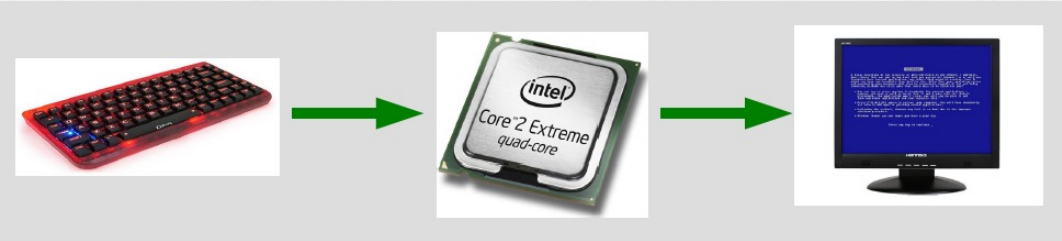
Comp1003 Computer Organization

Lecture 12 Input & Output Mechanism



United International College

I/O: Connecting to the outside world

- So far, we have learned how to
  - Compute values with registers
  - Load data from memory to registers
  - Store data from registers to memory
- But how to get the from the outside world into the memory?
- And how to get information out of the box so that human can understand?

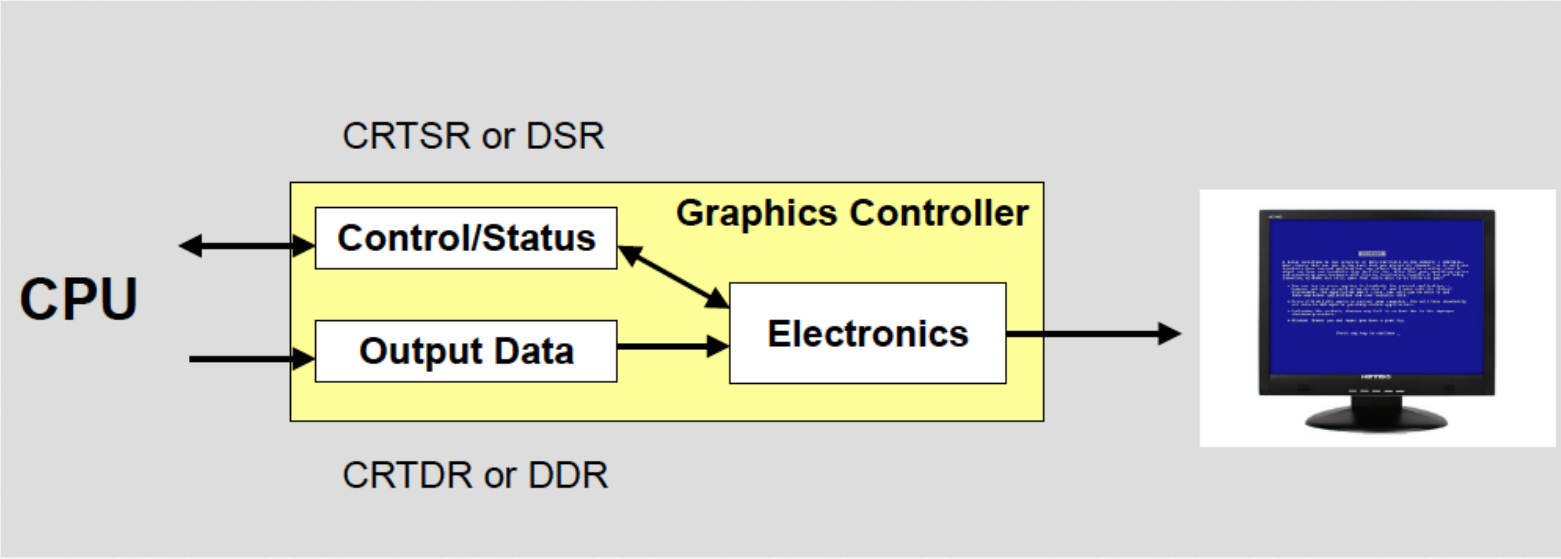
I/O Devices

- Types of I/O devices
  - By behavior
    - **input: keyboard, motion detector, network interface**
    - **output: monitor, printer, network interface**
    - **storage: disk, CD-ROM**
  - By speed (data rate) how fast can data be transfered?
    - **keyboard: 100 bytes/sec**
    - **disk: 30 MB/s**
    - **network: 1 Mb/s - 1 Gb/s**

I/O Devices & Electronics

- Device Registers
  - **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, etc.

