CSE 4305
Computer Organization and Architecture

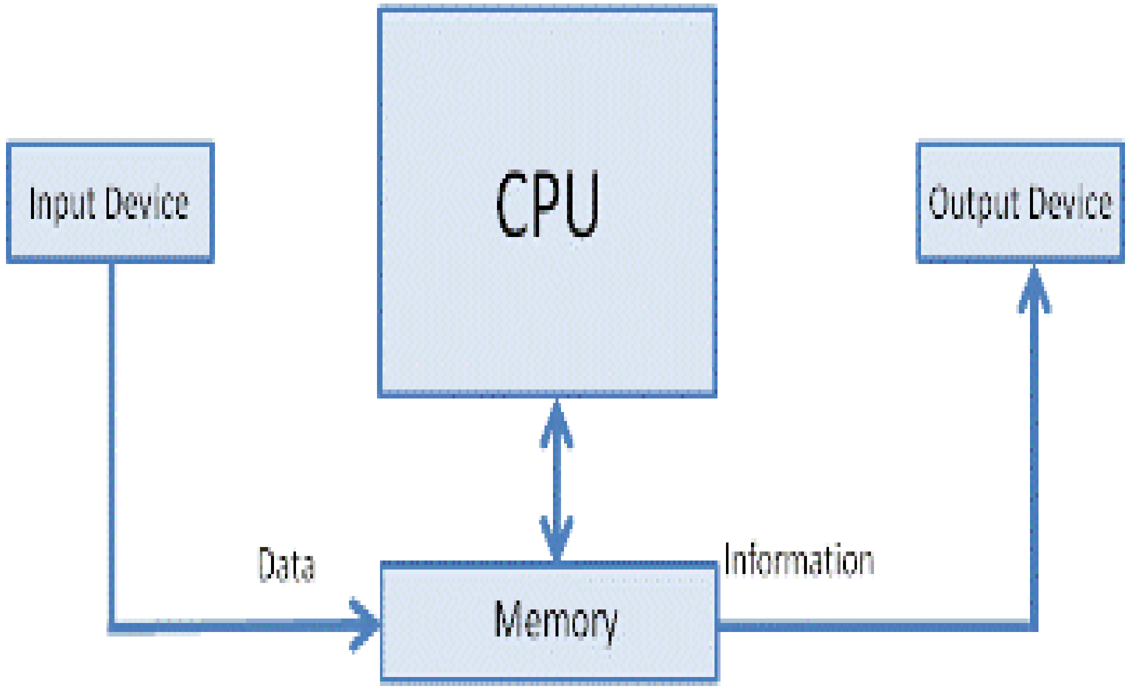
Input/ Output

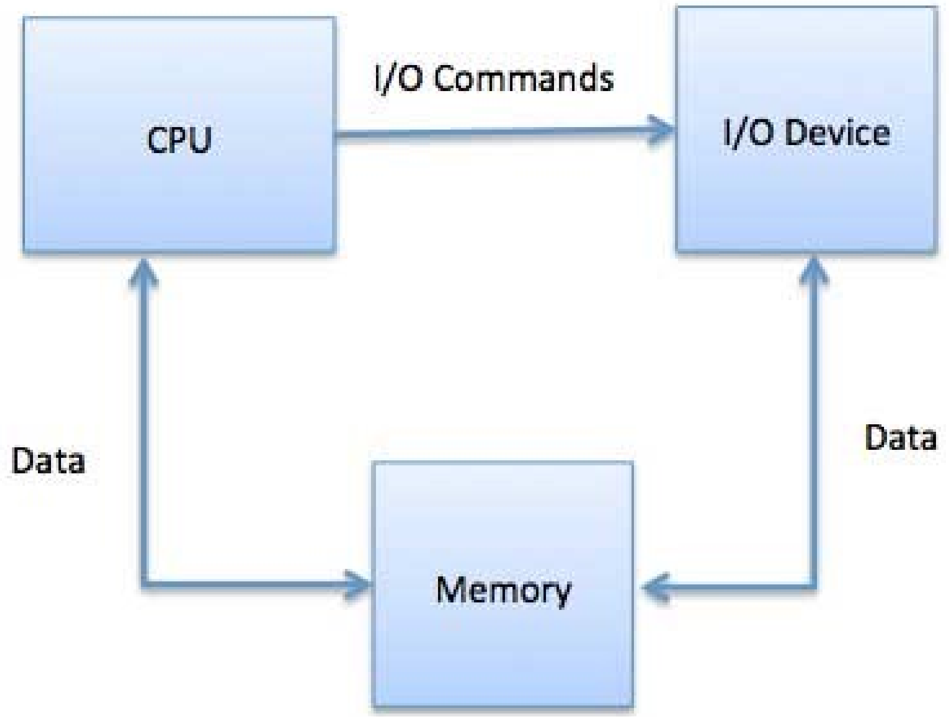
Course Teacher: Md. Hamjajul Ashmafee

Lecturer, CSE, IUT

Email: ashmafee@iut-dhaka.edu

I/O

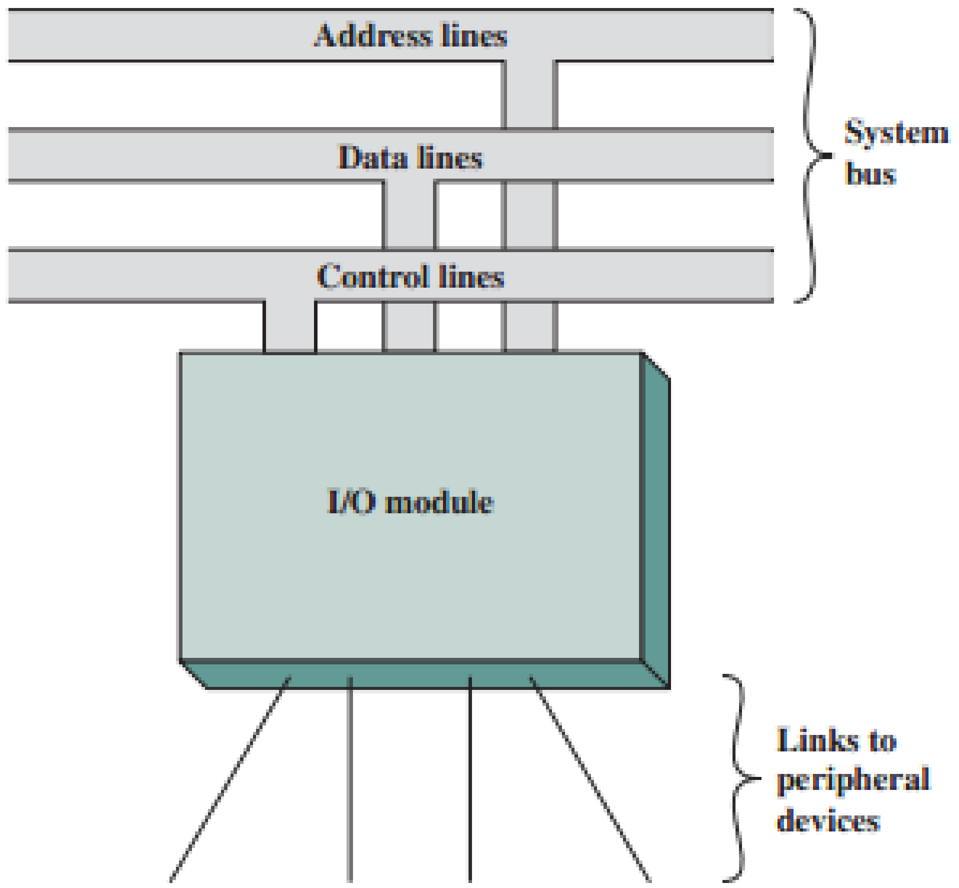

Input Device
CPU
Output Device
Data
Memory
Information


CPU
I/O Commands
I/O Device
Data
Memory
Data

I/O Modules

- **I/O modules** – third key element of a computer system other than *processor and memory* – builds **interface** to system bus or central switch and **controls peripheral devices**.
- Not simply the wire connecting devices with the system bus, **rather, contains logic for communication function between peripheral and system bus**
- **Why we cannot connect peripherals directly to the system:**
  - A wide variety of peripherals with various methods of operations
  - Data transfer rate of peripherals is often much slower than processor/ memory
  - On the other hand, some peripherals has faster data transfer rate (clock)
  - Peripherals may use different data formats and word lengths than the computer
  - I/O modules have **two major functions** – interface to the processor and memory via *system bus or central switch*, and interface to peripherals by *data links*

