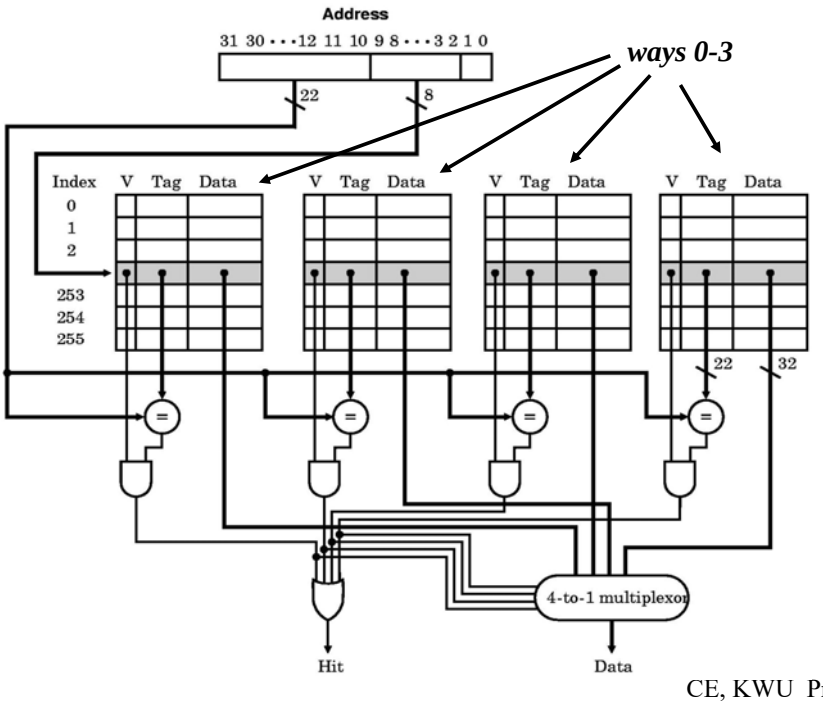
Chapter Five (2/3)

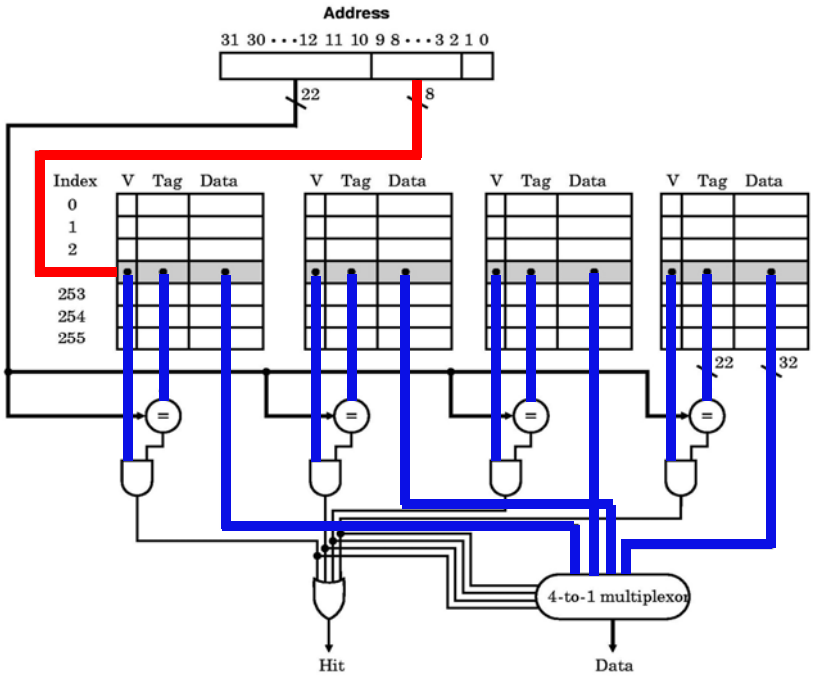
Set associative cache

- **Block placed in one *way* of set *index***
- **Number of sets = cache size/(block size*ways)**



