LC3
Input and Output

(Textbook chapter 8)

Doing before
Instruction Processing
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 for I/O

• This lecture details
  – how I/O is read and written to
  – interrupts
I/O devices types

• Types of I/O devices characterized by:
  – **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
Programming Interface

• How are device registers identified?
  – Memory-mapped vs. I/O-mapped (special instructions)

• How is timing of transfer managed?
  – Synchronous vs. Asynchronous (not do)

• Who controls transfer?
  – CPU (polling) vs. device (interrupts)
Memory-Mapped I/O

- assign a range of memory addresses to I/O “ports”
- use same memory data movement instructions (load/store) for control and data transfer
- the hardware is designed to determine whether the instruction refers to a device or memory
I/O-mapped I/O

- specific opcode(s) for I/O (e.g. IN and OUT in x86 and DOS)
- two separate addressing spaces
But I/O isn’t Memory!

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
Transfer Control

• Who determines when the next data transfer occurs?

• Polling
  – CPU keeps checking status register until new data arrives OR device ready for next data
  – “Are we there yet? Are we there yet? Are we there yet?”

• Interrupts
  – Device sends a special signal to CPU when 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 I/O (Table A.3 in the textbook for the five I/O addresses in the LC-3)

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

- In LC3, addresses from xFE00 to xFFFF are reserved for I/O devices
Input from Keyboard

• When a character is typed:
  1. its ASCII code is placed in bits [7:0] of $\text{KBDR}$ (bits [15:8] are always zero)
  2. the “ready bit” ($\text{KBSR}[15]$) is set to one
  3. keyboard is disabled -- any typed characters will be ignored

• When $\text{KBDR}$ is read:
  – $\text{KBSR}[15]$ is set to zero
Basic Input Routine

Polling

new char?

NO

YES

read character

POLL  LDI  R0, KBSRPtr
BRzp  POLL
LDI  R0, KBDRPtr

KBSRPtr  .FILL  xFE00
KBDRPtr  .FILL  xFE02

(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
  - character in DDR[7:0] is displayed
  - any other character data written to DDR is ignored while DSR[15] is zero
Basic Output Routine

Polling

- screen ready?
  - NO
  - YES
- write character

POLL  LDI  R1, DSRPtr  
BRzp  POLL  
STI  R0, DDRPtr  

...  

DSRPtr  .FILL  xFE04  
DDRPtr  .FILL  xFE06  

Keyboard Read/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 
```
Echo Code with Registers

;page 226
;when MSB=1 of KBSR --> read Keyboard

Input      LDI     R3, KBSR ; Char typed?
BRzp       Input
LDI       R0, KBDR ; R0 <- char
L3         LDI   DSR
BRzp       L3
STI       R0, DDR ; echo to monitor
(bunch of instructions)

DSR       .FILL    xFE04
DDR       .FILL    xFE06
KBSR      .FILL    xFE00
KBDR      .FILL    xFE02
Interrupt-Driven I/O

Polling consumes a lot of cycles, especially for rare events – these cycles can be used for computation.

Example: Process previous input while collecting current input.

In a more efficient approach, an external device can:
(1) Force currently executing program to stop;
(2) Have the processor satisfy the device’s needs; and
(3) Resume the stopped program as if nothing happened.
Interrupt-Driven I/O

To implement an interrupt mechanism, we need:

– 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.

– Routine to “service” the interrupt

– Mechanism to return to interrupted program
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 Interrupt Signal

• CPU looks at signal after instruction is executed.
• If not set, continues with next instruction.
• If set, transfers control to interrupt service routine (after saving PC and CCR).
Priority

• Every instruction executes at a stated level of urgency.
• LC-3 has 8 priority levels, from PL0 (lowest) to PL7
  – Example:
    • Payroll program runs at PL0.
    • Nuclear power correction program runs at PL6.
  – It’s OK for PL6 device to interrupt PL0 program, but not the other way around.

• Priority encoder selects highest-priority device, compares to current processor priority level, and generates interrupt signal if appropriate.
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 – we’ll see that soon)
More Homework due November 9

• Good review questions: 8.3, 8.6, 8.9, 8.14