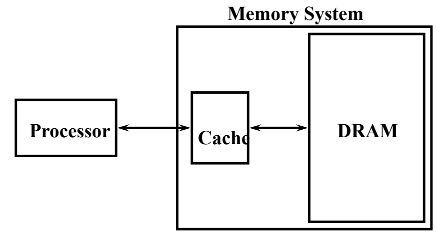
EEL-4713
Computer Architecture
Memory hierarchies

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.

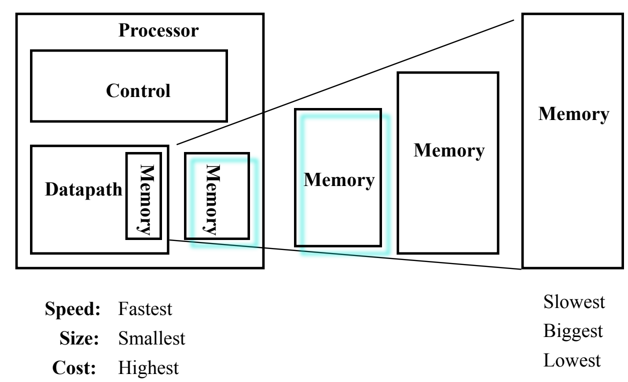
° Reduce the *bandwidth* required of the large memory

Outline of Today's Lecture

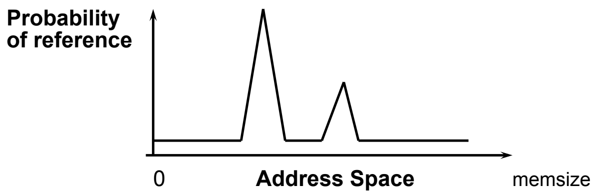
° Introduction to Memory Hierarchies & caches

° Cache write and replacement policies

*An Expanded View of the Memory System

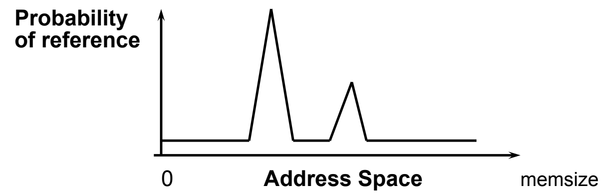


The Principle of Locality



What are the principles of Locality?

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

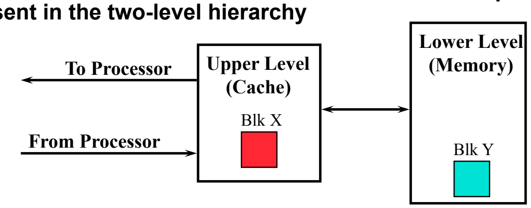
° Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.

Memory Hierarchy: Principles of Operation

° 2-level hierarchy example

° 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

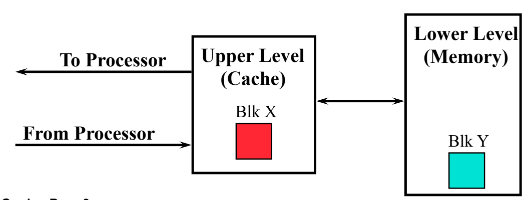


Memory Hierarchy: Terminology

° Hit: data appears in some block in the upper level (example: Block X)
  - Hit Rate: the fraction of memory accesses 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 block in the 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



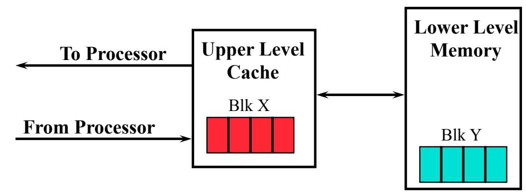
Basic Terminology: Typical Values

	Typical Values
Block (line) size	4 - 128 bytes
Hit time	1 - 4 cycles
Miss penalty	10 - 100 cycles (and increasing)
Miss rate	1% - 20%
Cache Size	64 KB - 8 MB

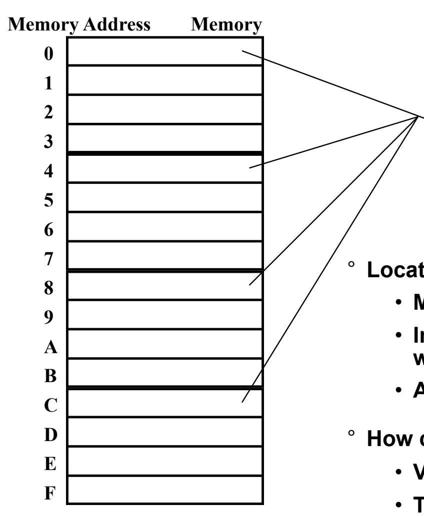
How Do Caches Work?

