4
Cache Organization

18-548/15-548 Memory System Architecture
Philip Koopman
September 2, 1998

Required Reading: Cragon 2.1, 2.1.2, 2.1.3, 2.2-2.2.2
Supplemental Reading: Hennessy & Patterson 5.1, 5.2, pp. 390-393

Assignments

- **By next class read:**
  - Cragon 3.0-3.3.1; 3.4 to top of page 166
    - You are *not* accountable for the details of the examples
- **Supplemental reading:**
  - Hennessy & Patterson: 5.7, 5.8
  - http://www.cne.gmu.edu/modules/vm/submap.html
  - Virtual Memory: Issues of Implementation (*Computer*, June 1998)
- **Homework #2 due next Wednesday**
  - Physical Memory
  - Cache organization & access
- **Lab #1 due Friday 3:00 PM to course secretary**
  - *Short* answers are in general better than long ones (single sentence is fine)
  - BUT, be sure to put something down for each question asked.

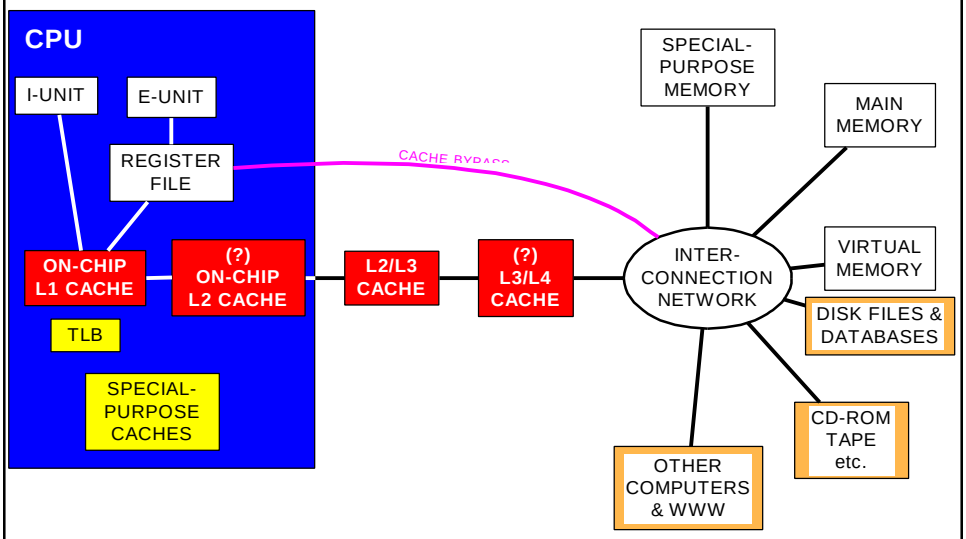
Where Are We Now?

- **Where we've been:**
  - Physical memory hierarchy
    - CPU registers
    - Cache
    - Bus
    - Main Memory
    - Mass storage
- **Where we're going today:**
  - Cache organization -- a key piece of the physical memory hierarchy
    - How it works
    - Performance metrics
- **Where we're going next:**
  - Virtual memory architecture -- address mapping within memory hierarchy

Preview

- **Cache organization**
  - Terminology
  - Data Organization
  - Control status bit organization
- **Conceptual cache operation**
  - Hits, misses, access sequences
- **Quantifying cache performance**
  - 3 "C"s -- different causes of cache misses
  - Effective memory access time

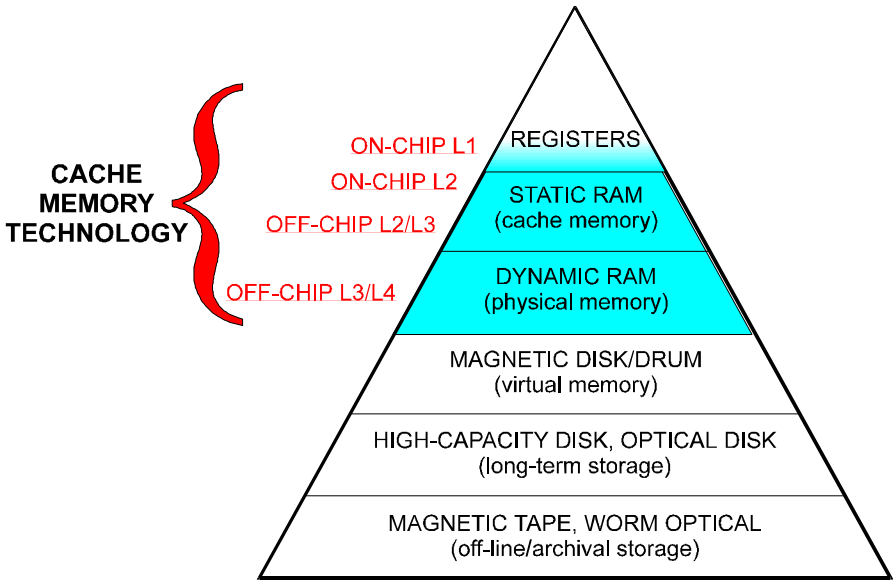
Caches Within the Reference Memory Hierarchy



