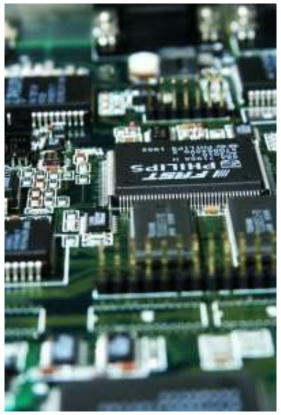
CSE 321b

Computer Organization (2)

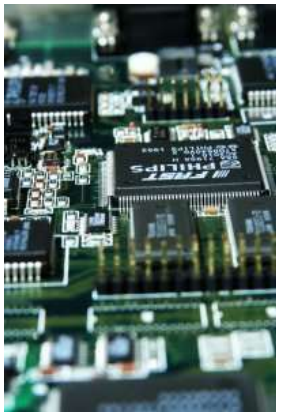
تنظيم الحاسب (2)



3rd year, Computer Engineering

Spring 2018

Lecture #6



Dr. Ahmed Amer Shahin

http://www.aashahine.faculty.zu.edu.eg

Credits to Dr. Ahmed Abdul-Monem & Dr. Hazem Shehata for the slides

Chapter 7. Input / Output

Outline

- External Devices
  - Types
  - Structure
- I/O Modules
  - Function
  - Structure
- I/O Techniques
  - Programmed I/O
  - Interrupt-Driven I/O
  - Direct Memory Access
- I/O Channels & Processors

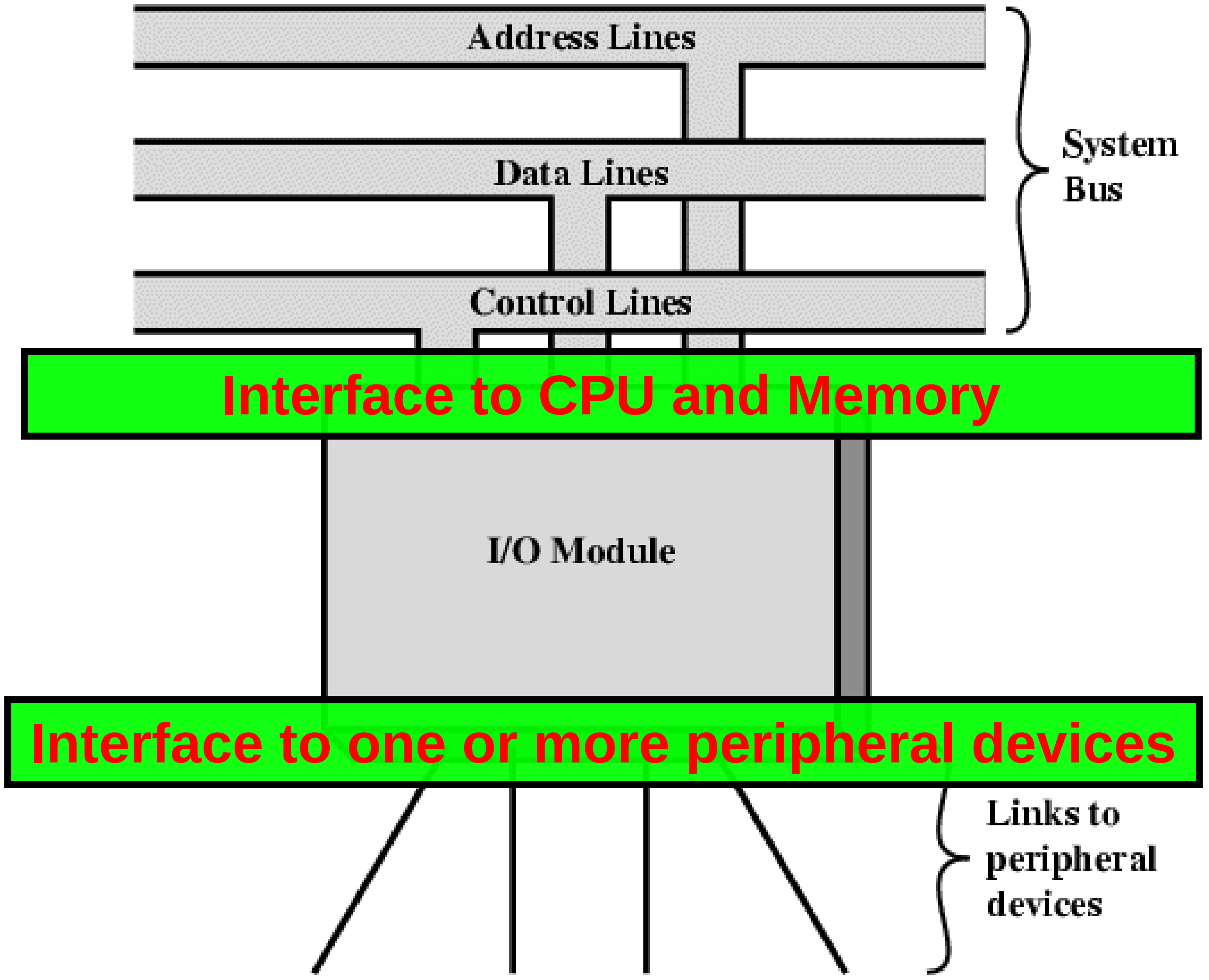
Terms

- Essential Computer Units
  - CPU and Memory
- Peripheral (or External or I/O) devices
  - Any device attached to a computer in order to increase its functionality.
    - Input: keyboard, mouse, scanner, … etc.
    - Output: printers, speakers, … etc.
    - Input and output: hard disk, modem, … etc.
- I/O (Input/Output) Operations
  - Transfer of data to/from computer from/to peripheral device (done by program, operation, or device).
  - Input: from a device to the computer
  - Output: from the computer to a device.

Input/Output Problems

- There is a wide variety of peripherals!
  - Different methods of operation (H/W).
  - Delivering different amounts of data.
  - At different speeds (which are also different from CPU and memory).
  - In different formats (e.g., word length).
- Conclusion: Hard to connect such variety of different devices directly to same Bus!!
- Solution: **I/O Module**

I/O Module

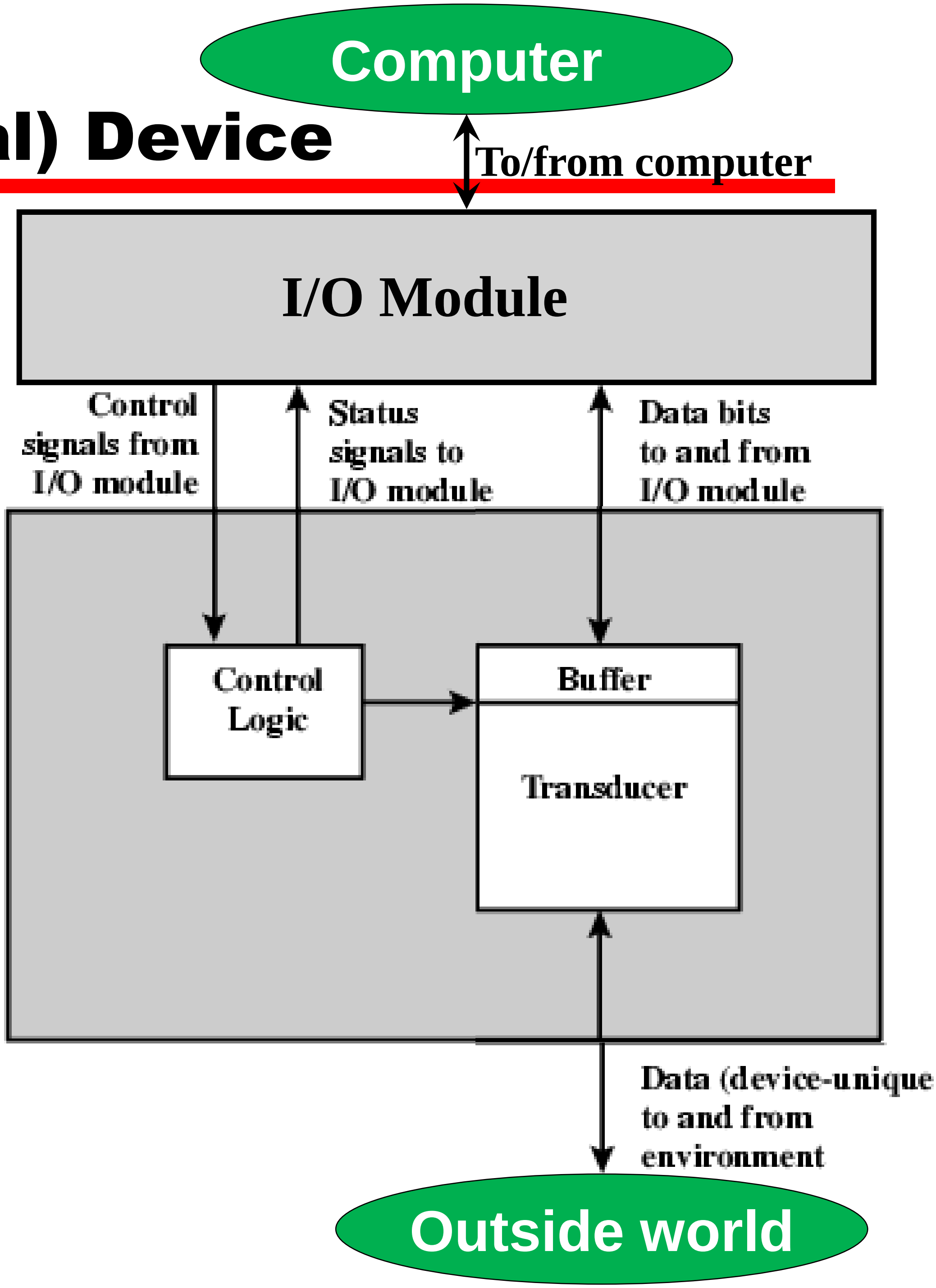

Address Lines
Data Lines
Control Lines
System Bus
Interface to CPU and Memory
I/O Module
Interface to one or more peripheral devices
Links to peripheral devices

Types of Peripherals

- Human readable
  - Screen, printer, keyboard, … etc.
- Machine readable
  - Magnetic disk, tape, … etc.
- Communication
  - Modem, Network Interface Card (NIC), Wireless Network Adaptor, … etc.

