Associative Mapping

- main memory block can load into any line of cache
- memory address is interpreted as tag and word select in block
- tag uniquely identifies block of memory !
- every line's tag is examined for a match
- cache searching gets expensive

Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size – tag size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+ w}/2^{w} = 2^{s}$
- Number of lines in cache = undetermined
- Size of tag = s bits

Fully Associative Mapping Address Structure

Tag 22 bit	Word 2 bit

- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which word is required. There are 4 words in a block. Each word is 8-bits. Block is 32 bit.
- e.g.
  - Address Tag Data Cache line
  - FFFFFC 3FFFFF 24682468 any, e.g. 3FFF

Replacement Algorithms

- *When a block is fetched, which block in the target set should be replaced?*
- Optimal algorithm:
  - replace the block that will not be used for the longest time (must know the future)
- Usage based algorithms:
  - Least recently used (LRU)
    - replace the block that has been referenced least recently
    - hard to implement
- Non-usage based algorithms:
  - First-in First-out (FIFO)
    - treat the set as a circular queue, replace head of queue.
    - easy to implement
  - Random (RAND)
    - replace a random block in the set
    - even easier to implement

Write Policy: Write-Through vs Write-Back

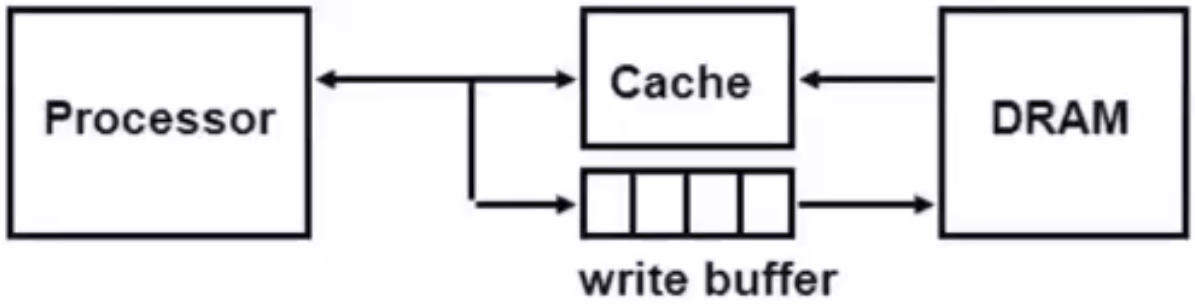
- Write-through: all writes update cache and underlying memory/cache
  - Can always discard cached data - most up-to-date data is in memory
  - Cache control bit: only a valid bit
- Write-back: all writes simply update cache
  - Can't just discard cached data - may have to write it back to memory
  - Cache control bits: both valid and dirty bits
- Other Advantages:
  - Write-through:
    - memory (or other processors) always have latest data
    - Simpler management of cache
  - Write-back:
    - much lower bandwidth, since data often overwritten multiple times
    - Better tolerance to long-latency memory?

Write Policy 2: Write Allocate vs Non-Allocate

- What happens on write-miss:
- Write allocate: allocate new cache line in cache
  - Usually means that you have to do a “read miss” to fill in rest of the cache-line!
  - Alternative: per/word valid bits
  - + Decreases read misses (next read to block will hit)
  - – Requires additional bandwidth
  - Commonly used (especially with write-back caches)
- Write non-allocate (or “write-around”):
  - Simply send write data through to underlying memory/cache - don’t allocate new cache line!
  - – Potentially more read misses
  - + Uses less bandwidth
  - Use with write-through

Write Buffers

- Write buffer: a small buffer
  - Stores put address/value to write buffer, keep going
  - Write buffer writes stores to D$ in the background
  - Loads must search write buffer (in addition to D$)
  - + Eliminates stalls on write misses (mostly) WBB
  - – Creates some problems (later)



Basics about Caches
COMPLETED

Cache Basics

- Capacity <<< memory
- Locality of reference
  - Spatial
  - Temporal
- Block valid?
- Cache Hit / Miss
- Tag-bits, index, byte offset
- Associativity: direct, set, full
- Replacement policies
- Write-back
- Write-through
- write-allocate
- Write-no-allocate

Cache Coherence Problem