Cache Memory

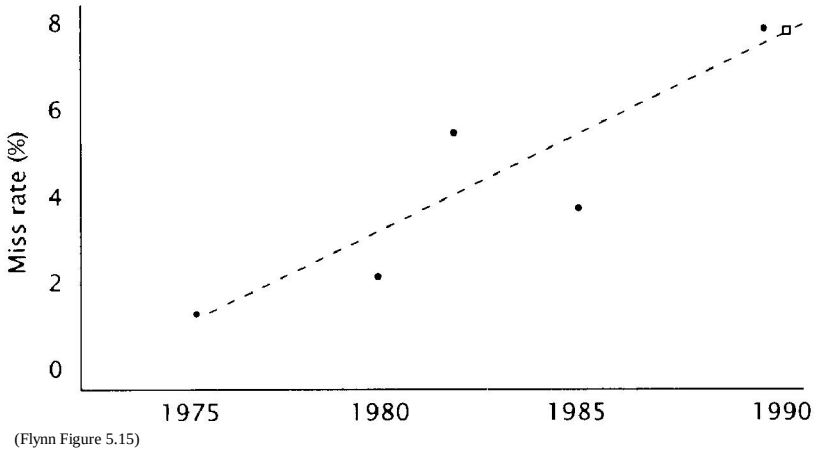
- **A small, high speed memory close to the CPU**
  - Each line of cache memory can be "occupied" or "empty"
  - "Occupied" lines map to a memory location
  - Hardware cache management mechanism (potentially with software hints)
  - Typically used for both programs and data
- **Also, by extension, other specialized hardware or software buffers**
  - TLB for mapping physical to virtual memory addresses
  - Disk sector storage
  - Network machine name to IP address translation
  - Read and write buffers
- **Works because of locality**
  - (Doesn't work when there isn't locality)
- **First commercial use:** _____

Typical Implementation Technologies



Isn't Cache a Solved Problem?

- **8 KB cache effectiveness over time**
  - But for practical purposes total L1 cache is limited to 8-32KB on chip in most designs



(Flynn Figure 5.15)

CACHE ORGANIZATION

Major Cache Design Decisions

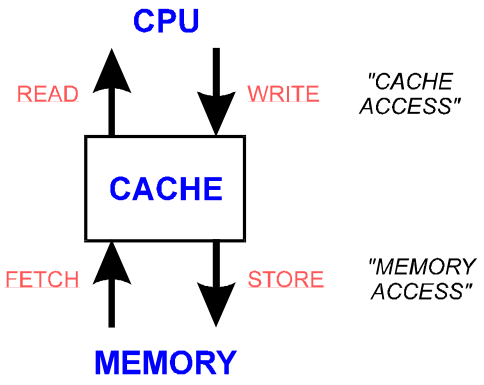
- **Cache Size -- in bytes**
- **Split/Unified -- instructions and data in same cache?**

  Associativity -- how many sectors in each set?
- **Sector/Block size -- how many bytes grouped together as a unit?**
  - Cache "Block" also called Cache "Line"
- **How many levels of cache?**
  - Perhaps L1 cache on-chip; L2 cache on-module; L3 cache on motherboard

  +
- **Management policies**
  - Choosing victim for replacement on cache miss
    - FIFO, Least Recently Used, random
  - Handling writes (when is write accomplished?; is written data cached?)
    - Write allocate: allocates a cache block if a write results in a cache miss
    - Write through: all written data goes "through" cache and onto bus
    - Write back: written data remains in cache until cache block is replaced, then to bus
  - Ability to set or over-ride policies in software

Terminology

- **Cache functions:**
  - I-cache: holds instructions
  - D-cache: holds data
  - Unified cache: holds both instructions and data
  - Split cache: system with separate I-cache and D-cache

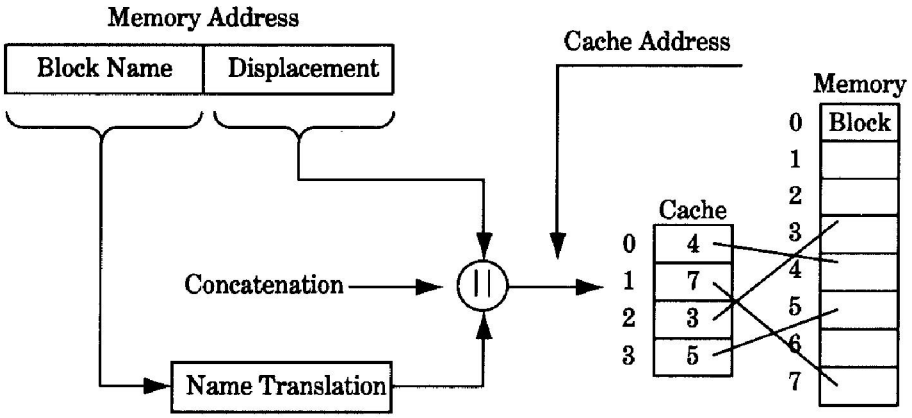


More Terminology

- **Hit - a cache access finds data resident in the cache memory**
- **Miss - a cache access does not find data resident, forcing access to next layer down in memory hierarchy**
- **Miss ratio - percent of misses compared to all accesses = P_{miss}**
  - When performing analysis, *always refer to miss ratio!*
  - 5% miss ratio compared to 10% miss ratio -- not 95% compared to 90% hit ratios
- **Traffic ratio - percent of data words to "bus" (or next level down) compared to data words transferred to "CPU" (or next level up)**
  - Caches can act as a "filter" to attenuate bus traffic; traffic ratio is an indication of effectiveness
- **Hit access time - number of clocks to return a cache hit**
- **Miss penalty - number of clocks to process a cache miss (typically, in addition to at least one clock of the hit time)**

