Where We Are in This Course Right Now

- So far:
  - We know how to design a processor that can fetch, decode, and execute the instructions in an ISA
  - We understand how to design caches and memory
- Now:
  - We learn about the lowest level of storage (disks)
  - We learn about input/output in general
- Next:
  - Multicore processors
  - Evaluating and improving performance

This Unit: I/O

- I/O system structure
- Devices, controllers, and buses
- Device characteristics
  - Disks
  - Bus characteristics
  - I/O control
    - Polling and interrupts
    - DMA

Readings

- Patterson and Hennessy
  - Chapter 6
Computers Interact with Outside World

- **Input/output (I/O)**
  - Otherwise, how will we ever tell a computer what to do...
  - ...or exploit the results of its work?
- Computers without I/O are not useful
- ICQ: What kinds of I/O do computers have?

One Instance of I/O

- Have briefly seen one instance of I/O
  - **Disk**: bottom of memory hierarchy
  - Holds whatever can't fit in memory
  - ICQ: What else do disks hold?

A More General/Realistic I/O System

- A computer system
  - CPU, including cache(s)
  - Memory (DRAM)
  - I/O peripherals: disks, input devices, displays, network cards, ...
    - With built-in or separate I/O (or DMA) controllers
    - All connected by a **system bus**

I/O: Control + Data Transfer

- I/O devices have two ports
  - **Control**: commands and status reports
    - How we tell I/O what to do
    - How I/O tells us about itself
    - Control is the tricky part (especially status reports)
  - **Data**
    - Labor-intensive part
    - "Interesting" I/O devices do data transfers (to/from memory)
      - Display: video memory → monitor
      - Disk: memory ↔ disk
      - Network interface: memory ↔ network
Operating System (OS) Plays a Big Role

- I/O interface is typically under OS control
  - User applications access I/O devices indirectly (e.g., SYSCALL)
  - Why?
  - Device drivers are "programs" that OS uses to manage devices

- Virtualization: same argument as for memory
  - Physical devices shared among multiple programs
  - Direct access could lead to conflicts – example?

- Synchronization
  - Most have asynchronous interfaces, require unbounded waiting
  - OS handles asynchrony internally, presents synchronous interface

- Standardization
  - Devices of a certain type (disks) can/will have different interfaces
  - OS handles differences (via drivers), presents uniform interface

I/O Device Characteristics

- Primary characteristic
  - Data rate (aka bandwidth)

- Contributing factors
  - Partner: humans have slower output data rates than machines
  - Input or output or both (input/output)

<table>
<thead>
<tr>
<th>Device</th>
<th>Partner</th>
<th>I/O</th>
<th>Data Rate (KB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>Human</td>
<td>Input</td>
<td>0.01</td>
</tr>
<tr>
<td>Mouse</td>
<td>Human</td>
<td>Input</td>
<td>0.02</td>
</tr>
<tr>
<td>Speaker</td>
<td>Human</td>
<td>Output</td>
<td>0.60</td>
</tr>
<tr>
<td>Printer</td>
<td>Human</td>
<td>Output</td>
<td>200</td>
</tr>
<tr>
<td>Display</td>
<td>Human</td>
<td>Output</td>
<td>240,000</td>
</tr>
<tr>
<td>Modem</td>
<td>Machine</td>
<td>I/O</td>
<td>7</td>
</tr>
<tr>
<td>Ethernet card</td>
<td>Machine</td>
<td>I/O</td>
<td>~1,000,000</td>
</tr>
<tr>
<td>Disk</td>
<td>Machine</td>
<td>I/O</td>
<td>~10,000</td>
</tr>
</tbody>
</table>

I/O Device Bandwidth: Some Examples

- Keyboard
  - 1 B/key * 10 keys/s = 10 B/s

- Mouse
  - 2 B/transfer * 10 transfers/s = 20 B/s

- Display
  - 4 B/pixel * 1M pixel/display * 60 displays/s = 240 MB/s

I/O Device: Disk

- **Disk**: like stack of record players
  - **Collection of platters**
    - Each with read/write head
  - Platters divided into concentric tracks
    - Head seeks (forward/backward) to track
    - All heads move in unison
  - Each track divided into sectors
    - ZBR (zone bit recording)
      - More sectors on outer tracks
      - Sectors rotate under head
  - **Controller**
    - Seeks heads, waits for sectors
    - Turns heads on/off
    - May have its own cache (made w/DRAM)
Disk Parameters