Model of I/O Modules



External Devices

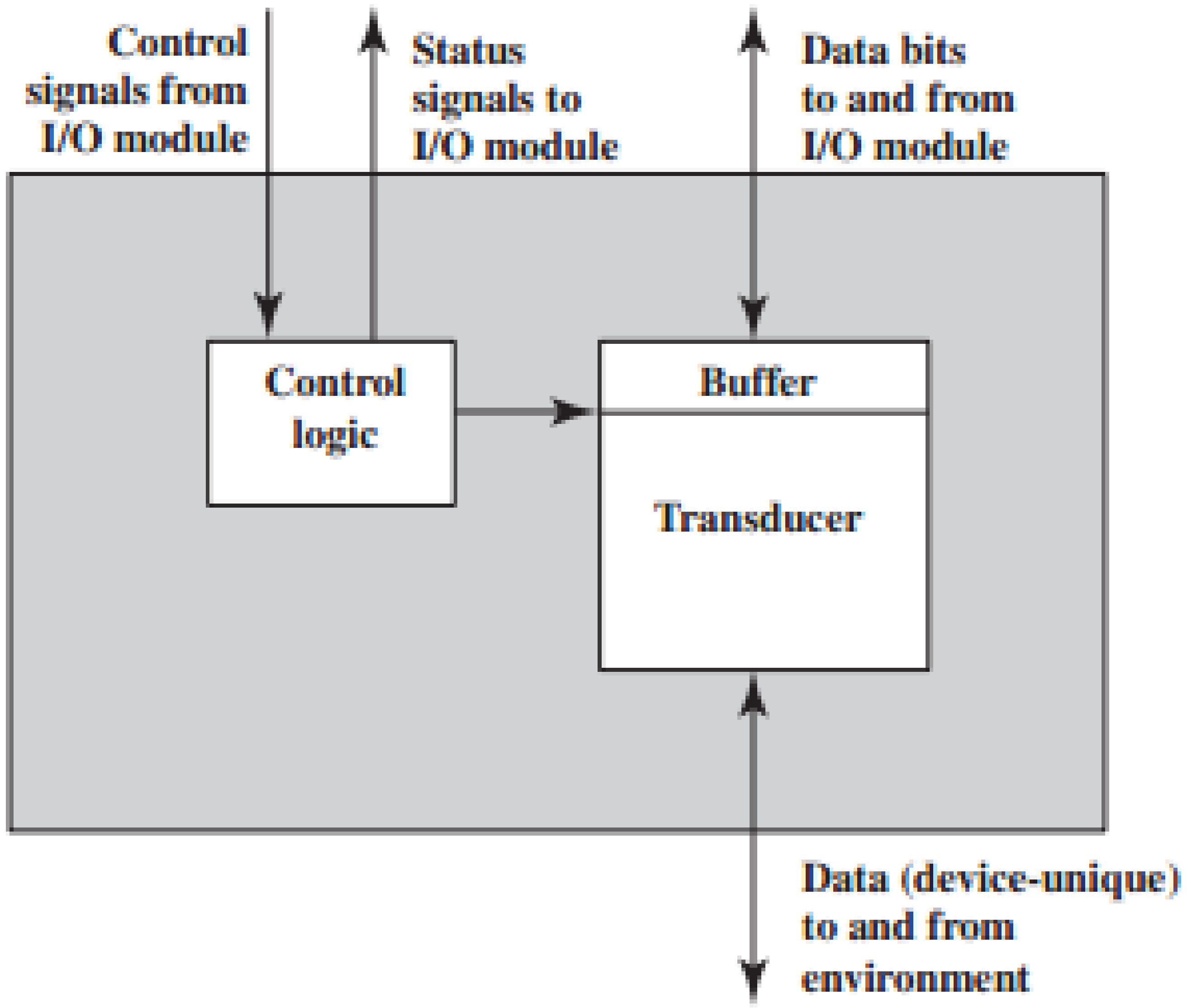
- **I/O operations** are performed through a varieties of **external devices** - also named as **peripheral devices/ peripherals**
- They **exchange data** between external environment and the computer
- External device attaches to the computer **by a link to an I/O module** – **link** is used to **exchange control, status, data** between the I/O module and the external devices
- **Classification** of external devices based on their interactions:
    - **Human readable:** communicate with user – video display, printer
    - **Machine readable:** communicate with equipment – magnetic disk, tape, sensors, actuators
    - **Communication:** communicate with remote devices – router, switch, terminal
- Disk and tape system – from a **functional** point of view, it is part of memory hierarchy – from **structural** point of view, it is controlled by I/O modules

External Devices…

It has several components:

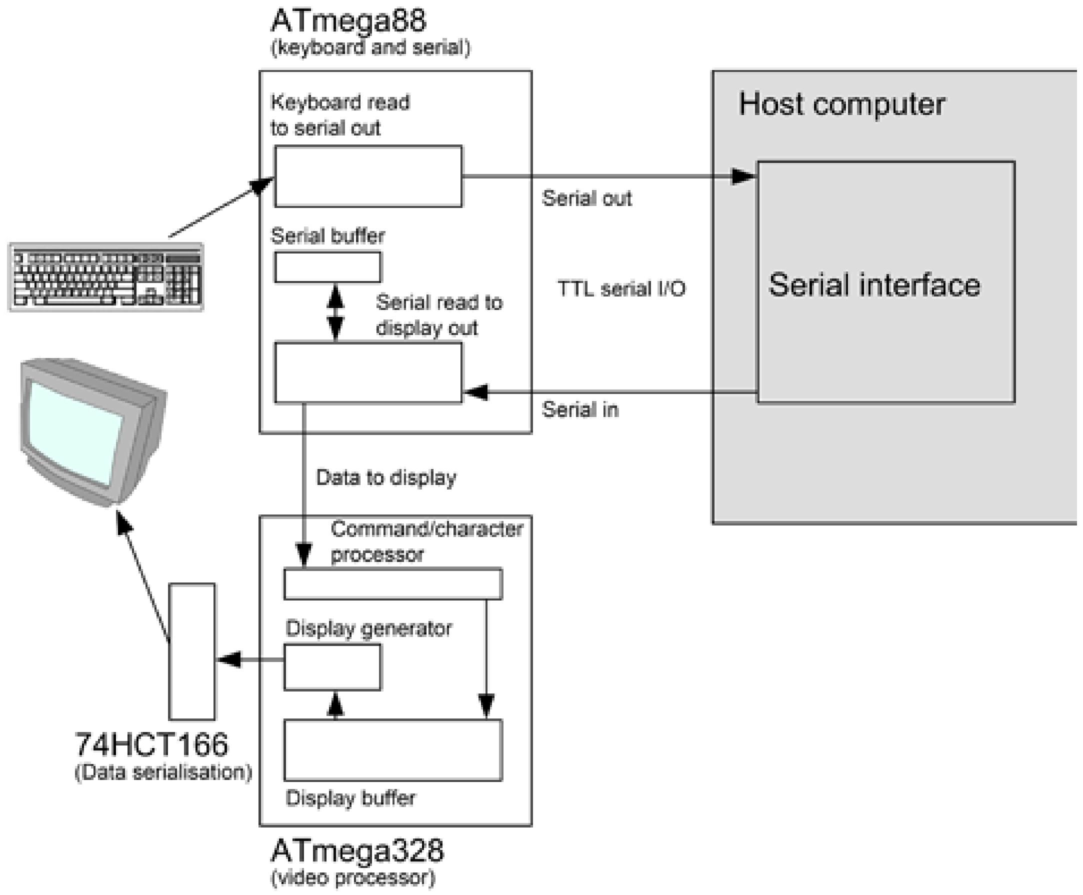
- **Interface – to communicate** with the I/O modules – in the form of control, data and status signals
  - **Control signal** – determine the function that the device will perform – INPUT/READ, OUTPUT/WRITE
  - **Data** – a set of bits to send or receive
  - **Status signal** – indicate the state of the device – READY/ NOT READY
- **Control Logic** – associated with the device, **control the operation** of the device according to the I/O module
- **Transducer –**
  - **Convert** the data from electrical to other forms of energy during output
  - **Convert** the data from other forms of energy to electrical during input
- **Buffer** - to hold data temporarily during data transfer – **size** of 8 to 16 bits for serial device, but for block oriented device (e.g. disk drive) it may be larger

Block Diagram of External Devices


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

Example: Keyboard - monitor

- User provides **input through the keyboard**
- Input character can be **displayed on the monitor**
- **Basic unit of data exchange is character** – a code is associated with that character of 7/8 bits in length – IRA, ASCII code
- **Two kinds of characters** – printable and control



Example: Disk Driver

- Contains electronics **for exchanging data, control, and status signals** with an I/O module (as it itself is an I/O device)
- Also contains electronics **for controlling the disk read/write mechanism**.
- In a **fixed-head disk**, the transducer is capable of **converting between** the **magnetic patterns** on the moving disk surface and **bits in the device's buffer**
- A **moving-head disk** must also be able to **cause the disk arm to move radially** in and out across the disk's surface.

I/O Modules: Module Function

- It allows the processor **to view a wide range of devices in a simple way**
- **Hides the details of an external device to the processor**, so that it can **function with simple read/ write commands** – <u>also leave some controlling works from processor interfere like rewind tape</u>
- It functions as:
  - Control and Timing
  - Processor Communication
  - Device Communication
  - Data Buffering
  - Error Detection

Module Function: Control & Timing

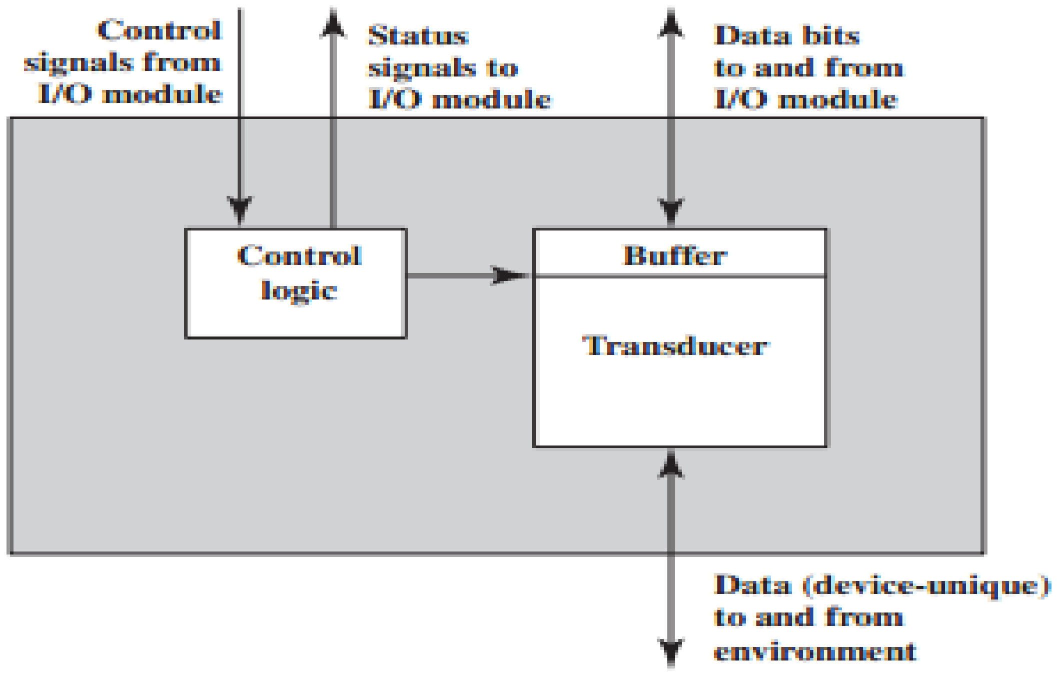
- Processor may communicate with one or more external devices **based on program's need** – **internal resources** like main memory, bus can be **shared**
- **Control and coordinate** the flow of traffic between internal and external devices.
- To transfer data from external device to processor
    - Processor **asks I/O module to check the status** of the attached device
    - I/O module **returns the device status**
    - If the device is **operational (ready)**, processor requests for data transfer by sending the command to I/O module
    - I/O module obtains data from external device
    - Data is transferred to processor from I/O module
- If **system bus is involved**, bus arbitration is also required

Module Function: Processor Communication

- **Command Decoding:** I/O module accepts commands from the processor as signal through **control bus** like READ SECTOR, WRITE SECTOR, SEEK track number, SCAN record ID for disk drive
- **Data:** Data can be exchanged between processor and I/O module through data bus
- **Status Reporting:** As peripherals are slow, it is required to know the status through I/O modules like READY, BUSY. There are also some other signals to **report various error conditio**n
- **Address Recognition:** Each I/O device has its address like main memory. I/O module recognizes one unique address for each peripherals

