

Chapter 8

Input/Output

Based on slides © McGraw-Hill
Additional material © 2004/2005 Lewis/Martin
Modified by Diana Palsetia

Input/Output: Connecting to the Outside World

So far, we've learned how to...

- Compute with values in registers
- Move data between memory and registers

But how do we interact with computers?

- Game console (Playstation, Xbox)
- DVD player
- MP3 player (iPod)
- Cell phone
- Automated Teller Machine (ATM)
- Car's airbag controller
- Web server

CIT 593

8-2

Examples of Input/Output (I/O) Devices

User output

- Display, printer, speakers

User input

- Keyboard, mouse, trackball, game controller, scanner, microphone, touch screens, camera (still and video)

Storage

- Disk drives, CD & DVD drives, flash-based storage, tape drive

Communication

- Network (wired, wireless, optical, infrared), modem

Sensor inputs

- Temperature, vibration, motion, acceleration, GPS
- Barcode scanner, magnetic strip reader, RFID reader

Control outputs

- Motors, actuators

CIT 593

8-3

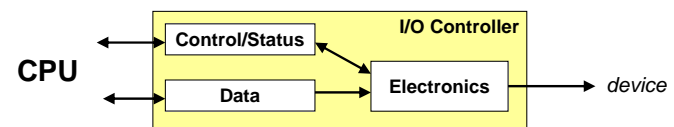
I/O Basics

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

CIT 593

8-4

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)

CIT 593

8-5

Memory-Mapped vs. I/O Instructions

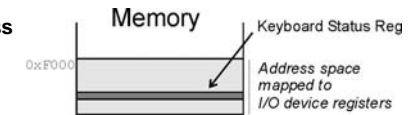
Instructions

- 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
- Hardware intercepts these address
- No actual memory access performed



CIT 593

8-6

Transfer Timing

Synchronous

- Data supplied at a fixed, predictable rate
- CPU reads/writes every X cycles
- Infrequently used because of speed difference
 - I/O is much slower than the processor
 - E.g. A 300 Mhz processor can execute an instruction every 33 nanoseconds
 - Humans don't type as fast as a processor can read

Asynchronous

- Data rate less predictable
- CPU must synchronize with device, so that it doesn't miss data or write too quickly
- Handles the speed mismatch

CIT 593

8-7

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 be performing other tasks instead of polling device
- “Wake me when we get there.”

CIT 593

8-8

LC-3 I/O

Memory-mapped I/O (Table A.3)

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	Display Status Register (DSR)	Bit [15] is one when device ready to display another char on screen.
xFE06	Display Data Register (DDR)	Character written to bits [7:0] will be displayed on screen.
xFE08	Timer Status Register (TSR)	Bit [15] is one when timer goes off; cleared when read.
xFE0A	Timer Interval Register (TIR)	Timer interval in msecs.

Asynchronous devices

- Synchronized through status registers

Polling and Interrupts

- We'll talk first about polling, a bit on interrupts later

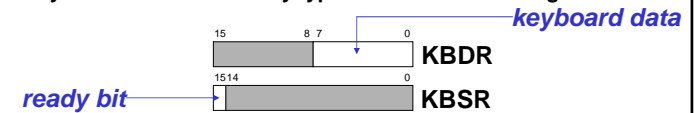
CIT 593

8-9

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



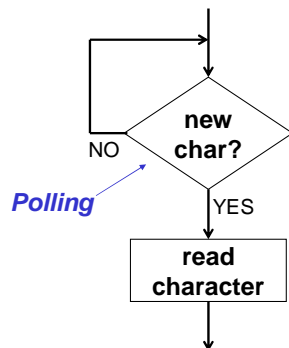
When KBDR is read by processor:

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

CIT 593

8-10

Basic Input Routine



```

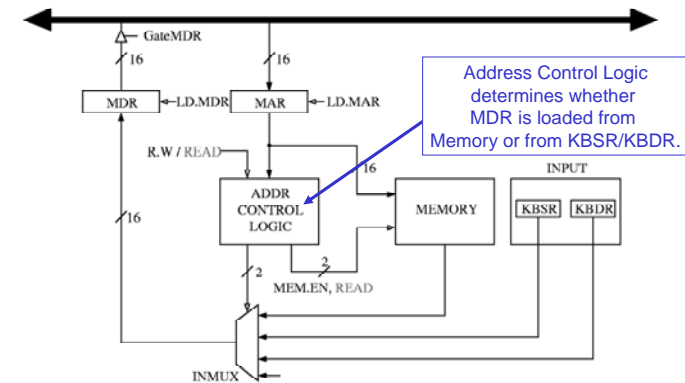
POLL  LDI R0, KBSRPtr
      BRzp POLL
      LDI R0, KBDRPtr
      ...
      KBSRPtr .FILL xFE00
      KBDRPtr .FILL xFE02
  
