2011-06-16

2011年 前学期 TOKYO TECH

計算機アーキテクチャ 第一 (E)

8. メモリ2: キャッシュ

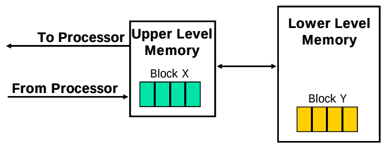
吉瀬 謙二　計算工学専攻
kise_at_cs.titech.ac.jp
W641講義室　木曜日13:20 ー 14:50

Acknowledgement

- Lecture slides for Computer Organization and Design, Third Edition, courtesy of **Professor Mary Jane Irwin**, Penn State University
- Lecture slides for Computer Organization and Design, third edition, Chapters 1-9, courtesy of **Professor Tod Amon,** Southern Utah University.

The Memory Hierarchy: Why Does it Work?

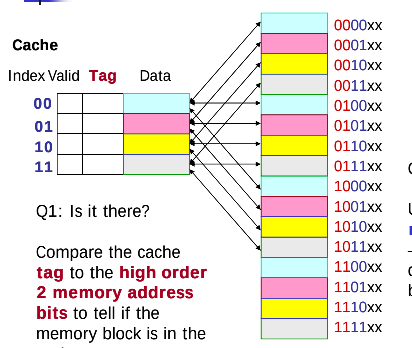
- **Temporal Locality** (時間的局所性, Locality in Time):
  - ⇒ Keep **most recently accessed** data items closer to the processor
- **Spatial Locality** (空間的局所性, Locality in Space):
  - ⇒ Move blocks consisting of **contiguous words** to the upper levels



Cache

- Two questions to answer (in hardware):
  - Q1: **How do we know if a data item is in the cache?**
  - Q2: **If it is, how do we find it?**
- **Direct mapped**
  - For each item of data at the lower level, there is exactly one location in the cache where it might be - so lots of items at the lower level must **share** locations in the upper level
  - Address mapping: **(block address) modulo (# of blocks in the cache)**
  - First, consider block sizes of **one word**

Caching: A Simple First Example



Two low order bits define the byte in the word (32-b words)

Q2: How do we find it?

Use **next 2 low order memory address bits** – the **index** – to determine which cache block

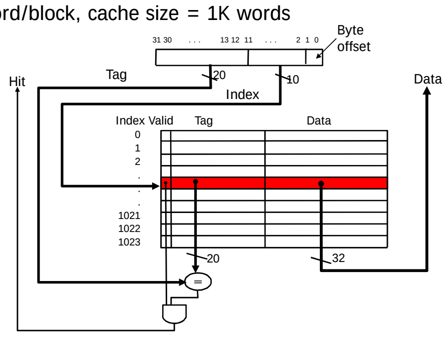
Q1: Is it there?

Compare the cache **tag** to the **high order 2 memory address bits** to tell if the memory block is in the cache

(block address) modulo (# of blocks in the cache)

MIPS Direct Mapped Cache Example

- One word/block, cache size = 1K words



What kind of locality are we taking advantage of?

Direct Mapped Cache

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15

Tag **0** miss

00	Mem(0)

1 miss

00	Mem(0)
00	Mem(1)

2 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)

3 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 miss

01 ~~00~~	~~Mem(0)~~ 4
00	Mem(1)
00	Mem(2)
00	Mem(3)

3 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

15 miss

01	Mem(4)
00	Mem(1)
00	Mem(2)
11 ~~00~~	~~Mem(3)~~ 15

- 8 requests, 6 misses

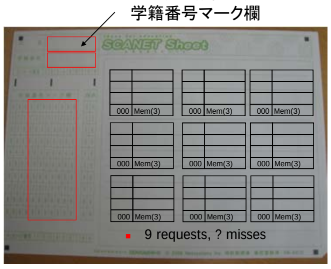
Exercise

- Consider the main memory word reference string
  - 3, 2, 18, 3, 16, 2, 3, 18, 3