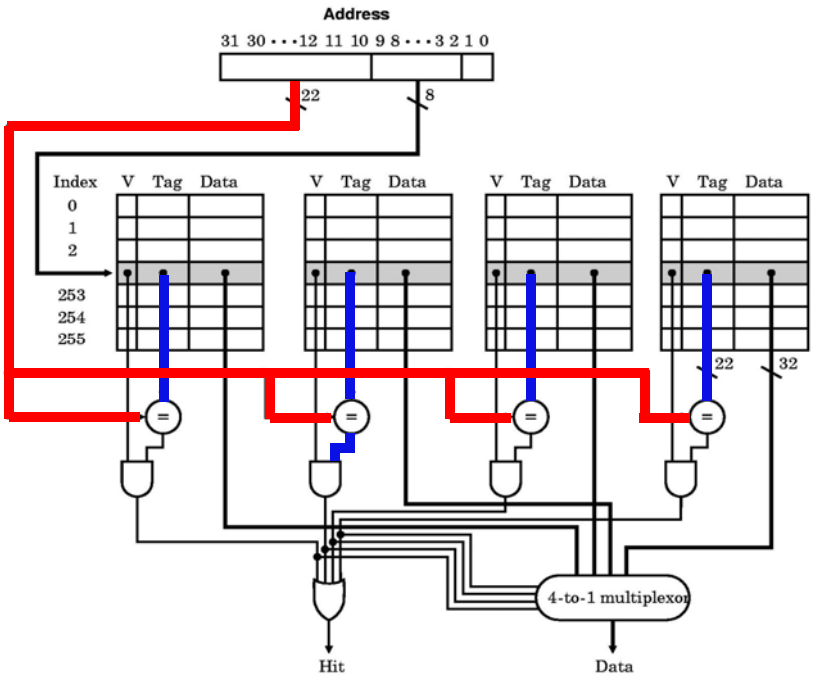
Set associative cache operation

- The index bits are used to select one of the sets
- The data, tag, and Valid bit from all ways of the selected entry are simultaneously accessed



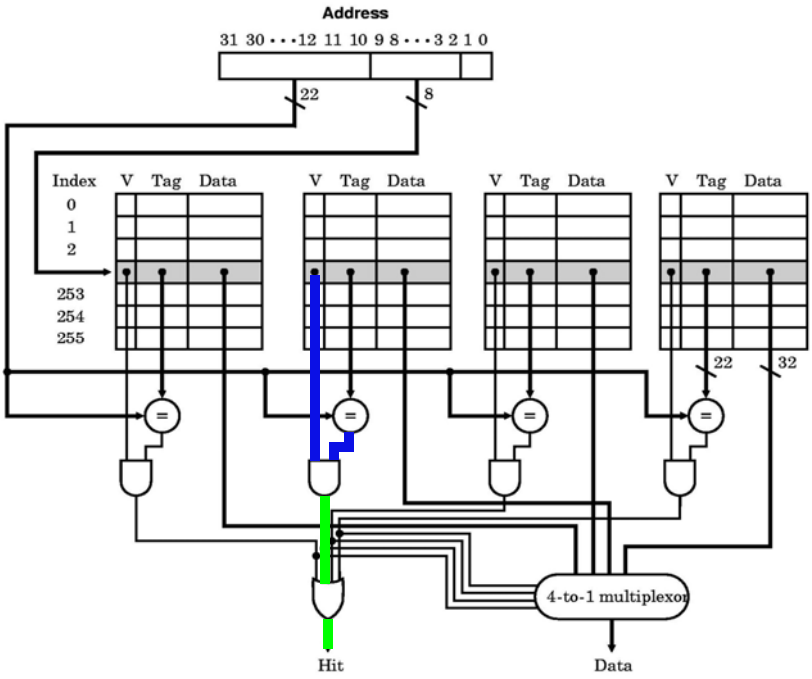
Set associative cache operation

- The tags from all ways of the selected entry are compared with the tag field of the address



