Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores

Memoria Caché

Profesor: Jurgen Heysen

¿Cuál es el flujo de una instrucción, por ejemplo ADD [BX], AL, en un computador x86?

1.	Lectura de instrucción desde memoria (Fetch)	50 ns
2.	Decode (Unidad de Control)	0,5 ns
3.	Obtener dato de memoria	50 ns
4.	Execute (ALU)	0,5 ns
5.	Escribir resultado en memoria (Write Back)	50 ns

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

Dirección	Label	Instrucción/Dato
	CODE:	
0x00	start:	MOV CL, [var1]
0x01	while:	MOV AL,[res]
0x02		ADD AL,**[var2]**
0x03		MOV [res],AL
0x04		SUB CL,1
0x05		CMP CL,0
0x06		JNE while
	DATA:	
0x07	var1	3
0x08	**var2**	**2**
0x09	res	0

Generalmente, el código que escribimos cumple dos principios de localidad

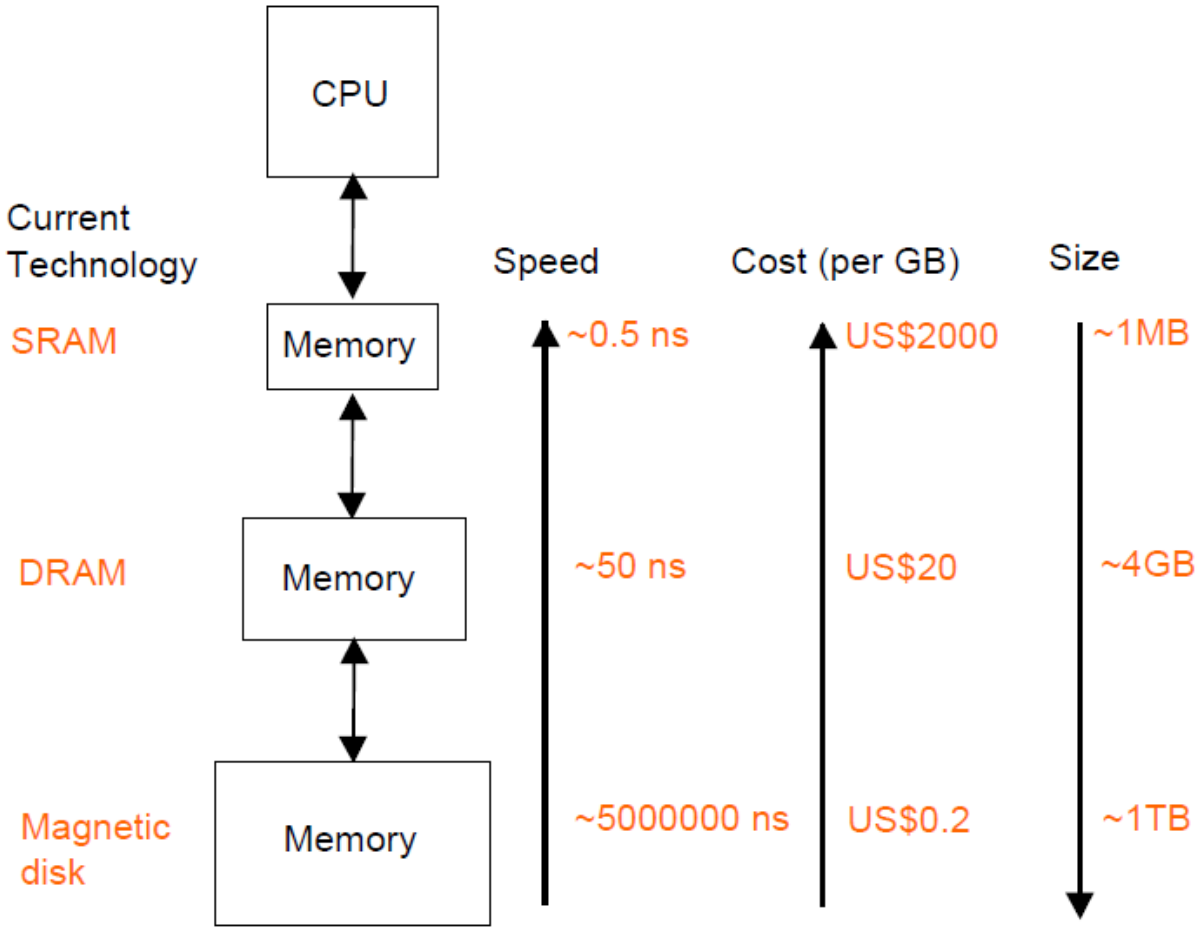
1. Principio de localidad temporal
   Es probable que un dato obtenido de memoria sea usado posteriormente en otra operación

