ECE 341

Lecture # 19

Instructor: Zeshan Chishti
zeshan@ece.pdx.edu

December 3, 2014

Portland State University

Announcements

- Final exam is on Monday, **December 8** from **5:30** PM to **7:20** PM
  - Similar format and rules as the midterm
  - Approximately 70% coverage will be from the material covered after the midterm
  - Sample final exams from previous terms have been posted on the class website

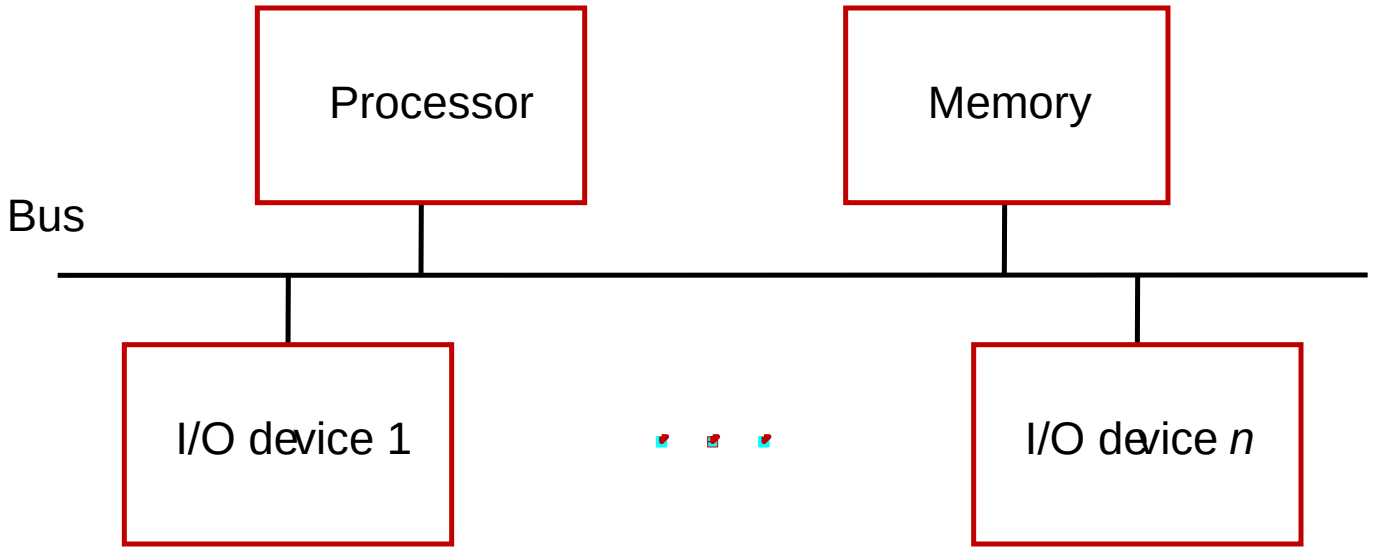
Lecture Topics

- Basic Input/Output
  - Accessing I/O Devices
  - Memory-mapped vs. Port-mapped I/O
  - Program-controlled I/O vs. Interrupts
- Direct Memory Access (DMA)

- Reference:
  - Chapter 3 (Sections 3.1, 3.2.1, 3.2.3)
  - Chapter 8 (Section 8.4)

Basic Input/Output (I/O)

Accessing I/O Devices



- Multiple I/O devices may be connected to the processor &memory via a bus
- Bus consists of three sets of lines (i) address, (ii) data and (iii) control signals
- Each I/O device is assigned a unique address
- To access an I/O device, the processor places the address on the address lines
- The device recognizes the address, and responds to the control signals.

