



CS5375 Computer Systems Organization and Architecture

Lecture 6

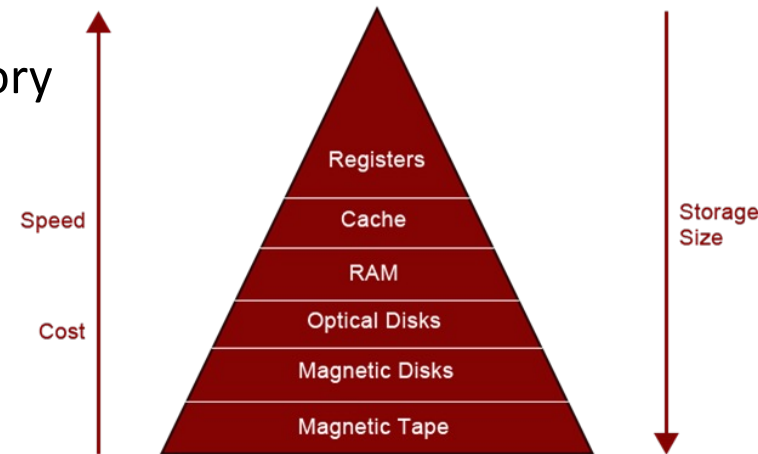
Instructor: Yong Chen, Ph.D.
Department of Computer Science
Texas Tech University
Yong.Chen@ttu.edu, 806-834-0284

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) 20B (basement)
 - Office hours: 4 – 5:30 pm on Mondays and 5 – 6:30 pm on Wednesdays

Review of Last Lecture

- Amdahl's Law and Scaled Computing (cont.)
 - Scaled computing concept focuses on **scaling the problem size**, i.e., solving a larger problem with a more powerful computer, instead of solving the same, fixed-size problem in Amdahl's analysis
 - Shows the scaled speedup is a linear function of n , demonstrates **great promise of parallelism**
- Memory Hierarchy Design
 - Critical to bridge the performance gap between processor and memory
 - Based on the **Principle of Locality**
 - Temporal locality (locality in time)
 - Spatial locality (locality in space)
 - Multiple levels of memory with different speeds and sizes
 - Terminologies
 - **Cache memory**: the level of the memory hierarchy closest to the CPU