2. Principio de localidad espacial
   Es probable que datos cercanos al buscado también sean usados (bloque)

Dirección	Label	Instrucción/Dato
	CODE:	
0x00	start:	MOV SI, 0
0x01		MOV AX, 0
0x02		MOV BX, arreglo
0x03		MOV CL, [n]
0x04	while:	CMP SI, CX
0x05		JGE end
0x06		MOV DX, **[BX + SI]**
0x07		ADD AL, DL
0x08		INC SI
0x09		JMP while
0x0A	end:	DIV CL
0x0B		MOV [prom], AL
	DATA:	
0x0C	**arreglo**	**6**
0x0D		**7**
0x0E		**4**
0x0F		**5**
0x10		**3**
0x11	n	5
0x12	prom	0

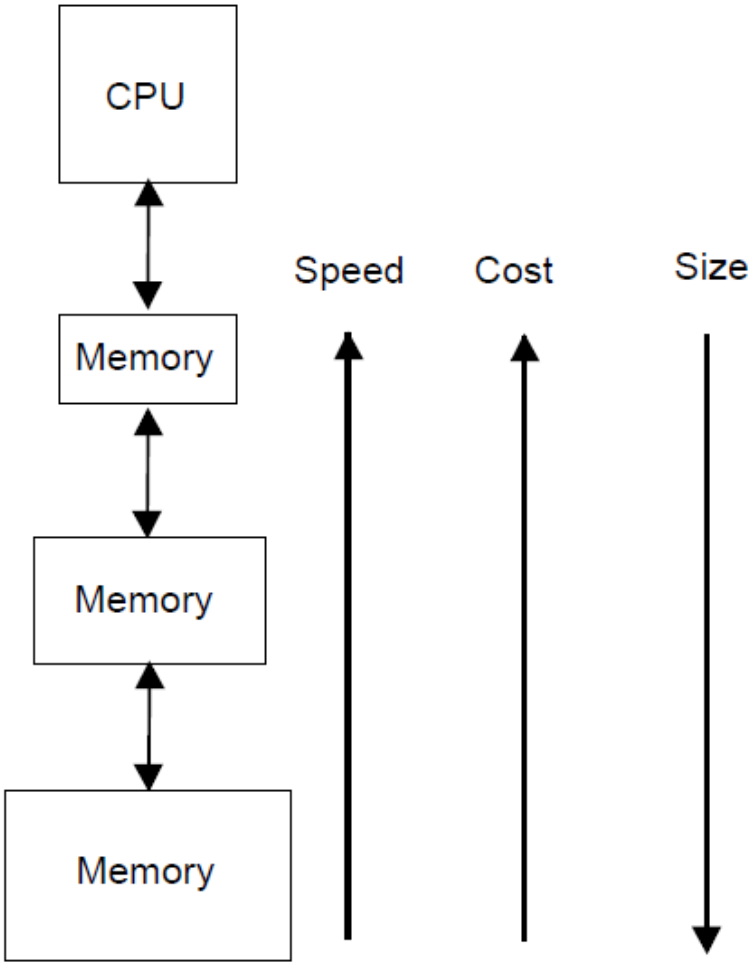
¿Como podemos aprovechar estos principios para acelerar la ejecución de una instrucción?

Jerarquía de Memoria es una buena opción para mejorar el rendimiento



¿Cómo evaluamos una jerarquía de memoria?

