

CSE 141 – Computer Architecture Spring 2005

Lectures 16 Cache

Pramod V. Argade
May 23, 2005

CSE141: Introduction to Computer Architecture

Instructor: Pramod V. Argade (p2argade@cs.ucsd.edu)
Office Hour:
Mon. 5:00 - 6:00 PM (AP&M 5218)

TAs:

Baris Arslan: barslan@cs.ucsd.edu
Chris Roedel: croedel@cs.ucsd.edu
Raid Ayoub: rayoub@cs.ucsd.edu
Leo Porter: leporter@cs.ucsd.edu

Textbook: Computer Organization & Design
The Hardware Software Interface, 3rd Edition.
Authors: Patterson and Hennessy

Web-page: <http://www.cse.ucsd.edu/classes/sp05/cse141>

Announcements

- **Reading Assignment**

Chapter 7: The Basics of Caches

– 7.1, 7.2

- **Homework 6: Due Wednesday May 25th**

– 6.31, 6.32

– 7.2, 7.3, 7.4, 7.5, 7.9, 7.10, 7.12, 7.23

- **Quiz**

When: Wed., May 25th, First 10 minutes of the class

Topic: Cache Chapter 7 **Need:** Paper, pen

Course Schedule

Lecture #	Date	Day	Lecture Topic	Quiz Topic	Homework Due
1	3/28	Monday	Introduction, Ch. 1	-	-
2	3/30	Wednesday	Performance, Ch. 4	-	-
3	4/4	Monday	ISA, Ch. 2	Performance	#1
4	4/6	Wednesday	Arithmetic, Ch. 3	-	-
5	4/11	Monday	Arithmetic, Ch. 3	ISA	#2
6	4/13	Wednesday	Single cycle CPU, Ch. 5	-	-
7	4/18	Monday	Single cycle CPU, Ch. 5	Arithmetic	#3
8	4/20	Wednesday	Multi-cycle CPU, Ch. 5	-	-
9	4/25	Monday	Multi-cycle CPU, Ch. 5	Single Cycle CPU	#4
10	4/27	Wednesday	Review for the Midterm	-	-
	5/2	Monday	Mid-term Exam Center 101	-	-
11	5/4	Wednesday	Exceptions, Ch. 5 and Pipelining, Ch. 6	-	-
12	5/9	Monday	Pipelining, Ch. 6	-	-
13	5/11	Wednesday	Data and control hazards, Ch. 6	-	-
14	5/16	Monday	Data and control hazards, Ch. 6	Pipeline Hazards	#5
15	5/18	Wednesday	Memory & cache design, Ch. 7	-	-
16	5/23	Monday	Memory & cache design, Ch. 7	-	-
17	5/25	Wednesday	Virtual Memory & cache design, Ch. 7	Cache	#6
No Class	5/30	Monday	Memorial Day Holiday	-	-
18	6/1	Wednesday	Course Review	-	-
	6/10	Friday	7 - 10 PM, SOLIS 107: Final Exam	-	-

Dealing with Stores

- Stores must be handled differently than loads, because...
 - They don't necessarily require the CPU to stall
 - stores don't produce register values used by other instructions
 - They change the content of cache/memory (creating memory *consistency* issues)
- Policy decisions for stores
 - write-through => all writes go to both cache and main memory
 - write-back => writes go only to cache. Modified cache lines are written back to memory when the line is replaced.
 - write-allocate => on a store miss, bring written line into the cache
 - write-around => on a store miss, write to main memory, & ignore cache

Handling a Cache Write Miss

- Write-through Cache
 - Write data to cache as well as memory
 - Don't need to consider whether the write hits or misses the cache
 - Disadvantage: Every write causes the data to be written to the main memory
 - Use write buffer so CPU can proceed with the following instructions
- Write-back Cache
 - When write occurs, write the new value only to the block in the cache
 - Write cache data to memory when it is about to be overwritten for another address
 - Improves performance over write-through cache
 - More complex to implement

Summary for Stores

- On a store hit, write the new data to cache. In a *write-through* cache, write the data immediately to memory. In a *write-back* cache, mark the line as dirty.
- On a store miss, initiate a cache block load from memory for a write-allocate cache. Write directly to memory for a write-around cache.
- On any kind of cache miss in a write-back cache, if the line to be replaced in the cache is dirty, write it back to memory.

