

CS61C – Machine Structures

Lecture 33 - Caches III

4/14/2006

John Wawrzynek

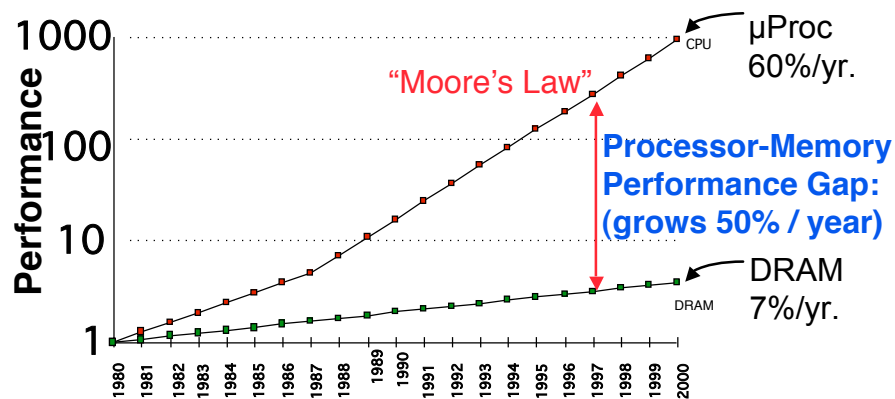
(www.cs.berkeley.edu/~johnw)

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

CS 61C L33 Caches III (1)

Wawrzynek Spring 2006 © UCB

Review: Why We Use Caches



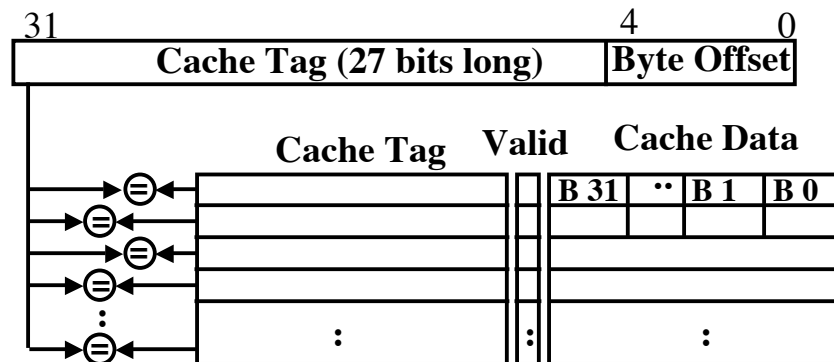
- 1989 first Intel CPU with cache on chip
- 1998 Pentium III has two levels of cache on chip

CS 61C L33 Caches III (2)

Wawrzynek Spring 2006 © UCB

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!
 - Alternatively, use fewer comparisons, but compare sequentially - too slow!

Third Type of Cache Miss

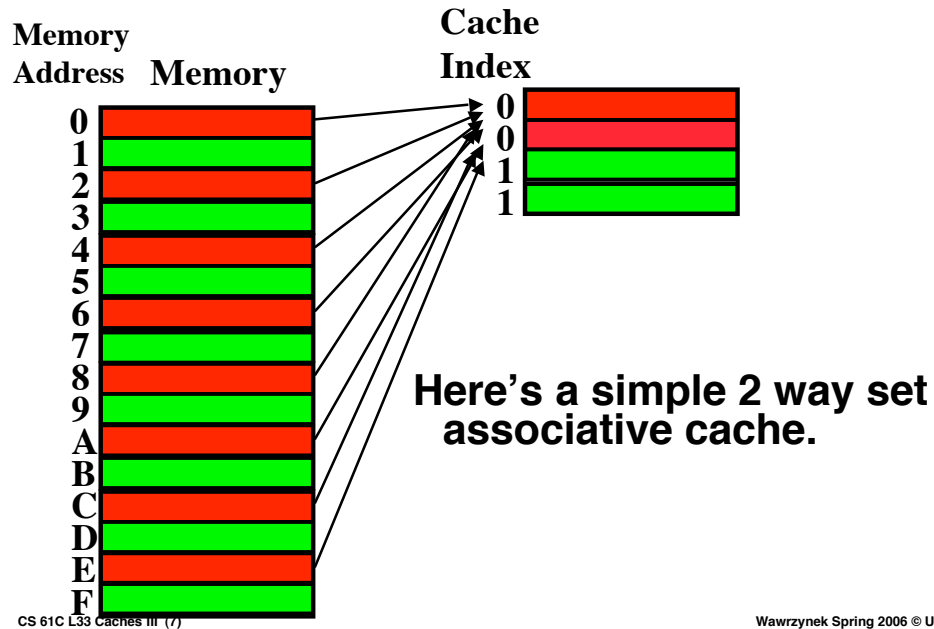
◦ Capacity Misses

- 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

Set Associative Cache Example



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)

