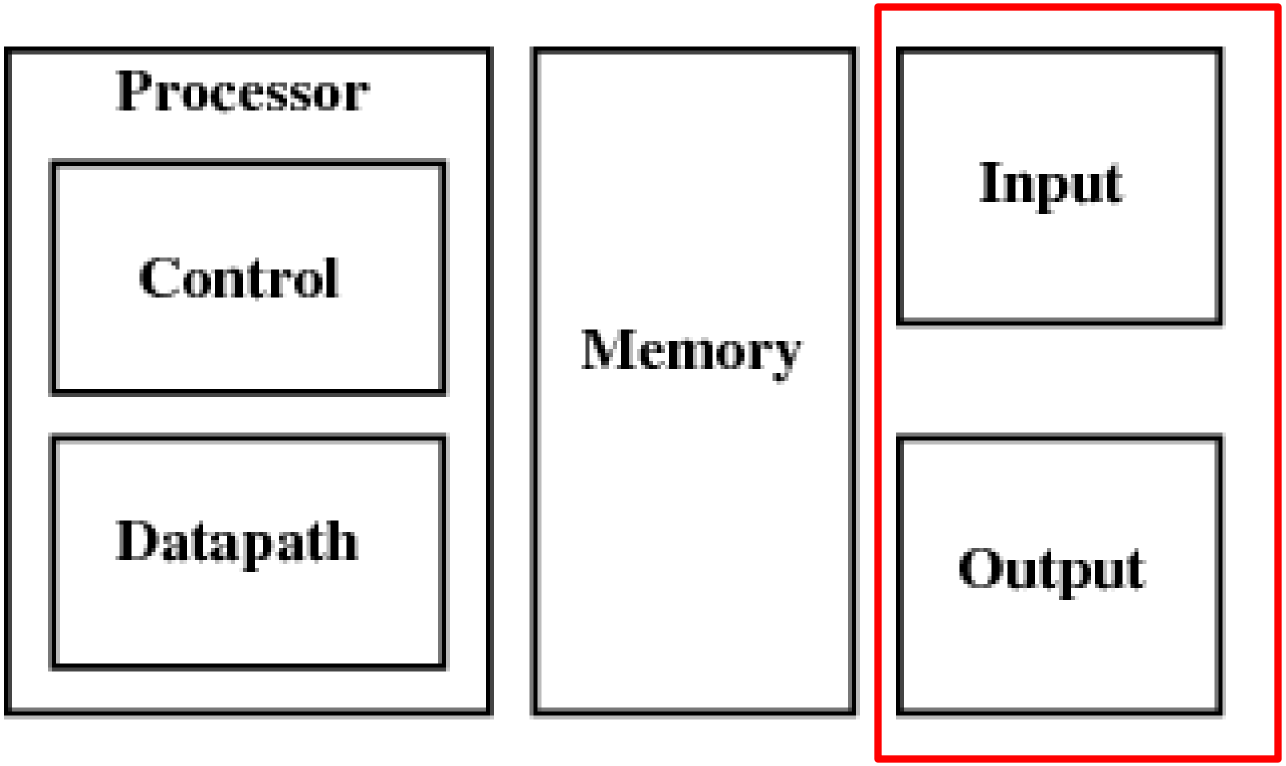
ACK: Some of these slides are based on Stallings

COMPUTER SYSTEMS ORGANIZATION

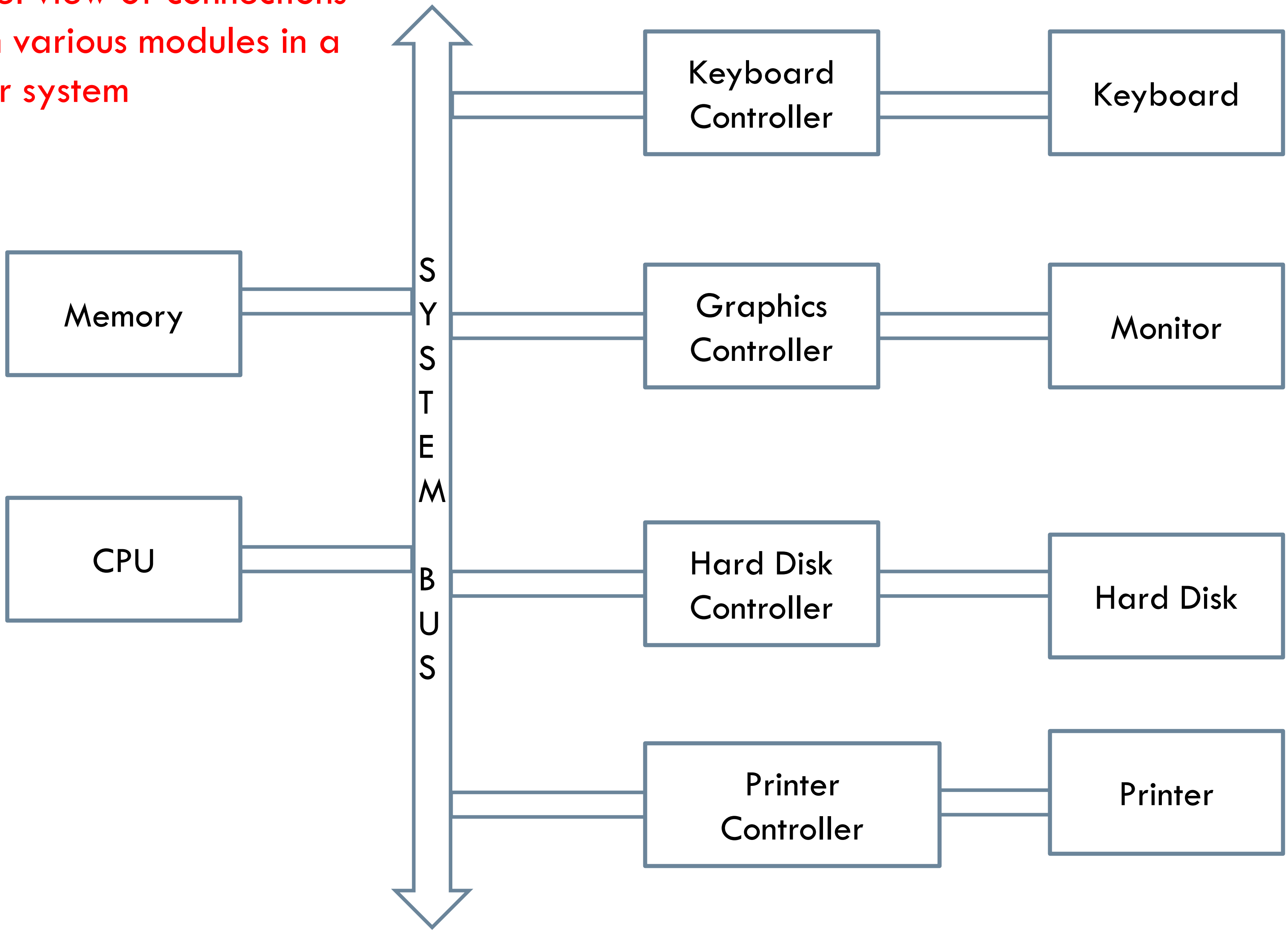
Input/Output -- Spring 2012 -- IIIT-H -- Suresh Purini

The Big Picture: Where are we now?

Input and Output

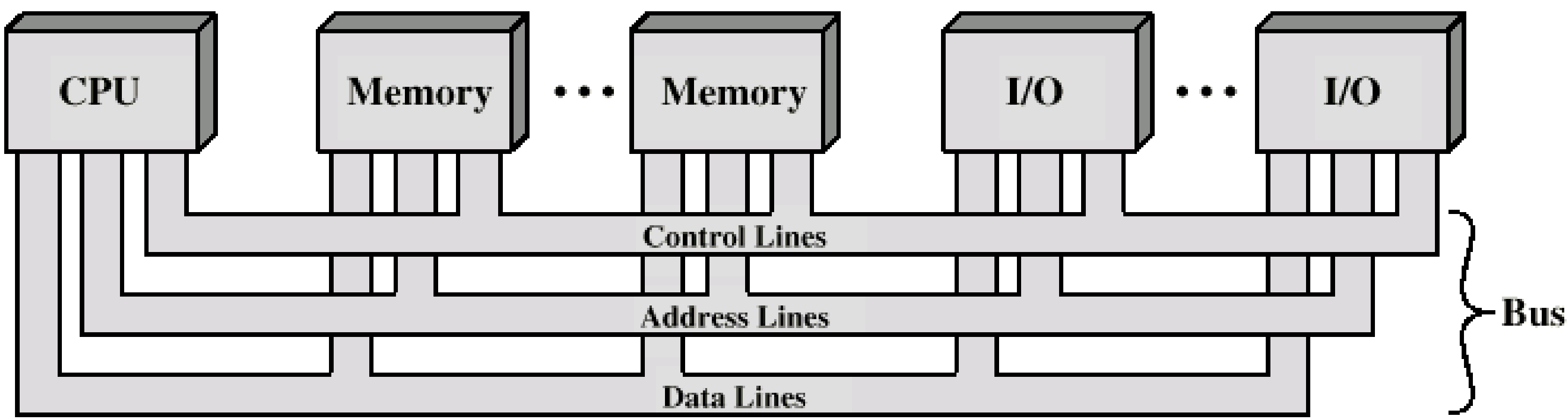


High level view of connections between various modules in a computer system



System Bus Consists of:

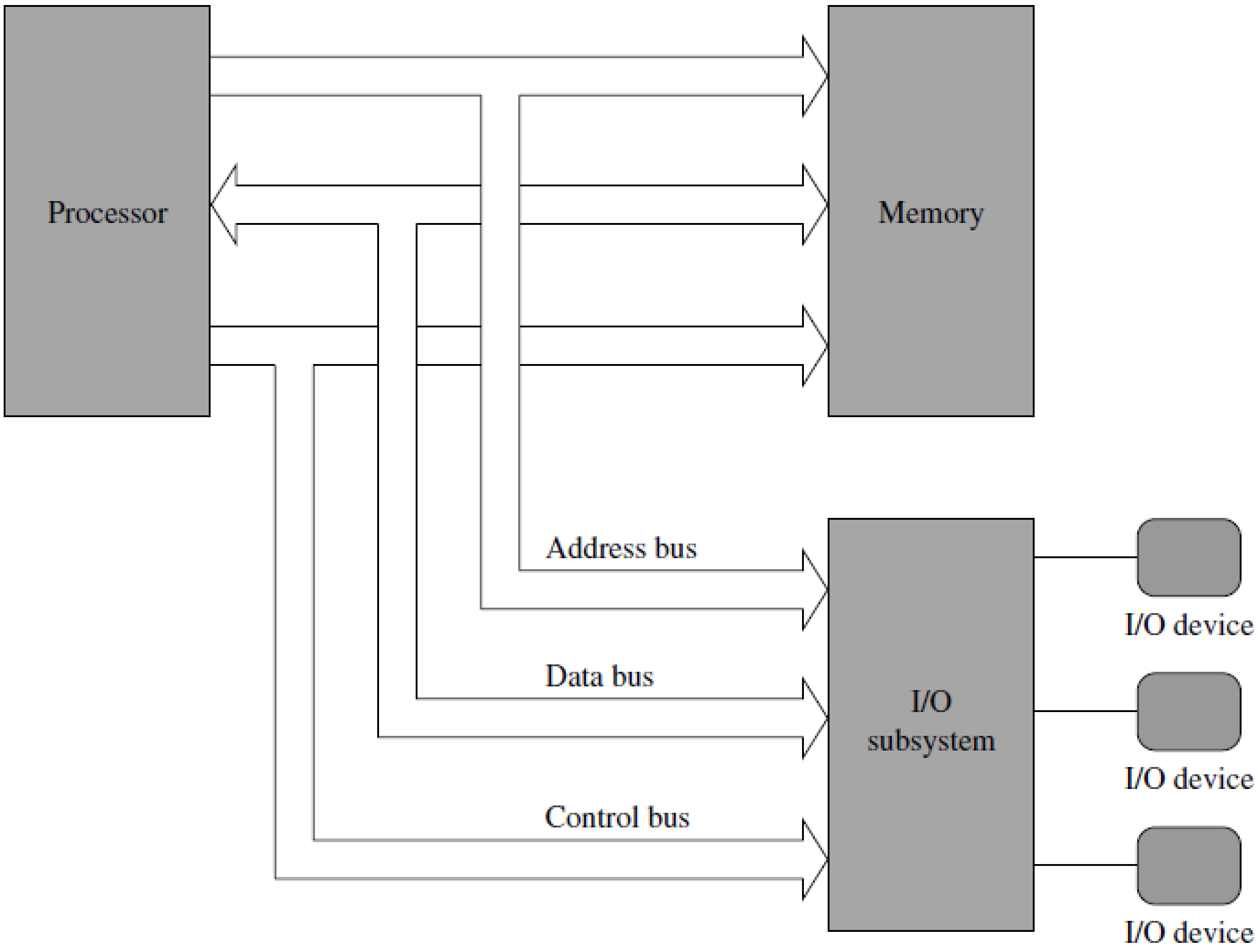
- Address lines
- Data lines
- Control lines

