TEXAS TECH UNIVERSITY
Department *of* Computer Science

CS5375 Computer Systems Organization and Architecture

Lecture 7

Instructor: Yong Chen, Ph.D.
Department of Computer Science
Texas Tech University
Yong.Chen@ttu.edu, 806-834-0284

Announcements

- HW#1 posted and due on 9/20, Tue., 11:59 p.m.

- Our TA (she'll grade all assignments and assist learning):
  - Ms. Nabonita Mitra
  - Email: nmitra@ttu.edu
  - Office: EC (Engineering Center) 203A
  - Office hours: 4 – 5:30 pm on Mondays and 5 – 6:30 pm on Wednesdays

- Conference trip next week
  - Class on 9/20 is skipped, leave the time for you to work on HW#1
  - Class on 9/22 will be provided via recording

Review of Last Lecture

- Cache Memory
  - Direct mapped cache: only one choice, use the below mapping to find a block

  (Block address) modulo (#Blocks in cache)

  Block address = memory address (byte address) / block size (i.e., how many bytes in one block)
  - Tags and Valid Bits
  - Examples
  - Address subdivision

- Measuring Cache Performance
  - Impact of cache (instruction and data accesses) on CPI

Outline

- Measuring Cache Performance (cont.)
- Associative Caches
- Multilevel Caches

Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent

- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11

- Better solution: write through with a write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write-Back

- Alternative: on data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
- Called write-back cache

Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses

- With simplifying assumptions:

$$\text{Memory stall cycles}$$

$$= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions

- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44

- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU with perfect cache is 5.44/2 =2.72 times faster