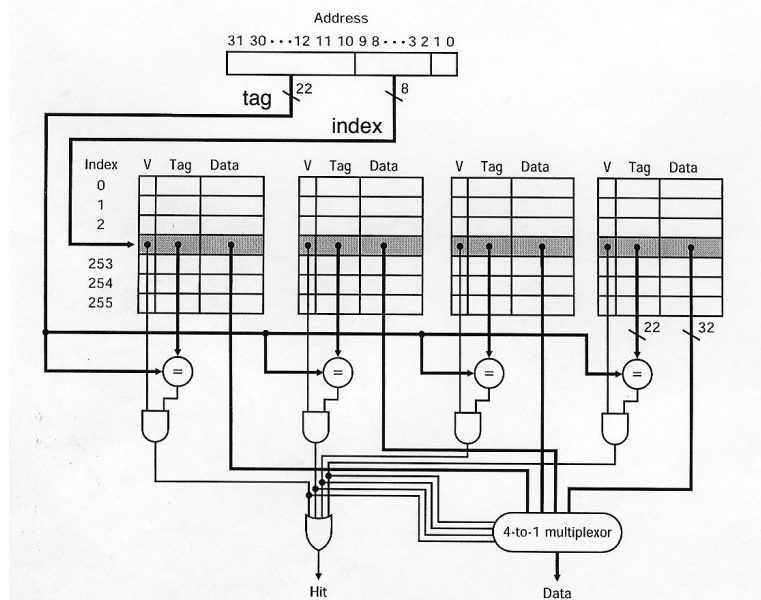
◦ What's so great about this?

- 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

4-Way Set Associative Cache Circuit



CS 61C L33

ing 2006 © UCB

Block Replacement Policy (2/2)

- 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.

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

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

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	<i>lru</i>
set 1		
set 0	<i>lru</i> 0	2
set 1		
set 0	0	<i>lru</i> 2
set 1		
set 0	0	<i>lru</i> 2
set 1	1	<i>lru</i>
set 0	<i>lru</i> 0	4
set 1	1	<i>lru</i>
set 0	0	<i>lru</i> 4
set 1	1	<i>lru</i>

Big Idea

- How to choose between associativity, block size, replacement policy?
- Design against a performance model
 - Minimize: **Average Memory Access Time**

$$= \text{Hit Time} + \text{Miss Penalty} \times \text{Miss Rate}$$
 - influenced by technology & program behavior
- Create the illusion of a memory that is large, cheap, and fast - on average

Example

- **Assume**
 - Hit Time = 1 cycle
 - Miss rate = 5%
 - Miss penalty = 20 cycles
 - Calculate AMAT...
- **Avg mem access time**
 - = $1 + 0.05 \times 20$
 - = 1 + 1 cycles
 - = 2 cycles

Administrivia

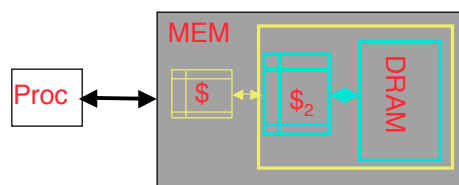
- **Do your reading! VM is coming up, and it's shown to be hard for students!**
- **Project 5 out**
- **Exam**
 - Wed 4/19, 1 Pimentel 7-9pm
 - Covers weeks 6-12 (focus on lecture material)
 - TA Review Monday evening

Ways to reduce miss rate

- **Larger cache**
 - limited by cost and technology
 - hit time of first level cache < cycle time
- **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: $N=1$

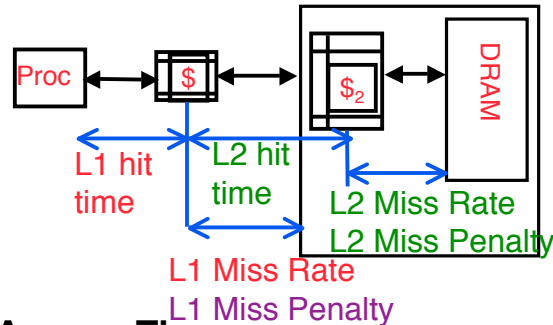
Improving Miss Penalty

- 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!**



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

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 =

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

CS 61C L33 Caches III (21)

Wawrzynek Spring 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?

CS 61C L33 Caches III (22)

Wawrzynek Spring 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 = $5 + 0.15 * 200 = 35$

◦ Avg mem access time = $1 + 0.05 * 35$
= **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 = $1 + 0.05 * 200$
= **11 cycles**

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

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?

Generalized Caching

◦ We’ve discussed memory caching in detail. Caching in general shows up over and over in computer systems

- Filesystem cache
- Web page cache
- Game Theory 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.

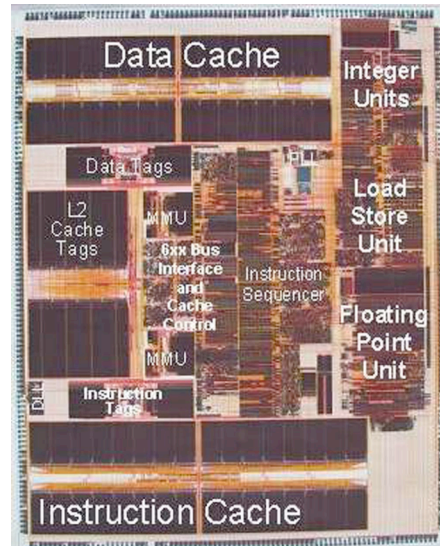
An actual CPU -- Early PowerPC

◦ 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)



CS 61C L33 Caches III (27)

Wawrzynek Spring 2006 © UCB

And in Conclusion...

◦ Cache design choices:

- 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?
- Write through v. write back?

◦ Use performance model to pick between choices, depending on programs, technology, budget, ...

CS 61C L33 Caches III (29)

Wawrzynek Spring 2006 © UCB