CSCI 210: Computer Architecture
Lecture 36: Associative Caches

Stephen Checkoway

Oberlin College

Jan. 10, 2022

Slides from Cynthia Taylor

Announcements

- Problem Set 12 due Friday

- Cache Lab (final project) due on the day of the final exam

- Course evals now available!
  - Extra credit for everyone if more than 90% of the class fills them out

- Office Hours Tuesday 13:30 – 14:30
  - On Zoom

Store-hit policy: write-through

- Update cache block AND memory

- 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

- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Store-hit policy: write-back

- Only update the block in cache
  - Keep track of whether each block is "dirty" (i.e., it has a different value than in memory)
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first
- Faster than write-through, but more complex

V	D	Tag	Data
1	0	0000420	FE FF 3C …
0			
1	1	0012345	32 A0 5C …
0			
0			
1	0	000F3CB	00 00 00 …
0			
0			

Store-miss policy: write-allocate

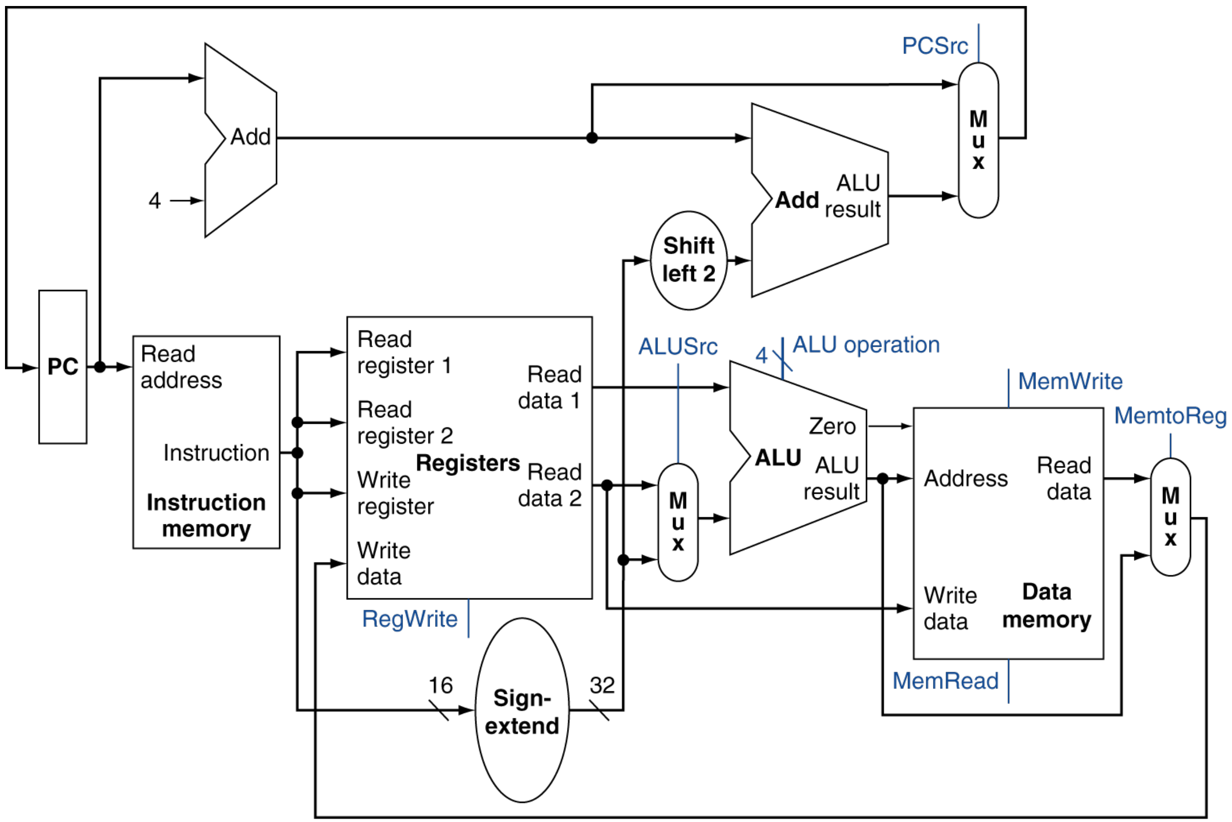
- Read a block from memory (just like a load miss)
- Perform the write according to the store-hit policy (i.e., write in cache or write in both cache and memory)