Module Function: Device Communication

- I/O module must be able to communicate with external devices through data, signal and status lines


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

Module Function: Data Buffering

- **As the data rate is very high with main memory and processor**, data are buffered in the I/O module and then **sent to the peripherals at their rate** – in opposite direction, data also buffered **not to tie memory/ processor with slower peripherals**
- I/O module must be able to **operate at both external device and memory at their own speeds**
- It also manages in the same **if the I/O device is faster in data rate**

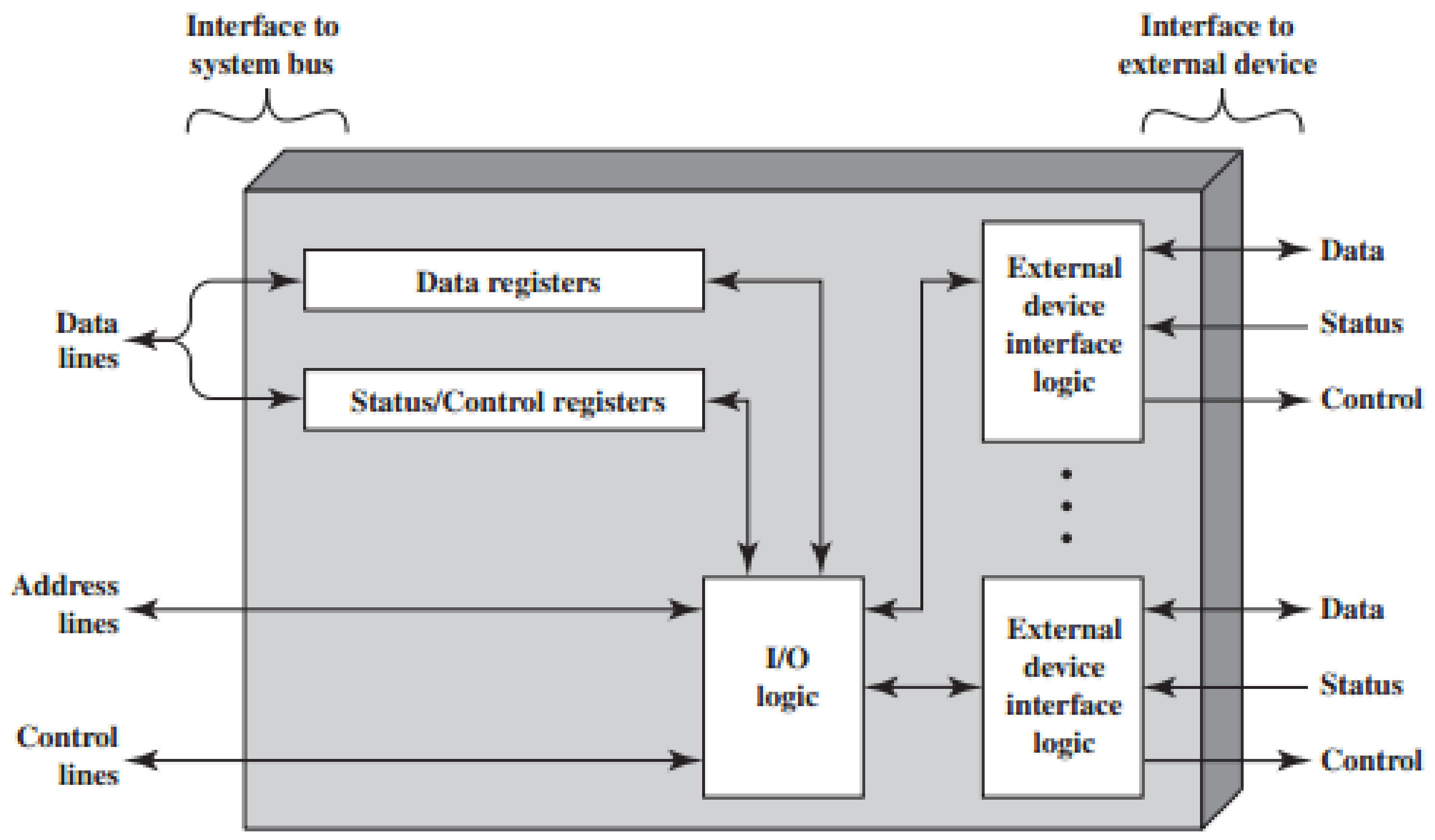
Module Function: Error Detection

- It **detects err**or and **reports to the processor**
- Different kinds of error are there:
  - **Mechanical and electrical malfunction:** reported by the device – paper jam, bad disk track
  - **Unintentional change in bit pattern**: during the data transmission
- Error detecting code is there to detect transmission error – parity bit, hamming code

I/O Module Structure

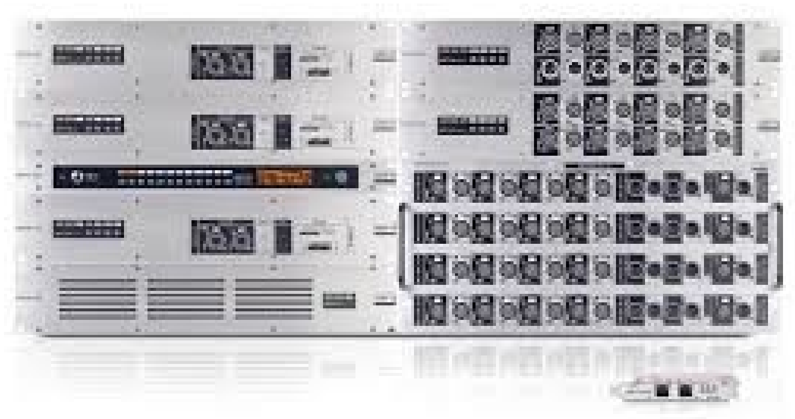
- The module connects to the computer through a set of signal lines – system bus lines
- It contains:
  - **Data register:** To buffer transferring data in one or more registers
  - **Status register:** To provide current status information.
  - **Control register:** To accept detailed control information from the processor through control lines
  - **I/O logic:** Interacts with the processor via a set of control lines
  - **External device interface logic:** Interface with each external devices that it controls

I/O Module Structure…



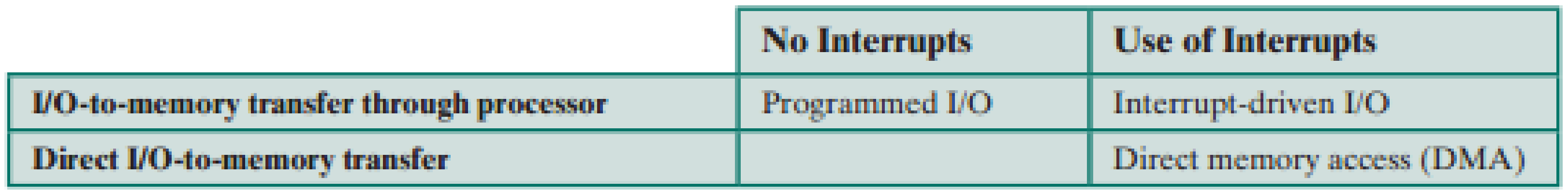
I/O Module Structure…

- **Sometimes it takes the detailed processing burden** – presents high level interface to the processor – **referred to as I/O channel or I/O processor** – used in mainframes
- Others **primitive I/O modules** which requires detailed control – I/O controller/ device controller – used in microcomputers



I/O operations' Techniques

- **Three** techniques:
  - **Programmed I/O** - data exchanged *between processor and I/O module* executing program that gives the direction (like sense status, send read/ write command) to control I/O operation – processor waits until the I/O operation is complete
  - **Interrupt Driven I/O** – the processor issues an I/O command and continues to execute other instructions. When *I/O module finishes its work latter, it interrupts the processor*.
  - **Direct Memory Access (DMA)** – for previous techniques, processor is responsible for data transfer. But for alternative, DMA exchanges data between I/O module and main memory directly, with processor involvement

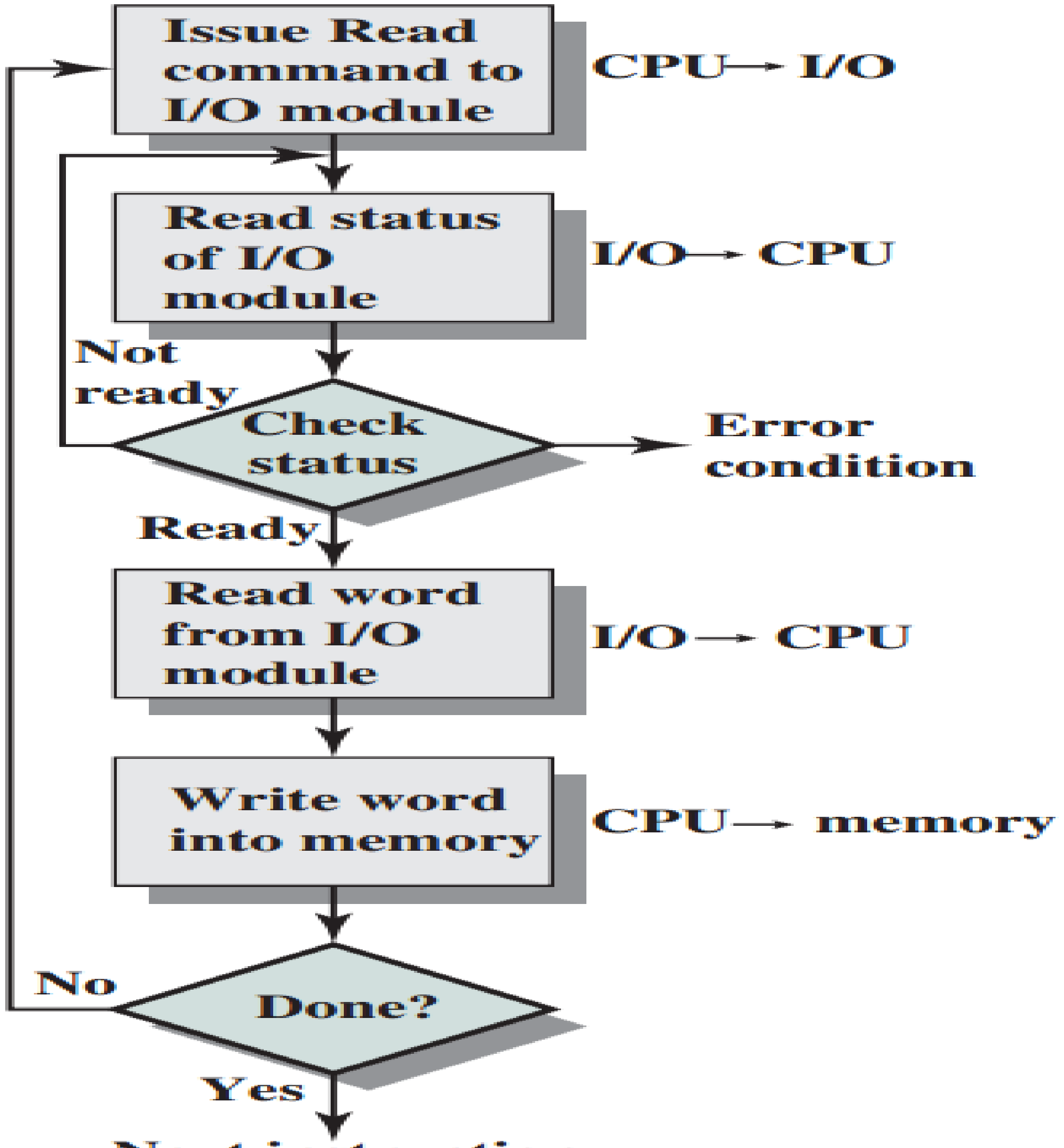


	No Interrupts	**Use of Interrupts**
I/O-to-memory transfer through processor	Programmed I/O	Interrupt-driven I/O
Direct I/O-to-memory transfer		Direct memory access (DMA)