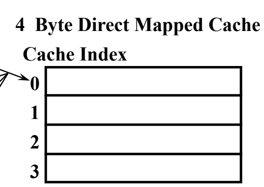
° Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - Keep more recently accessed data items closer to the processor

° Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.
  - Move blocks consisting of contiguous words to the cache



The Simplest Cache: Direct-Mapped Cache



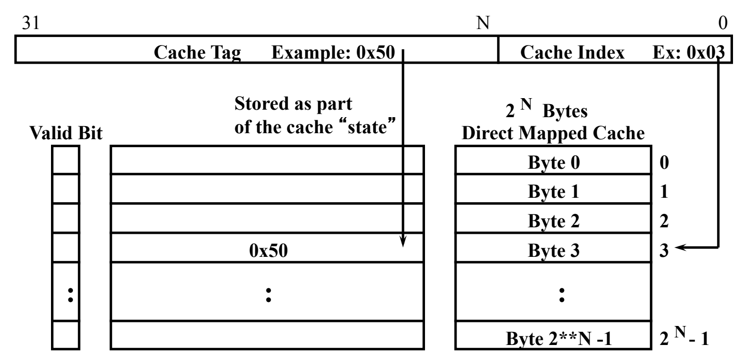


° Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ... etc.
  - In general: any memory location whose 2 LSBs of the address are 0s
  - Address<1:0> => cache index

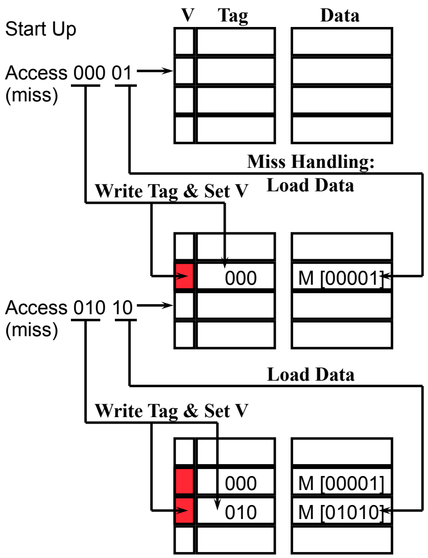
° How can we tell which block is in the cache?
  - Valid bit
  - Tag

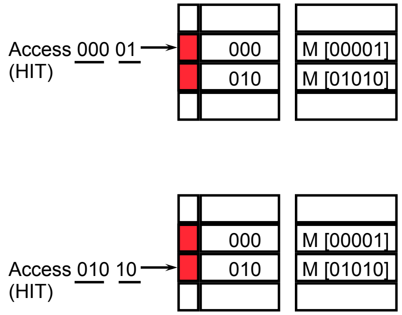
Cache Tag and Cache Index

° Assume a 32-bit memory (byte) address:
  - A 2^N bytes direct mapped cache:
    - Cache Index: The lower N bits of the memory address
    - Cache Tag: The upper (32 - N) bits of the memory address



Cache Access Example

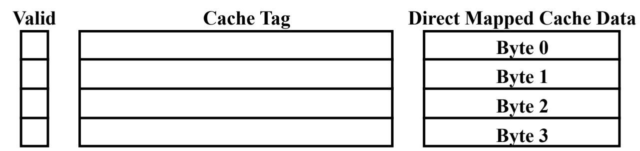




- Caches begin empty:
  - A lot of misses at start up:
    Compulsory Misses
    - (Cold start misses)

