Cache Memory

Computer Science 104

Administrivia

- Midterm II next Monday
- Homework #4
- Project Due Date: Wed April 22 in class

Reading

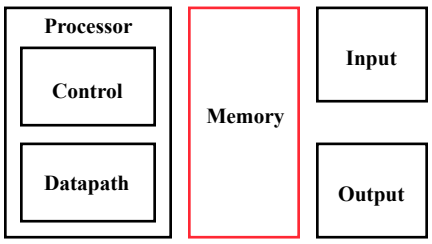
- Chapter 5

Outline of Today's Lecture

- Review
- The Memory Hierarchy
- Direct-mapped Cache
- Two-Way Set Associative Cache
- Fully Associative cache
- Replacement Policies
- Write Strategies

The Big Picture: Where are We Now?

- The Five Classic Components of a Computer



- Today's Topic: Cache Memory

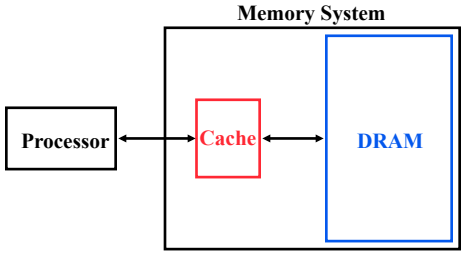
Issues for Memory Systems

- Capacity/Size
- Cost
  - What technology is cheap?
- Performance
  - What technology is fast?
- Ease of Use
  - How much do programmers have to worry about it?

Cache

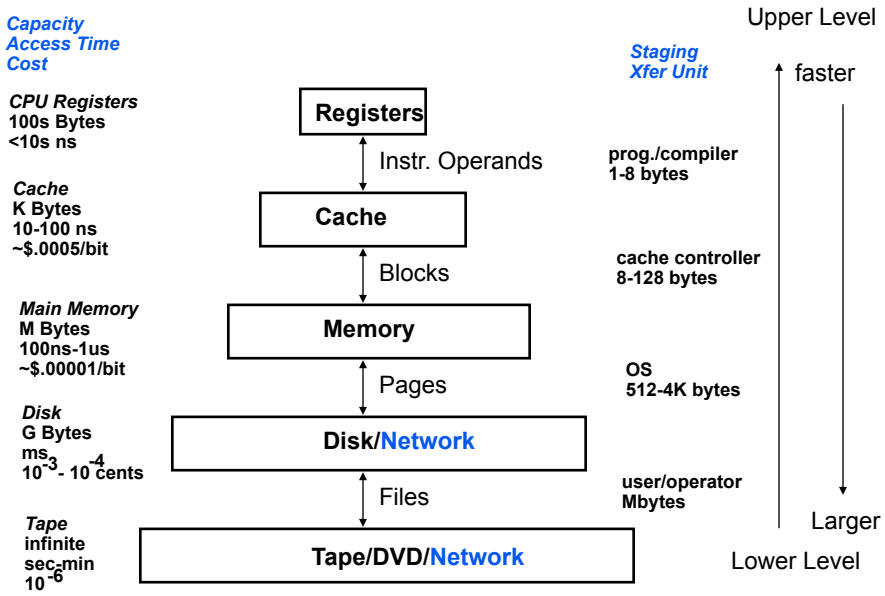
- What is a cache?
- What is the motivation for a cache?
- Why do caches work?
- How do caches work?

The Motivation for Caches



- Motivation:
  - Large memories (DRAM) are slow
  - Small memories (SRAM) are fast
- Make the *average access time* small by:
  - Servicing most accesses from a small, fast memory.
- Reduce the *bandwidth* required of the large memory

Levels of the Memory Hierarchy

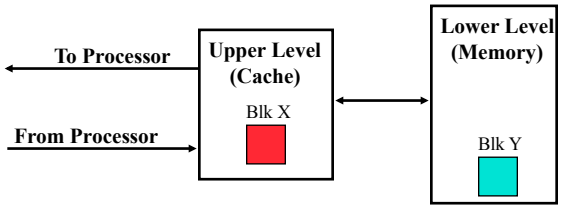


The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
  - Example: 90% of time in 10% of the code
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.