Cache Treats Memory as a Set of Blocks

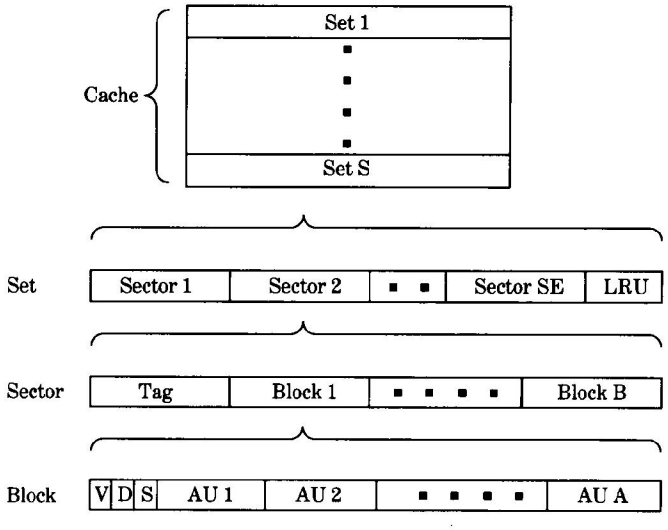
- **Cache is addressed using a partial memory address**
  - High-order bits determine correspondence between cache and memory block
- **Translation may include virtual to physical translation (covered later)**



(Cragon Fig. 2.2)

Elements of Cache Organization

- V, D, S, LRU are status bits
- Cragon's "AU" can also be called a "word" (*e.g.*, 32 bits)



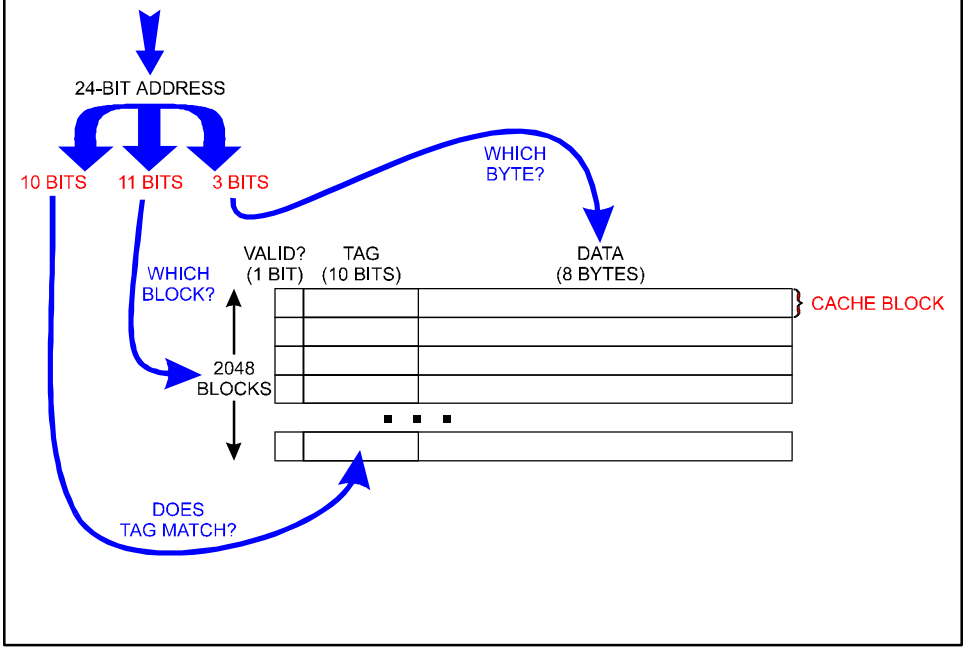
(Cragon Figure 2.3)

Per-Block Control bits

- **Control bits are per block (as opposed to tags per sector) to reduce memory accesses**
- **Valid: does entry contain valid data (as opposed to being "empty")**
  - Could be in middle of filling a cache sector
  - Could have been invalidated by a program or by coherence protocol
- **Dirty: is data modified from what is resident in memory?**
  - "write-back" cache holds modifications in cache until line is flushed to memory
- **Shared: is data shared within a multiprocessor (may be more than one bit depending on coherence scheme)?**
- **LRU: bit to track Least Recently Used replacement status**

