Chapter 10
MEMORY SYSTEM

In this chapter

- Memory design objectives
- Memory hierarchy
- Average latency
- Cache mapping
- Cache coherency
- Virtual memory

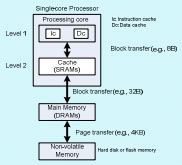
Why a Memory System?

- No single memory technology exists today that meets design objectives:
  - High speed (low latency)
    1. SRAM, the fastest
    2. SDRAM
    3. Flash
    4. Magnetic disk, the slowest
  - large capacity
    - Any size non-volatile (disk and flash) memory is readily available
  - low cost (e.g., per byte)
    1. Hard disk and flash memory, the cheapest
    2. SDRAM
    3. SRAM as cache memory, the most expensive
- Other requirements
  - Low power
  - Small physical size
  - Error correction
- What is the solution to meet the design objective?

  Use different memory technologies in hierarchy

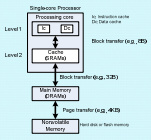
Memory Hierarchy (why it works)

- Programs contain loops
  - Instructions in a loop execute repeatedly
    - This is known as Temporal locality
- Instructions executed and data typically accessed in sequence
  - This is known as spatial locality
- Keeping recently accessed instructions and data in cache minimizes reduces average memory latency
  - Memory system behaves as if it is high speed



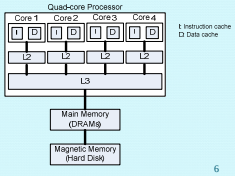
Memory Hierarchy (how it works)

- Disk space for program code and data viewed as virtual memory divided into pages (e.g., 4KB each)
- Code and data pages are copied from virtual memory to physical memory (SDRAM) as needed
  - This is known as virtual memory management (later)
  - DMA and page-mode SDRAM access accelerate transfers between disk and memory
- Code and data in physical memory divided into blocks
  - E.g., 32B or 64B blocks
  - SDRAM burst mode access accelerates transfers between memory and cache
- Bust transfer is also used between cache levels



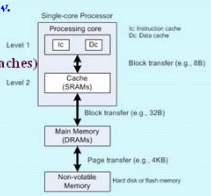
UMA within Processor

- Typically 3 levels of caches
  - L1 instruction and data caches
    - Smaller, low latency
  - L2 cache holds both instructions and data
  - Shared L3 cache, minimizes accesses to SDRAM
    - E.g., Blocks shared by two threads accessed once from shared cache
- Cache hit: If cache has target instruction or data, else cache miss
- Hit ratio or hit rate
  - % of times cache access is a hit
- Miss ratio or miss rate
  - 1 – hit ratio



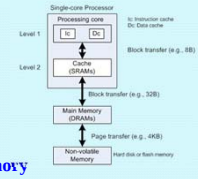
Average Memory Hierarchy Latency

- Example, consider the memory system below:
  - IC and DC
    - latency = 1ns
    - hit ratio = 0.95 (assume the same for both the caches)
  - L2 cache
    - 32B block size (also called cache line),
    - latency = 3ns
    - hit ratio = 0.9
  - Main memory
    - 400MHz SDRAMs
    - 32-bit Data bus
- Program execution suspended if block not in memory
- Average latency, depends on hit ratios
  - Estimate, assume peak performance

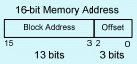


Average Latency = (0.95)(1ns) + (1 – 0.95)(0.90)(3ns) + (1 – 0.95)(1 – 0.90)(20ns)
= 0.95ns + 0.135ns + 0.1ns
= 1.185ns

Cache Mapping (Example)



- Consider two memory levels:
  - L2 cache and main memory
  - Main memory: 64KB = 2^{16}B
  - L2 Cache: 1KB = 2^{10}B
  - Block (line) size: 8B
1. Determine number of blocks in main memory
   64KB/8B = 8192 or 2^{13} blocks
2. Determine number of block in cache, also called slots
   1KB/8B = 128 or 2^{7} blocks
3. Partition a memory address into two regions
   - block address
   - block offset

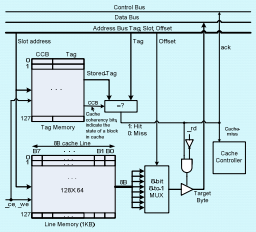


Direct Mapping

- Determine slot number
  - Use modular arithmetic
    - Easy when slot numbers are powers of 2
  - Examples:
    - 129 = (000001,0000001)
      - 129 mod 128 = 1, low 7 bits
    - 1153 = (001001,0000001)
      - 1153 mod 128 = 1, lower 7 bits
- Determine tag value
  - Use integer division
    - Easy when slot numbers are powers of 2
  - Examples:
    129 / 128 = 1, upper 6 bits
    1153 / 128 = 9, upper 6 bits

Direct Mapped Cache Data Path

- Two SRAM memories accessed at the same time
  - One to store tags
  - Another to store cached blocks
- Tags compared
  - cache hit if tags match
- Cache miss triggers access from lower-level memory