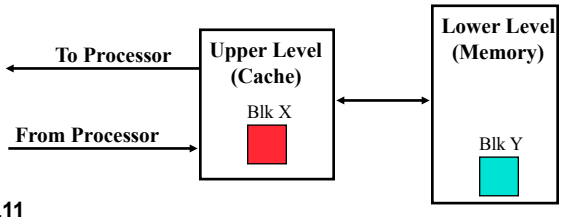
Memory Hierarchy: Principles of Operation

- At any given time, data is copied between only 2 adjacent levels:
  - Upper Level (Cache) : the one closer to the processor
    - Smaller, faster, and uses more expensive technology
  - Lower Level (Memory): the one further away from the processor
    - Bigger, slower, and uses less expensive technology
- Block:
  - The minimum unit of information that can either be present or not present in the two level hierarchy



Memory Hierarchy: Terminology

- Hit: data appears in some block in the upper level (example: Block X)
  - Hit Rate: the fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieved from a lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty = Time to replace a block in the upper level + Time to deliver the block the processor
- Hit Time << Miss Penalty



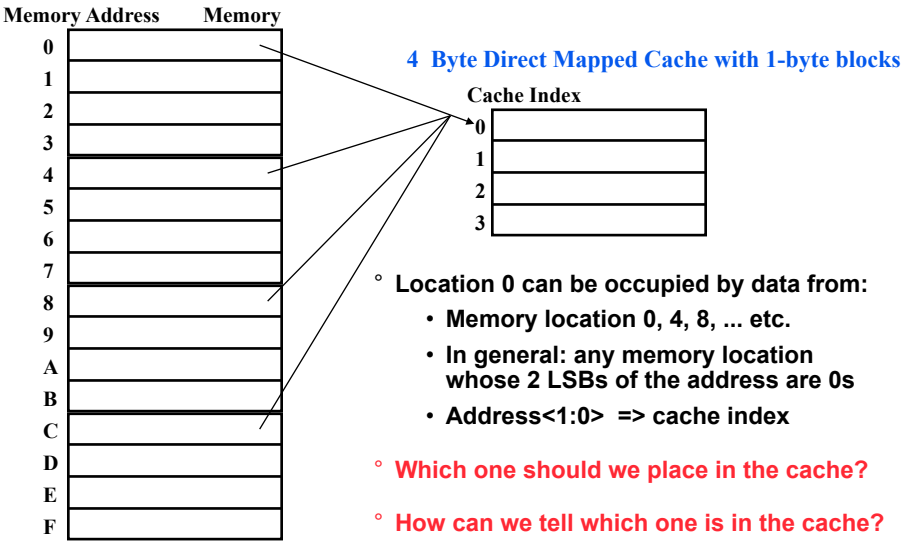
Direct Mapped Cache

- Direct Mapped cache: array of fixed size frames.
- Each frame holds consecutive bytes of main memory data (block).
- The Tag Array holds the Block Memory Address.
- A valid bit associated with each cache block tells if the data is valid.

- Cache Index: The location of a block (and it's tag) in the cache.
- Block Offset: The byte location in the cache block.

Cache-Index = (<Address> Mod (Cache_Size))/ Block_Size
Block-Offset = <Address> Mod (Block_Size)
Tag = <Address> / (Cache_Size)

The Simplest Cache: Direct Mapped Cache



° Location 0 can be occupied by data from:
- Memory location 0, 4, 8, ... etc.
- In general: any memory location whose 2 LSBs of the address are 0s
- Address<1:0> => cache index

° Which one should we place in the cache?

° How can we tell which one is in the cache?

Direct Mapped Cache (Cont.)

For a Cache of 2^M bytes with block size of 2^L bytes

- There are 2^{M-L} cache blocks,
- Lowest L bits of the address are Block-Offset bits
- Next (M - L) bits are the Cache-Index.
- The last (32 - M) bits are the Tag bits.