Peripheral (External) Device

- Control Signals
  - Send data to module, receive data from module, send status, position disk head.
- Status Signals
  - READY, NOT READY, …
- Buffer
  - Temporarily hold data being transferred. Size: x bytes ➔ x Kbytes!!
- Transducer
  - Converts energy: electrical ←→ other.
- Control logic
  - Controls operation.


Computer
To/from computer
I/O Module
Control signals from I/O module
Status signals to I/O module
Data bits to and from I/O module
Control Logic
Buffer
Transducer
Data (device-unique to and from environment
Outside world

Examples: Keyboard/Monitor, and Disk Drive

- Keyboard (input)
  - A key is pressed.
  - Transducer translates signal into ASCII.
  - ASCII is transmitted to I/O module in the computer.
  - Text can be stored as ASCII in the computer.
- Monitor (output)
  - Computer sends ASCII to I/O module. I/O module sends ASCII to external device (monitor).
  - Transducer at the monitor sends electronic signals to display the character.
- Hard Disk Drive (input/output)
  - Head moves in and out across disk surface.
  - Transducer converts magnetic patterns to/from bits.

Functions of I/O Module

1. Control & Timing.
2. CPU Communication.
3. Device Communication.
4. Data Buffering.
5. Error Detection.

1. Control & Timing

- I/O includes control & timing requirement to coordinate the flow of traffic between internal resources (CPU, MM, …) and external devices.
- Ex.: Transfer data from input device to CPU:
  1. CPU checks I/O module device status.
  2. I/O module returns status.
  3. If ready, CPU requests data transfer (command to I/O module).
  4. I/O module gets data from external device.
  5. I/O module transfers data to CPU.
- If transfer goes through bus, each CPU/module interaction may involve 1^+ bus arbitrations.

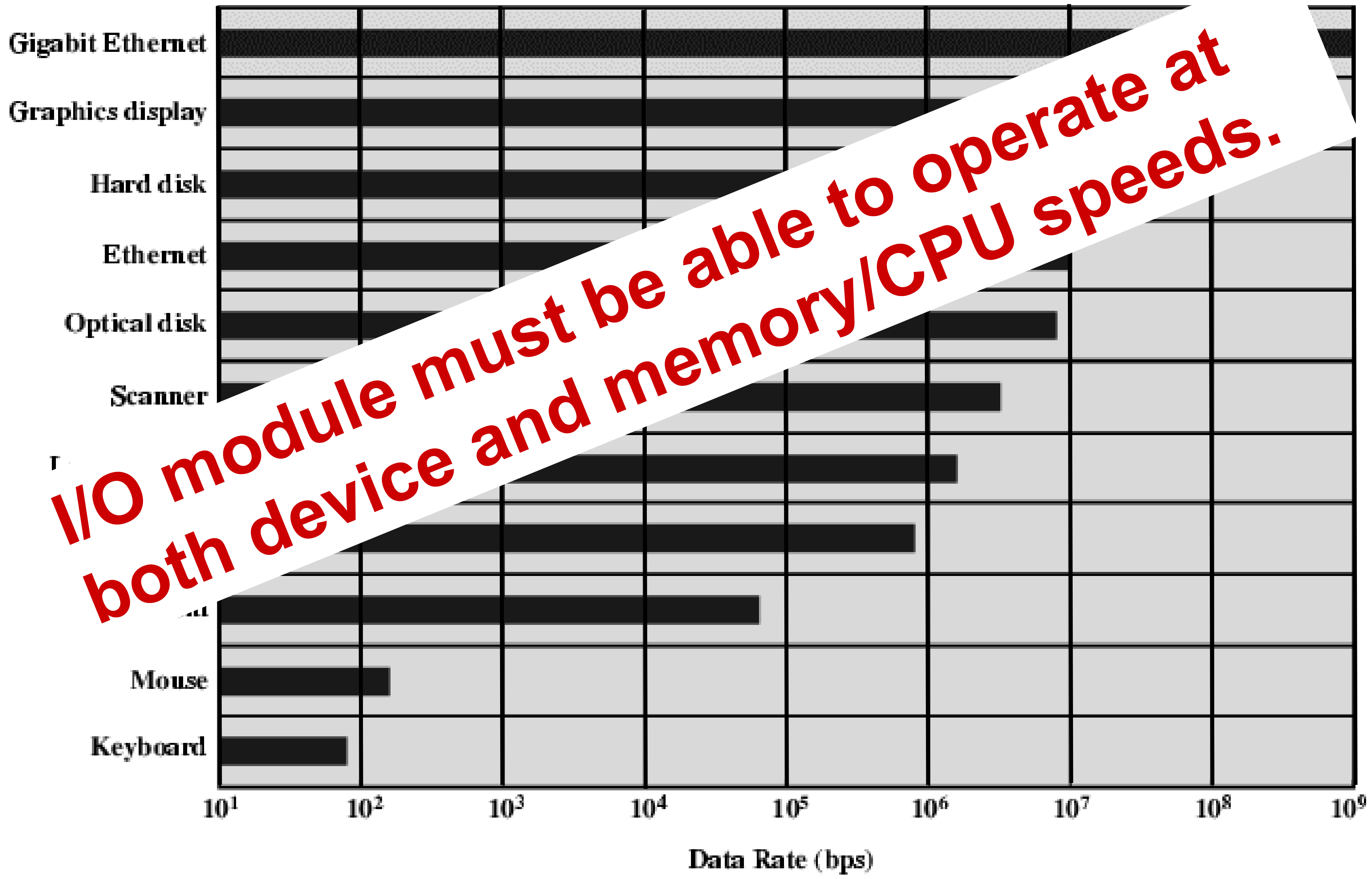
2. CPU Communication

- CPU communication involves the following:
  - Command decoding
    - Module accepts commands from CPU on control lines.
    - Command parameters can be sent over data line.
    - e.g., SEEK track in a disk drive: SEEK command sent on control lines and track # sent on data lines.
  - Address recognition
    - One unique address for each peripheral it controls.
  - Data exchange
    - Between CPU and device over the data bus.
  - Status reporting
    - BUSY, READY, or some error conditions.

3. Device Communication

- I/O module must also be able to do device communication:
  - Commands
  - Status information
  - Data

