I/O

Impact of I/O performance depends on applications being run:

- Web server: need fast disk access, high network bandwidth
- PC: rapid response to keyboard input, mouse
- Databases (transaction processing): fast disk access
- Supercomputing: high disk throughput
- Video games: high graphics performance, rapid processing of button inputs.

Can have a greater impact than processor speed!

- Expensive machines: main difference is I/O performance (fancy controllers)

- Assume you know disks, etc. from 202

BUSES (section 8.4 in text)

Buses - the set of wires connecting CPU, memory, and I/O devices.
Simplesst Picture:

```
  CPU
    ▼
     | ▼
     |  ▼
  I/O device
  I/O device
  I/O device
```

"Backplane"

Data typically flows between I/O devices and memory (with varying degrees of CPU involvement):

- Inexpensive systems: CPU has to perform the transfers, wasting CPU time.
- More expensive systems: Memory & I/O devices interact w/o using CPU cycles.

Bus generally has DATA & CONTROL lines:

DATA lines: Addresses & data are sent over these.
Control lines: Signal requests & acknowledgments.
Example (page 657 in book)

Output operation (transfer of data from memory to I/O device)

1) Memory sends request is signalled (either by CPU or I/O device) on control lines. The address of data to be read is put on the data lines.

2) Memory accesses data.

3) Memory puts data on data lines and signals the availability of data on the control lines. I/O device takes data off the data lines.
Problem: I/O devices often transfer data much more slowly than the speed of the processor-memory bus.

Solution: Don't attach devices directly to processor-memory bus.

- Faster "tops out" on I/O bus
- Less contention for I/O bus

Diagram:

```
  CPU    | Processor/Mem bus |
        | Bus Adapter |
        | I/O bus |
        | I/O    |
        |        |
    I/O    | Bus Adapter |
        | I/O bus |
        | I/O    |
        |        |
    I/O    | Bus Adapter |
        | I/O    |
        |        |
    I/O    | Bus Adapter |
        | I/O    |
        |        |
    I/O    | Bus Adapter |
        | I/O    |
        |        |
```
''Peripheral component interconnect''

Modern PCI bus systems:

```
<table>
<thead>
<tr>
<th>CPU</th>
<th>L2 cache</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RAM controller</td>
</tr>
<tr>
<td></td>
<td>MEMORY</td>
</tr>
<tr>
<td></td>
<td>Bus bridge</td>
</tr>
<tr>
<td></td>
<td>PCI bus</td>
</tr>
<tr>
<td></td>
<td>PCI I/O device</td>
</tr>
<tr>
<td></td>
<td>PCI I/O device</td>
</tr>
<tr>
<td></td>
<td>AGP chipset</td>
</tr>
<tr>
<td></td>
<td>Graphics card</td>
</tr>
<tr>
<td></td>
<td>Frontside (system) bus</td>
</tr>
<tr>
<td></td>
<td>Backside bus</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>ISA device</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISA device</td>
</tr>
</tbody>
</table>

Synchronous and Asynchronous Buses

- Synchronous: uses a clock signal on a control line (see by processor, memory, I/O device)
  - I/O is synchronized by the clock
    - simple and easy to implement
    - e.g., address read request at cycle 1, memory puts data on bus at cycle 6.

http://computer.huisstaff.work.com/pci2.htm
Problem: 1) Every I/O device must operate at the same speed.
2) Bus cannot be long due to "clock skew" because of the delay in sending signal down the wire. Clock has to stay up long enough for both devices to synchronize.

Asynchronous Bus - not clocked
uses "handshaking protocol" between devices on control lines.

Example in book (pg. 661)
- 3 control lines used in protocol: Read, Ack, data ready
- Data sent on data lines.
- Device requests data from memory, memory provides data.
A. I/O device puts address on data line and raises ReadReq to signal a read request.

B. Memory raises Ack to signal that it saw read request.

C. Memory has fetched requested data, so it puts data on data line and raises DataReady to signal that the data is available.

D. I/O device reads data and raises Ack to signal that it has seen the data.

E. Bus sees Ack line and drops DataReady & data line.

F. I/O device sees DataReady drop and drops Ack.
To increase bus performance, transfer many words of data in a "block transfer", without requiring a new request in between each word.

Also:
- increase bus width to transfer several words simultaneously
- send address and data (for writes) simultaneously or separate data lines

Obtaining access to the bus
- can't allow every device to assert control line (e.g. read/write) any time it wants.
- another device might be in the middle of a transaction.

Need a bus master (possibly multiple masters)
- controls access to the bus
- initiates and controls all bus requests
- the CPU is always a master, since it initiates memory operations
- memory is usually a slave.
Simplest system: Single bus master (CPU)

1. Device sends request for bus on bus request line
2. Processor generates signal on control line (e.g., read Req)
3. Processor notifies I/O device bus that it can send address on data lines
4. Device places address on data line.

More complicated: Multiple masters
e.g., CPU and a DMA controller (direct memory access)
- doesn't require CPU intervention every time a transfer between memory and an I/O device is required.
- e.g., when a disk block has been read and is sent to memory.
For multiple master busses, an arbitration scheme is needed.

- e.g. daisy-chain centralized arbitration, priority, collision detection (see book)

---

Coordinating I/O Devices w/ the CPU

- The CPU can communicate w/ an I/O via:
  - special I/O instructions, which write data to a register on the I/O device
  or
  - memory mapped I/O

- some of the physical address space of the processor is actually assigned to the I/O device.

Reading and writing from/to this area of memory actually causes communication with the I/O device.

- the physical memory controller ignores write/reads to those addresses, but the I/O device controller sees them.
Retrieving data from an I/O device generally happens in one of two ways:

- **Polling**: the CPU occasionally polls the device it is expecting data from to see if the data is ready.
  - Waste of CPU cycles but easy to implement (often used for mouse).

- **Interrupt-driven I/O**: when the device has the data available, it causes an interrupt to occur on the CPU. At that point, the O.S. takes over and initiates the transfer of data (if necessary).

The actual data transfer can be accomplished:

- by the CPU reading the data from the I/O device and writing to memory.

- via DMA (and the DMA controller)
  - processor indicates I/O device operation and address in memory to the DMA controller.
- DMA controller arbitrates for the bus
  requests the data from the I/O device
  and writes to memory
  - perhaps repeating many times

- DMA controller might buffer data
to deal with transfer delays to
memory or free up the bus for
CPU use

- When transfer is complete, DMA
  interrupts processor.