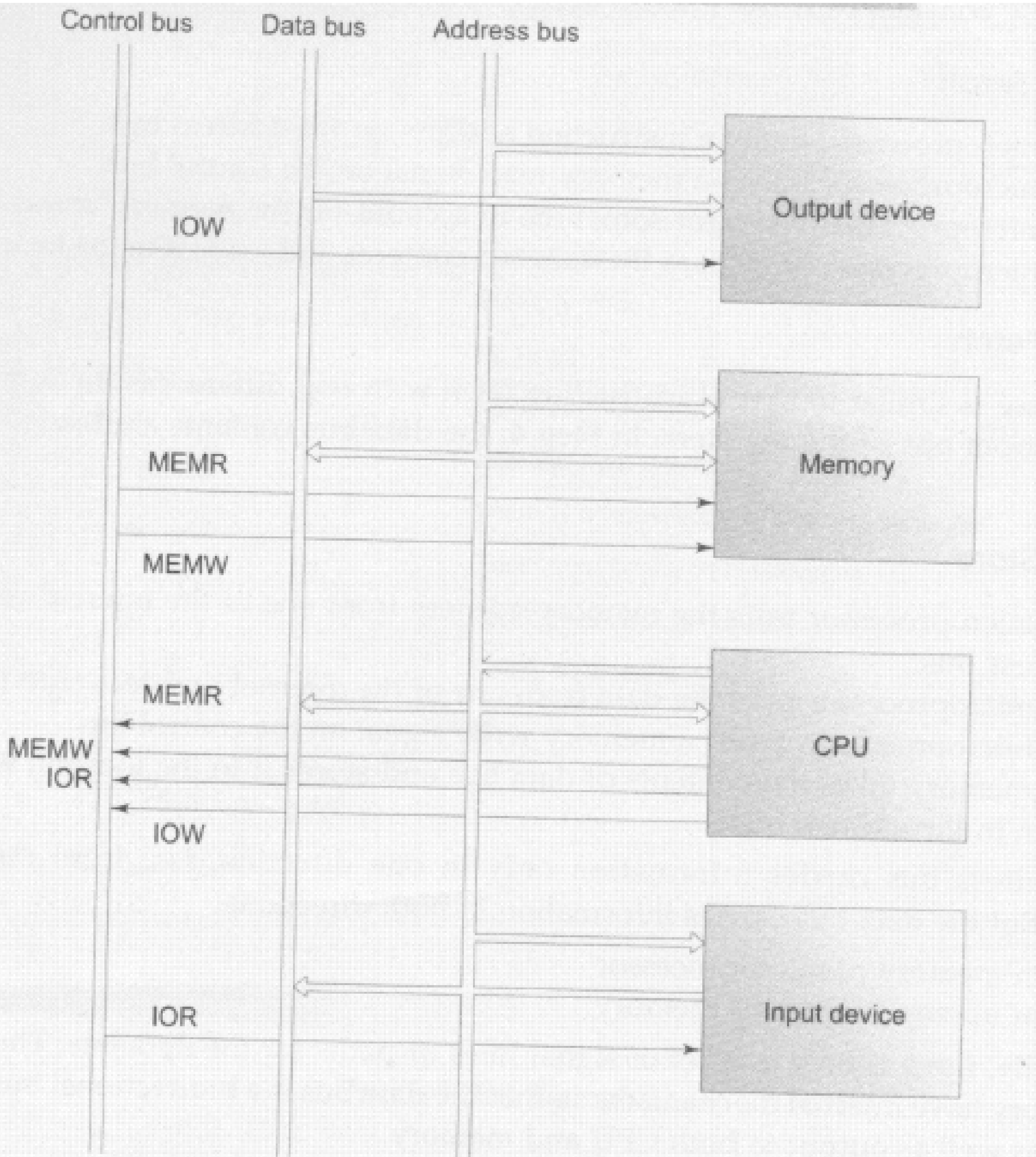


Typical Control Lines:

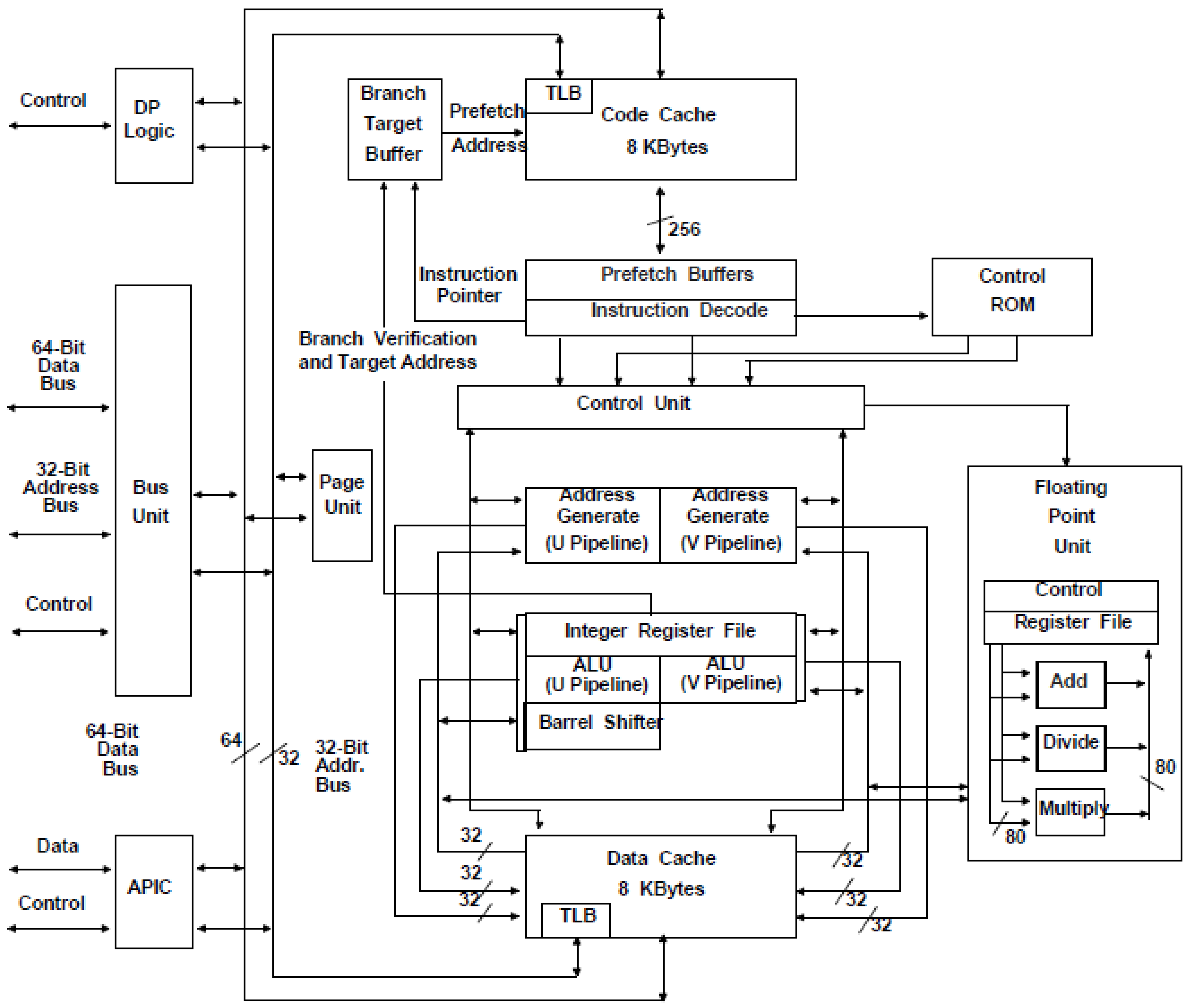
- Memory write
- Memory read
- I/O write
- I/O read
- Transfer ACK
- Bus request
- Bus grant
- Interrupt request
- Interrupt ACK
- Clock

Basic Components of a Computer and their Interconnection





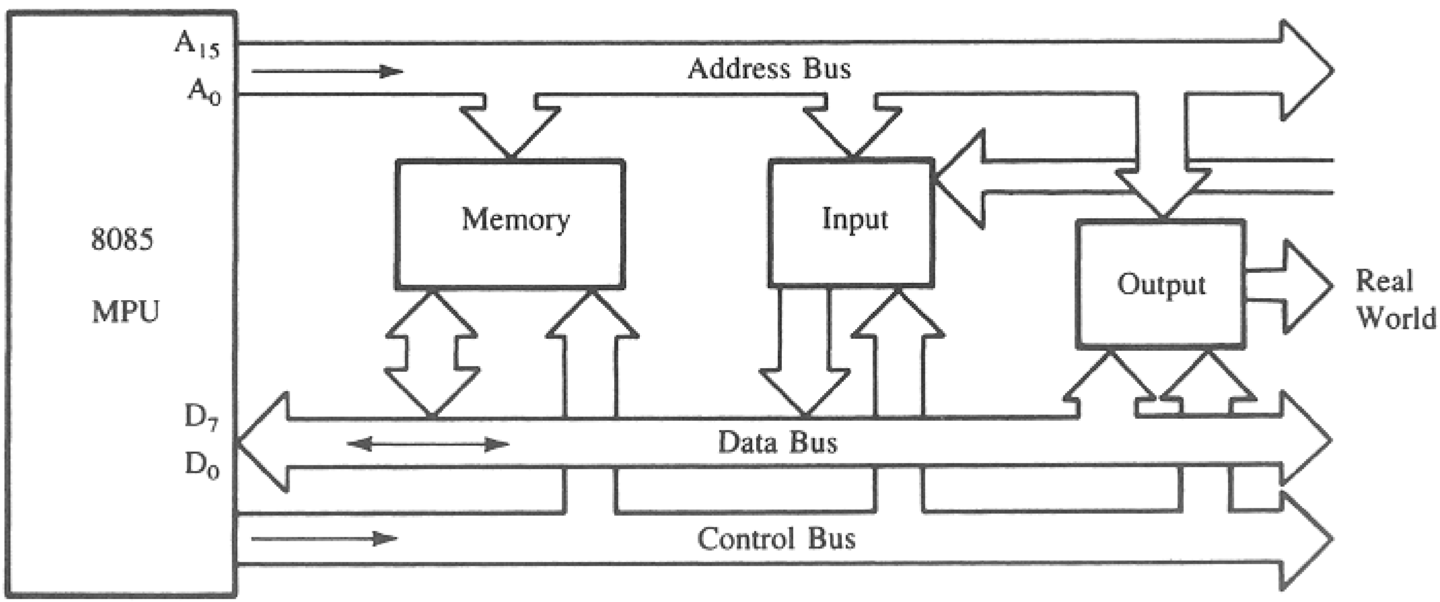
MEMR : Memory Read
IOR : I/0 Read
MEMW : Memory Write
IOW : I/0 Write
Not all control bus signals are shown