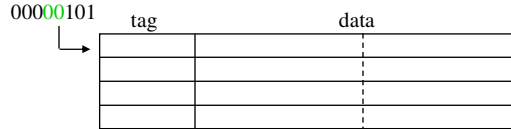
Taking advantage of Spatial Locality

- Consider following address trace:
 - 0,1,2,3,17,8,9,10,11,17,4,5,6,7...
- Notice that addresses lie in the vicinity of each other
- Instructions show high degree of spatial locality
 - Typically accessed sequentially
 - Generally, code consists of a lot of loops
- Data also shows spatial locality
 - Typically less than that of instructions
 - Different elements of a structure may be accessed
- Why not bring multiple words on a cache miss?
 - Instead of bringing a single?

Spatial Locality: Larger Cache Blocks

Byte address string:

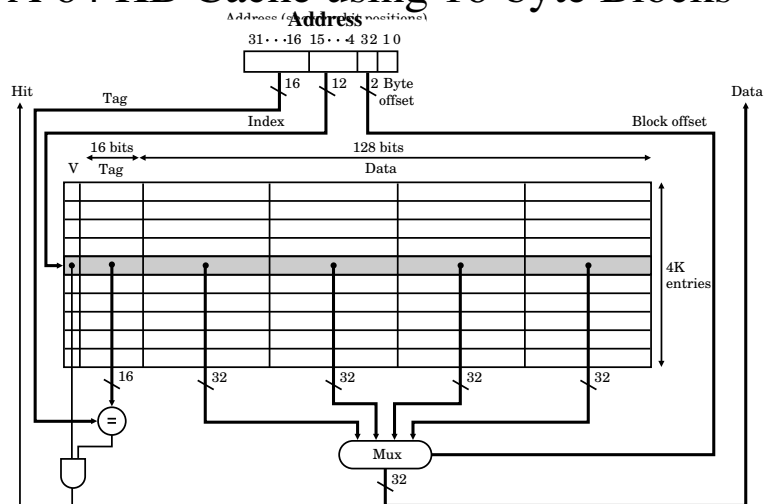
5	00000101
10	00001010
12	00001100
4	00000100
9	00001001
20	00010100
7	00000111
8	00001000
21	00010101
24	00011000
14	00001110
11	00001011
4	00000100



4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of *spatial locality*.
- Too large of a block size can waste cache space.
- Larger cache blocks require less tag space

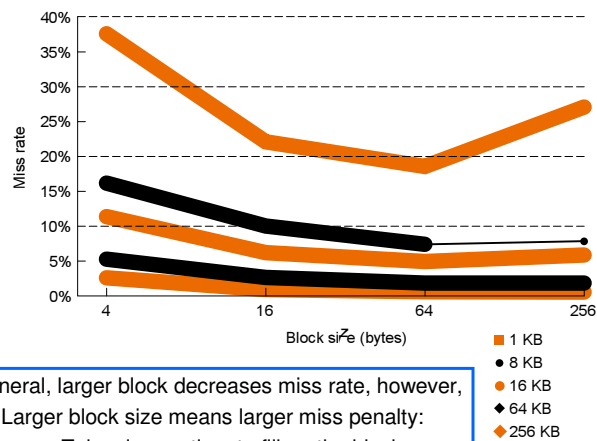
A 64 KB Cache using 16-byte Blocks



Complication with Larger Blocks

- Write-through cache: Can't write to the cache while performing a tag comparison
 - Ok if there is a hit in the cache
 - Not ok if there is a cache miss:
 - The block has to be fetched from memory and placed in the cache
 - Rewrite the word that caused the miss into the cache

Impact of Block Size on Miss Rate



- In general, larger block decreases miss rate, however,
 - Larger block size means larger miss penalty:
 - Takes longer time to fill up the block
 - Miss rates go up if block size is too big
 - Since there are too few cache blocks