4. Data Buffering (Speed Mismatch)



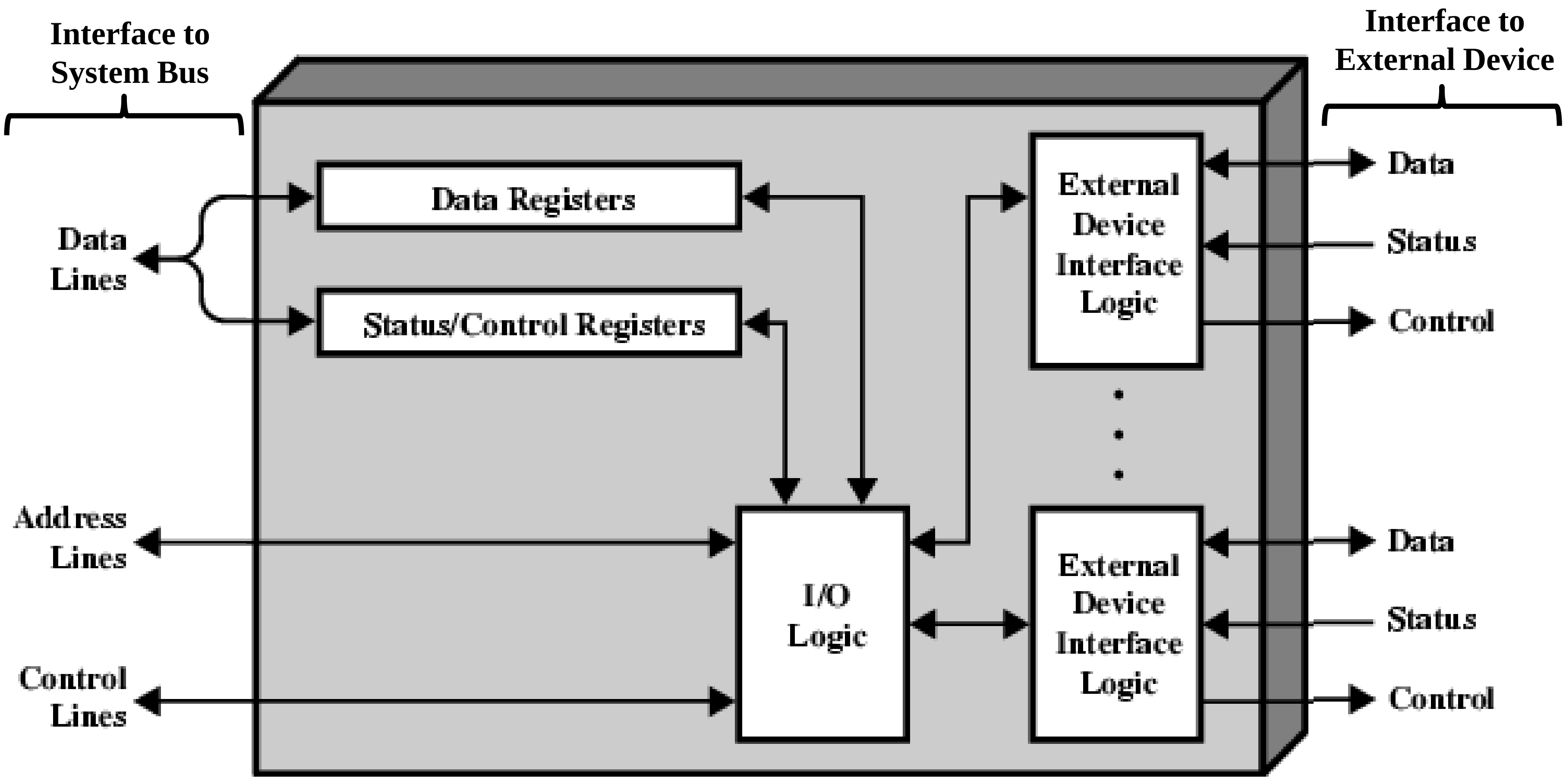
I/O module must be able to operate at both device and memory/CPU speeds.

5. Error Detection

- Mechanical and electrical malfunctions
  - Report to CPU.
  - e.g., paper jam, bad disk sector/track.
- Unintentional changes to transmitted bit pattern
  - Detected using error-detecting codes.
  - e.g., parity bit (ASCII).

I/O Module Structure

- St./Ctrl. registers: hold device status or accept control info from CPU.
- CPU issues commands to I/O module via control lines.
- Some control lines are also used by I/O module for bus arbitration.
- Module controlling more than 1 device has a set of unique addresses.



I/O Module Design Decisions

- I/O module lets CPU view a wide range of devices in a simple-minded way.
- Module may hide device properties from CPU:
  - —Quite complex module design.
  - —Simple CPU commands (e.g., render object).
  - —Referred to as **I/O channel** (**I/O processor**).
  - —Common in mainframes.
- Module may reveal device properties to CPU:
  - —Relatively simple module design.
  - —Detailed CPU commands (e.g., rewind tape).
  - —Referred to as **I/O controller** (**device controller**).
  - —Common in microcomputers.

Chapter 7. Input / Output (*Cont.*)

Outline

- External Devices
  - Types
  - Structure
- I/O Modules
  - Function
  - Structure
- I/O Techniques
  - Programmed I/O
  - Interrupt-Driven I/O
  - Direct Memory Access
- I/O Channels & Processors

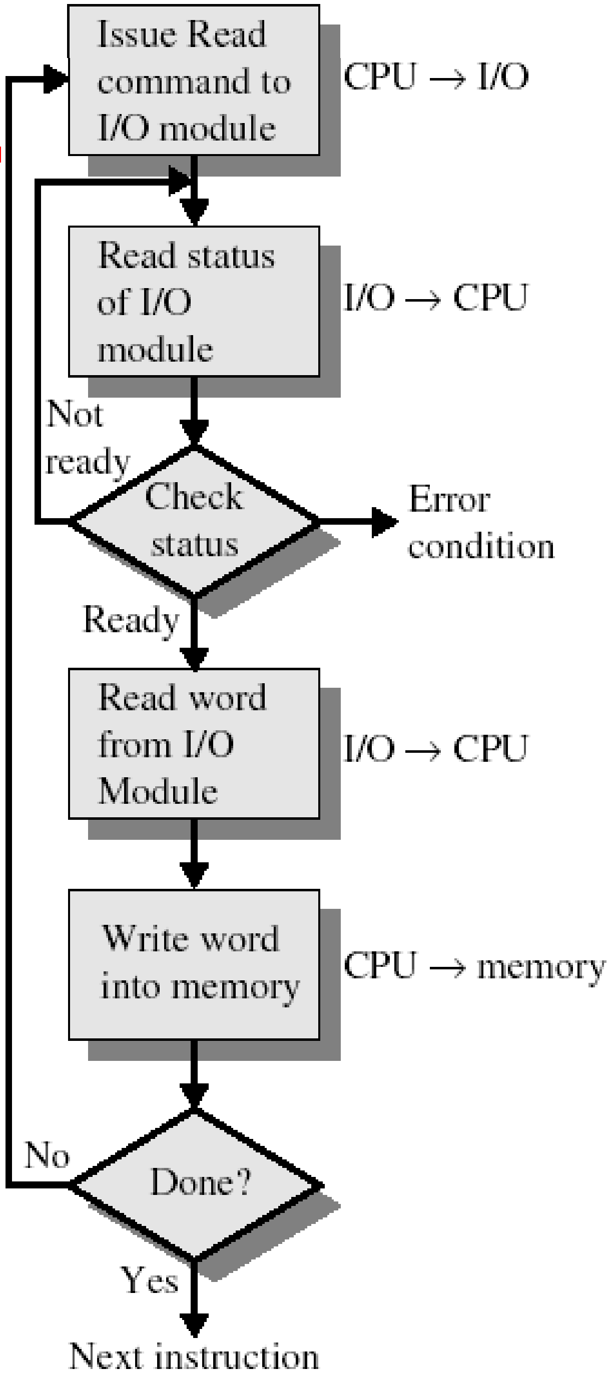
Input Output Techniques

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

	No Interrupts	Use of Interrupts
I/O-to-memory transfer through CPU	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct Memory Access (DMA)

Programmed I/O

- CPU (program) has direct control over I/O.
  - Issuing commands
  - Sensing status
  - Transferring data
- CPU issues a command to I/O module.
- CPU checks status bits periodically.
  - This process is called: pooling.
- I/O module performs operation.
- I/O module sets status bits.
- CPU transfers data: device ←→ memory.
- Notes:
  - I/O module does not inform CPU directly.
    - I/O module does not interrupt CPU.
  - CPU waits for I/O operation to complete.
    - **Disadvantage**: Wasting CPU time!



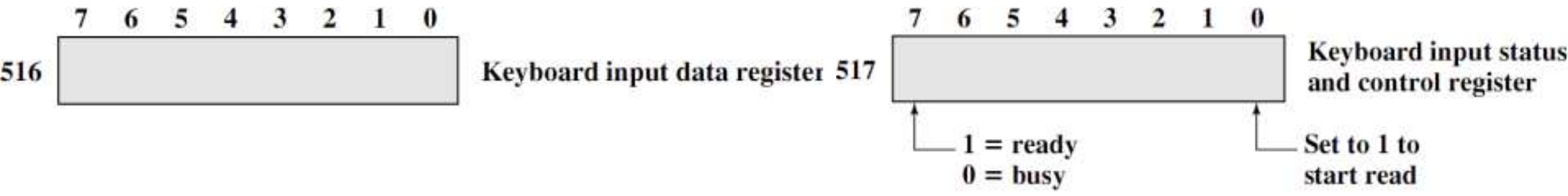
I/O Commands vs. I/O Instructions

- I/O Command
  - Signal: issued by (or Sent from) CPU to I/O module.
  - Types:
    - Control: activate device & tell it what to do (e.g., rewind tape).
    - Test: check status (e.g., is power on? is error detected?)
    - Read/Write: transfer data CPU ←→ buffer ←→ peripheral
- I/O instruction
  - Step in program: fetched from MM & executed by CPU.
  - To execute an I/O instruction: (1) CPU issues address of I/O module & device, (2) CPU issues an I/O command.
  - Instruction form depends on how devices are addressed.
- In programmed I/O, there is a one-to-one mapping between I/O instructions and I/O commands.

Addressing Techniques of I/O Devices

- Two ways to assign addresses to I/O devices:
  1. Memory-mapped I/O
     - Devices & memory share same address space.
     - I/O looks just like memory read/write.
     - No special instructions for I/O ➔ "load", "store", … etc.
     - Bus has one Read & one Write control line.
     - **Pros**: Large selection of memory access instructions.
     - **Cons**: Valuable memory address space is used up!
  2. Isolated I/O
     - Separate address space for devices.
     - Special instructions for I/O ➔ "in", "out", "test", … etc.
     - Need two Rd & two Wr control lines.
     - **Pros**: efficient use of memory address space.
     - **Cons**: Not so many I/O instructions.