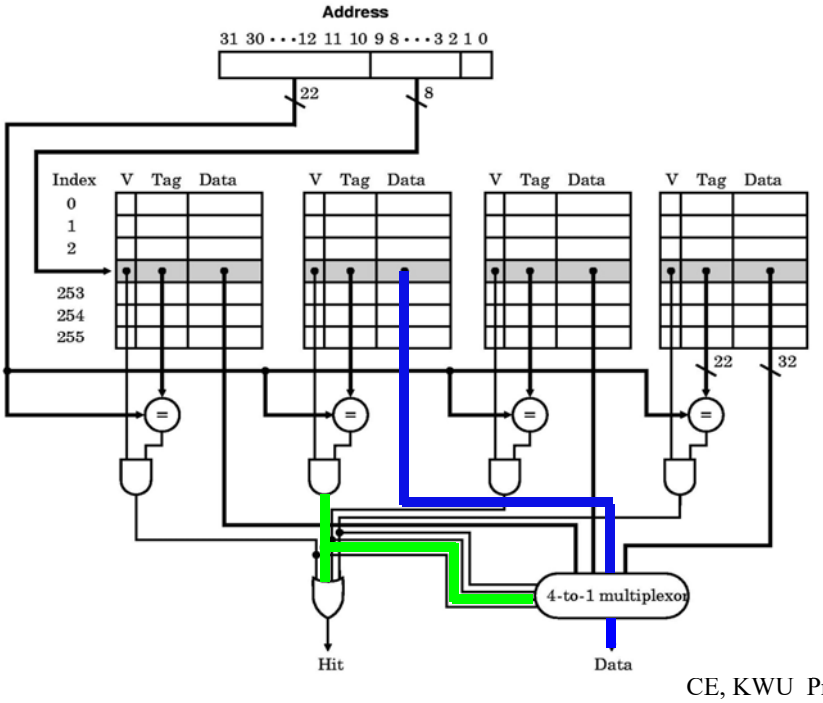
Set associative cache operation

- **A match between the tags and a Valid bit that is set indicates a cache hit (hit in way1 shown)**



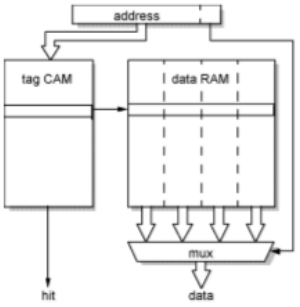
Set associative cache operation

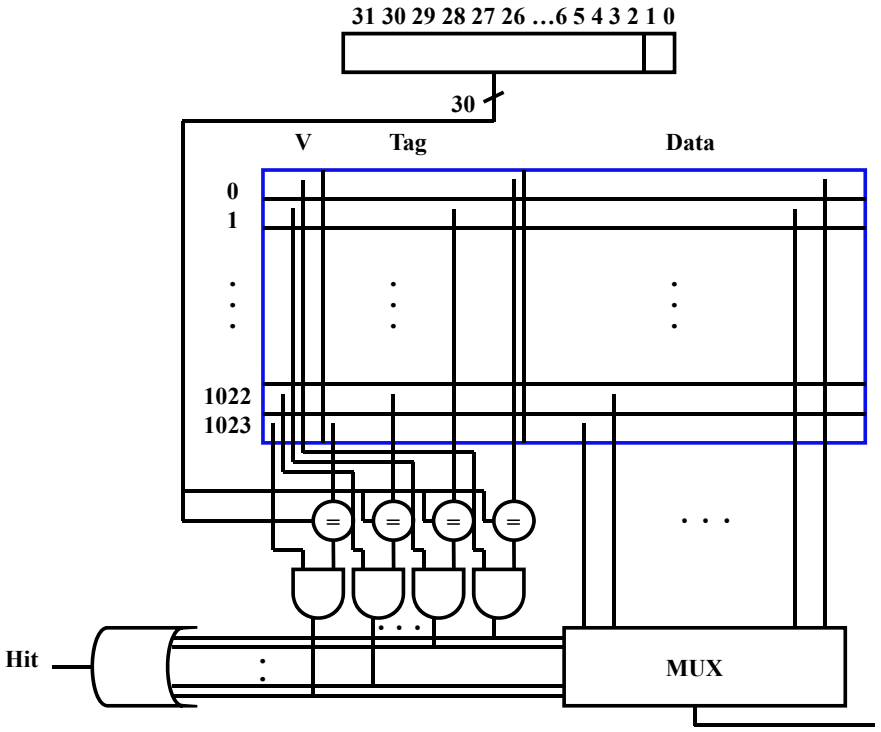
- **The data from the way that had a hit is returned through the MUX**



