LC-3 Input and Output
Summer 2008

I/O: Connecting to Outside World

- So far, we’ve learned how to:
  - compute with values in registers
  - load data from memory to registers
  - store data from registers to memory
  - use the **TRAP** calls to deal with I/O

- How do the TRAP calls work?
I/O devices types

- I/O devices are characterized by
  - Behavior
  - Data rate
- Behavior: input, output, storage
  - Input: keyboard, motion detector, network interface
  - Output: monitor, printer, network interface
  - Storage: disk, CD-ROM
- Data rate: how fast can data be transferred?
  - Keyboard: 100 bytes/sec
  - Disk: 30 MB/s
  - Network: 1 Mb/s - 1 Gb/s

I/O Controller

- Control/Status Registers
  - CPU tells device what to do -- write to control register
  - CPU checks whether task is done -- read status register
- Data Registers
  - CPU transfers data to/from device
- Device electronics
  - Performs actual operation
    - pixels to screen, bits to/from disk, characters from keyboard
Programming Interface

- How are device registers identified?
  - Memory-mapped vs. I/O-mapped (special instructions)
- How is timing of transfer managed?
  - Asynchronous vs. synchronous
- Who controls transfer?
  - CPU (polling) vs. device (interrupts)

I/O-mapped I/O

- Specific opcode(s) for I/O (e.g. IN and OUT in x86)
- Two separate addressing spaces
Memory-Mapped I/O

- Assign a memory address to each device register
- Use the same memory data movement instructions (load/store) for control and data transfer
- The hardware will figure out that the instruction refers to a device and not to the memory

Transfer Timing

- I/O events generally happen much slower than CPU cycles
  - Synchronous
    - Data supplied at a fixed, predictable rate
    - CPU reads/writes every X cycles
  - Asynchronous
    - Data rate less predictable
    - CPU must synchronize with device, so that it doesn’t miss data or write too quickly
Transfer Control

- Who determines when the next data transfer occurs?
- Polling
  - CPU keeps checking status register until \(\text{new data arrives OR device ready}\) for next data
  - “Are we there yet? Are we there yet?”
- Interrupts
  - Device sends a special signal to CPU when \(\text{new data arrives OR device ready}\) for next data
  - CPU can be performing other tasks instead of polling device.
  - “Wake me up when we get there.”

LC-3: Memory-mapped IO (Table A.3 in text)

<table>
<thead>
<tr>
<th>Location</th>
<th>I/O Register</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFE00</td>
<td>Keyboard Status Reg (KBSR)</td>
<td>Bit [15] is one when keyboard has received a new character.</td>
</tr>
<tr>
<td>0xFE02</td>
<td>Keyboard Data Reg (KBDR)</td>
<td>Bits [7:0] contain the last character typed on keyboard.</td>
</tr>
<tr>
<td>0xFE04</td>
<td>Display Status Register (DSR)</td>
<td>Bit [15] is one when device ready to display another char on screen.</td>
</tr>
<tr>
<td>0xFE06</td>
<td>Display Data Register (DDR)</td>
<td>Character written to bits [7:0] will be displayed on screen.</td>
</tr>
</tbody>
</table>
Input from the keyboard

- When a character is typed:
  - Its ASCII code is placed in bits [7:0] of KBDR (bits [15:8] are always zero)
  - The “ready bit” (KBSR[15]) is set to one
  - Keyboard is disabled -- any typed characters will be ignored

- When KBDR is read:
  - KBSR[15] is set to zero, that is
  - Keyboard is enabled

---

Basic polling routine: GETC

```
POLL LDI R0, KBSRPtr
BRzp POLL
LDI R0, KBDRPtr
...
KBSRPtr .FILL xFE00
KBDRPtr .FILL xFE02
KBDR (xFEO2)
```

(look it up – it’s GETC, at x0400)
Output to Monitor

- When Monitor is ready to display another character:
  - The “ready bit” (DSR[15]) is set to one
- When data is written to Display Data Register:
  - DSR[15] is set to zero (clear ready bit)
  - Character in DDR[7:0] is displayed
  - Any other character data written to DDR is ignored while DSR[15] is zero
Basic polling routine: `PUTC` (out)

- Polling
  - `POLL LDI R1, DSRPtr`  
  - `BRzp POLL`  
  - `STI R0, DDRPtr`  
  - `...`
  - `DSRPtr .FILL xFE04`
  - `DDRPtr .FILL xFE06`
  - DSR (xFEO4)
  - DDR (xFEO6)

Hardware implementation of memory-mapped output

- Sets LD.DDR or selects DSR as input.
Keyboard Echo Routine

- Usually, input character is also printed to screen.

```
POLL1  LDI  R0, KBSRPtr
       BRzp POLL1
       LDI  R0, KBDRPtr
POLL2  LDI  R1, DSRPtr
       BRzp POLL2
       STI  R0, DDRPtr
...  
KBSRPtr .FILL xFE00
KBDRPtr .FILL xFE02
DSRPtr  .FILL xFE04
DDRPtr  .FILL xFE06
```
Interrupt-Driven I/O

- To implement an interrupt mechanism, we need three things
  - A way for the I/O device to signal the CPU that an interesting event has occurred
  - A way for the CPU to test whether the interrupt signal is set
  - A way for the CPU to test whether its priority is higher than the current program’s

Interrupt generation

- Software sets "interrupt enable" (IE) bit in device register (interrupt mask)
- When ready bit is set and IE bit is set, interrupt is signaled.
Testing for the interrupt signal

- CPU looks at signal between STORE and FETCH phases
  - If not set, continue with next instruction
  - If set, transfer control to interrupt service routine

Priority

- Every instruction executes at a stated level of urgency
- LC-3 has 8 priority levels, from PL0 (lowest) to PL7
  - It’s OK for PL6 device to interrupt PL0 program, but not the other way around
  - Example:
    - Payroll program runs at PL0.
    - Nuclear power correction program runs at PL6.
- Priority encoder selects highest-priority device, compares to current processor priority level, and generates interrupt signal if appropriate
Priority encoding for interrupt

Full implementation of LC-3 memory-mapped I/O

Because of interrupt enable bits, status registers (KBSR/DSR) must be writable, as well as readable.
Ok, the interrupt interrupted

- How am I (the program) going to resume execution after the interrupt service routine is done?
- Where do I save my stuff?
- We need a stack

Recommended exercises

- Good review questions: 8.3, 8.6, 8.9, 8.10, 8.14
- Interesting for interrupt enable: 8.15 (but the code requires… adjustments to actually work in the simulator)