

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?

- Outside World
 - Receiving Information to process
 - Outputting processed information

CIT 593

2/32

Examples of Input/Output (I/O) Devices

Standard Interfaces

1. User output
 - Console or Prompt (just text), Video Display, printer, speakers
2. User input
 - Keyboard, mouse, trackball, game controller, scanner, microphone, touch screens, camera (still and video), game controllers
3. Storage
 - Disk drives, CD & DVD drives, flash-based storage, tape drive

More...

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

3/32

LC-3 I/O

I/O is done via the TRAP instruction

| Code | Equivalent | Description |
|-------------|------------|--|
| OUT | TRAP x21 | Write one character (in R0[7:0]) to console. |
| GETC | TRAP x20 | Read one character from keyboard. Character stored in R0[7:0]. |
| IN | TRAP x23 | Read (and echo) one character from keyboard. Character stored in R0[7:0]. |
| PUTS | TRAP x22 | Write null-terminated string to console. Starting address of string is in R0. |

CIT 593

4/32

Example 1

```

; Prints ABCDEFGHI. Each Character is printed on a newline
.ORIG x3000
LD R3, A      ; R3 = x0041 = 'A'
LD R4, J      ; R4 = x004A = 'J'
NOT R4, R4
ADD R4, R4, #1 ; 2's COMP of R4
PCHAR: AND R0, R0, #0 ; CLEAR R0
      ADD R0, R0, R3 ; R0 = R3
      OUT          ; DISPLAY CHAR
      AND R0, R0, #0 ; CLEAR R0
      LD R0, NEWL   ; R0 = x000A
      OUT          ; DISPLAY NEWLINE
      ADD R3, R3, #1 ; R3 + 1 for NEXT CHAR
      AND R5, R5, #0
      ADD R5, R3, R4 ; CHECK if we reached J
      BRn PCHAR
      HALT         ; TERMINATE
;*****DATA*****
A: .FILL x0041
NEWL: .FILL x000A
J: .FILL x004A
.END

```

CIT 593

5/32

EXAMPLE 2

;Prompt the user to enter a number, quits when pressed 3, prints DONE before terminating

```

.ORIG x3000
LD R1, VAL3
NOT R1, R1
ADD R1, R1, #1
LEA R0, ENTER
PUTS          ; displays the string
LOOP: IN      ; read and echo
      AND R3, R3, #0
      ADD R3, R1, R0
      BRnp LOOP
      LEA R0, ENDMSG ; R0 = starting address of string
      PUTS
      HALT
ENTER: .STRINGZ "Enter a number from 1 to 9:\n"
ENDMSG: .STRINGZ "\nDONE"
VAL3: .FILL x0033
.END

```

CIT 593

6/32

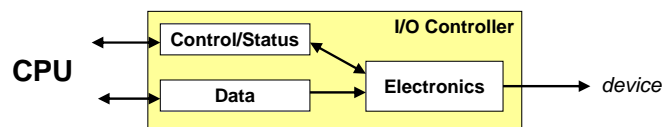
I/O Basics

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

7/32

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)

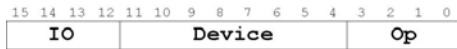
CIT 593

8/32

Memory-Mapped vs. I/O Instructions

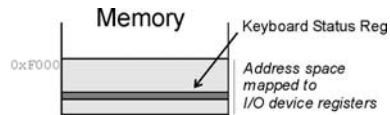
Instructions

- 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

9/32

Transfer Timing

Synchronous

- Data supplied at a fixed, predictable rate
- CPU reads/writes every X time units
- 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
 - E.g. 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

10/32

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

11/32

LC-3 I/O

Memory-mapped I/O (Table A.3 in Appendix A of text)

| 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. |

Asynchronous devices

- Synchronized through status registers
- Two ways: **Polling** and **Interrupts**
- We'll talk first about polling, a bit on interrupts later

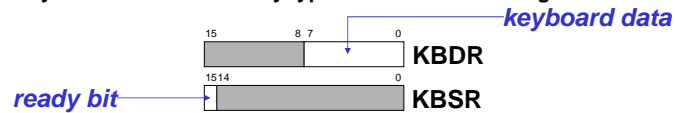
CIT 593

12/32

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



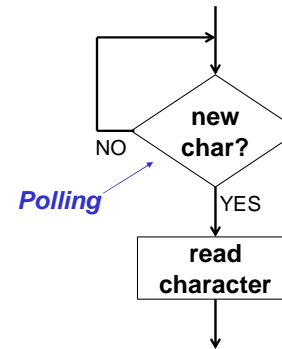
When KBDR is read by processor:

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

CIT 593

13/32

Basic Input Routine (GETC)



