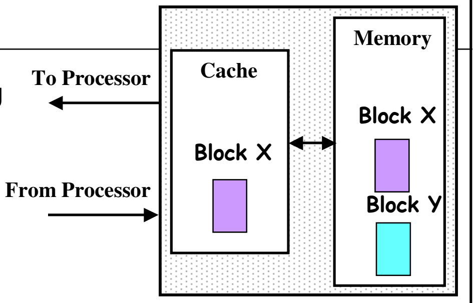
ECE232: Hardware Organization and Design

Part 15: Cache
Chapter 5 (4th edition), 7 (3rd edition)

http://www.ecs.umass.edu/ece/ece232/

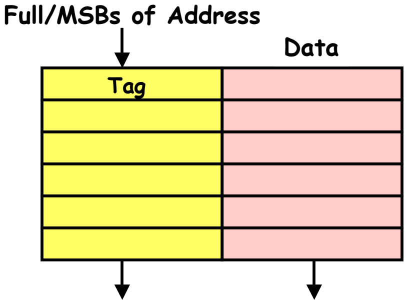
Adapted from *Computer Organization and Design*, Patterson & Hennessy, UCB

Cache addressing



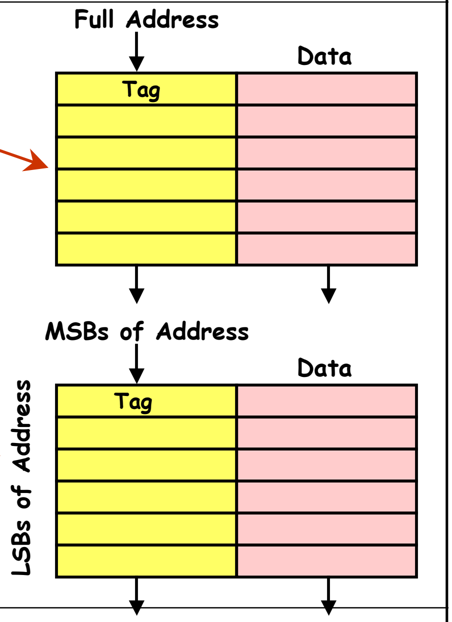
- How do you know if something is in the cache? (Q1)
- If it is in the cache, how to find it? (Q2)

- Traditional Memory
  - Given an address, provide the data (has address decoder)
- Associative Memory
  - AKA "Content Addressable Memory"
  - Each line contain the address (or part of it) and the data



Cache Organization

- Fully-associative: any memory location can be stored anywhere in the cache
  - Cache location and memory address are unrelated
- Direct-mapped: each memory location maps onto exactly one cache entry
  - Some of the memory address bit are used to index the cache
- N-way set-associative: each memory location can go into one of N sets