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty

- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%

  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

Outline

- Measuring Cache Performance (cont.)
- Associative Caches
- Multilevel Caches

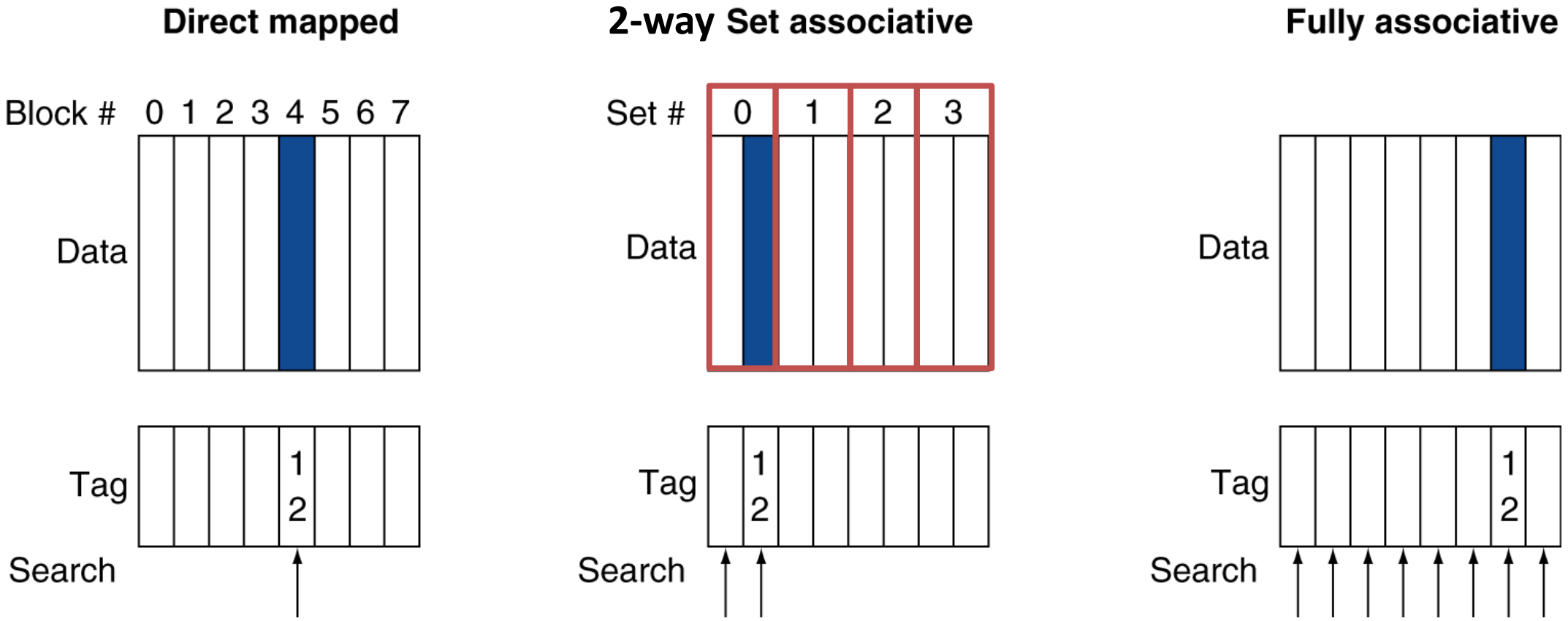
Associative Caches

- Focuses on reducing the miss rate of direct-mapped cache by reducing the probability that two different memory blocks contend for the same cache location

- Fully associative
  - Allow a given cache block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)

- *n*-way set associative
  - Cache is divided into multiple/many sets, and each set contains *n* entries
  - A given cache block is mapped to a set (below formula), and the cache block can go in any entry in that set
    (Block address) modulo (#Sets in cache)
  - Requires searching all entries in a given set at once
  - *n* comparators (less expensive)

Associative Cache Example



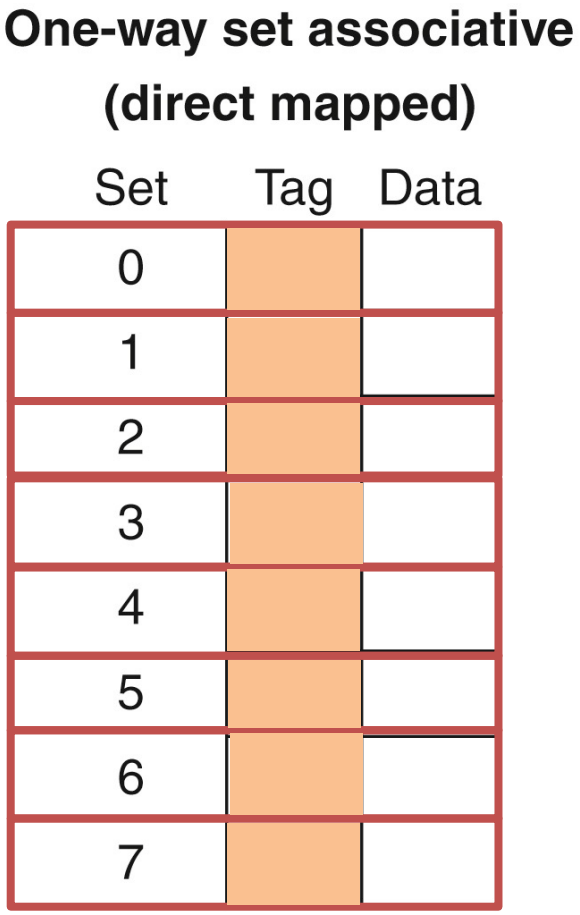
A cache block can only go in exactly a single location in the cache

Cache is divided into sets (4 sets here), and a given cache block is mapped to a set and can go in any cache entry in that set (2-way set here)

A cache block can go in any entry in the cache

Spectrum of Associativity

- For a cache with 8 entries
  - Can be one-way set associative, i.e., direct mapped