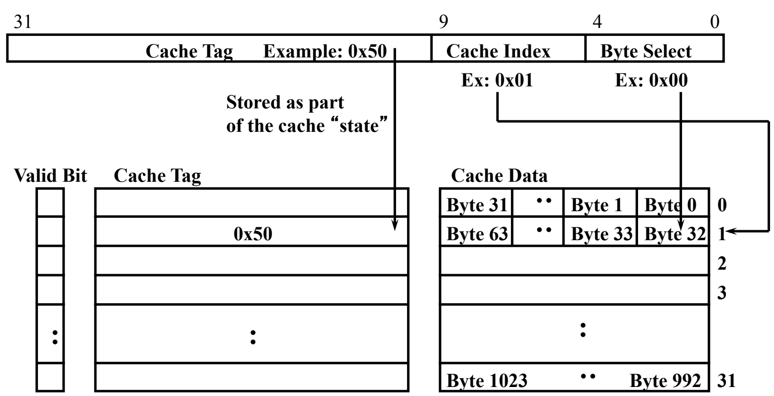
Definition of a Cache Block

- Cache Block (cache "line"): cache data that has its own cache tag
- Our previous "extreme" example:
  - 4-byte Direct Mapped cache: Block Size = 1 Byte
  - Takes advantage of Temporal Locality: If a byte is referenced, it will tend to be referenced soon.
  - Did not take advantage of Spatial Locality: If a byte is referenced, its adjacent bytes will be referenced soon.
- In order to take advantage of Spatial Locality: increase the block size



Example: 1 KB Direct Mapped Cache with 32 B Blocks

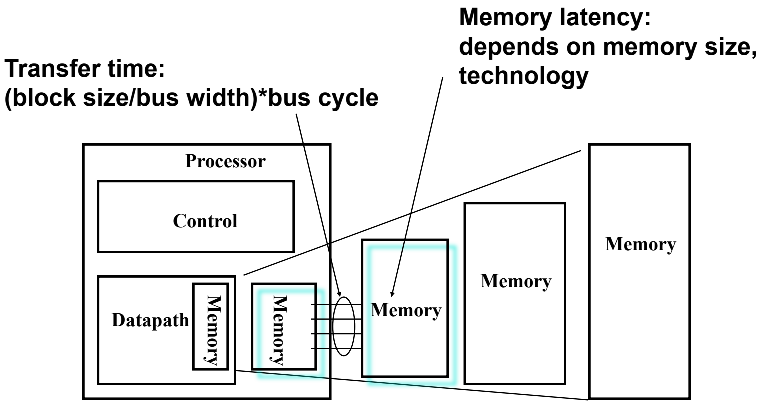
- For a 2 ** N byte cache:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = 2 ** M)



Block size tradeoffs

- Larger block helps with spatial locality
- However, transferring a large block from memory to cache takes longer than transferring a small block
  - "miss penalty"
- Important to understand implication of block sizes in the *average time to service a memory request*
- Average memory access time (AMAT)
  - Hit_time * (1 – miss_rate) + miss_rate*miss_penalty
- Cache design goals:
  - Short hit time
  - Small miss rate
  - Short miss penalty

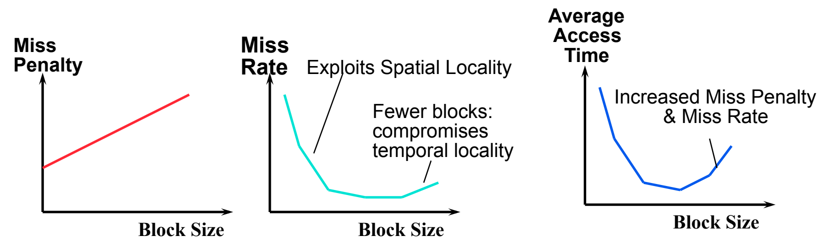
Components of miss penalty



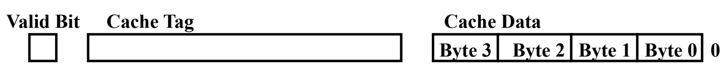
Penalty ≈ latency + transfer time

*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
- Average Access Time:
  - = Hit Time x (1 - Miss Rate) + Miss Penalty x Miss Rate



Another Extreme Example



- Cache Size = 4 bytes Block Size = 4 bytes
  - Only ONE entry in the cache
- If an item is accessed, likely that it will be accessed again soon
  - But it is unlikely that it will be accessed again immediately!!!
  - The next access will likely to be a miss again
    - Continually loading data into the cache but discard (force out) them before they are used again
- Conflict Misses are misses caused by:
  - Different memory locations mapped to the same cache index
    - Solution 1: make the cache size bigger
    - Solution 2: Multiple entries for the same Cache Index

