inst.eecs.berkeley.edu/~cs61c
UCB CS61C : Machine Structures

Lecture 14 – Caches III

Asst. Proflecturer
SOE Miki Garcia

WHEN FIBER OPTICS IS TOO SLOW

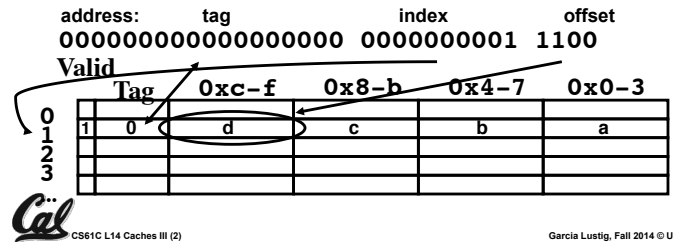07/16/2014: Wall Street Buys NATO Microwave Towers in Quest for Speed.

An 800-foot microwave tower in a Belgian cow pasture transmitted messages for the U.S. armed forces…. Now it's being used by high-frequency traders….. Microwave use has more recently moved to Europe, where trading firms are using former military towers to shoot signals at light speed from bourses in Frankfurt to exchanges in London….. **Fiber can be slowed by obstacles… light travels more slowly through cable than it does through the atmosphere!!!!**

http://www.bloomberg.com/news/2014-07-15/wall-street-grabs-nato-towers-in-traders-speed-of-light-quest.html

---

Review

- Mechanism for transparent movement of data among levels of a storage hierarchy
  - set of address/value bindings
  - address $\Rightarrow$ index to set of candidates
  - compare desired address with tag
  - service hit or miss
    - load new block and binding on miss

address:

tag	index	offset
00000000000000000	0000000001	1100

Valid

	Valid	Tag	0xc-f	0x8-b	0x4-7	0x0-3
0	1	0	d	c	b	a
1						
2						
3						

What to do on a write hit?

- Write-through
  - Update both cache and memory
- Write-back
  - update word in cache block
  - allow memory word to be "stale"
  - $\Rightarrow$ add 'dirty' bit to block
    - memory & Chache inconsistent
    - needs to be updated when block is replaced
  - $\Rightarrow$ OS flushes cache before I/O…
- Performance trade-offs?

Block Size Tradeoff

- 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
  - Works well for sequential array accesses
- 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

Extreme Example: One Big Block

Valid Bit	Tag	Cache Data
☐		B 3	B 2	B 1	B 0

- Cache Size = 4 bytes   Block Size = 4 bytes
  - Only ONE entry (row) 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
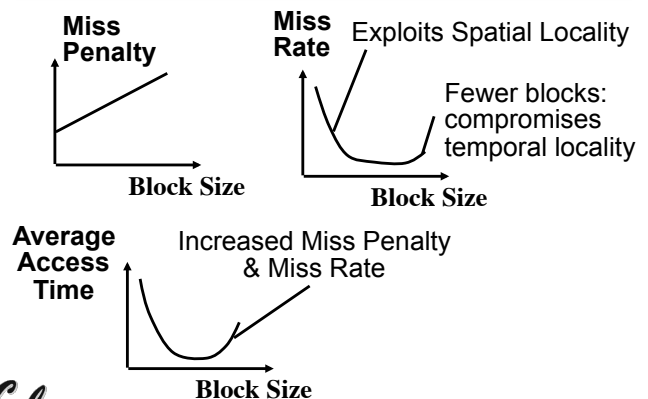
Miss Penalty vs **Block Size**

Miss Rate vs **Block Size** — Exploits Spatial Locality — Fewer blocks: compromises temporal locality

Average Access Time vs **Block Size** — Increased Miss Penalty & Miss Rate

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?

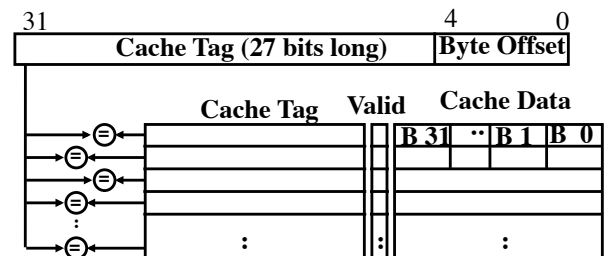## Fully Associative Cache (1/3)

- 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: infeasible

Final Type of Cache Miss

- 3rd C: 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 (1/3)

