MEMORY SYSTEM

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

THE UNIVERSITY OF UTAH

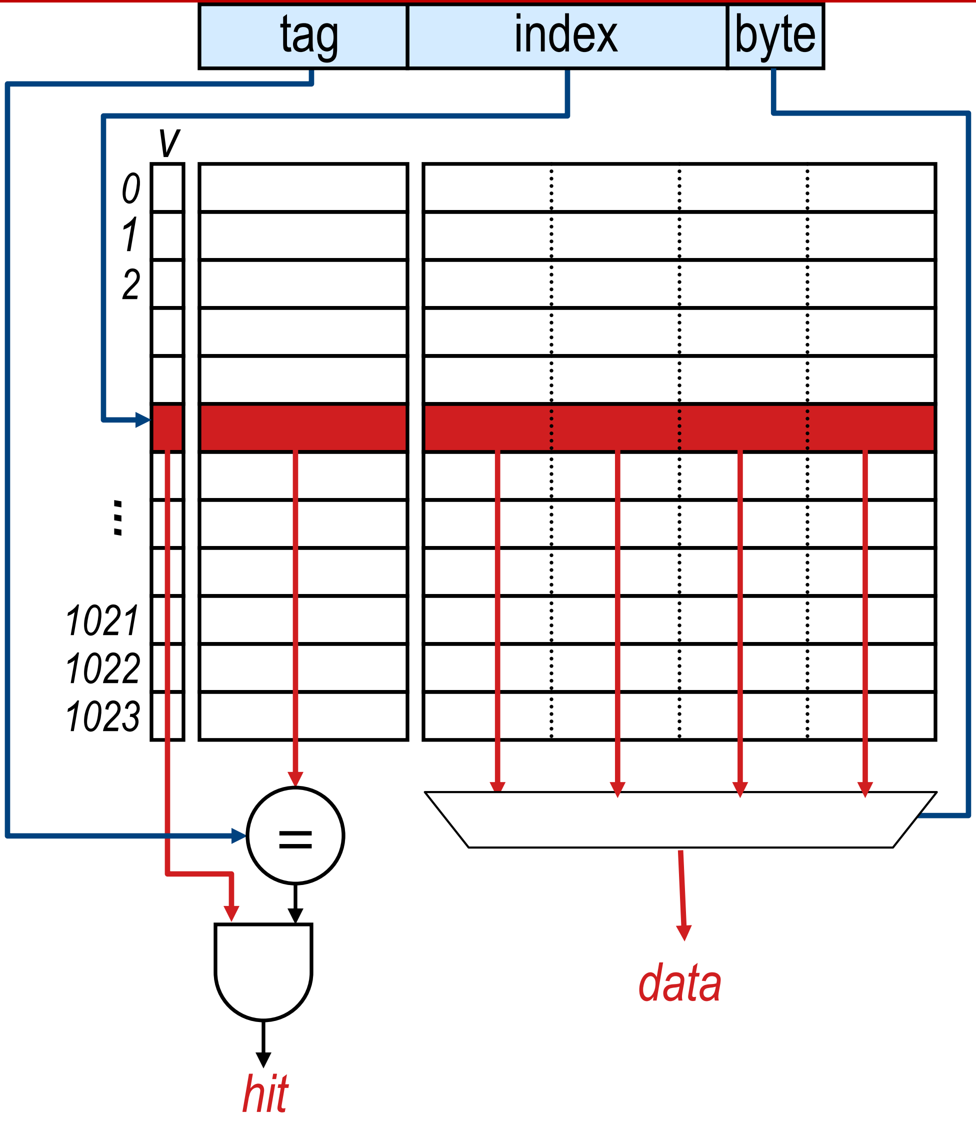
CS/ECE 3810: Computer Organization

Overview

- This lecture
  - Set-associative cache
  - Cache write policies

Recall: Direct-Mapped Lookup

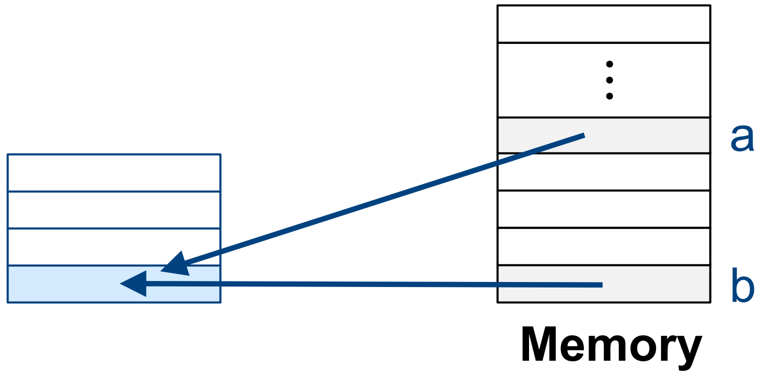
- Byte offset: to select the requested byte
- Tag: to maintain the address
- Valid flag (v): whether content is meaningful
- Data and tag are always accessed