```

Polling

CIT 593

8-11

Simple Implementation: Memory-Mapped Input



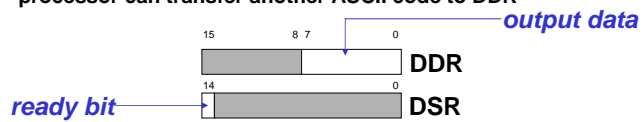
CIT 593

8-12

Output to Monitor

When Monitor is ready to display another character:

- The "ready bit" (DSR[15]) is set to one, indicating that the processor can transfer another ASCII code to DDR



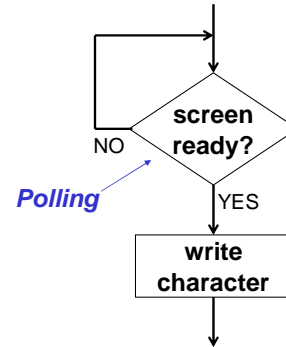
When data is written to Display Data Register:

- DSR[15] is set to zero
- While DSR[15] is zero, the monitor is still processing the previous character
- Character in DDR[7:0] is displayed

CIT 593

8-13

Basic Output Routine



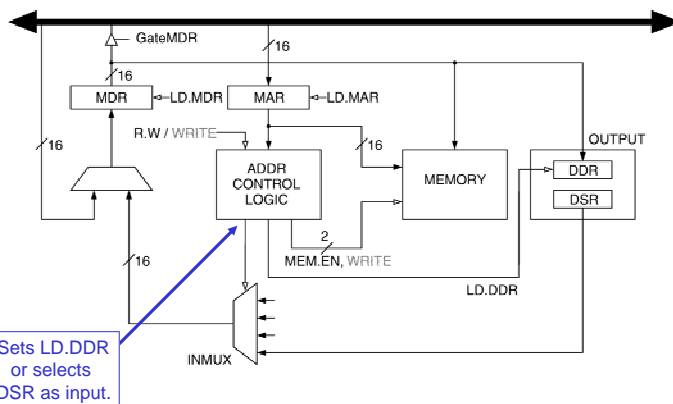
Polling

```
POLL  LDI  R1, DSRPtr
      BRzp POLL
      STI  R0, DDRPtr
      ...
      DSRPtr .FILL xFE04
      DDRPtr .FILL xFE06
```