Programmed I/O

- When a processor executes a program and encounters an instruction of I/O, it executes that instruction issuing an I/O command to appropriate I/O module –**I/O module** performs the required task and sets the bits in I/O status register – no further action to alert the processor; not interrupt the processor – it is responsibility of the processor to check the status periodically until it finds the operation is complete
- **I/O commands** – issued by the processor to I/O module
- **I/O instructions** – executed by the processor
- **Example:** While reading data, the processor will remain in status checking cycle, until it determines that data is available in I/O module's data register
- **Disadvantage:** Time consuming that keeps the processor busy needlessly

Programmed I/O…



(a) Programmed I/O

I/O Commands

- To execute an I/O instruction, the processor issues address, specify particular I/O module and external devices and I/O commands
- **4 kinds of I/O commands** received by I/O module
  - **Control**: Used to activate the peripherals and tell what to do. Example: rewind the magnetic tape, move forward one record
  - **Test:** Used to test various status conditions associated with an **I/O module and peripherals**. Example: the interested peripheral is powered on/ available or not, last I/O operation is completed or not, any error or not
  - **Read:** Causes an I/O module to obtain data from peripherals place it in internal buffer and then place it on system data bus
  - **Write:** Causes an I/O module to take data from system data bus and transmit to the peripherals
- As **each I/O device has its own identifier/ address**, the I/O commands contain the address of the desired device through the address line interpreted by the I/O module

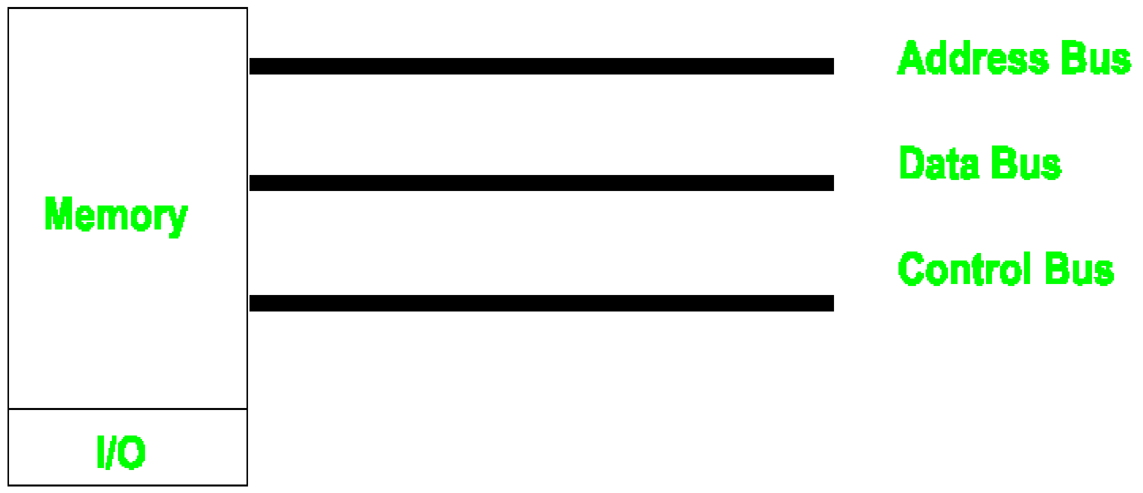
I/O Instructions

- **Close relation between** the I/O instructions fetched from memory by the processor and the I/O commands issued to the I/O module by the processor to execute the instructions – easy **one-to-one mapping** from I/O instructions to I/O commands
- **Form of the instruction** depends on external devices – IN, OUT, INS, OUTS

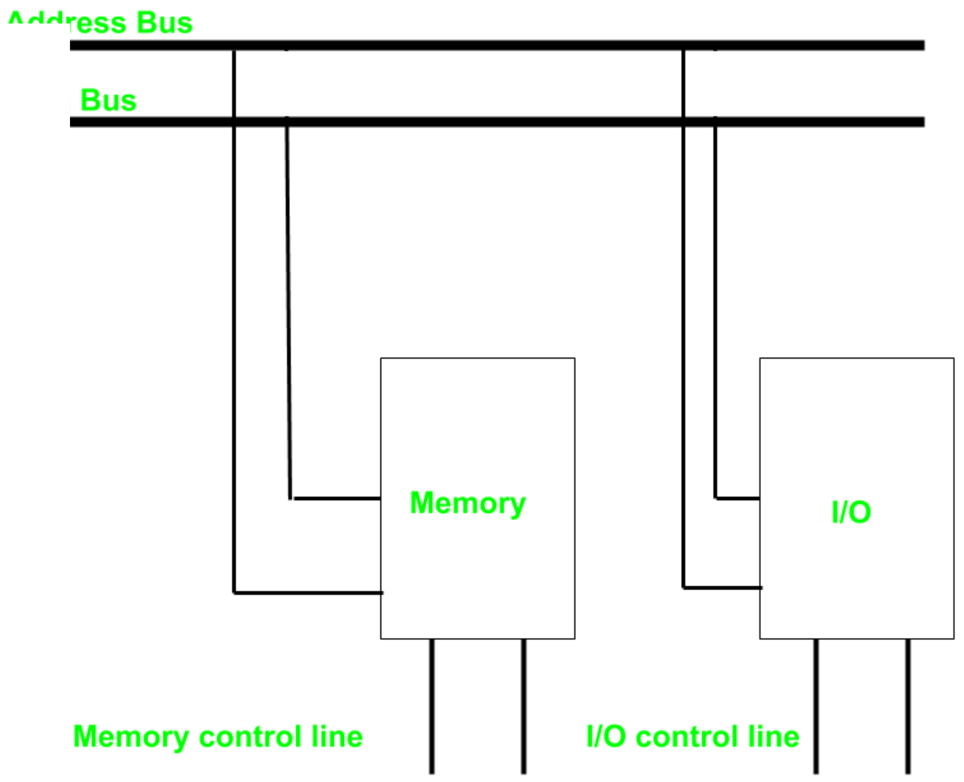
Modes of Addressing an I/O device

- When processor, main memory and I/O **share the common bus**, two modes of addressing are possible:
  - **Memory Mapped (I/O)** – single address space for memory locations and I/O devices – use same registers both for I/O modules and memory location – same instructions to access both memory and I/O devices (read/write) – range of addresses will be divided for memory locations and port addresses
  - **Isolated (I/O)** – separate address spaces for memory and I/O devices – different instructions to access them (read/write) – Special I/O commands to access I/O ports – also known I/O mapped I/O

