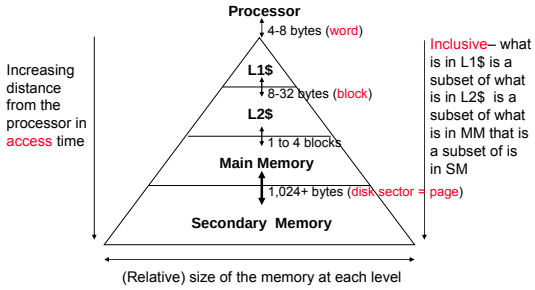
EEC 170
Computer Architecture
Fall 2005

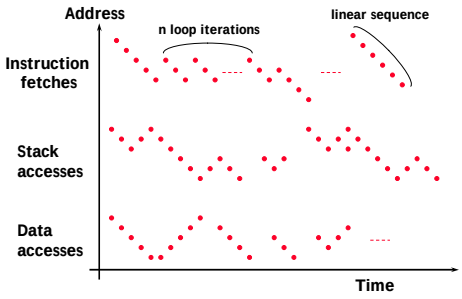
Cache Introduction Review

Review: The Memory Hierarchy

- Take advantage of the principle of locality to present the user with as much memory as is available in the cheapest technology at the speed offered by the fastest technology

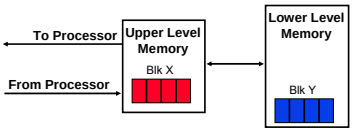


Typical Memory Reference Patterns



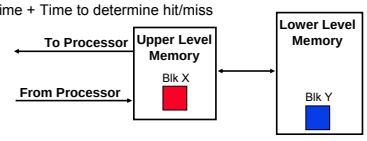
The Memory Hierarchy: Why Does it Work?

- Temporal Locality (Locality in Time):
  - ⇒ Keep most recently accessed data items closer to the processor
- Spatial Locality (Locality in Space):
  - ⇒ Move blocks consisting of contiguous words to the upper levels



The Memory Hierarchy: Terminology

- Hit: data appears in some block in the upper level (Blk 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 retrieve from a block in the lower level (Blk 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

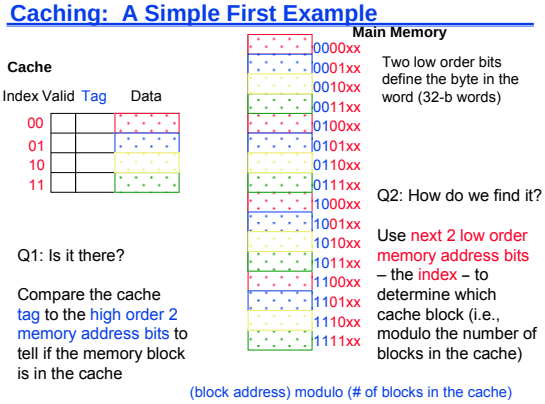
How is the Hierarchy Managed?

- registers ↔ memory
  - by compiler (programmer?)
- cache ↔ main memory
  - by the cache controller hardware
- main memory ↔ disks
  - by the operating system (virtual memory)
  - by the programmer (files)
  - virtual to physical address mapping assisted by the hardware (TLB)

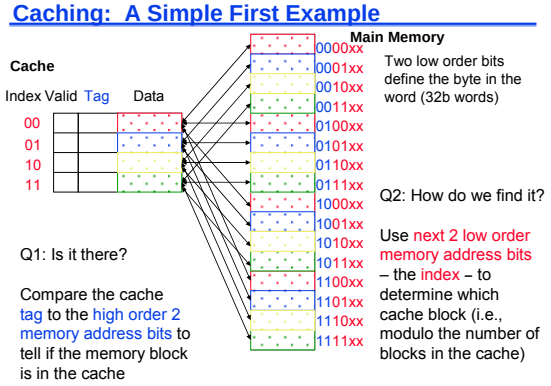
Cache

- Two questions to answer (in hardware):
  - Q1: How do we know if a data item is in the cache?
  - Q2: If it is, how do we find it?
- Direct mapped
  - For each item of data at the lower level, there is exactly one location in the cache where it might be - so lots of items at the lower level share locations in the upper level
  - Address mapping:
    (block address) modulo (# of blocks in the cache)
  - First consider when block is one word

