EECS 370

Set-associative Caches

Poll: What storage is needed per block?

Class Problem—Storage overhead

- Consider the following cache:
  32-bit memory addresses, byte addressable, 64KB cache
  64B cache block size, write-allocate, write-back, *fully associative*
  This cache will need 512 kilobits for the data area (64 kilobytes times 8 bits per byte). Note that in this context, 1 kilobyte = 1024 bytes (NOT 1000 bytes!) Besides the actual cached data, this cache will need other storage. Consider tags, valid bits, dirty bits, bits to keep track of LRU, and anything else that you think is necessary.
- How many additional bits (not counting the data) will be needed to implement this cache ?

Tag bits = 32 – log(64) = 26 bits
#lines = 64KB/64B = 1024
LRU = log(1024) = 10 bits
1 valid bit, 1 dirty bit

Live Poll + Q&A: slido.com #eecs370

Class Problem—Analyze performance

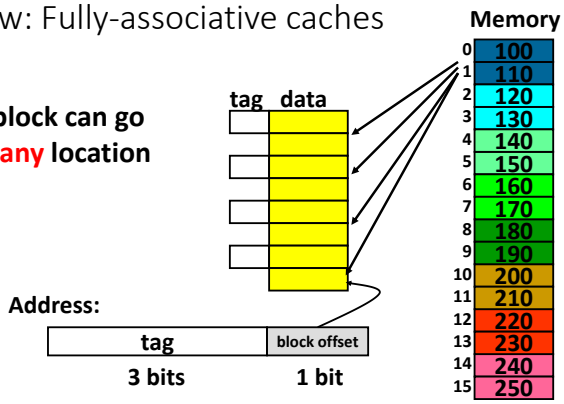
- Suppose that accessing a cache takes 10ns while accessing main memory in case of cache-miss takes 100ns. What is the average memory access time if the cache hit rate is 97%?
  AMAT = 10 + (1 - 0.97)*100 = 13 ns
- To improve performance, the cache size is increased. It is determined that this will increase the hit rate by 1%, but it will also increase the time for accessing the cache by 2ns. Will this improve the overall average memory access time?

AMAT = 12 + (1 - 0.98)*100 = 14 ns

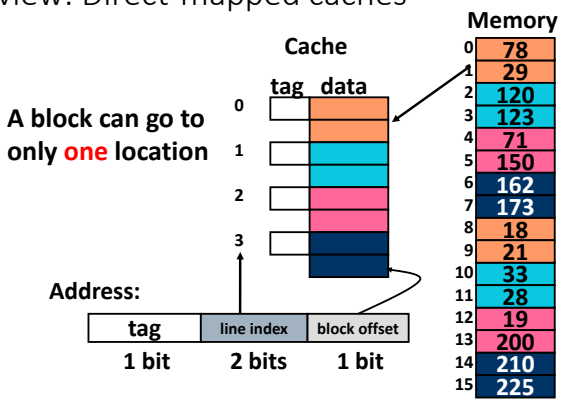
Agenda

- **Set-associativity overview**
- Example
- Class problem
- Integrating caches into our processor

Review: Fully-associative caches



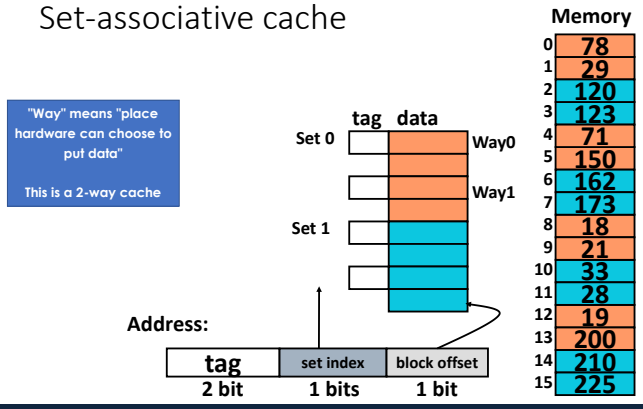
Review: Direct-mapped caches



Set-associative caches

- Fully-associative & direct mapped are two extremes:
  - Slow & full placement flexibility vs fast & no placement flexibility
  - Can we do something in the middle?
- Set associative caches:
  - Partition memory into regions
    - like direct mapped but fewer partitions
  - Associate a region to a set of cache lines
    - Check tags for all lines in a set to determine a HIT
- Treat each set like a small fully associative cache
  - LRU (or LRU-like) policy generally used