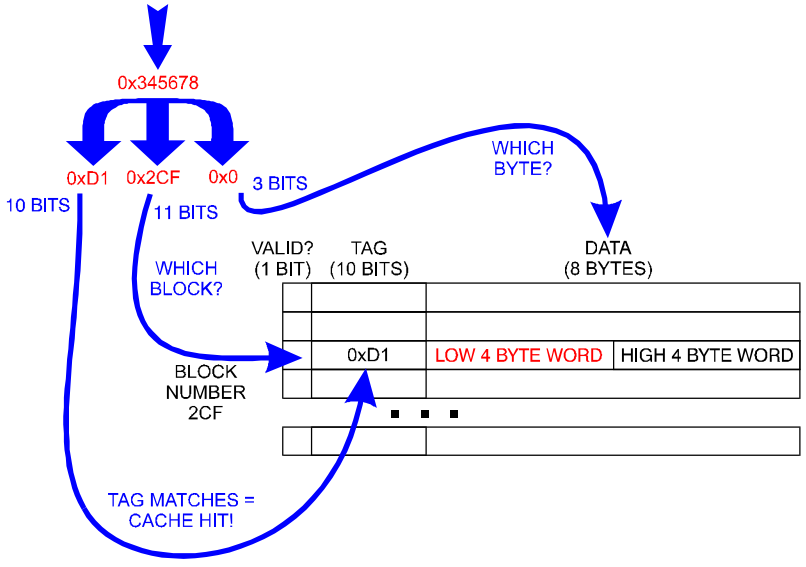
CACHE OPERATION

Conceptual Cache Operation



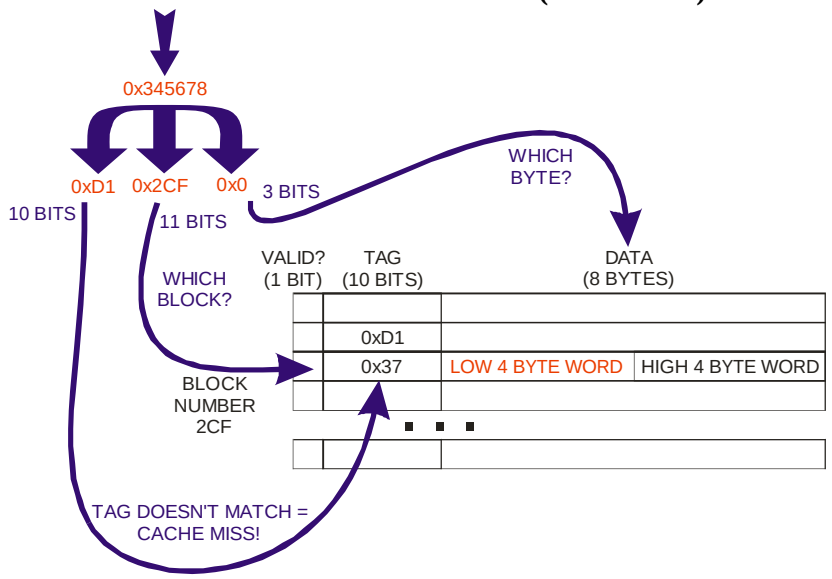
Example Cache Lookup Operation -- Hit

- **Assume cache "hit" at address 0x345678 (word access)**



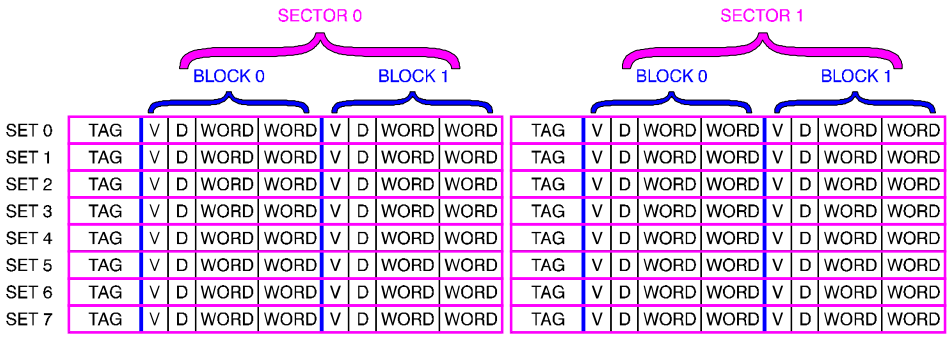
Example Cache Lookup Operation -- Miss

- **Assume cache "miss" at address 0x345678 (word access)**



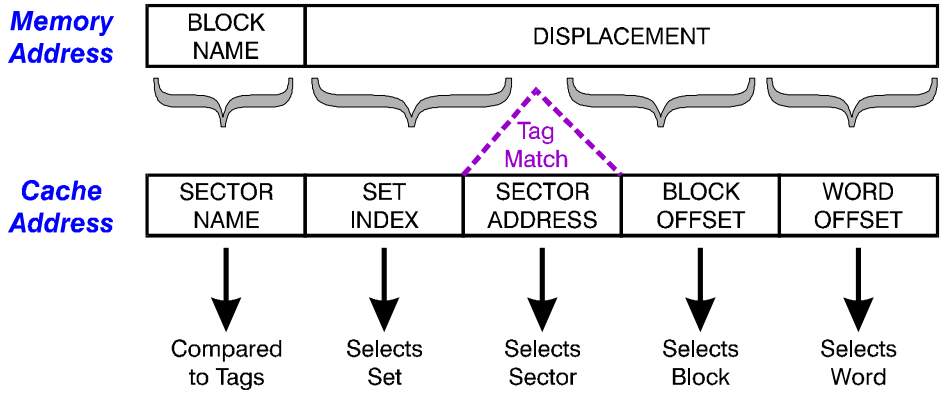
Example Cache Organization: [8,2,2,2]

- **2-way Set Associative**
- **[8 sets, 2 sectors, 2 blocks, 2 words] = 64 words in cache**
  - 2 words / block
  - 2 blocks / sector
  - 2 sectors / set
  - 8 sets / entire cache