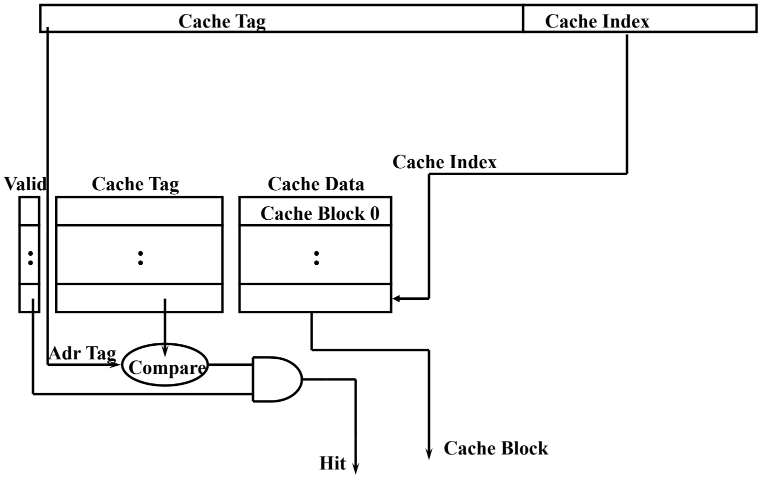
Outline

- Set-associative caches
- Simulation experiments
- Replacement and write policies

*Our original direct-mapped cache

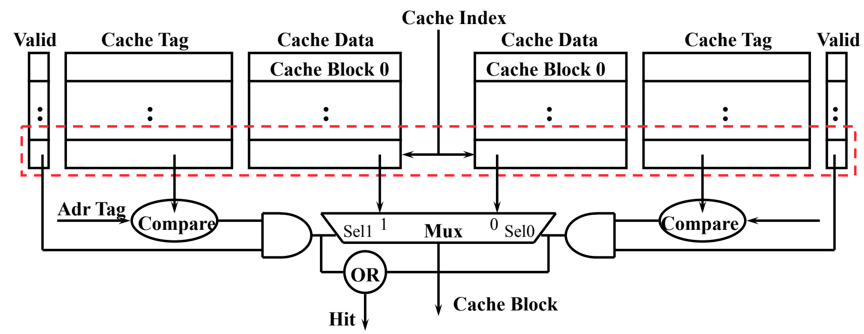
- Implementation diagram

Address of a lw/sw



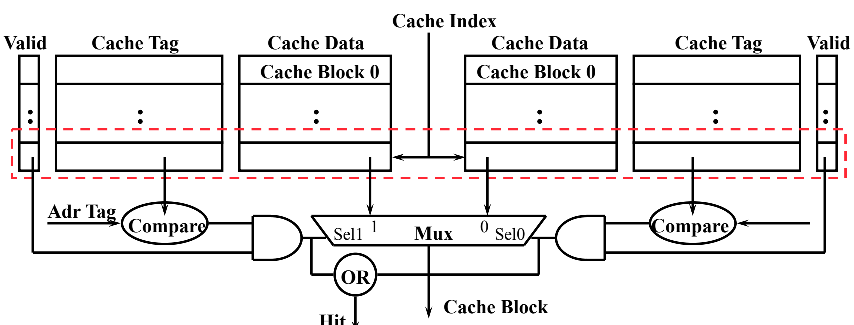
A Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operate 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



