

CSCI 210: Computer Architecture

Lecture 35: Associative Caches

Stephen Checkoway

Oberlin College

Jan. 7, 2022

Slides from Cynthia Taylor

Announcements

- Problem Set 12 due next Friday
- Cache Lab (final project) due Friday, Jan. 21 at 16:00
- Office Hours today 13:30 – 14:30
 - On zoom

Cache Size vs Memory Size

- USB-C Charge Cable (2 m)

Configure to Order

Configure your MacBook Pro with these options, only at apple.com:

- 2.4GHz 8-core Intel Core i9, Turbo Boost up to 5.0GHz, with 16MB shared L3 cache
- 32GB of 2400MHz DDR4 memory

Memory is 2048 times bigger than cache

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

Cache replacement policy

- On a hit, return the requested data
- On a miss, load block from lower level in the memory hierarchy and write in cache; return the requested data
- Policy: Where in cache should the block be written? (With direct-mapped caches, there's only one possible location: $\text{block_address} \% \text{number_of_blocks_in_cache}$)

Cache policy for stores

- Policy choice for a hit: Where do we write the data?
 - Write-back: Write to cache only
 - Write-through: Write to cache and also to the next lowest level of the memory hierarchy
- Policy choice for a miss
 - Write-allocate: Bring the block into cache and then do the write-hit policy
 - Write-around: Write only to memory

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

Store Policies

- Given either high store locality or low store locality, which policies might you expect to find?
- Write-allocate: create block in cache. Write-around: don't create block. Write-through: update cache + memory. Write-back: update cache only.

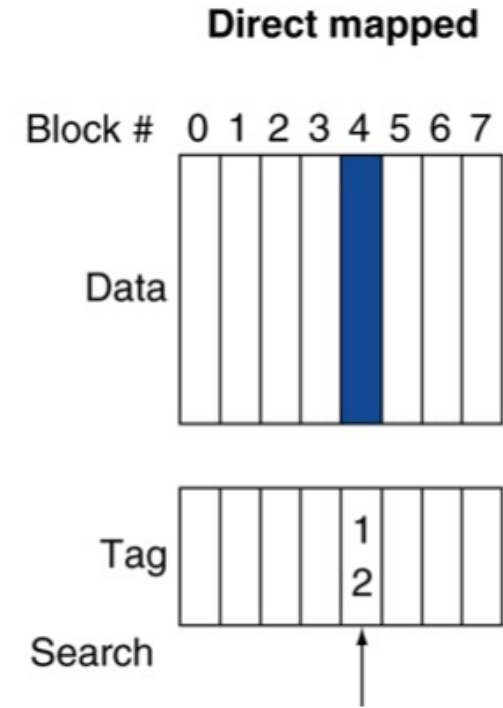
Select ion	High Locality		Low Locality	
	Miss Policy	Hit Policy	Miss Policy	Hit Policy
A	Write-allocate	Write-through	Write-around	Write-back
B	Write-around	Write-through	Write-allocate	Write-back
C	Write-allocate	Write-back	Write-around	Write-through
D	Write-around	Write-back	Write-allocate	Write-through
E	None of the above			

Common policy choices

- Write-back + write-allocate
 - Dirty blocks are written to memory only when replaced
 - Stores bring block into cache
 - Subsequent loads/stores will cause cache hits (unless the block is evicted)
- Write-through + write-around
 - Writes always go to memory
 - Cache is mostly for loads

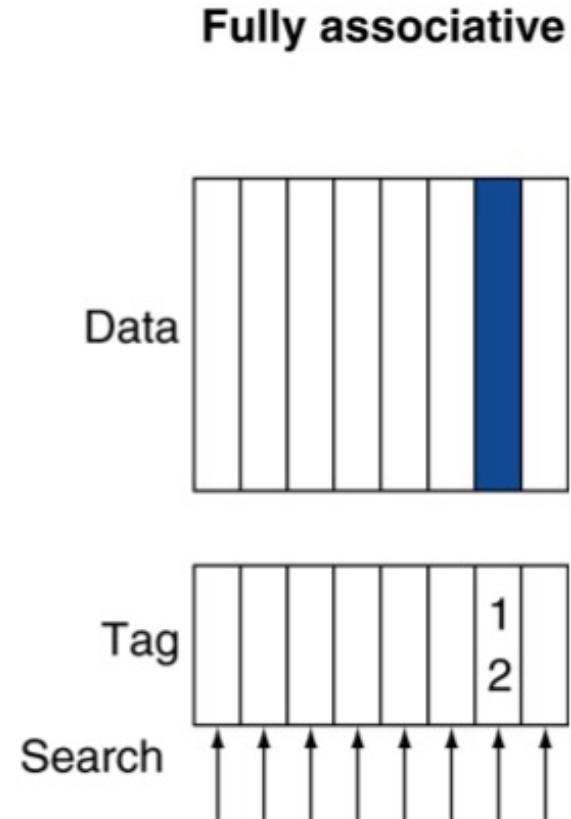
Associative Caches

- 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?