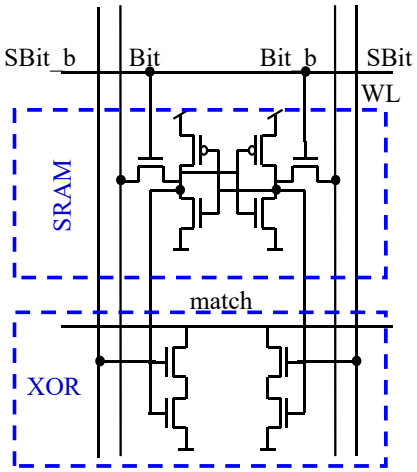
Fully associative cache

- **A block can be placed in any set**

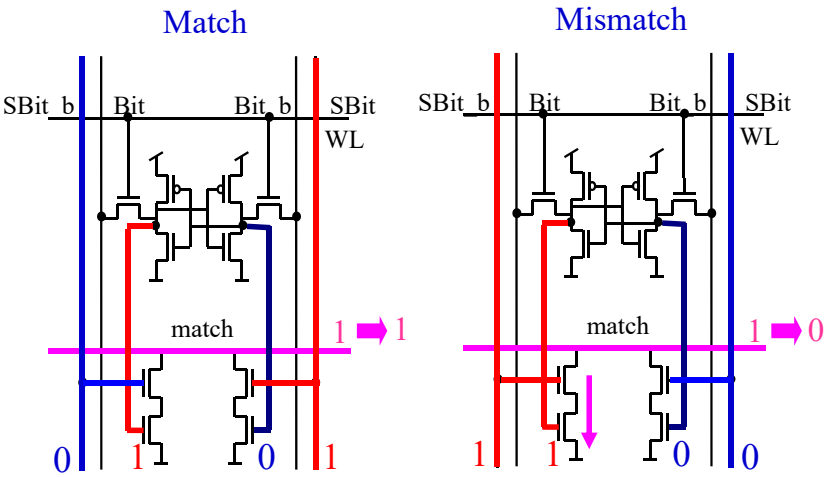




CAM Functionality

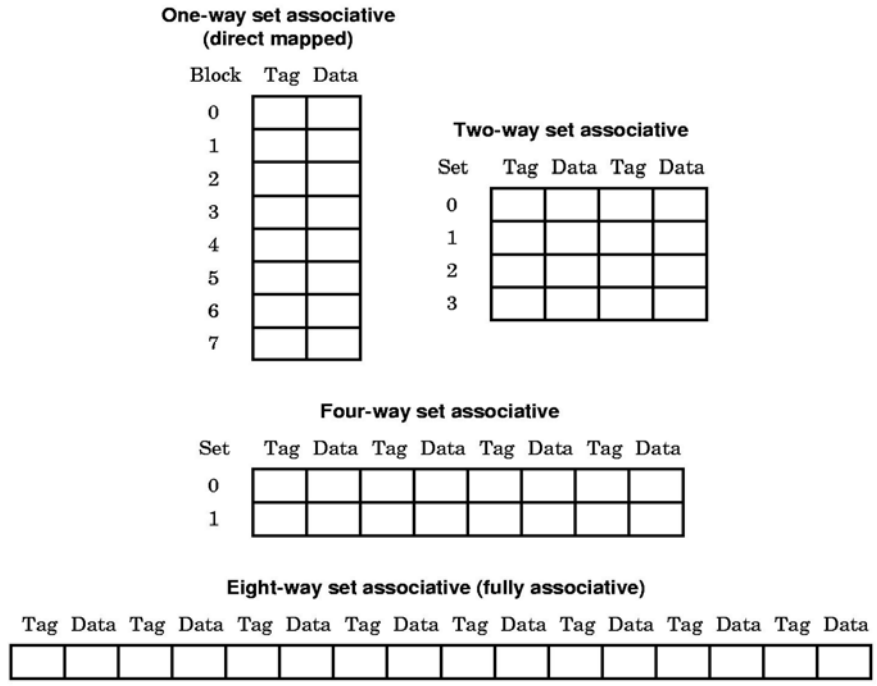


10-T CAM Cell
With Separate
Write/Search Lines
And Match Line



Different degrees of associativity

- Four different caches of size 8 blocks



Calculating Bits in Cache (1/2)

- How many total bits are needed for a direct-mapped cache with 64 KBytes of data and one word blocks, assuming a 32-bit address?
  - 64 Kbytes = 16 K words = 2^14 words = 2^14 blocks
  - block size = 4 bytes => offset size = 2 bits,
  - #sets = #blocks = 2^14 => index size = 14 bits
  - tag size = address size - index size - offset size = 32 - 14 - 2 = 16 bits
  - bits/block = data bits + tag bits + valid bit = 32 + 16 + 1 = 49
  - bits in cache = #blocks x bits/block = 2^14 x 49 = 98 Kbytes
- How many total bits would be needed for a 4-way set associative cache to store the same amount of data
  - block size and #blocks does not change
  - #sets = #blocks/4 = (2^14)/4 = 2^12 => index size = 12 bits
  - tag size = address size - index size - offset = 32 - 12 - 2 = 18 bits
  - bits/block = data bits + tag bits + valid bit = 32 + 18 + 1 = 51
  - bits in cache = #blocks x bits/block = 2^14 x 51 = 102 Kbytes

