Chapter 6
Input/Output
Input/Output Problems

- Wide variety of peripherals
  - Human readable (screen, printer, keyboard, …)
  - Machine readable (storage, communication, …)
  - Delivering different amounts of data
  - At different speeds
  - In different formats

- All slower than CPU and RAM

- To keep CPU simple I/O modules are needed to proper interface peripherals and CPU/RAM
Input/Output Module

- Interface to CPU and Memory
- Interface to one or more peripheral devices
A peripheral (abstract view)

I/O Module

Control signals → State signals → Data bits

Control Logic → Buffer → Transducer

Data bits

Physical medium
I/O Module Function

- Control & Timing
- CPU Communication (command, data, status, address)
- Device Communication
- Data Buffering (to compensate different speeds)
- Error Detection (storage, transmission, …)
I/O Module Diagram

Data Registers
Status/Control Registers

I/O Logic

External Device Interface Logic
External Device Interface Logic

Data Lines
Address Lines
Control Lines

Interface to System Bus
Interface to External Device

Data
Status
Control

Data
Status
Control
I/O Steps

• CPU checks I/O module device status
• I/O module returns status
• If ready, CPU requests data transfer
• I/O module gets data from device
• I/O module transfers data to CPU
• Variations for output, DMA, etc.
Input Output Techniques

- Programmed
  - CPU directly control I/O operation
- Interrupt driven
- Direct Memory Access (DMA)
  - No CPU involvement
I/O Techniques

(a) Programmed I/O

(b) Interrupt-driven I/O

(c) Direct memory access
Programmed I/O

- CPU has direct control over I/O
  - Sensing status
  - Read/write commands
  - Transferring data
- CPU waits for I/O module to complete operation
- Wastes CPU time
Programmed I/O - detail

- CPU requests I/O operation
- I/O module performs operation
- I/O module sets status bits
- CPU checks status bits periodically
  - I/O module does not inform CPU directly
  - I/O module does not interrupt CPU
  - CPU may wait or come back later
Programmed I/O - Commands

- CPU issues command
  - Control - telling module what to do
    - e.g. spin up disk, move head
  - Test - check status
    - e.g. power failure? read error? data ready?
  - Read/Write
    - Module transfers data via buffer from/to device
Programmed I/O - Addressing Devices

- Under programmed I/O, data transfer is very like memory access (CPU viewpoint)
- Each device is given a unique identifier (i.e., memory address)
- CPU commands contain address
  - Identifies module (and device if there is more than one per module)
I/O Mapping

• Memory mapped I/O
  ▪ Devices and memory share an address space
  ▪ I/O looks just like memory read/write
  ▪ No special commands for I/O
    • Large selection of memory access commands available

• Isolated I/O
  ▪ Separate address spaces
  ▪ Need I/O or memory select lines
  ▪ Special commands for I/O
    • Limited set
Interrupt Driven I/O

- The biggest problem of programmed I/O is CPU waste of time in waiting for data to be read/written or checking status of I/O module
- Solution:
  - CPU issues commands to device and continues with other activities
  - No waiting time for CPU
  - No repeated CPU checking of device
  - I/O module **interrupts** when ready
Interrupt Driven I/O
Basic Operation

• CPU issues read command to I/O module
• I/O module gets data from the peripheral while CPU does other work
• When data have been received I/O module interrupts CPU
• CPU requests data to I/O module
• I/O module transfers data to CPU
Interrupt processing

Hardware

Device controller or other system hardware issues an interrupt

Processor finishes execution of current instruction

Processor signals acknowledgment of interrupt

Processor pushes PSW and PC onto control stack

Processor loads new PC value based on interrupt

Software

Save remainder of process state information

Process interrupt

Restore process state information

Restore old PSW and PC
CPU Viewpoint

- Issue read command
- Do other work
- Check for interrupt at end of each instruction cycle
- If interrupted:
  - Save context (registers)
  - Serve the interrupt signal
    - Proper interrupt routine: fetch data & store
Interrupt Processing (servicing)
Interrupt Processing (return)
Design Issues

• How do you identify the module issuing the interrupt?