Modes of Addressing…

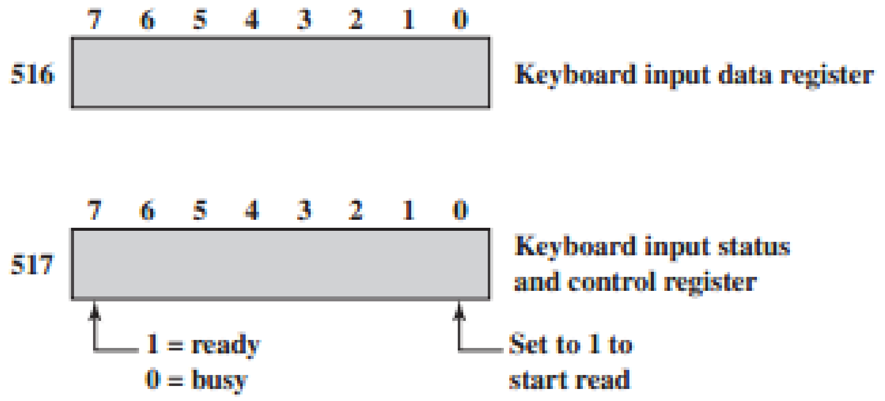


Memory mapped I/O



Isolated I/O

Modes of Addressing…



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

Keyboard port address = 5

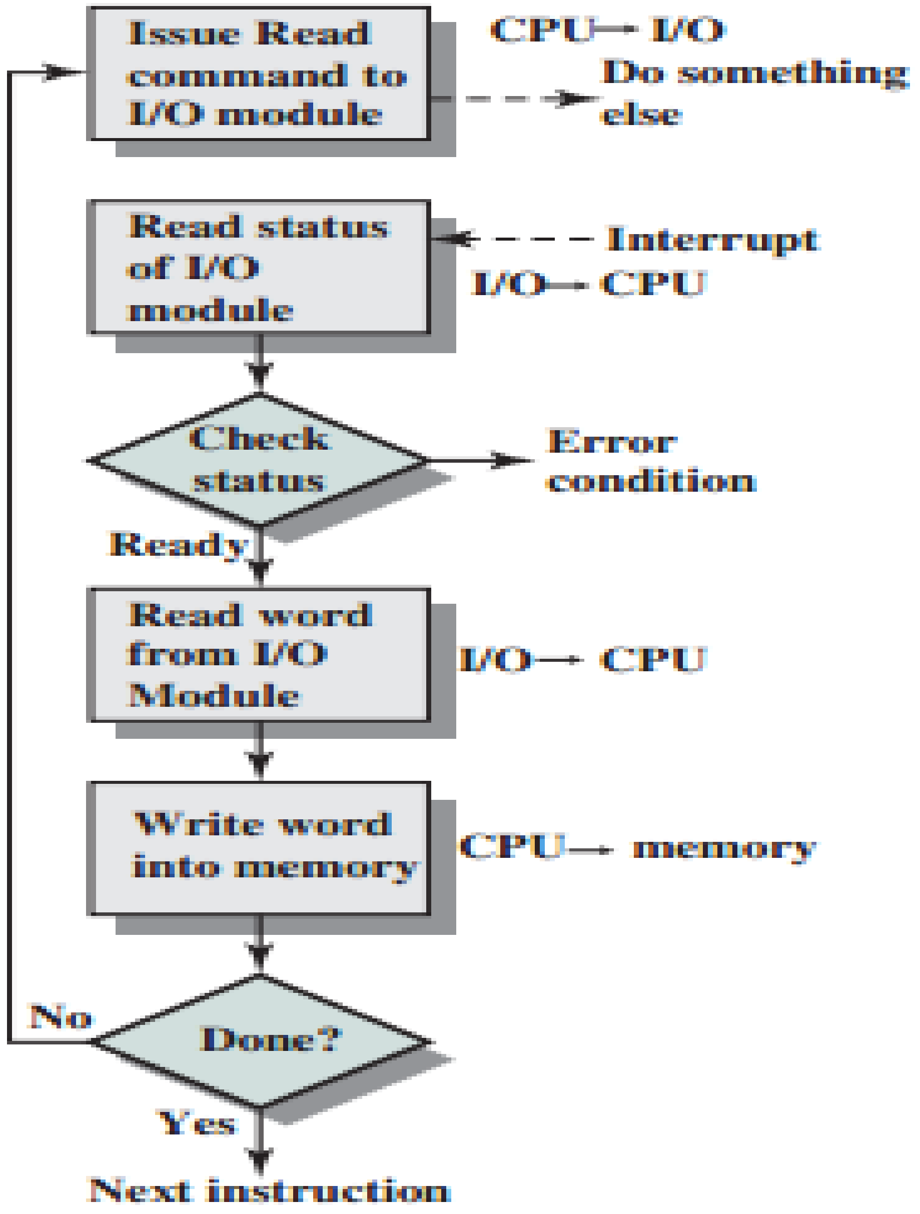
Interrupt Driven I/O

- The **main drawback** of **Programmed I/O** is processor has to wait for long time to receive or send data completely – processor repeatedly interrogate the status of the I/O module – results in **processor's performance degradation**
- **An alternative** – processors issues an I/O command to a I/O module and does some other useful tasks – I/O module will interrupt processor when it is ready to transfer data – processor then executes data transfer (from I/O module to memory or vice versa) and resumes and former task
- Interrupt driven I/O is **more efficient** than Programmed I/O as it eliminates the needless waiting
- But the **disadvantage** of this I/O method is for each transfer of data it involves processor

Interrupt Driven I/O…

- **From the I/O module's view**, for input, it will get READ command from processor – I/O modules reads data from the associated peripheral into data register and interrupt the processor – I/O modules wait until the request from processor – when the request is made by the processor, it will place that on the system bus
- **From the Processor's view**, for input, when it executes an I/O instruction, it will issue a READ command – then it goes off and does something else – after finishing each instruction cycle, it checks for interrupt – when an interrupt from I/O module is occurred, it switches its current context of running program and responses to that interrupt – after finishing the interrupt handler (read data word from I/O module), it restores the context of the last running program

Interrupt Driven I/O…



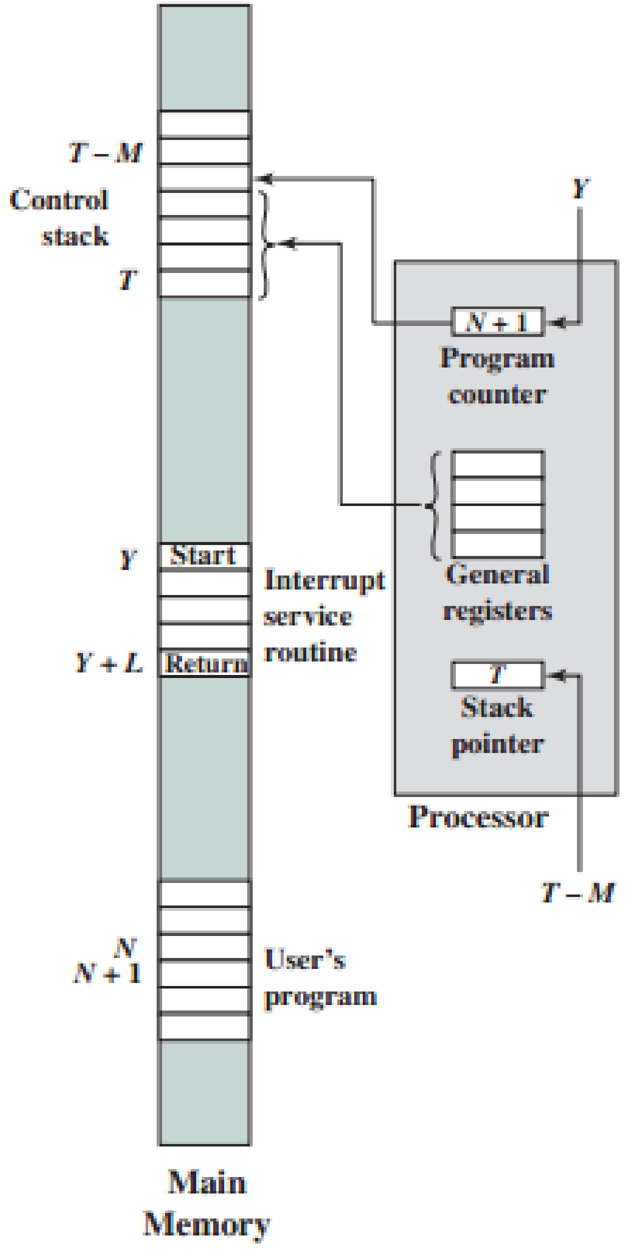
(b) Interrupt-driven I/O

Interrupt Processing Finishing An I/O Operation

- **Role of Processor** in Interrupt driven I/O in more details when an I/O device **completes** as I/O operation:
  1. The device (I/O module) **issues an interrupt signal** to the processor
  2. The processor **finishes current instruction execution of other program** before responding that interrupt
  3. The processor now **tests for an interrupt to determine its existence**, and send an ACK signal to the I/O module to remove the interrupt signal
  4. The processor **switches its context** (storing current **P**rogram **S**tatus **W**ord (PSW) and PC in stack) **with minimum information**
  5. The processor now **loads the PC with the entry point of ISR** – may **single generalized** type ISR, **Device wise** ISR or **Interrupt based** ISR determined by processor with the help of Interrupt request or Interrupting device. Thus the control will be transferred to the ISR

Interrupt Processing…

- **When the processor executes the ISR**, following operations will be happened:
    6. Some information may be transferred from the interrupted program to ISR through the processor registers (relates the state of the executing program). So at the very beginning, **registers values also should be stacked by the ISR**. PC along with TOS will be got updated at this moment.
    7. **ISR processes the interrupt** – e.g. examines the state of I/O devices or other events that causes the interrupt, sends command, sends acknowledgements to the I/O devices