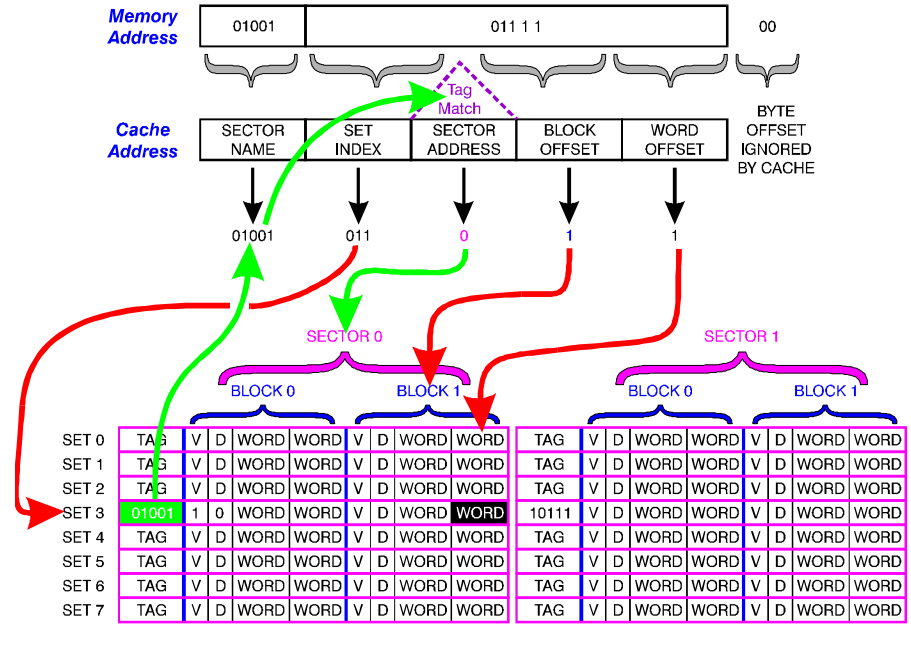
Address Formation (generalized)

- **Memory address broken into fields to compose cache address**



(improved from Cragon Figure 2.5)

Cache Addressing Example

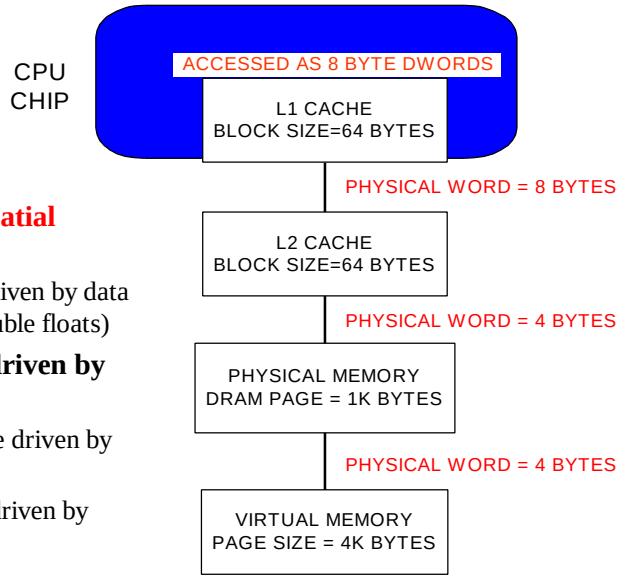


Cache Size

- **Number of Words (AUs) = [S x SE x B x W]**
  - S = SETs in cache
  - SE = SECTORs (degree of associativity) in set
  - B = BLOCKs in sector
  - W = WORDs in block

- **Example: [128, 4, 2, 8] cache**
  128 x 4 x 2 x 8 = 8 Kwords
  - 128 sets
  - 4 sectors per set (4-way set associative)
  - 2 blocks / sector
  - 8 words / block (for 32-bit words = 32 bytes/block)

Cache Interface Example



- **Block sizes exploit spatial locality**
  - CPU-L1 interface driven by data size (*e.g.*, 8-byte double floats)
- **Physical word sizes driven by pinout limitations**
  - Off chip L2 interface driven by pin count limits
  - Memory interfaces driven by memory bus costs