- Good for when data is likely to be read shortly after being written (temporal locality)

Store-miss policy: write-around

- Only write the data to memory
- Good for initialization where lots of memory is written at once but won’t be read again soon

I-cache vs D-cache



- Separate caches for instruction memory and data memory
- I-cache: instruction cache
- D-cache: data 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:

$$\text{Memory stall cycles}$$

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

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

Miss Cycles Per Instruction

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

	I-cache	D-cache
A	.02 * 100	.04 * 100
B	.02	.04
C	.02 * .36 * 100	.04 * .36 * 100
D	.02 * 100	.04 * .36 * 100

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 is 2
  - Speedup = 5.44/2 = 2.72

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT =

Performance Summary

- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Can’t neglect cache behavior when evaluating system performance

We need the cache to be fast!

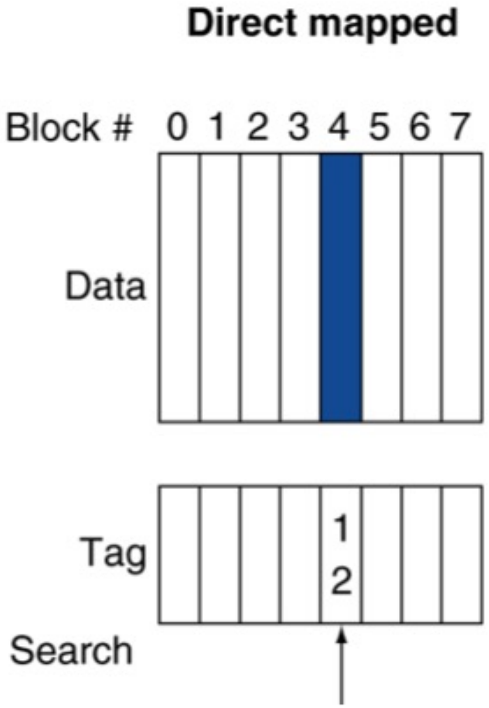
- Memory lookup time
- Hit rate
- Size
- Frequency of collisions

Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks ⇒ fewer of them
    - More competition ⇒ increased miss rate
- Larger miss penalty
  - Can override benefit of reduced miss rate

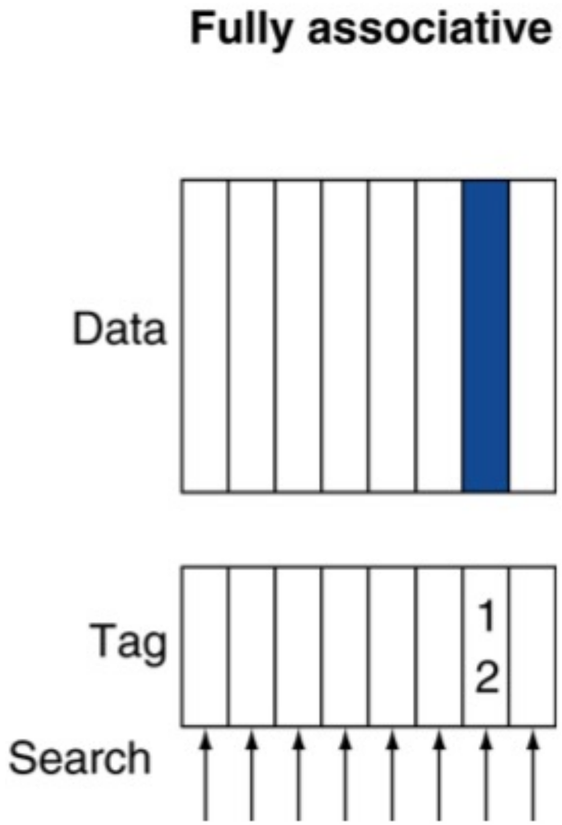
Cache associativity

- Direct mapped
  - Each block goes into **1** spot
  - Only search one entry
  - Associativity = 1
- What if we allow blocks to go into more than one spot?