- 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
  - Size of $ is #of sets x N blocks x block size

Associative Cache Example
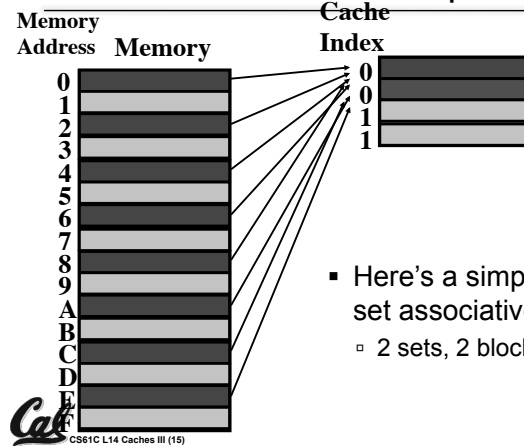
- Here's a simple 2-way set associative cache.
  - 2 sets, 2 blocks in set

N-Way Set Associative Cache (2/3)

- Basic Idea
  - cache is direct-mapped w/respect to sets
  - each set is fully associative with N blocks in it
- 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 (3/3)

- 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
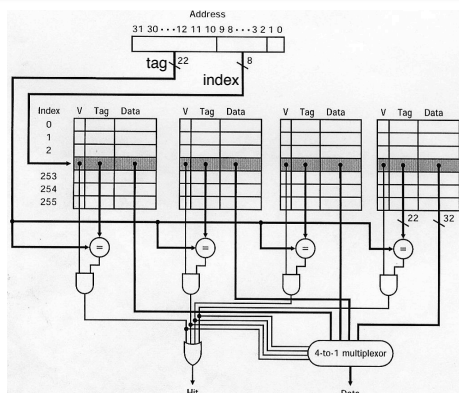 - so these two are just special cases of the more general set associative design

4-Way Set Associative Cache Circuit

Block Replacement Policy

- Direct-Mapped Cache
  - index completely specifies position which position a block can go in on a miss
- N-Way Set Assoc
  - index specifies a set, but block can occupy any position within the set on a miss
- Fully Associative
  - block can be written into any position
- Question: if we have the choice, where should we write an incoming block?
  - If there's a valid bit off, write new block into first invalid.
  - If all are valid, pick a replacement policy
    - rule for which block gets "cached out" on a miss.

Block Replacement Policy: LRU

- LRU (Least Recently Used)
  - Idea: cache out block which has been accessed (read or write) least recently
  - Pro: temporal locality ⇒ recent past use implies likely future use: in fact, this is a very effective policy
  - Con: with 2-way set assoc, easy to keep track (one LRU bit); with 4-way or greater, requires complicated hardware and much time to keep track of this

Block Replacement Example

- We have a 2-way set associative cache with a four word total capacity and one word blocks. We perform the following word accesses (ignore bytes for this problem):

    0, 2, 0, 1, 4, 0, 2, 3, 5, 4

- How many hits and how many misses will there be for the LRU block replacement policy?

Block Replacement Example: LRU

loc 0   loc 1

0: miss, bring into set 0 (loc 0)

| set 0 | lru | |
| set 1 | | |

2: miss, bring into set 0 (loc 1)

| set 0 | lru 0 | 2 |
| set 1 | | |

0: hit

| set 0 | 0 | lru 2 |
| set 1 | | |

1: miss, bring into set 1 (loc 0)

| set 0 | 0 | lru 2 |
| set 1 | 1 | lru |

4: miss, bring into set 0

| set 0 | lru 0 | |
| set 1 | 1 | lru |

Addresses 0, 2, 0, 1, 4, 0, ...   **0: hit**

| set 0 | 0 | lru 4 |
| set 1 | 1 | lru |

Big Idea

- How to choose between associativity, block size, replacement & write policy?
- Design against a performance model
  - Minimize: Average Memory Access Time
    = Hit Time
      + Miss Penalty x Miss Rate
  - influenced by technology & program behavior
- Create the illusion of a memory that is large, cheap, and fast - on average
- How can we improve miss penalty?

Improving Miss Penalty

- When caches first became popular, Miss Penalty ~ 10 processor clock cycles
- Today 2400 MHz Processor (0.4 ns per clock cycle) and 80 ns to go to DRAM
  ⇒ 200 processor clock cycles!

