

CS61C : Machine Structures

Lecture 6.2.1 Cache II

2004-07-28

Kurt Meinz

inst.eecs.berkeley.edu/~cs61c



CS 61C L6.2.1 Cache II (1)

K. Meinz, Summer 2004 © UCB

Review: Memory Hierarchy Basics

- Programs exhibit “temporal locality” ...
 - If we just accessed it, chances are we’ll access it again soon.
 - Or, more formally,
 - The probability of accessing a particular piece of data varies inversely with the time since we last accessed it.



CS 61C L6.2.1 Cache II (2)

K. Meinz, Summer 2004 © UCB

Review: Memory Hierarchy Basics

- Also “Spatial Locality”:
 - Probability of accessing a certain piece of data X increases as we access pieces located around X.
- We use temporal and spatial locality to approximate the working set of a program.



CS 61C L6.2.1 Cache II (3)

K. Meinz, Summer 2004 © UCB

Review: DM

- 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



CS 61C L6.2.1 Cache II (4)

K. Meinz, Summer 2004 © UCB

Review: Cache Design Decisions

- Direct Mapped is *Very Good*
 - Fast Logic
 - Efficient Hardware
- However, we can test out some changes ...



CS 61C L6.2.1 Cache II (5)

K. Meinz, Summer 2004 © UCB

Review: Cache Design: Five Questions

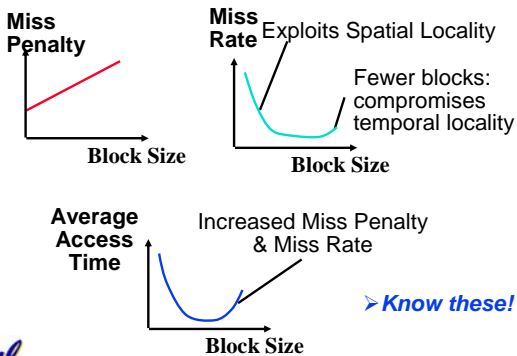
- Q0: How big are blocks?
- Q1: Where can a block be placed in the cache? (*Block placement*)
- Q2: How is a block found if it is in the cache? (*Block identification*)
- Q3: Which block should be replaced on a miss? (*Block replacement*)
- Q4: What happens on a write? (*Write strategy*)



CS 61C L6.2.1 Cache II (6)

K. Meinz, Summer 2004 © UCB

Review: Block Size Tradeoff Conclusions



CS 61C L6.2.1 Cache II (7)

K. Mehta, Summer 2004 © UCB

Outline

- DM Implementation
- Cache Miss Analysis
- Cache Associativity
- Cache Performance Model
- Write Strategies

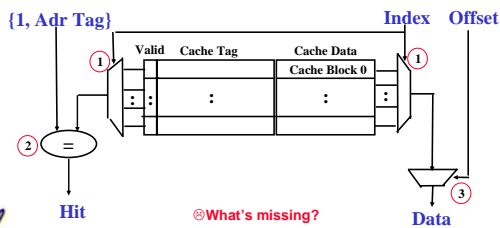


CS 61C L6.2.1 Cache II (8)

K. Mehta, Summer 2004 © UCB

Direct Mapped Cache

- Implementation of DM Cache
 - Index selects block via mux ①
 - Cache block tag and valid bit compared to {1, Requested Tag} ②
 - Data is muxed again by offset. ③



CS 61C L6.2.1 Cache II (9)

K. Mehta, Summer 2004 © UCB

Cache Design: Five Questions

- Q0: How big are blocks?
- • Q1: Where can a block be placed in the cache? (*Block placement*)
- Q2: How is a block found if it is in the cache? (*Block identification*)
- Q3: Which block should be replaced on a miss? (*Block replacement*)
- Q4: What happens on a write? (*Write strategy*)



CS 61C L6.2.1 Cache II (10)

K. Mehta, Summer 2004 © UCB

Analysis 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



CS 61C L6.2.1 Cache II (11)

K. Mehta, Summer 2004 © 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
- 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?!