Cache Performance

- 64 KB each instruction cache and data cache (direct mapped)

Program	Block Size in words	Instruction miss rate	Data miss rate	Effective Combined miss rate
gcc	1	6.1%	2.1%	5.4%
	4	2.0%	1.7%	1.9%
spice	1	1.2%	1.3%	1.2%
	4	0.3%	0.6%	0.4%

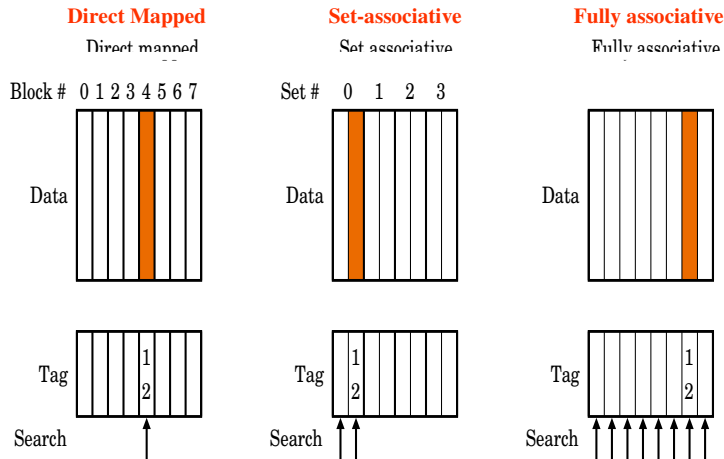
- In general, Average Access Time:
= [Hit Time * (1 - Miss Rate)] + [Miss Penalty * Miss Rate]
- Limitations of direct mapped cache
 - A block can go in exactly one place in the cache

Flexible Placement of Blocks

- Direct mapped cache
 - A block can go in exactly one place in the cache
 - Leads to collision among blocks
- Fully associative cache
 - A block can go in any place in the cache
 - All addresses have to be compared simultaneously
 - Slow and expensive
- N-way set-associative cache
 - Consists of a number of sets
 - Each set consists of N blocks
 - Each block in memory maps to a unique set
 - A block can be placed in any element of the set

Locating a Block in a Cache

Block address = 12_{10}



Cache Configurations

- An eight-block cache with various configurations

On-way set associative
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Which cache has the largest number of tag bits?
Which cache has the least?

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

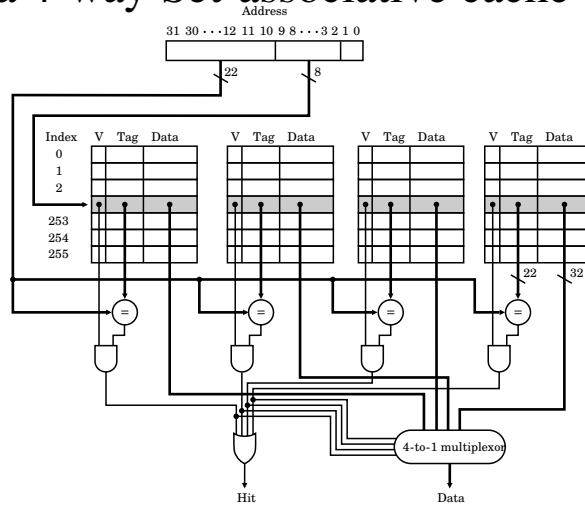
Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Accessing a 4-way Set-associative cache

Number of Blocks?

Number of Sets?

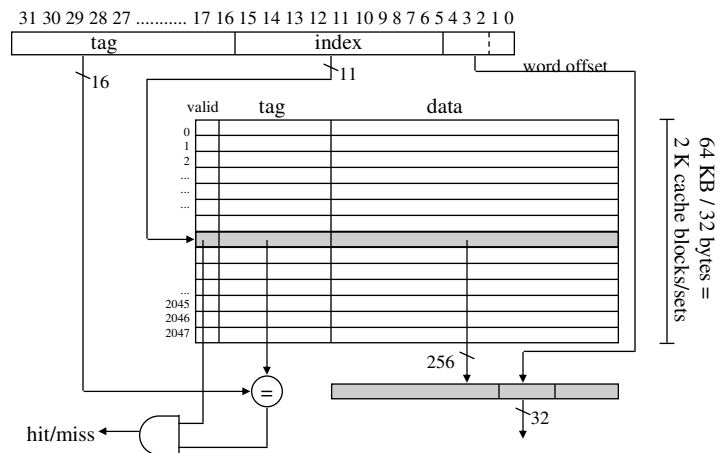
Index bits?