Proc ↔ MEM [$ | $2 | DRAM]
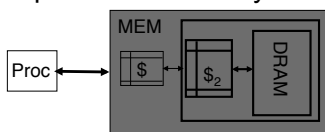
Solution: another cache between memory and the processor cache: Second Level (L2) Cache

Peer Instruction

1. A 2-way set-associative cache can be outperformed by a direct-mapped cache.
2. Larger block size ⇒ lower miss rate

	12
a)	FF
b)	FT
c)	TF
d)	TT

And in Conclusion…

- We've discussed memory caching in detail. Caching in general shows up over and over in computer systems
  - Filesystem cache, Web page cache, Game databases / tablebases, Software memoization, Others?
- Big idea: if something is expensive but we want to do it repeatedly, do it once and cache the result.
- Cache design choices:
  - Size of cache: speed v. capacity
  - Block size (i.e., cache aspect ratio)
  - Write Policy (Write through v. write back
  - Associativity choice of N (direct-mapped v. set v. fully associative)
  - Block replacement policy
  - 2nd level cache?
  - 3rd level cache?
- Use performance model to pick between choices, depending on programs, technology, budget, ...

Bonus slides

- These are extra slides that used to be included in lecture notes, but have been moved to this, the "bonus" area to serve as a supplement.
- The slides will appear in the order they would have in the normal presentation
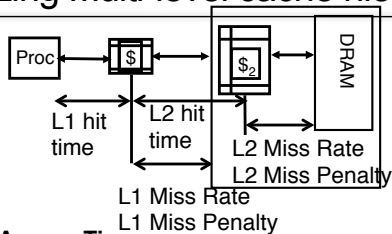
Bonus

Analyzing Multi-level cache hierarchy

Avg Mem Access Time =
L1 Hit Time + L1 Miss Rate * L1 Miss Penalty

L1 Miss Penalty =
L2 Hit Time + L2 Miss Rate * L2 Miss Penalty

Avg Mem Access Time =
L1 Hit Time + L1 Miss Rate *
(L2 Hit Time + L2 Miss Rate * L2 Miss Penalty)

Example

- Assume
  - Hit Time = 1 cycle
  - Miss rate = 5%
  - Miss penalty = 20 cycles
  - Calculate AMAT…
- Avg mem access time
  - = 1 + 0.05 x 20
  - = 1 + 1 cycles
  - = 2 cycles

Ways to reduce miss rate

- Larger cache
  - limited by cost and technology
  - hit time of first level cache < cycle time (bigger caches are slower)
- More places in the cache to put each block of memory – associativity
  - fully-associative
    - any block any line
  - N-way set associated
    - N places for each block
    - direct map: N=1

Typical Scale

- L1
  - size: tens of KB
  - hit time: complete in one clock cycle
  - miss rates: 1-5%
- L2:
  - size: hundreds of KB
  - hit time: few clock cycles
  - miss rates: 10-20%
- L2 miss rate is fraction of L1 misses that also miss in L2
  - why so high?
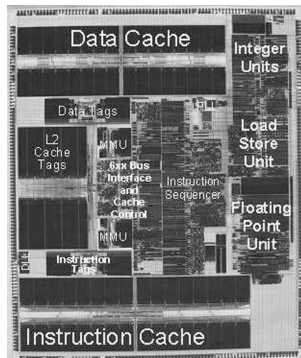
Example: with L2 cache

- Assume
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L2 Hit Time = 5 cycles
  - L2 Miss rate = 15% (% L1 misses that miss)
  - L2 Miss Penalty = 200 cycles
- L1 miss penalty = 5 + 0.15 * 200 = 35
- Avg mem access time = 1 + 0.05 x 35
                                = 2.75 cycles

Example: without L2 cache

- Assume
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L1 Miss Penalty = 200 cycles
- Avg mem access time = 1 + 0.05 x 200
                                = 11 cycles

- 4x faster with L2 cache! (2.75 vs. 11)

An actual CPU – Early PowerPC

- Cache
  - 32 KB Instructions and 32 KB Data L1 caches
  - External L2 Cache interface with integrated controller and cache tags, supports up to 1 MByte external L2 cache
  - Dual Memory Management Units (MMU) with Translation Lookaside Buffers (TLB)
- Pipelining
  - Superscalar (3 inst/cycle)
  - 6 execution units (2 integer and 1 double precision IEEE floating point)

An Actual CPU – Pentium M