Memory-Mapped vs. Port-Mapped I/O

- Just like memory, each I/O device must appear to the processor as consisting of some addressable locations. There are two methods to assign addresses to I/O devices:

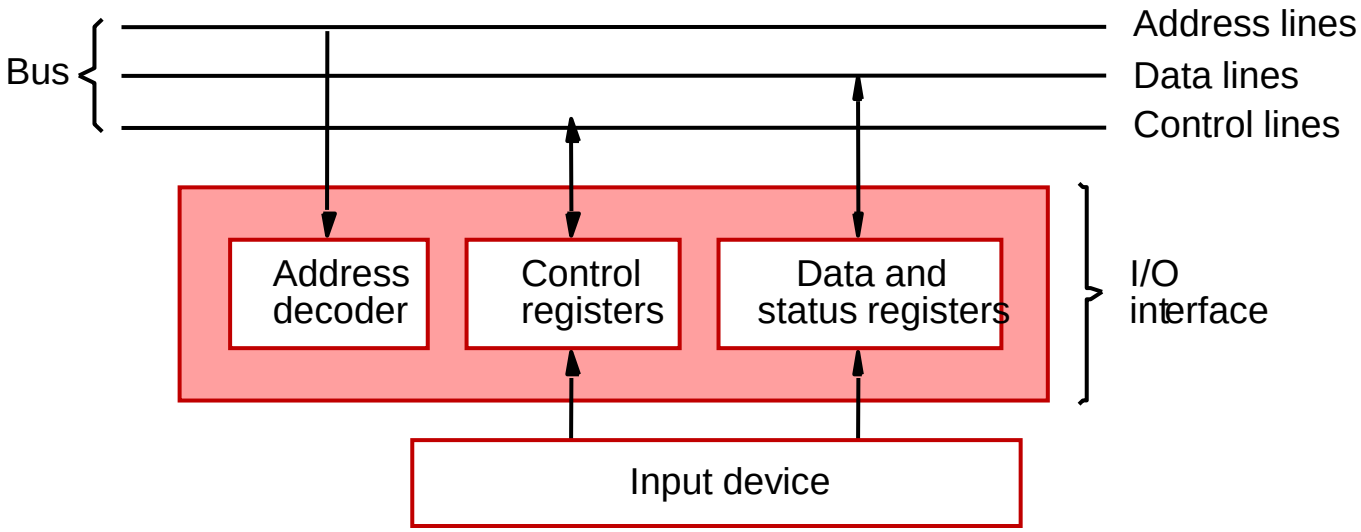
Memory-mapped I/O

- I/O devices and memory share the same address space:
  - Any machine instruction that can access memory can be used to transfer data to or from an I/O device
  - Simpler software

Port-Mapped I/O

- I/O devices and memory may have different address spaces:
  - Special instructions needed to transfer data to and from I/O devices
  - In processors with few address bits, entire address space is available for memory
  - I/O devices need fewer address lines

I/O Device Interface



- I/O device is connected to the bus using an I/O interface circuit
- I/O interface circuit coordinates transfers between the processor and I/O device
  - Address decoder: decodes the address placed on the address lines, thus enabling the device to recognize its address
  - Control register: stores information that controls the behavior of I/O device
  - Data register: serves as a buffer for data transfers
  - Status register: holds information about the current status of the device
- I/O registers are accessed by program instructions as if they are memory locations

Program Controlled I/O (Polling)

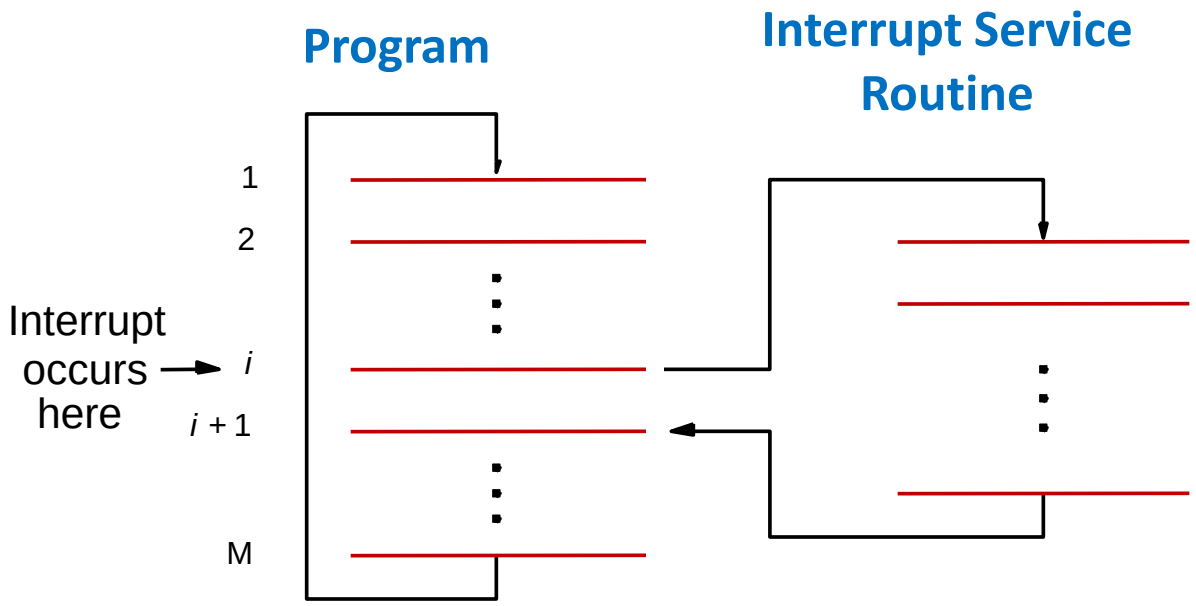
- I/O devices transfer data at a slower rate than the processor speed
- This creates a need to synchronize data transfer between processor and I/O devices
- Two mechanisms used to synchronize data transfer between processor and I/O devices: (i) Program-controlled I/O, (ii) Interrupts