I/O Mapping - Example



ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load AC	"1"	Load accumulator
	Store AC	517	Initiate keyboard read
202	Load AC	517	Get status byte
	Branch if Sign = 0	202	Loop until ready
	Load AC	516	Load data byte

(a) Memory-mapped I/O

ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load I/O	5	Initiate keyboard read
201	Test I/O	5	Check for completion
	Branch Not Ready	201	Loop until complete
	In	5	Load data byte

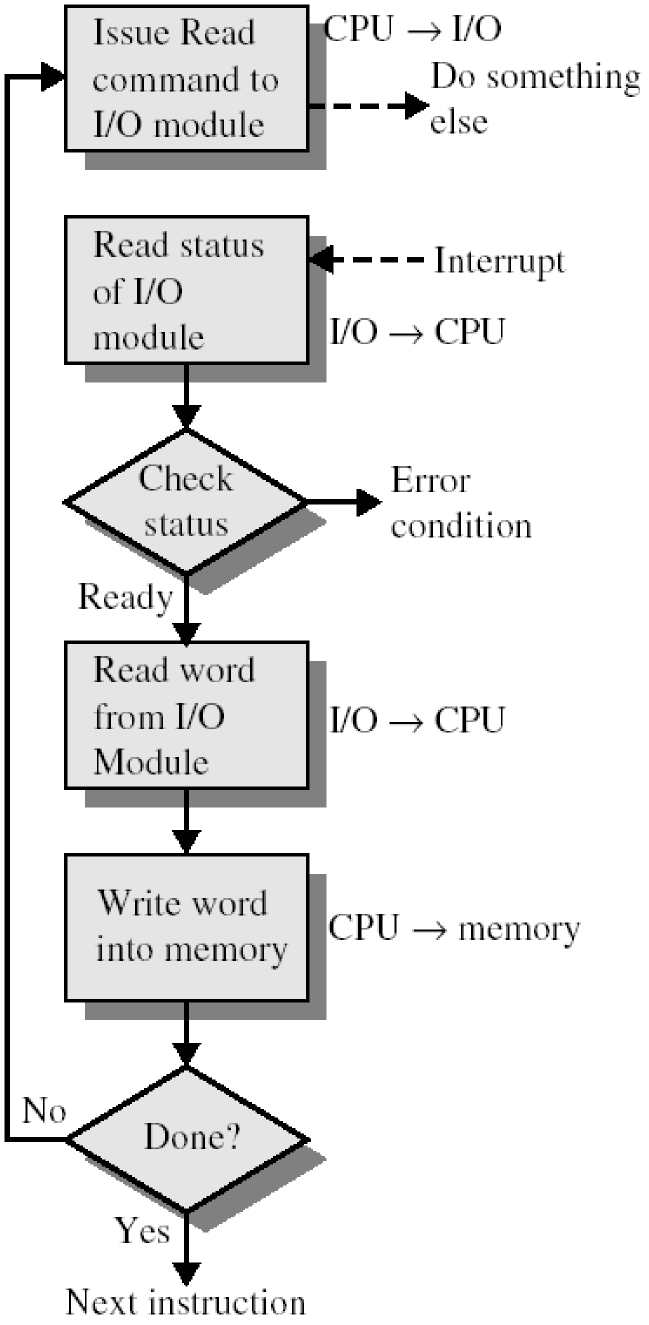
(b) Isolated I/O

Interrupt-Driven I/O

- Purpose: To overcome CPU waiting.
- No repeated CPU checking of device.
- CPU issues command and moves on to do other useful work.
- I/O module interrupts CPU when ready.

Interrupt-Driven I/O

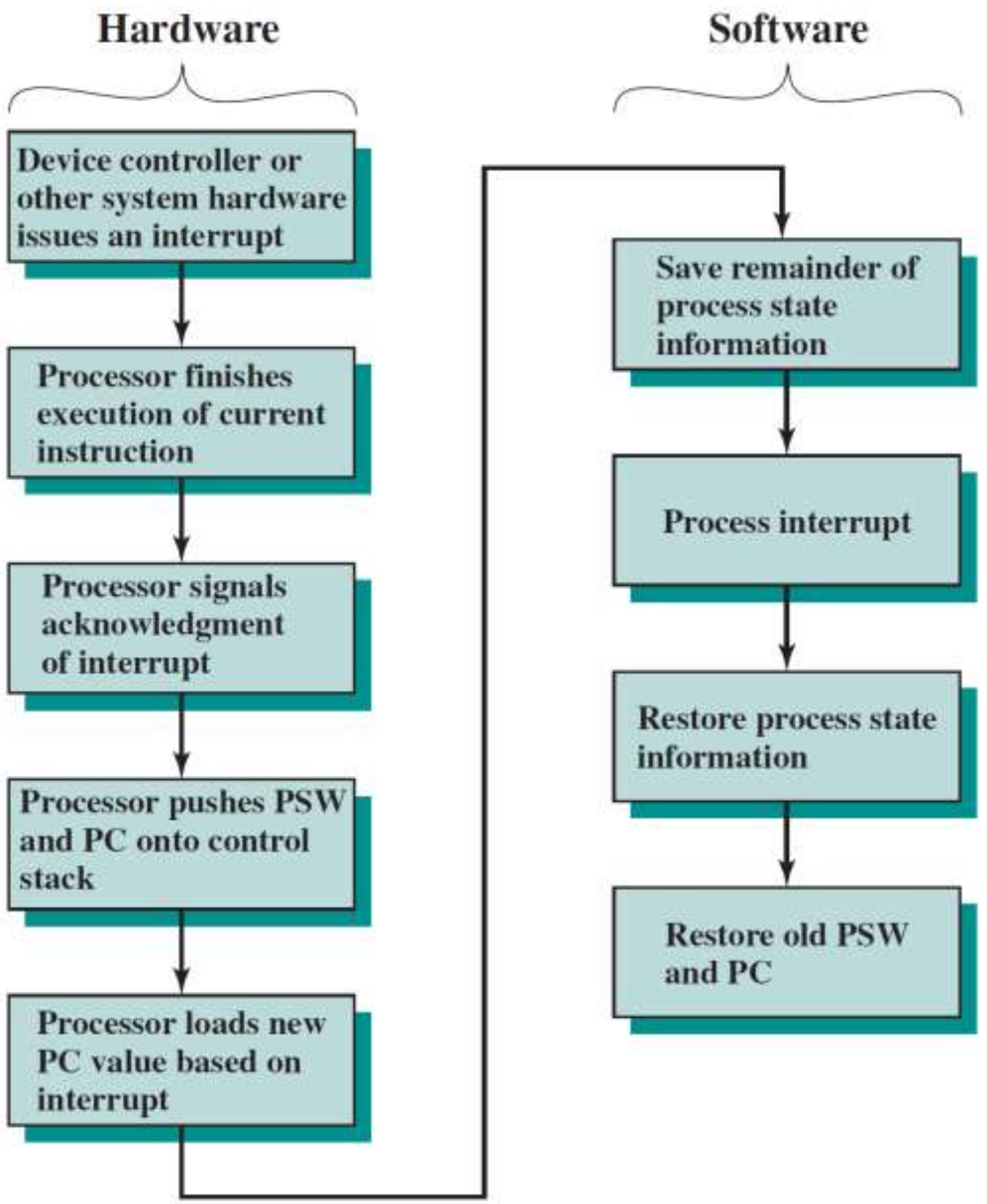
- CPU issues read command
- I/O module gets data from peripheral whilst CPU does other work.
- I/O module interrupts CPU.
- CPU reads data from I/O module.
- CPU writes data to memory.



