CS24: Introduction to Computing Systems

Spring 2015
Lecture 13
COMPUTER MEMORY

- So far, have viewed computer memory in a very simple way
- Two memory areas in our computer:
  - The register file
    - Small number of addressable locations
    - Keeps instruction size reasonably small
  - Main memory
    - Very large number of addressable locations
    - Use registers to address memory
    - RISC: Introduce load/store instructions for memory access
    - CISC: Directly encode addresses into many instructions

- Also know (intuitively) that:
  - Registers are fast to access; main memory is slower
  - Data alignment is also a performance consideration
**Computer Memory, In Practice**

- In practice:
  - Small memories tend to be faster than large ones
  - Larger memories are denser than small ones
    - Also affects speed of access
- Due to physical constraints!
- Signals take time to propagate through a wire
  - Travel ~1 foot in 1 nanosecond (Grace Hopper)
    - 3GHz signal $\Rightarrow$ ~4 inches per clock
  - Physical size of circuits is *directly* linked to how fast they can go!
  - Drives chip vendors’ efforts to shrink manufacturing processes
    - The smaller the chip, the shorter the signal paths
    - The shorter the signal paths, the faster it can be clocked
Signals take even more time to propagate through gates

- Propagation delay: time it takes for the output of a gate to become stable, once its inputs become stable
  - e.g. 0.5 ns to 5+ ns
  - The more things a gate feeds its signal to, the greater the propagation delay
  - The longer a wire a gate must drive, the greater the propagation delay
- The more gates a signal must propagate through, the slower the device

Both physical limits are affected by temperature
- The hotter a circuit is, the slower it performs
- Cooling a circuit improves its performance
**Example: Memory Addressing**

- Use multiplexers and demultiplexers to retrieve individual locations
  - Fixed-size component for accessing memories of different sizes
  - For now, ignore how to store individual values

- A simple 4-cell memory:
  - Demultiplexer to direct input data to addressed cell
  - Multiplexer to direct output data from addressed cell
EXAMPLE: MEMORY ADDRESSING (2)

- Now, scale our memory up to 16 locations:

- Need two levels of multiplexers and demultiplexers to decode addresses
EXAMPLE: MEMORY ADDRESSING (3)

- \( T_{\text{mem}} = T_{\text{addr}} + T_{\text{cell}} \)
  - \( T_{\text{addr}} \) = time spent in addressing logic
  - \( T_{\text{cell}} \) = time the memory cell takes to read or write a value
  - \( T_{\text{cell}} \) is constant, regardless of memory size

For \( N \) memory locations, \( T_{\text{addr}} = ?? \)

- \( T_{\text{addr}} = C \times \log(N) \)
- As memory grows larger, access time increases.
Example: Memory Access

- Memory cells laid out on a flat 2D surface
  - silicon wafer
- A reasonable layout minimizes wire length without increasing complexity

- Example shape: square

- \( T_{\text{mem}} = T_{\text{logic}} + T_{\text{wire}} \)
  - \( T_{\text{logic}} \) includes \( T_{\text{cell}}, T_{\text{addr}} \)
  - \( T_{\text{logic}} \) includes \( \log(N) \) term
  - \( T_{\text{wire}} = \) signal propagation times, solely down wires

- For \( N \) memory locations, \( T_{\text{wire}} = ??? \)

- \( T_{\text{wire}} = C \times \sqrt{N} \)
Example: Memory Performance

Memory access includes these components:

- $T_{\text{mem}} = T_{\text{addr}} + T_{\text{wire}} + T_{\text{cell}}$
- $T_{\text{mem}} = C_1 \times \log(N) + C_2 \times \sqrt{N} + C_3$
- As $N$ grows very large, square-root term clearly dominates

Important result:

- Due to physical constraints, larger memories are slower than smaller memories
Many different storage technologies in modern computing systems!
  - Different strengths and weaknesses

Volatile memory loses its state when powered off
  - Static RAM (SRAM)
  - Dynamic RAM (DRAM)

Nonvolatile memory retains its state when powered off
  - Read-Only Memory (ROM)
  - Electrically-Erasable, Programmable Read-Only Memory (EEPROM) (a.k.a. “flash” memory)
  - Magnetic disk storage

Computers employ all of these in various ways
**STATIC RAM**

- **RAM = Random-Access Memory**
  - Any data value can be retrieved in constant time, regardless of its address, and regardless of previous accesses
  - (compare to magnetic tape storage, which is a sequential access medium)

- **Static RAM uses bistable memory cells**
  - State is stable as either 0 or 1
  - State doesn’t change, as long as power is applied

- **Idea behind SRAM cell:**
  - D = data, C = clock
  - Outputs of some gates are fed back to inputs, to produce a circuit that retains its state
Static RAM (2)