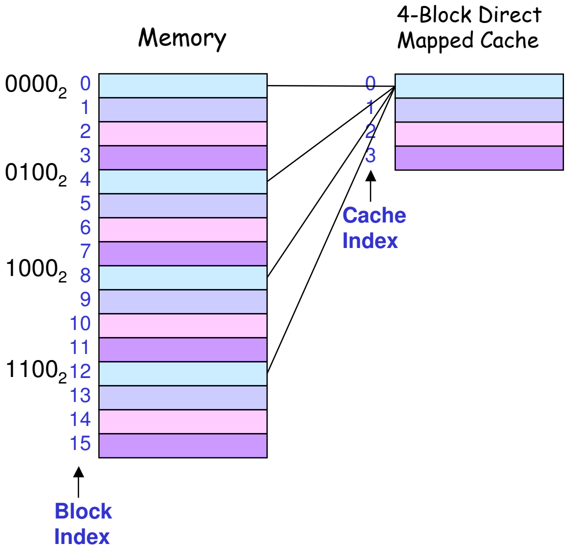


Direct Mapped Cache

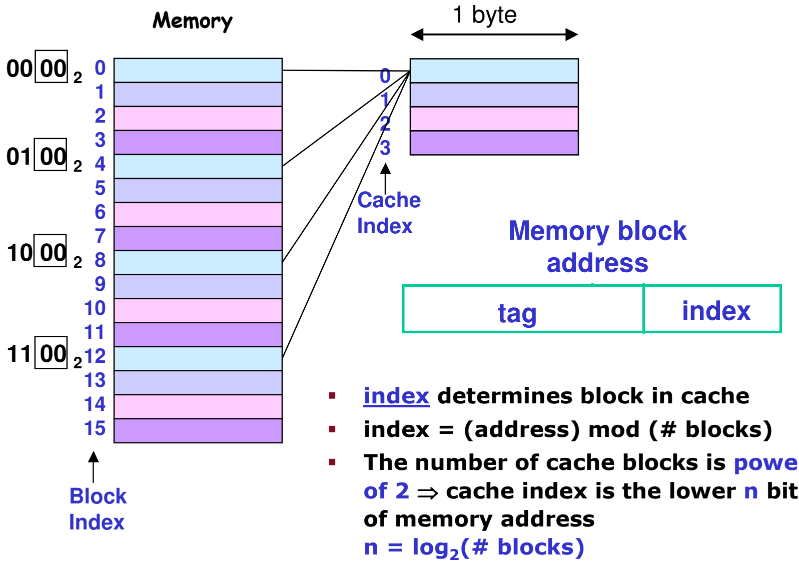
- Simplest mapping is a direct mapped cache
- Each memory address is associated with one possible block within the cache
  - Therefore, we only need to look in a single location in the cache for the data if it exists in the cache

Direct mapped cache (assume 1 byte/Block)

- Cache Block 0 can be occupied by data from
  - Memory blocks 0, 4, 8, 12
- Cache Block 1 can be occupied by data from
  - Memory blocks 1, 5, 9, 13
- Cache Block 2 can be occupied by data from
  - Memory blocks 2, 6, 10, 14
- Cache Block 3 can be occupied by data from
  - Memory blocks 3, 7, 11, 15

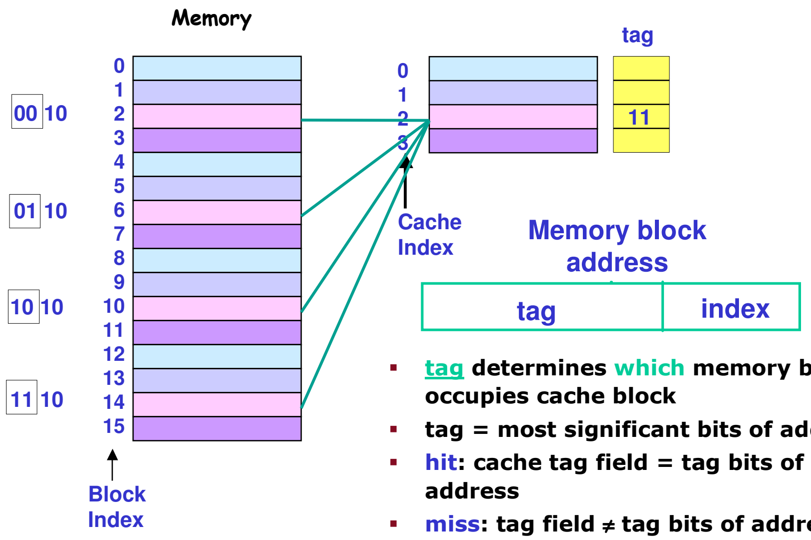


Direct Mapped Cache – Index and Tag



- index determines block in cache
- index = (address) mod (# blocks)
- The number of cache blocks is power of 2 ⇒ cache index is the lower n bits of memory address
  $n = \log_2(\# \text{ blocks})$

Direct Mapped w/Tag



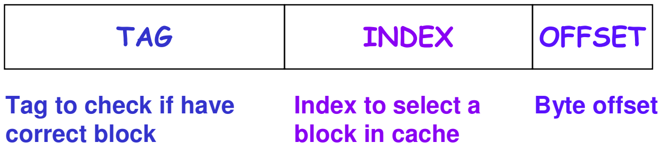
- tag determines which memory block occupies cache block
- tag = most significant bits of address
- hit: cache tag field = tag bits of address
- miss: tag field ≠ tag bits of address

Finding Item within Block

- In reality, a cache block consists of a number of bytes/words to
  (1) increase cache hit due to locality property and
  (2) reduce the cache miss time
- Mapping: memory block ***i*** is mapped to cache block with index ***i mod k***, where ***k*** is the number of blocks in the cache
- Given an address of item, index tells which block of cache to look in
- Then, how to find requested item within the cache block?
- Or, equivalently, “What is the byte offset of the item within the cache block?”

