**UCB CS61C: Machine Structures**

**Lecture 34 – Input / Output**

**2008-04-23**

Hi to Gary McCay from Tampa Florida!

**Arduino Allows Way Cool I/O Projects!**

*Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.* Available for Mac / Windows / Linux. You can buy one or build your own.

www.arduino.cc

---

**Recall: 5 Components of Any Computer**

Earlier Lectures

- Processor (active)
- Control ("brain")
- Datapath ("brawn")

Current Lectures

- Memory (passive)
- Devices
  - Input
  - Output
- Keyboard, Mouse
- Disk, Network
- Display, Printer

---

**I/O Device Examples and Speeds**

- **I/O Speed:** bytes transferred per second (from mouse to Gigabit LAN: 7 orders of mag!)

<table>
<thead>
<tr>
<th>Device</th>
<th>Behavior</th>
<th>Partner</th>
<th>Data Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>Input</td>
<td>Human</td>
<td>0.01</td>
</tr>
<tr>
<td>Mouse</td>
<td>Input</td>
<td>Human</td>
<td>0.02</td>
</tr>
<tr>
<td>Voice output</td>
<td>Output</td>
<td>Human</td>
<td>5.00</td>
</tr>
<tr>
<td>Floppy disk</td>
<td>Storage</td>
<td>Machine</td>
<td>50.00</td>
</tr>
<tr>
<td>Laser Printer</td>
<td>Output</td>
<td>Human</td>
<td>100.00</td>
</tr>
<tr>
<td>Magnetic Disk</td>
<td>Storage</td>
<td>Machine</td>
<td>10,000.00</td>
</tr>
<tr>
<td>Wireless Network</td>
<td>I or O</td>
<td>Machine</td>
<td>10,000.00</td>
</tr>
<tr>
<td>Graphics Display</td>
<td>Output</td>
<td>Human</td>
<td>30,000.00</td>
</tr>
<tr>
<td>Wired LAN Network</td>
<td>I or O</td>
<td>Machine</td>
<td>125,000.00</td>
</tr>
</tbody>
</table>

When discussing transfer rates, use 10^m

---

**Review**

- Manage memory to disk? Treat as cache
  - Included protection as bonus, now critical
  - Use Page Table of mappings for each process vs. tag/data in cache
  - TLB is cache of Virtual ⇒ Physical addr trans
- Virtual Memory allows protected sharing of memory between processes
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well

---

**Motivation for Input/Output**

- I/O is how humans interact with computers
- I/O gives computers long-term memory.
- I/O lets computers do amazing things:
  - Read pressure of synthetic hand and control synthetic arm and hand of fireman
  - Control propellers, fins, communicate in BOB (Breathable Observable Bubble)
  - Computer without I/O like a car w/no wheels; great technology, but gets you nowhere

---

**What do we need to make I/O work?**

- A way to connect many types of devices
- A way to control these devices, respond to them, and transfer data
- A way to present them to user programs so they are useful

---

**Operating System**

- Files
- APIs
- Operating System

- Proc
- Mem
- PCI Bus
- SCSI Bus
- Ctrl reg
- Data reg
Instruction Set Architecture for I/O

- What must the processor do for I/O?
  - Input: reads a sequence of bytes
  - Output: writes a sequence of bytes
- Some processors have special input and output instructions
  - Alternative model (used by MIPS):
    - Use loads for input, stores for output
    - Called Memory Mapped Input/Output
    - A portion of the address space dedicated to communication paths to Input or Output devices (no memory there)

Processor-I/O Speed Mismatch

- 1GHz microprocessor can execute 1 billion load or store instructions per second, or 4,000,000 KB/s data rate
  - I/O devices data rates range from 0.01 KB/s to 125,000 KB/s
- Input: device may not be ready to send data as fast as the processor loads it
  - Also, might be waiting for human to act
- Output: device not be ready to accept data as fast as processor stores it
- What to do?