- **Program-controlled I/O:**
  - Processor repeatedly monitors a status flag to check if the I/O device is ready or not
    - For example, a processor sending data to a display needs to check whether the display is ready to receive the next character
  - This is also called "**polling**" the I/O device

Interrupts

- In program-controlled I/O, when the processor continuously “polls” device, it cannot perform any other useful tasks
- An alternate approach would be for the I/O device to alert the processor when it becomes ready
  - Do so by sending a hardware signal called an **interrupt** to the processor
  - At least one of the bus control lines, called an interrupt-request line (IRQ) is dedicated for this purpose
- Processor can perform other useful tasks while it is waiting for the device to be ready

Interrupt Service Routine



- Processor is executing the instruction at address i, when an interrupt occurs
- Routine executed in response to an interrupt is called interrupt-service routine (ISR)
- When an interrupt occurs, control must be transferred to ISR
- Before transferring control, the current PC contents must be saved in a known location
    - This enables the return-from-interrupt instruction to resume execution at instruction i+1
- PC contents are usually stored on the processor stack

Saving and Restoring Registers

- Treatment of an ISR is very similar to that of a regular subroutine
- However there are significant differences:
  - A subroutine performs a task that is required by the calling program
  - ISR may not have anything in common with the program it interrupts
  - ISR and the program that it interrupts may belong to different users
- As a result, before branching to the ISR, not only the PC, but other information such as processor registers used by both the interrupted program and the ISR must be stored
- Saving and restoring information can be done automatically by the processor or explicitly by program instructions
- Most processors save only the minimal amount of information
  - PC and processor status registers
- Any additional registers that need to be saved must be saved explicitly by program instructions

Enabling and Disabling Interrupts

- Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:
  - Sometimes such alterations may be undesirable, and must not be allowed
  - For example, the processor may not want to be interrupted by the same device while executing its ISR
- Processors generally provide the ability to enable and disable such interruptions as desired
- One simple way is to provide machine instructions such as ***Interrupt-enable*** and ***Interrupt-disable*** for this purpose
- To avoid interruption by the same device while executing an ISR:
  - First instruction of an interrupt service routine can be Interrupt-disable
  - Last instruction of an interrupt service routine can be Interrupt-enable

Direct Memory Access (DMA)