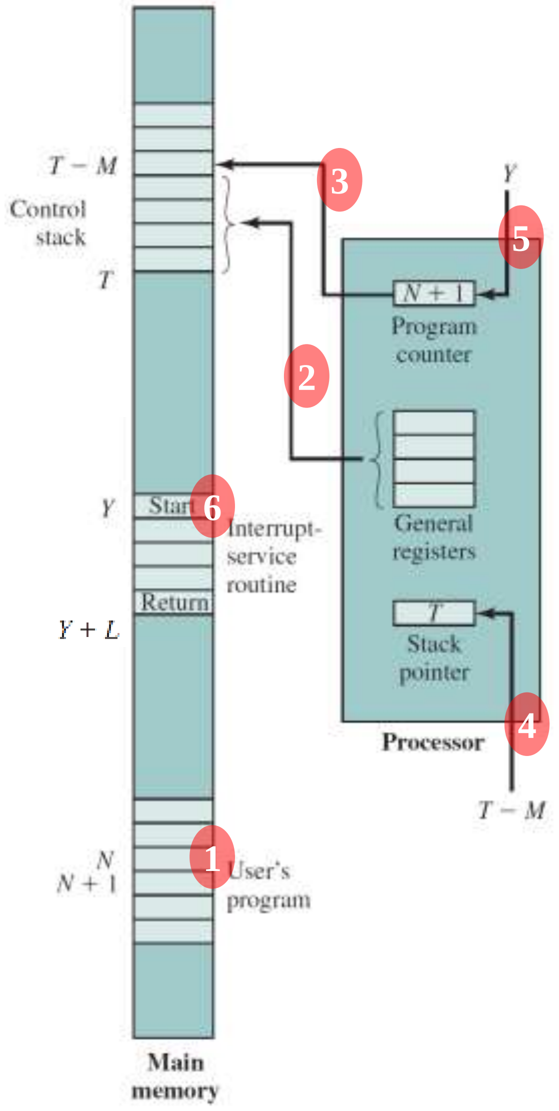
CPU Viewpoint

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

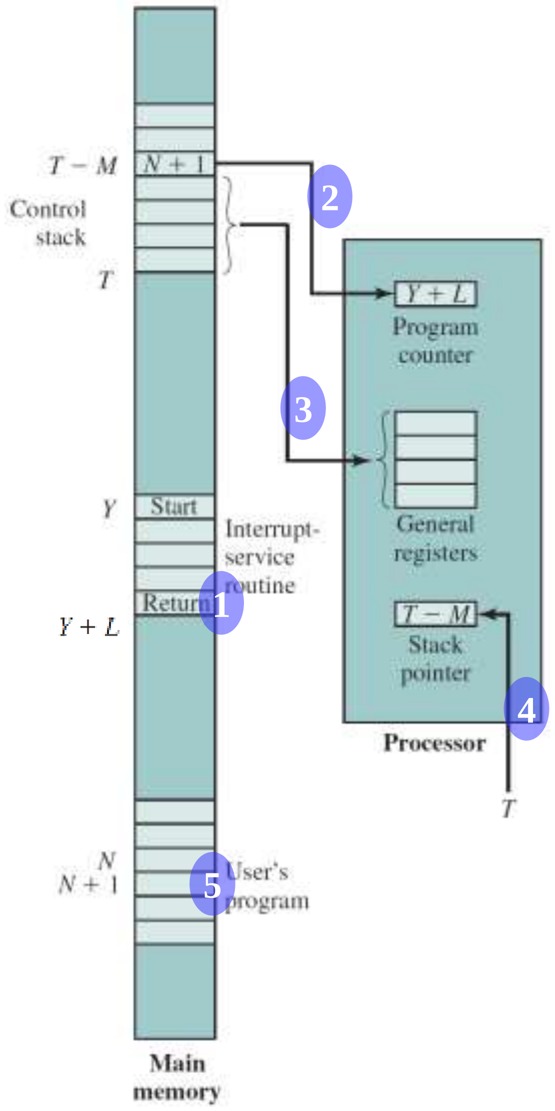
Simple Interrupt Processing


Hardware
Software
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
Save remainder of process state information
Process interrupt
Restore process state information
Restore old PSW and PC

Changes in Mem. & Reg.'s upon Interrupt



(a) Interrupt occurs after instruction at location N



(b) Return from interrupt

Identifying Interrupting Module

- How to identify the module issuing the interrupt?
  1. Different line for each module
     – Limits number of devices.
  2. Software poll (single line)
     – CPU asks each module in turn ➔ time consuming!!
     – e.g., Send TESTI/O ➔ Set address lines ➔ Check status reg.
  3. Daisy chain or Hardware poll (single line)
     – All modules share a single interrupt request line.
     – Interrupt acknowledge sent down a chain.
     – Module (that issued interrupt signal) places its vector on bus.
     – CPU uses vector to identify handler routine.
  4. Bus Master (single line)
     – Module must claim the bus before it can raise interrupt.
     – e.g., PCI & SCSI.

Multiple Interrupts & Priorities

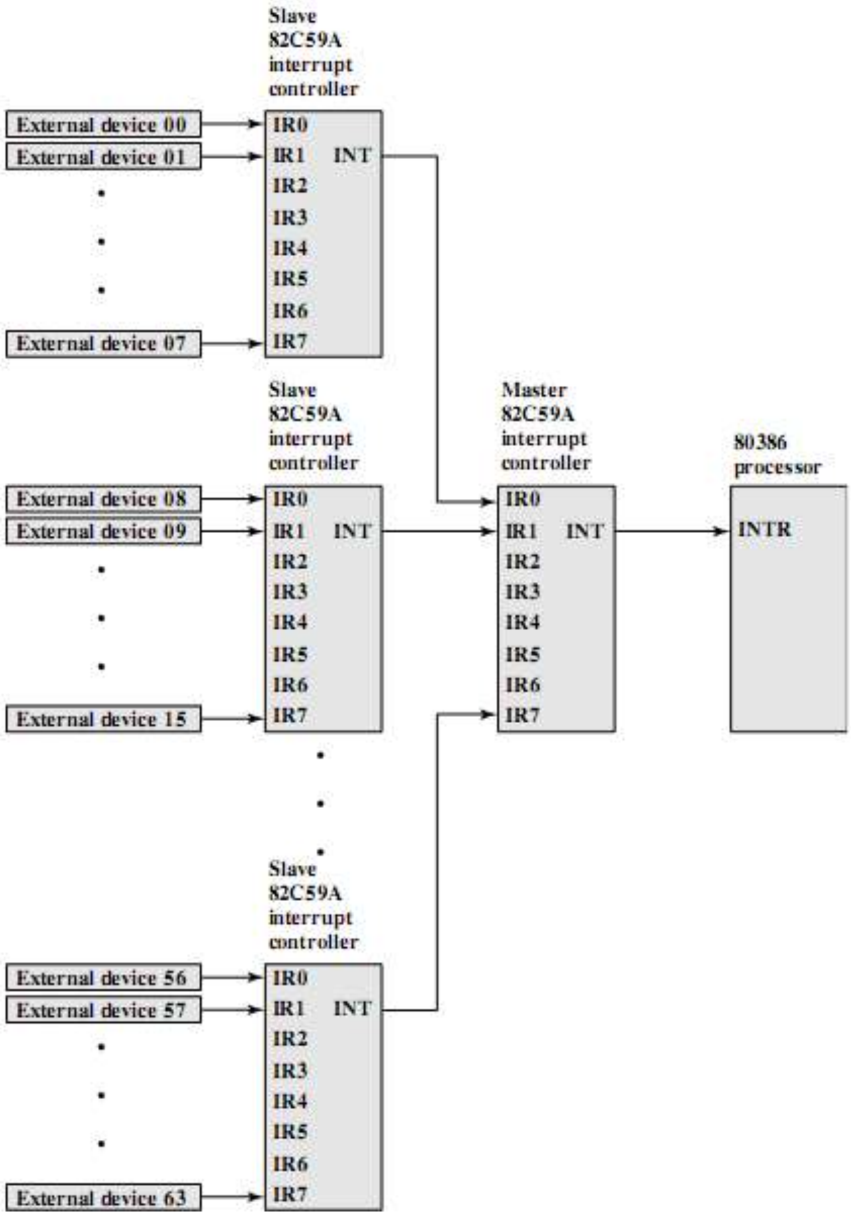
- How to deal with simultaneous interrupts?
- Solution: prioritize interrupts!

1. With multiple lines:
   - Each interrupt line has a priority.
   - Higher priority lines can interrupt lower priority lines.
2. With software polling:
   - Priority determined by order in which modules are polled.
3. With daisy chain:
   - Priority determined by order of modules on chain.
   - Closer modules have higher priority.
4. With bus arbitration:
   - Only current master can interrupt.
   - Priority is defined by the bus arbitration protocol.

Reading

Example - PC Bus

- 80386 has one interrupt line!
- To handle more interrupts, connect 1 (or more) interrupt arbiter ➔ Intel 8259.
- Intel 8259 has 8 intrpt. lines.
- Sequence of events:
  - 8259 accepts interrupts.
  - 8259 determines priority.
  - 8259 signals 8086 (raises INTR line).
  - CPU Acknowledges.
  - 8259 puts correct vector on data bus.
  - CPU processes interrupt.

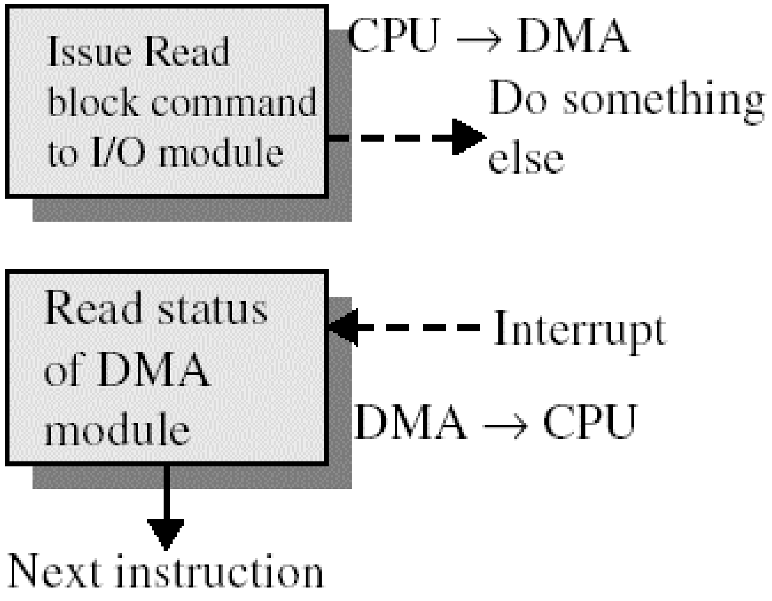


Direct Memory Access (DMA)

- Interrupt driven and programmed I/O require active CPU intervention.
  - CPU tests and services a device.
    - Transfer rate is limited (depending on CPU availability)!!
  - Many instructions are executed for every I/O.
    - CPU is tied up in managing an I/O transfer!!
- DMA is a more efficient technique (when transferring large volumes of data, i.e., blocks).
  - Additional module on the system bus → DMA controller.
  - DMA controller mimics CPU and takes over the bus to transfer the data with no CPU intervention!