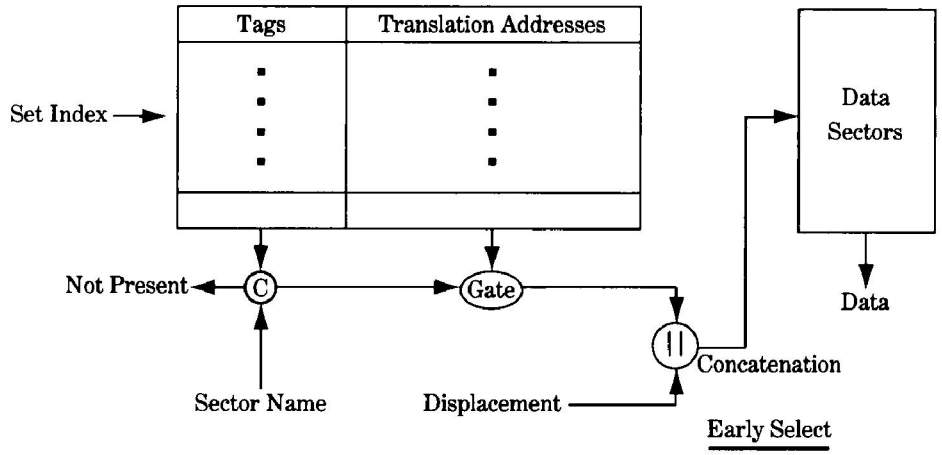
CACHE ACCESS SEQUENCE

Simplified Cache Functions

- **Determine if cache hit or miss**
- **If miss:**
  - Use policy information to determine whether a cache block must be allocated
  - If allocation required, select a cache location to allocate (a “victim”)
    - If selected location has modified data, copy data to lower hierarchy level
    - Set tag to map cache address to new memory level address
    - Fetch any required data from memory to fill cache word(s)
- **If read: return datum value**
- **If write: update datum value**

Early Select Cache

(Cragon Fig 2.11)

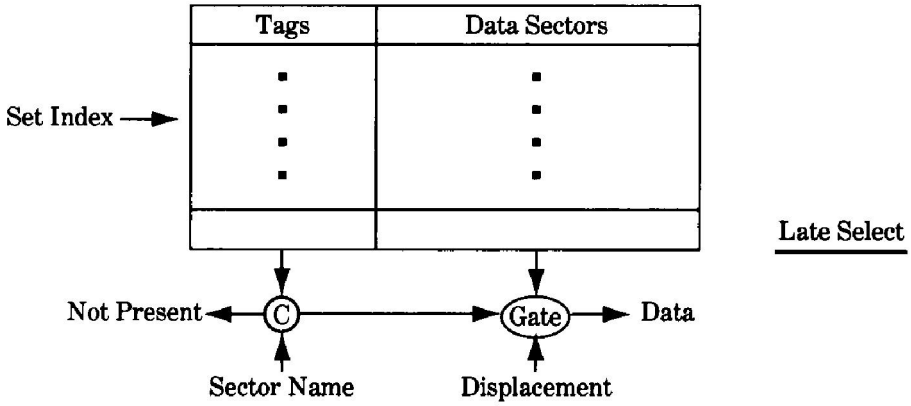


Early Select Operation

- **Tag lookup done *before* data access**
  - Data access waits for tag to be available
- **Translation address used to access data memory**
  - Flexibility of arbitrary data location ("fully associative")
  - *BUT* potentially slow critical path
- **Best used when tag matching is much faster than data retrieval**
  - This could happen when tags are in faster storage than data they refer to, *e.g.*:
    - On-chip tags but off-chip data memory
    - Tags in RAM but data on disk
    - Note that this is mostly applicable to virtual memory
  - It's not that the tag matching is free, but rather that the flexibility it gives (presumably) makes up for a relatively small addition to the serialized lookup path delay.
  - Also used on Alpha L2 cache for power savings
    - Only cycles one third of cache in 3-way set associative organization

Late Select Cache

- **Tag and data lookup in parallel; gate data at end of cycle**



(Cragon Fig 2.11)

Late Select Operation

- **Data lookup done in parallel with tag match**
  - Data either used or not used depending on whether any tag matched
  - Set associativity adds an element of late select operation
    - Which tag matches controls which data is gated
    - If no associativity, data may be available *before* knowing if there is a hit
- **Late select used for most cache memories**

CACHE MISSES

Baseline Actions on Cache Access

- **Read:**
  - Determine if data is in cache (hit or miss)
  - If miss, determine victim block for replacement
  - If victim sector has any “dirty” blocks (unsaved data), flush to memory
  - Annotate tag on victim sector to refer to new address
  - Fill victim block with new data (modify LRU bit; set valid bit; prefetch?)
- **Write:**
  - Determine if data is in cache (hit or miss)
  - If hit, write data to appropriate cache block
    - If write through policy, also write data to main memory
  - If miss, either:
    - If no-write-allocate policy, write data to main memory
    - If write allocate policy
      - » Perform a cache read if block size > size of written datum
      - » Write datum into cache
      - » If write through policy, also write data to main memory

Cache Misses

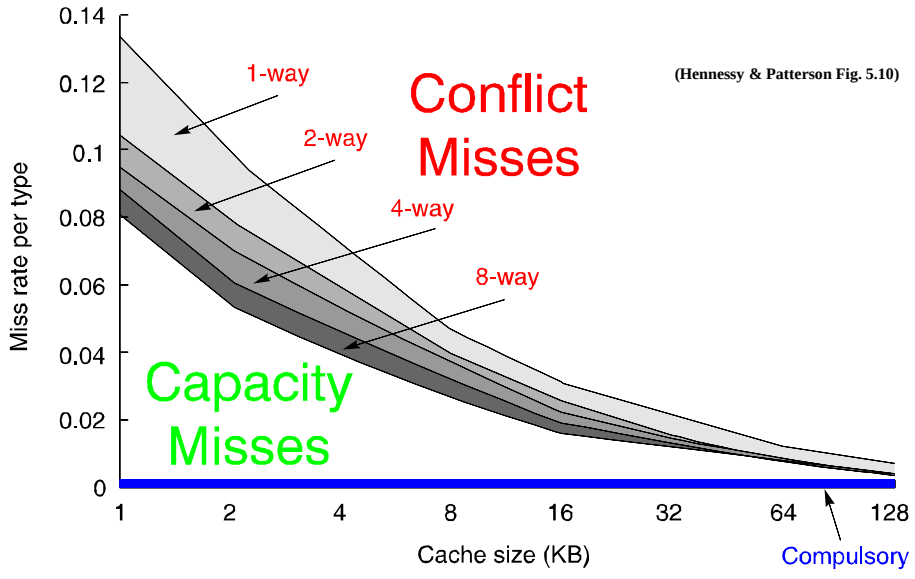
$P_{miss} = P_{compulsory} + P_{capacity} + P_{conflict}$

