

# Notes: I/O (Chapter 8)

## A. Why?

- Shipping data between the outside world and CPU/memory brings up problems with speed differences, asynchronous processes, and data representation and transmission.

## B. Outcomes

By the end of class you should:

- Understand what data and control I/O registers do.
- Understand the difference between memory-mapped I/O and special I/O instructions.
- Understand the difference between polled I/O and interrupt-driven I/O.

## C. Peripherals

- A **peripheral devices** is any device outside the CPU
- Characterize by direction of data
  - **input**: keyboard, motion detector, network interface
  - **output**: monitor, printer, network interface
  - **storage**: disk, CD-ROM
- Characterize by data transfer rate (higher rates impose more restrictions)
  - keyboard: 100 bytes/sec
  - disk: 30 MB/s
  - network: 1 Mb/s - 1 Gb/s
- I/O registers are interfaces to device
  - Data registers hold data going to/from device
  - Control/status registers tell a device what to do/tell us what's been done
  - Programming questions: How do we access device registers? Who controls changes? Who has to check statuses?
- LC-3 only has two I/O devices: keyboard and monitor
  - Status and data registers for each device, registers read/written by CPU
  - Status register (device is busy/idle/error)
  - Data register (data to be moved to/from device)
  - Similar to memory registers (MAR and MDR are data registers)
- LC-3 has 4 device registers:
  - KBSR (Keyboard Status Register)
    - Bit [15] = 1: Keyboard has received a new character.
  - KBDR (Keyboard Data Register)
    - Bits [7:0] = Last character typed on keyboard.

- DSR (Display Status Register)
  - Bit [15] = 1: Device ready to display another char on screen.
- DDR (Display Data Register)
  - Character in bits [7:0] will be displayed on screen.

## D. Addressing Device Registers

### Technique # 1: Special I/O Instructions

- Read or write to device registers using specialized I/O instructions.
- Need opcode for I/O; instruction includes codes for device, operation, data (or pointer to data).

### Technique #2: Memory Mapped I/O (used by LC-3)

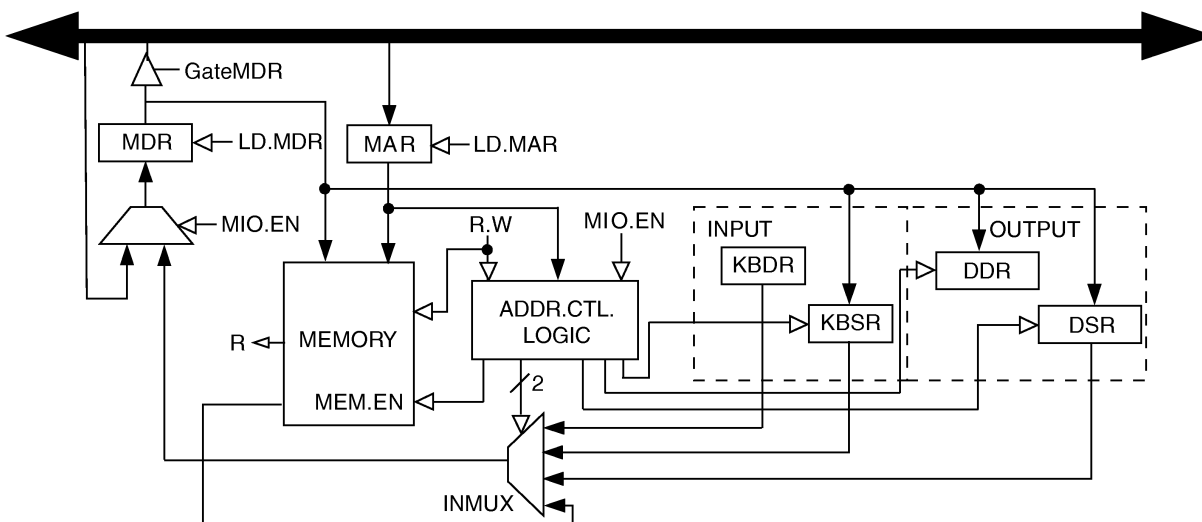
- Map each device register to a fixed memory address.
- Use Load and Store to access the device registers (as if they were memory locations).
- Actual memory locations no longer accessible to program.
- LC-3 device registers and locations:
  - KBSR: xFE00; KBDR: xFE02; DSR: xFE04; DDR: xFE06

- Example

```
LD    R1, xFE00    ; Get keyboard status
BRN   HAVECHAR    ; Branch if keyboard has a character
...           ; (If we're here, there wasn't a character)
```

- We don't deal with device registers directly; TRAP code does it for us.
- LC-3 Keyboard Input
  - When character is typed, keyboard hardware
    - Sets KBDR[7:0] ← ASCII code of char
    - Sets KBDR[15:8] ← 0
    - Sets KBSR[15] ← 1 (enable "ready bit": Ready to be read)
    - Disables keyboard (hitting keys will do nothing)
  - Program gets character from keyboard by loading KBDR; keyboard hardware
    - Sets KBSR[15] ← 0 (disable ready bit)
    - Re-enables keyboard (hitting a key types a character)
- LC-3 Monitor Output
  - When monitor is ready to display a char:
    - DSR[15] ← 1 (enable "ready bit")
  - To display a character, set DDR[7:0] ← ASCII code of char
    - Monitor sets DSR[15] ← 0 (disable "ready bit"), starts displaying character
  - When ready bit = 0, writes to DDR[7:0] are ignored.

## LC-3 Memory-mapped Input/Output



### E. Handling I/O Operation Timing

- I/O is typically much slower than CPU operation
  - CPU: 2 GHz (but maybe 5 cycles per load instruction)
  - Hard disks: 125 MB/sec; keyboards: Maybe 100 words/min
- Synchronization problems
  - CPU produces/consumes data too quickly? Waits a lot?
  - Is data transfer rate constant (= synchronous)? Unpredictable (= asynchronous)?
- Polling versus interrupts
  - Polled I/O: Check for I/O at regular intervals
  - Interrupt-driven I/O: Change in I/O status causes change in program execution
    - External device sends interrupt signal: “Event E just occurred”
    - CPU stops executing current program P
    - Then executes code for this kind of interrupt
    - Then CPU goes back to executing program P
    - Interrupted program doesn't notice interruption (unless it checks wall clock).

### F. First Look at Interrupts

- How do we notice interrupts?
  - Modify the instruction cycle
  - Add new step between STORE RESULT and FETCH next instruction
    - If interrupt is being signaled
      - Set PC ← address of code that handles this kind of interrupt
      - More details in Ch. 10 (like, how we remember where we were).

- LC-3 interrupts:
  - Keyboard ready: KSBR[15] gets set to 1; Monitor ready: DSR[15] gets set to 1
- Other machines:
  - System timer: Occurs periodically; Hard disk, other various devices; Page fault (for virtual memory); Power off
- Handling interrupts is complicated (See Ch. 10)
  - What if two interrupts occur simultaneously or a second interrupt occurs while you're handling the first interrupt?
  - Establish priorities, handle higher-priority interrupt.
    - Ignore if lower priority? Keep it pending until you finish handling this interrupt?
- Disabling/Enabling Interrupts
- Operating system may disable interrupts for limited time
  - For "critical sections" of code
  - Hardware implements this by having "Interrupt Enable" bit
  - LC-3 uses bit 14 of keyboard and monitor status registers (CPU can disable interrupts by setting bit 14  $\leftarrow$  0).
  - Other machines might have one master bit, might attach priorities