Cache associativity

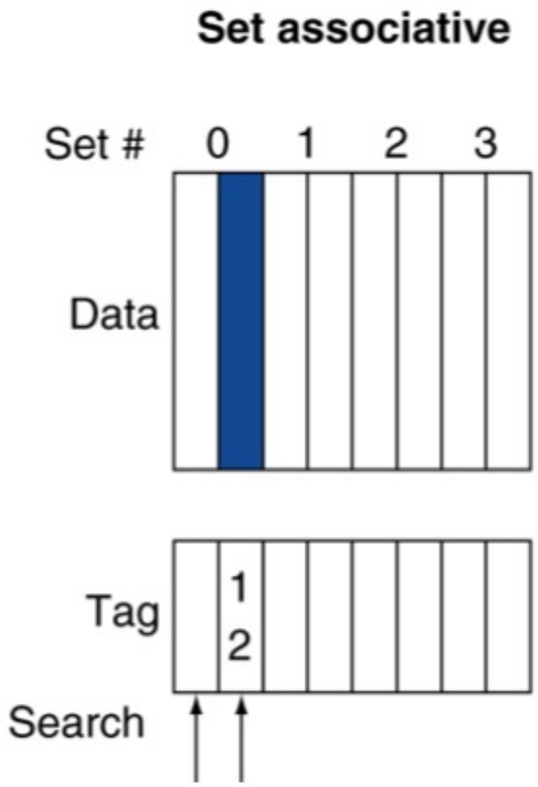
- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)



Cache associativity

n-way set associative

- Each set contains *n* entries
- Block number determines which set
  - – (Block number) % (#Sets in cache)
- Search all entries in a given set at once
- *n* comparators (less expensive)



Spectrum of associativity for 8-entry cache

One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

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	Tag	Data

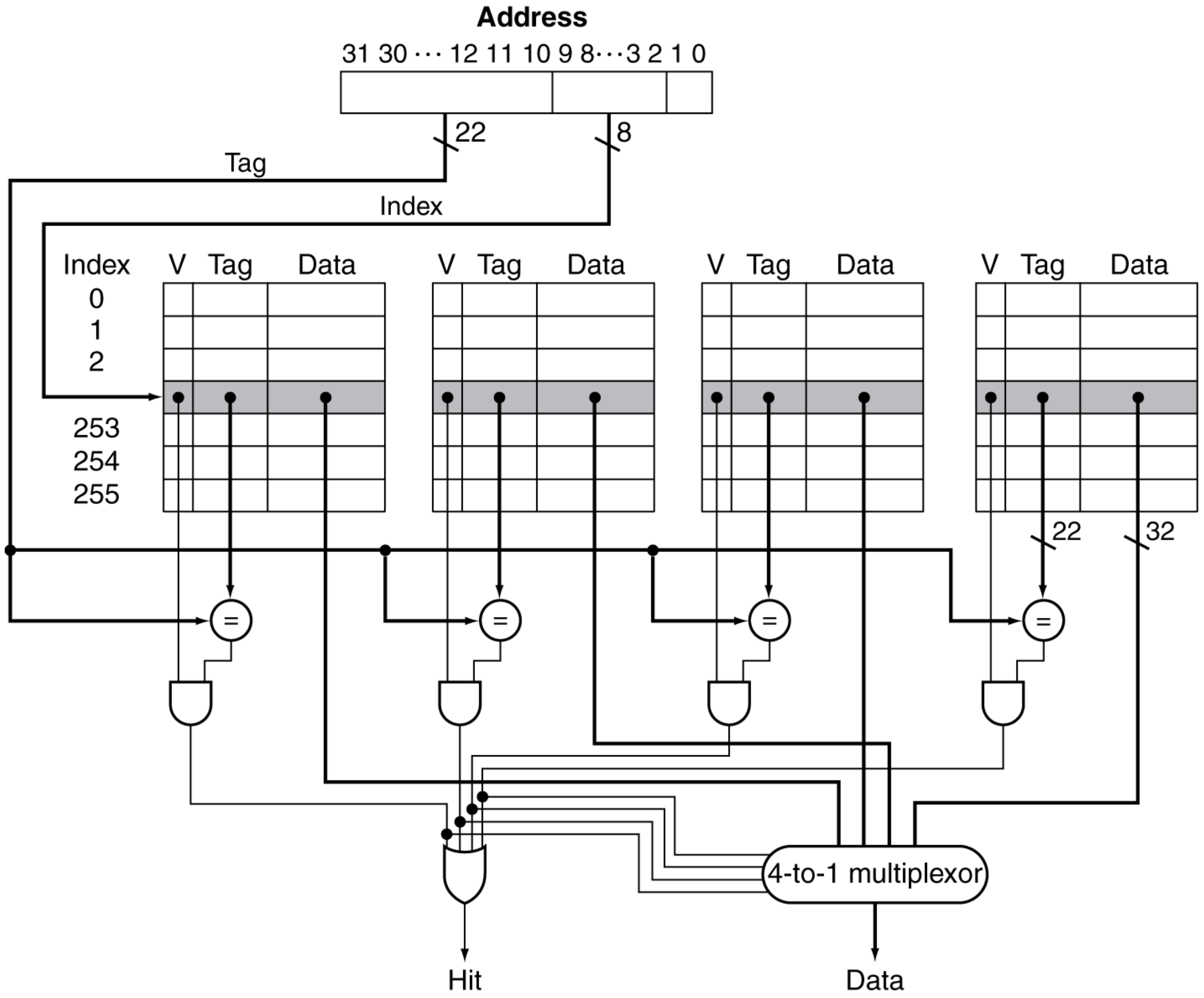
Memory addresses, block addresses, offsets