The Big Picture



I/O 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)

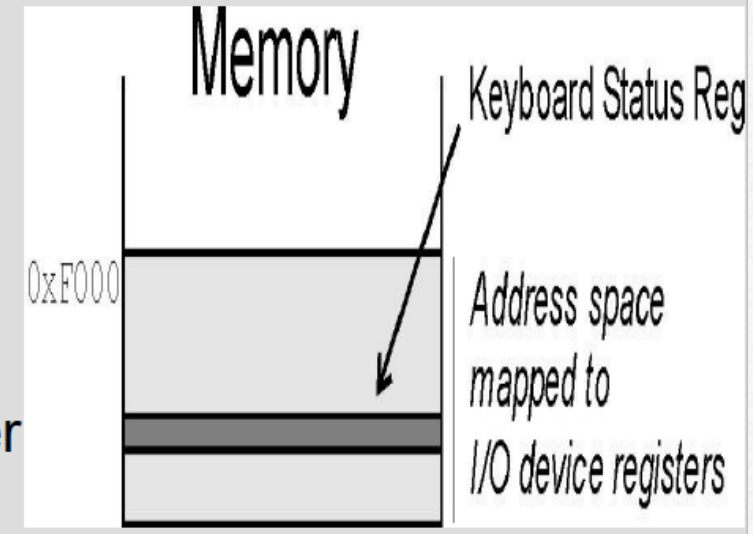
Memory-mapped I/O vs Special I/O Instructions

I/O Instructions

- designate opcode(s) for I/O
- register and operation encoded in instruction

Memory-mapped

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



The main problem: 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: Polling

- 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?" .....

Transfer Control: Interrupts

- Interrupts (中断)
  - 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.
  - “call me when you need my attention.”

LC-3 I/O

- Address xFE00 to xFFFF are reserved for input/output registers

Location	*I/O Register*	*Function*
xFE00	Keyboard Status Reg (KBSR)	Bit [15] is one when keyboard has received a new character.
xFE02	Keyboard Data Reg (KBDR)	Bits [7:0] contain the last character typed on keyboard.
xFE04	CRT Status Register (CRTSR)	Bit [15] is one when device ready to display another char on screen.
xFE06	CRT Data Register (CRTDR)	Character written to bits [7:0] will be displayed on screen.

Asynchronous devices

- **synchronized through status registers**

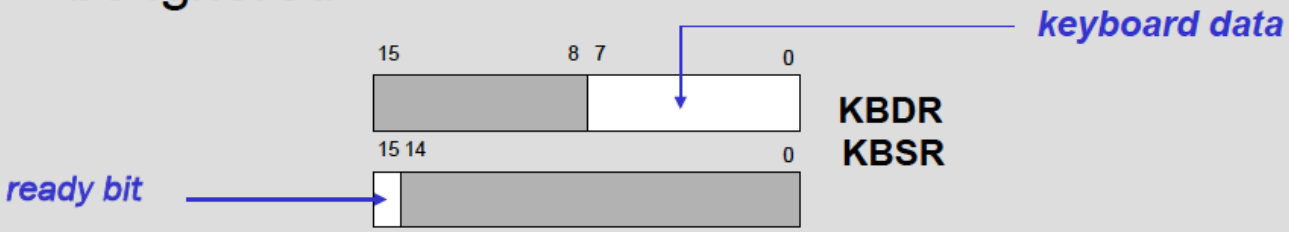
Polling and Interrupts

- **the details of interrupts will be discussed in Chapter 10**

Keyboard Input

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



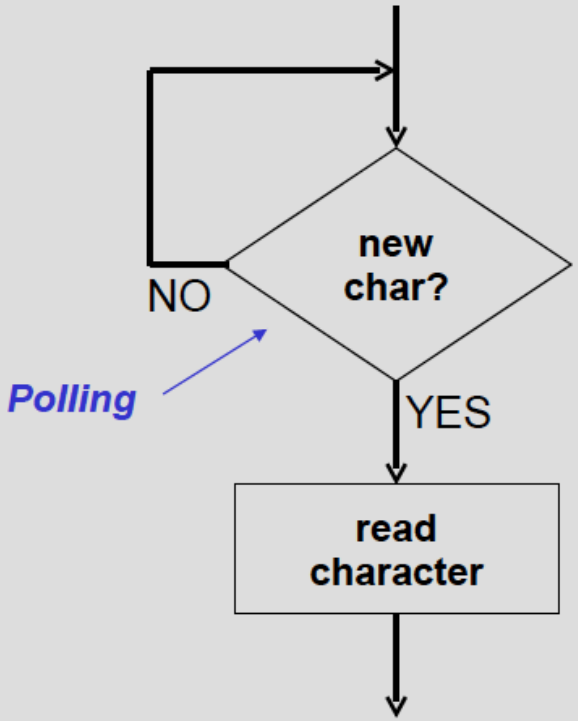
When KBDR is read:

- KBSR[15] is set to zero
- keyboard is enabled

Memory-mapped Operations

- How do we read ready bit?
- How do we test whether the bit is one?
- How do we read keyboard data?

