada línea de caché se le asocia un *dirty bit*
  - 1 indica que la línea fue modificada, 0 que no.
- Al reemplazar una línea, se controla el *dirty bit*
  - Si es 0, la línea puede descartarse.
  - Si es 1, la línea se copia en el siguiente nivel de la jerarquía (memoria principal)

Políticas de escritura

Write-through caché

- Además de modificar la caché, se escribe en el siguiente nivel de la jerarquía.

✔Mantiene la consistencia entre ambos niveles.

✔No hay necesidad del *dirty bit*.

✗ Hay mayor tráfico entre memoria y caché en comparación con *writeback.*

Writeback vs. Write-through

- Ventajas de *writeback*
  - El procesador escribe las palabras individuales a la velocidad de la caché.
  - Múltiples escrituras en el mismo bloque requieren una única escritura en el nivel inferior.
  - Cuando se escribe el bloque completo el sistema puede realizar eficientemente la transferencia.
- Ventajas de *write-through*
  - La resolución de los faltantes de caché es más simples y económica.
  - Es más simple de implementar.

Políticas de escritura

Write-allocate

Tratarlo como si fuera primero una lectura y luego la escritura. Es decir, se soluciona el faltante en caché por la lectura y luego la dirección a escribir va a estar en la caché.

Write-around

Se escribe solo en el siguiente nivel de la jerarquía, salteando la caché.

Write buffer

Write-through caché

- En una escritura el procesador debe esperar a que se escriba el dato en memoria.
  Esa espera puede ser del orden de los 100 ciclos.
- El *write buffer* almacena temporariamente el dato a escribir y la dirección destino.
  - Una vez que el dato está en el buffer, el procesador puede continuar ejecutando.
  - Cuando se completa la escritura en memoria, se libera la entrada del buffer.
- Si el write buffer está lleno, el procesador debe frenarse (stall) hasta que se libere un lugar.

Write buffer

Writeback caché

- Cuando hay que reemplazar una línea con dirty bit en 1
  - No se escribe directamente en memoria.
  - Se escribe la línea y la dirección en un *write buffer.*
- Cuando hay un faltante de caché, debe revisarse primero el write buffer.
- Se puede transformar en una pequeña caché *fully associative*.
  - La información a escribir puede sobrescribir (o fusionarse con) la línea que exista en el buffer.

Clasificación de los faltantes

- Compulsivo:
  - Ocurre la primera vez que un bloque es referenciado.
- Conflictivo:
  - Ocurre cuando más de *m* bloques compiten por la misma entrada de caché en una caché *m*-way set associative.
  - Se eliminan en una caché fully associative del mismo tamaño.
- Capacitivo:
  - Ocurre cuando la caché no puede contener todos los bloques que necesita el programa para ejecutar.
  - Hay que volver a llevar a caché un bloque que fue reemplazado.

Caché multinivel

- Para mejorar el rendimiento de la caché
  - Disminuir la penalización de los faltantes
  - Disminuir la tasa de fallos.
- Velocidad de procesador >> velocidad de MP
- Capacidad de procesador << Capacidad de MP
- El objetivo es lograr una caché rápida y grande.
- Varios niveles
  1) Un primer nivel rápido con poca capacidad
  2) Un segundo nivel de mayor capacidad

Caché multinivel

- Tasa de fallos local:
  - Es el número de faltantes en la caché dividido la cantidad de accesos a esa caché.
- Tasa de fallos global:
  - El número de faltantes en la caché dividido el total de accesos a memoria generados por el procesador.

Caché multinivel

Tasa de fallos local:

- Es el número de faltantes en la caché dividido la cantidad de accesos a esa caché.
  - *Miss rate* L_i
    = cantidad de fallos en L_i / cantidad de accesos a L_i
  - *Miss rate* L_1
    = cantidad de fallos en L_1 / cantidad de accesos a L_1
    = cantidad de fallos en L_1 / cantidad total de accesos
  - *Miss rate* L_2
    = cantidad de fallos en L_2 / cantidad de accesos a L_2
    = cantidad de fallos en L_2 / cantidad de fallos en L_1

Caché multinivel

Tasa de fallos global:

- El número de faltantes en la caché dividido el total de accesos a memoria generados por el procesador.
  - *Miss rate global* L_i
    = cantidad de fallos en L_i / cantidad total de accesos
  - *Miss rate global* L_1
    = cantidad de fallos en L_1 / cantidad total de accesos
  - *Miss rate global* L_2
    = cantidad de fallos en L_2 / cantidad total de accesos
    = *Miss rate* L_1 * *Miss rate* L_2

Caché multinivel

- La velocidad del primer nivel de caché afecta la velocidad del procesador
- La velocidad del segundo nivel de caché solo afecta el *miss penalty* del primer nivel.

- Diseño del segundo nivel de caché
  - Tamaño
  - Grado de asociatividad
  - Inclusión/Exclusión