• How do you deal with multiple interrupts?
  ▪ i.e. an interrupt handler being interrupted
Identifying Interrupting Module (1)

- Different interrupt line for each module
  - Simple
  - Limits number of devices
- Software poll
  - CPU asks each module in turn
  - Slow and time wasting
Identifying Interrupting Module (2)

• Daisy Chain or Hardware poll
  ▪ Interrupt Acknowledge (ACK) is sent down a line connecting all devices
  ▪ ACK goes from a module to the next on the line until the responsible module is found, which places a word (vector) on data bus
  ▪ CPU uses the vector to identify proper handler routine

• Bus Master
  ▪ Module must claim the bus before it can raise interrupt
Dealing with Multiple Interrupts

- Each interrupt line has a priority
- Higher priority lines can interrupt lower priority lines
- If bus mastering only current master can interrupt
Direct Memory Access (DMA)

- Interrupt driven and programmed I/O require active CPU intervention
  - CPU is tied up with transferring data in and out
  - Transfer rate is limited since CPU is not fully serving the device
- DMA is the answer
  - Additional Module (hardware) on bus
  - DMA controller takes over CPU for I/O
DMA Operation

- CPU tells DMA controller
  - Read/Write
  - Device address
  - Starting address of memory block for data
  - Amount of data to be transferred
- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished
DMA Cycle Stealing

- DMA controller takes control over system bus for one (or more) clock cycle(s)
- One word of data is transferred for each stolen cycle
- Not an interrupt
  - CPU does not switch context
- CPU is suspended just before it accesses bus
  - i.e. before an operand or data fetch or a data write
- Slows down CPU but not as much as CPU doing transfer
DMA Configurations (1)

- Single Bus, Detached DMA controller
- Each transfer uses bus twice
  - I/O ↔ DMA and DMA ↔ memory
- CPU is suspended twice
DMA Configurations (2)

- Single Bus, Integrated DMA controller
- Controller may support more than one device
- Each transfer uses bus once
  - DMA ↔ memory
- CPU is suspended once per transfer
DMA Configurations (3)

- Separate I/O Bus
- Bus supports all DMA enabled devices
- Each transfer uses bus once
  - DMA ↔ memory
- CPU is suspended once per transfer
I/O Channels

• I/O devices getting more sophisticated
  ▪ e.g. 3D graphics cards
• CPU instructs I/O controller to do transfer
• I/O controller does entire transfer
• I/O controller needs more processing power (is a small CPU, called I/O channel or processor)
• Improves overall system speed, since takes load off CPU
Interface to external devices

- Serial (1 bit at a time) or parallel (1 word at a time)
- Speed
- e.g.: SCSI, USB, FireWire
Small Computer Systems Interface (SCSI)

- Parallel interface (8, 16, 32 bit data lines)
- Daisy chained, but devices are independent
- Chain must be terminated at each end
  - Usually one end is host adapter
  - Plug in terminator or switch(es)
- Devices can communicate with each other as well as host
- SCSI-1, 1980, 8 bit, 5 MHz, 5MB/s, 7 devices
- SCSI-2, 1991, 16/32 bit, 10 MHz, 20/40MB/s
- Ultra-SCSI
USB - Universal Serial Bus

• A single bus for all desktop devices (keyboard, mouse, parallel, RS-232, …), up to 127 devices
• Serial transmission, from 1.5 (low-speed) - 12 Mb/s (high-speed) of USB-1 to 480 Mb/s of USB-2
• Hierarchical topology, protocol, and cables
• “Hot” connection of devices (no need to turn power off) and automatic configuration
IEEE 1394 FireWire

- High performance serial bus
- Fast, low cost, and easy to implement
- Also being used in digital cameras, VCRs and TV
- Daisy chain up to 63 devices
- Automatic configuration (no terminators) and tree-topologies are possible
- Data rates from 25 to 400 Mb/s