Chapter 7: The Memory Hierarchy

Computers have several **Levels of Memory**:

- **Registers** – on processor, very fast, small capacity
- **Cache** – on/near processor, faster than main memory, smaller capacity
- **Main Memory (DRAM)** – larger capacity, slower than cache
- **Disk –** largest capacity, very slow

Memory Transfers Controlled By:

- Registers ↔Main Memory: Compiler or Programmer
- Cache ↔ Main Memory: Cache Controller
- Main Memory ↔ Disk: Operating System

Key Points:

- Each level trades off capacity vs. speed
- Information flows between adjacent levels in the hierarchy
- Different components control moving data between levels

7.1 Caching

Memory closer to CPU is faster but more expensive; caching copies subset of main memory close to CPU; multiple layers of caches before reaching main memory

- Locality:
  - *Temporal locality* - if data used now, likely to use again later
  - *Spatial locality* - if data used now, nearby data likely used soon
- Caches exploit locality:
  - *Temporal* - keep recently used data in cache
  - *Spatial* - cache blocks of data including nearby data
- Benefits:
  - Caching leverages faster memory to optimize performance
  - Improves access time by having frequently used data in fast cache
- **Direct Mapped Cache**

  Data transferred between memory and cache in fixed blocks; each address maps to exactly one block in the cache; only need to check one location in cache for a given address
  - Address Split Into:
    - Offset - byte within a block
    - Index - specifies which block in the cache
    - Tag - remaining bits, uniquely identifies block
  - Cache Lookup:
    - Index specifies which cache block to check
    - Tag in cache compared against tag in address
    - Match indicates block is cached

    - Key Points of Direct Mapped Cache:
        - Like a hashmap - index is key, tag is value
        - Cache stores both data block and tag
        - Each address can only be in one cache block
        - Benefits:
            - Simple, fast lookup by only checking one cache location
- **N-way set associative caches**

  Cache is divided into sets which each contain N blocks; each address maps to one set based on index bits; the set acts as a fully associative cache
    - Address Split Into:
        - Offset - byte within block
        - Index - selects which set
        - Tag - remaining bits
    - Cache Lookup:
        - Use index to select a set
        - Check tags of all N blocks in the set
        - Matching tag indicates block is cached
    - Benefits:
        - More flexible than direct mapped
        - Don't have to check entire cache
        - Only need N comparators per set
        - Combines advantages of direct mapped and fully associative caches
        - Reduces conflicts compared to direct mapped
- **Fully Associative Cache**

  Cache is divided into blocks; each block contains a tag and data; no concept of positioning within cache
    - Address Split Into:
        - Offset - byte within block
        - Tag - remaining bits
    - Cache Lookup:
        - Check tags of all blocks in parallel
        - Matching tag indicates block is cached
    - Benefits:
        - No conflict misses
        - Any address can map to any block
    - Drawbacks:
        - Need comparator for every cache block

        - Entire cache must be searched for a hit
        - Like searching an array
        - More flexible mapping than direct mapped
- **Cache Performance Metrics**
    - Cache Access Flow:
        - CPU issues address to cache
        - Cache hit - data sent from cache to CPU
        - Cache miss - data loaded from memory to cache, then sent to CPU
    - Definitions:
        - Cache hit - requested data is in cache
        - Cache miss - requested data not in cache
    - Miss Types:
        - Compulsory - cache empty when program starts
        - Conflict - multiple addresses map to same block
        - Capacity - cache is full
    - Metrics:
        - Hit rate - fraction of accesses that hit
        - Miss rate - fraction of accesses that miss
        - Miss penalty - time to replace missed data
        - Hit time - time to access cached data
    - Key Ideas:
        - Want high hit rate to avoid miss penalty
        - But also want low hit time
        - Balancing act based on cache size, block size, associativity
- **Cache Write policies**
    - Write-Through Cache:
        - Writes update both cache and main memory
        - Memory always consistent with cache
        - More writes to main memory
    - Write-Back Cache:
        - Writes only update cache initially
        - Memory updated when replaced from cache
        - Adds dirty bit to track modified blocks
        - Allows inconsistent cache and memory
    - Benefits:
        - Write-through is simpler, ensures consistency
        - Write-back reduces writes to slow main memory
        - Write-back allows multiple writes before updating memory
    - Trade Offs:

        - Write-through has more memory traffic
        - Write-back has complexity of dirty bits
        - Choose policy based on write frequency
- **Cache Block Replacement policies**:
    - Direct Mapped:
        - Only option is to replace whole block at index
    - N-Way & Fully Associative:
        - Can choose which block to replace on miss
    - FIFO Replacement:
        - Remove the oldest block in set/cache
        - Doesn't consider block access frequency
    - LRU Replacement:
        - Remove least recently used block
        - Ensures frequent blocks stay in cache
    - Summary:
        - Direct mapped only does whole block replacement
        - FIFO is simple but can remove frequent blocks
        - LRU keeps hot data in cache longer
        - Policy matters once sets/cache are full

7.2 Virtual Memory

Here are the key points on **virtual memory and paging**:

- Modern computers run multiple processes simultaneously
- Virtual memory allows programs to use virtual addresses
- OS maps virtual to physical addresses

Address Spaces (Both split into pages [like cache blocks]):

- **Virtual** - addresses seen by program
- **Physical** - actual addresses in RAM

Page Table:

- Maps virtual pages to physical pages
- Stores access rights for each page

Page Table Base Register:

- Points to location of page table in memory

TLB (Translation Lookaside Buffer):

- Caches page table entries
- Avoids accessing page table on every memory reference

  Key Ideas:
    - Virtual memory allows oversubscription of physical memory
    - OS manages mapping of virtual to physical addresses
    - TLB caches mappings for faster lookup

Here are **the key steps for memory access using virtual addresses**:

1. Split virtual address into VPN and page offset
2. Split VPN into TLB tag and TLB index
3. Check TLB at index for matching tag
    - If hit, TLB returns corresponding PPN
4. If TLB miss:
    - Use VPN to look up page table
    - Page table provides PPN
5. Concatenate PPN with page offset
    - Gives physical address
6. Use physical address to access cache/memory

   Key points:
    - TLB caches page table lookups
    - TLB hit avoids slow page table access
    - TLB miss requires full page table lookup
    - Physical address used for actual memory access

7.2.2 Virtual Memory and the Disk

- Pages not in physical memory stored on disk (swap space)
- OS manages transferring pages between disk and memory
- Page fault occurs when page needed but not in physical memory
- On page fault:
    - OS suspends process
    - Loads required page from disk to free page in memory
    - Updates page table
    - Resumes process
- **Thrashing**: when pages constantly moved between disk and memory
    - Process runs → page fault → suspend, load page → resume → page fault...
    - Overall throughput tanks due to overhead of swapping

7.2.3 Virtual Memory Performance Metrics

TLB Miss Rate:

- Fraction of TLB lookups that miss
- Miss requires full page table walk

Page Fault Rate:

- Fraction of page table lookups that fault
- Miss requires loading page from disk
- Much higher penalty than TLB miss
- Want to minimize both rates to avoid overhead
- TLB miss just costs extra memory access
- Page fault costs disk access and context switch