SPIM I/O Simulation

- SPIM simulates 1 I/O device: memory-mapped terminal (keyboard + display)
  - Read from keyboard (receiver); 2 device regs
  - Writes to terminal (transmitter); 2 device regs

Memory Mapped I/O

- Certain addresses are not regular memory
- Instead, they correspond to registers in I/O devices

Processor Checks Status before Acting

- Path to device generally has 2 registers:
  - Control Register, says it’s OK to read/write I/O ready (think of a flagman on a road)
  - Data Register, contains data
- Processor reads from Control Register in loop, waiting for device to set Ready bit in Control reg (0 ⇒ 1) to say its OK
- Processor then loads from (input) or writes to (output) data register
  - Load from or Store into Data Register resets Ready bit (1 ⇒ 0) of Control Register

SPIM I/O

- Control register rightmost bit (0): Ready
  - Receiver: Ready == 1 means character in Data Register not yet been read;
    1 ⇒ 0 when data is read from Data Reg
  - Transmitter: Ready == 1 means transmitter is ready to accept a new character;
    0 ⇒ Transmitter still busy writing last char
    - I.E. bit discussed later
- Data register rightmost byte has data
  - Receiver: Last char from keyboard; rest = 0
  - Transmitter: when write rightmost byte, writes char to display
**I/O Example**

- **Input:** Read from keyboard into $v0
  
  ```
  lui $t0, 0xffff
  Waitloop: lw $t1, 0($t0) #control
  andi $t1,$t1,0x1
  beq $t1,$zero, Waitloop
  lw $v0, 4($t0) #data
  ```

- **Output:** Write to display from $a0
  
  ```
  lui $t0, 0xffff
  Waitloop: lw $t1, 8($t0) #control
  andi $t1,$t1,0x1
  beq $t1,$zero, Waitloop
  sw $a0, 12($t0) #data
  ```

- Processor waiting for I/O called “Polling”
- “Ready” bit is from processor’s point of view!

**Administrivia**

- Only 8 lectures after this one! :-(
  - About every third by an outstanding TA
  - Project 3 will be graded face-to-face, check web page for scheduling
  - Project 4 (Cache simulator) out already
  - You may work in pairs for this project!
  - Do the performance competition!
    - You may work in pairs for this project!
  - Final Exam: M 2008-05-19 @ 5-8pm loc TBA

---

**Upcoming Calendar**

<table>
<thead>
<tr>
<th>Week</th>
<th>Mon</th>
<th>Wed</th>
<th>Thu Lab</th>
<th>Fri</th>
</tr>
</thead>
<tbody>
<tr>
<td>#16</td>
<td>I/O Basics P4 out</td>
<td>VM</td>
<td>I/O Networks (Blum)</td>
<td></td>
</tr>
<tr>
<td>#15</td>
<td>I/O Disks</td>
<td>Performance P4 due</td>
<td>I/O Polling</td>
<td>Writing really fast code (Casey)</td>
</tr>
<tr>
<td>#16</td>
<td>Parallelism in Processor Design</td>
<td>MITI-machine Parallelism (Matt)</td>
<td>Parallel</td>
<td>I/O interrupt handling</td>
</tr>
<tr>
<td>#17</td>
<td>LAST CLASS</td>
<td>Summary, Review, EHNN Evils</td>
<td>FINAL EXAM</td>
<td>Final exam location TBA</td>
</tr>
</tbody>
</table>

---

**What is the alternative to polling?**

- Wasteful to have processor spend most of its time “spin-waiting” for I/O to be ready
- Would like an unplanned procedure call that would be invoked only when I/O device is ready
- Solution: use exception mechanism to help I/O. Interrupt program when I/O ready, return when done with data transfer

---

**I/O Interrupt**

- An I/O interrupt is like overflow exceptions except:
  - An I/O interrupt is “asynchronous”
  - More information needs to be conveyed
- An I/O interrupt is asynchronous with respect to instruction execution:
  - I/O interrupt is not associated with any instruction, but it can happen in the middle of any given instruction
  - I/O interrupt does not prevent any instruction from completion

---

**Interrupt-Driven Data Transfer**
**SPIM I/O Simulation: Interrupt Driven I/O**

- I.E. stands for Interrupt Enable
- Set Interrupt Enable bit to 1 have interrupt occur whenever Ready bit is set