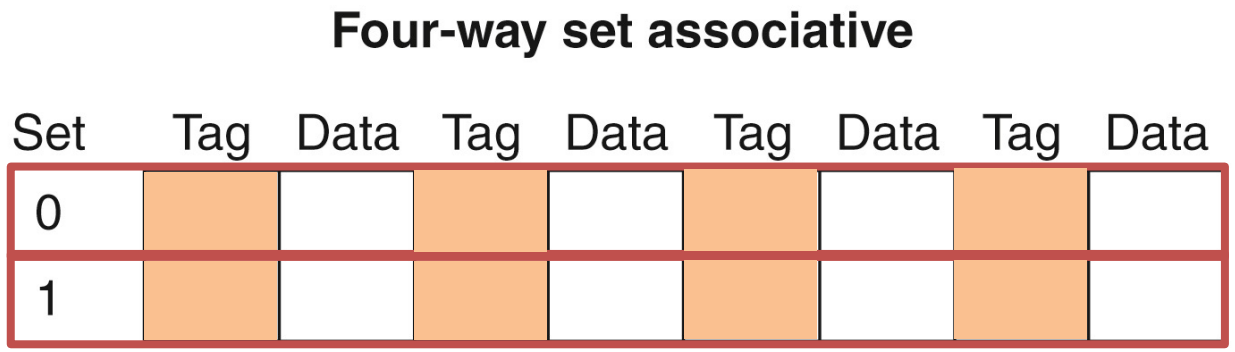


Spectrum of Associativity

- For a cache with 8 entries
  - Can be two-way set associative

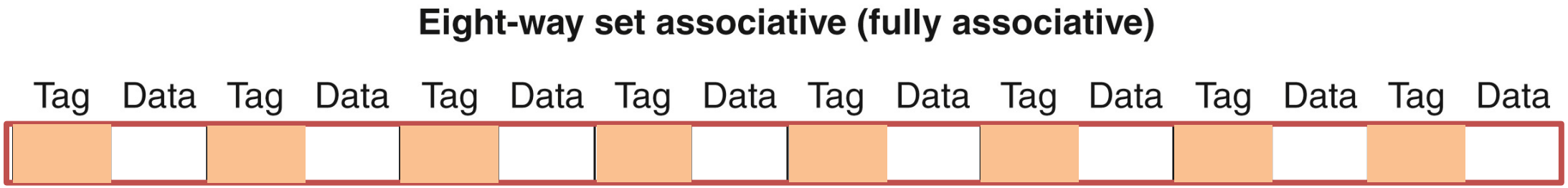
Spectrum of Associativity

- For a cache with 8 entries
  - Can be four-way set associative



Spectrum of Associativity

- For a cache with 8 entries
  - Can be eight-way set associative, i.e., fully-associative



Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped: (Block address) modulo (#Blocks in cache)

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

Block address	Hit/ miss	Cache content after access			
		0	1	2	3
0	miss	Mem[0]			
8	miss	**Mem[8]**			
0	miss	**Mem[0]**			
6	miss	Mem[0]		Mem[6]	
8	miss	**Mem[8]**		Mem[6]	

Five misses!

Associativity Example

- 2-way set associative

  Set is determined by (Block address) modulo (#Sets in cache)

  Within that set, the cache block can go in any entry

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

Block address	Hit/miss	Cache content after access			
		Set 0		Set 1	
0	miss	Mem[0]			
8	miss	Mem[0]	Mem[8]		
0	hit	Mem[0]	Mem[8]		
6	miss	Mem[0]	**Mem[6]**		
8	miss	**Mem[8]**	Mem[6]		

Four misses!

A replacement rule is needed for replacing which block, e.g., replacing "least recently used (LRU)"

Associativity Example

- Fully associative

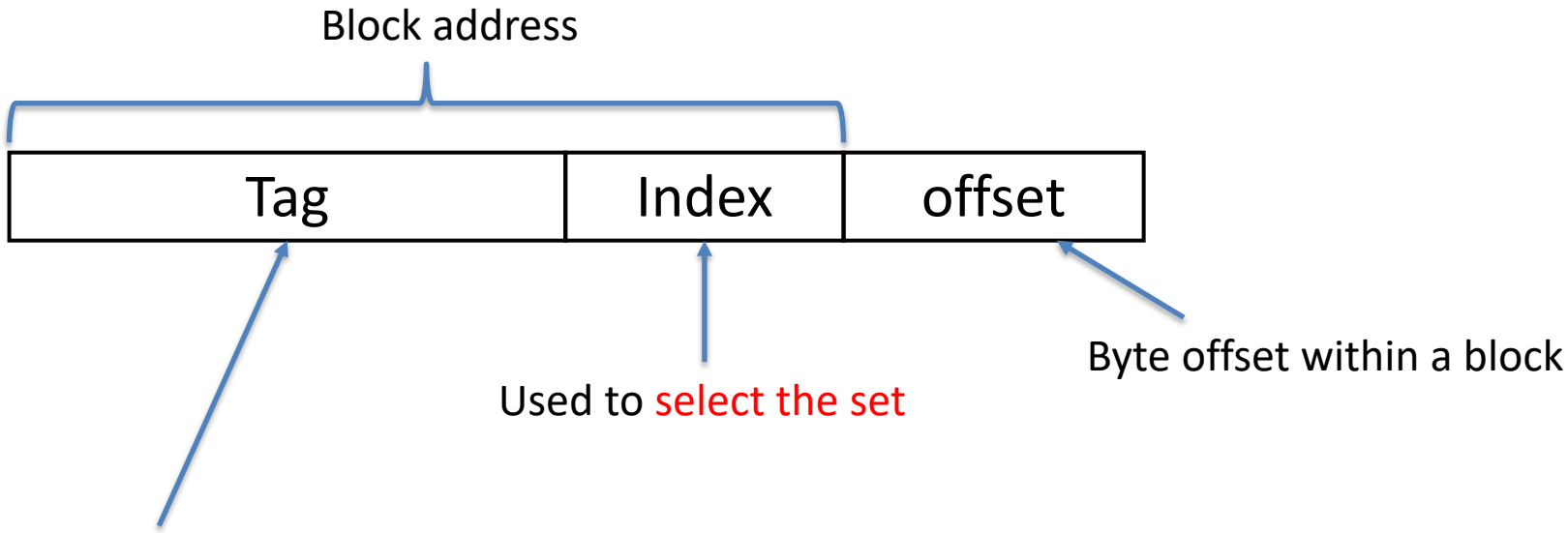
Block address	Hit/miss	Cache content after access			
0	miss	Mem[0]			
8	miss	Mem[0]	Mem[8]		
0	hit	**Mem[0]**	Mem[8]		
6	miss	Mem[0]	Mem[8]	Mem[6]	
8	hit	Mem[0]	**Mem[8]**	Mem[6]	