Set Associative Caches

- Improve cache hit rate by allowing a memory location to be placed in more than one cache block
  - N-way set associative cache
  - Fully associative
- For fixed capacity, higher associativity typically leads to higher hit rates
  - more places to simultaneously map cache lines
  - 8-way SA close to FA in practice

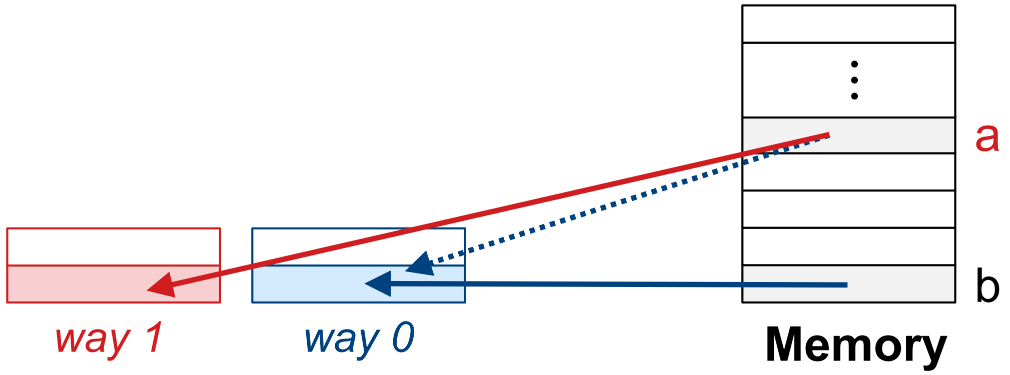
```
for (i=0; i<10000; i++) {
    a++;
    b++;
}
```



Set Associative Caches

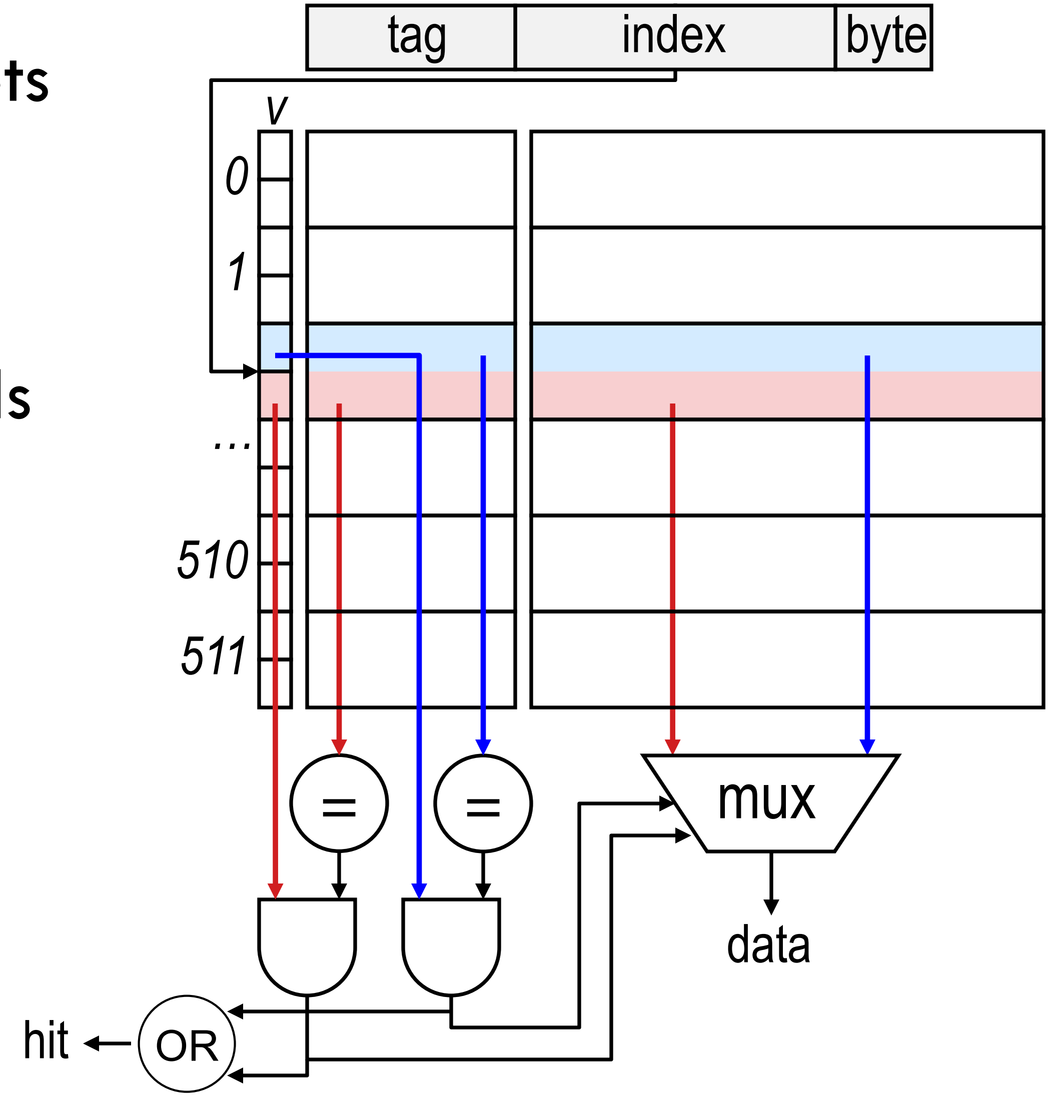
- Improve cache hit rate by allowing a memory location to be placed in more than one cache block
  - N-way set associative cache
  - Fully associative
- For fixed capacity, higher associativity typically leads to higher hit rates
  - more places to simultaneously map cache lines
  - 8-way SA close to FA in practice

```
for (i=0; i<10000; i++) {
    a++;
    b++;
}
```