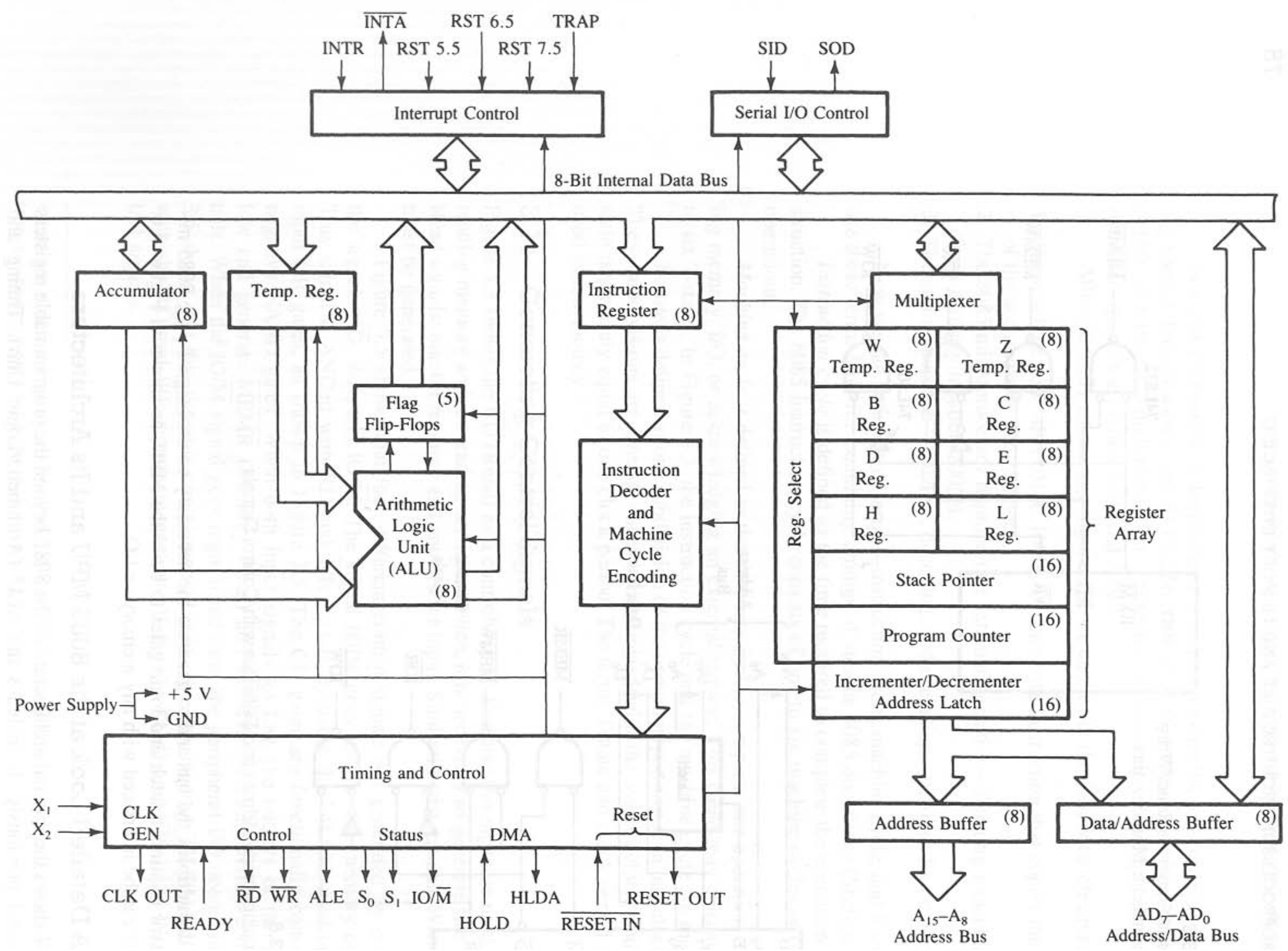
Control
DP Logic
Branch Target Buffer
Prefetch Address
TLB
Code Cache 8 KBytes
256
Instruction Pointer
Prefetch Buffers
Instruction Decode
Control ROM
Branch Verification and Target Address
Control Unit
64-Bit Data Bus
32-Bit Address Bus
Bus Unit
Control
Page Unit
Address Generate (U Pipeline)
Address Generate (V Pipeline)
Floating Point Unit
Control
Register File
Integer Register File
ALU (U Pipeline)
ALU (V Pipeline)
Barrel Shifter
Add
Divide
Multiply
80
80
64-Bit Data Bus
64
32
32-Bit Addr. Bus
Data
Control
APIC
32
32
32
Data Cache 8 KBytes
TLB
32
32
32

The 8085 Bus Structure

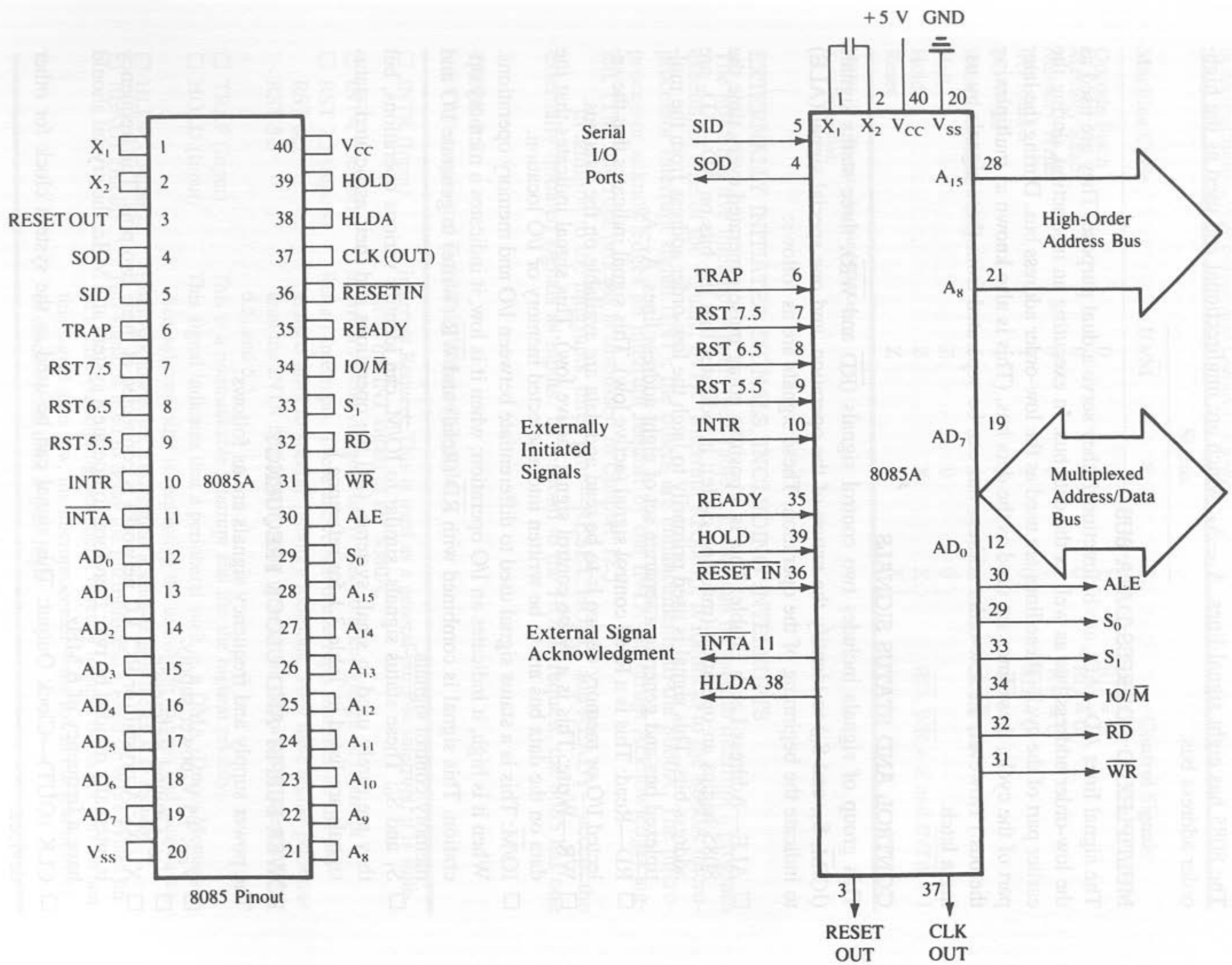
The 8-bit 8085 CPU (or MPU – Micro Processing Unit) communicates with the other units using a 16-bit address bus, an 8-bit data bus and a control bus.



8085 Functional Block Diagram



The 8085 Microprocessor

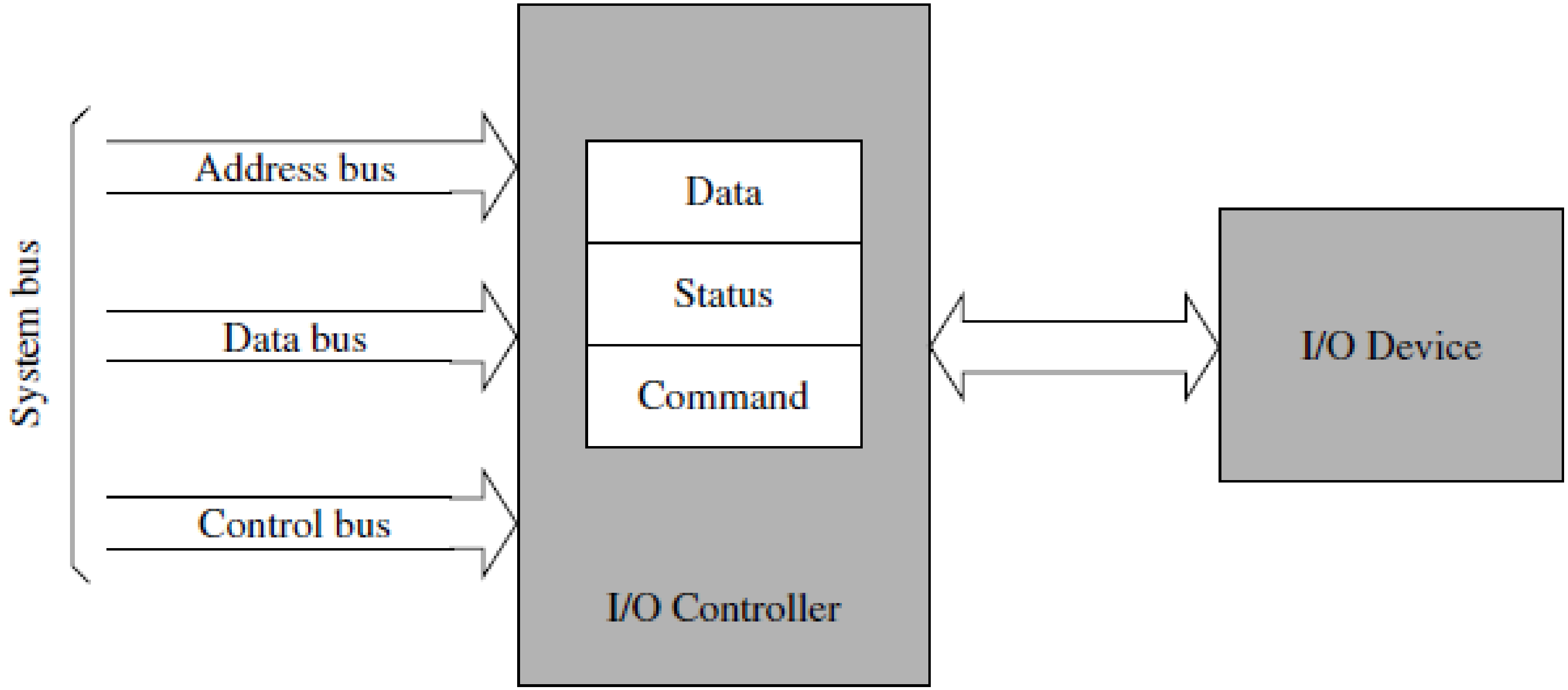


Input/Output Problems

- Wide variety of peripherals
  - Delivering different amounts of data
  - At different speeds
  - In different formats
- Could be slower/faster than CPU and/or RAM
- Need I/O modules (with their Device Drivers and the corresponding Operating System support)