<table>
<thead>
<tr>
<th>Diameter</th>
<th>Seagate ST31200</th>
<th>Seagate Savvio</th>
<th>Toshiba MK1003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>3.5&quot;</td>
<td>2.5&quot;</td>
<td>1.8&quot;</td>
</tr>
<tr>
<td>RPM</td>
<td>200 GB</td>
<td>73 GB</td>
<td>10 GB</td>
</tr>
<tr>
<td>Cache</td>
<td>7200 RPM</td>
<td>10000 RPM</td>
<td>4200 RPM</td>
</tr>
<tr>
<td>Cache</td>
<td>8 MB</td>
<td>7</td>
<td>512 KB</td>
</tr>
<tr>
<td>Disks/Heads</td>
<td>2/4</td>
<td>2/4</td>
<td>1/2</td>
</tr>
<tr>
<td>Average Seek</td>
<td>8 ms</td>
<td>4.5 ms</td>
<td>7 ms</td>
</tr>
<tr>
<td>Peak Data Rate</td>
<td>150 MB/s</td>
<td>200 MB/s</td>
<td>200 MB/s</td>
</tr>
<tr>
<td>Sustained Data Rate</td>
<td>58 MB/s</td>
<td>94 MB/s</td>
<td>16 MB/s</td>
</tr>
<tr>
<td>Interface</td>
<td>ATA</td>
<td>SCSI</td>
<td>ATA</td>
</tr>
<tr>
<td>Use</td>
<td>Desktop</td>
<td>Laptop</td>
<td>iPod</td>
</tr>
</tbody>
</table>

- Slightly newer disk from Toshiba
  - 0.85", 4 GB drives, used in iPod-mini

Disk Read/Write Latency

- Disk read/write latency has four components
  - Seek delay \( t_{\text{seek}} \): head seeks to right track
  - Rotational delay \( t_{\text{rotation}} \): right sector rotates under head
    - On average: time to go halfway around disk
  - Transfer time \( t_{\text{transfer}} \): data actually being transferred
  - Controller delay \( t_{\text{controller}} \): controller overhead (on either side)

- Example: time to read a 4KB page assuming...
  - 128 sectors/track, 512 b/sector, 6000 RPM, 10 ms \( t_{\text{seek}} \), 1 ms \( t_{\text{controller}} \)
  - 6000 RPM \( \rightarrow 100 \text{ R/s} \rightarrow 10 \text{ ms/R} \rightarrow t_{\text{rotation}} = 10 \text{ ms} / 2 = 5 \text{ ms} \)
  - 4 KB page \( \rightarrow 8 \text{ sectors} \rightarrow t_{\text{transfer}} = 10 \text{ ms} * 8/128 = 0.6 \text{ ms} \)
  - \( t_{\text{disk}} = t_{\text{seek}} + t_{\text{rotation}} + t_{\text{transfer}} + t_{\text{controller}} = 10 + 5 + 0.6 + 1 = 16.6 \text{ ms} \)

Disk Bandwidth

- Disk is bandwidth-inefficient for page-sized transfers
  - Actual data transfer \( t_{\text{transfer}} \) a small part of disk access (and cycle)

- Increase bandwidth: **stripe data across multiple disks**
  - Stripping strategy depends on disk usage model
  - "File System" or "web server": many small files
  - Map entire files to disks
  - "Supercomputer" or "database": several large files
  - Stripe single file across multiple disks

- Both bandwidth and individual transaction latency important

Error Correction: RAID

- **Error correction**: more important for disk than for memory
  - Mechanical disk failures (entire disk lost) is common failure mode
  - Entire file system can be lost if files striped across multiple disks

- **RAID (redundant array of inexpensive disks)**
  - Similar to DRAM error correction, but...
  - Major difference: which disk failed is known
  - Even parity can be used to recover from single failures
  - Parity disk can be used to reconstruct data faulty disk
  - RAID design balances bandwidth and fault-tolerance
  - Many flavors of RAID exist
    - RAID0: extra disks (cost) vs. performance vs. reliability
    - Deeper discussion of RAID in ECE 252 and ECE 254
  - RAID doesn't solve all problems → can you think of any examples?
### The System Bus

- **System bus**: connects system components together
  - Important: insufficient bandwidth can bottleneck entire system
  - Performance factors
    - Physical length
    - Number and type of connected devices (taps)