- Actual SRAM cells are much more efficient
  - (they don’t use D flip-flops...)
  - Each cell requires as few as 6 transistors
  - Same idea: some outputs of circuit fed back to inputs

- Layout is similar to before:
  - Individual cells actively maintain their own state
  - Entire address sent to decoder
  - Decoder activates specified row of memory cells
  - Depending on read/write signal, cells either read or write to $D_i$
**NOTE ON MEMORY TERMINOLOGY**

- **Horizontal wires are called word-lines**
  - Each one activates an entire word of data in the memory

- **Vertical wires are called bit-lines**
  - Each one corresponds to a specific bit of the data input/output
SRAM AND DRAM

- SRAM memory cells require 6 transistors apiece
  - Each cell takes up a certain amount of space...
  - Can only fit so much memory on a chip!

- Another circuit to store bits, *much* more densely:
  - Use a simple capacitor!
  - States are “charged” (1) or “discharged” (0)
  - Also need one transistor per cell, for addressing

- Problem with using a capacitor to store data:
  - The charge only lasts for so long! (10ms-100ms)
  - Need to periodically refresh each capacitor so that it won’t lose its data
  - Refresh operation is simple: read value, then write it back

- This is called Dynamic RAM (DRAM)
  - Memory cells are unstable and must be refreshed
**Dynamic RAM**

- DRAM memory cells consist of a single capacitor
  - Also need a transistor to control access to the capacitor
  - When transistor input is 1, capacitor is connected to the data line

- DRAM memories receive addresses in two parts
  - For large memories, dramatically reduces pin counts!
  - Memory is (typically) square

- Access procedure:
  - Specify a row address, then signal the DRAM
  - Specify a column address, then signal the DRAM
    - Read/write operation occurs at this point
**Dynamic RAM Example**

- Example: 16x1bit DRAM
  - 16 is number of cells \( d \)
  - 1 is bit-width \( w \) of each cell
    - CS:APP calls them “supercells” when \( w > 1 \)

- 16 cells are broken into 4 rows and 4 columns
  - 4 address bits total
  - 2 bits for row address
  - 2 bits for column address

- As before:
  - Horizontal word-lines
  - Vertical bit-lines
DRAM EXAMPLE (2)

- Address is specified in two steps
  - Step 1:
    - Specify row address on A0..A1
    - Activate RAS
      - “Row Address Strobe”
  - Row address is latched into the memory chip
  - Entire row is activated!
    - Capacitor values are turned into bits by amplifiers
    - Entire row is latched into the row-buffer
    - All capacitors in row are also refreshed by this step
DRAM Example (3)

- Address is specified in two steps
- Step 2:
  - Specify column address on A0..A1
  - Activate CAS
    - “Column Address Strobe”
- Column address latched into the memory chip
- Column address used to read or write a specific bit in the memory
Our example: 16x1bit DRAM
- 16 is total number of cells $d$
- 1 is bit-width $w$ of each cell
  - CS:APP calls them “supercells” when $w > 1$

DRAM circuits usually have $w = 8$ or 16
- 8 or 16 capacitors + transistors per cell
  - (hence named “supercell”)
- Column address selects entire word value
DRRAM Controller

- Processors usually can’t interface directly with DRAM
- Given a memory address:
  - Break into row and column addresses
  - Feed row to DRAM, then activate RAS
  - Feed column to DRAM, then activate CAS
- Also, something needs to periodically refresh the DRAM contents!
A DRAM controller handles interaction between CPU and memory
- Performs steps necessary to read or write a value
- Also traverses DRAM periodically, to refresh it
  - Can refresh an entire row simply by reading it
    - Bit sense/amplify logic refreshes capacitor charge when reading the value
  - Send address of row to refresh; activate RAS
  - No need to send CAS when refreshing a row’s data
Some advanced processors include a DRAM controller on-chip

Problem:
- Many variations on the basic DRAM theme!
- If new DRAM technology released, processor must be updated & re-released

Thus, memory controllers are usually provided by motherboard chipset, not by the processor itself
**Variation: FPM DRAM**

- Many variations on DRAM theme!
  - Optimizations for common cases
- When RAS is activated, an entire row is loaded into DRAM row-buffer
  - ...but, then only one value is read, and rest is discarded!
- Fast Page Mode DRAM
  - As long as RAS is held active, row data is kept in buffer
  - Can read multiple values from the row by specifying different column addresses, then setting CAS to get each value
- Memory accesses are often adjacent...
  - Reduces need to perform RAS portion of access cycle
COMPARISON: SRAM AND DRAM

- SRAM is much faster than DRAM, for several reasons
  - SRAM: 1-10ns (e.g. 4ns access time)
  - DRAM: 30-100ns (e.g. 60ns access time)