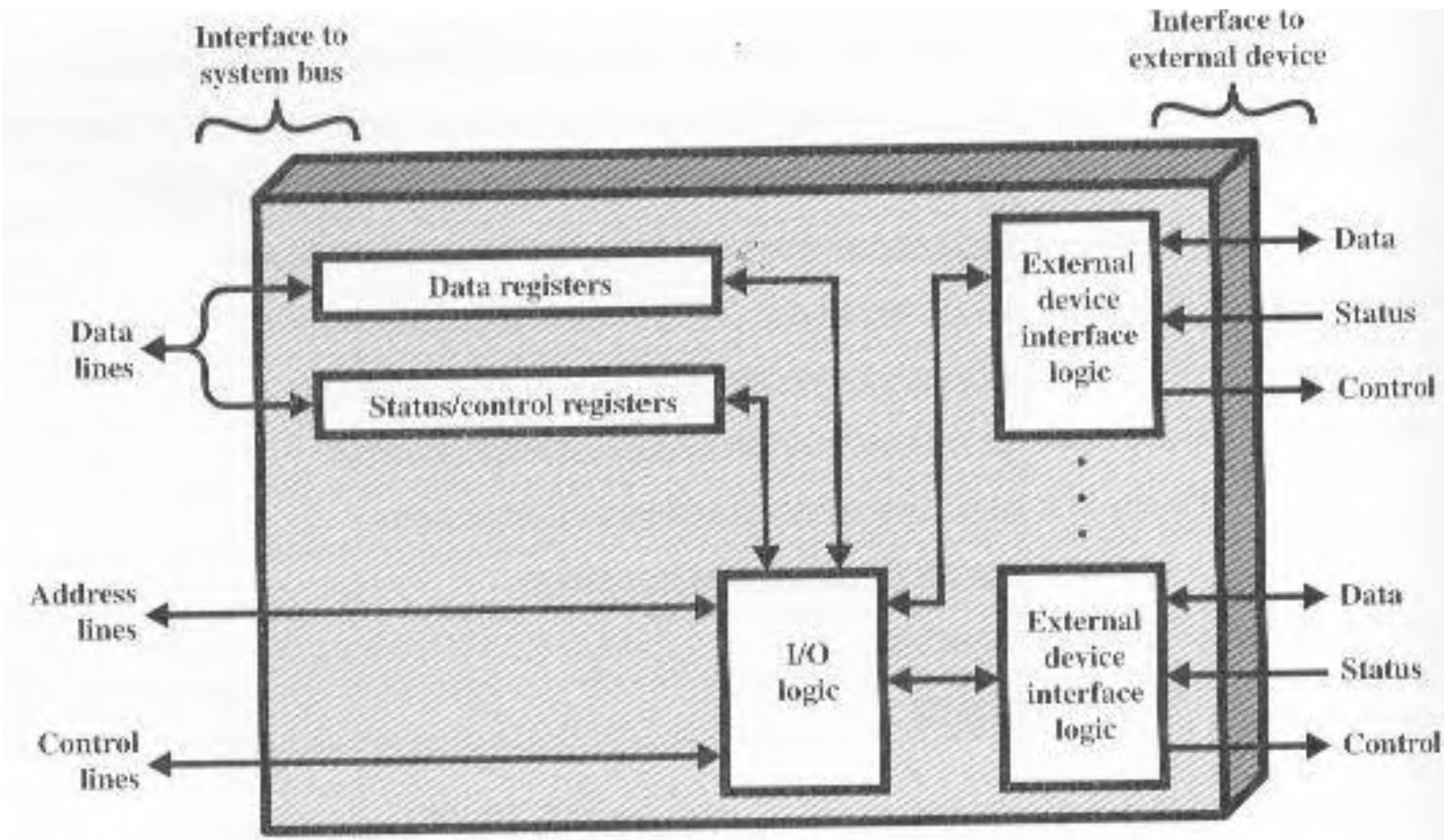
I/O Device Interface

- Interface to CPU and Memory
- Interface to one or more peripherals

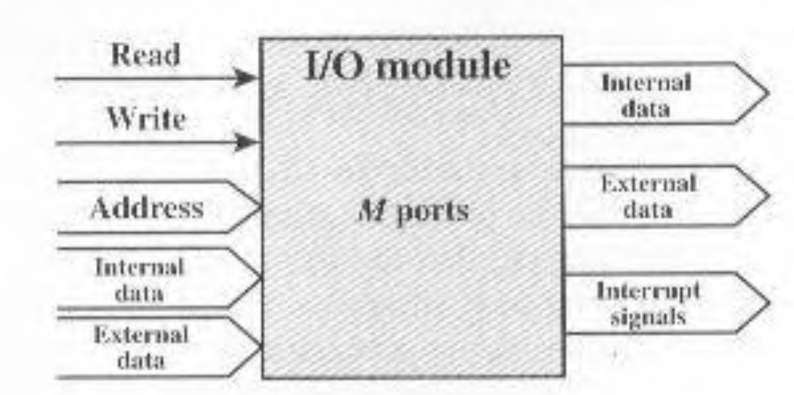


I/O Device Interface

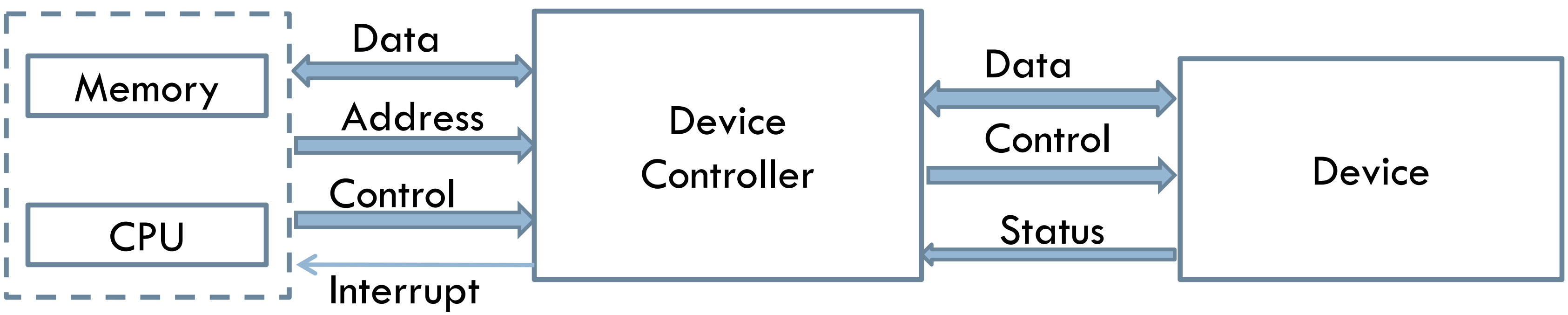
- Interface to CPU and Memory
- Interface to one or more peripherals



Typical Signal Lines for I/O Modules (Device Controllers)

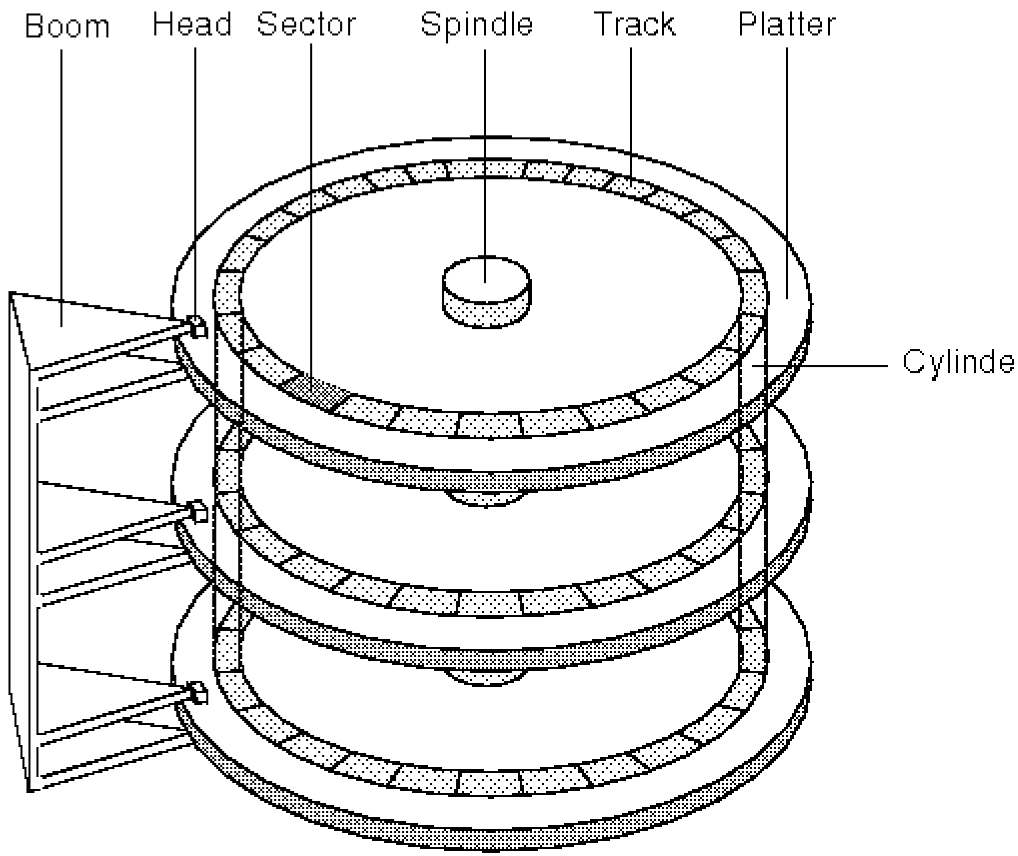


We need a device drive which knows how to interface with the Device Controller

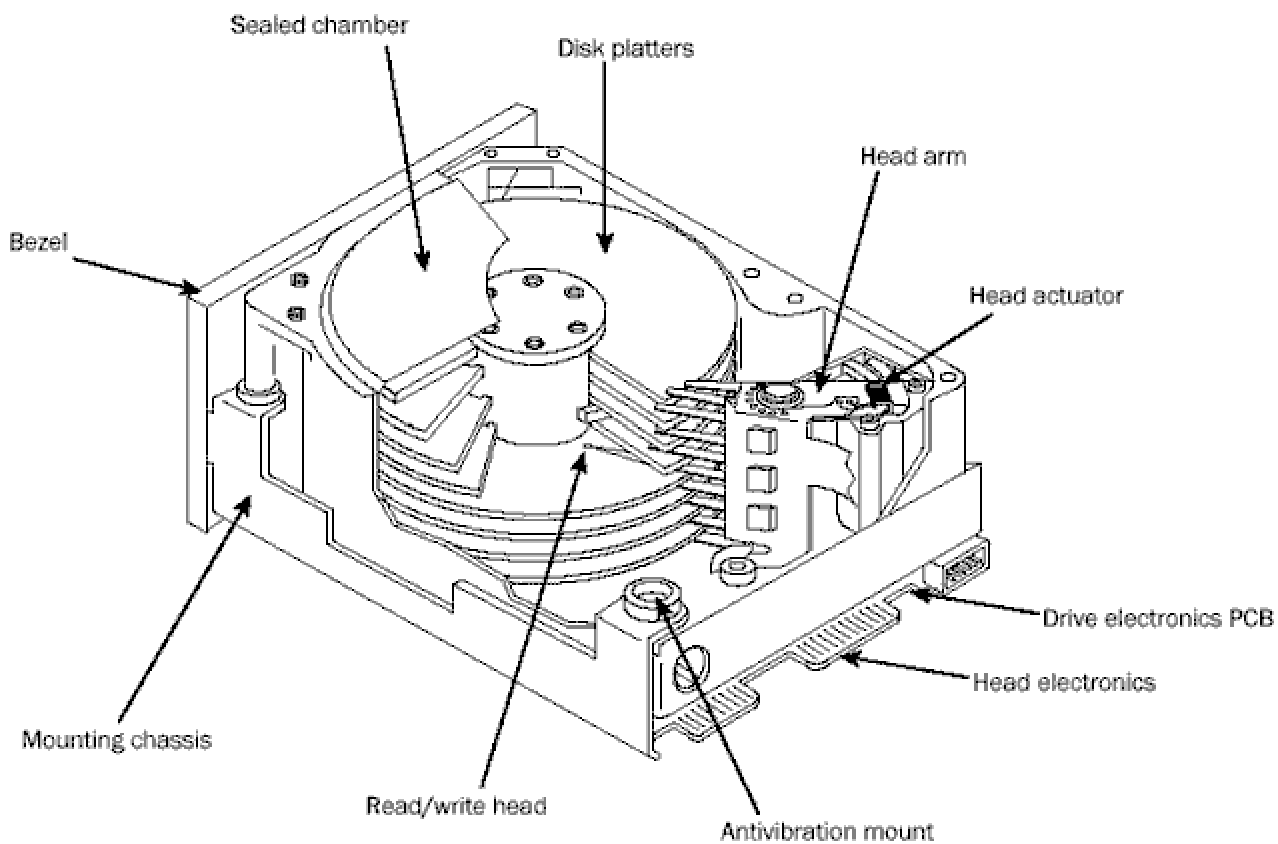


I/O Module Functions

- Control & Timing
- CPU Communication
- Device Communication
- Data Buffering
- Error Detection