Tag **3** miss

000	Mem(3)

氏名，学籍番号，
学籍番号マーク欄



- 9 requests, ? misses

Another Reference String Mapping

- Consider the main memory word reference string

0 4 0 4 0 4 0 4

0 miss

00	Mem(0)

4 miss

01 ~~00~~	~~Mem(0)~~ 4

0 miss

00 ~~01~~	~~Mem(4)~~ 0

4 miss

01 ~~00~~	~~Mem(0)~~ 4

0 miss

00 ~~01~~	~~Mem(4)~~ 0

4 miss

01 ~~00~~	~~Mem(0)~~ 4

0 miss

00 ~~01~~	~~Mem(4)~~ 0

4 miss

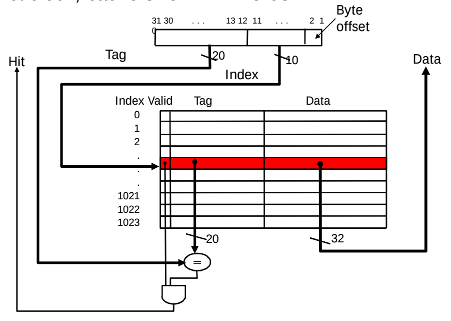
01 ~~00~~	~~Mem(0)~~ 4

- 8 requests, 8 misses

- Ping pong effect due to **conflict** misses - two memory locations that map into the same cache block

MIPS Direct Mapped Cache Example

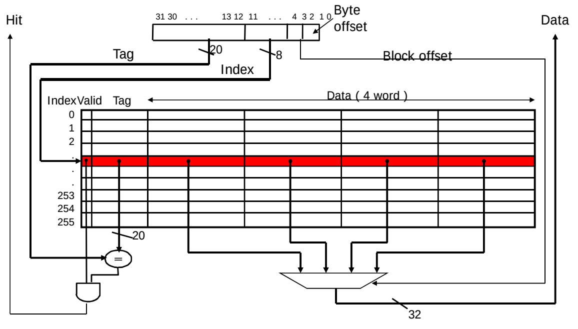
- One word/block, cache size = 1K words



What kind of locality are we taking advantage of?

Multiword Block Direct Mapped Cache

- Four words/block, cache size = 1K words



What kind of locality are we taking advantage of?

Direct Mapped Cache again!

- Consider the main memory word reference string

0 1 2 3 4 3 4 15

0 miss

00	Mem(0)

1 miss

00	Mem(0)
00	Mem(1)

2 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)

3 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 miss

01 ~~00~~	~~Mem(0)~~ 4
00	Mem(1)
00	Mem(2)
00	Mem(3)

3 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

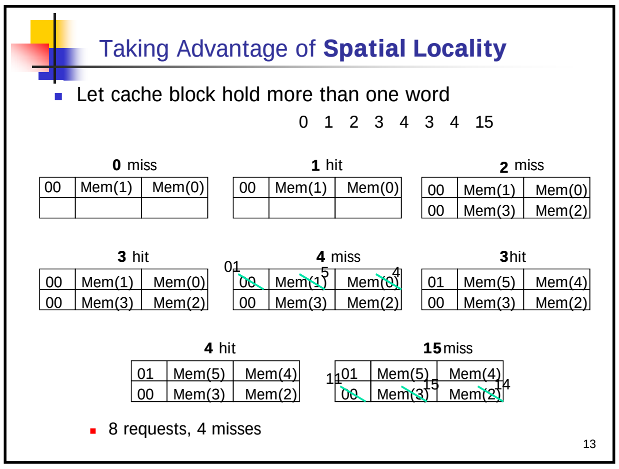
4 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

15 miss

01	Mem(4)
00	Mem(1)
00	Mem(2)
11 ~~00~~	~~Mem(3)~~ 15

- 8 requests, 6 misses



Taking Advantage of **Spatial Locality**

- Let cache block hold more than one word

0 1 2 3 4 3 4 15

0 miss

00	Mem(1)	Mem(0)

1 hit

00	Mem(1)	Mem(0)

2 miss

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

4 miss

01 ~~00~~	5 ~~Mem(1)~~	4 ~~Mem(0)~~
00	Mem(3)	Mem(2)

3hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

4 hit

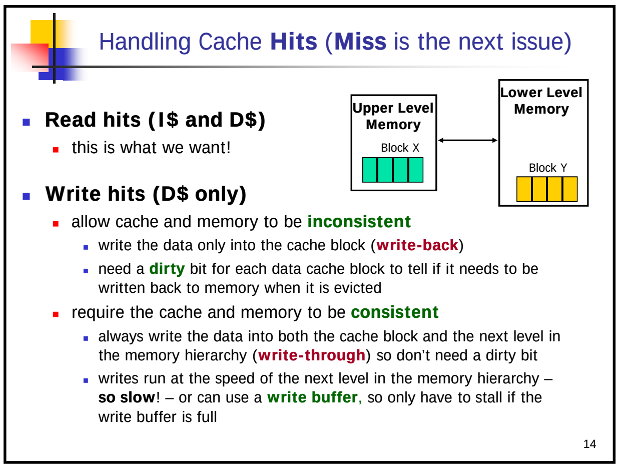
01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

15miss

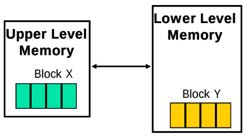
01	Mem(5)	Mem(4)
11 ~~00~~	15 ~~Mem(3)~~	14 ~~Mem(2)~~

- 8 requests, 4 misses

13

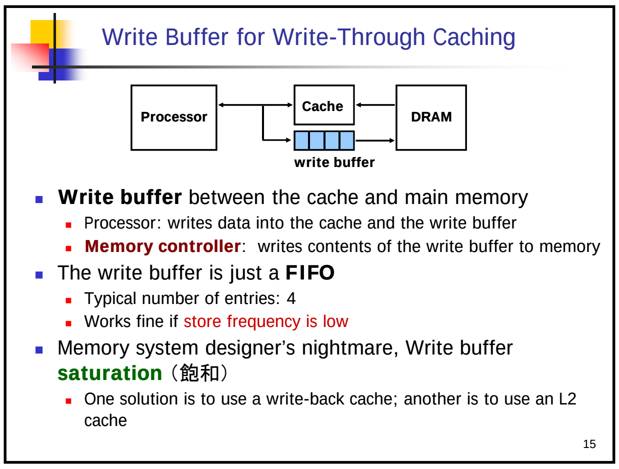


Handling Cache **Hits** (**Miss** is the next issue)

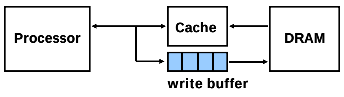


- **Read hits (I$ and D$)**
  - this is what we want!
- **Write hits (D$ only)**
  - allow cache and memory to be **inconsistent**
    - write the data only into the cache block (**write-back**)
    - need a **dirty** bit for each data cache block to tell if it needs to be written back to memory when it is evicted
  - require the cache and memory to be **consistent**
    - always write the data into both the cache block and the next level in the memory hierarchy (**write-through**) so don't need a dirty bit
    - writes run at the speed of the next level in the memory hierarchy – **so slow**! – or can use a **write buffer**, so only have to stall if the write buffer is full

14

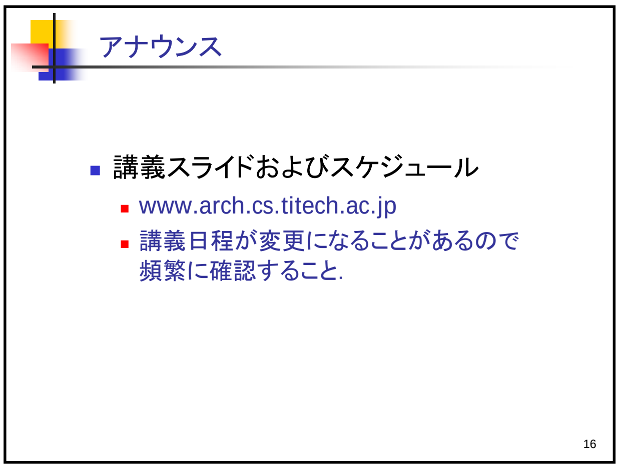


Write Buffer for Write-Through Caching



- **Write buffer** between the cache and main memory
  - Processor: writes data into the cache and the write buffer
  - **Memory controller**: writes contents of the write buffer to memory
- The write buffer is just a **FIFO**
  - Typical number of entries: 4
  - Works fine if store frequency is low
- Memory system designer's nightmare, Write buffer **saturation**（飽和）
  - One solution is to use a write-back cache; another is to use an L2 cache

15



アナウンス

- 講義スライドおよびスケジュール
  - www.arch.cs.titech.ac.jp
  - 講義日程が変更になることがあるので頻繁に確認すること.

16