n-Way Set Associative Lookup

- Index into cache sets
- Multiple tag comparisons
- Multiple data reads
- Special cases
  - Direct mapped
    - Single block sets
  - Fully associative
    - Single set cache



Example Problem

- Find the size of tag, index, and offset bits for an 4MB, 4-way set associative cache with 32B cache blocks. Assume that the processor can address up to 4GB of main memory.

Example Problem

- Find the size of tag, index, and offset bits for an 4MB, 4-way set associative cache with 32B cache blocks. Assume that the processor can address up to 4GB of main memory.
- 4GB = 2^{32} B → address bits = 32
- 32B = 2^{5} B → byte offset bits = 5
- 4MB/(4x32B) = 2^{15} → index bits = 15
- tag bits = 32 – 5 – 15 = 12

Example

- Consider a 32 kilobyte (KB) 4-way set-associative data cache array with 32-byte line sizes
  - How many sets?
  - How many index bits, offset bits, tag bits?
  - How large is the tag array?

Example

- Consider a 32 kilobyte (KB) 4-way set-associative data cache array with 32-byte line sizes
  - cache size = no. sets x no. ways x block size
  - How many sets?
  - How many index bits, offset bits, tag bits?
  - How large is the tag array?

Example

- Consider a 32 kilobyte (KB) 4-way set-associative data cache array with 32-byte line sizes
  - cache size = no. sets x no. ways x block size
  - How many sets?
  - no. sets = 32x1024 / (4 x 32) = 256
  - How many index bits, offset bits, tag bits?
  - 8 5 19
  - How large is the tag array?
  - no. sets x no. ways x tag bits = 256 x 4 x 19 = 19Kb