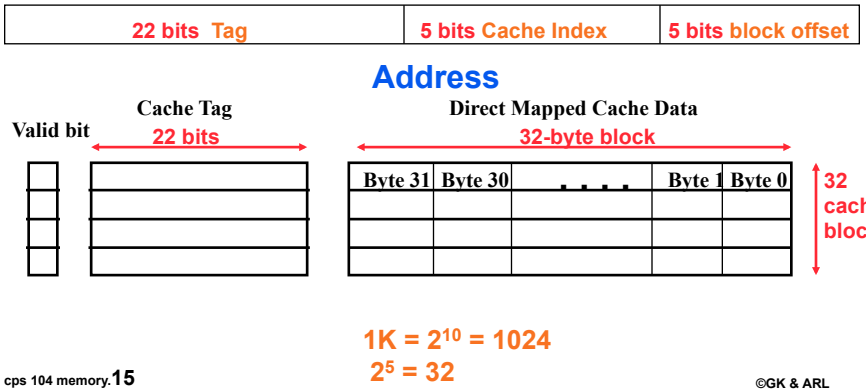
Data Address

Example: 1-KB Cache with 32B blocks:

Cache Index = (<Address> Mod (1024))/ 32

Block-Offset = <Address> Mod (32)

Tag = <Address> / (1024)



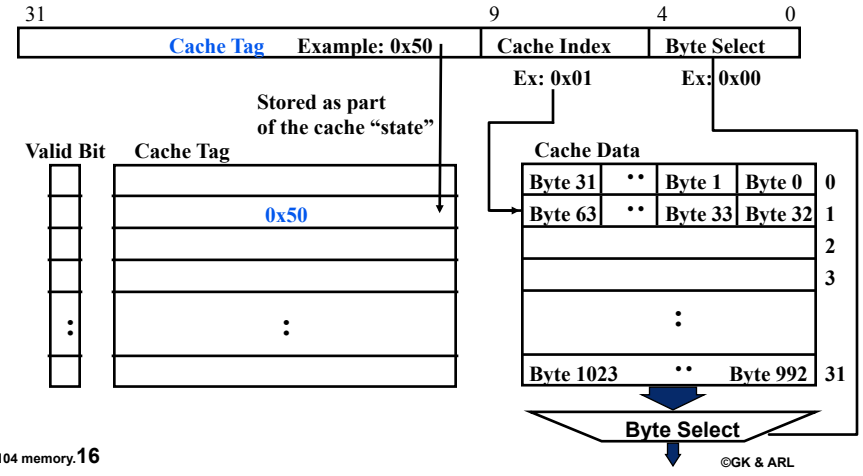
$1K = 2^{10} = 1024$

$2^5 = 32$

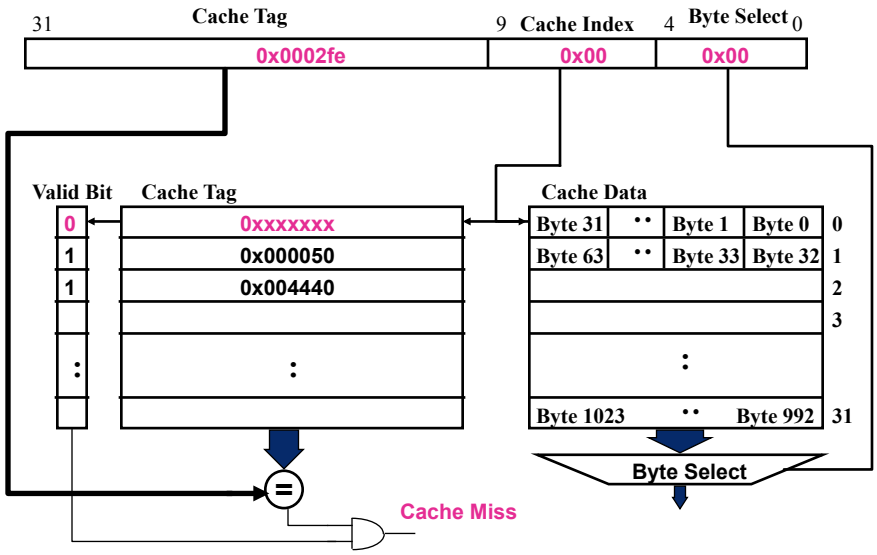
Example: 1KB Direct Mapped Cache with 32B Blocks