CS 61C L6.2.1 Cache II (12)

K. Mehta, Summer 2004 © UCB

Fully Associative Cache (1/3)

- Memory address fields:
 - Tag: same as before
 - Offset: same as before
 - Index: non-existent ← !!
- What does this mean?
 - any block can go anywhere in the cache
 - must compare with all tags in entire cache to see if data is there

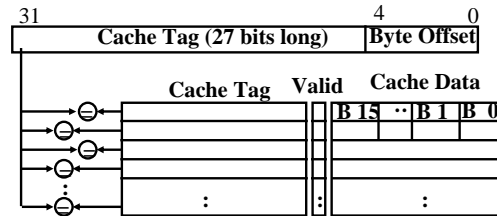


CS 61C L6.2.1 Cache II (13)

K. Meinel, Summer 2004 © UCB

Fully Associative Cache (2/3)

- Fully Associative Cache (e.g., 16 B block)
 - compare tags in parallel



CS 61C L6.2.1 Cache II (14)

K. Meinel, Summer 2004 © 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 ~16,000 comparators: infeasible

➤ So where are FA caches feasible?



CS 61C L6.2.1 Cache II (15)

K. Meinel, Summer 2004 © 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
 - Capacity miss on data X \leftrightarrow If cache has N blocks, and last access to X was > N unique accesses ago.
 - C.f. conflict miss on X \leftrightarrow last access to X was < N unique accesses ago.
- This is the primary type of miss for Fully Associative caches.



CS 61C L6.2.1 Cache II (16)

K. Meinel, Summer 2004 © UCB

Compromise

- Can we compromise between FA and DM?
- Insight: For most programs, it becomes increasingly unlikely to have more than 4 (or so) blocks in the working set map to the same index.
- So, keep indexes from DM, but have a little FA cache in each index ...



CS 61C L6.2.1 Cache II (17)

K. Meinel, Summer 2004 © UCB

N-Way Set Associative Cache (1/4)

- Memory address fields:
 - Tag: same as before
 - Offset: same as before
 - Index: points us to the correct index (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



CS 61C L6.2.1 Cache II (18)

K. Meinel, Summer 2004 © UCB

Cache Design: Five Questions

- Q0: How big are blocks?
- Q1: Where can a block be placed in the cache? (*Block placement*)
- Q2: How is a block found if it is in the cache? (*Block identification*)
- Q3: Which block should be replaced on a miss? (*Block replacement*)
- ▶ • Q4: What happens on a write? (*Write strategy*)



CS 61C L6.2.1 Cache II (25)

K. Meinel, Summer 2004 © UCB

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?



CS 61C L6.2.1 Cache II (26)

K. Meinel, Summer 2004 © UCB

Cache Design: Five Questions

- Q0: How big are blocks?
- Q1: Where can a block be placed in the cache? (*Block placement*)
- Q2: How is a block found if it is in the cache? (*Block identification*)
- ▶ • Q3: Which block should be replaced on a miss? (*Block replacement*)
- Q4: What happens on a write? (*Write strategy*)



CS 61C L6.2.1 Cache II (27)

K. Meinel, Summer 2004 © UCB

Block Replacement Policy (1/2)

- **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?



CS 61C L6.2.1 Cache II (28)

K. Meinel, Summer 2004 © 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.



CS 61C L6.2.1 Cache II (29)

K. Meinel, Summer 2004 © UCB

Block Replacement Policy: LTNA

- **Best replacement scheme**:
 - “Longest Time to Next Access”
 - Kick out the block that won’t be used for the longest time.

☹ What’s wrong with this?



CS 61C L6.2.1 Cache II (30)

K. Meinel, Summer 2004 © 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
 - Usually resort to random replacement



CS 61C L6.2.1 Cache II (31)

K. Mehta, Summer 2004 © 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?



CS 61C L6.2.1 Cache II (32)

K. Mehta, Summer 2004 © 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	lrU
set 1		
set 0	lrU	2
set 1		
set 0	0	lrU 2
set 1		
set 0	0	lrU 2
set 1	1	lrU
set 0	0	4
set 1	1	lrU
set 0	0	lrU 4
set 1	1	lrU



CS 61C L6.2.1 Cache II (33)

K. Mehta, Summer 2004 © UCB

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
 - Note: **Hit Time encompasses Hit Rate!!!**
- Create the illusion of a memory that is large, cheap, and fast - on average



CS 61C L6.2.1 Cache II (34)

K. Mehta, Summer 2004 © UCB