0	0	0	1	0	1	0	1	1	1	0	0	1	0	0	1	1	0	1	0	1	1	0	0	1	0	1	0	0	0	1	1

- Block size of 32 bytes (not bits!)
- 16-block, 2-way set associative cache
- Each address
  - A (32 – 5)-bit block address (in purple an
  - A 5-bit offset into the block (in green)
- Block address can be divided into
  - A (32 – 3 – 5)-bit **tag** (purple)
  - A 3-bit cache **index** (blue)

V	Tag	Data	V	Tag	Data
0			0		
0			0		
0			1	3F2084	…
0			0		
0			0		
1	15C9AC	…	0		
0			0		
0			0		

Set Associative Cache Organization



Given a 256-entry, 8-way set associative cache with a block size of 64 bytes, how many bits are in the tag, index, and offset?

	Tag bits	Index bits	Offset bits
A	32 – 5 – 6 = 21	5	6
B	32 – 3 – 5 = 24	3	5
C	32 – 8 – 6 = 18	8	6
D	32 – 6 – 5 = 21	6	5
E	32 – 6 – 3 = 23	6	3

Given a 256-entry, fully associative cache with a block size of 64 bytes, how many bits are in the tag, index, and offset?

	Tag bits	Index bits	Offset bits
A	32 – 5 – 6 = 21	1	6
B	32 – 3 – 5 = 24	3	5
C	32 – 8 – 6 = 18	8	6
D	32 – 6 – 5 = 21	6	5
E	32 – 0 – 6 = 26	0	6

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	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0					
8	0					
0	0					
6	2					
8	0					

Associativity Example: 0, 8, 0, 6, 8

- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0					
8	0					
0	0					
6	0					
8	0					

- Fully associative

Block address		Hit/miss	Cache content after access			
0						
8						
0						
6						
8						

Replacement Policy

- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that

- Random
  - Gives approximately the same performance as LRU for high associativity

How Much Associativity

- Increased associativity decreases miss rate
  - – But with diminishing returns
- Simulation of a system with 64 kB D-cache, 64-byte blocks
  - – 1-way: 10.3%
  - – 2-way: 8.6%
  - – 4-way: 8.3%
  - – 8-way: 8.1%

Reading

- Next lecture: More Caches!
  - Section 6.4
- Problem Set 12 due Friday
- Cache lab