Selecting part of a block

- If block size > 1, rightmost bits of index are really the **offset** within the indexed block



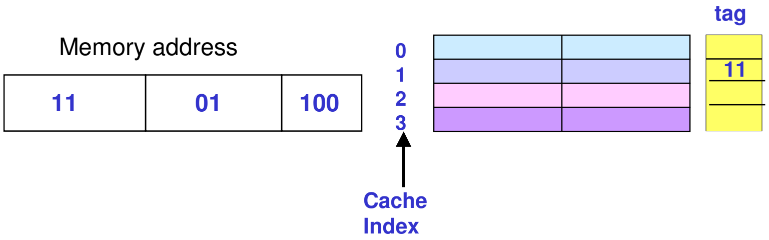
TAG | INDEX | OFFSET

Tag to check if have correct block

Index to select a block in cache

Byte offset

- Example: Block size of 8 bytes; select 2nd word

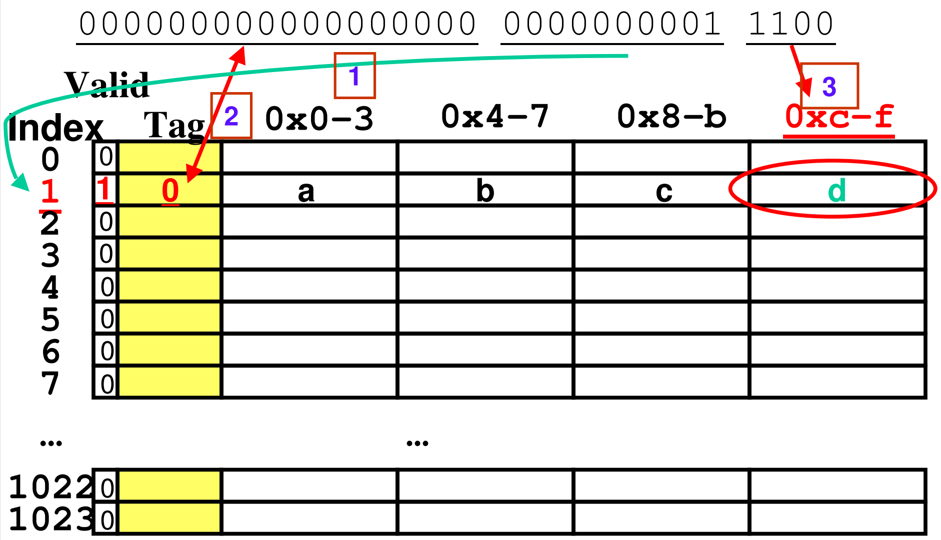


Accessing data in a direct mapped cache

- Three types of events:
- cache hit: cache block is valid and contains proper address, so read desired word
- cache miss: nothing in cache in appropriate block, so fetch from memory
- cache miss, block replacement: wrong data is in cache at appropriate block, so discard it and fetch desired data from memory
- Cache Access Procedure: (1) Use Index bits to select cache block (2) If valid bit is 1, compare the tag bits of the address with the cache block tag bits (3) If they match, use the offset to read out the word/byte