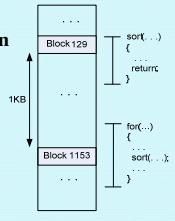


Types of Cache Misses

- cold (compulsory) miss
  - When cache is initially empty
- conflict miss
  - When two addresses map to same slot in cache
  - E.g., blocks 129 and 1153
- capacity miss
  - Conflict misses that if cache was made bigger will become hits
- Others (for shared blocks)
  - True-sharing miss
  - False-sharing miss

Set-Associative Mapping

- Limitations of Direct Mapped Caches
  - Consider program example below when
  - Blocks 1153 and 129 keep causing conflict misses as for-loop executes
- Set-associative reduces the probability of such conflicts

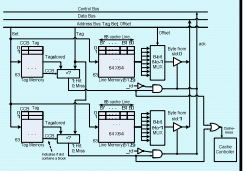


Set-Associative mapping

- Cache is organized as 2, 3, etc. slots per set
- Memory blocks are direct mapped to to sets (not to slots)
- Example:
  - Consider 128 slots organized as 2-way set-associative cache
  - Blocks 1153 and 129 would map to set 1 with two slots
    - Block 1153: Maps to set 1, stored in slot 0
    - Block 129: Maps to set 1, stored in slot 1
- With set-associative mapping there are only two cold misses for blocks 1153 and 129 in the example program example

2-way Set-associative Data path

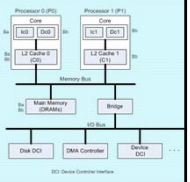
- Two sets of tag and data memories
- Slots with a set searched in parallel
  - Consumes more power
- Way-prediction can reduce power consumption
  - Miss-prediction however would result in higher cache latency



Cache Coherency

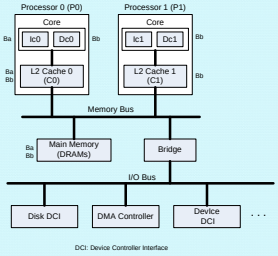
- Problem
  - Copying blocks from memory to cache results in having multiple copies in the system.
  - Copies may not be the same
- Simple definition for cache coherency:

  Accessing memory location X returns the last value written to X no matter where last write took place (in caches or memory)



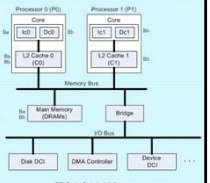
Where do writes come from?

- From DMA transfers
- From other processing cores
  - Consider two threads executing on two separate cores/processors share a cache block



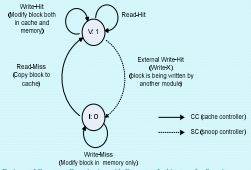
Snoopy Cache

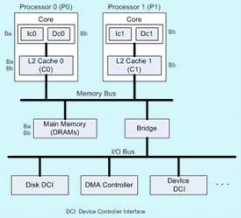
- Each snoopy cache uses two controllers
  1. Cache controller
     - responds to requests from processing core or higher-level cache
  2. Snoop controller
     - Responds to request from bus or lower-level cache



Write-Through Protocol

- The state-diagram determines the state of each block in cache
  - Solid lines: this cache
  - Dotted line: another cache
- Can increase memory traffic





Write-back Protocols

- MESI Used by many in UMA systems
  - M, modified
  - E, exclusively owned
  - S, two or more shared copies
  - I, invalid or not present
- MESIF used by Intel in NUMA systems
  - F, forward
- MOESI used by AMD in NUMA systems
  - O, owned

Virtual Memory Management (1)

- Objectives:
  1. Run multiple processes (running programs)
  2. Run large processes, one with many instructions and large data structures too big to fit in memory
  3. Protect processes from one another
     - Each process should accesses its own memory space

Virtual Memory Management (2)

- How does it work?
  - Virtual page contents are copied to main memory as needed from disk
  - Unlike caches, mapping is fully-associative
    - A virtual page content can be stored in any page in memory
  - Mapping info stored in a table, called page table
  - Page tables are stored in memory or even on hard disk

How Virtual-Physical Page Mapping Works?

- Virtual page number is mapped to a Page Table entry
- Each entry contains page status and physical page number (if any)
- Page table may be organized as multi-level if it is too big
- Latest mapping info kept in a specialized cache called Translation Lookaside Buffer (TLB)

Processor Organization

- Three ways to organize TLB and caches

1. TLB before caches
   - Advantage:
     - Physically addressed caches
   - Disadvantage:
     - L1 cache latency longer
2. Virtually addressed caches
   - Advantage:
     - No change in L1 cache latency
   - Disadvantage:
     - Caches must be flushed on process switch
3. Combined TLB and L1 Cache hardware
   - Advantage:
     - No change in L1 cache latency
     - Physically addressed caches (i.e., virtually addressed but physically tagged)
   - Disadvantage:
     - Cache size = page size, can be too small for high end processors