Three misses!

Causes/Classifications of Misses

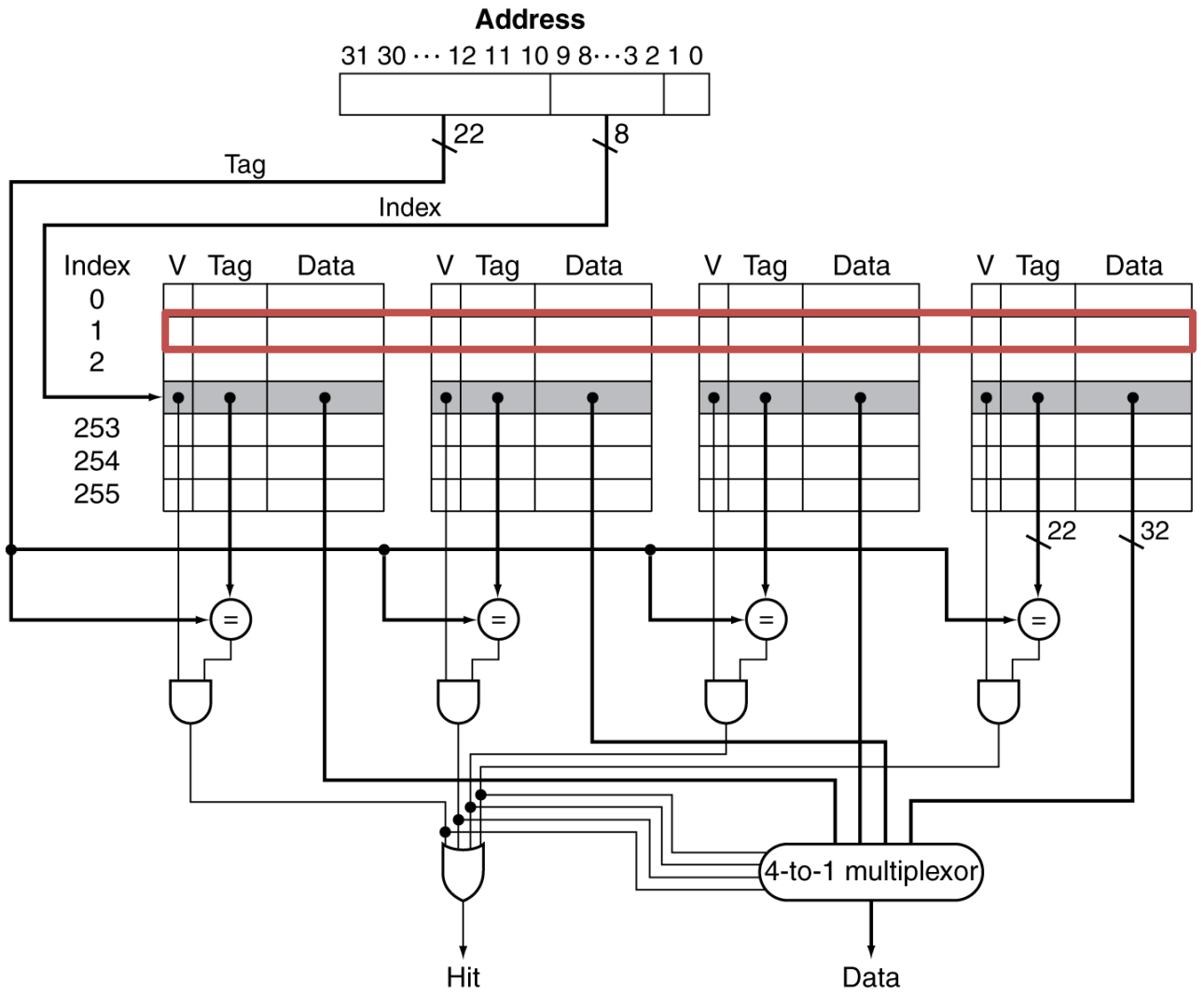
- Compulsory misses
  - First reference to a block
- Capacity misses
  - Blocks discarded and later used, due to limited capacity of cache, even it's fully-associative
- Conflict misses
  - Repeated references to multiple addresses from different blocks that map to the same location in the cache
  - Misses caused by limited associativity, i.e., those misses won't exist if it's a fully-associative cache