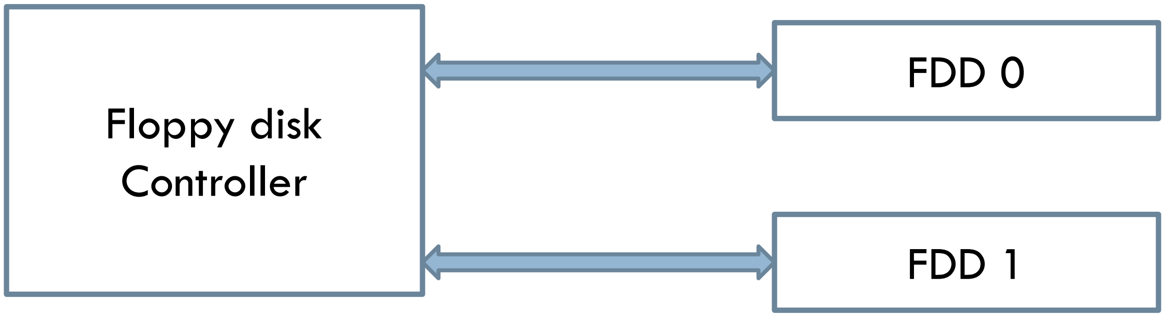


Boom
Head
Sector
Spindle
Track
Platter
Cylinder



Sealed chamber
Disk platters
Head arm
Bezel
Head actuator
Drive electronics PCB
Head electronics
Mounting chassis
Read/write head
Antivibration mount

Floppy Disk Controller



Control Signal	Action by the device
Drive Select	FDD gets logically connected
Head Select	FDD selects either the top or bottom head
Direction	Indicate the direction of head movement (inward or outward)
Step	Move one track
………………	………

Status Signal	Action by the device
Track 0	Read/write head is positioned over the track 0
Write Protect	FDD is write protected
Ready	FDD is ready for operation
………………	………

Sample Code – 1

C Code to Read Character

```
#define KBDataRegAddr 0x11
#define TTYDataRegAddr 0x12
#define STATUSRegAddr 0x13

unsigned char ch, status;

status = inb(STATUSRegAddr);
while ((status & 0x80) == 0) {
    status = inb(STATUSRegAddr); /* do nothing */
}

ch = inb(KBDataRegAddr);
```

Sample Code – 1

C Code Explanation (Read)

- **First three lines set up the constants to represent the addresses of the registers**
- **The** `inb()` **command used to read in the value of the STATUS register**
- **"Busy-wait" Loop**
  - Mask off all bits of the STATUS register except for C_{in} (bit 7)
  - Tests the resulting byte for zero or non-zero
  - Forces program to be busy while waiting for input

Sample Code – 2

C Code to Write Character

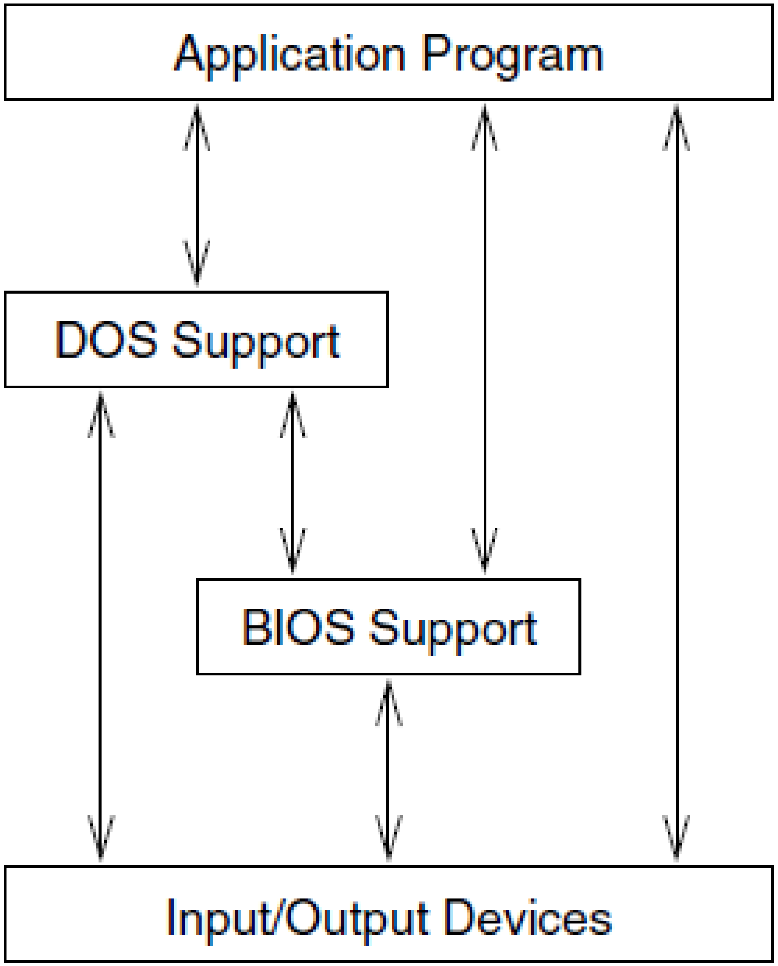
```
#define KBDataRegAddr 0x11
#define TTYDataRegAddr 0x12
#define STATUSRegAddr 0x13

unsigned char ch, status;

status = inb(STATUSRegAddr);
while ((status & 0x40) != 0) {
    status = inb(STATUSRegAddr); /* do nothing */
}

outb(ch, TTYDataRegAddr);
```

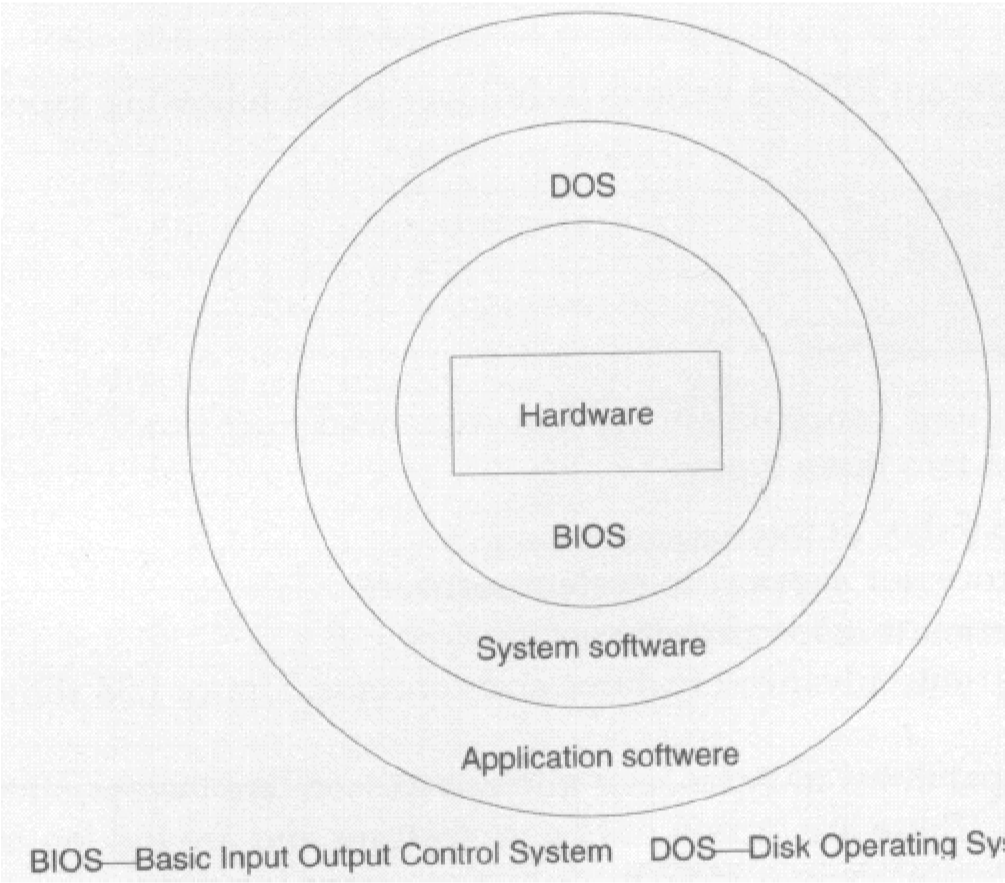
Sample Code – 2

C Code Explanation (Write)

- **The `inb()` command again used to read in the value of the STATUS register**
- **Mask off all bits of the STATUS register except for C_{out} (bit 6)**
- `while` **statement polls the C_{out} bit of STATUS**
  - Insure the most recently written character has been displayed on the TTY device
  - Once C_{out} changes to 0, the condition will fail and program will proceed after `while` loop to write the character

Interacting with I/O Devices





Input Output Techniques

- Programmed I/O
- Interrupt driven
- Direct Memory Access (DMA)

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.

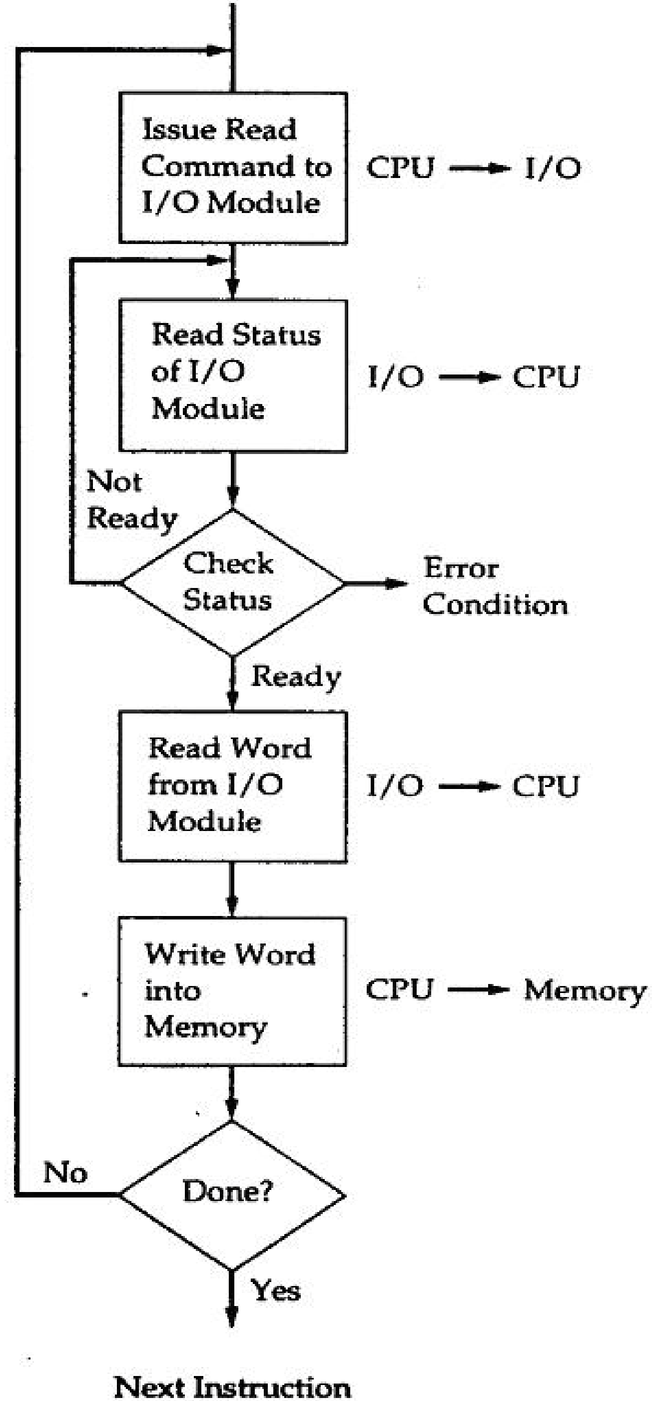
Programmed I/O

- 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



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 (I/O Mapped I/O)
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O
    - Limited set