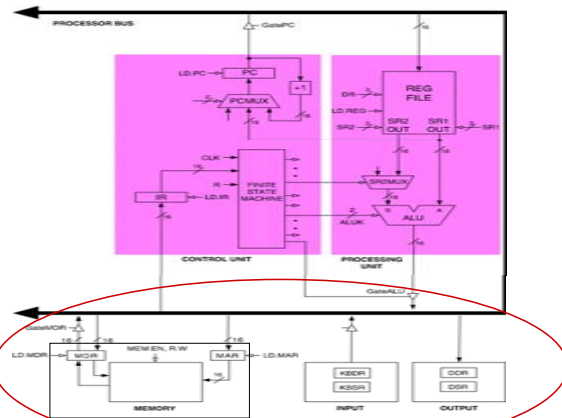
Polling

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

CIT 593

14/32

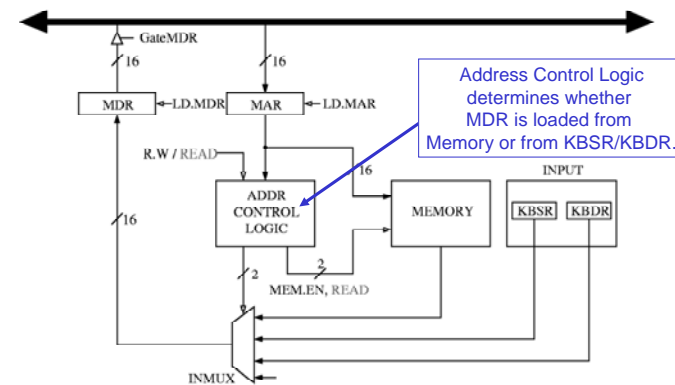
LC-3 Von Nuemann Model



CIT 593

15/32

Simple Implementation: Memory-Mapped Input



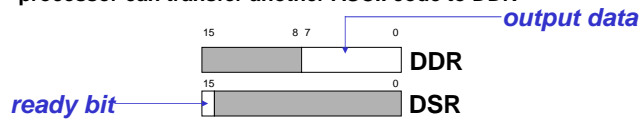
CIT 593

16/32

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



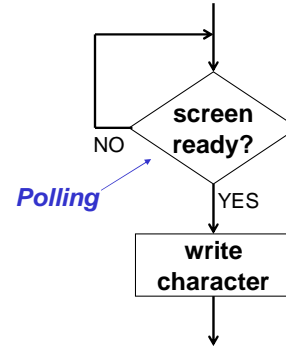
When data is written to Display Data Register:

- 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

17/32

Basic Output Routine



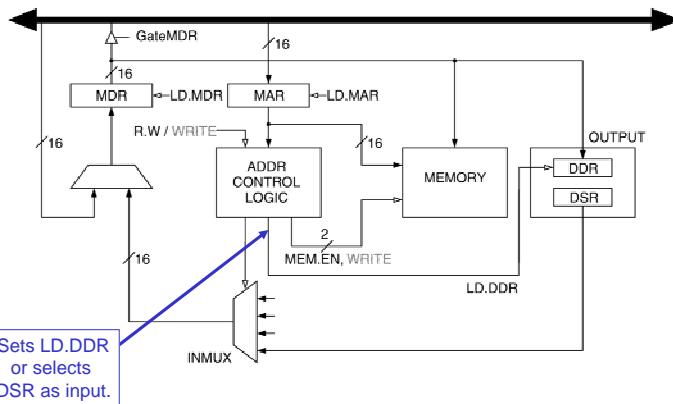
Polling

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

CIT 593

18/32

Simple Implementation: Memory-Mapped Output



CIT 593

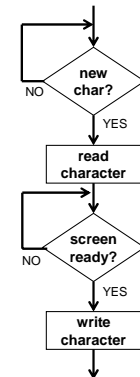
19/32

Keyboard Echo Routine (IN)

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

20/32

General I/O

The interactions described can be applied to most I/O devices

- The status register in a printer might tell you (via computer)
 - If the printer is printing, or
 - Is out of paper, or out of toner

Data/Information that is transferred can be more than just a byte

- Some device have higher data transfer rate
 - E.g. Disk

CIT 593

21/32

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.

CIT 593

22/32

Interrupt-Driven I/O (contd..)

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

23/32

To implement an Interrupt mechanism

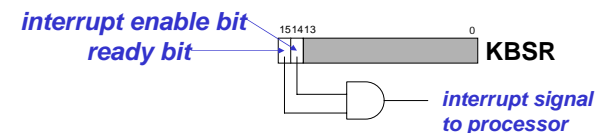
1. Way for I/O device to **signal** CPU that its wants service

- 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

24/32

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) with 7 being higher priority
- 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
 - A special hardware compare priority levels of the interrupts generated, the one with higher priority gets to use the processor

CIT 593

25/32

Maintaining the state of the machine

You were in the middle of adding 2 numbers in your program, but there was interrupt

- Did I complete my operation?
- After I resume, is my data in the registers that I was storing the same as I left of ?

Before FETCH stage in the instruction cycle:

- Check for Interrupt
- If interrupt, then save to memory the state of the machine, i.e. Registers and PC and CCs
 - This way I can come back start executing where I left
- Load the PC with starting address of the program that is to carry out the requirements of the I/O

CIT 593

26/32

Role of 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 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

27/32

Memory Protection

Code in **privileged/supervisor** 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

28/32

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

29/32

Next Time

Detailed working of 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)

CIT 593

30/32

Discussion Questions I/O

1. What is the danger of not testing the KBSR before reading data from the keyboard?
2. Do you think polling is a good approach for other devices, such as a disk or a network interface?
3. Do you think polling is a good approach for other devices, such as a disk or a network interface?

CIT 593

31/32

Next Time

Topic

- Chapter 9

Reading

- Chapter 9

Homework 3

- Assigned due Friday 10/19/07

CIT 593

32/32