

Lecture 32 – Caches III



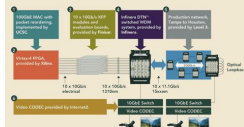
2006-11-15

Lecturer SOE Dan Garcia

www.cs.berkeley.edu/~ddgarcia

100 Gig Eth?! =>

Yesterday Infinera demoed a system that took a 100 GigE signal, broke it up into ten 10GigE signals and sent it over existing optical networks. Fast!!!



What to do on a write hit?

- **Write-through**
 - update the word in cache block and corresponding word in memory
- **Write-back**
 - update word in cache block
 - allow memory word to be “stale”
 - ⇒ add ‘dirty’ bit to each block indicating that memory needs to be updated when block is replaced
 - ⇒ OS flushes cache before I/O...
- **Performance trade-offs?**

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 array accesses 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, minimize **Average Memory Access Time (AMAT)**
 - = Hit Time + Miss Penalty x Miss Rate

Block Size Tradeoff (3/3)

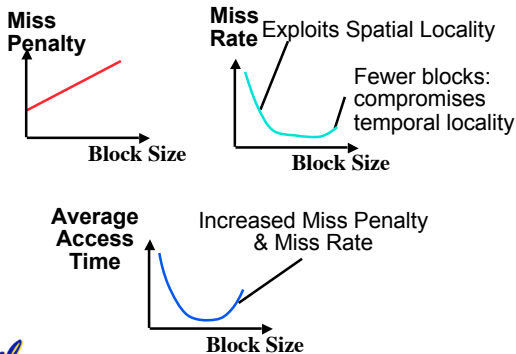
- **Hit Time** = time to find and retrieve data from current level cache
- **Miss Penalty** = average time to retrieve data on a current level miss (includes the possibility of misses on successive levels of memory hierarchy)
- **Hit Rate** = % of requests that are found in current level cache
- **Miss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

Extreme Example: One Big Block

Valid Bit	Tag	Cache Data
□	[]	[B3 B2 B1 B0]

- 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



CS61C L32 Caches III (7)

Garcia, Fall 2006 © UCB

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



CS61C L32 Caches III (8)

Garcia, Fall 2006 © UCB

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?



CS61C L32 Caches III (9)

Garcia, Fall 2006 © UCB

Fully Associative Cache (1/3)

• Memory address fields:

- Tag: same as before
- Offset: same as before
- Index: non-existent

• 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



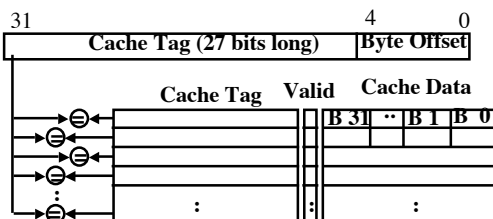
CS61C L32 Caches III (10)

Garcia, Fall 2006 © UCB

Fully Associative Cache (2/3)

• Fully Associative Cache (e.g., 32 B block)

- compare tags in parallel



CS61C L32 Caches III (11)

Garcia, Fall 2006 © UCB

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



CS61C L32 Caches III (12)

Garcia, Fall 2006 © UCB

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.**



CS61C L32 Caches III (13)

Garcia, Fall 2006 © UCB

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?**

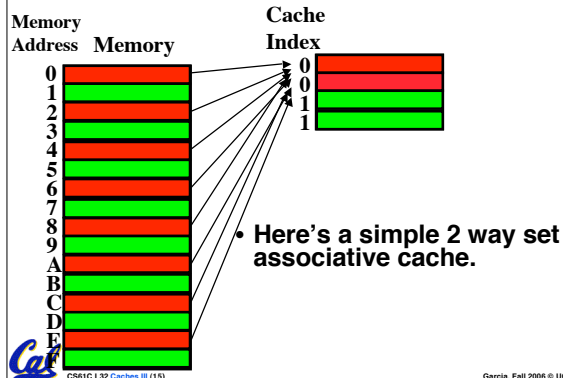
- each set contains multiple blocks
- once we’ve found correct set, must compare with all tags in that set to find our data



CS61C L32 Caches III (14)

Garcia, Fall 2006 © UCB

Associative Cache Example



CS61C L32 Caches III (15)

Garcia, Fall 2006 © UCB

N-Way Set Associative Cache (2/3)

- **Basic Idea**

- 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

- **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.



CS61C L32 Caches III (16)

Garcia, Fall 2006 © UCB

N-Way Set Associative Cache (3/3)

- **What’s so great about this?**

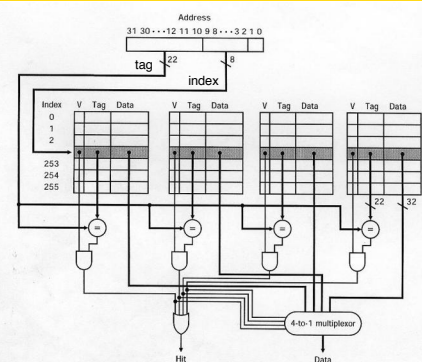
- even a 2-way set assoc cache avoids a lot of conflict misses
- hardware cost isn’t that bad: only need N comparators
- In fact, for a cache with M blocks,
 - it’s Direct-Mapped if it’s 1-way set assoc
 - it’s Fully Assoc if it’s M-way set assoc
 - so these two are just special cases of the more general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



CS61C L32 Caches III (17)

Garcia, Fall 2006 © UCB

4-Way Set Associative Cache Circuit



Fall 2006 © UCB

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 are any locations with valid bit off (empty), then usually write the new block into the first one.
 - If all possible locations already have a valid block, we must pick a **replacement policy:** rule by which we determine which block gets “cached out” on a miss.