Locating a Block in the Associative Cache



Used to compare against all the tags in the selected set.
As speed is of essence, all tags in the selected set are searched in parallel, instead of a sequential search

Set Associative Cache Organization



256 sets in total

One set, 4-way

A four-way set-associative cache requires four comparators and a 4-to-1 multiplexor

Replacement Policy

- Direct-mapped
  - Always mapped to one single entry in the cache, no choice

- Set associative and fully associative
  - Fully associative is a special case of n-way set associative, where there is only 1 set in the cache
  - Replace a non-valid entry first, if there is one (equivalent as the set is not full yet)
  - Otherwise, use a replacement algorithm to choose among entries in the set

Replacement Policy (cont.)

- Least-recently used (LRU) replacement algorithm (e.g., your cell phone recent call list)
  - Choose the one unused for the longest time
    - Need to keep track the access history
    - Simple for 2-way, manageable for 4-way, too hard beyond that
  - Often a pseudo LRU (PLRU) is used

- Random replacement algorithm
  - Hardware generates a random number and selects a replacement candidate out of *n* choices
  - Low cost, does not require keeping any information about the access history
  - Gives approximately the same performance as LRU for high associativity

- Other replacement algorithms exist too, e.g., NRU (Not Recently Used)

Outline

- Measuring Cache Performance (cont.)
- Associative Caches
- Multilevel Caches

Multilevel Caches

- Focusing on reducing miss penalty
  - Further close the gap between fast clock rates of processors and increasingly long time required to access memory (DRAMs)

- Primary cache (Level-1, or L1, or L1$) attached to CPU
  - Small, but very fast

- Level-2 cache (L2, or L2$) services misses from primary cache
  - Larger, slower, but still faster than main memory, ~10 cycles
  - If L2 cache is the LLC (last-level cache), main memory services L2 cache misses, ~100 cycles

- Some high-end systems include Level-3 cache (L3 or L3$), ~20 cycles
  - Then L3 cache is the LLC, main memory services L3 cache misses

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate per instruction = 2%
  - Main memory access time = 100ns

- With just primary cache
  - Miss penalty? 100ns/0.25ns = 400 cycles
  - Effective CPI? $1 + 0.02 \times 400 = 9$

Example (cont.)

- Now add L2 cache
  - Access time = 5ns
  - Global miss rate to main memory (out of all memory accesses from instructions) is reduced to 0.5%

- Primary cache miss, but with L2 hit
  - Penalty? 5ns/0.25ns = 20 cycles

- Primary cache miss, and with L2 miss
  - Penalty? 20 cycles + 400 cycles = 420 cycles

Those memory references that go to main memory, which must include the cost to access the secondary cache as well as the main memory access time

- Effective CPI? $1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- Performance improvement = 9/3.4 = 2.6X

Multilevel Cache Considerations

- Design considerations for an L1 and L2 cache are significantly different
- Primary cache (i.e., L1 cache)
  - Focus on minimal hit time
- L2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L1 cache usually smaller than a single cache
  - L1 block size smaller than L2 block size
  - L2 often uses higher associativity than L1 as focusing on reducing miss rates

Readings

- Chapter 2, 2.1-2.2