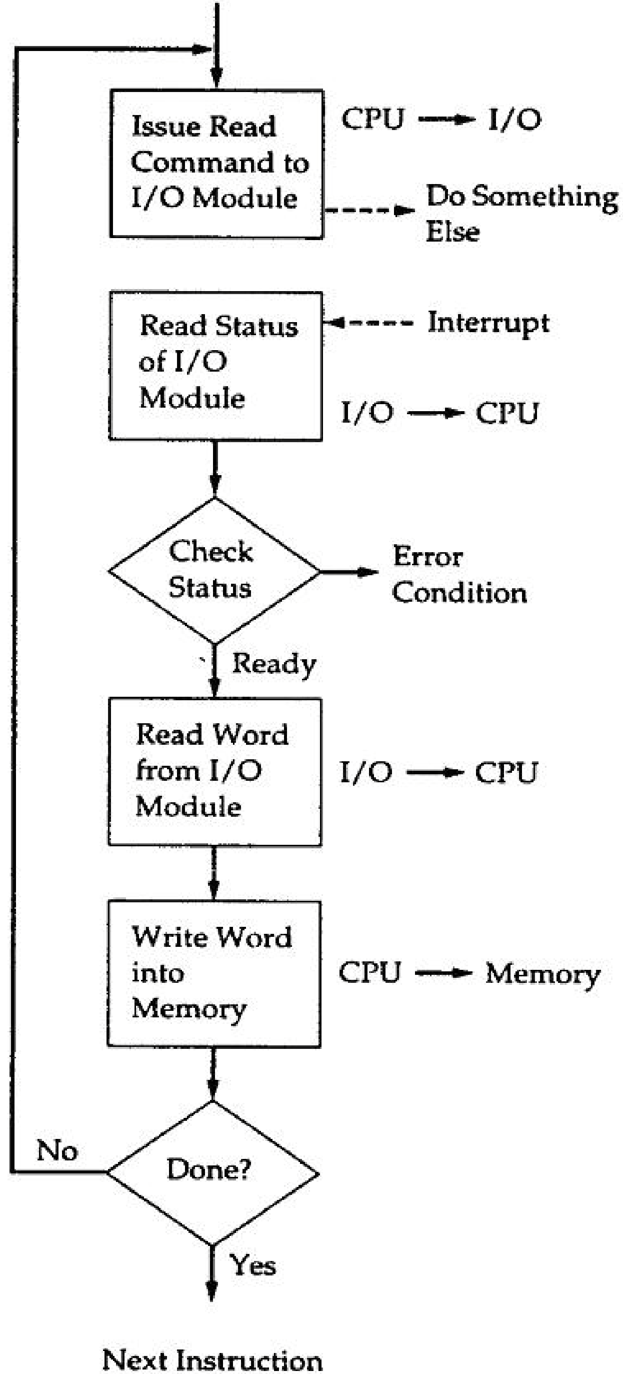
Interrupt Driven I/O

- Overcomes CPU waiting
- No repeated CPU checking of device
- I/O module interrupts when ready

Interrupt Driven I/O Basic Operation

- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

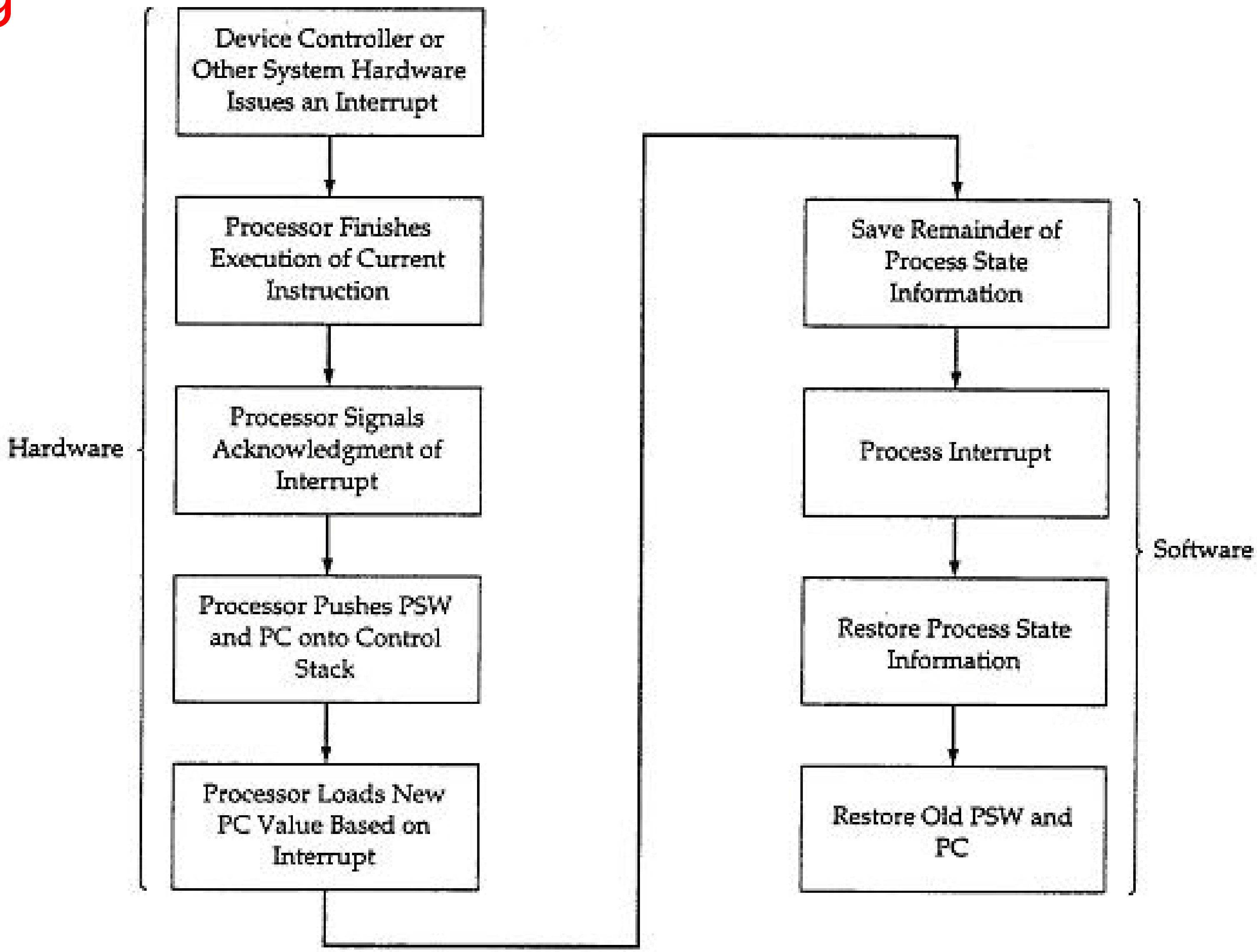
Interrupt Driven I/O



CPU Viewpoint

- Issue read command
- Do other work
- Check for interrupt at end of each instruction cycle
- If interrupted:-
  - Save context (registers)
  - Process interrupt
    - Fetch data & store

Interrupt Processing



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

- Different line for each module
  - PC
  - Limits number of devices
- Software poll
  - CPU asks each module in turn
  - Slow

Software Poll

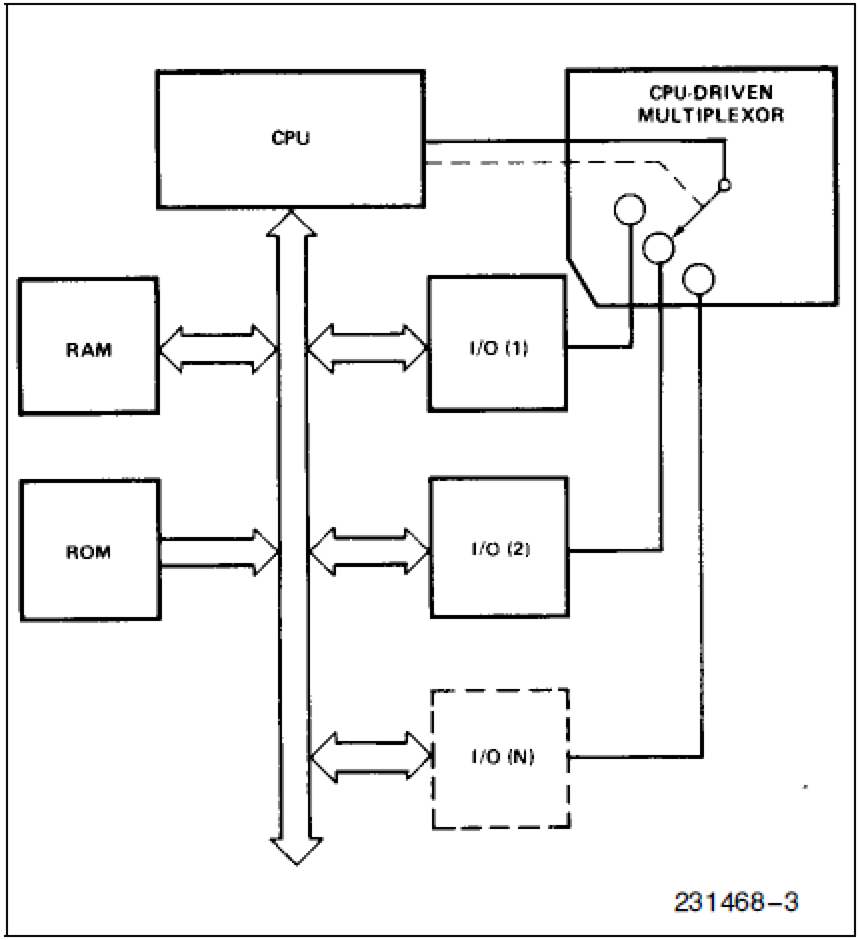
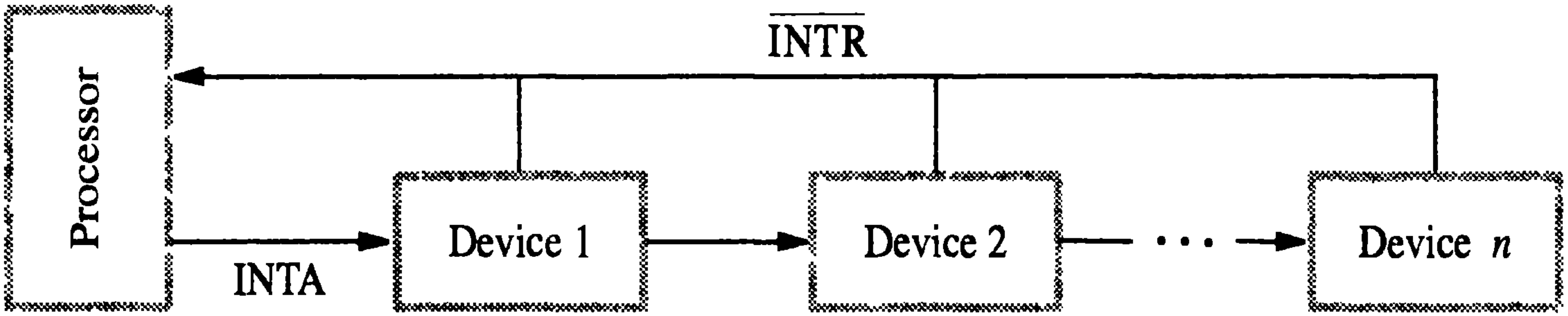


Figure 3a. Polled Method

Identifying Interrupting Module

- Daisy Chain or Hardware poll
  - Interrupt Acknowledge sent down a chain
  - Module responsible places vector on bus
  - CPU uses vector to identify handler routine