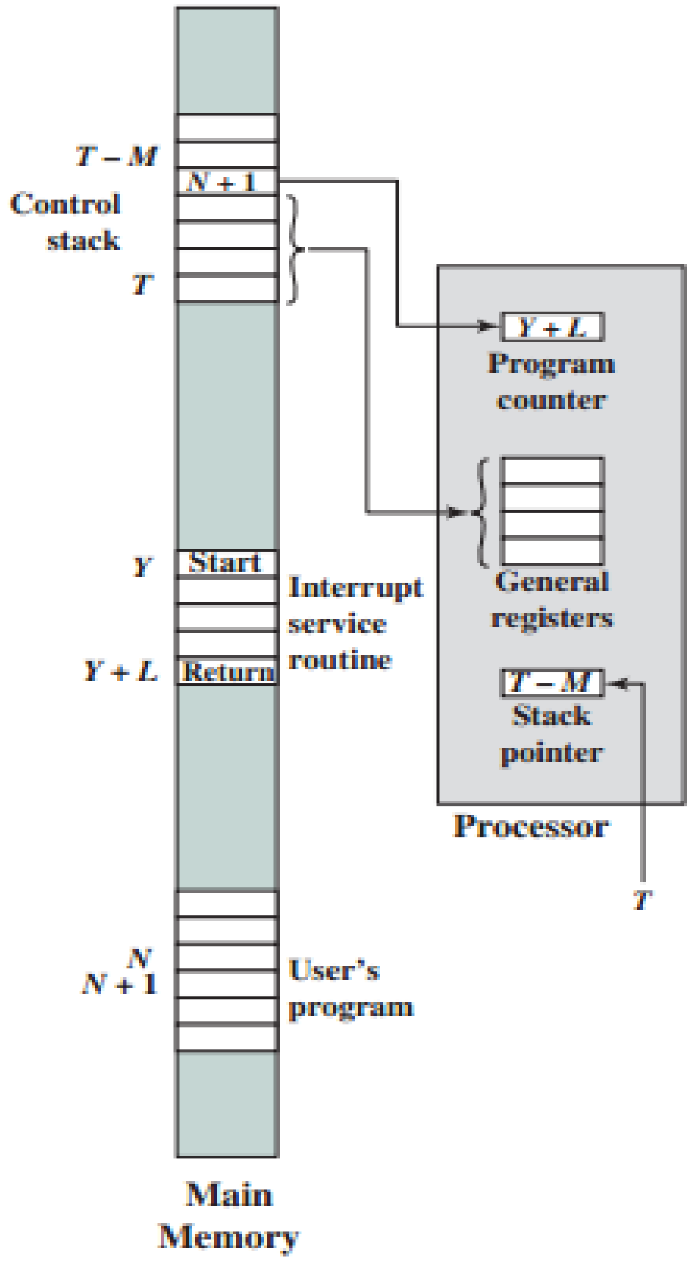


(a) Interrupt occurs after instruction at location N

Interrupt Processing…

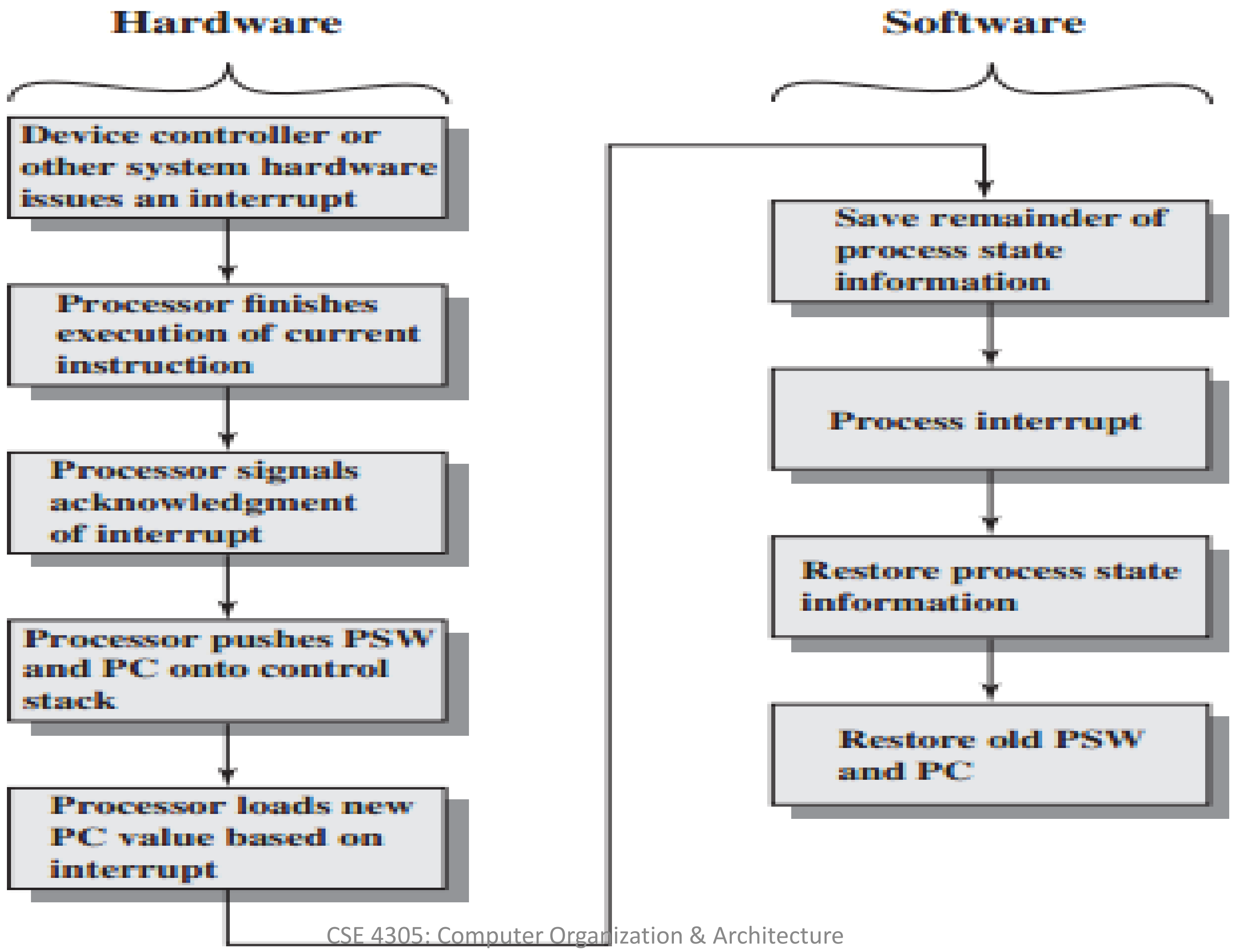
8. After completing the ISR, **the saved registers' values will be restored to the registers** from the stack (by the ISR or Processor)
9. Finally, **restores the PC and PSW from the stack**. As a result, the next instruction will be executed from previously interrupted program

Finally, **Interrupt is not a routine called by the User Program**, rather called at any time, at any place of the program (Unpredictable). So it is better to save all the state information about the interrupted program to resume properly later on.



(b) Return from interrupt

Interrupt Processing At a glance



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

ID I/O: Design Issues

- **Two design issues** arisen implementing ID I/O:
  - As there are various I/O modules, how the processor determines which device issued the interrupt? (**Device Identification**)
  - If multiple interrupts occurred simultaneously, how does the processor decide which one to process? (**Assigning Priority**)

Design Issues: Device Identification

- **Multiple Interrupt Line:**
  - Straightforward approach
  - Implement multiple interrupt lines between the processor and all I/O devices.
  - **Impractical** because it is not possible to dedicate more than a few bus lines or processor pins for interrupt lines
  - Although multiple lines are used, it is likely that each line will have multiple I/O devices attached to it where other techniques are used among themselves

Design Issues: Device Identification…

- **Software Poll:**
  - When the processor detects an interrupt, **it branches to an ISR** that polls each I/O module to identify the interrupting module
  - **Poll** – in the form of command (e.g. **TESTI/O**) and place the address of the particular I/O module in the address lines – I/O module **responses** if it sets interrupt
  - Alternatively, all the I/O module could have an addressable status register (along with data register). The processor then check the status register of each I/O module to identify the interrupting module.
  - After identifying the interrupting I/O module, the processor brunches to a device specific device service routine
  - Disadvantage: **Time Consuming**

Design Issues: Device Identification…

- **Daisy Chain:**
  - Kind of **hardware poll**
  - For interrupt, all I/O modules share a common interrupt request line
  - **Interrupt Acknowledge Line** is actually daisy chained through the I/O modules
  - When the processor senses an interrupt, it sends out an Interrupt Acknowledge - it is propagated through a series of I/O modules until it gets the requesting module
  - The requesting module responds through a **vector** placing on the data lines (module address/ unique identifier) – used by the processor to point appropriate ISR – also known as **vectored interrupt**

Design Issues: Device Identification…

- **Bus Arbitration:**
  - **An I/O module first gains the control of the bus** before it can raise the interrupt request line – <u>only one module is allowed to raise line at a time</u>
  - When the processor detects the interrupt, *it responds through the Interrupt Acknowledge Line* – <u>the requesting module then places its vector on the data lines</u>

Assigning Priority

- **With multiple lines**, the processor just picks the interrupt line with highest priority
- **With software polling**, the order in which modules are polled determines their priority
- The order of modules in a **daisy chain** determines their priority
- **Bus arbitration** employ the priority scheme to break the tie among the recipients

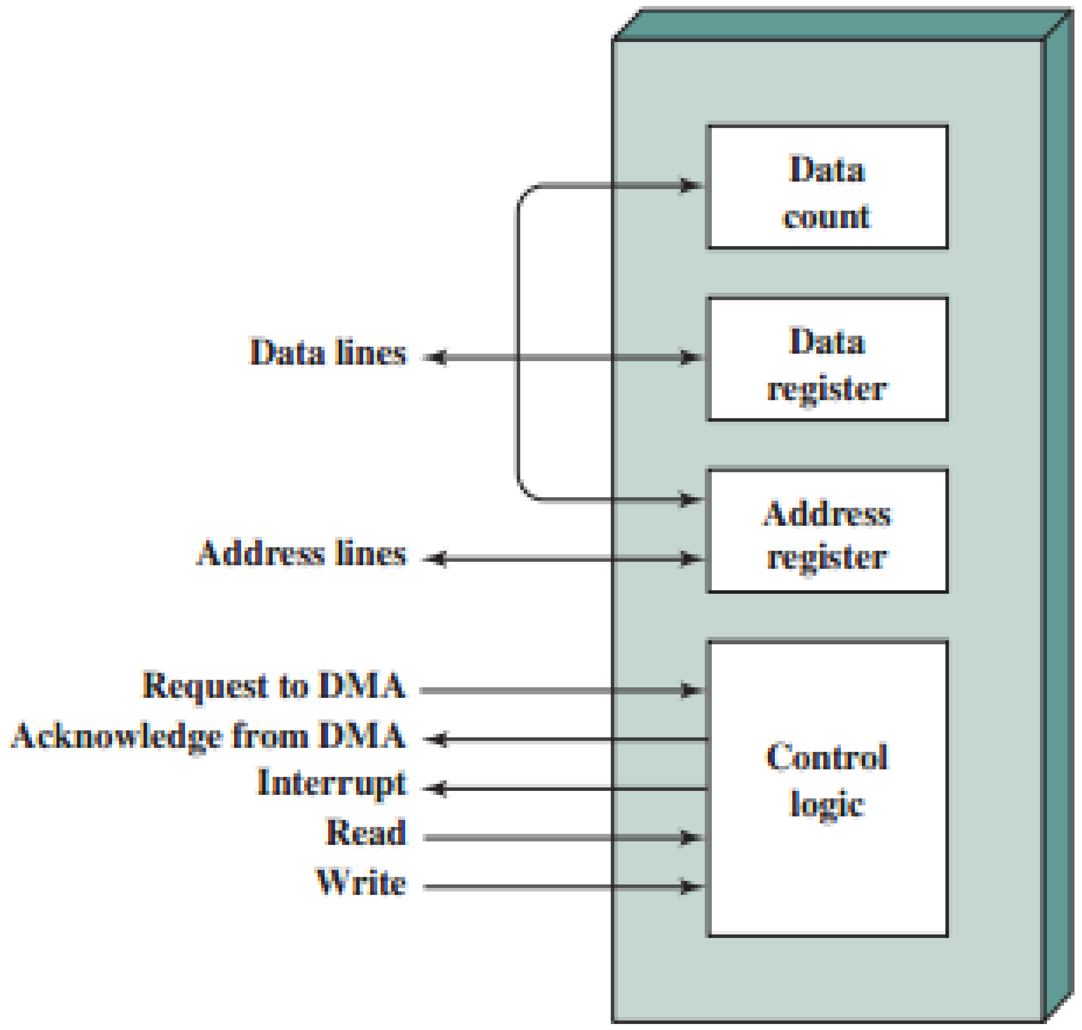
Problems with Previous Techniques

- **Drawbacks of Programmed I/O and ID I/O** – requires active intervention of the processor to transfer data and data transfer traverses a way through the processor.
- **Two inherent problems** they both faced:
  - I/O transfer rate is **limited by the speed** with which the processor can test and service a device
  - The processor is tied up in managing an I/O transfer (a number of instructions must be executed)
- **To transfer a block of data** – **Programmed I/O** dedicates the processor in the task of I/O operations doing nothing else – **ID I/O** frees up the processor for a moment to do I/O operations – **both have adverse impact**

Direct Memory Access

- **More efficient technique** – Direct Memory Access (DMA)
- Involves an **additional module** on the system bus – **DMA module** – performs the job of processor to transfer data (taking the control from processor)
- Transfer data to or from memory **over the system bus**
- So **DMA module uses the bus** when processor does not use it (expected) or forces the processor to suspend operation temporarily which one is more common (**cycle stealing**)

