2011-06-28

2012年 前学期 TOKYO TECH

計算機アーキテクチャ 第一 (E)

メモリ3: 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

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	4
00	Mem(0)
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

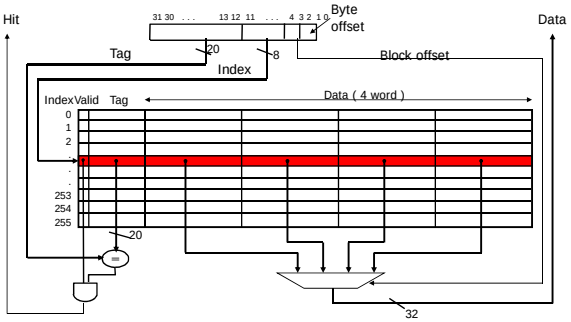
01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

11 15

- 8 requests, 6 misses

Multiword Block Direct Mapped Cache

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



What kind of locality are we taking advantage of?

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	5	4
00	Mem(1)	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)

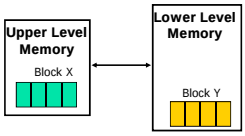
15miss

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

15

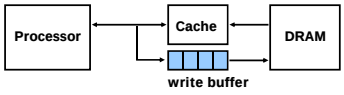
- 8 requests, 4 misses

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

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

Sources of Cache Misses

- **Compulsory** (**初期参照ミス**, cold start or process migration, first reference):
  - First access to a block, "cold" fact of life, not a whole lot you can do about it
  - If you are going to run "millions" of instruction, compulsory misses are insignificant
- **Conflict** (**競合性ミス**, collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase **associativity**
- **Capacity** (**容量性ミス**):
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size

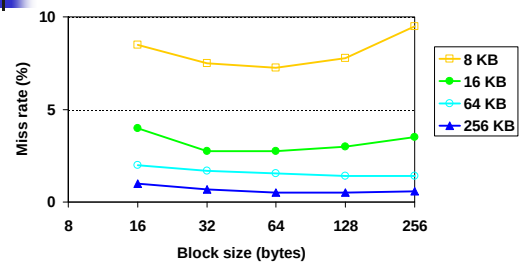
7

Handling Cache **Misses**

- **Read misses (I$ and D$)**
  - **stall** (ストール，立ち往生させる)the entire pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume
- **Write misses (D$ only)**
  - **Write allocate**
    - (a) write the word into the cache updating both the tag and data, no need to check for cache hit, no need to stall
    - **(b) stall** the pipeline, fetch the block from next level in the memory hierarchy, install it in the cache, write the word from the processor to the cache, then let the pipeline resume
  - **No-write allocate** – skip the cache write and just write the word to the write buffer (and eventually to the next memory level), no need to stall if the write buffer isn't full;
    must invalidate **the cache block** since it will be **inconsistent**

8

Miss Rate vs **Block Size** vs **Cache Size**

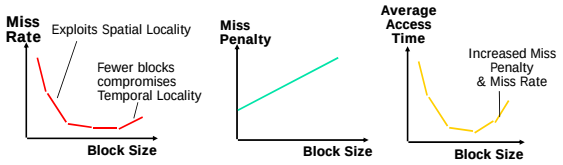


- Miss rate goes up if the block size becomes a significant fraction of the cache size
  because the number of blocks that can be held in the same size cache is smaller

9

Block Size **Tradeoff**

- Larger block sizes take advantage of spatial locality **but**
  - If the block size is too big relative to the cache size, the miss rate will go up
  - Larger block size means larger miss penalty
    - Latency to first word in block + transfer time for remaining words



❑ In general, **Average Memory Access Time**
= Hit Time + **Miss Penalty x Miss Rate**

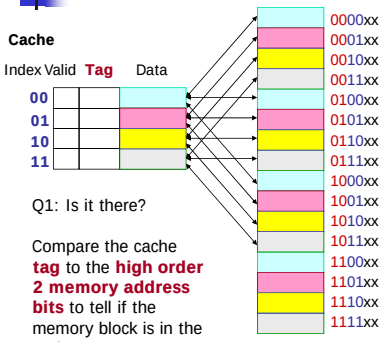
10

Reducing Cache Miss Rates, **associativity**

- **Allow more flexible block placement**
  - In a **direct mapped cache** a memory block maps to exactly **one cache block**
  - At the other extreme, could allow a memory block to be mapped to any cache block – **fully associative cache**
  - A compromise is to divide the cache into **sets** each of which consists of **n "ways"** (**n-way set associative**).
    A memory block maps to a unique set and can be placed in any way of that set (so there are **n choices**)

11

Caching: **Direct mapped (First Example)**



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

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

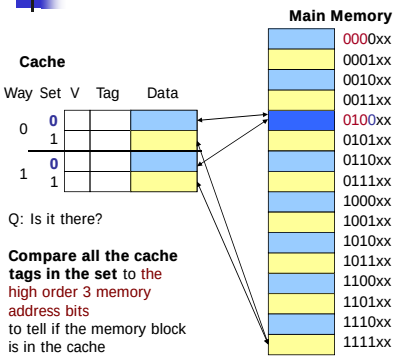
Q2: How do we find it?

