Chapter 8
I/O
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
• jump or branch depending on various conditions

But where does data in memory come from?

And how does data get out of the system so that humans can use it?
I/O: Connecting to the Outside World

Types of I/O devices characterized by:

- **behavior**: input, output, storage
  - input: keyboard, mouse, motion detector, etc
  - output: monitor, printer, etc
  - Input/output: disk, network, etc
- **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. special instructions

How is timing of transfer managed?
  • Asynchronous vs. synchronous

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

Memory-mapped

• assign a memory address to each device register
• use data movement instructions (LD/ST) for control, status and data transfer

I/O Instructions

• designate opcode(s) for I/O
• device and operation encoded in instruction
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 *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 perform other tasks instead of polling device.
- “Wake me when we get there.”
LC-3
Memory-mapped I/O  (Table A.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 character 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>

Asynchronous devices
- synchronized through status registers

Polling and Interrupts
- the details of interrupts will be discussed in Chapter 10
Input from 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
- keyboard is enabled
Basic Input Routine

Polling

new char?

START LDI R1, KBSR
BRzp START
LDI R0, KBDR

KBSR .FILL xFE00
KBDR .FILL xFE02
Simple Implementation: Memory-Mapped Input

Address Control Logic determines whether MDR is loaded from Memory or from KBSR/KBDR.
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

START
LDI R1, DSR
BRzp START
STI R0, DDR

... 

DSR .FILL xFE04
DDR .FILL xFE06
Sets LD.DDR or selects DSR as input.
Keyboard Echo Routine

Usually, input character is also printed to screen.

- User gets feedback on character typed and knows it's ok to type the next character.

```
START  LDI  R1, KBSR
       BRzp START
ECHO   LDI  R0, KBDR
       LDI  R1, DSR
       BRzp ECHO
       STI  R0, DDR
...
KBSR   .FILL xFE00
KBDR   .FILL xFE02
DSR    .FILL xFE04
DDR    .FILL xFE06
```
Interrupt-Driven I/O

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.

Why?

• Polling consumes a lot of cycles, especially for rare events – these cycles can be used for more computation.
• Example: Process previous input while collecting current input.
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 and whether its priority is higher than the current program.

Generating Signal

• Software sets "interrupt enable“ (IE) bit in device register.
• When ready bit is set and IE bit is set, interrupt is signaled.
Priority

Every instruction executes at a stated level of urgency.

**LC-3: 8 priority levels (PL0-PL7)**

- Example:
  - Payroll program runs at PL0.
  - Nuclear power plant 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.
Testing for Interrupt Signal

CPU looks at the interrupt signal between the STORE and FETCH phases, which makes instructions “atomic”. If not set, continues with next instruction. If set, transfers control to interrupt service routine (ISR).

More details in Chapter 10.
Because of interrupt enable bits, status registers (KBSR/DSR) must be written, as well as read.
꼭 기억해야 할 것

- I/O devices
  - Ways to connect to the outside world
- Device registers
  - Control registers
  - Status registers
  - Data registers
- Classifications
  - Memory-mapped vs. I/O instructions
  - Polling vs. Interrupt
  - Asynchronous vs. Synchronous