° For a 1024 (2^{10}) byte cache with 32-byte blocks:

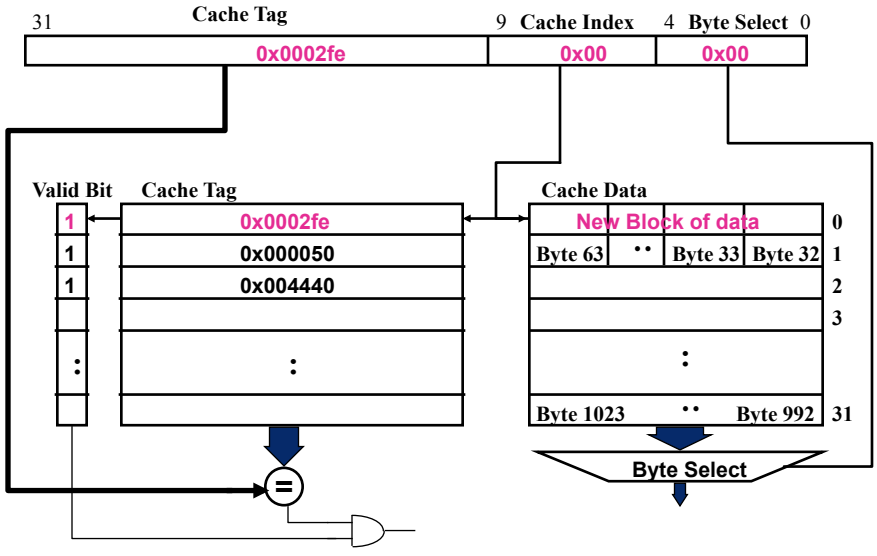
- The uppermost 22 = (32 - 10) address bits are the Cache Tag
- The lowest 5 address bits are the Byte Select (Block Size = 2^5)
- The next 5 address bits (bit5 - bit9) are the Cache Index