- **Compulsory:**
  - Previously unreferenced blocks
  - Compulsory miss ratio is that of an infinitely large cache
- **Capacity:**
  - Sectors that have been discarded to make room for new sectors
  - A fully associative cache experiences only compulsory and capacity misses
- **Conflict misses:**
  - Set associative caches must discard a sector within a set to allocate a block on a miss, even if other sets have unused sectors

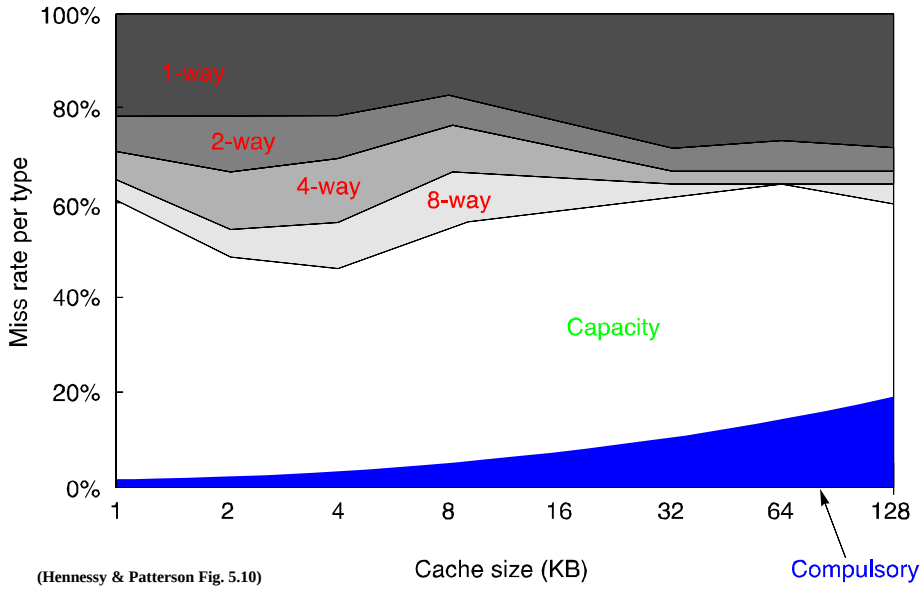
- **Warm cache: process has been running "a long time"**
- **Cold cache: cache has been flushed**
  - Some other task might have "stomped on" the cache
  - Software I/O synchronization might flush/invalidate cache

Example Absolute Miss Rates

- **Reducing associativity introduces conflict misses**



Example Relative Miss Rates



(Hennessy & Patterson Fig. 5.10)

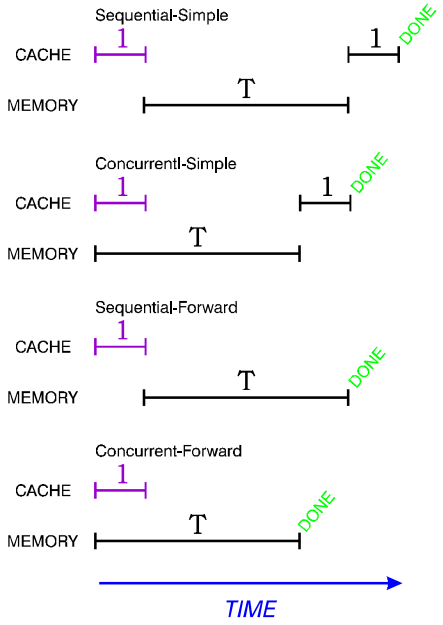
EFFECTIVE MEMORY ACCESS TIME

Cache Effectiveness Metrics

- **Access time: (LATENCY)**
  - t_{hit} often 1 clock cycle for L1 cache, 2 or more clocks for L2 cache
  - t_{miss} can be tens of clock cycles in many cases
  - t_{ea} = average access time (depends on miss ratio)
- **Bus traffic ratio: (BANDWIDTH)**
  ratio of bus traffic with cache to traffic without cache
  - Can be > 1with write-back caches
  - Traffic ratio can be critically important on multiprocessors, where cache is used as a filter to reduce shared bus traffic

Cache Access Strategies

- **Sequential -- memory access starts after cache miss**
- **Forward -- data available during cache write**
- **Sequential-Forward is commonly used model**
  - T starts with address sent to memory
  - T ends with data returned from memory
  - Memory not activated unless there is a cache miss



(After Cragon Fig. 2.12)

