

Clase 19 - Memoria Caché

Profesor: **IIC2343 - Arquitectura de Computadores**

- Felipe Valenzuela González

Correo:

frvalenzuela@alumni.uc.cl

Resumen de la clase pasada

Generalmente, el código que escribimos cumple dos principios de localidad

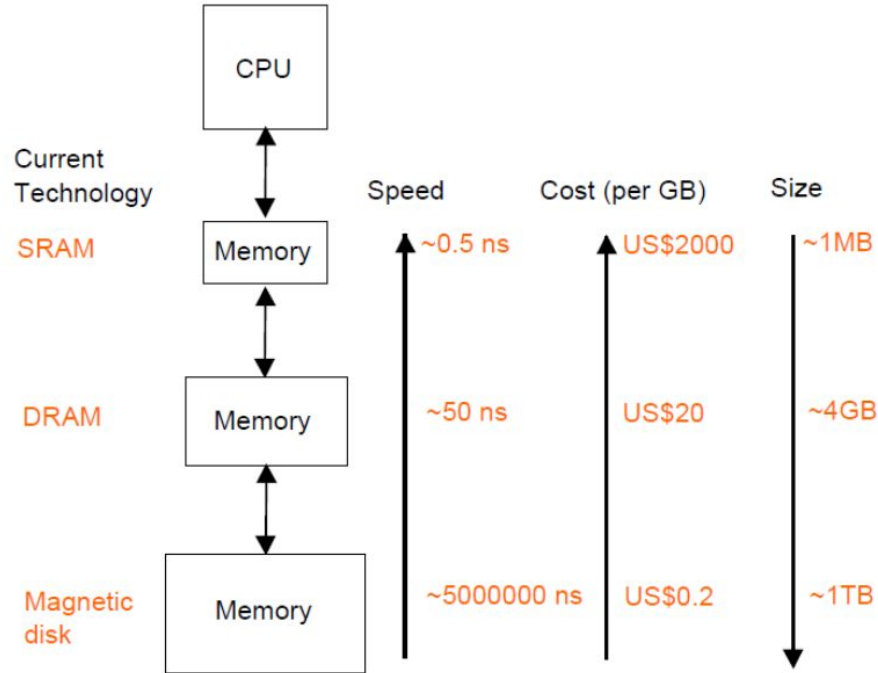
1. Principio de localidad temporal

Es probable que un dato obtenido de memoria sea usado posteriormente en otra operación

2. Principio de localidad espacial

Es probable que datos cercanos al buscado también sean usados (bloque)

Jerarquía de Memoria es una buena opción para mejorar el rendimiento



Evaluacion Tiempo promedio

Tiempo promedio en caso de *hit*

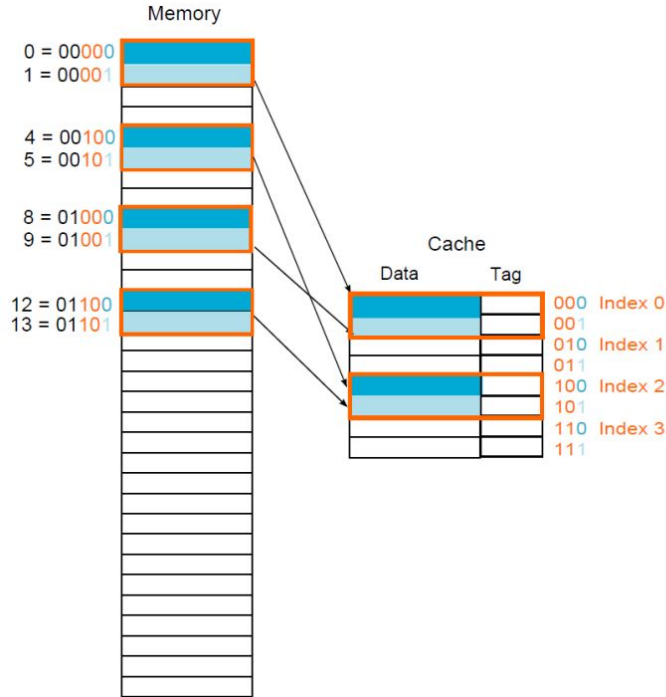
Tiempo promedio en caso de *miss* *

$$\begin{aligned} TP &= HR * HT + (1 - HR) * (HT + MP) \\ &= HT + (1 - HR) * MP \end{aligned}$$

Mapeo directo (directly mapped) es la función de correspondencia más simple

Revisemos un ejemplo con los siguientes datos:

- Memoria principal: 32 bytes (32 palabras)
- Caché de 8 bytes y 4 líneas



Mapeo directo (directly mapped) es la función de correspondencia más simple

Además de almacenar las palabras y el tag, cada línea debe tener un bit de validez.