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

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

12

Set Associative Cache Example



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

Q: How do we find it?

Use next 1 low order memory address bit to determine which cache set

Q: Is it there?

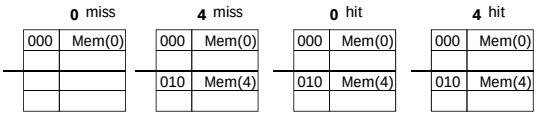
Compare all the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache

Another Reference String Mapping

- Consider the main memory word reference string

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

0 4 0 4 0 4 0 4

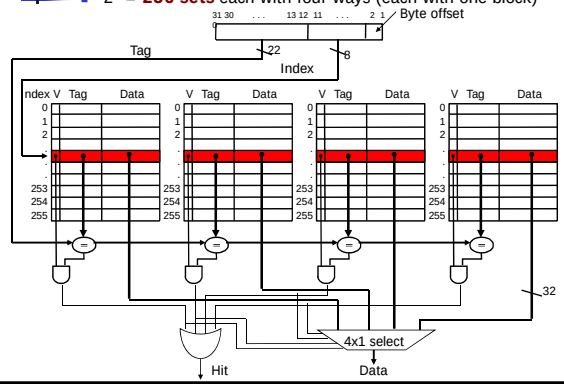


- 8 requests, 2 misses

- Solves the **ping pong effect** in a direct mapped cache due to conflict misses

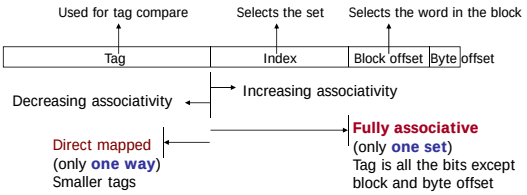
Four-Way Set Associative Cache

- 2^8 = **256 sets** each with four ways (each with one block)



Range of Set Associative Caches

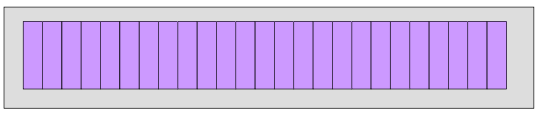
- For a fixed size cache



Costs of Set Associative Caches

- **N-way set associative** cache costs
  - N comparators (delay and area)
  - MUX delay (set selection) before data is available
  - Data available **after** set selection and Hit/Miss decision.
- **When a miss occurs**,
  which way's block do we pick **for replacement** ?
  - **Least Recently Used (LRU):**
    the block replaced is the one that has been unused for the longest time
    - Must have hardware to keep track of when each way's block was used
    - For **2-way set associative**, takes **one bit per set** → set the bit when a block is referenced (and reset the other way's bit)
  - **Random**

Cache

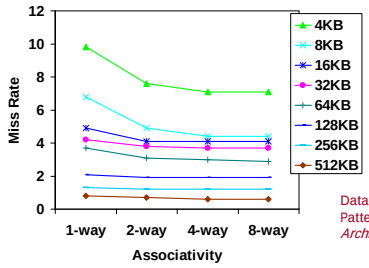


本棚

机

Benefits of Set Associative Caches

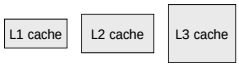
- The choice of direct mapped or set associative depends on **the cost of a miss** versus **the cost of implementation**



Data from Hennessy & Patterson, *Computer Architecture*, 2003

- Largest gains are in going from **direct mapped** to **2-way**

Reducing Cache Miss Rates by **multiple levels**



- Enough room on the die for **bigger L1 caches** *or* for a **second level of caches** – normally a **unified** L2 cache (i.e., it holds both instructions and data) and in some cases even a **unified L3 cache**
- For our example,
  - CPI_{ideal} of 2,
  - 100 cycle miss penalty (to main memory),
  - 36% load/stores,
  - a 2% (4%) L1I$ (D$) miss rate,
  - **add a UL2$ that has a 25 cycle miss penalty and a 0.5% miss rate**

$$CPI_{stalls} = 2 + .02 \times \mathbf{25} + .36 \times .04 \times \mathbf{25} + \mathbf{.005 \times 100} + \mathbf{.36 \times .005 \times 100} = 3.54$$

(as compared to **5.44** with no L2$)