Hit-rate, miss-rate, hit-time, miss-penalty



Memoria caché ocupa el primer nivel de la jerarquía

- La memoria caché se encuentra en la CPU…
- …pero la CPU no tiene idea que existe.
- Memoria principal y caché se dividen en bloques y líneas, respectivamente (localidad espacial).
- Para describir el funcionamiento, sólo es necesario entender su comunicación con la CPU y con el siguiente nivel de la jerarquía (memoria principal) .

Controlador de caché es esencial para implementar una jerarquía de memoria

- CPU sólo conoce, con respecto a la jerarquía de memoria, el tamaño de la memoria principal.
- Controlador de caché es encargado de realizar la comunicación y coordinación.
- Las funciones que debe implementar el controlador pueden definirse a grandes rasgos en:

1. Mecanismo de acceso a datos
2. Política de escritura

Métodos de acceso y escritura definen el comportamiento y rendimiento del caché

1. Mecanismos de acceso a datos:
   - Funciones de correspondencia
   - Políticas de reemplazo (localidad temporal)

2. Políticas de escritura

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

- Con este esquema, cada bloque de la memoria principal se asocia a sólo una línea de la caché.
- Cada línea en la caché es identificada con un índice.
- Tamaño de líneas y cantidad de estas siempre serán potencias de 2 (por simplicidad).
- La dirección del bloque de memoria será la dirección de la primera palabra de éste.

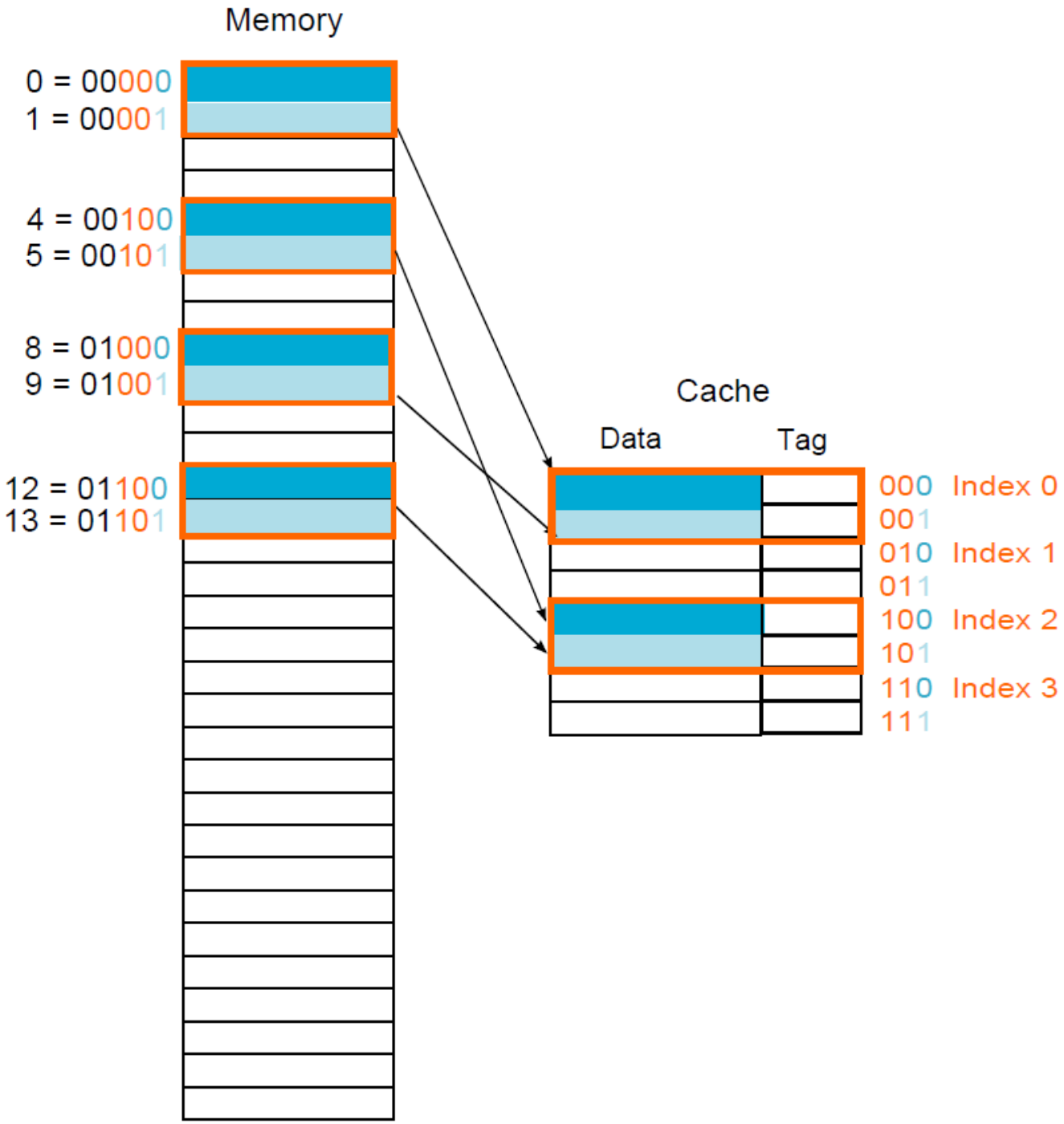
Mapeo directo (*directly mapped*) es la función de correspondencia más simple