Set-associative cache



Calculating all the bit sizes

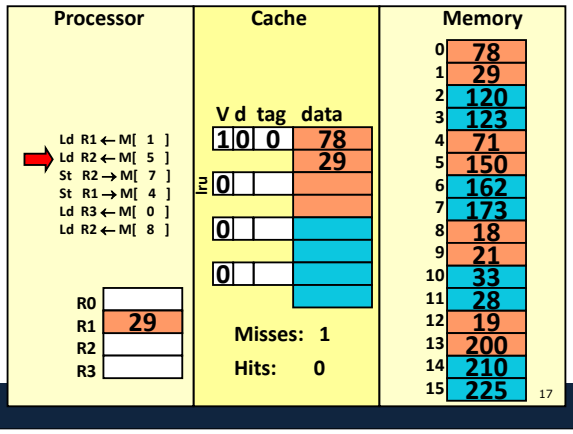
Poll: How many sets are in a direct mapped cache with N cache lines?

- For a set-associative cache:
  - # block offset bits = $\log_2$(block size)
  - # set index bits = $\log_2$(# of sets)
  - # tag bits = rest of address bits
- Fully-associative
  - Special case where (# sets) = 1
- Direct-mapped:
  - Special case where (# sets) = (# cache lines)

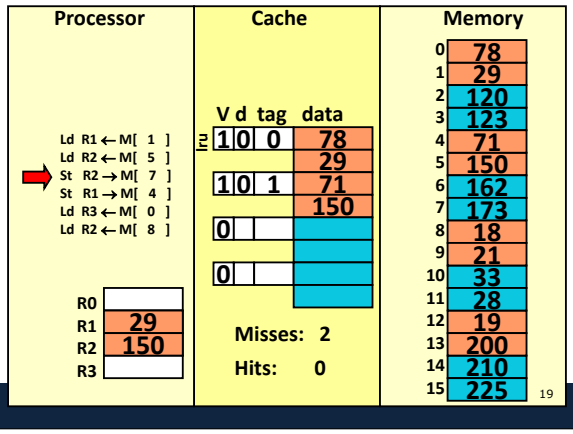
Agenda

- Set-associativity overview
- **Example**
- Class problem
- Integrating caches into our processor

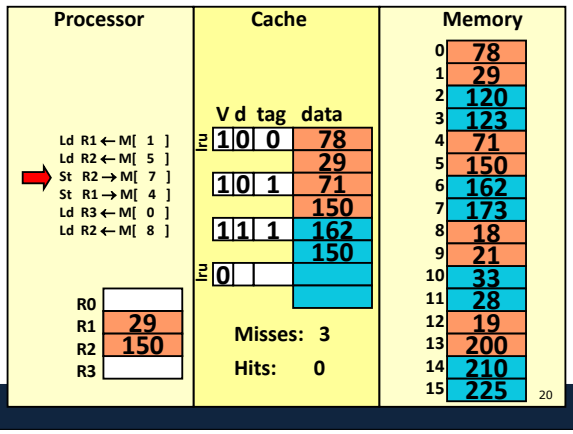
Set-associative cache (REF 2)



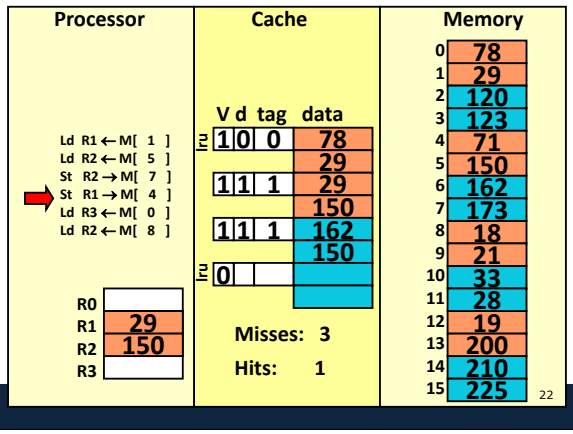
Set-associative cache (REF 3)



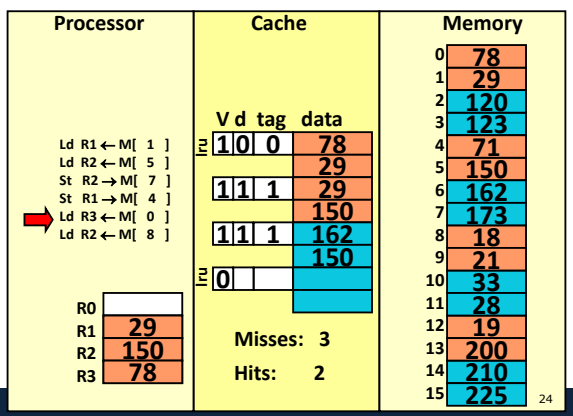
Set-associative cache (REF 3)