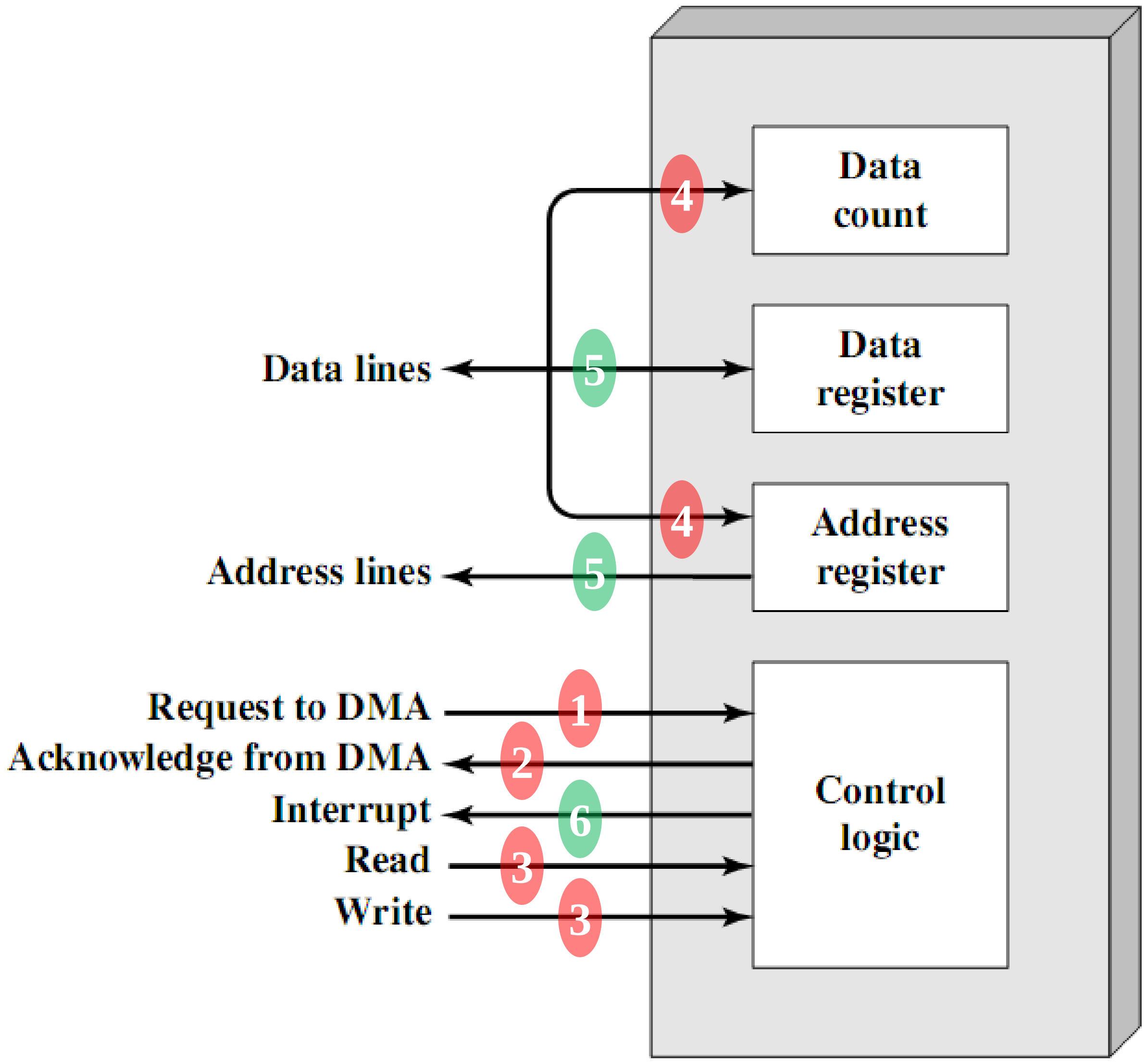
DMA Operation

- CPU tells DMA controller:
  - —Type of Operation (Rd/Wr).
  - —Address of device.
  - —Starting address of a data block in memory.
  - —Amount of data to be transferred.


Issue Read block command to I/O module
CPU → DMA
Do something else
Read status of DMA module
Interrupt
DMA → CPU
Next instruction

- CPU carries on with other work.
- DMA controller performs the transfer.
- DMA controller sends interrupt when finished.

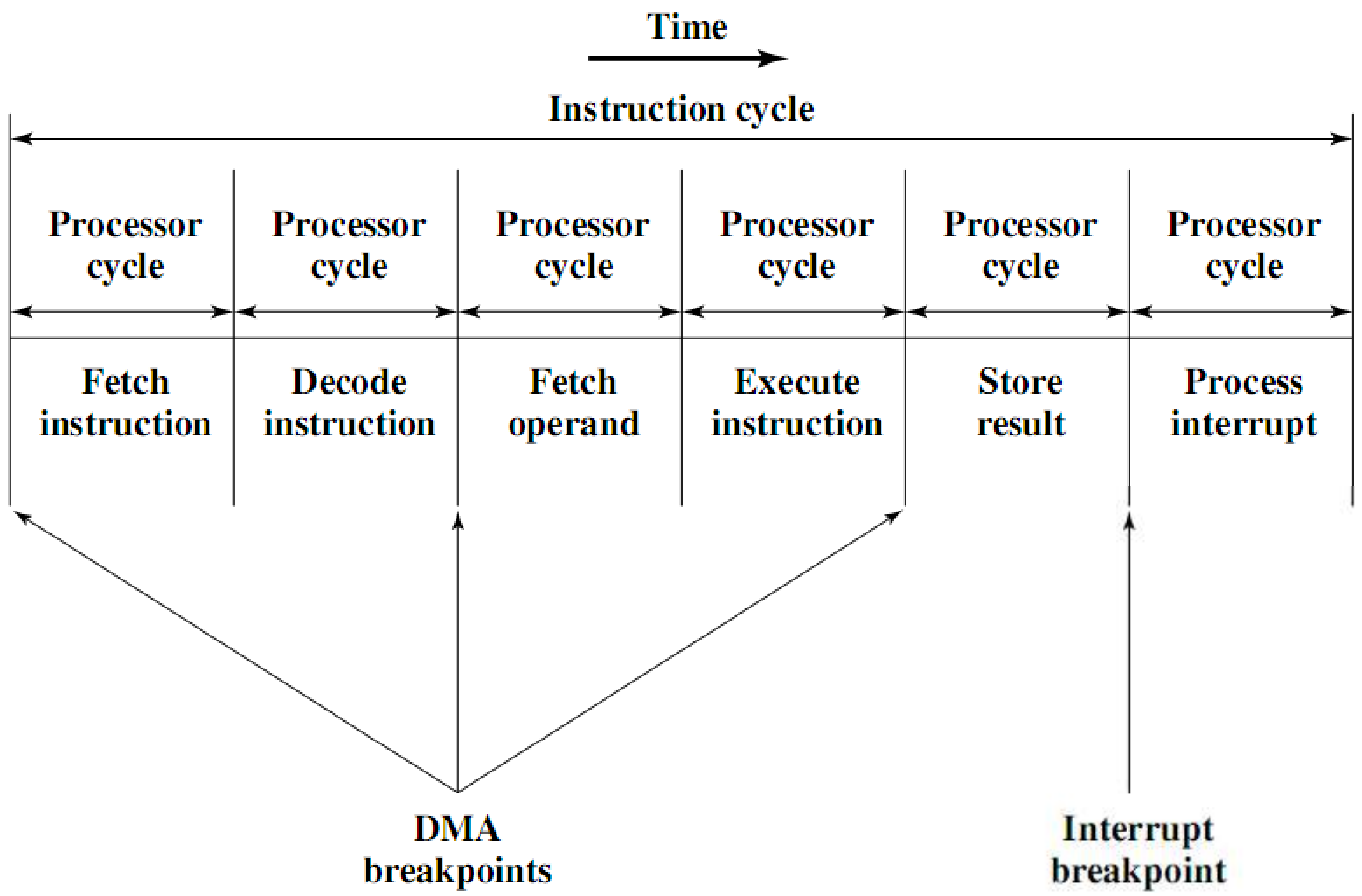
Typical DMA Controller



DMA Transfer modes

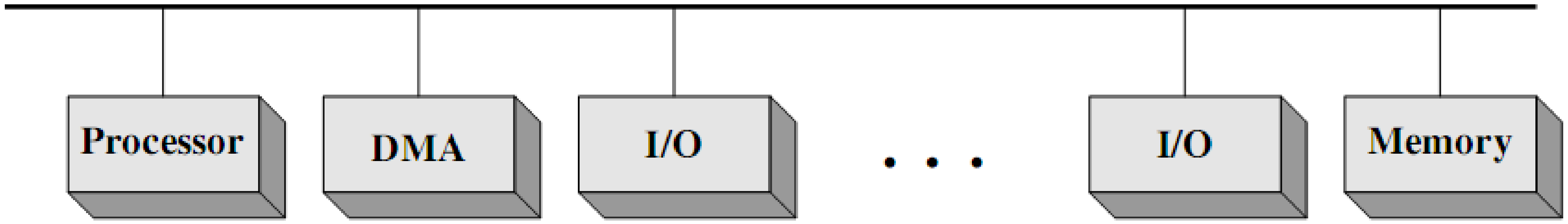
- DMA controller transfers data from/to MM over the system bus.
- DMA controller takes over bus for a bus cycle to transfer data by one of the following techniques:
  - —Use bus only when CPU not using it: transparent mode.
  - —Force CPU to suspend operation temporarily ➔ DMA steals bus cycles from CPU: cycle stealing mode.
- Notice this is not an interrupt!
  - —CPU does not switch context.
- CPU gets suspended just before it accesses bus.
  - —i.e. before an operand or data fetch or a data write.
- Slows down CPU. Faster than CPU doing transfer!