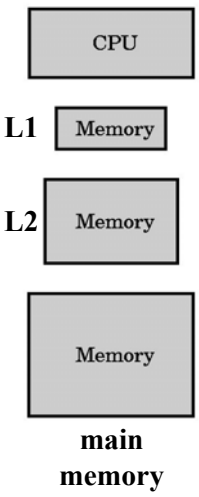
- Increase associativity ➔ increase bits in cache

Calculating Bits in Cache (2/2)

- How many total bits are needed for a direct-mapped cache with 64 KBytes of data and 8 word blocks, assuming a 32-bit address?
  - 64 Kbytes = 2^14 words = (2^14)/8 = 2^11 blocks
  - block size = 32 bytes => offset size = 5 bits,
  - #sets = #blocks = 2^11 => index size = 11 bits
  - tag size = address size - index size - offset size = 32 - 11 - 5 = 16 bits
  - bits/block = data bits + tag bits + valid bit = 8x32 + 16 + 1 = 273 bits
  - bits in cache = #blocks x bits/block = 2^11 x 273 = 68.25 Kbytes

- Increase block size ➔ decrease bits in cache

Q3: Which Block Should be Replaced on a Miss?



- A cache miss occurs when the block is not found in the cache
- The block is requested from the next level of the hierarchy
- When the block returns, it is loaded into the cache and provided to the requester
- A copy of the block remains in the lower levels of the hierarchy
- The *cache miss rate* is found by dividing the total number of misses by the total number of accesses (misses/accesses)
  - The hit rate is 1-miss rate

Block replacement policy

- Determines what block to replace on a cache miss to make room for the new block
- Least recently used (LRU)
  - Pick the one that has been unused for the longest time
  - Based on temporal locality
  - Requires ordering bits to be kept with each set
  - Too expensive beyond 4-way
- Random
  - Pseudo-randomly pick a block
  - Generally not as effective as LRU (higher miss rates)
  - Simple even for highly associative organizations
- Most recently used (MRU)
  - Keep track of which block was accessed last
  - Randomly pick a block from other than that one
  - Compromise between LRU and random

Replacement on a Miss

- Easy for Direct Mapped - only on choice
- Set Associative or Fully Associative:
  - Random - easier to implement
  - Least Recently used - harder to implement
- Miss rates for caches with different size, associativity and replacemnt algorithm.

Associativity:		2-way		4-way		8-way
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

- For caches with low miss rates, random is almost as good as LRU.

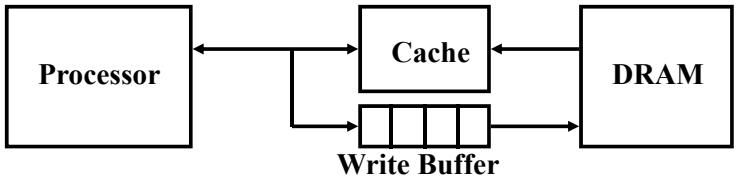
Q4: What Happens on a Write?

- **The L1 data cache needs to handle writes (stores) in addition to reads (loads) → Need to check for a hit before writing**
  - **Write through: The information is written to both the block in the cache and to the block in the lower-level memory.**
  - **Write back: The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.**
    - is block clean or dirty? (add a dirty bit to each block)
- **Pros and Cons of each:**
  - **Write through**
    - Read misses cannot result in writes to memory,
    - Easier to implement
    - Always combine with write buffers to avoid memory latency
  - **Write back**
    - Less memory traffic
    - Perform writes at the speed of the cache

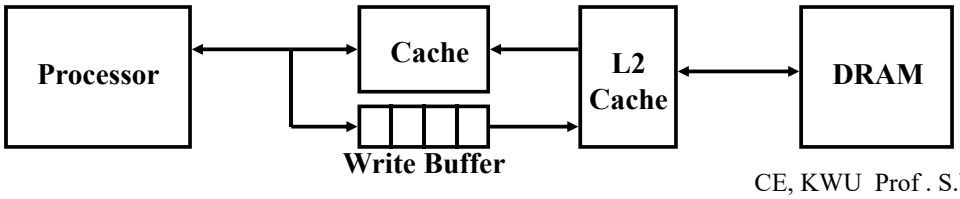
Cache Replacement on Write

- **Since data does not have to be brought into the cache on a write miss, there are two options:**
- **Write allocate**
  - **The block is brought into the cache on a write miss**
  - **Used with write-back caches**
  - **Hope subsequent writes to the block hit in cache**
- **No-write allocate**
  - **The block is modified in memory, but not brought into the cach**
  - **Used with write-through caches**
  - **Writes have to go to memory anyway, so why bring the block into the cache**