- Bus Master
  - Module must claim the bus before it can raise interrupt

Interrupt Processing in x86

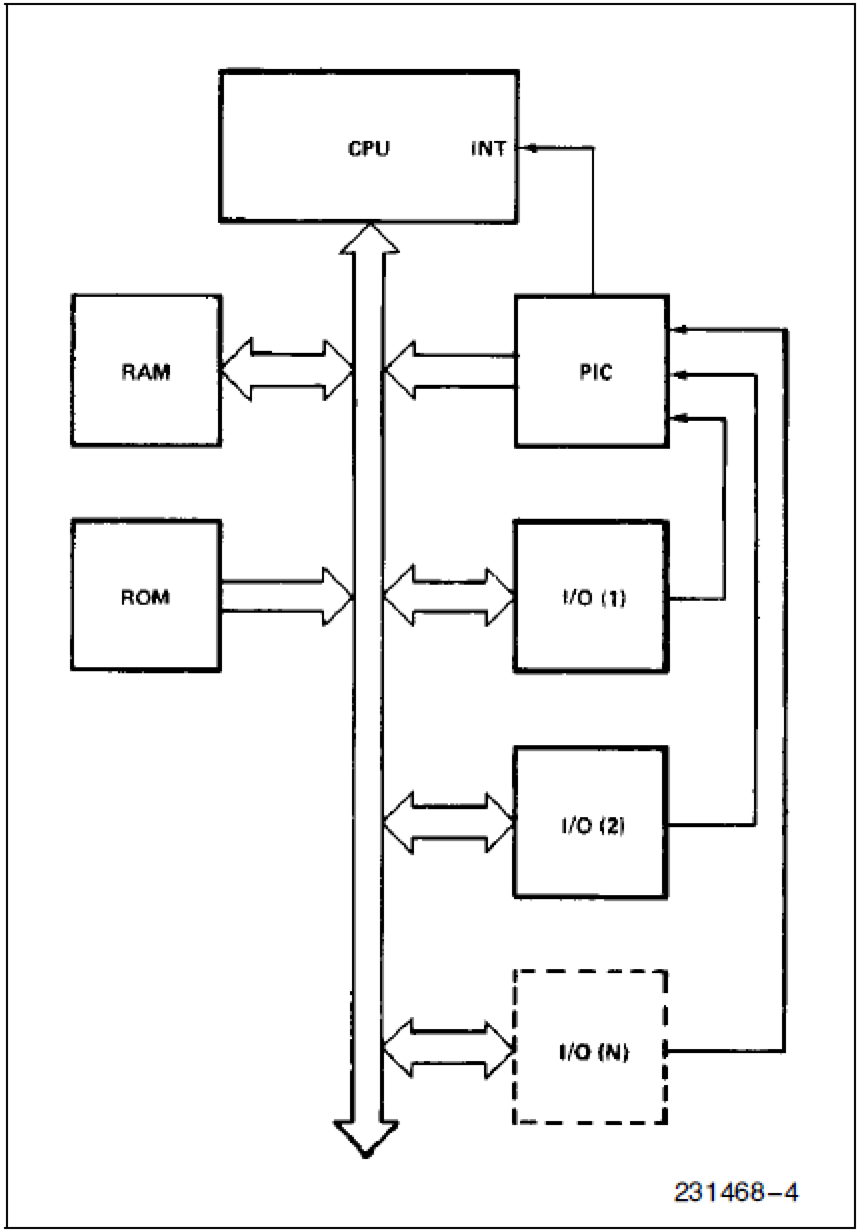


Figure 3b. Interrupt Method

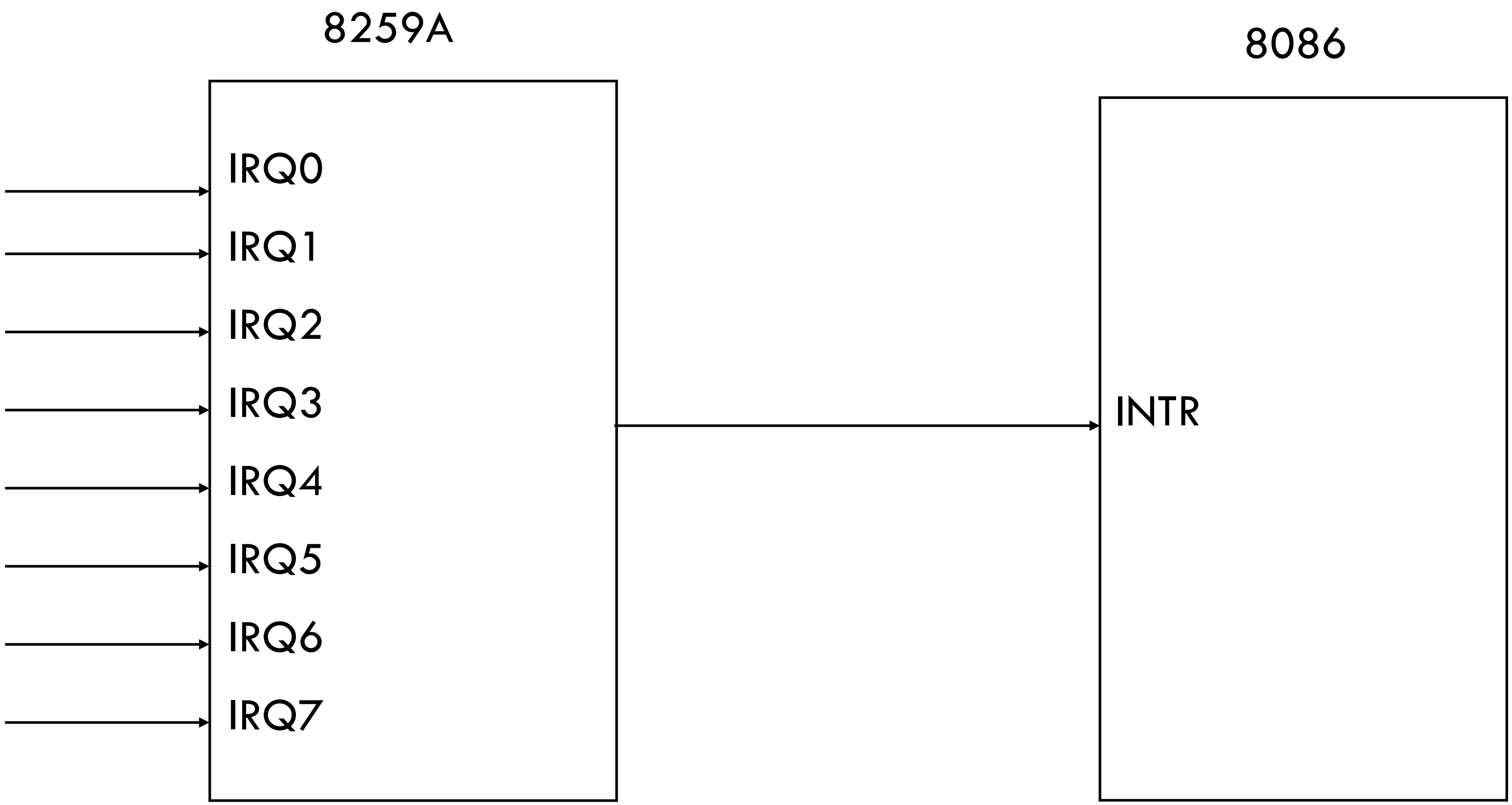
Example

- 80x86 has one interrupt line
- 8086 based systems use one 8259A interrupt controller
- 8259A has 8 interrupt lines

Sequence of Events

- 8259A accepts interrupts
- 8259A determines priority
- 8259A signals 8086 (raises INTR line)
- CPU Acknowledges
- 8259A puts correct vector on data bus
- CPU processes interrupt