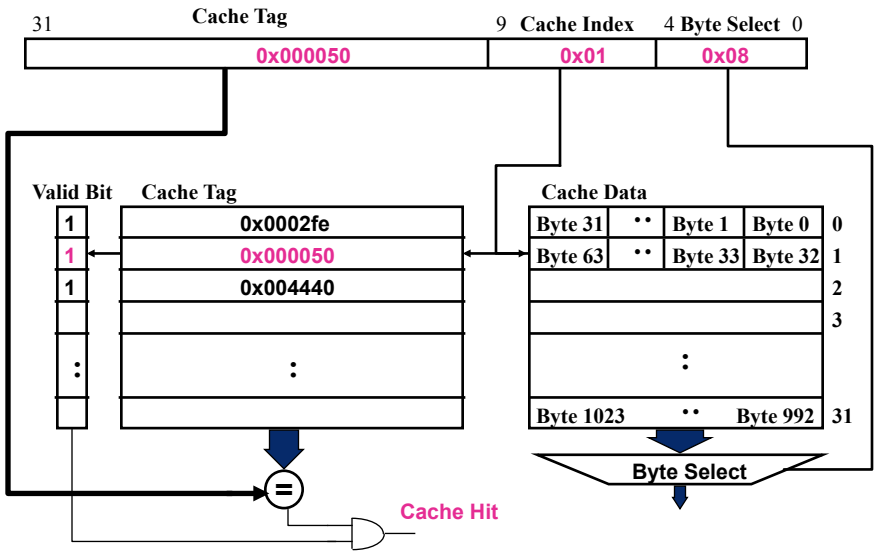
Example: 1K Direct Mapped Cache



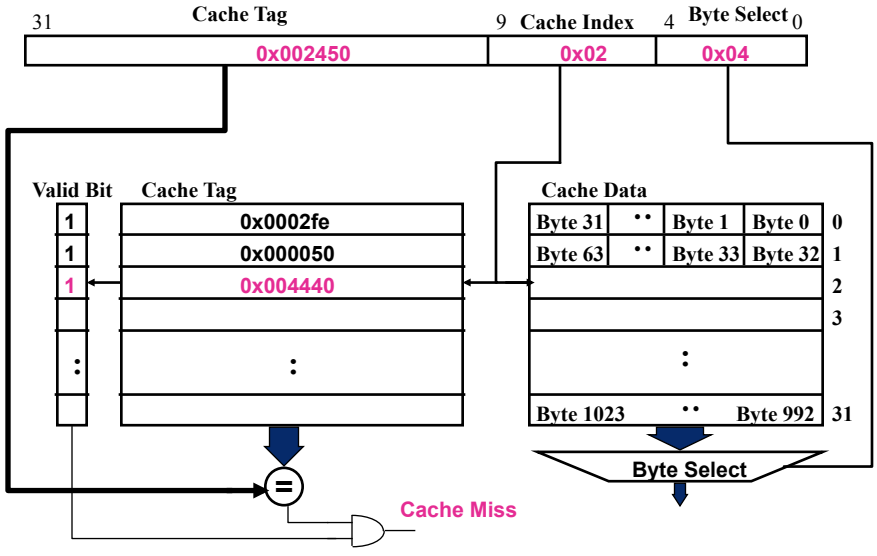
Example: 1K Direct Mapped Cache



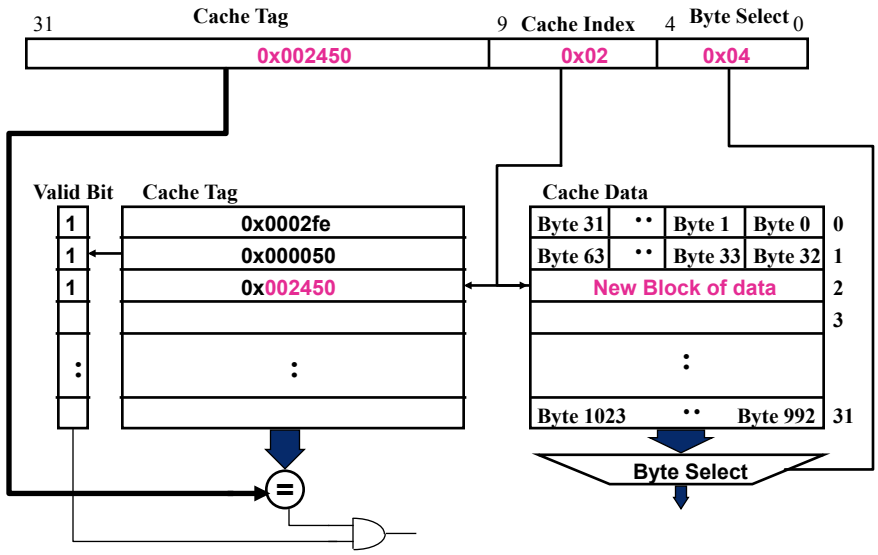
Example: 1K Direct Mapped Cache



Example: 1K Direct Mapped Cache

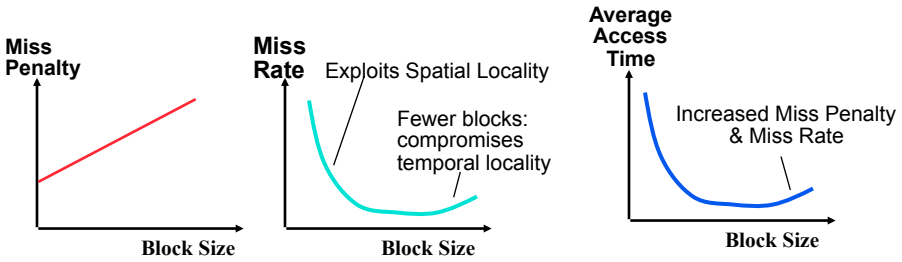


Example: 1K Direct Mapped Cache



Block Size Tradeoff

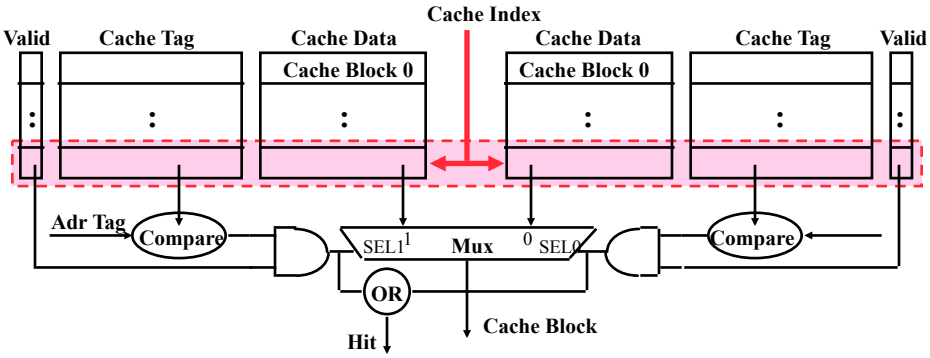
- In general, larger block size take advantage of spatial locality BUT:
  - Larger block size means larger miss penalty:
    - Takes longer time to fill up the block
  - If block size is too big relative to cache size, miss rate will go up
    - Too few cache blocks
- In general, Average Access Time:
  - Hit Time x (1 - Miss Rate) + Miss Penalty x Miss Rate