• 4 K-byte 4-way set-associative cache, with a block size of 4 bytes

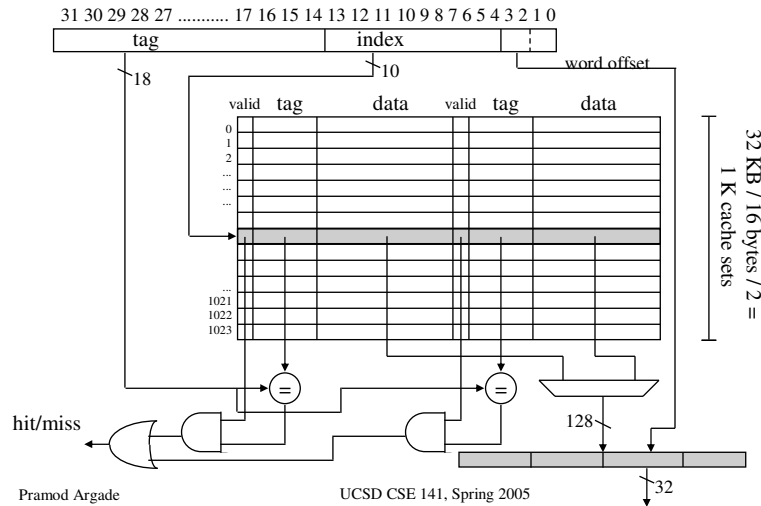
Accessing a Direct Mapped Cache

- 64 KB cache, direct-mapped, 32-byte cache block size



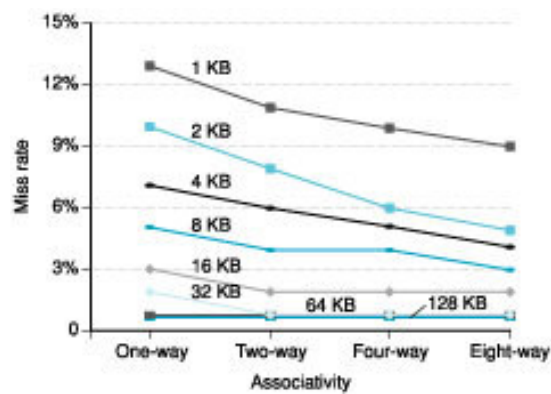
Accessing a Set-associative Cache

- 32 KB cache, 2-way set-associative, 16-byte block size



Cache Associativity and Miss Rate

Data cache miss rate for SPEC2000 benchmark



- Benefit of going from direct mapped to two-way set associative cache is significant.
- Benefits of further increase in associativity are smaller.

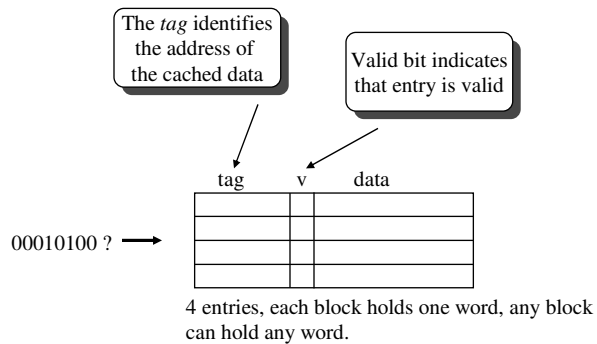
Associative Caches: Higher hit rates, but...

- Longer access time (longer to determine hit/miss, more muxing of outputs)
- More space (longer tags)
 - 16 KB, 16-byte blocks, direct mapped, tag = ?
 - 16 KB, 16-byte blocks, 4-way, tag = ?