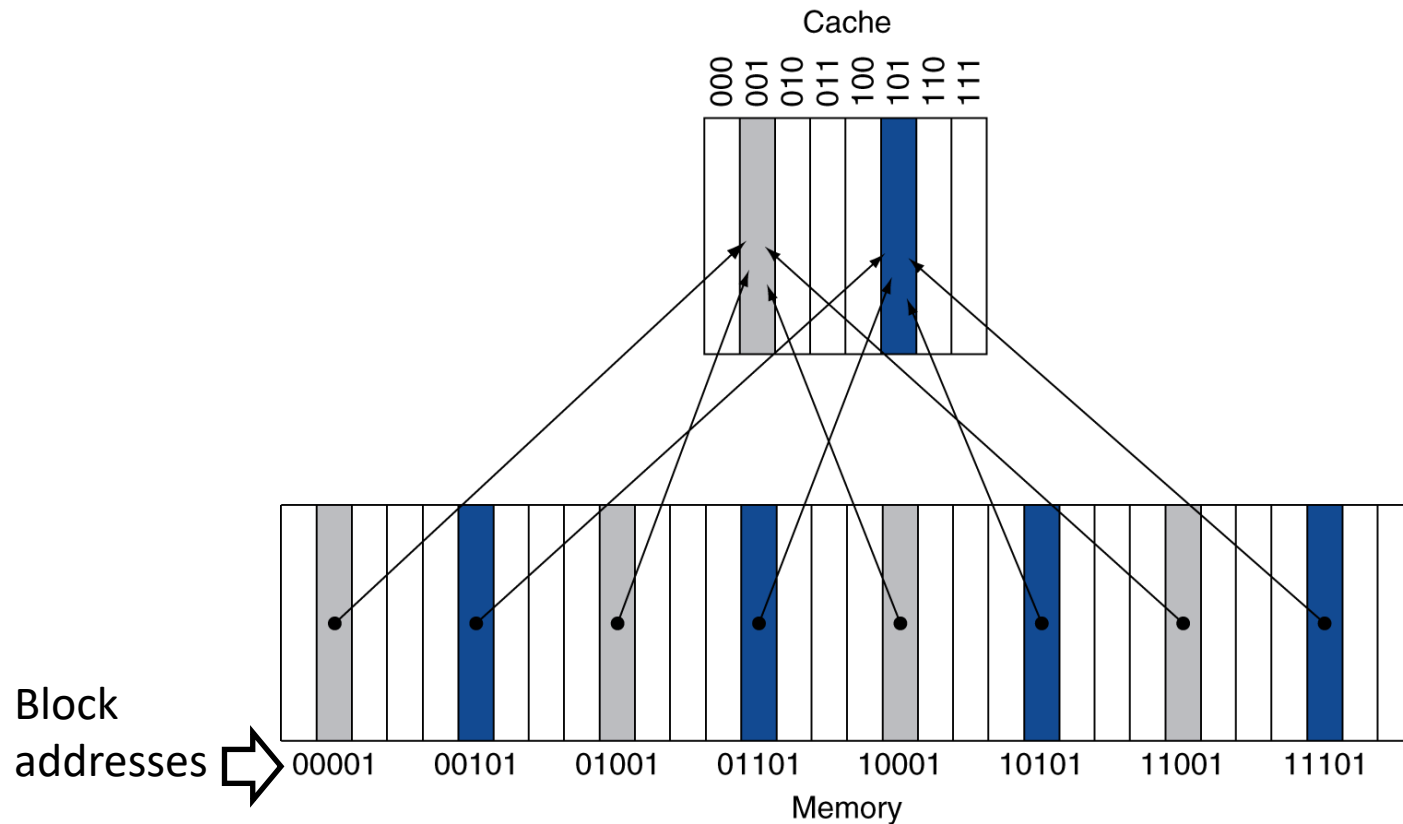


Outline

- Cache Memory
- Measuring Cache Performance
- Associative Caches

Direct Mapped Cache

- Location determined by the memory address, more specifically by **Block address** = memory address (byte address) / block size (i.e., how many bytes in one block)
- **Direct mapped: only one choice**, use the below mapping to find a block
(**Block address**) modulo (**#Blocks in cache**)
- Granularity/size of a cache block (or cache line)



- #Blocks is a power of 2
- Modulo can be simply calculated by using **low-order $\log_2(\text{cache size in blocks})$ address bits**
- E.g. in this example, cache size in blocks, i.e. #Blocks in cache, is 8. We use low-order 3 bits of block address

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
 - Store block address as well as the data
 - Actually, **only need the high-order bits**
 - Called the **tag**
- What if there is no data in a location?
 - **Valid bit**: 1 = present, 0 = not present
 - Initially 0

Cache Example

- 8-blocks, cache block size (or cache line size): 1 word, direct mapped
 - Since the block size is same as the word size, therefore, block address == word address
- Assume accessing a sequence of 9 words from memory: 22, 26, 22, 26, 16, 3, 16, 18, 16
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