Data valid, tag OK, so read offset return word d

000000000000000000 0000000001 1100



Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

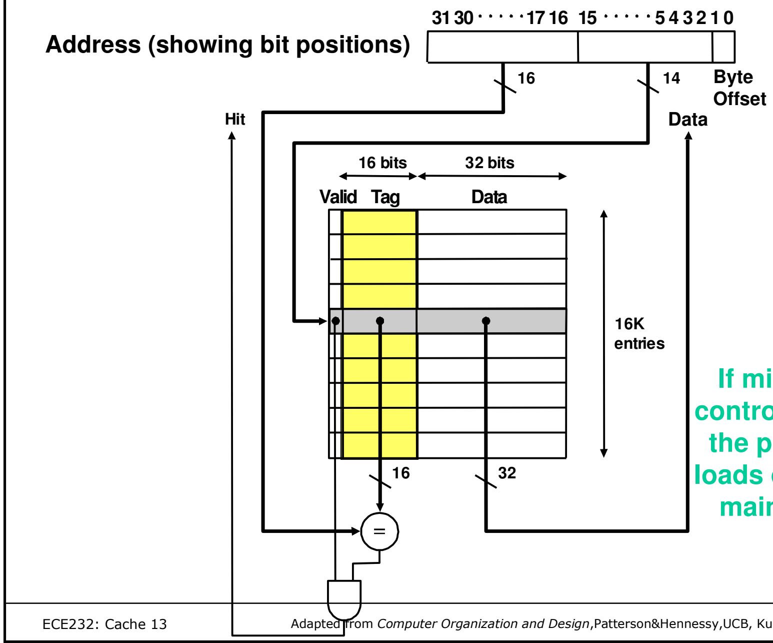
An Example Cache: DecStation 3100

- Commercial Workstation: ~1985
- MIPS R2000 Processor
- Separate instruction and data caches:
  - direct mapped
  - 64K Bytes (16K words) each
  - Block Size: 1 Word (Low Spatial Locality)

  Solution:

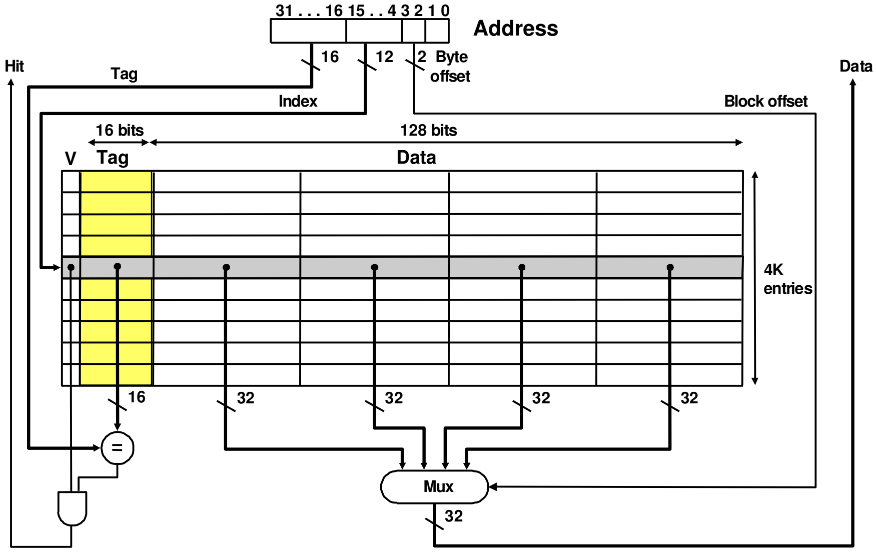
  Increase block size – 2nd example

DecStation 3100 Cache (Block size 1 word)