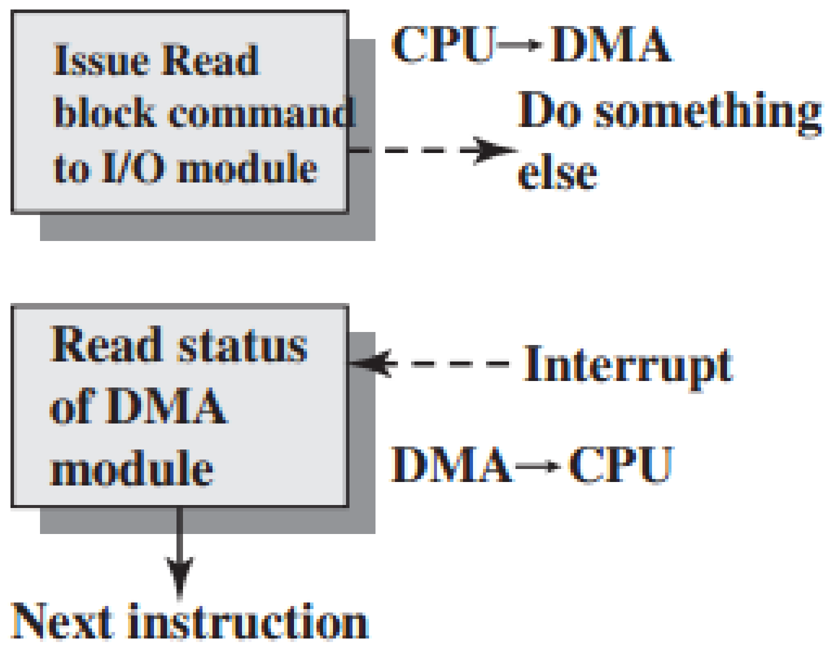
DMA Module Block Diagram



DMA Function

- To write or read a block of data, **the processor issues some commands to DMA module with the information of:**
  - Whether READ or WRITE request (using read/write control lines)
  - Address of I/O device involved (using data lines)
  - Starting location of memory to read or write (using data lines and store it in address register of DMA module)
  - The number of words to read/ write (using data lines and store it in data count of DMA module)
- **Then the processor starts another works** – delegates the I/O operation to DMA module
- **DMA module transfer entire block of data**
- After finishing transfer, **DMA module interrupts the processor** – processor is only involved at the beginning or ending of the transfer

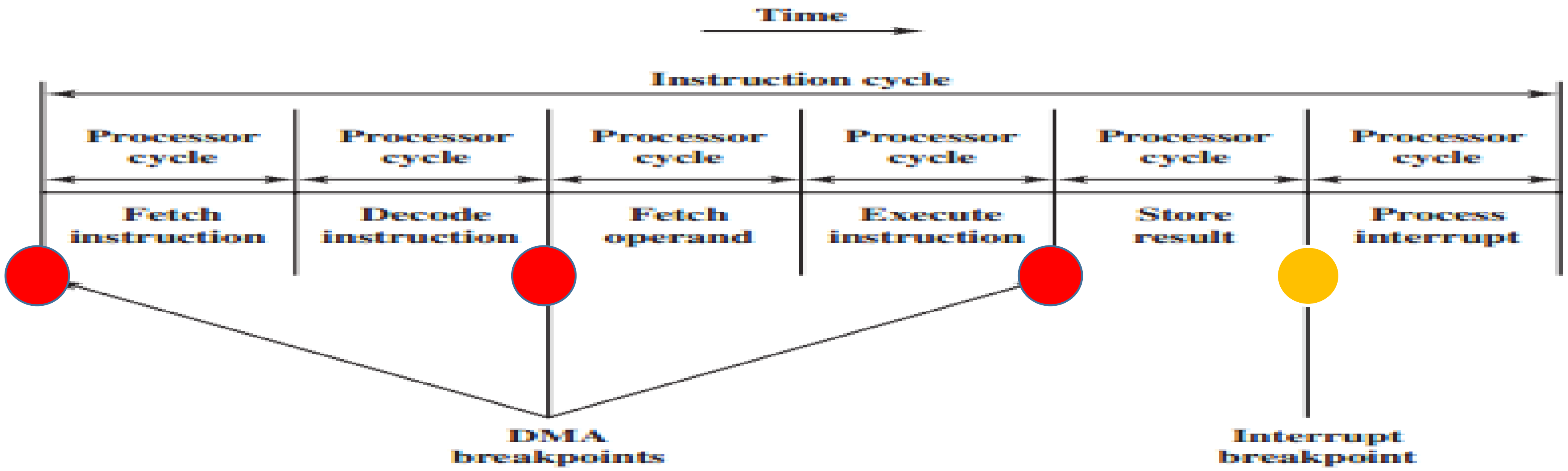
DMA Function…


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

(c) Direct memory access

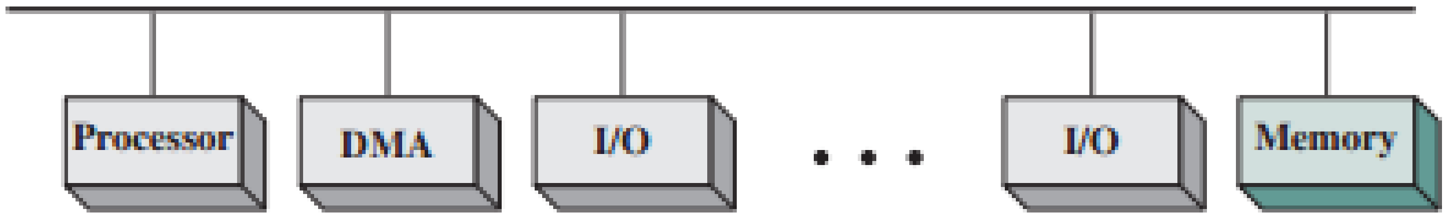
DMA and Interrupt Breakpoints

- **The processor is suspended just before it needs to use bus** – at that time DMA module transfers the block of data and returns the control to the processor – **it is not an interrupt** (no context switch) rather processor pauses for one bus cycle – **causes the processor slow**



DMA Configuration

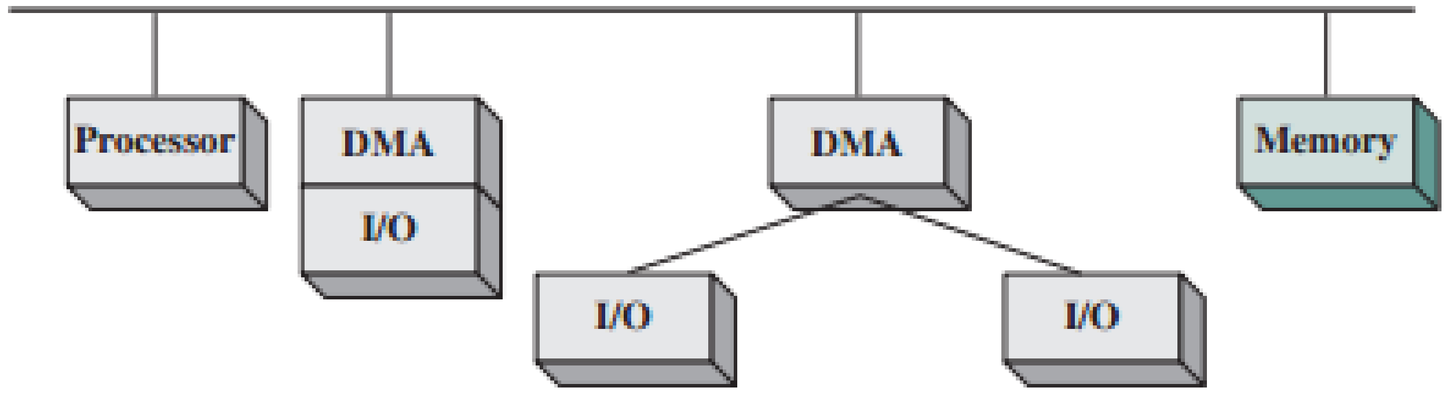
- **Single Bus – Detached DMA:**
  - All modules share the same bus system
  - DMA module uses Programed I/O by itself to exchange data
  - Though **inexpensive**, but **inefficient** as it requires two bus cycles to transfer data



(a) Single-bus, detached DMA

DMA Configuration…

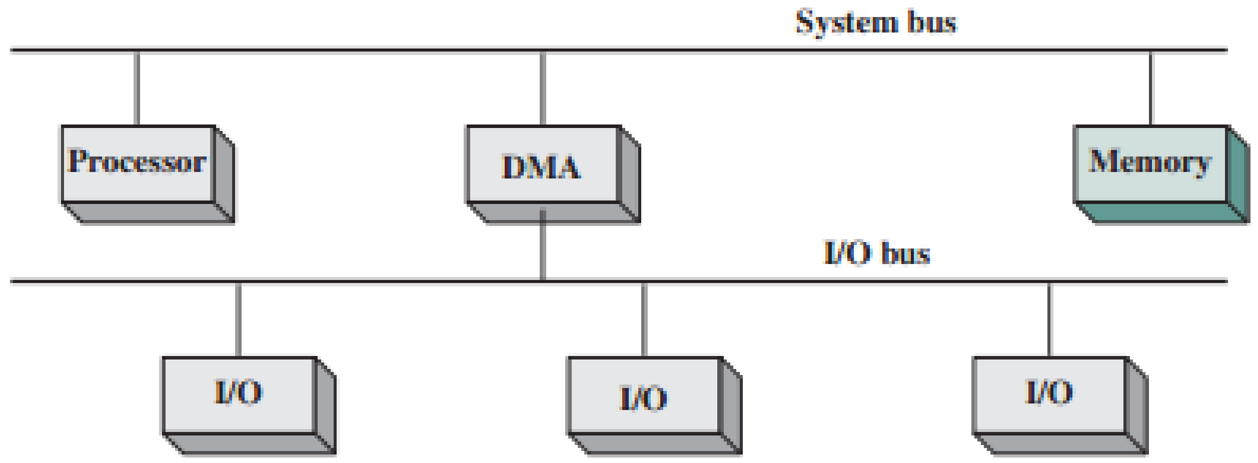
- **Single Bus – Integrated DMA-I/O:**
  - Integrates DMA and a group of I/Os together
  - No system bus among themselves
  - DMA controls one or more I/O modules
  - **Single bus cycle** to transfer data



(b) Single-bus, integrated DMA-I/O

DMA Configuration…

- **I/O Bus - Integrated DMA-I/O**
  - Developing the previous (Integrated DMA-I/O) concept – I/O modules connects with the DMA module using I/O Bus
  - It reduces the number of I/O interfaces to one for DMA module
  - Easily expandable
  - **Single bus cycle** to transfer data

