CS61C – Machine Structures

Lecture 32 - Caches II

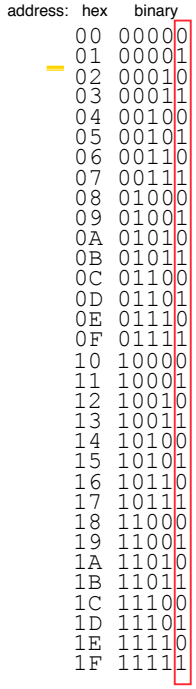
4/12/2006

John Wawrzynek

(www.cs.berkeley.edu/~johnw)

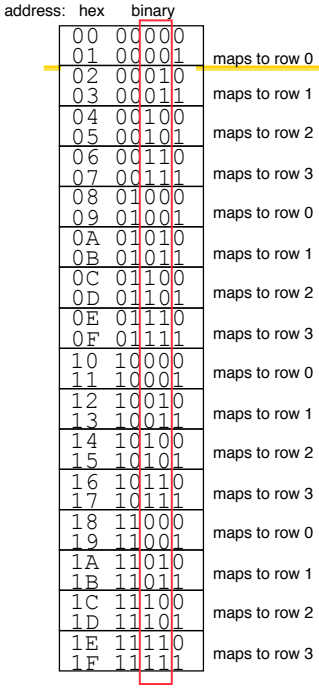
www-inst.eecs.berkeley.edu/~cs61c/

DM Cache Review



Assume: 32 Byte Main Memory (addresses shown to left, memory contents not shown)

2 Bytes/block ⇒ 1 bit for “offset”

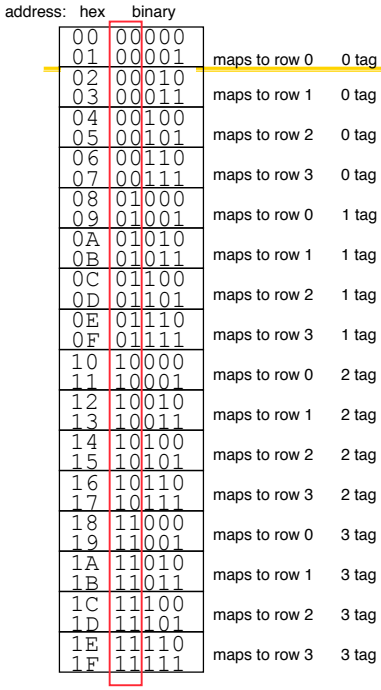


DM Cache Review

Assume: 32 Byte Main Memory
(addresses shown to left)

2 Bytes/block ⇒ 1 bit for “offset”

4 blocks in cache ⇒ 2 bits for “index”
Slots in cache called: row 0, 1, 2, & 3



DM Cache Review

Assume: 32 Byte Main Memory
(addresses shown to left)

2 Bytes/block ⇒ 1 bit for “offset”

4 blocks in cache ⇒ 2 bits for “index”
Slots in cache called: row 0, 1, 2, & 3

5 - 1 - 3 ⇒ 2 bits for “tag”

Accessing data in a direct mapped cache

- Ex.: 16KB of data, direct-mapped, 4 word blocks
- Read 4 addresses
  1. 0x00000014
  2. 0x0000001C
  3. 0x00000034
  4. 0x00008014
- Memory values on right:

Memory

Address (hex)	Value of Word
...	...
00000010	a
00000014	b
00000018	c
0000001C	d
...	...
00000030	e
00000034	f
00000038	g
0000003C	h
...	...
00008010	i
00008014	j
00008018	k
0000801C	l
...	...

Do an example yourself. What happens?

- Chose from: Cache: Hit, Miss, Miss w. replace
  Values returned: a ,b, c, d, e, ..., k, l
- Read address 0x00000030 ?
  0000000000000000 0000000011 0000
- Read address 0x0000001c ?
  0000000000000000 0000000001 1100

Cache

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	2	i	j	k	l
2	0					
3	1	0	e	f	g	h
4	0					
5	0					
6	0					
7	0					
...				...		