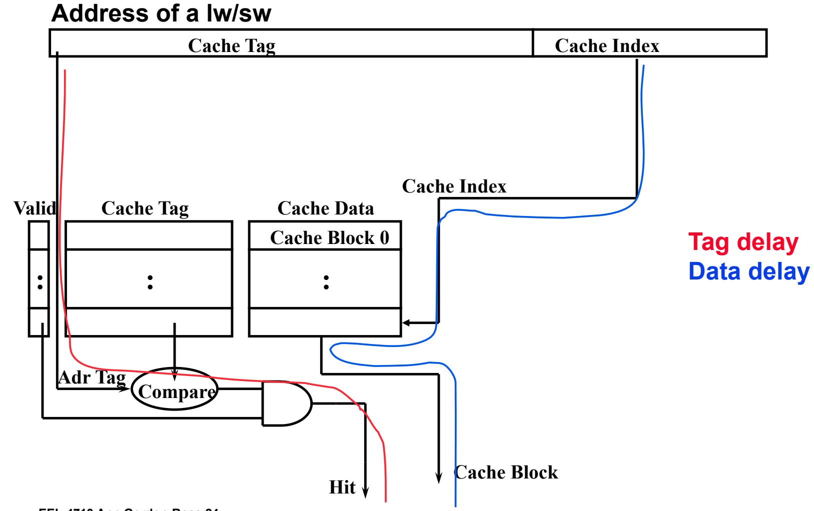
Disadvantage of Set Associative Cache

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

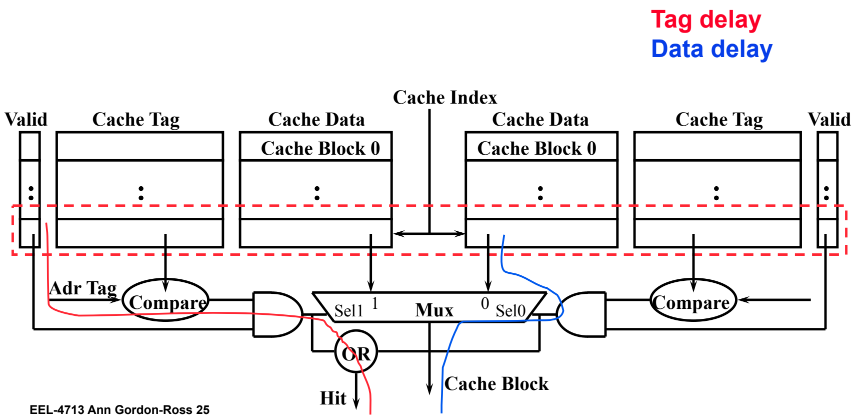


Critical path: direct-mapped cache

Address of a lw/sw



Critical path: Set Associative Cache



Examples

- Cacti simulation
  - Same size, block, feature size
  - Cycle times: Direct-mapped versus 2-way
    - DM: 0.654811 ns
    - 2-way: 0.725646 ns
- Simplescalar simulation
  - Hit rates: direct mapped versus 2-way
- Impact on Average Memory Access Time?

Simplescalar simulation

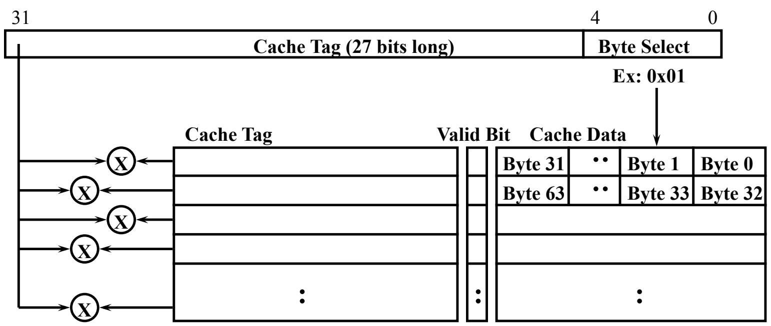
- Direct-mapped
- il1.accesses 46358 # total number of accesses
- il1.hits 42336 # total number of hits
- dl1.accesses 13278 # total number of accesses
- dl1.hits 12620 # total number of hits
- 2-way set associative
- il1.accesses 46358 # total number of accesses
- il1.hits 44368 # total number of hits
- dl1.accesses 13278 # total number of accesses
- dl1.hits 12870 # total number of hits

A Summary on Sources of Cache Misses

- Compulsory (cold start, 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, other CPU) updates memory

And yet Another Extreme Example: Fully Associative

- 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 = 2 B blocks, we need N 27-bit comparators
- By definition: Conflict Miss = 0 for a fully associative cache

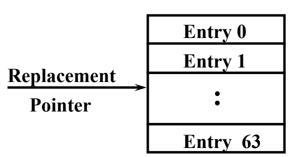


Making room for a new block

- Direct Mapped Cache:
  - Each memory location can only mapped to 1 cache location
  - No need to make any decision
    - Current item replaced the previous item in that cache location
- N-way Set Associative Cache:
  - 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
  - Need to make a decision on which block to throw out!