Example

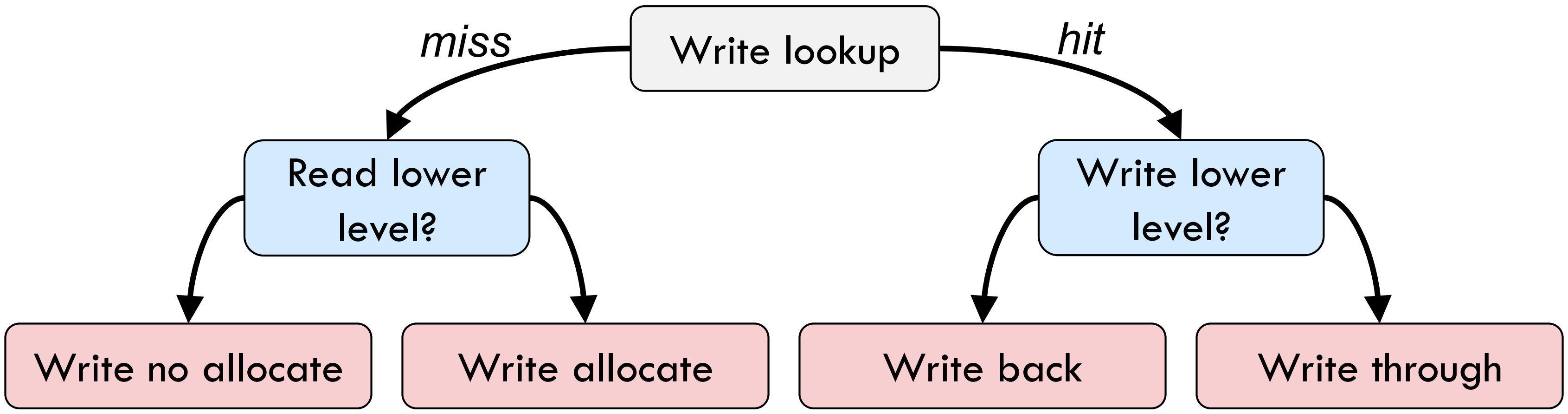
- A pipeline’s CPI is 1 if all loads/stores hit in cache
- Question: 40% of all instructions are loads/stores; 80% of all loads/stores hit in cache (1-cycle); memory access takes 100 cycles; what is the CPI?

Example

- A pipeline's CPI is 1 if all loads/stores hit in cache
- Question: 40% of all instructions are loads/stores; 80% of all loads/stores hit in cache (1-cycle); memory access takes 100 cycles; what is the CPI?
- Solution:
  - Consider 1000 instructions; 400 instructions are load/stores, of which 0.8x400 are hits (1 cycle) and 0.2x400 are misses (101 cycles).
  - CPI = (1x (600 + 320) + 101 x 80)/1000 = 9

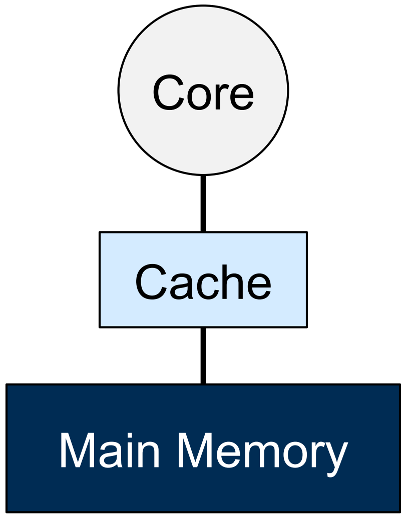
Cache Write Policies

- Write vs. read
  - Data and tag are accessed for both read and write
  - Only for write, data array needs to be updated
- Cache write policies



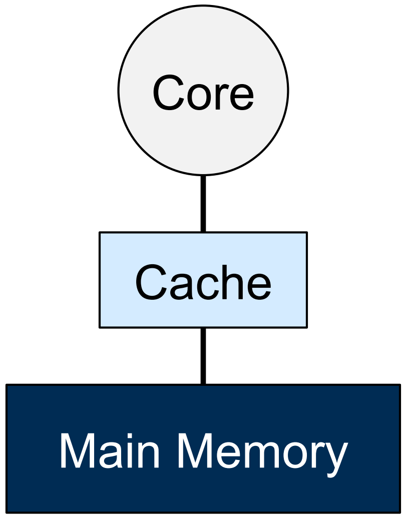
Write back

- On a write access, write to cache only
  - write cache block to memory only when replaced from cache
  - dramatically decreases bus bandwidth usage
  - keep a bit (called the *dirty* bit) per cache block



Write through

- Write to both cache and memory (or next level)
  - Improved miss penalty
  - More reliable because of maintaining two copies



Write (No-)Allocate

- *Write allocate*
  - allocate a cache line for the new data, and replace old line
  - just like a read miss

- *Write no allocate*
  - do not allocate space in the cache for the data
  - only really makes sense in systems with write buffers

- How to handle read miss after write miss?

Replacement Policy

- On a read miss, you always bring the block in (spatial and temporal locality) – but which block do you replace?
  - no choice for a direct-mapped cache
  - randomly pick one of the ways to replace
  - replace the way that was least-recently used (LRU)
  - FIFO replacement (round-robin)
- Which one is better?

Types of Cache Misses

- Compulsory misses: happens the first time a memory word is accessed – the misses for an infinite cache

- Capacity misses: happens because the program touched many other words before re-touching the same word – the misses for a fully-associative cache

- Conflict misses: happens because two words map to the same location in the cache – the misses generated while moving from a fully-associative to a direct-mapped cache