Answers

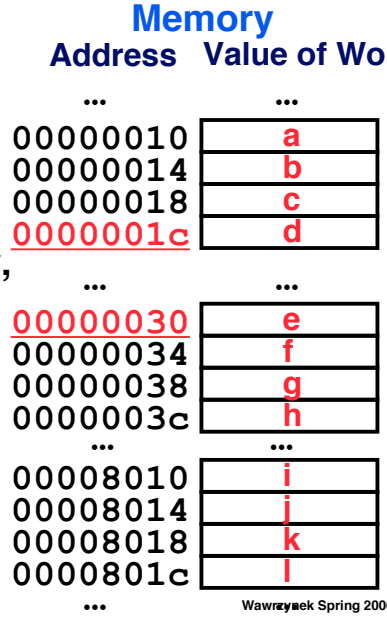
- 0x00000030 a hit

  Index = 3, Tag matches, Offset = 0, value = e

- 0x0000001c a miss

  Index = 1, Tag mismatch, so replace from memory, Offset = 0xc, value = d

- Since reads, values must = memory values whether or not cached:
  - 0x00000030 = e
  - 0x0000001c = d



Block Size Tradeoff (1/3)

- Benefits of Larger Block Size
  - Spatial Locality: if we access a given word, we're likely to access other nearby words soon
  - Very applicable with Stored-Program Concept: if we execute a given instruction, it's likely that we'll execute the next few as well
  - Works nicely in sequential access to arrays and structs too

Block Size Tradeoff (2/3)

- Drawbacks of Larger Block Size
  - Larger block size means larger miss penalty
    - on a miss, takes longer time to load a new block from next level
  - If block size is too big relative to cache size, then there are too few blocks
    - Result: miss rate goes up
- In general, try to minimize

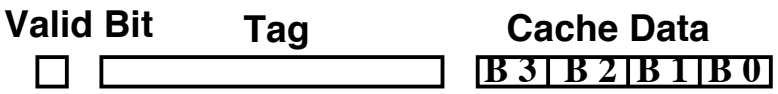
  Average Memory Access Time (AMAT)

  = Hit Time + Miss Penalty x Miss Rate

Block Size Tradeoff (3/3)

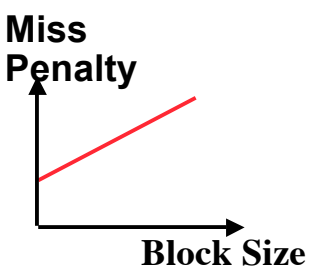
- Hit Time = time to find and retrieve data from current level cache
- Miss Penalty = average time to retrieve data on a current level miss (includes the possibility of misses on successive levels of memory hierarchy)
- Hit Rate = % of requests that are found in current level cache
- Miss Rate = 1 - Hit Rate

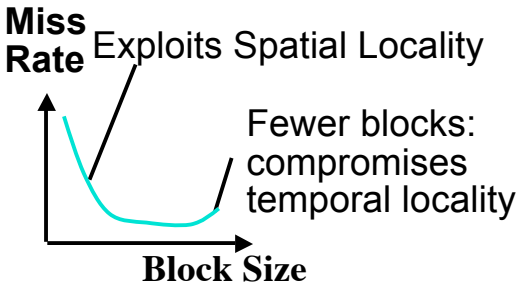
Extreme Example: One Big Block

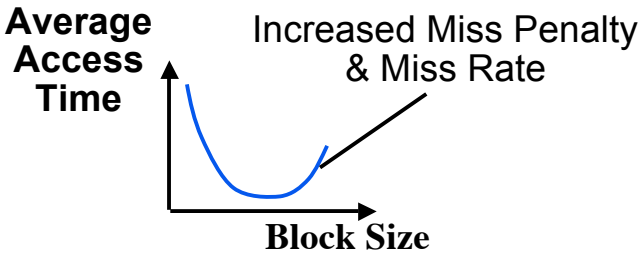


- Cache Size = 4 bytes Block Size = 4 bytes
  - Only ONE entry in the cache!
- If item accessed, likely accessed again soon
  - But unlikely will be accessed again immediately!
- The next access will likely to be a miss again
  - Continually loading data into the cache but discard data (force out) before use it again
  - Nightmare for cache designer: Ping Pong Effect

Block Size Tradeoff Conclusions







Administrivia

- Regraded Midterms handout after class.
- Exam 2 next Wednesday
  - 7-9pm, Pimentel.
  - Covers through pipelining (last week).
  - Special review session Monday evening.