Caching: A Simple First Example



Caching: A Simple First Example

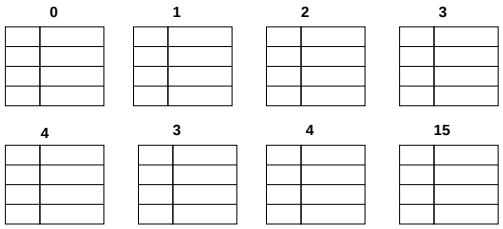


Direct Mapped Cache

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 14

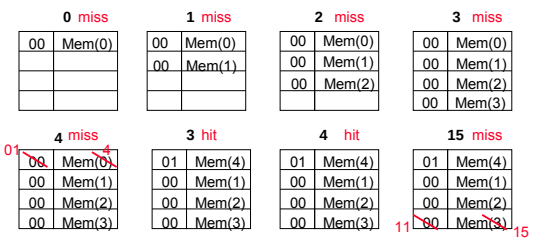


Direct Mapped Cache

- Consider the main memory word reference string

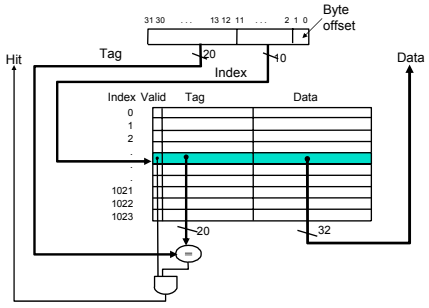
Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15



MIPS Direct Mapped Cache Example

- One word/block, cache size = 1K words

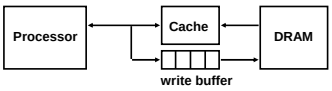


What kind of locality are we taking advantage of?

Handling Cache Hits

- Read hits (I$ and D$)
  - this is what we want!
- Write hits (D$ only)
  - allow cache and memory to be inconsistent
    - write the data only into the cache block (write-back the cache contents to the next level in the memory hierarchy when that cache block is "evicted")
    - need a dirty bit for each data cache block to tell if it needs to be written back to memory when it is evicted
  - require the cache and memory to be consistent
    - always write the data into both the cache block and the next level in the memory hierarchy (write-through) so don't need a dirty bit
    - writes run at the speed of the next level in the memory hierarchy – so slow! – or use a write buffer, a buffer that holds the data that is waiting to be written, so would only have to stall if the write buffer is full

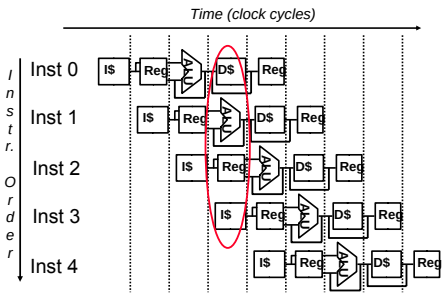
Write Buffer for Write-Through Caching



- Write buffer between the cache and memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: writes contents of the write buffer to memory
- The 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:
  - Store frequency (w.r.t. time) → 1 / DRAM write cycle
  - write buffer saturation
    - One solution is to use a write-back cache; another is to use an L2 cache (next lecture)

Review: Why Pipeline? For Throughput!

- To avoid a structural hazard need two caches on-chip: one for instructions (I$) and one for data (D$)



To keep the pipeline running at its maximum rate both I$ and D$ need to satisfy a request from the datapath every cycle.

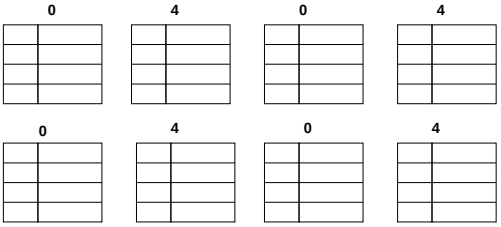
What happens when they can't do that?

Another Reference String Mapping

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 4 0 4 0 4 0 4

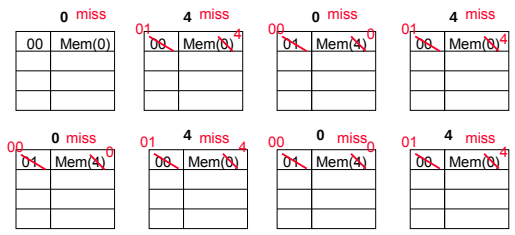


Another Reference String Mapping

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 4 0 4 0 4 0 4



- Ping pong effect due to conflict misses - two memory locations that map into the same cache block

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
  - If you are going to run "billions" of instruction, compulsory misses are insignificant
- 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 accessed by the program
  - Solution: increase cache size

