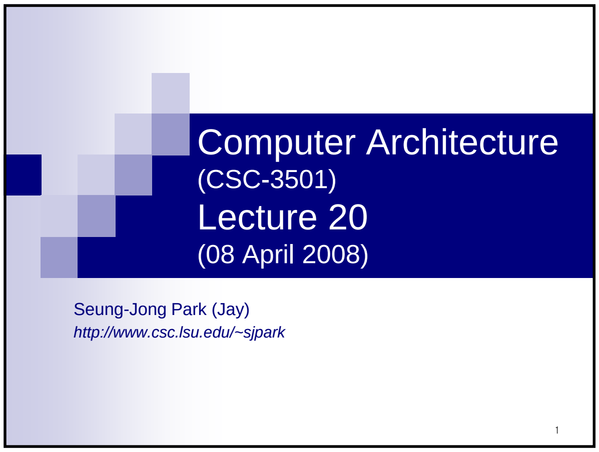


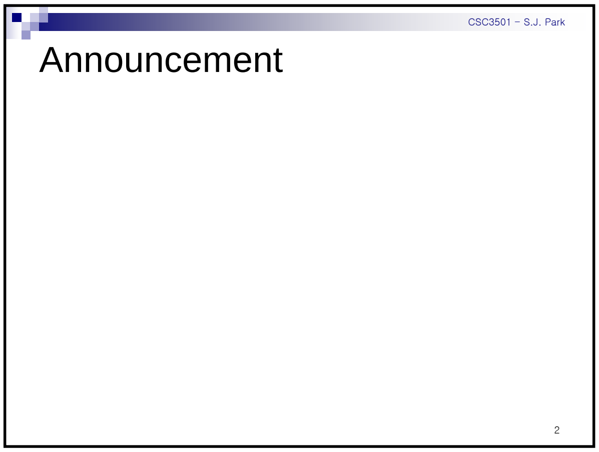
Computer Architecture

(CSC-3501)

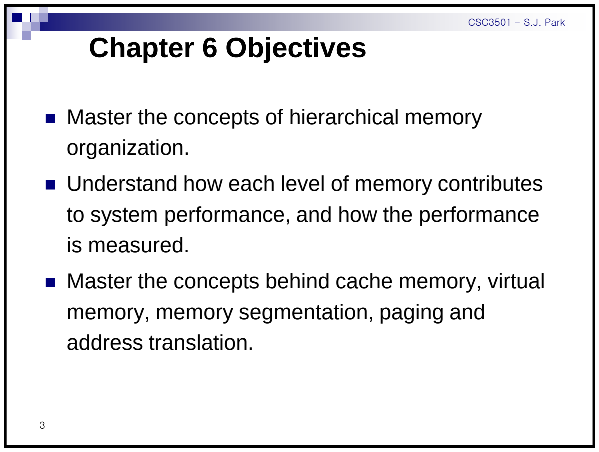
Lecture 20

(08 April 2008)