Types of Cache Misses (1/2)

- "Three Cs" Model of Misses
- 1st C: Compulsory Misses
  - occur when a program is first started
  - cache does not contain any of that program's data yet, so misses are bound to occur
  - can't be avoided easily, so won't focus on these in this course

Types of Cache Misses (2/2)

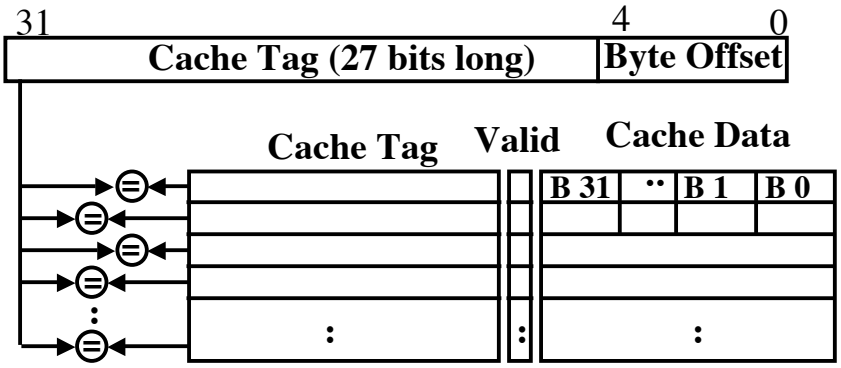
- 2nd C: Conflict Misses
  - miss that occurs because two distinct memory addresses map to the same cache location
  - two blocks (which happen to map to the same location) can keep overwriting each other
  - big problem in direct-mapped caches
  - how do we lessen the effect of these?
- Dealing with Conflict Misses
  - Solution 1: Make the cache size bigger
    - Fails at some point
  - Solution 2: Multiple distinct blocks can fit in the same cache Index?

Fully Associative Cache (1/3)

- Memory address fields:
  - Tag: same as before
  - Offset: same as before
  - Index: non-existant
- What does this mean?
  - no “rows”: any block can go anywhere in the cache
  - must compare with all tags in entire cache to see if data is there

Fully Associative Cache (2/3)

° Fully Associative Cache (e.g., 32 B block)
  • compare tags in parallel



Fully Associative Cache (3/3)

° Benefit of Fully Assoc Cache
  • No Conflict Misses (since data can go anywhere)

° Drawbacks of Fully Assoc Cache
  • Need hardware comparator for every single entry: if we have a 64KB of data in cache with 4B entries, we need 16K comparators: very expensive!

Third Type of Cache Miss

- **Capacity Misses**
  - miss that occurs because the cache has a limited size
  - miss that would not occur if we increase the size of the cache
  - sketchy definition, so just get the general idea
- This is the primary type of miss for Fully Associative caches.

N-Way Set Associative Cache (1/4)

- Memory address fields:
  - Tag: same as before
  - Offset: same as before
  - Index: points us to the correct "row" (called a set in this case)
- So what's the difference?
  - each set contains multiple blocks
  - once we've found correct set, must compare with all tags in that set to find our data

N-Way Set Associative Cache (2/4)

- Summary:
  - cache is direct-mapped w/respect to sets
  - each set is fully associative
  - basically N direct-mapped caches working in parallel: each has its own valid bit and data