- SRAM memory cells have gates that actively drive data lines high or low
  - DRAM memory cells connect a capacitor to the data line, which takes longer to drive it high or low

- SRAM receives entire memory address at once
  - DRAM usually has to receive and store address in two parts, which requires multiple clocks

- SRAM is stable, and doesn’t need to be refreshed
  - DRAM refresh is relatively infrequent, but definitely still impacts memory access performance
COMPARISON: SRAM AND DRAM (2)

DRAM is much denser than SRAM
- 1 transistor + 1 capacitor per memory cell, vs. 6 transistors per cell for SRAM

Because DRAM uses half the address lines of SRAM, can scale up DRAM sizes more easily

Clearly want to use DRAM and SRAM for different purposes in the computer, e.g.
- Use SRAM technology for register file and other fast memories
- Use DRAM for providing very large memories for programs to use
- ...but how to make it fast?
**MAGNETIC DISK STORAGE**

- Disks are used to store large amounts of data
- Disks contain one or more platters
  - Platter is covered with magnetic material
  - Platter’s surface divided into concentric rings called **tracks**
  - Each track is divided into a number of **sectors**
Magnetic Disk Storage (2)

- Platters are spun at a constant rate
  - e.g. 5400 RPM, up to 15000 RPM for fastest drives
  - 7200 RPM is typical for desktop HDDs
- Disk has a read/write head on end of an actuator arm
- To read and write data:
  - Position actuator arm to line up with a given track
  - Sector will eventually move under the read/write head
Magnetic Disk Storage (3)

- Can increase disk storage by adding multiple platters
- Actuator arm carries multiple read/write heads
- A cylinder $k$ is the set of tracks $k$ on each surface of each platter
- At a specific head position, can read all sectors of all tracks in the cylinder
Magnetic Disk Storage (4)

- With this simplistic layout, sectors are larger when further away from the center of the disk
  - Wastes a lot of space towards edge of disk!
- Modern disks divided into zones
- Different zones have different numbers of tracks
  - Zones further from center of disk have more sectors
- Increases storage density, which increases total available storage
**Magnetic Disk Storage (5)**

- Two major factors for disk access performance:
  - **Seek time:**
    - How long does it take to move the arm to a specific track?
    - (Clearly depends on previous location of arm, too...)
  - **Rotational latency:**
    - How long does it take for desired sector to move under the read/write head?
    - Could be up to 1 revolution
Magnetic Disk Storage (3)

- Seek time:
  - Worst case, can be up to 20ms
  - Average case usually 6-9ms

- Rotational latency (7200 RPM):
  - Worst case (1 revolution), is 8ms
  - Average case (1/2 revolution) is 4ms

- With these assumptions, accessing a given sector could take 10-30ms
**Storage Technologies**

- Modern computers use SRAM, DRAM, and magnetic disks
- Comparison:

<table>
<thead>
<tr>
<th>Technology</th>
<th>Access Time</th>
<th>Cost (2010 numbers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM</td>
<td>1-10ns</td>
<td>$60/MB</td>
</tr>
<tr>
<td>DRAM</td>
<td>30-100ns</td>
<td>$40/GB</td>
</tr>
<tr>
<td>Hard disk</td>
<td>8-30ms</td>
<td>$0.30/GB</td>
</tr>
</tbody>
</table>

- Compare these to processor performance!
  - 3GHz processor: 1 clock = 0.3ns

- If 1 clock were 1 second:
  - SRAM access would take 3-30 seconds
  - DRAM access would take 1½ to 5 minutes
  - Hard disk access would take 1-3 years!
THE PROCESSOR-MEMORY GAP

- Processor is significantly faster than memory...
- ...but it’s actually worse than that:

![Graph showing the Processor-Memory Performance Gap](Image)

*Source: Patterson 1997*
THE PROCESSOR-MEMORY GAP (2)

- Processor speed has been growing by ~60%/year
- DRAM speed has been growing by ~8%/year

- The gap has been growing by >50%/year!

Source: Patterson 1997

FIGURE 1. Processor-Memory Performance Gap.[Henz96].
**Computer Memory System: Goal**

- Would like to have memory the size of DRAM, but performance of SRAM or faster
  - Ideally, even performance of registers
- We will *never* get it with this design:

![Diagram](image)

- Idea: Introduce a **cache** between the processor and main memory
  - Use a faster storage technology than DRAM
  - Use the cache as a staging area for information in main memory
**Computer Memory System: Caching**

- **Idea:** Introduce a cache between the processor and main memory

  - When CPU reads a value from main memory:
    - Read an entire block of data from main memory
    - As long as subsequent accesses are within this block, just use the cache!
    - If an access is outside the cached data, need to retrieve and cache a new block of data from memory
**Next Time**

- What kinds of program behaviors will work best (or worst) with our caches?
- How do we design these caches? What are the design trade-offs?