Write Buffer for Write Through



- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO: typical number of entries = 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- Memory system designer's nightmare: Write buffer saturation
  - Store frequency (w.r.t. time) → 1 / DRAM write cycle
  - The CPU Cycle Time ← DRAM Write Cycle Time
- Solution for write buffer saturation: write back cache and/or L2 cache

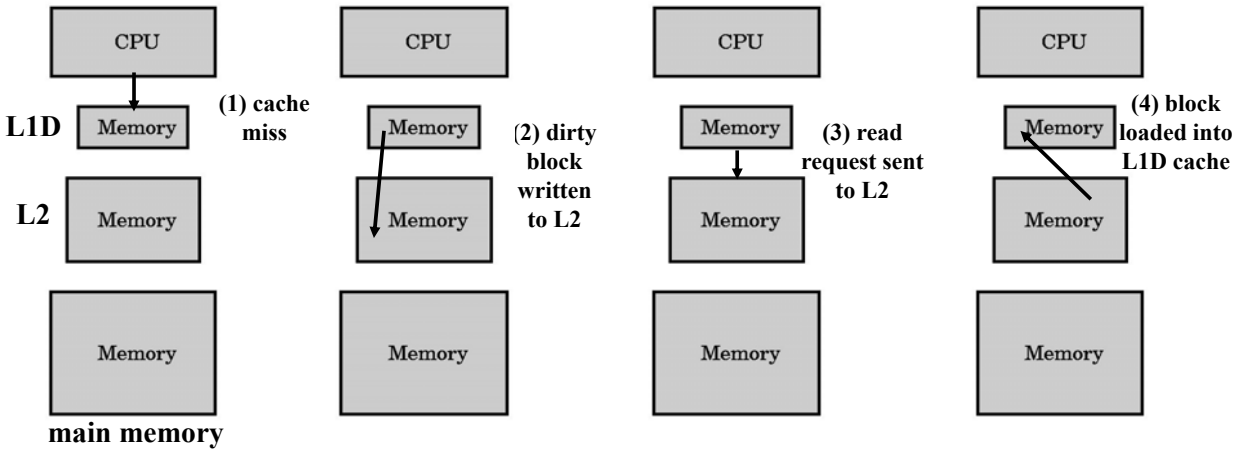


Write-Back Cache Operation

- Write Operation
  - Check for hit
  - If a hit, write the byte, halfword, or word to the correct location in the block
  - If a miss, request the block from the next level in the hierarchy
  - Load the block into the cache, and then perform the write

- Replacement Operation
  - A *dirty bit* is asociated with each cache block
  - The dirty bit is set if the block is written to
  - A block with a set dirty bit that is chosen for replacement is written to the next level before being overwritten with the new block

Cache writes and block replacement



- Problem: writing the block to the next level interferes with the read request at that level
  - Send the read request to L2
  - Put the dirty block in a temporary hardware buffer (*victim buffer*)
  - Place the new block in L1 when it returns
  - Write the dirty block to L2 from the victim buffer

Cache Performance Measures

- Hit rate: fraction found in the cache
  - So high that we usually talk about Miss rate = 1 - Hit Rate
- Hit time: time to access the cache
- Miss penalty: time to replace a block from lower level, including time to replace in CPU
  - access time: time to acccess lower level
  - transfer time: time to transfer block
- Average Memory-Access Time:

  AMAT = Hit time + Miss rate x Miss penalty (ns or clocks)

  - If a direct mapped cache has a hit rate of 95%, a hit time of 4 ns, and a miss penalty of 100 ns, what is the AMAT?

    AMAT = 4 + 0.05 x 100 = 9 ns
  - If replacing the cache with a 2-way set associative increases the hit rate to 97%, but increases the hit time to 5 ns, what is the new AMAT?