Cache Block Replacement Policy

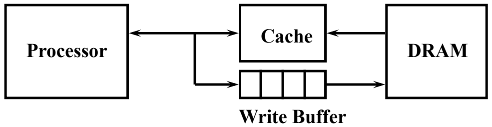
- Random Replacement:
  - Hardware randomly selects a cache item and throws it out
- Least Recently Used:
  - Hardware keeps track of the access history
  - Replace the entry that has not been used for the longest time
- Example of a Simple "Pseudo" Least Recently Used Implementation:
  - Assume 64 Fully 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

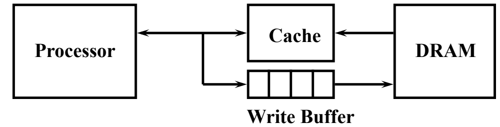
- Cache read is much easier to handle than cache write:
  - Instruction cache is much easier to design than data cache
- Cache write:
  - How do we keep data in the cache and memory consistent?
- Two options (decision time again :-)
  - Write Back: write to cache only. Write the cache block to memory 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.
    - Isn't memory too slow for this?

Write Buffer for Write Through

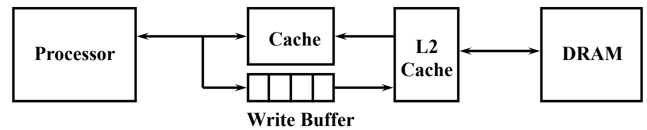


- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- However if:
  - Store frequency > 1 / DRAM write cycle
  - Write buffer saturation

Write Buffer Saturation



- Store frequency > 1 / DRAM write cycle
  - If this condition exist for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
    - Store buffer will overflow no matter how big you make it
- Solution for write buffer saturation:
  - Use a write back cache
  - Install a second level (L2) cache:

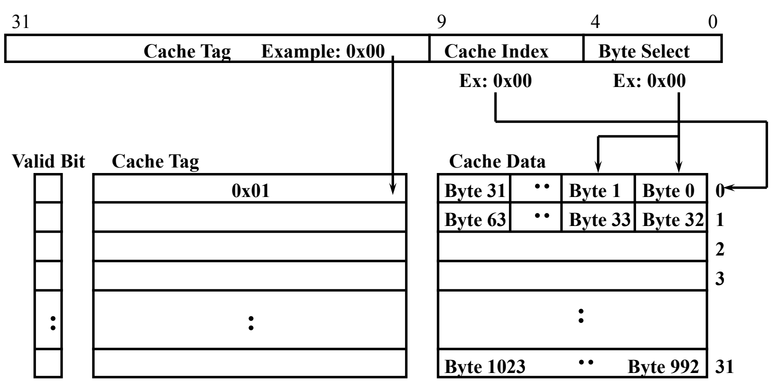


Write Allocate versus Not Allocate

- Assume: a 16-bit write to memory location 0x0 causes a miss
  - Do we read in the rest of the block (Bytes 2, 3, ... 31)?

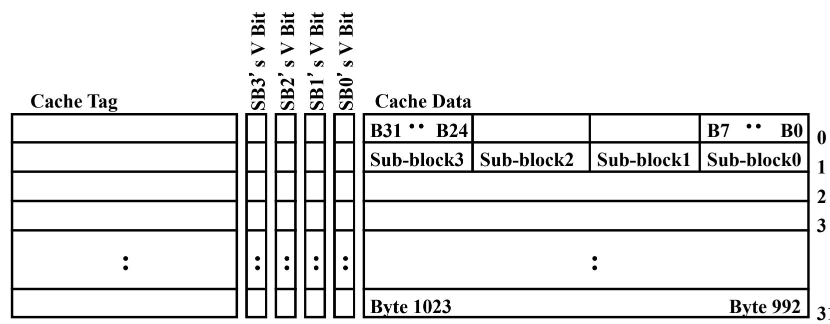
    Yes: Write Allocate (usually associated w/ write-back)

    No: Write Not Allocate (write-through)



What is a Sub-block?

- Sub-block:
  - A unit within a block that has its own valid bit
  - Example: 1 KB Direct Mapped Cache, 32-B Block, 8-B Sub-block
    - Each cache entry will have: 32/8 = 4 valid bits
- Write miss: only the bytes in that sub-block are brought in.