A Fully-associative cache

Word address string:

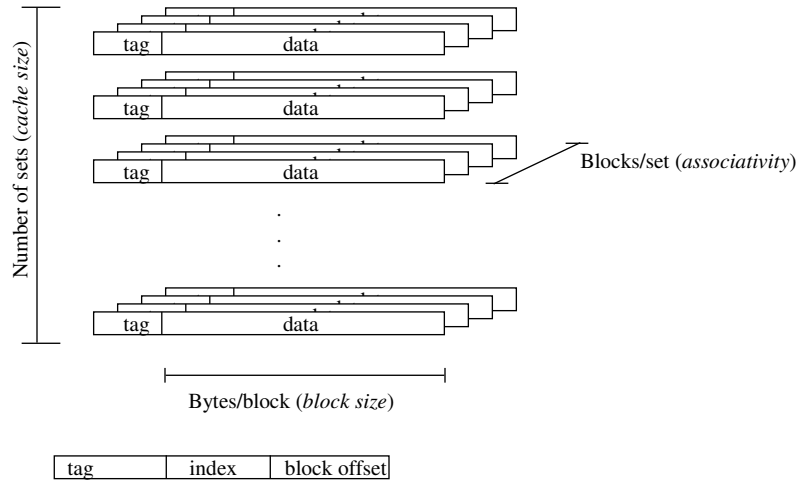
20	00010100
5	00000101
10	00001010
12	00001100
4	00000100
9	00001001
7	00000111
8	00001000
21	00010101
24	00011000
14	00001110
11	00001011
4	00000100



- A cache that can put a block of data anywhere is called *fully associative*
- To access the cache, address must be compared with all the entries in the cache

Cache Organization

- A typical cache has three dimensions



Pramod Argade

UCSD CSE 141, Spring 2005

16-23

The Three Cs

- Compulsory misses: **First time access**
 - Caused by the first access to a block that has never been in the cache
 - Also called cold-start misses
 - Can be reduced by increasing the block size
- Capacity misses: **Cache is not large enough**
 - Caused when cache cannot contain all the blocks needed
 - Occur because of blocks being replaced and later retrieved
 - Can be reduced by enlarging the cache
- Conflict misses: **Multiple addresses map to the same cache line**
 - Occur in direct mapped and set-associative caches
 - Multiple blocks compete for the same set
 - Can be eliminated by using fully associative cache

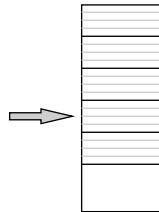
Pramod Argade

UCSD CSE 141, Spring 2005

16-24

Which Block Should be Replaced on a Miss?

- Direct Mapped is Easy
- Set associative or fully associative:
 - Random (large associativities)
 - LRU (smaller associativities)
- Miss rates for the two schemes:



Associativity:	2-way		4-way		8-way	
	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

LRU is preferred scheme for a small size cache

Measuring Cache Performance

CPU time = (CPU execution clock cycles + Memory stall cycles)
 * Clock cycle time

Memory stall cycles = Read-stall cycles + Write-stall cycles

Read-stall cycles = (Reads per program) * Read miss rate *
 Read miss penalty

Write-stall cycles = (Writes per program) * Write miss rate *
 Write miss penalty

Cache Performance

- Ideal CPI for a processor (i.e. without memory stalls) is 1.4.
- 35% of the instructions are loads and stores.
- Miss penalty is 10 cycles
- How much do we improve the performance if the miss rate is reduced from 10% to 2%?

Summary

- The Principle of Locality:
 - Program likely to access a relatively small portion of the address space at any instant of time.
 - Temporal Locality: Locality in Time
 - Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses:
 - Compulsory Misses: sad facts of life. Example: cold start misses.
 - Conflict Misses: increase cache size and/or associativity.
Nightmare Scenario: ping pong effect!
 - Capacity Misses: increase cache size
- Cache Design Space
 - total size, block size, associativity
 - replacement policy
 - write-hit policy (write-through, write-back)
- Caches give an illusion of a large, cheap memory with the access time of a fast, expensive memory