<table>
<thead>
<tr>
<th>Receiver Control</th>
<th>Unused (00...00)</th>
<th>Received Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Receiver Data</td>
<td>Unused (00...00)</td>
<td></td>
</tr>
<tr>
<td>0xFFFF0004</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transmitter Control</th>
<th>Unused (00...00)</th>
<th>Transmitted Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF0008</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transmitter Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xFFFF000C</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Peer Instruction**

A. A faster CPU will result in faster I/O.
B. Hardware designers handle mouse input with interrupts since it is better than polling in almost all cases.
C. Low-level I/O is actually quite simple, as it's really only reading and writing bytes.

**“And in conclusion…”**

- I/O gives computers their 5 senses
- I/O speed range is 100-million to one
- Processor speed means must synchronize with I/O devices before use
- Polling works, but expensive
  - processor repeatedly queries devices
- Interrupts works, more complex
  - devices causes an exception, causing OS to run and deal with the device
- I/O control leads to Operating Systems

**Bonus slides**

- These are extra slides that used to be included in lecture notes, but have been moved to this, the “bonus” area to serve as a supplement.
- The slides will appear in the order they would have in the normal presentation

**Definitions for Clarification**

- **Exception**: signal marking that something “out of the ordinary” has happened and needs to be handled
  - Interrupt: asynchronous exception
  - Trap: synchronous exception
- **Note**: Many systems folks say “interrupt” to mean what we mean when we say “exception”.

**Cost of Polling?**

- Assume for a processor with a 1GHz clock it takes 400 clock cycles for a polling operation (call polling routine, accessing the device, and returning). Determine % of processor time for polling
  - Mouse: polled 30 times/sec so as not to miss user movement
  - Floppy disk: transfers data in 2-Byte units and has a data rate of 30 KB/second. No data transfer can be missed.
  - Hard disk: transfers data in 16-Byte chunks and can transfer at 16 MB/second. Again, no transfer can be missed.
% Processor time to poll [p. 677 in book]

- **Mouse Polling (clocks/sec)**
  - \(30 \text{ polls/s} \times 400 \text{ clocks/poll} = 12K \text{ clocks/s}\)
- **% Processor for polling**
  - \(12^{109} \text{ clocks/s} / 1^{109} \text{ clocks/s} = 0.0012\%
  - Polling mouse little impact on processor
- **Frequency of Polling Floppy**
  - \(50 \text{ KB/s} / 2 \text{ B/poll} = 25K \text{ polls/s}\)
- **Floppy Polling, Clocks/sec**
  - \(25K \text{ polls/s} \times 400 \text{ clocks/poll} = 10M \text{ clocks/s}\)
- **% Processor for polling**
  - \(10^{106} \text{ clocks/s} / 1^{109} \text{ clocks/s} = 1\%
  - OK if not too many I/O devices

% Processor time to poll hard disk

- **Frequency of Polling Disk**
  - \(16 \text{ MB/s} / 16 \text{ B/poll} = 1M \text{ polls/s}\)
- **Disk Polling, Clocks/sec**
  - \(1M \text{ polls/s} \times 400 \text{ clocks/poll} = 400M \text{ clocks/s}\)
- **% Processor for polling**
  - \(400^{106} \text{ clocks/s} / 1^{109} \text{ clocks/s} = 40\%
  - Unacceptable

---

**Benefit of Interrupt-Driven I/O**

- Find the % of processor consumed if the hard disk is only active 5% of the time. Assuming 500 clock cycle overhead for each transfer, including interrupt:
  - **Disk Interrupts/s** = \(16 \text{ MB/s} / 16 \text{ B/interrupt}\) = 1M [interrupts/s]
  - **Disk Interrupts (clocks/s)**
    - \(1M \text{ interrupts/s} \times 500 \text{ clocks/interrupt}\) = 500,000,000 [clocks/s]
  - **% Processor for during transfer**
    - \(500^{109} \text{ clocks/s} / 1^{109} \text{ clocks/s} = 50\%
  - **Disk active 5%** \(\Rightarrow\) **5% * 50%** \(\Rightarrow\) **2.5% busy**