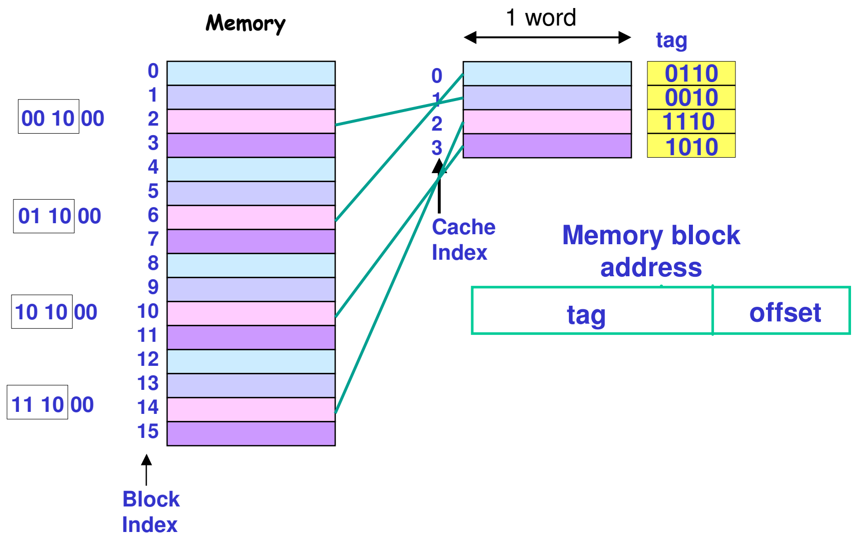


If miss, cache controller stalls the processor, loads data from main memory

64KB Cache with 4-word (16-byte) blocks



Fully Associative Cache



Memory

00 10 00

01 10 00

10 10 00

11 10 00

Block Index

1 word

tag

0110
0010
1110
1010

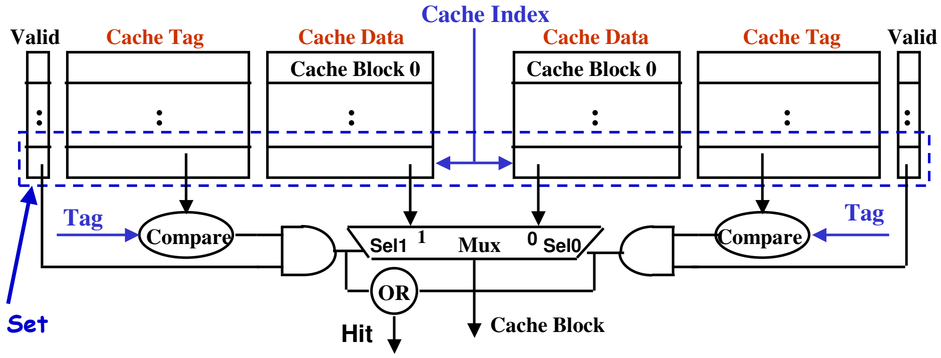
Cache Index

Memory block address

tag | offset

Two-way Set Associative Cache

- Two direct-mapped caches operate in parallel
- Cache Index selects a “set” from the cache (set includes 2 blocks)
- The two tags in the set are compared in parallel
- Data is selected based on the tag result



Cache Index

Valid

Cache Tag

Cache Data

Cache Block 0

Cache Data

Cache Block 0

Cache Tag

Valid

Tag

Compare

Sel1 1 Mux 0 Sel0

Compare

Tag

Set

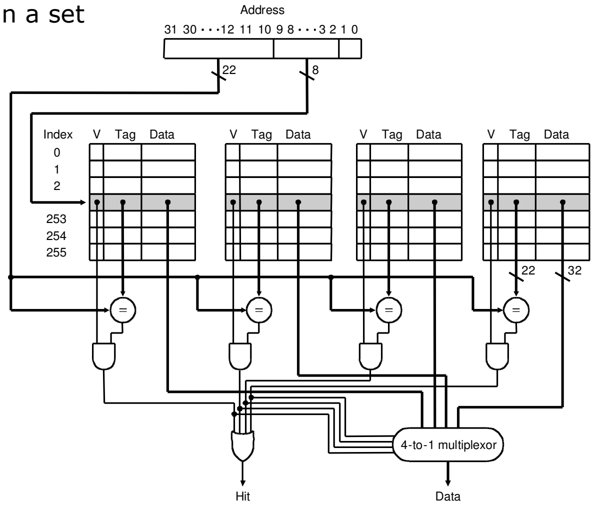
OR

Hit

Cache Block

4-way Set Associative Cache

- Allow block <u>anywhere</u> in a set
- Advantages:
  - Better hit rate
- Disadvantage:
  - More tag bits
  - More hardware
  - Higher access time



A Four-Way Set-Associative Cache, Block size = 4 bytes

Set Associative Cache - addressing

TAG	INDEX/Set #	OFFSET

Tag to check if have correct block anywhere in set

Index to select a set in cache

Byte offset

Types of Cache Misses (for 3 organizations)

- Compulsory (cold start): location has never been accessed - first access to a block not in the cache
- Capacity: since the cache cannot contain all the blocks of a program, some blocks will be replaced and later retrieved
- Conflict: when too many blocks try to load into the same set, some blocks will be replaced and later retrieved

Cache Design Decisions

- For a given cache size
  - Block (Line) size
    - Number of Blocks (Lines)
  - How is the cache organized
  - Write policy
  - Replacement Strategy

- Increase cache size
  - More Blocks (Lines)
  - More lines == Higher hit rate
  - Slower Memory
  - As many as practical