DMA and Interrupt Breakpoints During an Instruction Cycle



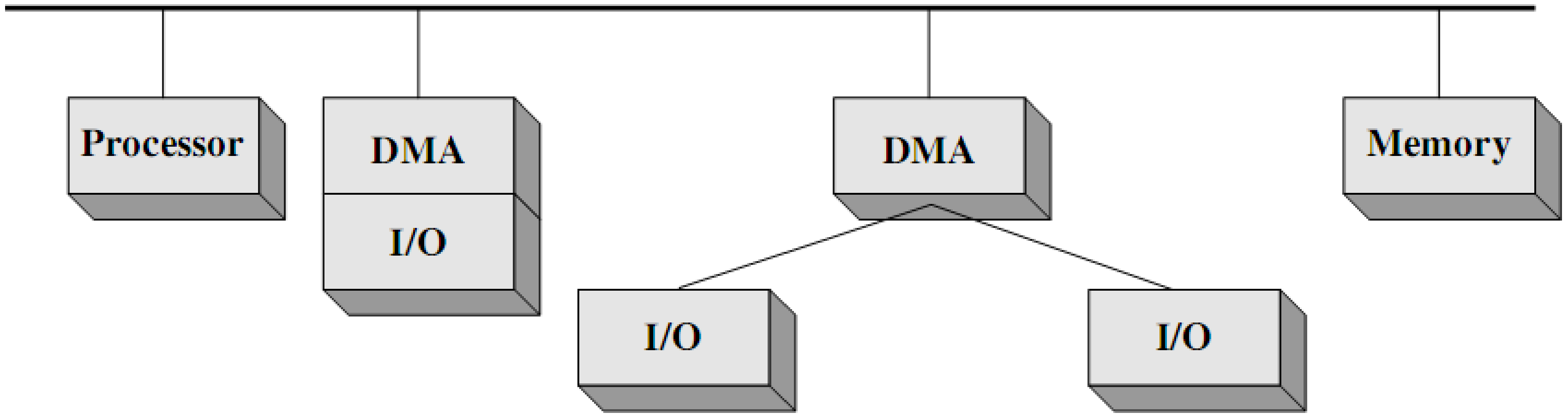
DMA Configurations (1)

- Features: single-bus, detached DMA controller.
- DMA module acts as a surrogate processor.
- DMA module uses programmed I/O.
- Each transfer uses system bus twice.
  - I/O device → DMA controller → memory.
- CPU is suspended twice per transfer.
- DMA controller has no I/O interfaces.
- Inexpensive yet inefficient!



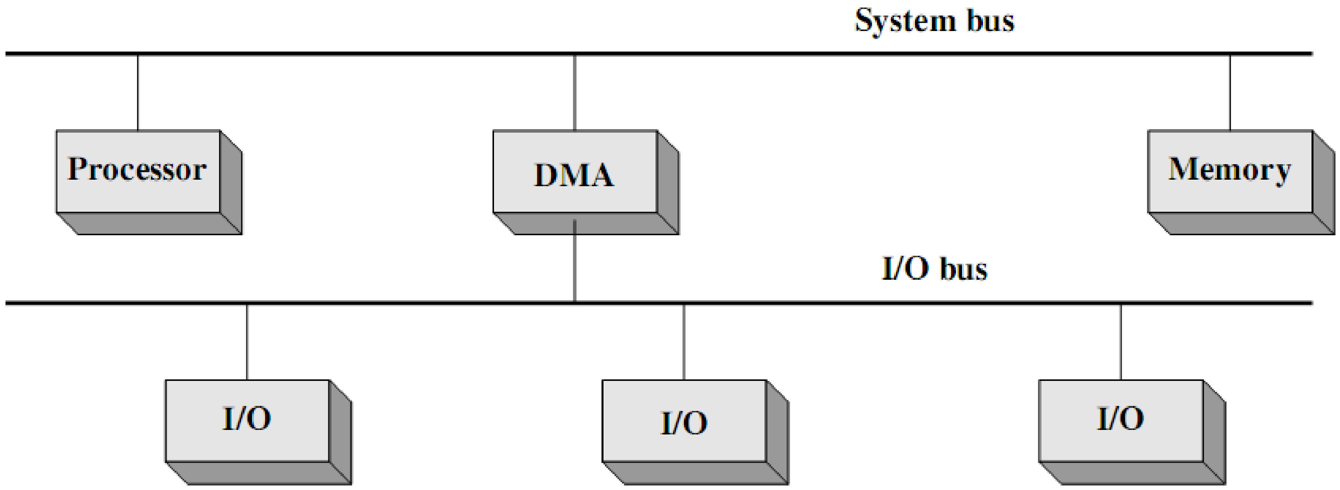
DMA Configurations (2)

- Features: single-bus, integrated DMA controller.
- Controller may support more than one device.
- Each transfer uses system bus once.
  - DMA controller ➔ memory.
- CPU is suspended once per transfer.
- DMA controller has one or more I/O interfaces.
- Efficient yet expensive!



DMA Configurations (3)

- Features: separate I/O bus.
  - —Connecting all DMA-capable devices.
- Each transfer uses system bus once.
  - —DMA controller → memory
- CPU is suspended once.
- DMA controller has only one I/O interface.
- Easily expandable configuration.



Evolution of I/O function

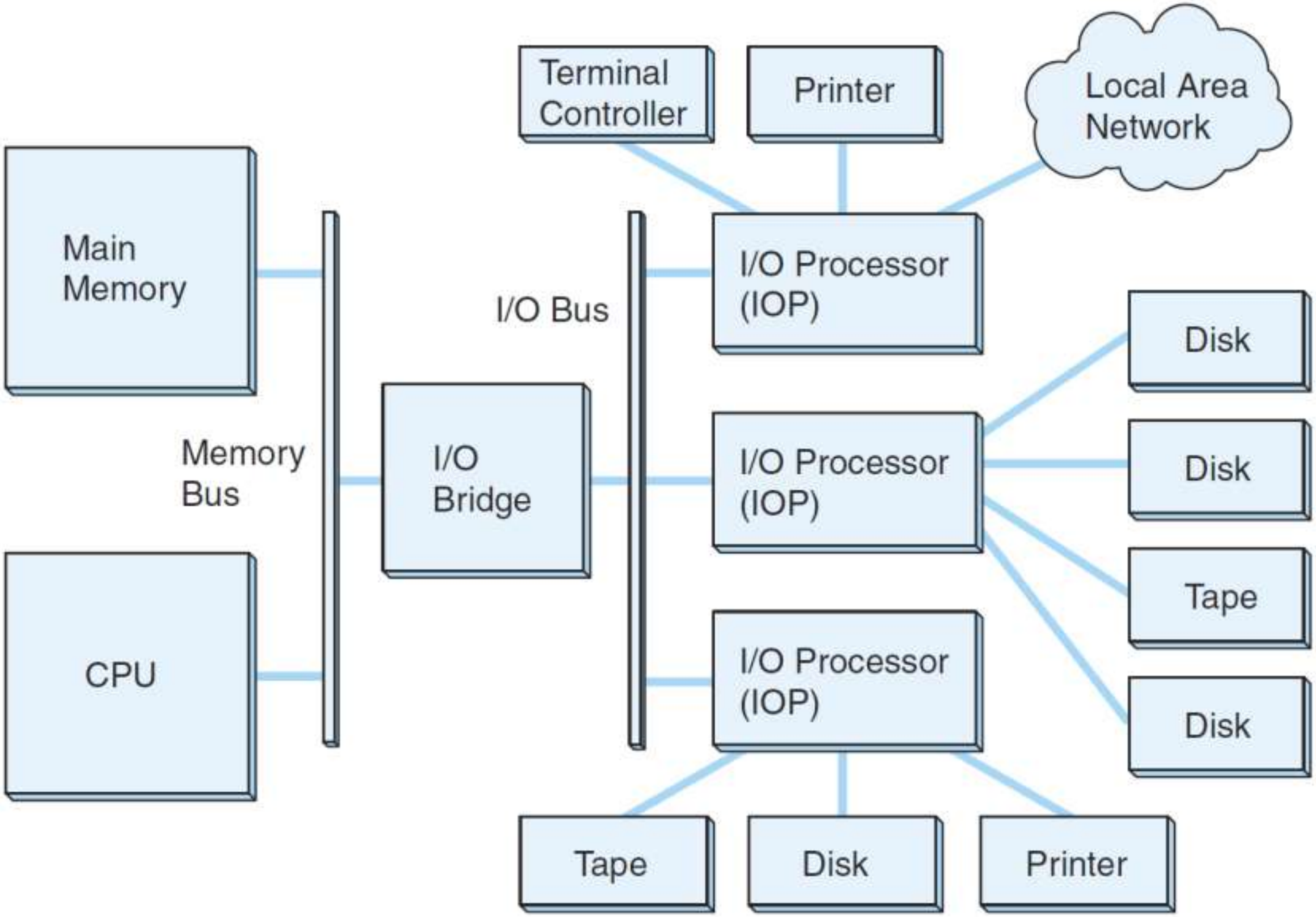
1. No I/O module.
   - CPU directly controls i/o device.
2. I/O module responding to CPU.
   - Programmed I/O.
3. I/O module interrupting CPU.
   - Interrupt-driven I/O.
4. I/O module accessing memory.
   - Direct-memory access (DMA).
5. I/O module executing program.
   - I/O channel.
6. I/O module executing program from local memory.
   - I/O processor, .e.g., GPU.