### Three Buses

- **Processor-memory bus**
  - Connects CPU and memory, no direct I/O interface
  - Short, few taps → fast, high-bandwidth
  - System specific

- **I/O bus**
  - Connects I/O devices, no direct P-M interface
  - Longer, more taps → slower, lower-bandwidth
  - Industry standard

- **Backplane bus**
  - CPU, memory, I/O connected to same bus
  - Industry standard, cheap (no adapters needed)
  - Processor-memory performance compromised

### Bus Design

- **Goals**
  - **High Performance**: low latency and high bandwidth
  - **Standardization**: flexibility in dealing with many devices
  - **Low Cost**: Processor-memory bus emphasizes performance, then cost
    - I/O & backplane emphasize standardization, then performance

- **Design issues**
  1. **Width/multiplexing**: are wires shared or separate?
  2. **Clocking**: is bus clocked or not?
  3. **Switching**: how/when is bus control acquired and released?
  4. **Arbitration**: how do we decide who gets the bus next?

### (1) Bus Width and Multiplexing

- **Wider**
  + More bandwidth
    - More expensive and more susceptible to skew

- **Multiplexed**: address and data share same lines
  + Cheaper
  - Less bandwidth

- **Burst transfers (bus parking)**
  - Multiple sequential data transactions for single address
    + Increase bandwidth at relatively little cost
(2) Bus Clocking

- **Synchronous**: clocked
  + Fast
  - Bus must be short to minimize clock skew

- **Asynchronous**: un-clocked
  + Can be longer: no clock skew, deals with devices of different speeds
  - Slower: requires "hand-shaking" protocol
    - For example, asynchronous read
      - Multiplexed data/address lines, 3 control lines
        1. Processor drives address onto bus, asserts Request line
        2. Memory asserts Ack line, processor stops driving
        3. Memory drives data on bus, asserts DataReady line
        4. Processor asserts Ack line, memory stops driving
- P-M buses are synchronous
- I/O and backplane buses asynchronous or slow-clock synchronous

(3) Bus Switching

- **Atomic**: bus "busy" between request and reply
  + Simple
  - Low utilization

- **Split-transaction**: requests/replies can be interleaved
  + Higher utilization → higher throughput
  - Complex, requires sending IDs to match replies to request

(4) Bus Arbitration

- **Bus master**: component that can initiate a bus request
  - Bus typically has several masters, including processor
  - I/O devices can also be masters (Why? See in a bit)

- **Arbitration**: choosing a master among multiple requests
  - Try to implement priority and fairness (no device "starves")
  - Daisy-chain: devices connect to bus in priority order
    - High-priority devices intercept/deny requests by low-priority ones
    - Simple, but slow and can’t ensure fairness
  - Centralized: special arbiter chip collects requests, decides
    - Ensures fairness, but arbiter chip may itself become bottleneck
  - Distributed: everyone sees all requests simultaneously
    - Back off and retry if not the highest priority request
    - No bottlenecks and fair, but needs a lot of control lines

Standard Bus Examples

<table>
<thead>
<tr>
<th>Type</th>
<th>Width</th>
<th>Multiplexed?</th>
<th>Clocking (MHz)</th>
<th>Data rate (MB/s)</th>
<th>Arbitration</th>
<th>Maximum masters</th>
<th>Maximum length</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCl/PCIe</td>
<td>32–64</td>
<td>Yes</td>
<td>33 (66)</td>
<td>133 (266)</td>
<td>Distributed</td>
<td>1024</td>
<td>0.5 m</td>
</tr>
<tr>
<td>SCSI</td>
<td>32–64</td>
<td>Yes</td>
<td>5 (10)</td>
<td>10 (20)</td>
<td>Distributed</td>
<td>256</td>
<td>2.5 m</td>
</tr>
<tr>
<td>I/O</td>
<td>8–32</td>
<td>Yes</td>
<td>33 (66)</td>
<td>133 (266)</td>
<td>Daisy-chain</td>
<td>127</td>
<td>–</td>
</tr>
<tr>
<td>USB</td>
<td>1</td>
<td>Yes</td>
<td>Asynchronous</td>
<td>0.2, 1.5, 60 MB/s</td>
<td>Distributed</td>
<td>256</td>
<td>–</td>
</tr>
</tbody>
</table>

**USB (universal serial bus)**

- Popular for low/moderate bandwidth external peripherals
  + Packetized interface (like TCP), extremely flexible
  + Also supplies power to the peripheral