Effective Access Time

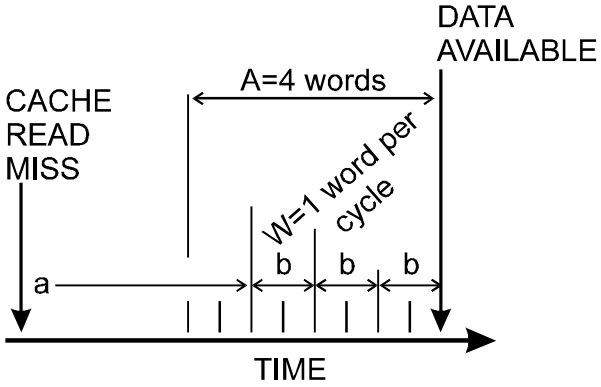
- $t_{ea} = (1 - P_{miss}) * t_{hit} + P_{miss} * t_{miss}$
- **Sequential-Forward is typical design**
  - Concurrent main memory reference hogs the memory bus
  - Forwarding reduces latency with minimal cost
- **"Transport time" = "miss penalty"**
- **For sequential-forward case:**
  - t_{ea} = cache access time Ü CACHE HITS
    + P_{miss} * transport time Ü CACHE MISSES

Simple (Read-only) Transport Time

- **Read model for either interleaved or burst caches**
  - **T** **transport time**
  - a latency to access first bus access
  - b latency for each subsequent bus accesses
  - A number of words in block
  - W number of words that fit on access bus (width of bus)

$$T = a - b + \left(\frac{b * A}{W}\right)$$

$$T = a + b\left(\frac{A}{W} - 1\right)$$



Simple (Read-only) Bus Traffic Model

- **Bus traffic ratio = P_{miss} * block size**

- **Example:**
  - P_{miss} = 0.05
  - block size = 4 words
  - Traffic ratio = 0.05 * 4 = 0.20 memory words per memory reference

- **Bus delay (lower bound on execution time)**
  - Traffic ratio * Transport Time
  - For 4+1+1+1 access: 0.20 * 7 = 1.4 clocks per memory reference

REVIEW

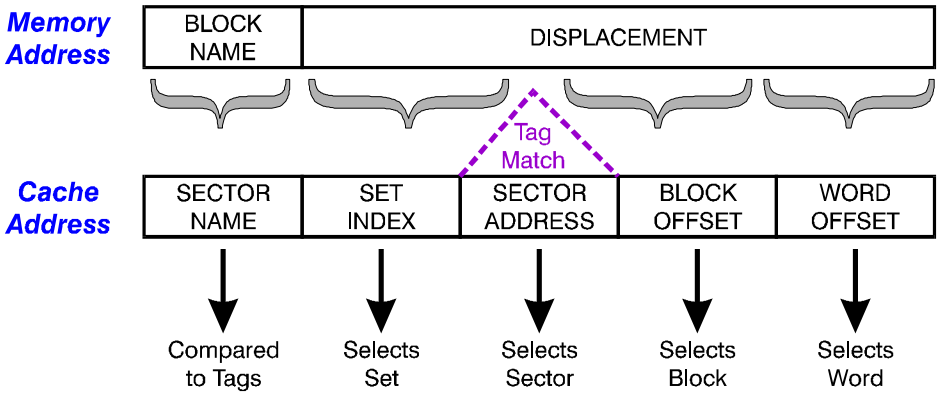
Review

- **Locality: temporal, spatial, sequential**
- **Cache organization**
  - Sets, sectors, blocks
    - [*S* sets, *SE* sectors, *B* blocks, *W* words] = S * SE * B * W words in cache
  - Tags, dirty, shared, LRU
- **Cache operation**
  - Search for match, decide if miss
  - Select & update contents
- **Cache misses: compulsory, capacity, conflict**
  - $t_{ea} = (1 - P_{miss}) * t_{hit} + P_{miss} * t_{miss}$

$$T = a + b\left(\frac{A}{W} - 1\right)$$

Address Formation (generalized)

- **Memory address broken into fields to compose cache address**



(improved from Cragon Figure 2.5)

Key Concepts

- **Latency**
  - Late select minimizes latency by accessing data & tag in parallel
  - Average cache access latency depends on compulsory, capacity, conflict misses
  - Transport time depends on miss ratios & access latencies
- **Bandwidth**
  - Block size determines bandwidth opportunities (can move an entire block at a time, data paths permitting)
  - Bus traffic ratio depends on miss rate and block size
- **Concurrency**
  - Multiple blocks per sector permits sharing cost of tag in cache
- **Balance**
  - Sector size (to save space) *vs.* number of sets in cache (to gain flexibility) is a key balance tradeoff that will be discussed in another lecture
  - Transport time (miss rate) vs. bus traffic ratio is another key balance issue

RECITATION MATERIAL

Example Read Transport Time

- **Embedded 32-bit CPU; cache block size of 4 x 32-bit words; 8-bit memory bus at half CPU speed; 1 byte/bus cycle + 2 bus cycles overhead to start transfers**
- **All clocks are in terms of CPU clocks**
  - a = 4+2=6 clocks (4 clocks to reach memory + 2 clocks first byte xfer)
  - b = 2 clocks (CPU clock cycles twice for each bus access)
  - A = 4 (4 @ 32-bit words in a block)
  - W = 0.25 (.25 words (1 byte of 4) transferred per bus cycle)

$$T = a + b\left(\frac{A}{W} - 1\right)$$

$$T = 6 + 2\left(\frac{4}{0.25} - 1\right) = 6 + 2(15) = 36 \text{ CPU clocks}$$