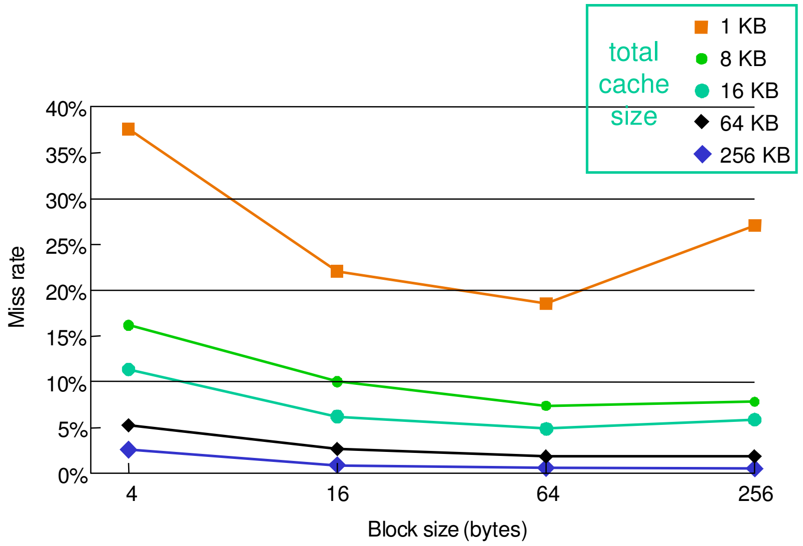
Block size

- Miss rate goes down with block size (why?)
- Extreme Example: choose block size = cache size.
  - only one block in cache
- Temporal Locality says if an item is accessed, it is likely to be accessed again soon
  - But it is unlikely that it will be accessed again immediately!!!
  - The next access is likely to be a miss
    - Continually loading data into the cache but forced to discard them before they are used again
    - Worst nightmare of a cache designer: Ping Pong Effect

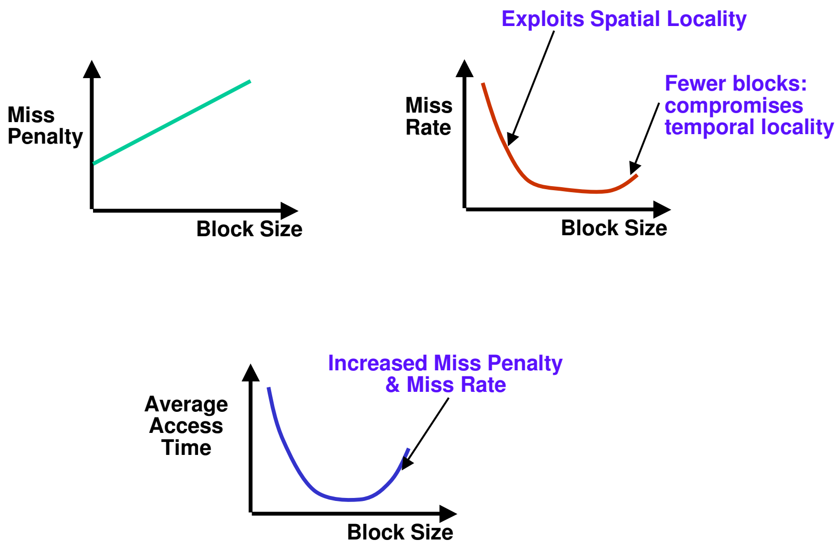
Block Size and Miss Penalty

- With increase in block size, the cost of a miss also increases
- Miss penalty: time to fetch the block from the next lower level of the hierarchy and load it into the cache
- With very large blocks, increase in miss penalty overwhelms decrease in miss rate
- Can minimize average access time if design memory system right