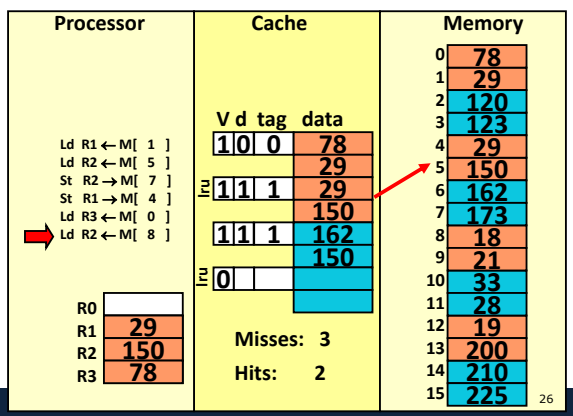
Set-associative cache (REF 4)



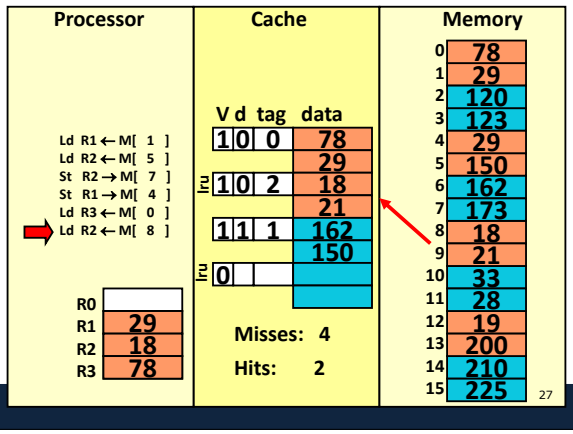
Set-associative cache (REF 5)



Set-associative cache (REF 6)



Set-associative cache (REF 6)



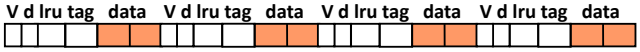
Agenda

- Set-associativity overview
- Example
- **Class problem**
- Integrating caches into our processor

Cache Organization Comparison

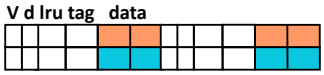
Block size = 2 bytes, total cache size = 8 bytes for all caches

1. Fully associative (4-way associative)



2. Direct mapped

3. 2-way associative



Class Problem 1

- For a 32-bit address and 16KB cache with 64-byte blocks, show the breakdown of the address for the following cache configuration:

A) fully associative cache

Block Offset = log2(64)=6 bits
Tag = 32 - 6 = 26 bits

B) 4-way set associative cache

Block Offset = 6 bits
#sets = #lines / ways = 64
Set Index = 6 bits
Tag = 32 - 6 - 6 = 20 bits

C) Direct-mapped cache

Block Offset = 6 bits
#lines = 256 Line Index = 8 bits
Tag = 32 – 6 – 8 = 18 bits

Agenda

- Set-associativity overview
- Example
- Class problem
- **Integrating caches into our processor**

Multi-Level Caches

- We've been considering proc -> cache -> memory
- This works well if working data set is <= size of cache
- But if data set is a little larger that cache, performance can plummet

Multi-Level Caches

- This is the motivation of multiple levels of caches
- L1 – smallest, fastest, closest to processor
- LN – biggest, slowest, closest to memory
- Allows for gradual performance degradation as data set size increases
- 3 levels of cache is pretty common in today's systems

What about cache for instructions

- We've been focusing on caching loads and stores (i.e. data)
- Instructions should be cached as well
- We have two choices:
  1. Treat instruction fetches as normal data and allocate cache lines when fetched
  2. Create a second cache (called the instruction cache or ICache) which caches instructions only
     - More common in practice

How do you know which cache to use?
What are advantages of a separate ICache?

Integrating Caches into Pipeline

- How are caches integrated into a pipelined implementation?
  - Replace instruction memory with Icache
  - Replace data memory with Dcache
- Issues:
  - Memory accesses now have variable latency
  - Both caches may miss at the same time

Next time

- How to properly choose cache parameters?
  - Start by classifying why misses occur