    AMAT = 5 + 0.03 x 100 = 8 ns

Split vs. Unified Cache

- Unified cache (mixed cache): Data and instructions are stored together (von Neuman architecture)
- Split cache: Data and instructions are stored separately (Harvard architecture)
- Why do instructions caches have a lower miss ratio?

Size	Instruction Cache	Data Cache	Unified Cache
1 KB	3.06%	24.61%	13.34%
2 KB	2.26%	20.57%	9.78%
4 KB	1.78%	15.94%	7.24%
8 KB	1.10%	10.19%	4.57%
16 KB	0.64%	6.47%	2.87%
32 KB	0.39%	4.82%	1.99%
64 KB	0.15%	3.77%	1.35%
128 KB	0.02%	2.88%	0.95%

Example: Split vs. Unified Cache

- Which has the lower average memeory access time?
  - Split cache : 16 KB instructions + 16 KB data
  - Unified cache: 32 KB instructions + data
- Assumptions
  - Use miss rates from previous chart
  - 1 out of 3 instructions is a memory access instruction
  - Hit time is 1 cycle and Miss penalty is 50 cycles
  - On the unified cache, a load or store hit takes an extra cycle, since there is only one port for instructions and data
- Average memory-access time = Hit time + Miss rate x Miss penalty

  AMAT = %instr x (instr hit time + instr miss rate x instr miss penalty) +
  %data x (data hit time + data miss rate x data miss penalty)

  For the split cache:

  AMAT = 75% x (1 + 0.64%x 50) + 25% (1 + 6.47% x 50) = 2.05

  For the unified cache

  AMAT = 75% x (1 + 1.99%x 50) + 25% x (2 + 1.99% x 50) = 2.24

  → The unified cache has a longer AMAT, even though its miss rate is lower, due to conflicts for instruction and data hazards.

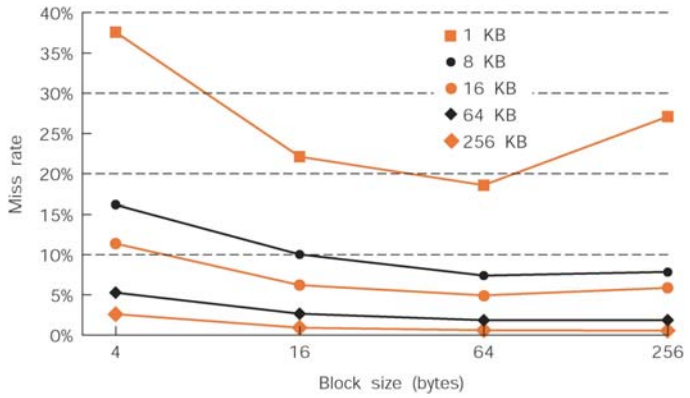
Increasing Cache Size

- Classifying Misses: 3 Cs
  - Compulsory — The first access to a block is not in the cache, so the block must be brought into the cache.
  - Capacity — If the cache cannot contain all the blocks needed during execution of a program, capacity misses will occur due to blocks being discarded and later retrieved.
  - Conflict — If the block-placement strategy is set associative or direct mapped, conflict misses will occur because a block can be discarded and later retrieved if too many blocks map to its set.

- One way to decrease misses is to increase the cache size
  - Reduces capacity and conflict misses
  - No effect on compulsory misses
- However a larger cache may increase the hit time
  - larger cache => larger access time
  - If cache is to large, can't fit it on the same chip as processor.

Increasing Block Size

- Another way to reduce the miss rate is to increase the block size
  - Take advantage of spacial locality
  - Decreases miss rate including compulsory misses
- However, larger blocks have disadvantages
  - May increase the miss penalty (need to get more data)
  - May increase hit time (need to read more data from cache)
  - May increase miss rate due to conflict misses
- As the block size increases AMAT decreases, but eventually increases



Increasing Associativity

- Increasing associativity helps reduce conflict misses

- 2:1 Cache Rule:
  - The miss rate of a direct mapped cache of size N is about equal to the miss rate of a 2-way set associative cache of size N/2
  - For example, the miss rate of a 32 Kbyte direct mapped cache is about equal to the miss rate of a 16 Kbyte 2-way set associative cache

- Disadvantages of higher associativity
  - Need to do large number of comparisons
  - Need n-to-1 multiplexor for n-way set associative
  - Could increase hit time

