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

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:
    - $\text{xFE00} \text{ KBIR: Keyboard Status Register}$
    - $\text{xFE02} \text{ KBDR: Keyboard Data Register}$
    - $\text{xFE04} \text{ DSR: Display Status Register}$
    - $\text{xFE06} \text{ DDR: Display Data Register}$
    - $\text{xFFFE} \text{ MCR: Machine Control Register}$

Memory-mapped Input

- 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

- 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:
      - $\text{xFE00} \text{ KBIR: Keyboard Status Register}$
      - $\text{xFE02} \text{ KBDR: Keyboard Data Register}$
      - $\text{xFE04} \text{ DSR: Display Status Register}$
      - $\text{xFE06} \text{ DDR: Display Data Register}$
      - $\text{xFFFE} \text{ MCR: Machine Control Register}$
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 & I/O 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)
  - There can be different priority levels of interrupt
  - Specialized instructions can mask an interrupt level

Polling Algorithm

- **Input (keyboard)**
  - 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 [KBSR][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 KBDIR
    - When KBDIR 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 1 (unlocked/ready) when the character has been displayed and it is ready for a new character

Simple Polling Routines

```
START LDI R1, A ; Loop if Ready not set
  BRp START
LDI R0, B ; If set, load char to R0
  BR NEXT_TASK
A FILL $FE00 ; Address of KBSR
B FILL $FE02 ; Address of KBDR

Input a character from keyboard
```

```
START LDI R1, A ; Loop if Ready not set
  BRp START
LDI R0, B ; If set, send char to DDR
  STI R0, B
BR NEXT_TASK
A FILL $FE04 ; Address of DSR
B FILL $FE06 ; Address of DDR

Output a character to the monitor
```
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.

Practice problems

- 8.7, 8.13, 8.14, 8.16