“Unified” versus “Split” caches

° Unified: both instructions and data co-reside in the same cache

° Split: different instruction (I) and data (D) caches

° Typically, today’s on-chip caches closest to processor (L1) are split
- I-cache typically has better locality and can be made smaller to be faster (remember instruction fetch is in the critical path!)
- Separate caches avoid data blocks replacing instruction blocks

° L2+ caches typically unified
- Much larger; chance of replacement small
- Not in the critical path
- A unified design is simpler to implement

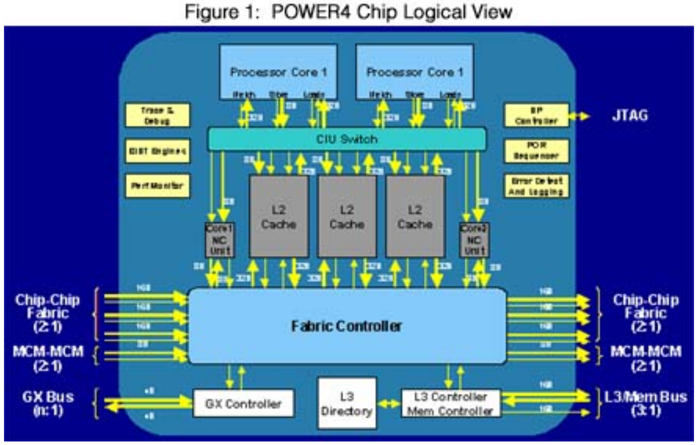
Multi-level caches

° Close to processor:
- Level-1 cache
- Goal: maximize hit rate while keeping cycle time as close as possible to datapath
- Instruction fetch in a cycle; data access in a cycle

° Far from (or outside) processor
- Level-2 (-3) caches
- Goal is to maximize hit rate while keeping cost (I.e. area) within a target design goal

Example: IBM Power4

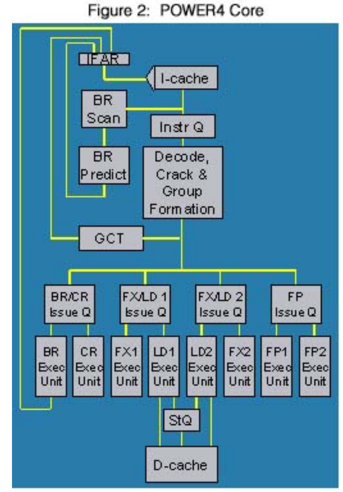
Figure 1: POWER4 Chip Logical View



Source: http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4.html

Example (cont)

Figure 2: POWER4 Core



Source: http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/power4.html

Power4 cache hierarchy

- Level 1, on-chip
- Power-4 has two processors on a chip
  - Each processor has its own L1 cache
  - Each cache is split
- L1 Instruction Cache
  - Direct mapped, 128-byte block managed as 4 32-byte sub-blocks
  - 128 KB (64 KB per processor)
- L1 Data Cache
  - 2-way, 128-byte block
  - 64 KB (32 KB per processor)
- One 32-byte read or write per cycle

Power4 cache hierarchy

- Level 2, on-chip
- Still SRAM
- Shared across two processors
- L2 configuration
  - unified
  - 8-way, 128-byte block
  - 1.5 MB

Power4 cache hierarchy

- Level 3, off-chip
  - Directory tags for multiprocessing on-chip
- Embedded DRAM
- L3 configuration
  - 8-way, 512-byte block managed as 4 128-byte sub-blocks
  - 32 MB
  - shared

Power5 cache hierarchy

- Improvements in size, associativity to reduce capacity/conflict misses
- L1 I-cache: 2-way set associative
- L1 D-cache: 4-way set associative
- L2 unified cache: 1.92MB, 10-way associative, 13-cycle latency
- L3 unified cache: 36MB, 12-way associative, 87-cycle latency
- Memory: 220-cycle latency
- http://www.realworldtech.com/page.cfm?ArticleID=RWT100404214638&p=3

Summary:

- The Principle of Locality:
  - Programs access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses:
  - Compulsory Misses: cold start misses.
  - Conflict Misses: increase cache size and/or associativity.
  - Capacity Misses: increase cache size
- Write Policy:
  - Write Through: need a write buffer
  - Write Back: control can be complex