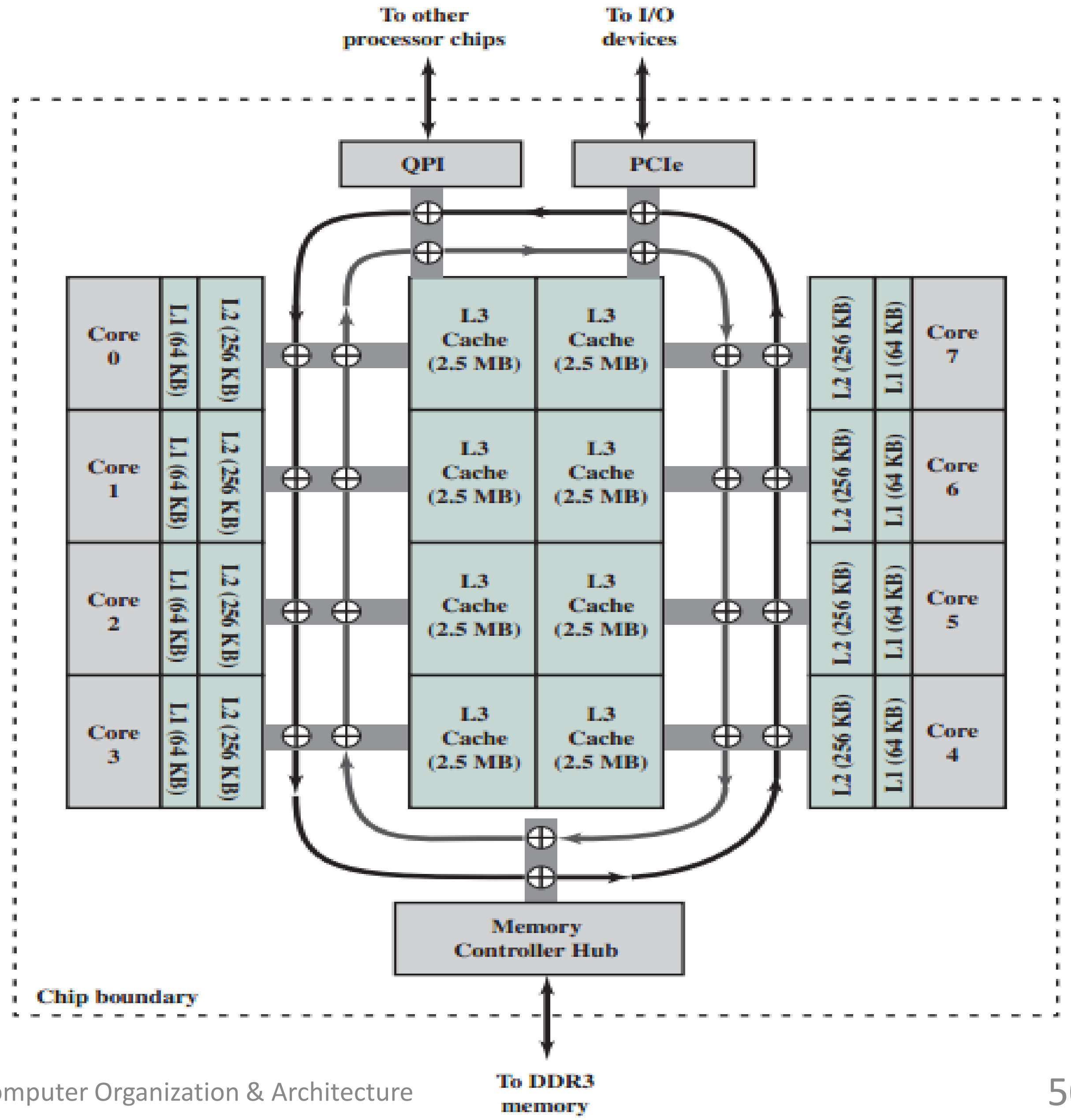


(c) I/O bus

Direct Cache Access

- Sometimes it is very hard to meet the increased data rate demand by DMA – e.g. Network traffic using high data rate using Ethernet, Wi-Fi
- So to enhance the performance, I/O devices have direct access to the Cache rather than main memory – Direct Cache Access
- Here I/O functions only have the access to that cache which is closest to the main memory – last level cache (L2 or L3 cache)
- DMA controller has the access to the shared cache
- Used for output operations only

Direct Cache Access…



Evaluation in I/O functions

1. CPU has direct controls to a peripheral devices (simple microprocessor controlled devices)
2. A controller or I/O module is added using programmed I/O without interrupt
3. Same configuration as in step 2 but nor interrupts are employed – CPU needs not to wait for I/O operations
4. I/O module is given access to memory via DMA
5. I/O module is enhanced to become a processor with specialized instruction set tailored for I/O
6. I/O module has a local memory (becomes a computer of its own) – a large set of I/O devices can be   controlled with minimal CPU involvement – used in communication with interactive terminals

I/O Channels & Processors

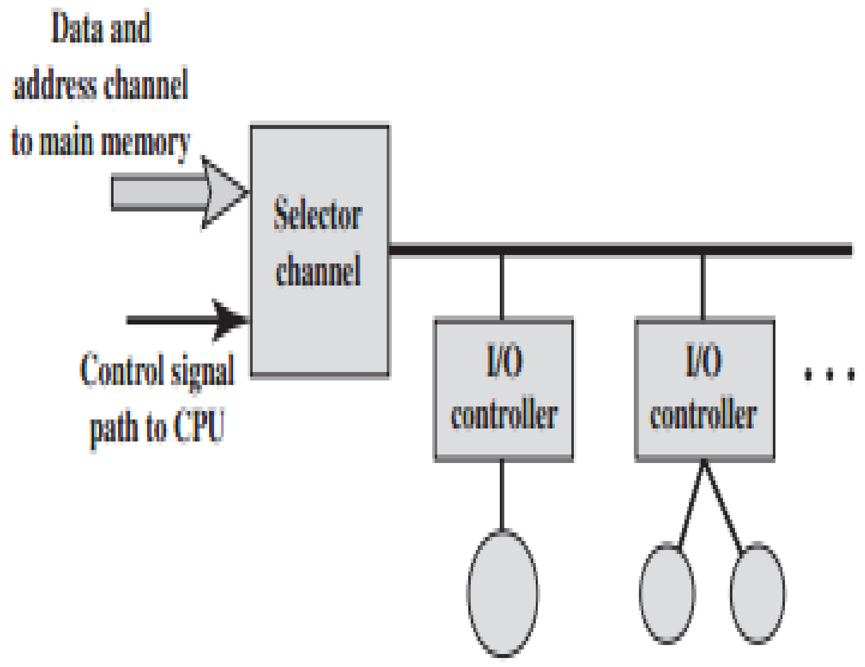
- As one proceeds, more and more I/O operations will be performed without CPU involvement – CPU will be relieved of I/O tasks – increase the performance
- A major change occurs with the introduction of the concept of an I/O module capable of executing a program
- For step 5, I/O module is often referred as I/O channel
- For step 6, I/O module is often referred as I/O processor
- But both terms can be used interchangeably.

Characteristics of I/O Channels

- Represents an extension of the DMA concept – can execute I/O operations having complete control over them
- CPU doesn't execute I/O instructions, just initiates an I/O transfer instructing to the I/O channels
- Those instructions are stored in main memory, executed by I/O channels – that program will specify the device identification, area of memory for storage, priority, actions for certain error

Types of I/O Channels

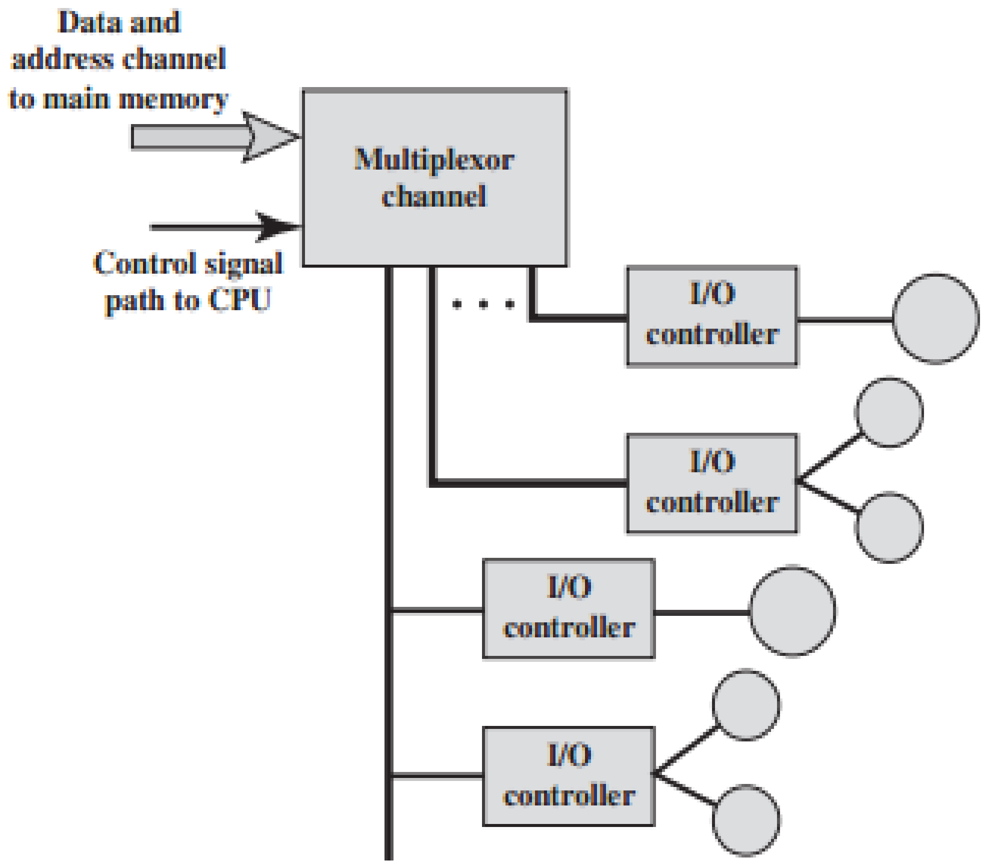
- **Selector Channels**
  - Controls multiple high speed devices – at any time, it is dedicated with one of those devices selectively
  - Each device or a set of devices is handled by a controller or I/O module
  - Here I/O channel serves as a CPU to control these I/O controllers



(a) Selector

Types of I/O Channels…

- **Multiplexor Channel:**
  - It handles I/O operations with multiple devices at the same time
  - For low speed devices, a byte multiplexor transmits character wise in different speed (device wise)
  - For high speed devices, a block multiplexor interleaves blocks of data from several devices



(b) Multiplexor

External Interconnection Standards

- USB
- FireWire Serial Bus
- Small Computer System Interface (SCSI)
- Thunderbolt
- InfiniBand
- PCI express
- SATA
- Ethernet
- Wi-Fi

Extra

- **Abc**