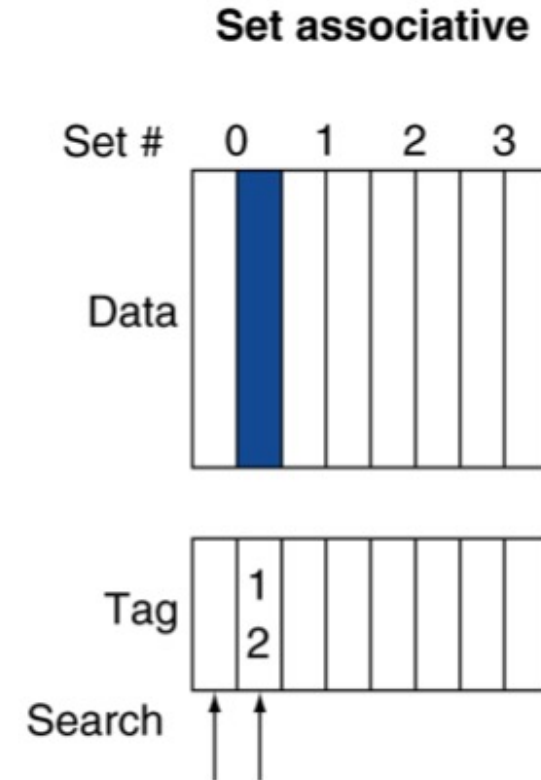
Associative Caches

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



Associative Caches

- *n*-way set associative
 - Each set contains *n* entries
 - Block number determines which set
 - (Block number) modulo (#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)

[illegible]

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 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 and
 - 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		

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

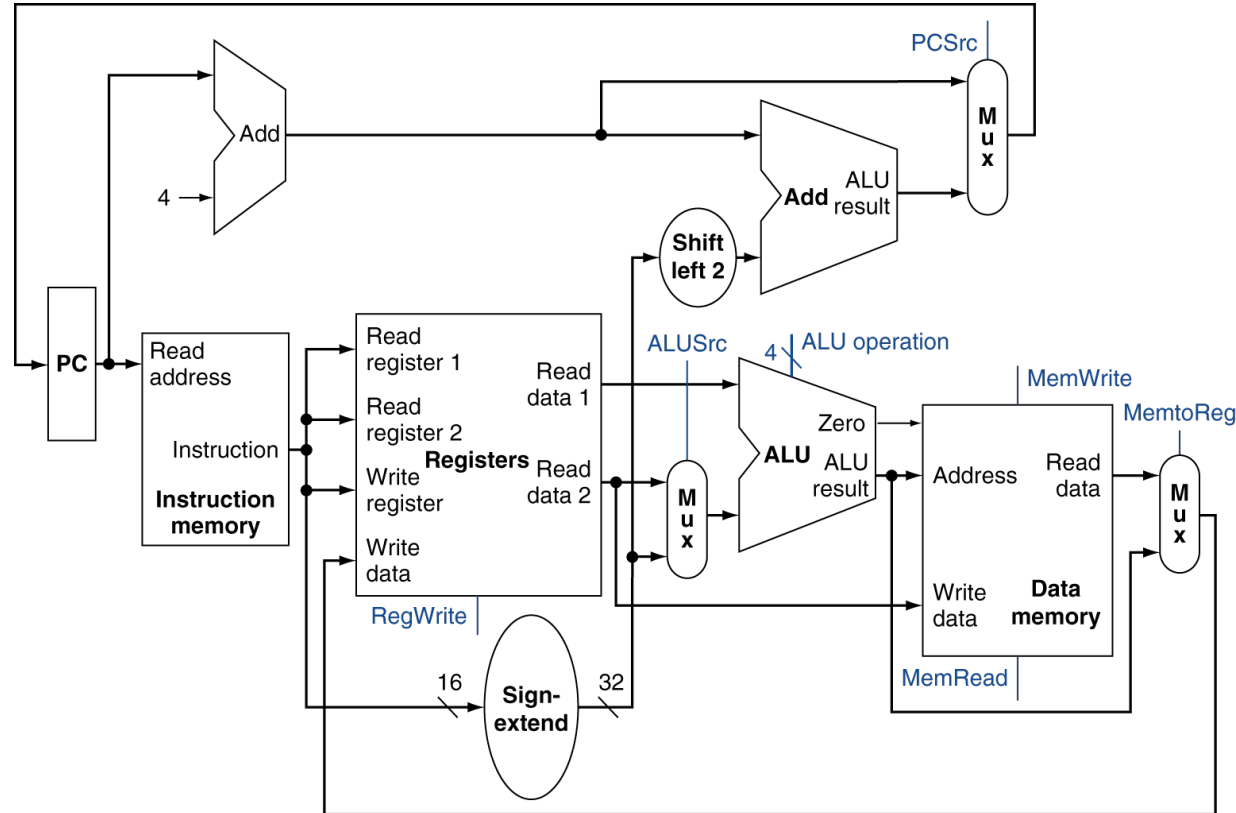
Replacement Policy

- Direct mapped: no choice
- Set associative
 - Prefer non-valid entry, if there is one
 - Otherwise, choose among entries in the set
 - Goal: Choose an entry we will not use in the future

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

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:
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: $0.02 \times 100 = 2$
 - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- Actual CPI = $2 + 2 + 1.44 = 5.44$
 - Ideal CPU is 2: $5.44/2 = 2.72$ 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
 - hit time = 1 cycle, miss penalty = 20 cycles, l-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 cache to be fast!

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

Reading

- Next lecture: More Caches!
 - Section 6.4
- Problem Set 12 due Friday
- Cache Lab (final project) due at the time of the final exam (which this class doesn't have)