Basic Input Routine: Polling

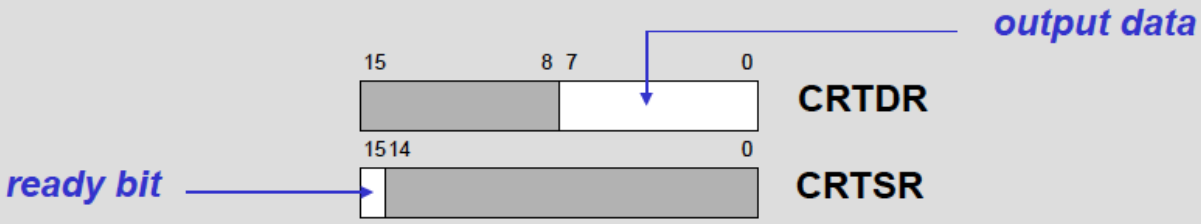



```
POLL    LDI  R0, KBSR
        BRzp POLL
        LDI  R0, KBDR

        ...

KBSR    .FILL xFE00
KBDR    .FILL xFE02
```

Output to Screen

When CRT device is ready to display another character:

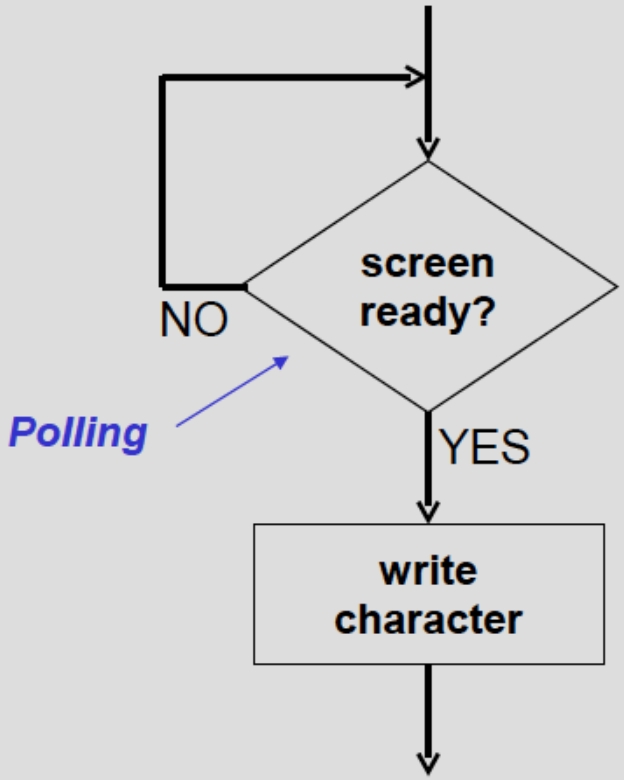
- the "ready bit" (CRTSR[15]) is set to one



When data is written to CRT data register:

- CRTSR[15] is set to zero
- character in CRTDR[7:0] is displayed
- any other character data written to CRTDR is ignored (while CRTSR[15] is zero)

Basic Output Routine



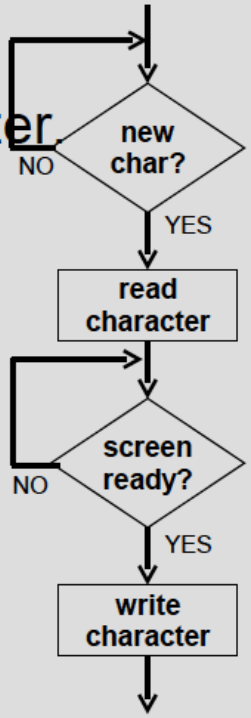

```
POLL    LDI  R1, CRTSR
        BRzp POLL
        STI  R0, CRTDR

        ...

CRTSR   .FILL xFE04
CRTDR   .FILL xFE06
```

Keyboard Echo Routine

Usually, input character is also printed to screen.