A N-way Set Associative Cache

° N-way set associative: N entries for each Cache Index
  • N direct mapped caches operating in parallel

° Example: Two-way set associative cache
  • Cache Index selects a "set" from the cache
  • The two tags in the set are compared in parallel
  • Data is selected based on the tag result



Advantages of Set associative cache

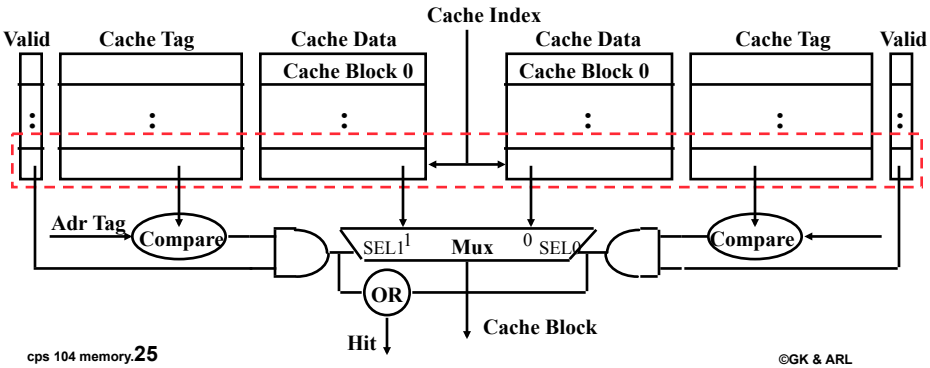
° Higher Hit rate for the same cache size.

° Fewer Conflict Misses.

° Can can have a larger cache but keep the index smaller

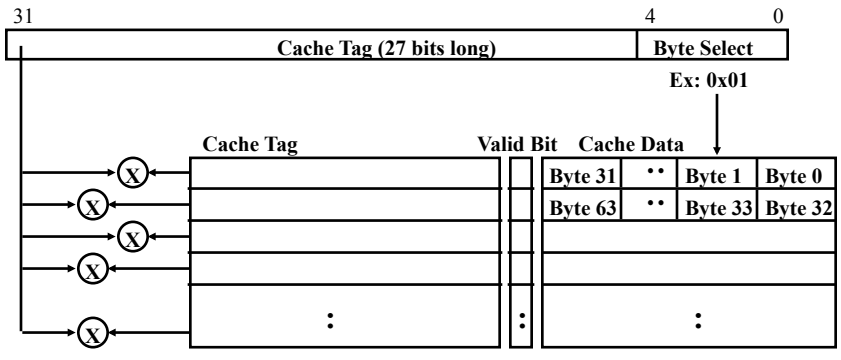
Disadvantage of Set Associative Cache

- N-way Set Associative Cache versus Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss decision and set selection
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.