Handling Cache Misses

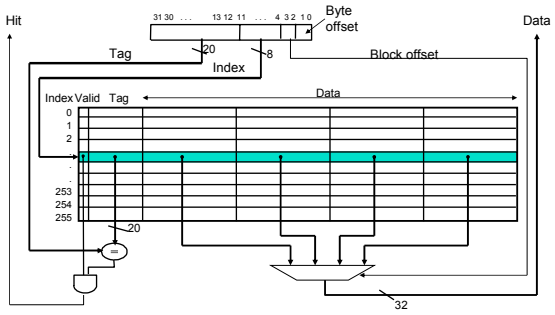
- Read misses (I$ and D$)
  - stall the entire pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache, then restart the pipeline and reissue the read request
- Write misses (D$ only)
  1. stall the pipeline, fetch the block from next level in the memory hierarchy, install it in the cache (which may involve having to evict a dirty block if a write-back cache), then restart the pipeline and reissue the write request

  or (normally used in write-back caches)
  2. Write allocate – just write the word into the cache updating both the tag and data, no need to check for cache hit, no need to stall

  or (normally used in write-through caches with a write buffer)
  3. No-write allocate – skip the cache write and just write the word to the write buffer (and eventually to the next memory level), no need to stall if the write buffer isn't full; must invalidate the cache block since it will become inconsistent (on a hit)

Multiword Block Direct Mapped Cache

- Four words/block, cache size = 1K words



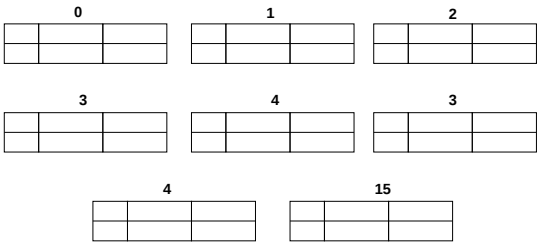
What kind of locality are we taking advantage of?

Taking Advantage of Spatial Locality

- Let cache block hold more than one word

Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15

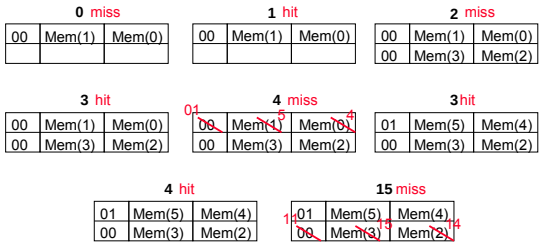


Taking Advantage of Spatial Locality

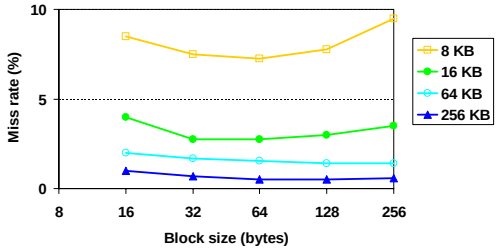
- Let cache block hold more than one word

Start with an empty cache - all blocks initially marked as not valid

0 1 2 3 4 3 4 15



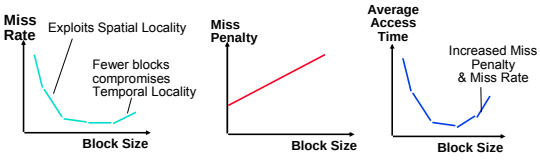
Miss Rate vs Block Size vs Cache Size



- Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)

Block Size Tradeoff

- Larger block sizes take advantage of spatial locality but
  - If the block size is too big relative to the cache size, the miss rate will go up
    - Too few cache blocks
  - Larger block size means larger miss penalty
    - Takes longer time to fill up the block



- In general, Average Memory Access Time
  = Hit Time + Miss Penalty x Miss Rate

Multiword Block Considerations

- Read misses (I$ and D$)
  - Processed the same as for single word blocks – a miss returns the entire block from memory
  - Miss penalty grows as block size grows
    - Latency to first word in block + transfer time for remaining words
    - Early restart – resume execution as soon as the requested word of the block is returned
    - Requested word first – requested word is transferred from the memory to the cache first
  - Nonblocking cache – allows the processor to access the cache while the cache is handling an earlier miss
- Write misses (D$)
  - Can't use write allocate or will end up with a "garbled" block in the cache (e.g., for 4 word blocks, a new tag, one word of data from the new block, and three words of data from the old block), so must fetch the block from memory first and pay the stall time

Cache Summary

- The Principle of Locality:
  - Program likely to 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: sad facts of life. Example: cold start misses
  - Conflict misses: increase cache size and/or associativity Nightmare Scenario: ping pong effect!
  - Capacity misses: increase cache size
- Cache design space
  - total size, block size, associativity (replacement policy)
  - write-hit policy (write-through, write-back)
  - write-miss policy (write allocate, write buffers)