PC Interrupt Layout



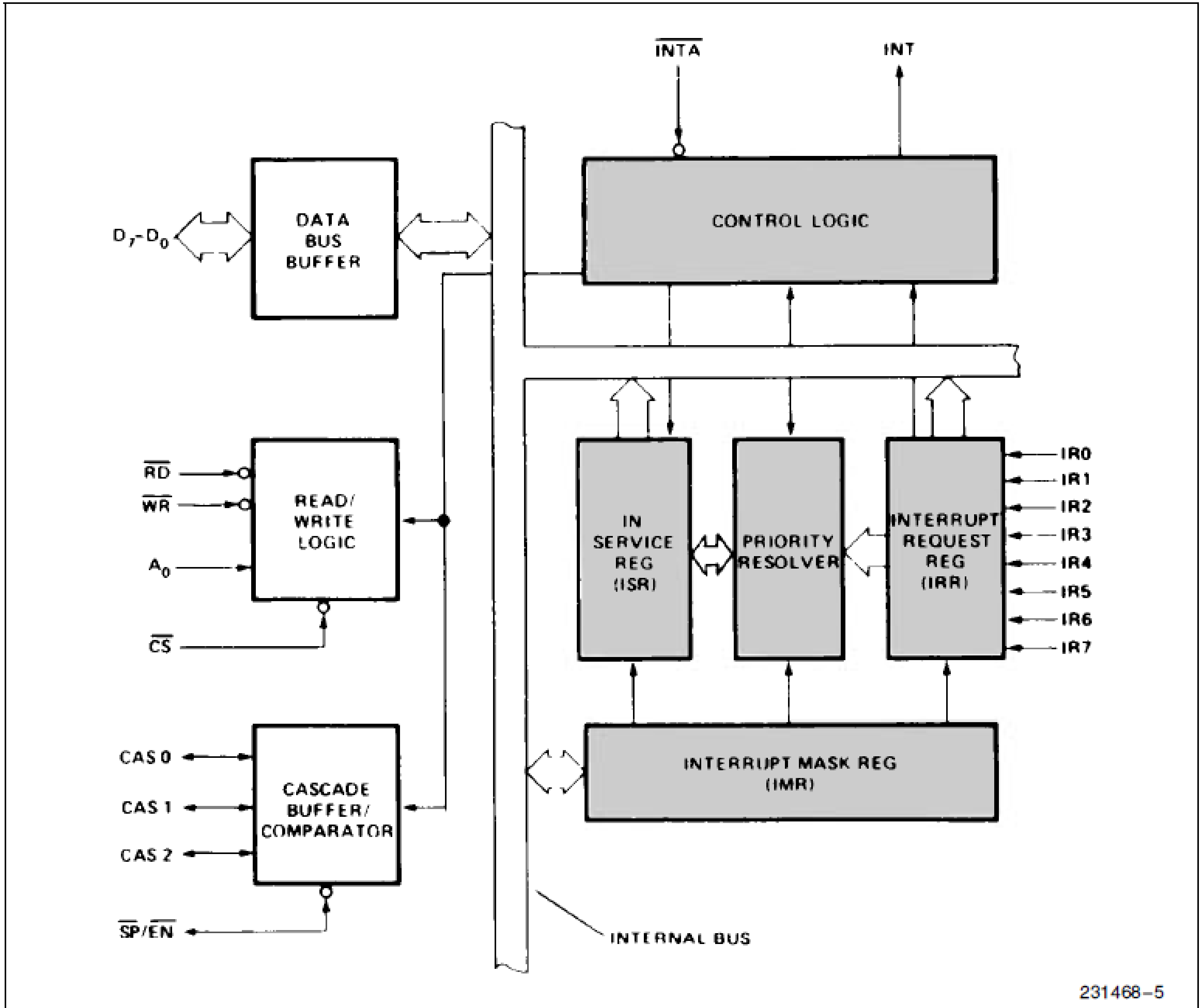


Figure 4a. 8259A Block Diagram

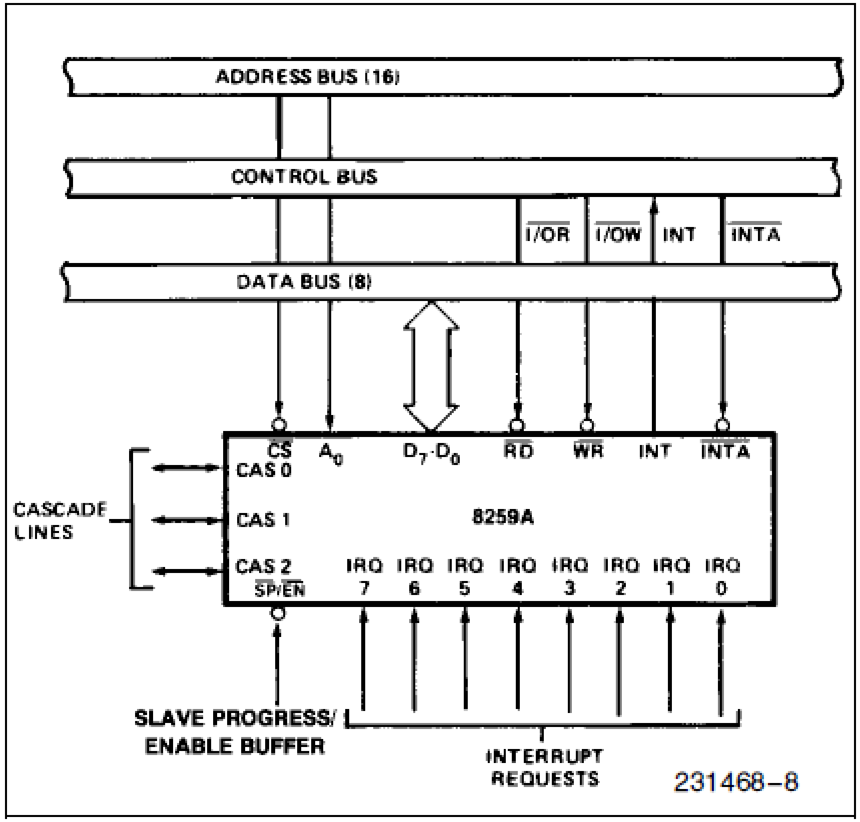


Figure 5. 8259A Interface to Standard System Bus

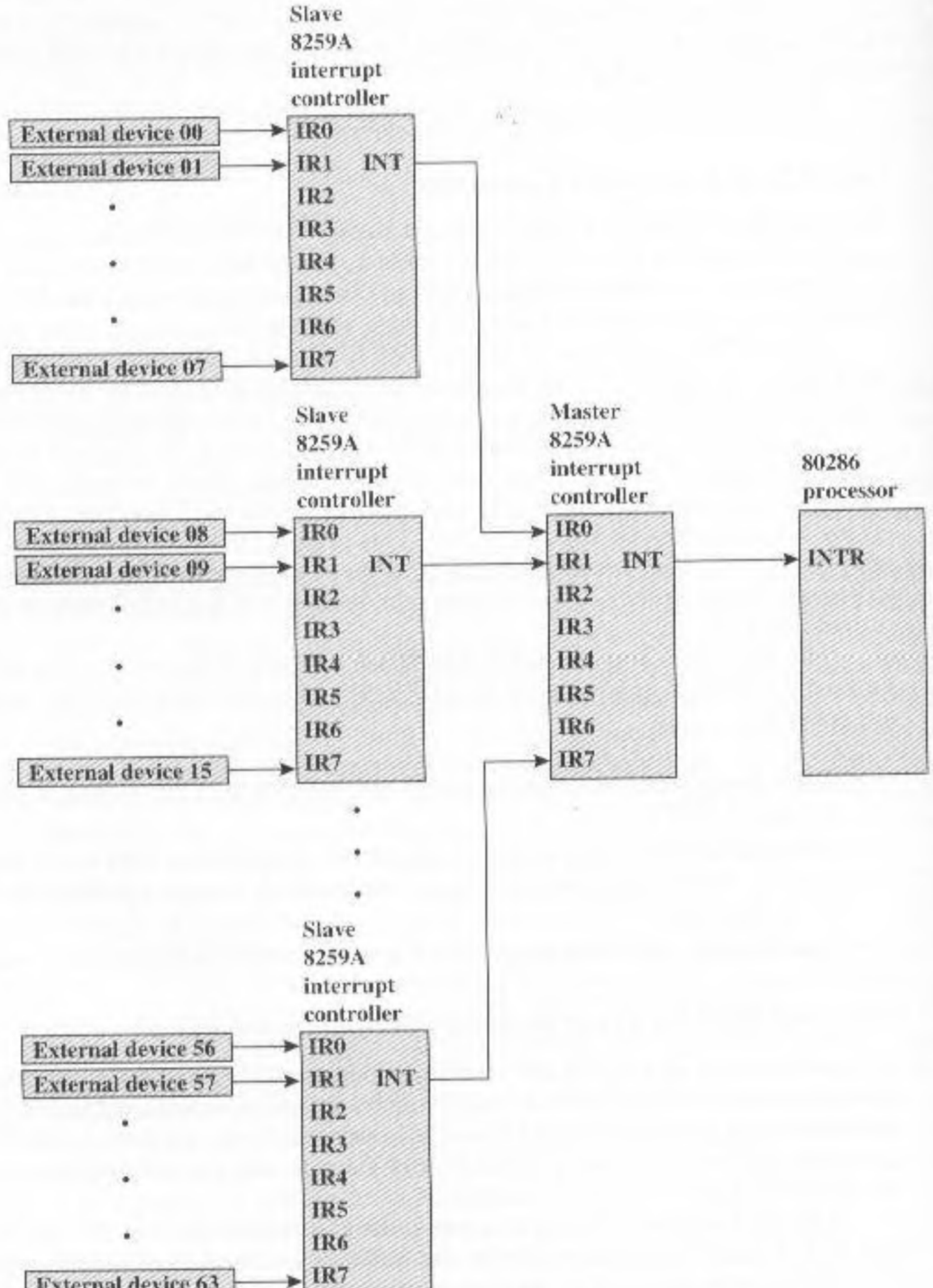
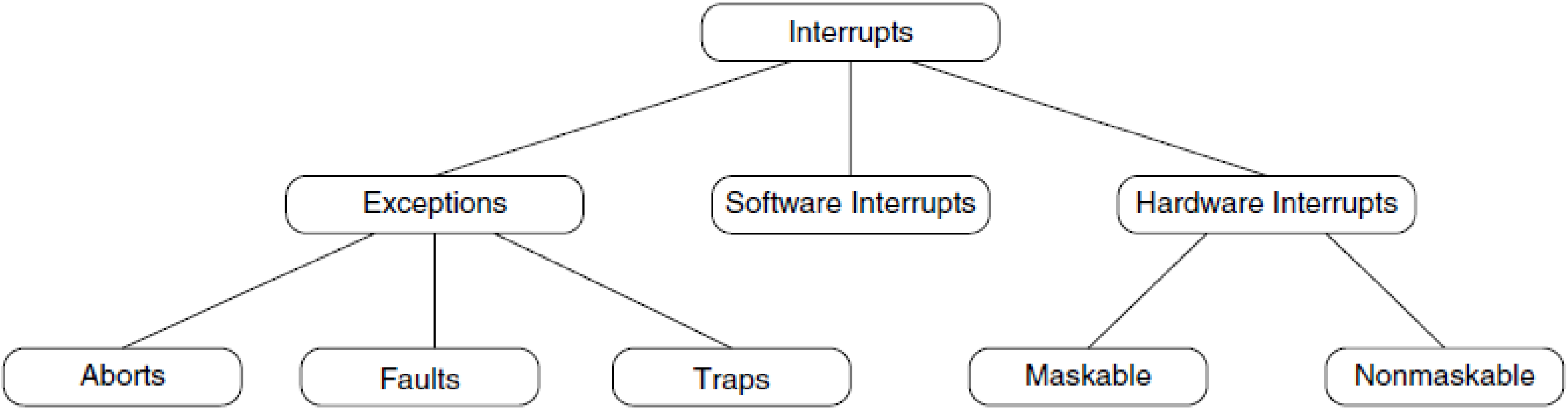


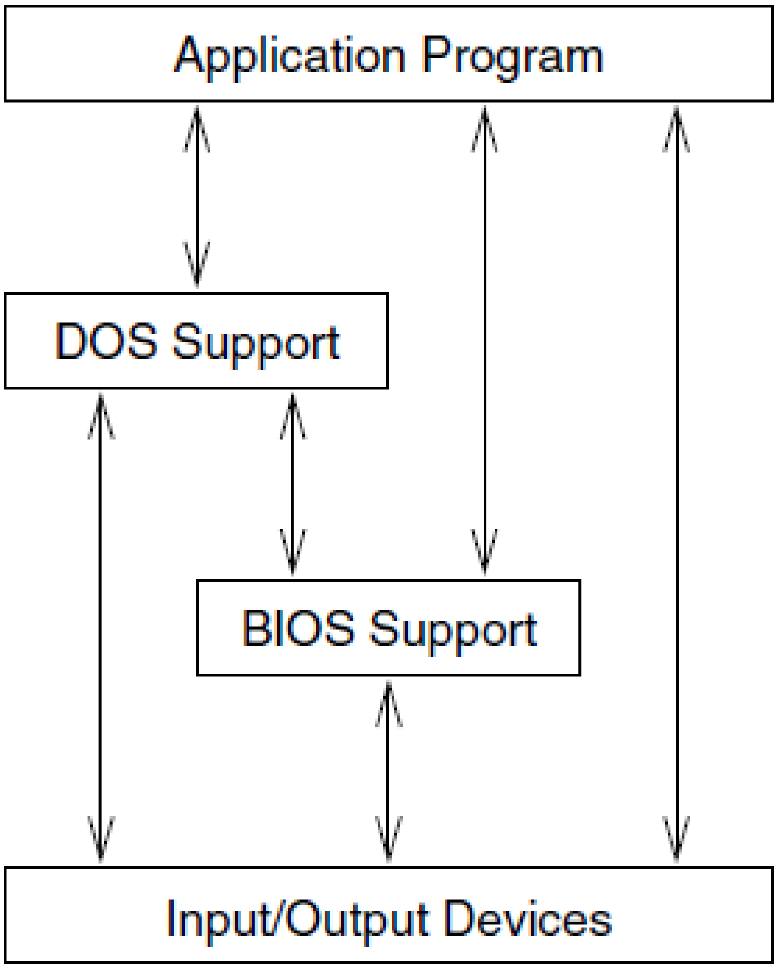
Figure 7.9 Use of the 82C59A Interrupt Controller

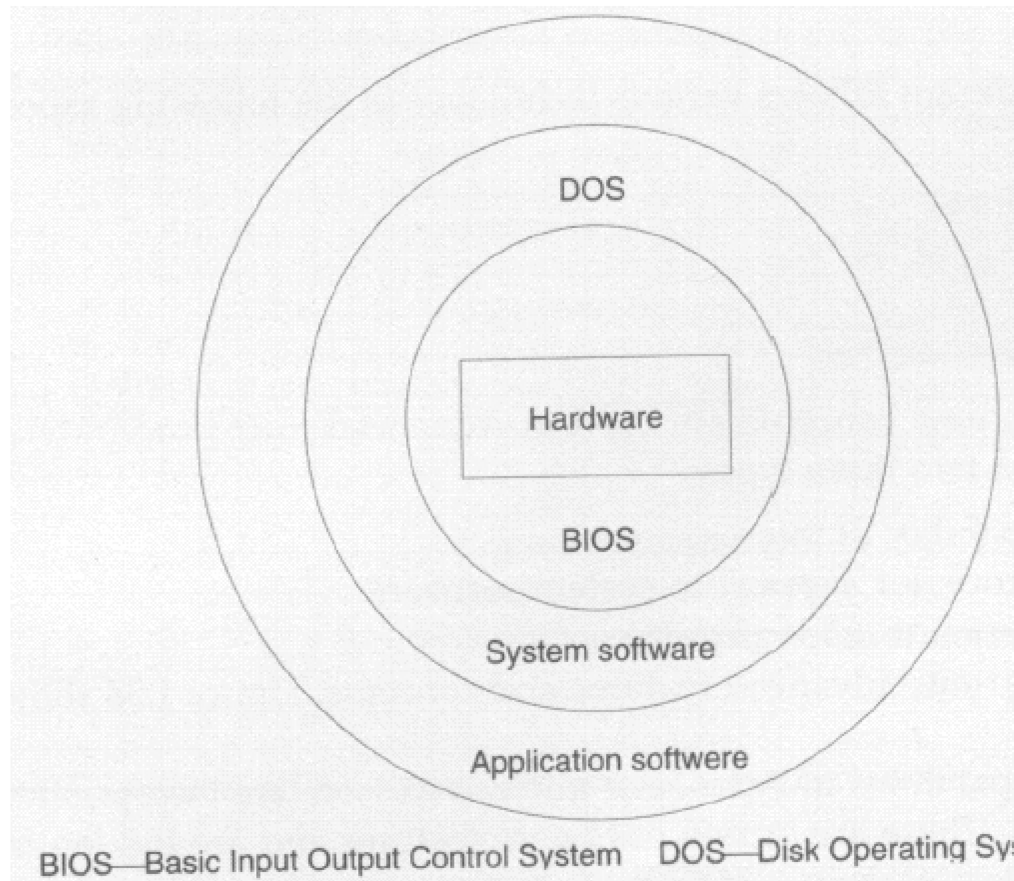
Taxonomy of Interrupts in Pentium



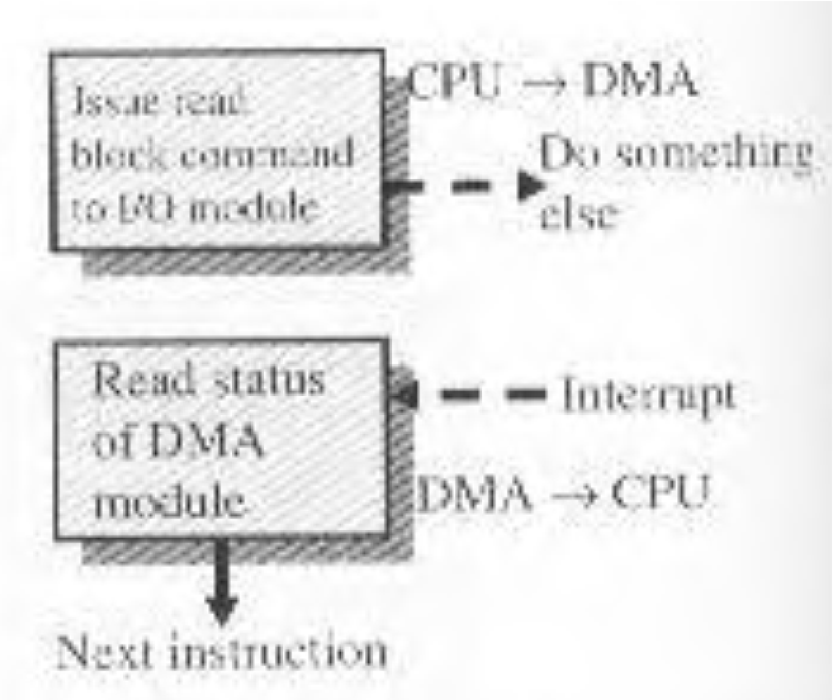
Software Interrupts are Synchronous Events whereas Hardware Interrupts are Asynchronous Events.

Interacting with I/O Devices





Direct Memory Access



I/O Techniques Summary

	No Interrupts	Use of Interrupts
I/O-to-memory Transfer through Processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct Memory Access