Seung-Jong Park (Jay)
http://www.csc.lsu.edu/~sjpark

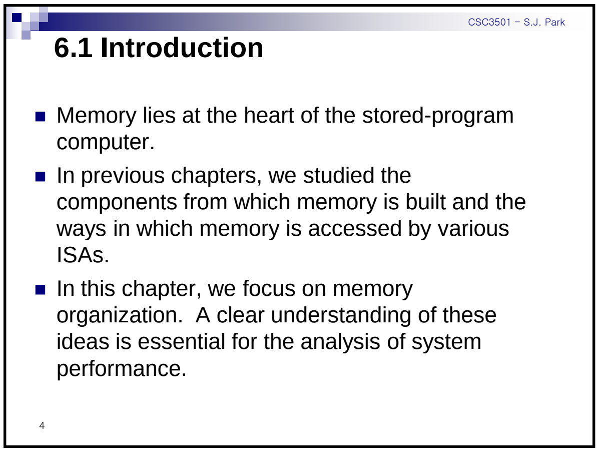


Announcement



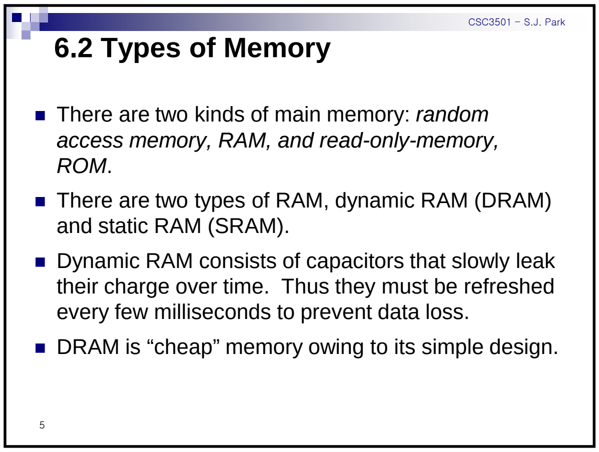
Chapter 6 Objectives

- Master the concepts of hierarchical memory organization.
- Understand how each level of memory contributes to system performance, and how the performance is measured.
- Master the concepts behind cache memory, virtual memory, memory segmentation, paging and address translation.



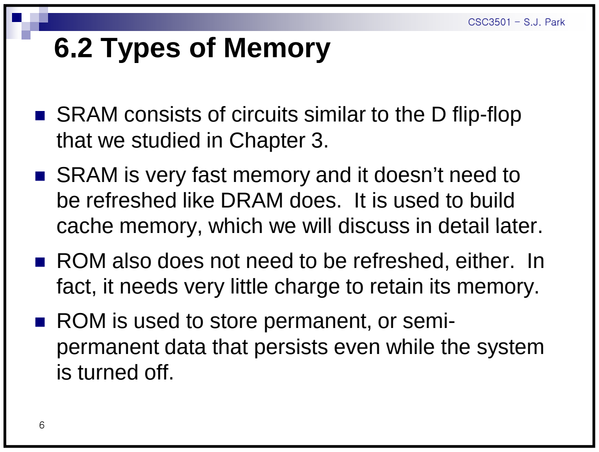
6.1 Introduction

- Memory lies at the heart of the stored-program computer.
- In previous chapters, we studied the components from which memory is built and the ways in which memory is accessed by various ISAs.
- In this chapter, we focus on memory organization. A clear understanding of these ideas is essential for the analysis of system performance.



6.2 Types of Memory

- There are two kinds of main memory: *random access memory, RAM, and read-only-memory, ROM*.
- There are two types of RAM, dynamic RAM (DRAM) and static RAM (SRAM).
- Dynamic RAM consists of capacitors that slowly leak their charge over time. Thus they must be refreshed every few milliseconds to prevent data loss.
- DRAM is “cheap” memory owing to its simple design.

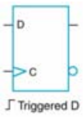


6.2 Types of Memory

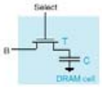
- SRAM consists of circuits similar to the D flip-flop that we studied in Chapter 3.
- SRAM is very fast memory and it doesn’t need to be refreshed like DRAM does. It is used to build cache memory, which we will discuss in detail later.
- ROM also does not need to be refreshed, either. In fact, it needs very little charge to retain its memory.
- ROM is used to store permanent, or semi-permanent data that persists even while the system is turned off.

Memories: Review

- SRAM:
  - □ value is stored on a pair of inverting gates
  - □ very fast but takes up more space than DRAM (more # of transistors)



- DRAM:
  - □ value is stored as a charge on capacitor (must be refreshed)
  - □ very small but slower than SRAM (factor of 5 to 10)
  - □ Capacitor can hold charge
  - □ Transistor acts as gate
  - □ No charge is a 0
  - □ Can close switch & add charge to store a 1
  - □ Then open switch (disconnect)
  - □ Can read by closing switch

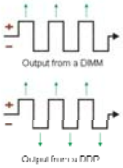


Synchronous DRAM (SDRAM)

