Chapter 8

Input/Output

- I/O basics
- Keyboard input
- Monitor output
- Interrupt driven I/O
  - DMA
CS Realities

- Computers do more than just execute your program
  - I/O
  - Interrupts
I/O Basics

• Definitions

  - Input
    • transfer data from the outside world to the computer:
      keyboard, mouse, scanner, bar-code reader, etc.

  - Output
    • transfer data from the computer to the outside:
      monitor, printer, LED display, etc.

  - Peripheral: any I/O device, including disks.

• LC-3 supports only a keyboard and a monitor
Device Registers

- **I/O Interface**
  - Through a set of Device Registers:
    - Status register (device is busy/idle/error)
    - Data register (data to be moved to/from device)
  - The device registers have to be read/written by the CPU.

- **LC-3**
  - KBDR: keyboard data register
  - KBSR: keyboard status register
  - DDR: display data register
  - DSR: display status register

  - **KBSR[15]** - keyboard ready (new character available)
  - **KBDR[7:0]** - character typed (ASCII)
  - **DSR[15]** - monitor ready
  - **DDR[7:0]** - character to be displayed (ASCII)
Addressing Device Registers

- **Special I/O Instructions**
  - Read or write to device registers using specialized I/O instructions.

- **Memory Mapped I/O**
  - Use existing data movement instructions (Load & Store).
  - Map each device register to a memory address (fixed).
  - CPU communicates with the device registers as if they were memory locations.
  - Frame buffers: Large areas of Memory Mapped I/O for video display

- **LC-3**
  - Uses memory mapped I/O:
    - xFE00  KBSR  Keyboard Status Register
    - xFE02  KBDR  Keyboard Data Register
    - XFE04  DSR   Display Status Register
    - XFE06  DDR   Display Data Register
    - XFFFE  MCR   Machine Control Register
Memory-mapped Input
Memory-mapped Output
Synchronizing CPU and I/O

- Problem
  - Speed mismatch between CPU and I/O
    - CPU runs at up to 4 GHz, while all I/O is much slower
  - Example: Keyboard input is both slow and irregular
  - We need a protocol to keep CPU & KBD synchronized
    - Two common approaches

- Polling (handshake synchronization)
  - CPU checks the KBD Ready status bit
  - If set, CPU reads the data register and resets the Ready bit
  - Repeat
  - Makes CPU-I/O interaction seem to be synchronous

- Interrupt-driven I/O
  - An external device is allowed to interrupt the CPU and demand attention
  - The CPU attends to the device in an orderly fashion (more later)
Polling v/s Interrupts (Who’s driving?)

● **Polling: CPU in charge**
  - CPU checks the ready bit of status register (as per program instructions).
    - If (KBSR[15] == 1) then load KBDR[7:0] to a register.
    - If the I/O device is very slow, CPU is kept busy waiting.

● **Interrupt: peripheral in charge**
  - Event triggered - when the I/O device is ready, it sets a flag called an interrupt
  - When an interrupt is set, the CPU is forced to an *interrupt service routine* (ISR)
    which services the interrupting device
  - There can be different priority levels of interrupt
  - Specialized instructions can *mask* an interrupt level
Polling Algorithm

- **Input (keyboard)**
  - The CPU loops checking the Ready bit
  - When bit is set, a new character is available
  - CPU loads the character waiting in the keyboard data register

- **Output (monitor)**
  - CPU loops checking the Ready bit
  - When bit is set, display is ready for next character
  - CPU stores a character in display data register
Polling details

- **Keyboard**
  - When key is struck
    - ASCII code of character is written to KBDR[7:0] (least significant byte of data register)
    - KBSR[15] (Ready Bit) is set to 1 – this locks the keyboard
    - The CPU checks Ready Bit and reads KBDR
    - When KBDR is ready, ready bit is cleared and keyboard is unlocked

- **Monitor**
  - When CPU is ready to output a character
    - CPU checks DSR[15] (Ready Bit) until it is set to 1 (ready!)
    - CPU writes character to DDR[7:0]
    - Monitor sets DSR[15] to 0 (locked/not ready) while it is busy displaying the character
    - Monitor sets DSR[15] to 1 (unlocked/ready) when the character has been displayed and it is ready for a new character
Simple Polling Routines

### Input a character from keyboard

```
START LDI R1, A ;Loop if Ready not set
    BRzp START
    LDI R0, B ;If set, load char to R0
    BR NEXT_TASK
A .FILL xFE00 ;Address of KBSR
B .FILL xFE02 ;Address of KBDR
```

### Output a character to the monitor

```
START LDI R1, A ;Loop if Ready not set
    BRzp START
    STI R0, B ;If set, send char to DDR
    BR NEXT_TASK
A .FILL xFE04 ;Address of DSR
B .FILL xFE06 ;Address of DDR
```

Wright State University, College of Engineering
Dr. Doom, Computer Science & Engineering
CEG 320/520 Comp. Org. & Assembly
Keyboard Echo: combine the above

<table>
<thead>
<tr>
<th>START</th>
<th>LDI</th>
<th>R1, KBSR</th>
<th>;Loop if KB not ready</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BRzp</td>
<td>START</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LDI</td>
<td>R0, KBDR</td>
<td>;Get character</td>
</tr>
<tr>
<td>ECHO</td>
<td>LDI</td>
<td>R1, DSR</td>
<td>;Loop if monitor not ready</td>
</tr>
<tr>
<td></td>
<td>BRzp</td>
<td>ECHO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>STI</td>
<td>R0, DDR</td>
<td>;Send character</td>
</tr>
<tr>
<td></td>
<td>BR</td>
<td>NEXT_TASK</td>
<td></td>
</tr>
<tr>
<td>KBSR</td>
<td>.FILL</td>
<td>xFE00</td>
<td>;Address of KBSR</td>
</tr>
<tr>
<td>KBDR</td>
<td>.FILL</td>
<td>xFE02</td>
<td>;Address of KBDR</td>
</tr>
<tr>
<td>DSR</td>
<td>.FILL</td>
<td>xFE04</td>
<td>;Address of DSR</td>
</tr>
<tr>
<td>DDR</td>
<td>.FILL</td>
<td>xFE06</td>
<td>;Address of DDR</td>
</tr>
</tbody>
</table>
Example: Print a string

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEA R1, STR</td>
<td>Load address of string</td>
</tr>
<tr>
<td>LOOP</td>
<td></td>
</tr>
<tr>
<td>LDR R0, R1, #0</td>
<td>get next char to R0</td>
</tr>
<tr>
<td>BRZ DONE</td>
<td>string ends with 0</td>
</tr>
<tr>
<td>LP2</td>
<td></td>
</tr>
<tr>
<td>LDI R3, DSR</td>
<td>Loop until MON is ready</td>
</tr>
<tr>
<td>BRzp LP2</td>
<td></td>
</tr>
<tr>
<td>STI R0, DDR</td>
<td>Write next character</td>
</tr>
<tr>
<td>ADD R1, R1, #1</td>
<td>Set address to next char</td>
</tr>
<tr>
<td>BR LOOP</td>
<td></td>
</tr>
<tr>
<td>STR .STRINGZ &quot;Char String&quot;</td>
<td></td>
</tr>
<tr>
<td>DONE HALT</td>
<td></td>
</tr>
</tbody>
</table>
There has got to be a better way!

- Later, we will talk about how *Interrupt Driven I/O* can be used to make this entire process much more processor efficient.
What about my PC?

PCs use Memory Mapped I/O
With Interrupts
Practice problems

- 8.7, 8.13, 8.14, 8.16