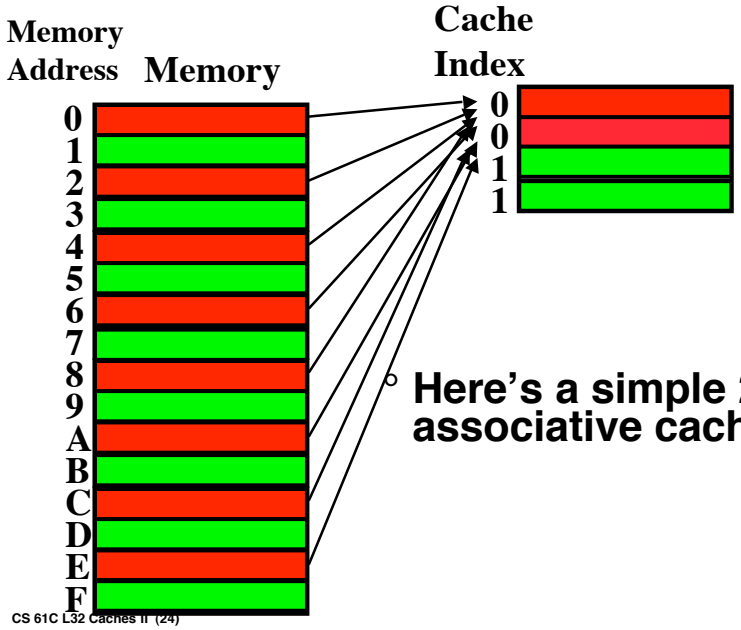
N-Way Set Associative Cache (3/4)

- Given memory address:
  - Find correct set using Index value.
  - Compare Tag with all Tag values in the determined set.
  - If a match occurs, hit!, otherwise a miss.
  - Finally, use the offset field as usual to find the desired data within the block.

N-Way Set Associative Cache (4/4)

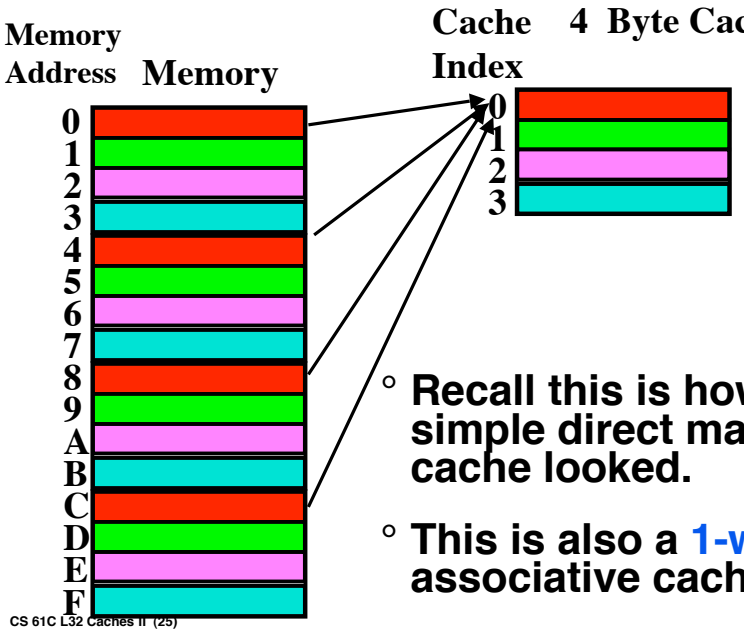
- What's so great about this?
  - even a 2-way set assoc cache avoids a lot of conflict misses
  - hardware cost isn't that bad: only need N comparators
- In fact, for a cache with M blocks,
  - it's Direct-Mapped if it's 1-way set assoc
  - it's Fully Assoc if it's M-way set assoc
  - so these two are just special cases of the more general set associative design

Associative Cache Example



- Here's a simple 2 way set associative cache.

Set Associative Cache Example



- Recall this is how a simple direct mapped cache looked.
- This is also a 1-way set-associative cache!

Cache Things to Remember

- Caches are NOT mandatory:
  - Processor performs arithmetic
  - Memory stores data
  - Caches simply make data transfers go *faster*
- Each level of Memory Hiererarchy subset of next higher level
- Caches speed up due to temporal locality: store data used recently
- Block size > 1 wd spatial locality speedup: Store words next to the ones used recently
- Cache design choices:
  - size of cache: speed v. capacity
  - N-way set assoc: choice of N (direct-mapped, fully-associative just special cases for N)