Multilevel Cache Design Considerations

- Design considerations for L1 and L2 caches are very different
  - Primary cache should focus on **minimizing hit time** in support of a shorter clock cycle
  - Secondary cache should focus on **reducing miss rate** to reduce the penalty of long main memory access times
- The miss penalty of the L1 cache is significantly reduced by the presence of an L2 cache – so it can be smaller (i.e., faster) but have a higher miss rate
- For the L2 cache, hit time is less important than miss rate
  - The L2$ hit time determines L1$'s miss penalty

Key Cache Design Parameters

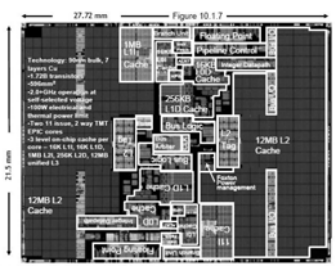
	L1 typical	L2 typical
Total size (blocks)	250 to 2000	4000 to 250,000
Total size (KB)	16 to 64	500 to 8000
Block size (B)	32 to 64	32 to 128
Miss penalty (clocks)	10 to 25	100 to 1000
Miss rates	2% to 5%	0.1% to 2%

Two Machines' Cache Parameters

	Intel P4	**AMD Opteron**
L1 organization	Split I$ and D$	Split I$ and D$
L1 cache size	8KB for D$, 96KB for trace cache (~I$)	64KB for each of I$ and D$
L1 block size	64 bytes	64 bytes
L1 associativity	4-way set assoc.	2-way set assoc.
L1 replacement	~ LRU	LRU
L1 write policy	write-through	write-back
L2 organization	Unified	Unified
L2 cache size	512KB	1024KB (1MB)
L2 block size	128 bytes	64 bytes
L2 associativity	8-way set assoc.	16-way set assoc.
L2 replacement	~LRU	~LRU
L2 write policy	write-back	write-back

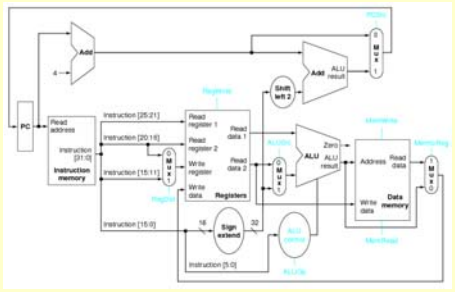
先端マイクロプロセッサ **Intel Montecito**

- 2個のEPICプロセッサコア
- 1MB L2, 12MB L3キャッシュ
- EPICコアは11 issue, 2way Temporal MT
- 初の10億超トランジスタ
  - 1.72BTrs
  - 21.5mm x 27.7mm
  - 90nm
  - 100W
- パワー制御用の専用チップFoxtonを搭載



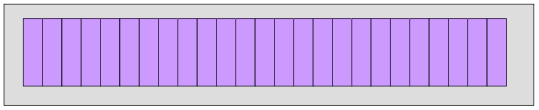
Source: ISSCC 2005 papers

プロセッサのデータパス（シングル・サイクル）



25

Cache



本棚

机

26

OPT: Optimal Replacement Policy

The Optimal Replacement Policy

- Replacement Candidates : On a miss any replacement policy could either choose to replace any of the lines in the cache or choose not to place the miss causing line in the cache at all.
- Self Replacement : The latter choice is referred to as a self-replacement or a cache bypass

Optimal Replacement Policy

On a miss replace the candidate to which an access is least imminent [Belady1966,Mattson1970,McFarling-thesis]

- Lookahead Window : Window of accesses between miss causing access and the access to the least imminent replacement candidate. Single pass simulation of OPT make use of lookahead windows to identify replacement candidates and modify current cache state [Sugumar-SIGMETRICS1993]

OPT: あまり切迫していないものを置き換える.

Optimal Replacement Policy の例

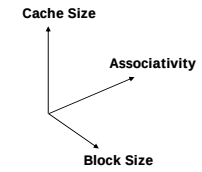
Understanding OPT

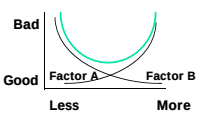
Access Sequence	A_5	A_1	A_6	A_3	A_1	A_4	A_5	A_2	A_5	A_7	A_6	A_8
OPT order for A_5		0		1		2	3	4				
OPT order for A_6				0	1	2	3				4	

- Consider 4 way associative cache with one set initially containing lines (A_1, A_2, A_3, A_4), consider the access stream shown in table
- Access A_5 misses, replacement decision proceeds as follows
  1. Identify replacement candidates : $(A_1, A_2, A_3, A_4, A_5)$
  2. Lookahead and gather imminence order : shown in table, lookahead window circled
  3. Make replacement decision : A_5 replaces A_2
- A_6 self-replaces, lookahead window and imminence order in table

The Cache Design Space

- **Several interacting dimensions**
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation
- **The optimal choice is a compromise**
  - depends on access characteristics
    - workload
    - I-cache, D-cache
  - depends on technology / cost
- **Simplicity** often wins





29

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.

30