AMAT vs. Associativity

Cache Size	Associativity			
(KB)	1-way	2-way	4-way	8-way
1	7.65	6.60	6.22	5.44
2	5.90	4.90	4.62	4.09
4	4.60	3.95	3.57	3.19
8	3.30	3.00	2.87	2.59
16	2.45	2.20	2.12	2.04
32	2.00	1.80	1.77	1.79
64	1.70	1.60	1.57	1.59
128	1.50	1.45	1.42	1.44

- Red means A.M.A.T. not improved by more associativity
- Does not take into account effect of slower clock on rest of program

Using a 2nd Level Cache

- A second level (L2) cache reduces the miss penalty by providing a large cache between the first level (L1) cahe and main memory

- L2 Equations

  $AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times Miss\ Penalty_{L1}$

  $Miss\ Penalty_{L1} = Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2}$

  $AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times (Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2})$

- If a direct mapped cache has a hit rate of 95%, a hit time of 4 ns, and a miss penalty of 100 ns, what is the AMAT?

  $AMAT = Hit\ time + Miss\ rate \times Miss\ penalty = 4 + 0.05 \times 100 = 9\ ns$

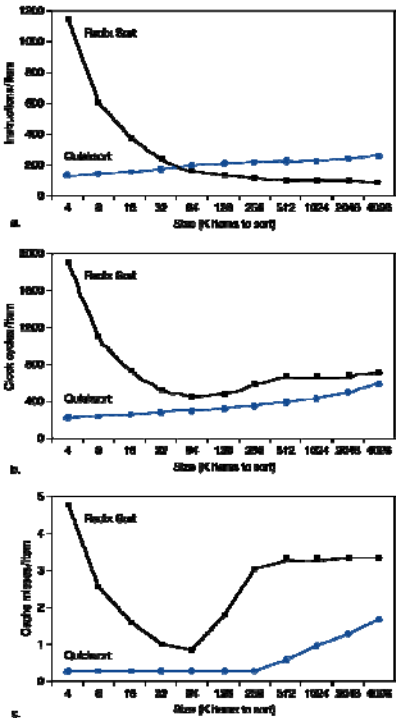
- If an L2 cache is added with a hit time of 20 ns and a hit rate of 50%, what is the new AMAT?

  $AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times (Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2})$

  $= 4 + 0.05 \times (20 + 0.5 \times 100) = 7.5\ ns$

Interactions with Advanced CPUs

- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyze
  - Use system simulation
- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access



Cache Coherence Problem

- **Suppose two CPU cores share a physical address space**
  - **Write-through caches**

Time step	Event	CPU A's cache	CPU B's cache	Memory
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0	1

Coherence Defined

- **Informally: Reads return most recently written value**
- **Formally:**
  - **P writes X; P reads X (no intervening writes)**
    ⇒ read returns written value
  - **P_1 writes X; P_2 reads X (sufficiently later)**
    ⇒ read returns written value
    - c.f. CPU B reading X after step 3 in example
  - **P_1 writes X, P_2 writes X**
    ⇒ all processors see writes in the same order
    - End up with the same final value for X

Cache Coherence Protocols

- **Operations performed by caches in multiprocessors to ensure coherence**
  - **Migration of data to local caches**
    - Reduces bandwidth for shared memory
  - **Replication of read-shared data**
    - Reduces contention for access
- **Snooping protocols**
  - **Each cache monitors bus reads/writes**
- **Directory-based protocols**
  - **Caches and memory record sharing status of blocks in a directory**

Invalidating Snooping Protocols

- **Cache gets exclusive access to a block when it is to be written**
  - **Broadcasts an invalidate message on the bus**
  - **Subsequent read in another cache misses**
    - Owning cache supplies updated value

CPU activity	Bus activity	CPU A's cache	CPU B's cache	Memory
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Invalidate for X	1		0
CPU B read X	Cache miss for X	1	1	1

Memory Consistency

- When are writes seen by other processors
  - "Seen" means a read returns the written value
  - Can't be instantaneously
- Assumptions
  - A write completes only when all processors have seen it
  - A processor does not reorder writes with other accesses
- Consequence
  - P writes X then writes Y
    ⇒ all processors that see new Y also see new X
  - Processors can reorder reads, but not writes

Cache Performance Summary

- AMAT = Hit time + Miss rate x Miss penalty
- Split vs. Unified Cache
- 3 C's of misses
  - Compulsory
  - Capacity
  - Conflict
- Methods for improving performance
  - increase (change) cache size
  - increase (change) block size
  - increase (change) associativity
  - add a 2nd level cache