- **NOTE**: On occasions, no distinction is made between the terms: I/O channel and I/O processer!!!

I/O Channels

- I/O channels:
  - an extension to the DMA concept
  - able to execute I/O instructions
- I/O channel equipped with special-purpose processor, referred to as I/O processor (IOP)!!!!
- CPU instructs I/O channel to do the transfer
  - I/O processor fetches and executes I/O program from memory.
- I/O channel does the entire transfer.
- Improves speed
  - Takes load off CPU.
  - Dedicated processor is faster.

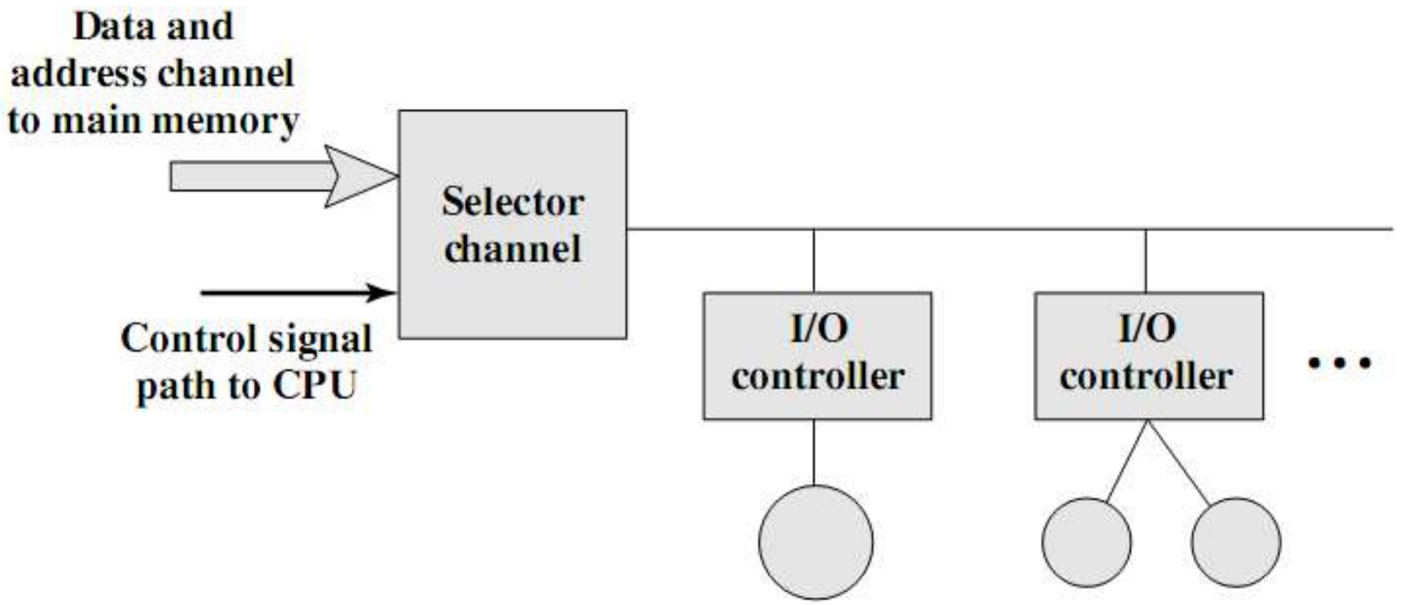
A Channel I/O Configuration



• **“The Essentials of Computer Organization and Architecture”, Null and Lobur**

I/O Channel Architecture – Selector

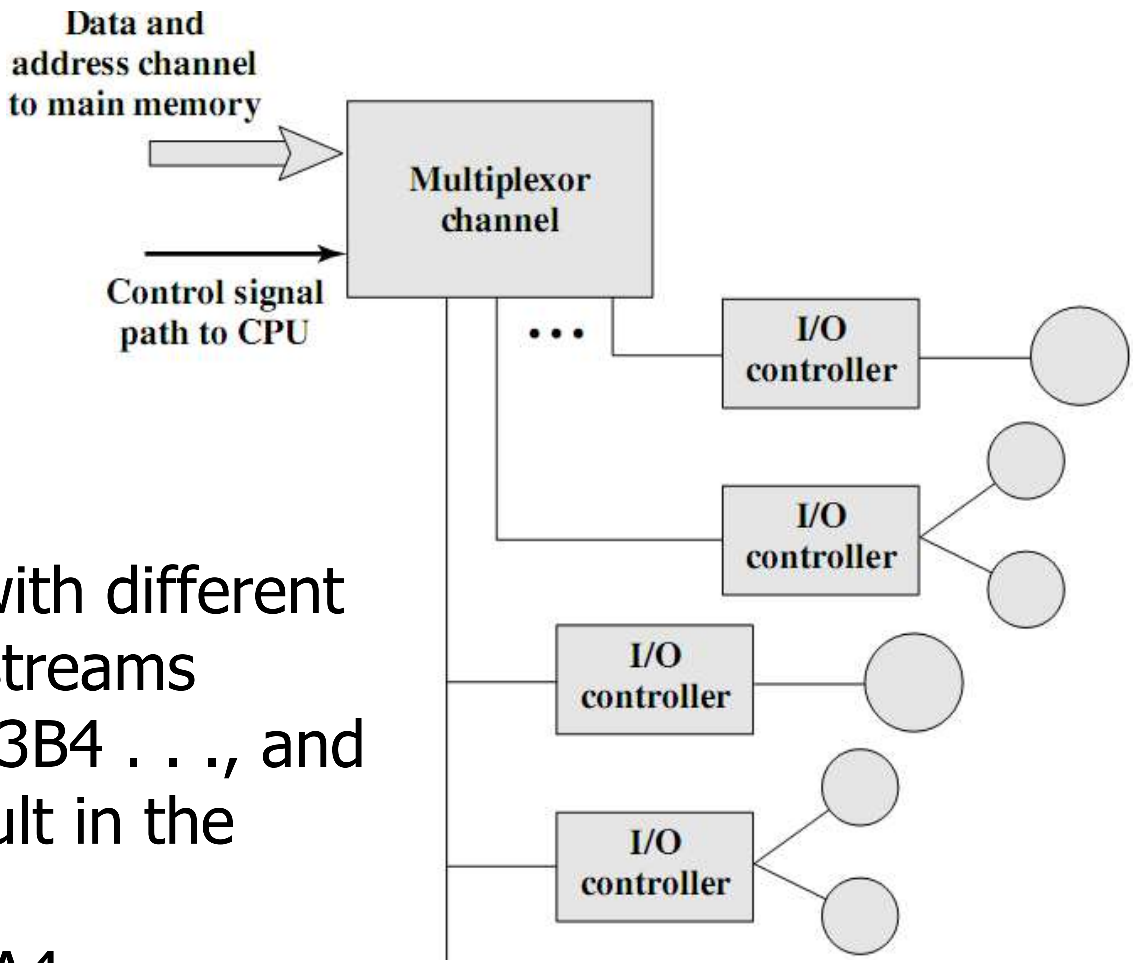
- Channel controls multiple high-speed devices.
- At any given time, channel is dedicated to data transfer with only one of these devices.
- Each device/set of devices is/are handled by a controller (I/O module).



Data and address channel to main memory
Selector channel
Control signal path to CPU
I/O controller
I/O controller
• • •

I/O Channel Architecture – Multiplexor

- Handles multiple low-speed devices at the same time.
- A byte multiplexor accepts or transmits characters as fast as possible to multiple devices.
- Example: 3 devices with different rates and individual streams A1A2A3A4 ..., B1B2B3B4 . . ., and C1C2C3C4 might result in the character stream A1B1C1A2C2A3B2C3A4 ...



Reading Material

- Stallings, Chapter 7:
  - Pages 222-237
  - Pages 240-243
  - Pages 246-248