- User gets feedback on character typed and knows its ok to type the next character

```
POLL1   LDI  R0, KBSR
        BRzp POLL1
        LDI  R0, KBDR
POLL2   LDI  R1, CRTSR
        BRzp POLL2
        STI  R0, CRTDR

        ...

KBSR    .FILL xFE00
KBDR    .FILL xFE02
CRTSR   .FILL xFE04
CRTDR   .FILL xFE06
```



Interrupt-Driven I/O

Polling: the interaction between the CPU and the I/O device is controlled by the CPU

Interrupt-driven : the interaction between the CPU and the I/O device is controlled by the I/O device which can

1. Force the current executing program to stop
2. Have the processor carry out the needs of I/O device
3. Have the stopped program resume execution afterwards as if nothing had happened

Why Interrupted-driven I/O?

- Polling consumes a lot of cycles checking the Ready bit, especially for rare I/O events – these cycles can be used for more computation.
  - Example: Process previous input while collecting current input. (See Example 8.1 on page 211.)

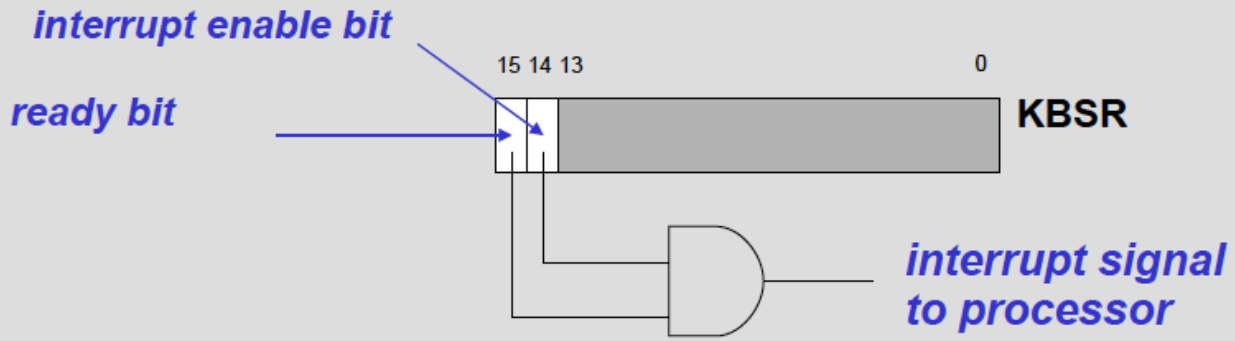
The Interrupt Mechanism

- 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 INT Signal

Generating Signal

- Software sets "interrupt enable" 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.

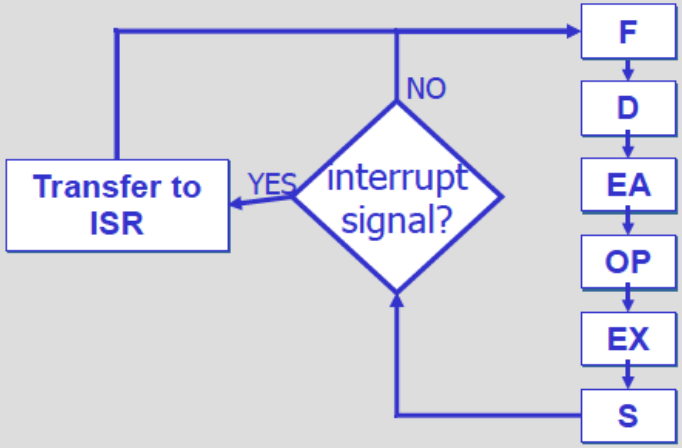
LC-3: 8 priority levels (PL0-PL7)

- Example:
  - Payroll program runs at PL0.
  - Nuclear power plant control 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 signal between STORE and FETCH phases
- If not set, continues with next instruction
- If set, transfers control to interrupt service routine



Questions

- What is the danger of not testing the CRTSR before writing data to the screen?
- What is the danger of not testing the KBSR before reading data from the keyboard?
- What if the CRT were a synchronous device, e.g., we know that it will be ready 1 microsecond after character is written.
  - Can we avoid polling? How?
  - What are advantages and disadvantages?

More Questions

- Do you think polling is a good approach for other devices, such as a disk or a network interface?
- What is advantage of using LDI/STI for accessing device registers?

What's Next?

- How does the CPU respond to the interrupt signals?
  - Trap Routines