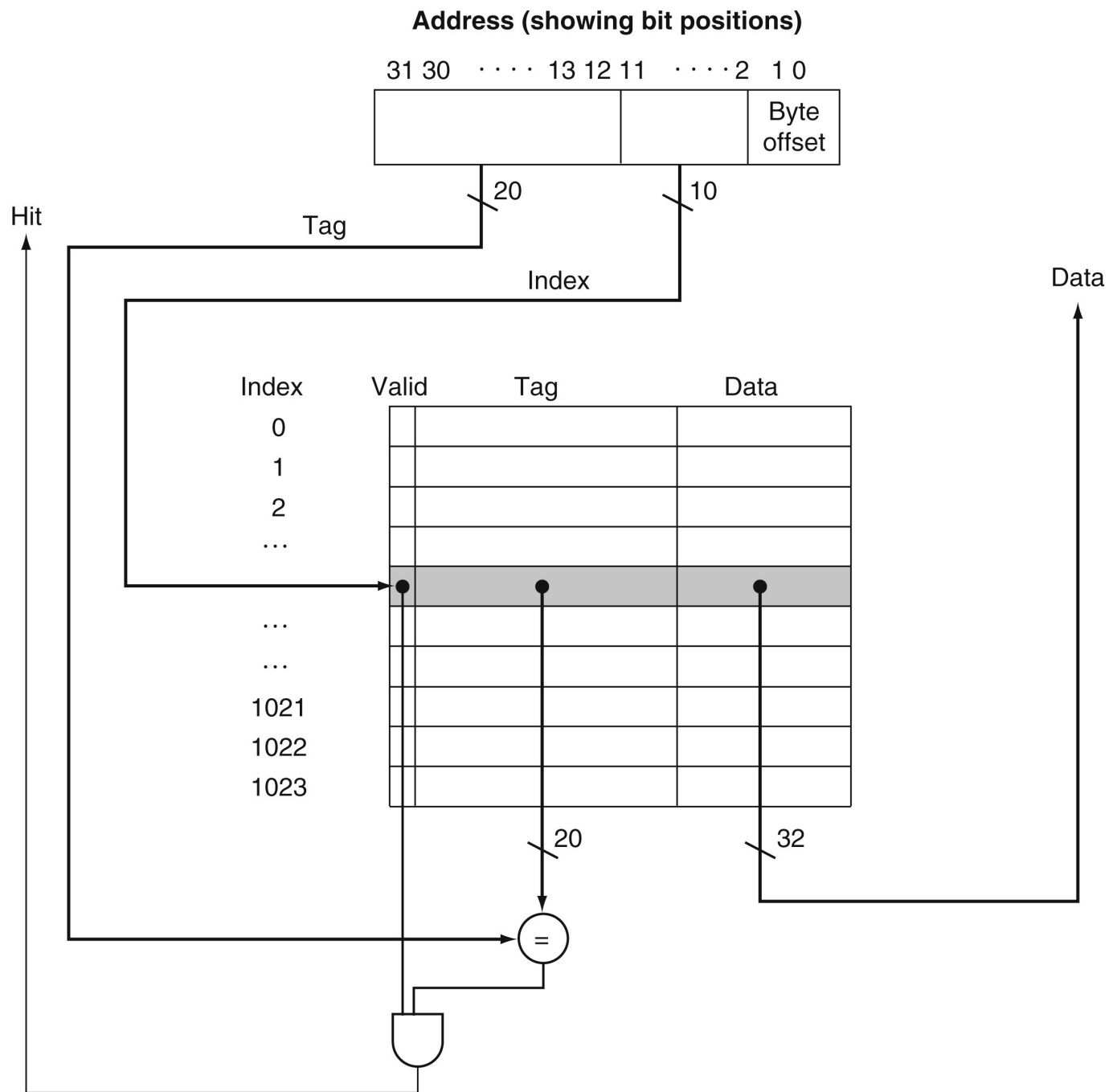
Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

A cache block is
being replaced

Example: Accessing a Cache

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss	$(10110_{\text{two}} \bmod 8) = 110_{\text{two}}$
26	11010_{two}	miss	$(11010_{\text{two}} \bmod 8) = 010_{\text{two}}$
22	10110_{two}	hit	$(10110_{\text{two}} \bmod 8) = 110_{\text{two}}$
26	11010_{two}	hit	$(11010_{\text{two}} \bmod 8) = 010_{\text{two}}$
16	10000_{two}	miss	$(10000_{\text{two}} \bmod 8) = 000_{\text{two}}$
3	00011_{two}	miss	$(00011_{\text{two}} \bmod 8) = 011_{\text{two}}$
16	10000_{two}	hit	$(10000_{\text{two}} \bmod 8) = 000_{\text{two}}$
18	10010_{two}	miss	$(10010_{\text{two}} \bmod 8) = 010_{\text{two}}$
16	10000_{two}	hit	$(10000_{\text{two}} \bmod 8) = 000_{\text{two}}$