- Has a clock
- Common type in PCs late-90s
  - □ Typical DRAMs still synchronous
- Multiple *banks*
- Pipelined
  - □ Start read in one bank after another
  - □ Come back and read the resulting values one after another

DDR DRAM

- Double Data Rate SDRAM
- Transfers data on both edges of the clock
- Currently popular
- Attempt to alleviate the pinout problems



Summary - Memory

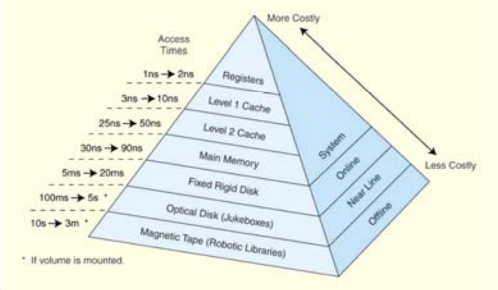
- RAMs with different characteristics
  - □ For different purposes
- Static RAM
  - □ Simple to use, small, expensive
  - □ Fast, used for cache
- Dynamic RAM
  - □ Complex to interface, largest, cheap
  - □ Needs periodic refresh

6.3 The Memory Hierarchy

- Generally speaking, faster memory is more expensive than slower memory.
- To provide the best performance at the lowest cost, memory is organized in a hierarchical fashion.
- Small, fast storage elements are kept in the CPU, larger, slower main memory is accessed through the data bus.
- Larger, (almost) permanent storage in the form of disk and tape drives is still further from the CPU.

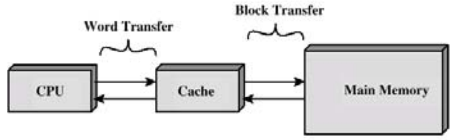
6.3 The Memory Hierarchy

- This storage organization can be thought of as a pyramid:



6.3 The Memory Hierarchy

- To access a particular piece of data, the CPU first sends a request to its nearest memory, usually cache.
- If the data is not in cache, then main memory is queried. If the data is not in main memory, then the request goes to disk.
- Once the data is located, then the data, and a number of its nearby data elements are fetched into cache memory.



6.3 The Memory Hierarchy

- This leads us to some definitions.
  - □ A *hit* is when data is found at a given memory level.
  - □ A *miss* is when it is not found.
  - □ The *hit rate* is the percentage of time data is found at a given memory level.
  - □ The *miss rate* is the percentage of time it is not.
  - □ Miss rate = 1 - hit rate.
  - □ The *hit time* is the time required to access data at a given memory level.
  - □ The *miss penalty* is the time required to process a miss, including the time that it takes to replace a block of memory plus the time it takes to deliver the data to the processor.

6.3 The Memory Hierarchy

- An entire blocks of data is copied after a hit because the *principle of locality* tells us that once a byte is accessed, it is likely that a nearby data element will be needed soon.
- There are three forms of locality:
  - □ *Temporal locality*- Recently-accessed data elements tend to be accessed again.
  - □ *Spatial locality* - Accesses tend to cluster.
  - □ *Sequential locality* - Instructions tend to be accessed sequentially.

6.4 Cache Memory

- The purpose of cache memory is to speed up accesses by storing recently used data closer to the CPU, instead of storing it in main memory.
- Although cache is much smaller than main memory, its access time is a fraction of that of main memory.
- Unlike main memory, which is accessed by address, cache is typically accessed by content; hence, it is often called *content addressable memory*.
- Because of this, a single large cache memory isn’t always desirable-- it takes longer to search.

6.4 Cache Memory

- The “content” that is addressed in content addressable cache memory is a subset of the bits of a main memory address called a *field*.
- The fields into which a memory address is divided provide a many-to-one mapping between larger main memory and the smaller cache memory.
- Many blocks of main memory map to a single block of cache. A *tag* field in the cache block distinguishes one cached memory block from another.