Revisemos un ejemplo con los siguientes datos:

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

¿Cómo asocio el bloque 5 de la memoria principal, a una línea de la caché?

Dada la dirección de memoria de una palabra en la memoria principal, ¿A qué línea está asociada en la caché?


Memory
0 = 00000
1 = 00001
4 = 00100
5 = 00101
8 = 01000
9 = 01001
12 = 01100
13 = 01101
Cache
Data
Tag
000 Index 0
001
010 Index 1
011
100 Index 2
101
110 Index 3
111

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

Volvamos a las funciones de correspondencia

- La retención entre bloques/líneas es el principal problema del mapeo directo.
- Además desaprovecha el resto del caché.
- ¿Qué otro esquema puede usarse?

Idea: Relajar la restricción de ubicaciones fijas.

Fully Associative

- En este esquema, cada bloque de memoria puede asociarse a cualquier línea de caché.
- Aprovecha todo el espacio y maximiza *hit-rate.*
- Hace al hardware mucho más complejo.
- ¿Cómo mejoramos esto?

N-way Associative

- Presenta un punto medio entre *directly mapped* y *fully associative.*
- Divide a la caché en conjuntos de N líneas.

  (!!!! no en N conjuntos !!!!!)
- Cada bloque tiene un mapeo directo a un conjunto.
- Dentro de un conjunto, cada bloque puede ubicarse en cualquier línea (fully associative).

Métodos de acceso y escritura definen el comportamiento y rendimiento del caché

1. Mecanismos de acceso a datos:
   - Funciones de correspondencia
   - Políticas de reemplazo

2. Políticas de escritura

¿Cómo manejamos el reemplazo de bloques en esquemas asociativos?

- Béládу: La línea que se utilizará más lejos en el futuro se saca. Óptimo no alcanzable en la práctica.
- First-in First-out (FIFO): El primero en entrar es el primero en salir. Simple, pero tonto.
- Least Frequently Used (LFU): La línea con menos accesos se saca. Mejor que FIFO, sólo un poco más complejo.
- Least Recently Used (LRU): La línea 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.

Métodos de acceso y escritura definen el comportamiento y rendimiento del caché

1. Mecanismos de acceso a datos:
   - Funciones de correspondencia
   - Políticas de reemplazo

2. Políticas de escritura

Es fundamental no perder lo ganado con los mecanismos de acceso a datos

- Write-through: la línea/bloque modificada es escrita inmediatamente en la memoria principal

- Write-back: la línea/bloque modificada es escrita en la memoria principal sólo cuando va a ser sustituida.