- Este bit se utiliza cuando la caché comienza a ser llenada => palabras son no válidas.
- Finalicemos el mapeo directo con un ejemplo de su funcionamiento:
 - Caché de 8 bytes y 4 líneas
 - Acceso a direcciones de memoria: 12, 13, 14, 4, 12, 0

Funciones de correspondencia - Mapeo Directo

Ventajas

- Cómputo sencillo y directo

Desventajas

- Retención entre bloques y líneas
- Desaprovechamiento del resto de la caché

¿Dudas?

Funciones de correspondencia - *Fully associative*

- En este esquema, cada bloque de memoria puede asociarse a **cualquier línea de caché**
- Aprovecha todo el espacio y maximiza *hit-rate*
- Como se puede ocupar cualquier línea, la composición del tag para identificar si un dato se encuentra en caché se **complejiza**

Funciones de correspondencia - *Fully associative*

- Finalicemos con un ejemplo de su funcionamiento:
 - Caché de 8 bytes y 4 líneas, Memoria principal de 16 bytes
 - Acceso a direcciones de memoria: 7, 10, 13 y 15

Funciones de correspondencia - *Fully associative*

Ventajas

- Se ocupa toda la caché, aprovechándola al máximo

Desventajas

- El *hit-time* es significativamente mayor al tener que buscar coincidencia de tag en todas las líneas
- El almacenamiento del tag crece sustancialmente

¿Dudas?

Funciones de correspondencia - *N way associative*

En este esquema, también llamado **Set associative**, cada bloque de memoria puede asociarse a un conjunto de líneas de la caché.

- Divide a la caché en conjuntos de **N líneas !!No son N conjuntos!!**
- Lo mejor de los dos mundos: Cada bloque posee un mapeo directo a un conjunto y dentro de un conjunto, un bloque se asocia a cualquier línea

Funciones de correspondencia - *N way associative*

- En este esquema, cada bloque de memoria puede asociarse a **cualquier línea de caché**
- Aprovecha todo el espacio y maximiza *hit-rate*
- Como se puede ocupar cualquier línea, la composición del tag para identificar si un dato se encuentra en caché se **complejiza**

Funciones de correspondencia - *N way associative*

- Finalicemos con un ejemplo de su funcionamiento:
 - Caché de 8 bytes y 4 líneas, Memoria principal de 16 bytes
 - Acceso a direcciones de memoria: 7, 10, 13

Funciones de correspondencia - *N way associative*

Ventajas

- Hit-time menor respecto a la función fully associative
- Mayor uso de caché respecto a directly mapped
- Ocupa menos bits de tag

Desventajas

- Hit-time mayor respecto a directly mapped
- Menor uso de caché respecto a fully associative

¿Dudas?

Políticas de reemplazos

- **Bélády**: El bloque que se utilizará más lejos en el futuro se saca. Óptimo no alcanzable en la práctica.
- **First-in First-out (FIFO)**: El primer bloque en entrar es el primero en salir. Simple, pero tonto.
- **Least Frequently Used (LFU)**: El bloque con menos accesos se saca. Mejor que FIFO, solo un poco más complejo.
- **Least Recently Used (LRU)**: El bloque con mayor tiempo sin accesos se saca. Complejo, requiere timestamp. En general el de mejor rendimiento.
- **Random**: Muy rápido y con rendimiento algo inferior a LFU y LRU.