CS61C L32 Caches III (19)

Garcia, Fall 2006 © UCB

Block Replacement Policy: LRU

• LRU (Least Recently Used)

- Idea: cache out block which has been accessed (read or write) least recently
- Pro: **temporal locality** \Rightarrow 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



CS61C L32 Caches III (20)

Garcia, Fall 2006 © UCB

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?



CS61C L32 Caches III (21)

Garcia, Fall 2006 © UCB

Block Replacement Example: LRU

- Addresses 0, 2, 0, 1, 4, 0, ...

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

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

0: hit

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

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

0: hit

	loc 0	loc 1
set 0	0 (ru)	
set 1		
set 0	0 (ru)	2
set 1		
set 0	0 (ru)	2
set 1		
set 0	0 (ru)	2
set 1	1 (ru)	
set 0	0 (ru)	4
set 1	1 (ru)	
set 0	0 (ru)	4
set 1	1 (ru)	



CS61C L32 Caches III (22)

Garcia, Fall 2006 © UCB

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



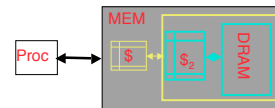
How can we improve miss penalty?

CS61C L32 Caches III (23)

Garcia, Fall 2006 © UCB

Improving Miss Penalty

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



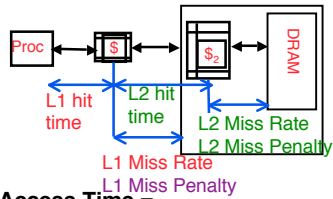
Solution: another cache between memory and the processor cache: **Second Level (L2) Cache**



CS61C L32 Caches III (24)

Garcia, Fall 2006 © UCB

Analyzing Multi-level cache hierarchy



Avg Mem Access Time =

$$\text{L1 Hit Time} + \text{L1 Miss Rate} * \text{L1 Miss Penalty}$$

L1 Miss Penalty =

$$\text{L2 Hit Time} + \text{L2 Miss Rate} * \text{L2 Miss Penalty}$$

Avg Mem Access Time =

$$\text{L1 Hit Time} + \text{L1 Miss Rate} * (\text{L2 Hit Time} + \text{L2 Miss Rate} * \text{L2 Miss Penalty})$$



CS61C L32 Caches III (23)

Garcia, Fall 2006 © UCB

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:
 - Write through v. write back
 - size of cache: speed v. capacity
 - direct-mapped v. associative
 - for N-way set assoc: choice of N
 - block replacement policy
 - 2nd level cache?
 - 3rd level cache?



CS61C L32 Caches III (28)

Garcia, Fall 2006 © UCB

BONUS SLIDES



CS61C L32 Caches III (27)

Garcia, Fall 2006 © UCB

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



CS61C L32 Caches III (28)

Garcia, Fall 2006 © UCB

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



CS61C L32 Caches III (29)

Garcia, Fall 2006 © UCB

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?



CS61C L32 Caches III (30)

Garcia, Fall 2006 © UCB

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 = $5 + 0.15 \times 200 = 35$
- Avg mem access time = $1 + 0.05 \times 35 = 2.75$ cycles



CS61C L32 Caches III (31)

Garcia, Fall 2006 © UCB

Example: without L2 cache

- Assume
 - L1 Hit Time = 1 cycle
 - L1 Miss rate = 5%
 - L1 Miss Penalty = 200 cycles
- Avg mem access time = $1 + 0.05 \times 200 = 11$ cycles
- 4x faster with L2 cache! (2.75 vs. 11)

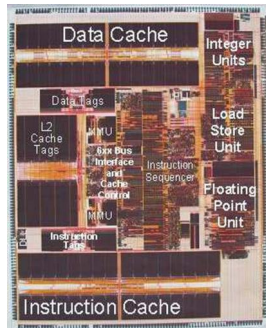


CS61C L32 Caches III (32)

Garcia, Fall 2006 © UCB

An actual CPU – Early PowerPC

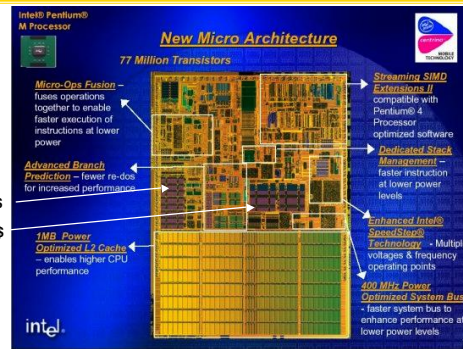
- Cache
 - 32 KByte Instructions and 32 KByte Data L1 caches
 - External L2 Cache interface with integrated controller and cache tags, supports up to 1 MByte external L2 cache
 - Dual Memory Management Units (MMU) with Translation Lookaside Buffers (TLB)
- Pipelining
 - Superscalar (3 Inst/cycle)
 -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



CS61C L32 Caches III (33)

Garcia, Fall 2006 © UCB

An Actual CPU – Pentium M



CS61C L32 Caches III (34)

Garcia, Fall 2006 © UCB