And yet Another Extreme Example: Fully Associative cache

- Fully Associative Cache -- push the set associative idea to its limit!
  - Forget about the Cache Index
  - Compare the Cache Tags of all cache entries in parallel
  - Example: Block Size = 32B blocks, we need N 27-bit comparators
- By definition: Conflict Miss = 0 for a fully associative cache



Sources of Cache Misses

- **Compulsory** (cold start or process migration, first reference): first access to a block
  - "Cold" fact of life: not a whole lot you can do about it
- **Conflict** (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- **Capacity**:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size
- **Invalidation**: other process (e.g., I/O) updates memory

Sources of Cache Misses

	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size	Big	Medium	Small
Compulsory Miss	Same	Same	Same
Conflict Miss	High	Medium	Zero
Capacity Miss	Low(er)	Medium	High
Invalidation Miss	Same	Same	Same

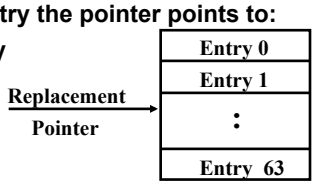
Note:
If you are going to run "billions" of instruction, Compulsory Misses are insignificant.

The Need to Make a Decision!

- Direct Mapped Cache:
  - Each memory location can only map to 1 cache location
  - No need to make any decision
    - Current item replaces the previous item in that cache location
- N-way Set Associative Cache:
  - For each memory location have a choice of N cache locations
- Fully Associative Cache:
  - Each memory location can be placed in ANY cache location
- Cache miss in a N-way Set Associative or Fully Associative Cache:
  - Bring in new block from memory
  - Throw out a cache block to make room for the new block
  - We need to make a decision on which block to throw out!

Cache Block Replacement Policy

- Random Replacement:
  - Hardware randomly selects a cache item and throw it out
- Least Recently Used:
  - Hardware keeps track of the access history
  - Replace the entry that has not been used for the longest time.
  - For two way set associative cache one needs one bit for LRU replacement.
- Example of a Simple "Pseudo" Least Recently Used Implementation (Not Most Recently Used):
  - Assume 64 Fully Associative Entries
  - Hardware replacement pointer points to one cache entry
  - Whenever an access is made to the entry the pointer points to:
    - Move the pointer to the next entry
  - Otherwise: do not move the pointer



Cache Write Policy: Write Through versus Write Back

- Cache read is much easier to handle than cache write:
  - Instruction cache is much easier to design than data cache
- Cache write:
  - How do we keep data in the cache and memory consistent?
- Two options (decision time again :-)
  - Write Back: write to cache only. Write the cache block to memory when that cache block is being replaced on a cache miss.
    - Need a "dirty bit" for each cache block
    - Greatly reduce the memory bandwidth requirement
    - Control can be complex
  - Write Through: write to cache and memory at the same time.
    - What!!! How can this be? Isn't memory too slow for this?

Four Questions for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level? *(Block placement)*
- Q2: How is a block found if it is in the upper level? *(Block identification)*
- Q3: Which block should be replaced on a miss? *(Block replacement)*
- Q4: What happens on a write? *(Write strategy)*

Summary

- Caches provide cost effective memory system
- Work by exploiting locality (temporal & spatial)
- Associativity, Blocksize, Capacity (ABCs of caches)
- Know how a cache works
  - Break address into tag,index, block offset
- Know how to draw a block diagram of a cache

Next Time

- Cache Performance and Programming