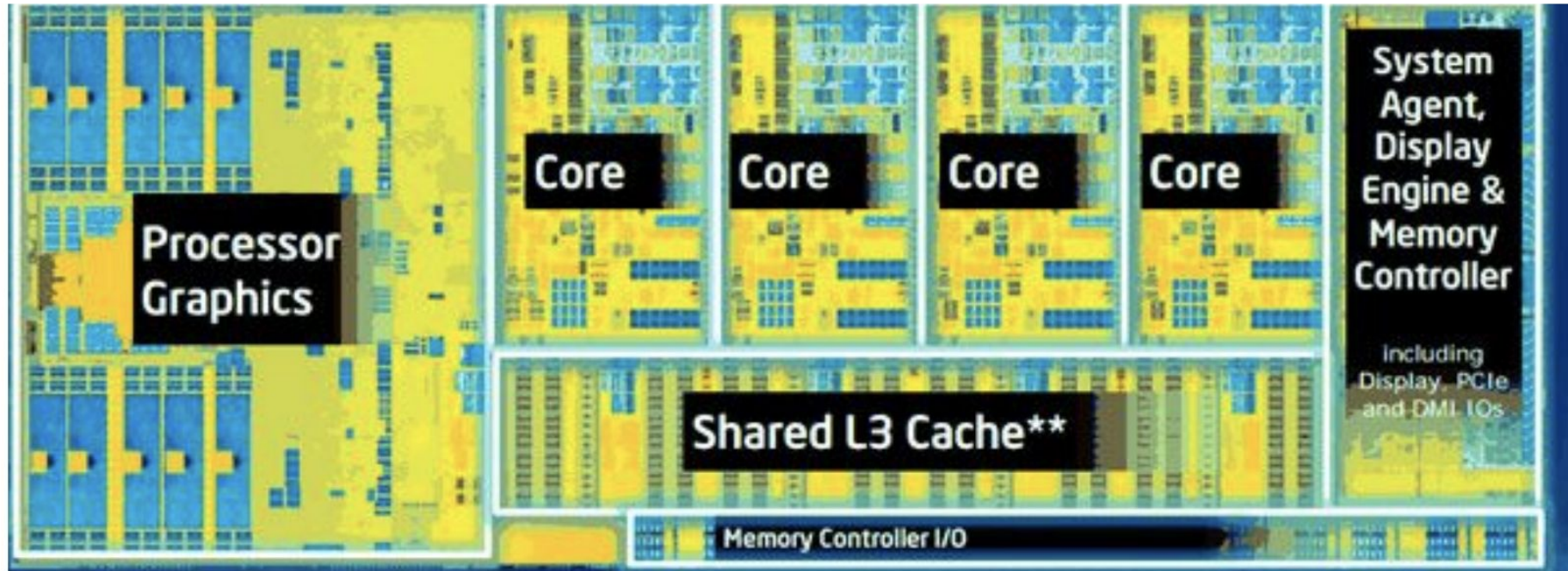
Políticas de escritura

- **Write-through:** el bloque modificado es escrito inmediatamente en la memoria principal
- **Write-back:** el bloque modificado es escrito en la memoria principal sólo cuando va a ser sustituido

Memoria caché - Tipos de memoria

- Sectores de la memoria visitados por accesos a datos e instrucciones son generalmente muy distintos.
- Patrones de acceso también son completamente distintos.
- Cachés que tratan igual a datos e instrucciones (**caché unified**) pueden sufrir grandes bajas en el *hitrate*.
- Para evitarlo, se utilizan cachés divididas internamente en datos e instrucciones (**caché split**)

Memoria caché - Tipos de memoria



¿Dudas?

Memoria caché - Ejercicio en clases

- Suponga que tiene una memoria principal de 16 bytes y una memoria caché de 8 bytes y 4 líneas. Además, asuma que tiene un programa que accede, en este orden, a las direcciones de la memoria principal: 0, 1, 5, 7, 10, 13, 4, 6. Obtenga el estado final de la memoria caché (en una tabla) y el hit-rate para cada una de las siguientes funciones de correspondencia: directly mapped, fully associative, 2-way associative. Puede asumir una política de reemplazo LRU, en caso de necesitarla.

Memoria caché - Ejercicio prueba 2024-1 - I3

- (c) (4 ptos.) Suponga que, para una memoria principal de 256 palabras de 1 byte, posee una caché de 16 líneas y 4 palabras por línea con el siguiente estado intermedio:

| Línea | Validez | Tag | Timestamp |
|-------|---------|-----|-----------|
| 0 | 1 | 20 | 1 |
| 1 | 1 | 5 | 11 |
| 2 | 1 | 10 | 4 |
| 3 | 1 | 25 | 9 |
| 4 | 1 | 31 | 8 |
| 5 | 1 | 16 | 3 |
| 6 | 1 | 1 | 6 |
| 7 | 1 | 8 | 2 |

| Línea | Validez | Tag | Timestamp |
|-------|---------|-----|-----------|
| 8 | 0 | 10 | - |
| 9 | 0 | 20 | - |
| 10 | 0 | 30 | - |
| 11 | 0 | 5 | - |
| 12 | 0 | 15 | - |
| 13 | 0 | 2 | - |
| 14 | 0 | 1 | - |
| 15 | 0 | 0 | - |

La columna “Línea” representa el índice de línea; “Validez” representa el *valid bit*; “Tag” representa el valor decimal del *tag* que indica el bloque de memoria almacenado en la línea; y “Timestamp” representa el tiempo **desde el último acceso a la línea**. A partir de este estado, se realiza el acceso a memoria de las siguientes direcciones: 0x54, 0xE4, 0xE3, 0xCA, 0xCC, 0xA1. Indique, para cada acceso, si existe un *hit* o *miss*; si corresponde, la línea de la caché que es modificada; y el *hit-rate* para la función de correspondencia *8-way associative*. Asuma una la política de reemplazo LRU. Para responder, complete las tablas adjuntas al enunciado. El *tag* de cada línea lo puede escribir en base binaria o decimal. No necesita indicar el *timestamp* final por línea, pero sí debe considerar su valor en caso de reemplazos.

Clase 19 - Memoria Caché

Profesor: **IIC2343 - Arquitectura de Computadores**

- Felipe Valenzuela González

Correo:

frvalenzuela@alumni.uc.cl