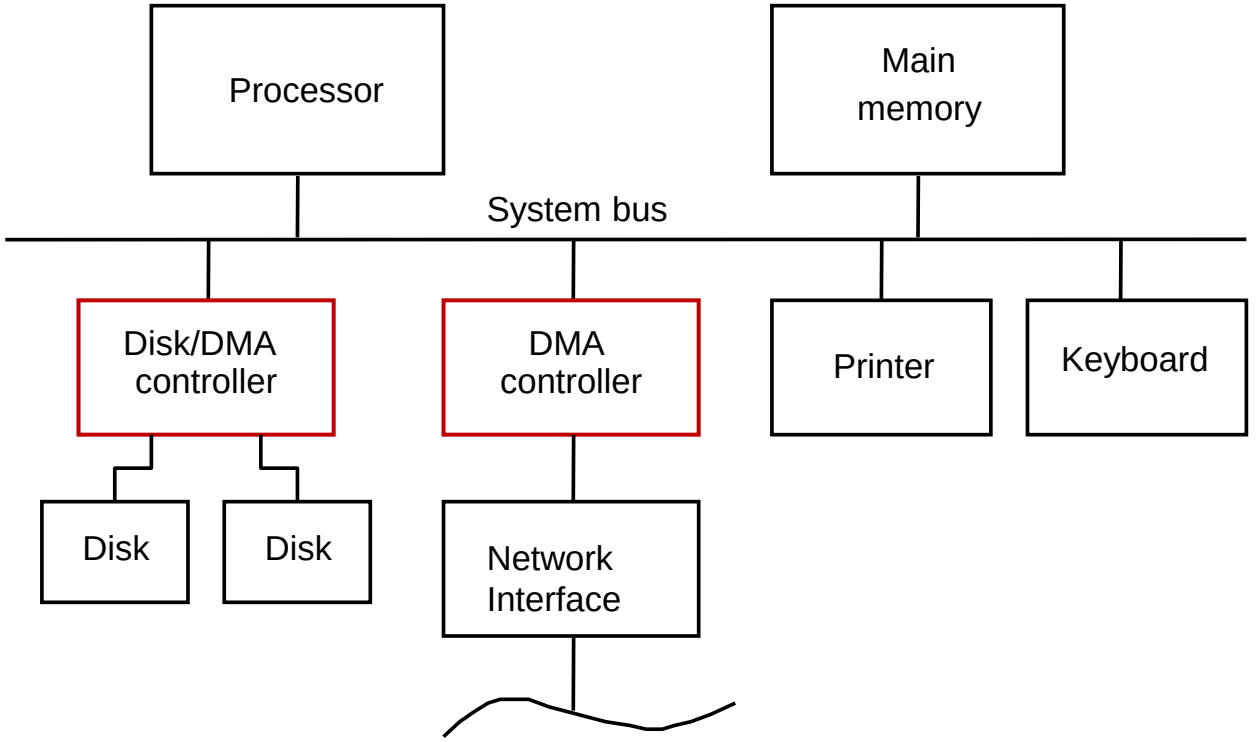
Direct Memory Access

- Blocks of data are often transferred between main memory and I/O devices such as disks
- One option is that the processor controls all these data transfers
  - Transfer happen via processor registers
  - Problem: Large overhead due to frequent interventions by the processor (instructions for incrementing the memory address, keeping track of word count, loading data into processor registers, interrupt service routine etc.)
- Solution: Direct Memory Access (DMA)
  - A special control unit is provided to transfer blocks of data directly between memory and I/O device without needing processor intervention
  - This control unit called "DMA controller" is a part of the I/O device's interface circuit

DMA Controller

- DMA controller performs functions that would normally be carried out by the processor:
  - For each word, it provides the memory address and all the control signals
  - To transfer a block of data, it increments the memory addresses and keeps track of the number of transfers
- The operation of the DRAM controller must be under processor control. That is, the processor must *initiate* the DMA transfer
- To initiate DMA transfer, processor informs the DMA controller about:
  - Starting address
  - Number of words in the block
  - Direction of transfer (I/O device to memory or memory device to I/O)
- After DMA transfer is completed, DMA controller informs the processor by raising an interrupt signal

Example Usage of DMA



- One DMA controller connects a high-speed network to the computer bus
- The second DMA controller (Disk/DMA controller) controls two disks
  - It can perform two independent DMA operations
  - The registers to store the memory address, word count & status & control information are duplicated, so that one set can be used with each disk

Example Usage of DMA (cont.)

Steps for transferring a data block from main memory to the disk

- The OS routine writes the *starting address* and *word count* information into the disk controller registers
- The Disk/DMA controller proceeds independently to implement the specified operation
- When the data transfer is completed, the Disk/DMA controller sets the *Done* bit
- If the *IE* bit is set, the DMA/Disk controller sets the IRQ bit to interrupt the processor