Diseño del segundo nivel de caché

Tamaño

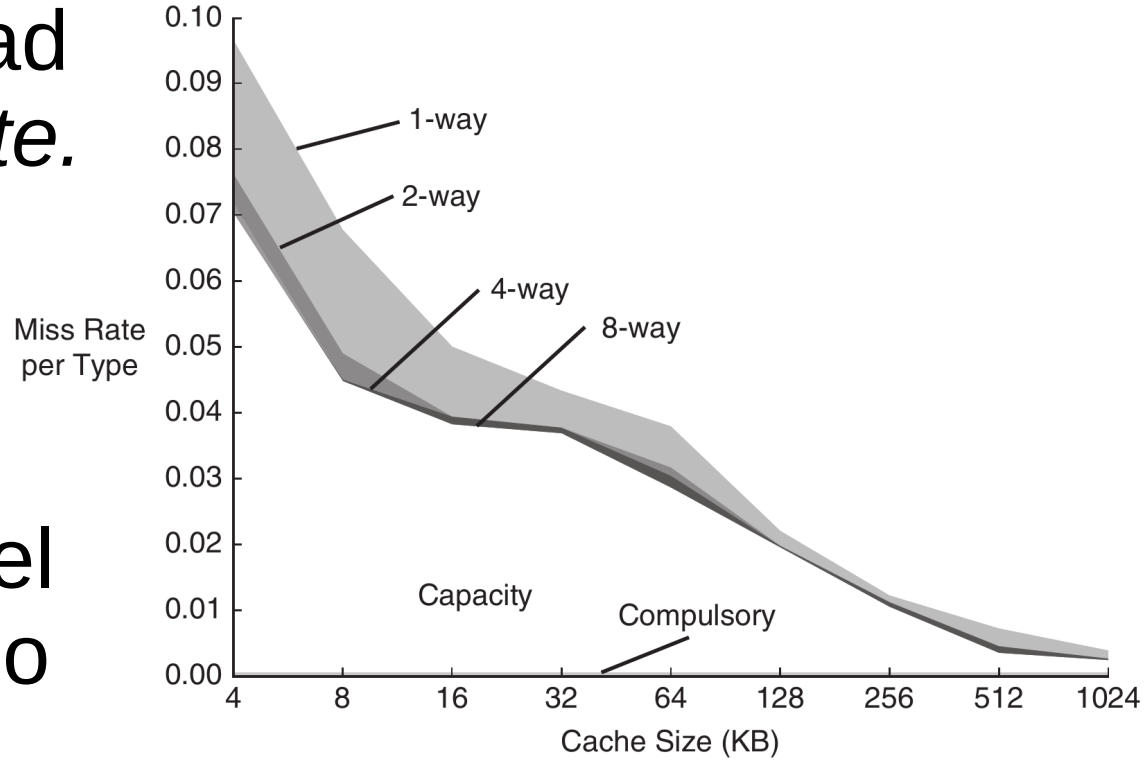
- Por lo general, el segundo nivel de cache contiene al primer nivel.
- El tamaño del segundo nivel debe ser mucho mayor que el primero (sino se incrementaría el *miss rate* local).

Diseño del segundo nivel de caché

Grado de asociatividad

- Reemplazar *direct-mapped* por algún grado de asociatividad disminuye el *miss rate.*
- Se puede reducir la penalización de un faltante en primer nivel, disminuyendo el *miss rate* del segundo nivel.

Miss rate vs Cache size
SPEC2000 benchmark



Diseño del segundo nivel de caché

Multilevel inclusion / exclusion

- ¿Los datos de L_1 están en L_2?
- *Multilevel inclusion*
  - Los datos de L_1 están siempre en L_2
  - Vista consistente de la jerarquía
  - L_2 debe ser mucho mayor que L_1
- *Multilevel exclusion*
  - Los datos en L_1 nunca están en L_2
  - L_2 puede ser apenas mayor que L_1
  - Un faltante en L_1 implica hacer un *swap* y no un reemplazo

Prefetching

- La implementación básica de la caché lleva los datos a caché bajo demanda.
- *Prefetching* es una técnica predictiva que trata de llevar datos e instrucciones a la caché antes de que vayan a ser usados.
  - Disminuye el tiempo de acceso a los datos
  - Puede hacerse con la asistencia de instrucciones especiales o por hardware.
  - Es aplicable a cualquier nivel de la jerarquía

Prefetching

Los predictores de prefetching se pueden evaluar en función de tres criterios:

- Precisión:

  número de prefetches útiles / número de prefetches generados

- Cobertura:

  número de prefetches útiles / número de misses sin prefetch

- Oportunidad:

  Mide cuán oportuno o a tiempo fue el momento en que se hizo el prefetch.

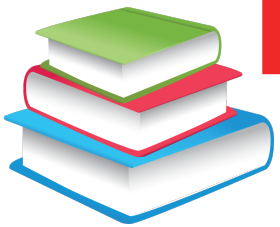
  Si el prefetch es muy temprano puede desplazar datos que que sean necesarios antes que el prefetch (polución de la caché).

  Si el prefetch es muy tarde, el dato se necesita antes de que concretar el prefetch (ciclos de hit-wait, igual se considera útil).

Caché no bloqueante

- En una microarquitectura en pipeline que permite ejecución fuera de orden las instrucciones que siguen a un load que ocasiona un fallo en la caché, podrían continuar a las estaciones de reservación y ejecutar si tienen todos los operandos.
- Permitir accesos a la caché mientras se está esperando que se resuelva un faltante no agrega mucha complejidad al diseño.
- Esta optimización de hit-under-miss reduce la penalización.
- La cache no bloqueante (*lockup-free o nonblocking cache*) permiten varios faltantes concurrentes.

Bibliografía



- *Capítulo 5 y Apéndice B.* David A. Patterson & John L. Hennessy. *Computer Organization and Design. The Hardware/Software Interface.* Elsevier Inc. 2014, 5ta Ed.
- *Capítulo 2 y 6*. *Multiprocessor Architecture. From simple pipelines to chip multiprocessor*. Jean-Loup Baer. Cambridge University Press. 2010.
- *Capítulo 8.* David Money Harris & Sarah L. Harris. *Digital Design and Computer Architecture*. Elsevier. 2013, 2da Ed.

Suplementaria

- *Apéndice B.* John L. Hennessy & David A. Patterson. *Computer Architecture. A Quantitative Approach*. Elsevier Inc. 2012, 5ta Ed.