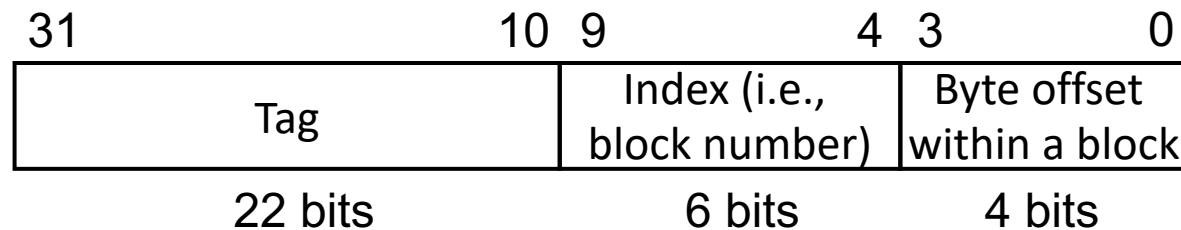
Accessing a Cache and Address Subdivision

Accessing a Cache and Address Subdivision

- What's the size of data in this cache?
 - 4 KiB (1K entries of 4B data each), called a 4 KiB cache as a convention
- The total number of bits needed for a cache includes the size of the valid field, the size of the tag field, and the size of the data
- What's the total number of bits needed for this particular direct-mapped cache?
 - Leave it as a homework question

Another Example: Larger Block Size

- Assume a 1KiB cache, 64 blocks, 16 bytes/block (or 4 words/block)
 - To what cache block number does memory address 1200_{ten} map?
- **Block address** = memory address (byte address) / block size = $\lfloor 1200_{\text{ten}} / 16_{\text{ten}} \rfloor = 75_{\text{ten}}$
 - Block address is calculated as floor(byte address/bytes per block)
- **Block number** = (Block address) modulo (#Blocks in cache) = 75_{ten} modulo $64_{\text{ten}} = 11_{\text{ten}}$



Outline

- Cache Memory
- **Measuring Cache Performance**
- Associative Caches

Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
 - Stall the CPU pipeline
 - Fetch block from next level of hierarchy
 - Instruction cache miss
 - Restart instruction fetch
 - Data cache miss
 - Complete data access

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 = $1 + 0.1 \times 100 = 11$
- 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:

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: $0.02 \times 100 = 2$
 - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- 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)
 - $AMAT = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example
 - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, l-cache miss rate = 5%
 - $AMAT = 1 + 0.05 \times 20 = 2\text{ns}$
 - 2 cycles per instruction

Outline

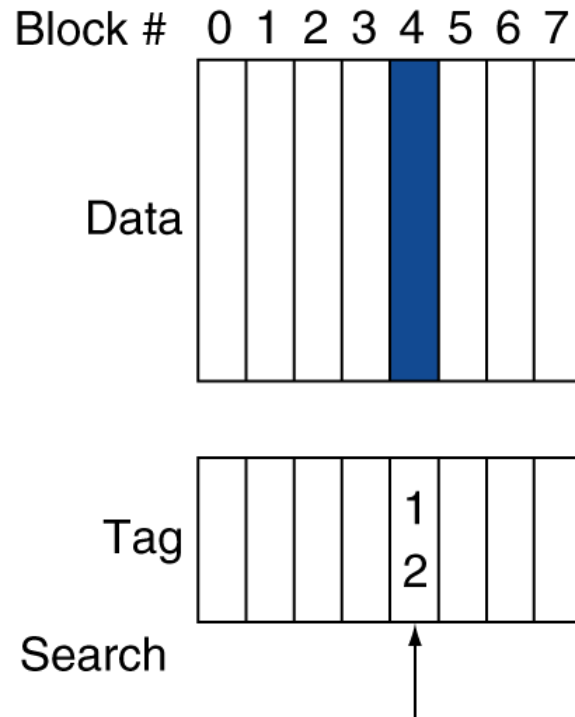
- Cache Memory
- Measuring Cache Performance
- **Associative Caches**