6.4 Cache Memory

- Different Kinds of Cache
  - □ Direct Mapped Cache
  - □ Set Associative Cache
  - □ Fully Associative Cache

6.4 Cache Memory (Direct Mapped Cache)

- The simplest cache mapping scheme is *direct mapped cache*.
- In a direct mapped cache consisting of *N* blocks of cache, block *X* of main memory maps to cache block *Y* = *X* mod *N*.
- Thus, if we have 10 blocks of cache, block 7 of cache may hold blocks 7, 17, 27, 37, . . . of main memory.
- Once a block of memory is copied into its slot in cache, a *valid* bit is set for the cache block to let the system know that the block contains valid data.

What could happen if there were no valid bit ?

6.4 Cache Memory(Direct Mapped Cache)

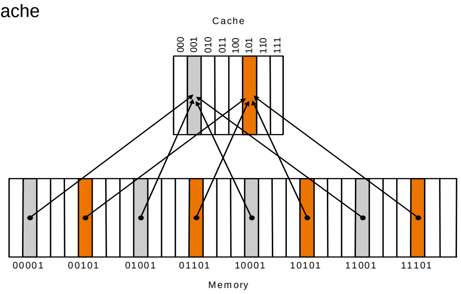
- The diagram below is a schematic of what cache looks like.

Block	Tag	Data	Valid
0	00000000	words A, B, C,...	1
1	11110101	words L, M, N,...	1
2	--------------		0
3	--------------		0

- Block 0 contains multiple words from main memory, identified with the tag 00000000. Block 1 contains words identified with the tag 11110101.
- The other two blocks are not valid.

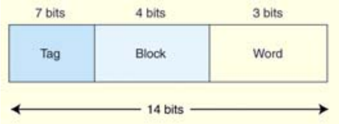
Direct Mapped Cache

- Mapping: address is modulo the number of blocks in the cache



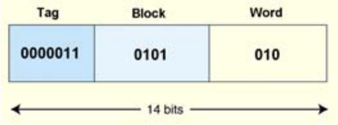
6.4 Cache Memory(Direct Mapped Cache)

- The size of each field into which a memory address is divided depends on the size of the cache.
- Suppose our memory consists of 2^{14} words, cache has 16 = 2^4 blocks, and each block holds 8 words.
  - □ Thus memory is divided into 2^{14} / 2^{8} = 2^{11} blocks.
- For our field sizes, we know we need 4 bits for the block, 3 bits for the word, and the tag is what's left over:



6.4 Cache Memory(Direct Mapped Cache)

- As an example, suppose a program generates the address **1AA**. In 14-bit binary, this number is: **00000110101010**.
- The first 7 bits of this address go in the tag field, the next 4 bits go in the block field, and the final 3 bits indicate the word within the block.



Tag	Block	Word
0000011	0101	010

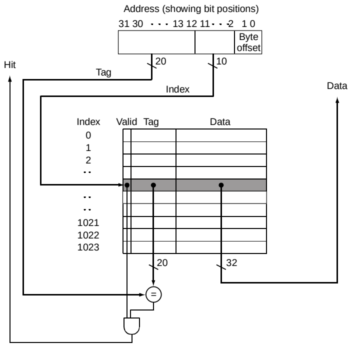
6.4 Cache Memory(Direct Mapped Cache)

- If subsequently the program generates the address **1AB**, it will find the data it is looking for in block **0101**, word **011**.

Tag	Block	Word
0000011	0101	010

- However, if the program generates the address, **3AB**, instead, the block loaded for address **1AA** would be evicted from the cache, and replaced by the blocks associated with the **3AB** reference.

Direct Mapped Cache



6.4 Cache Memory

- Suppose a program generates a series of memory references such as: **`1AB, 3AB, 1AB, 3AB`**, . . . The cache will continually evict and replace blocks.
- The theoretical advantage offered by the cache is lost in this extreme case.
- This is the main disadvantage of direct mapped cache.
- Other cache mapping schemes are designed to prevent this kind of thrashing.