CIT 593

8-14

Simple Implementation: Memory-Mapped Output



CIT 593

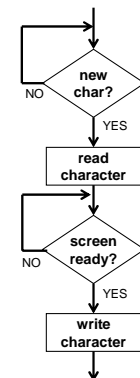
8-15

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, KBSRPtr
      BRzp POLL1
      LDI  R0, KBDRPtr
      LDI  R1, DSRPtr
      BRzp POLL2
      STI  R0, DDRPtr
      ...
      KBSRPtr .FILL xFE00
      KBDRPtr .FILL xFE02
      DSRPtr .FILL xFE04
      DDRPtr .FILL xFE06
```



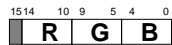
CIT 593

8-16

Pixel-Based Display

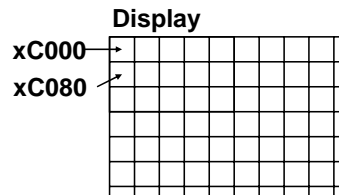
A display consists of many dots (pixels)

- Color of each pixel represented by a 16-bit value
 - 5 bits for each of Red/Green/Blue
 - 32 thousand distinct colors



Memory-mapped pixels

- One memory location per pixel
- 128x124 pixels
- Memory region xC000 to xFDFF
 - xC000 to xC07F is first row of display
 - xC080 to xC0FF is second row of display
- Set the corresponding location to change its color



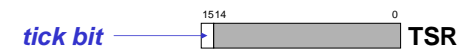
CIT 593

8-17

Timer Device

A periodic timer “tick”

- Allows a program to detect when an interval of time has passed
- Our implementation (for the LC-3) uses a simple fix-interval timer



Using TSR (Timer Status Register):

- “Tick” bit is set every n milliseconds
- Read the value of the bit from memory location (xFE08)
- Bit reset to zero after every read
- Change interval via Timer Interval Register (TIR, xFE0A)

CIT 593

8-18

Interrupt-Driven I/O

Why?

- Polling consumes a lot of machine processing cycles, especially for rare events – these cycles can be used for more computation
- Again, I/O devices are slow
- Example: re-executing again and again the LDI and BR instructions until the Ready bit is set.

I/o device can. . .

- Notify the processor that it needs attention
- Have the processor satisfy the device’s needs
 - force currently executing program to stop (only if it has a higher priority)
- Resume the stopped program as if nothing happened

CIT 593

8-19

To implement an Interrupt mechanism

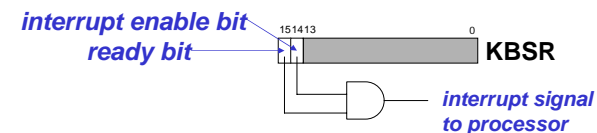
(1) Way for I/O device to **signal** CPU that its wants service

- We already have a **Ready** bit indicating that



(2) Way to know that device **has a right to request service**

- “Interrupt enable bit” in device register is set by processor
 - Depending on whether processor wants to give I/O device service
- When ready bit is set and IE bit is set, interrupt is signaled
I/O device will get service



CIT 593

8-20

To implement an Interrupt mechanism (contd..)

(3) Whether I/O device has higher **priority** among multiple I/O requests AND with the current program in execution

- Every instruction executes at a stated level of urgency
- LC-3: 8 priority levels (PL0-PL7)
- Example:
 - Payroll program runs at PL0
 - Nuclear power correction 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

CIT 593

8-21

Role of the Operating System

In real systems, only the operating system (OS) does I/O

- “Normal” programs ask the OS to perform I/O on its behalf

Hardware prevents non-operating system code from

- Accessing I/O registers
- Operating system code and data
- Accessing the code and data of other programs

Why?

- Protect programs from themselves
- Protect programs from each other
- Multi-user environments

CIT 593

8-22

Memory Protection

The hardware has two modes

- “Supervisor” or “privileged” mode
- “User” or “unprivileged” mode

Code in privileged mode

- Can do *anything*
- Used exclusively by the operating system

Code in user mode

- Can't access I/O parts of memory
- Can only access some parts of memory

Division of labor

- OS - make policy choices
- Hardware - enforce the OS's policy

CIT 593

8-23

OS and Hardware Cooperate for Protection

Hardware support for protected memory

- For example, consider a 16-bit protection register (MPR) in the LC3 processor (this not in text book)
 - MPR[0] corresponds to x0000 - x0FFF
 - MPR[1] corresponds to x1000 - x1FFF
 - MPR[2] corresponds to x2000 - x2FFF, etc.

When a processor performs a load or store

- Checks the corresponding bit in MPR
- If MPR bit is not set (and not in privileged mode)
 - Trigger illegal access

The OS must set these bits before running each program

- Example, if a program should access only x4000 - x6FFF
 - OS sets MPR[4, 5, 6] to 1 (clears rest)

CIT 593

8-24

Invoking the Operating System

How does non-privileged code perform I/O?

- Answer: it doesn't; it asks the OS to perform I/O on its behalf

How is this done?

- Making a system call into the operating system

In LC-3: The TRAP instruction

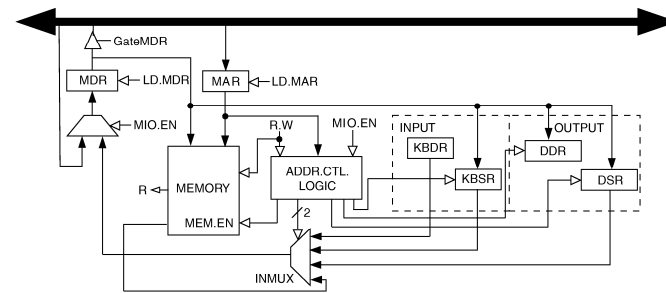
- Calls into the operating system (sets privileged mode)
- Different part of the OS called for each trap number
- OS performs the operations (in privileged mode)
- OS leaves privileged mode
- OS returns control back to user program (jumps to the PC after the TRAP instruction)

Topic of next chapter...

CIT 593

8-25

Full Implementation of LC-3 Memory-Mapped I/O



Because of interrupt enable bits, status registers (KBSR/DSR) must be written, as well as read.

CIT 593

8-26

Discussion Questions

What is the danger of not testing the DSR before writing data to the screen?

What is the danger of not testing the KBSR before reading data from the keyboard?

What if the display was 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?

CIT 593

8-27

Next Time

Topic

- Chapter 9

Reading

- Chapter 9

Quiz

- Tuesday 10th October in class (last 20 mins of lecture)
- Everything covered till Chapter 8 and Unix material
- Short answers, multiple choice, interpret or write small length assembly code

CIT 593

8-28