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)

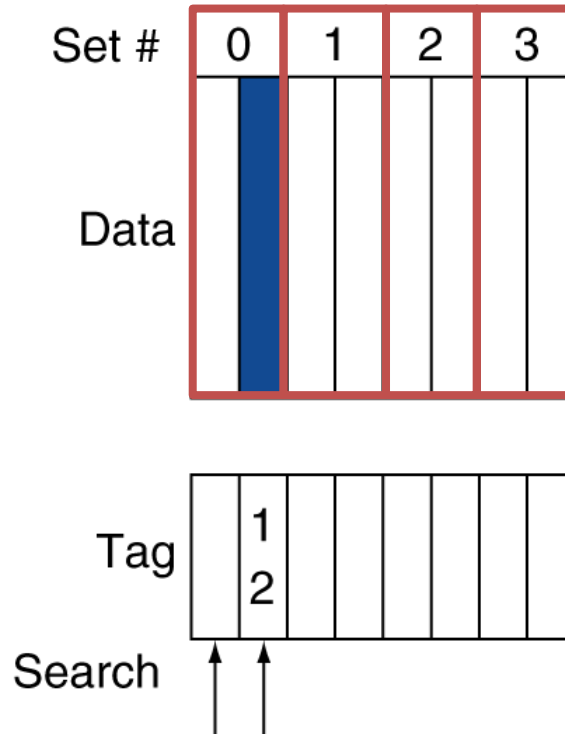
Associative Cache Example

Direct mapped



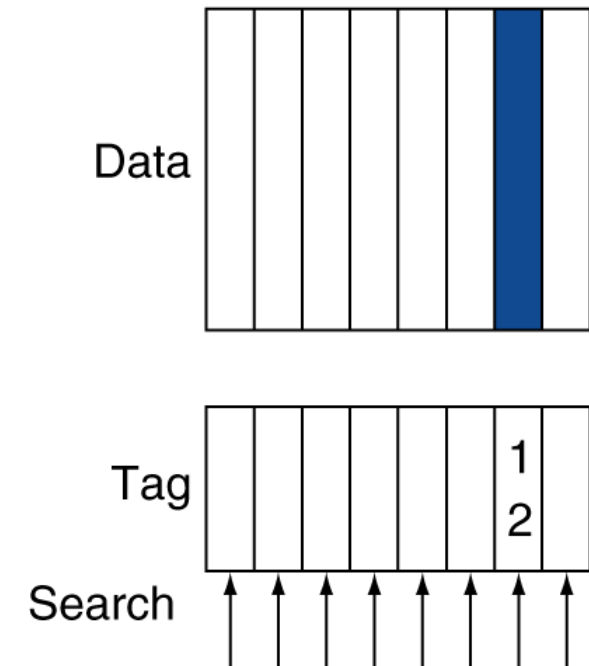
A cache block can only go in **exactly a single location in the cache**

2-way Set associative



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)**

Fully associative



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**

One-way set associative (direct mapped)

Set	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Spectrum of Associativity

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

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Spectrum of Associativity

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

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Spectrum of Associativity

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

Eight-way set associative (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	$(0 \text{ modulo } 2) = 0$
6	$(6 \text{ modulo } 2) = 0$
8	$(8 \text{ modulo } 2) = 0$

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!

Readings

- Chapter 2, 2.1-2.2