¿Qué problema puede traer este último esquema?

no se utiliza la nueva definicion de la variable cuando lo usa un dispositivo I/O

Dirección	Label	Instrucción/Dato
	CODE:	
0x00	start:	MOV CL, [var1]
0x01	while:	MOV AL,[res]
0x02		ADD AL,[**var2**]
0x03		MOV [res],AL
0x04		SUB CL,1
0x05		CMP CL,0
0x06		JNE while
	DATA:	
0x07	var1	3
0x08	**var2**	**2**
0x09	res	0

Distribuciones de accesos a datos e instrucciones son distintas

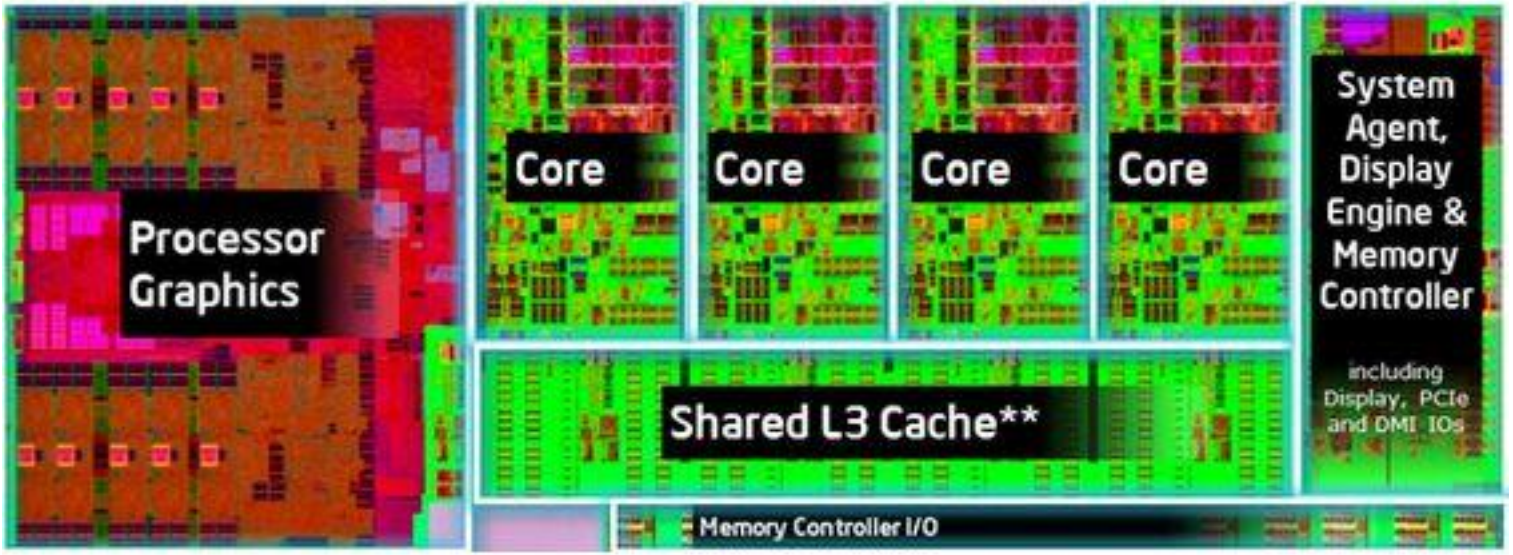
- Sectores de la memoria visitados por accesos a datos e instrucciones son generalmente muy distintos.
- Patrones de acceso también son completamente distintos.
- Cachés que tratan igual a datos e instrucciones (caché unified) pueden sufrir grandes bajas en el hit-rate.
- Para evitarlo, se utilizan cachés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

Un caso real: Intel Haswell (i3, i5, i7)

Caché de 3 niveles:

- **L1:** Caché *split* de 64KB por núcleo (32KB para instrucciones y 32KB para datos). Líneas de 64B, 8-way associative + pseudo LRU, write-back.
- **L2:** Caché *unified* 256KB por núcleo. Líneas de 64B, 8-way, 8-way associative + pseudo LRU, write-back.
- **L3:** Entre 2MB y 20 MB de caché *unified* compartida entre núcleos. Líneas de 64B, 12-way associative + pseudo LRU, write-back.

Un caso real: Intel Haswell (i3, i5, i7)



Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores

Memoria Caché

Profesor: Jurgen Heysen