Miss Rate Versus Block Size



Block Size Tradeoff



Exploits Spatial Locality

Fewer blocks: compromises temporal locality

Increased Miss Penalty & Miss Rate

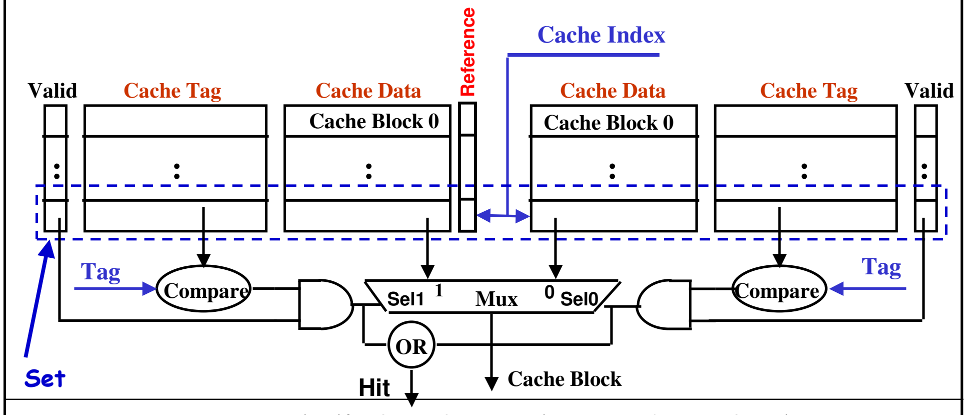
Writing to the Cache and Block Replacement

- Need to keep cache consistent with memory
  - Write to cache & memory simultaneously: "Write-through"
  - Or: Write to cache and mark as 'dirty'
    - Need to eventually copy back to memory: "Write-back"

- Need to make space in cache for a new entry
- Which Line Should be 'Evicted' (Q3)
  - Ideal?: Longest Time Till Next Access
  - Least-recently used
    - Complicated
  - Random selection
    - Simple
  - Effect on hit rate is relatively small

Replacement Policy

- For direct-mapped cache - easy since only one block is replaced
- For fully-associative and set-associative cache - two strategies:
  - Random
  - Least-recently used (LRU)—replace the block that has not been accessed for a long time. (Principle of temporal locality)



Measuring Cache Performance

- CPU time = Execution cycles × clock cycle time = Instruction_Count × CPI × clock cycle
- If cache miss: (Execution cycles + Memory stall cycles) × clock cycle time
- Memory-stall cycles
  = Memory accesses × miss rate × miss penalty
  = # instructions × misses/instruction × miss penalty

Example

Question: Cache miss penalty = 50 cycles and all instructions take 2.0 cycles without memory stalls. Assume cache miss rate of 2% and 1.33 (why?) memory references per instruction. What is the impact of cache?

Answer: CPU time= IC × (CPI + Memory stall cycles /instruction) × cycle time τ

Performance including cache misses is

CPU time = IC × (2.0 + (1.33 × .02 × 50)) × cycle time
= IC × 3.33 × τ

For a perfect cache that never misses CPU time =IC × 2.0 × τ

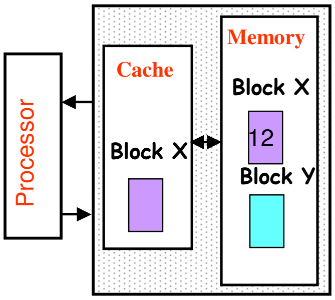
Hence, including the memory hierarchy stretches CPU time by 1.67

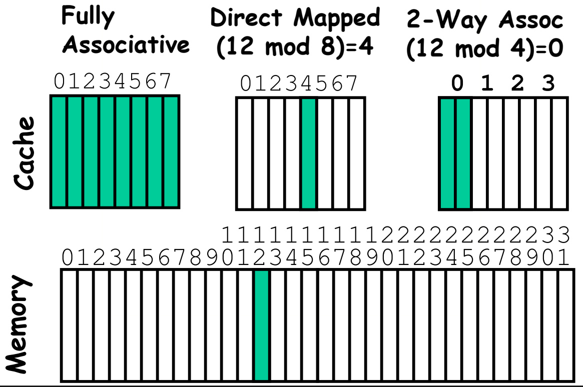
But, without memory hierarchy, the CPI would increase to 2.0 + 50 x 1.33 or 68.5 – a factor of over 30 times longer

Summary: cache organizations

- Direct-mapped: a memory location maps onto exactly one cache entry
- Fully-associative: a memory location can be stored anywhere in cache
- N-way set-associative: each memory location can go into one of n sets

Block #12 placed in a cache
with 8 block frames:


Processor
Cache
Block X
Memory
Block X
12
Block Y


Fully Associative
Direct Mapped (12 mod 8)=4
2-Way Assoc (12 mod 4)=0
01234567
01234567
0 1 2 3
Cache
1111111111222222222233
01234567890123456789012345678901
Memory