UNIT 3
Input and Output System

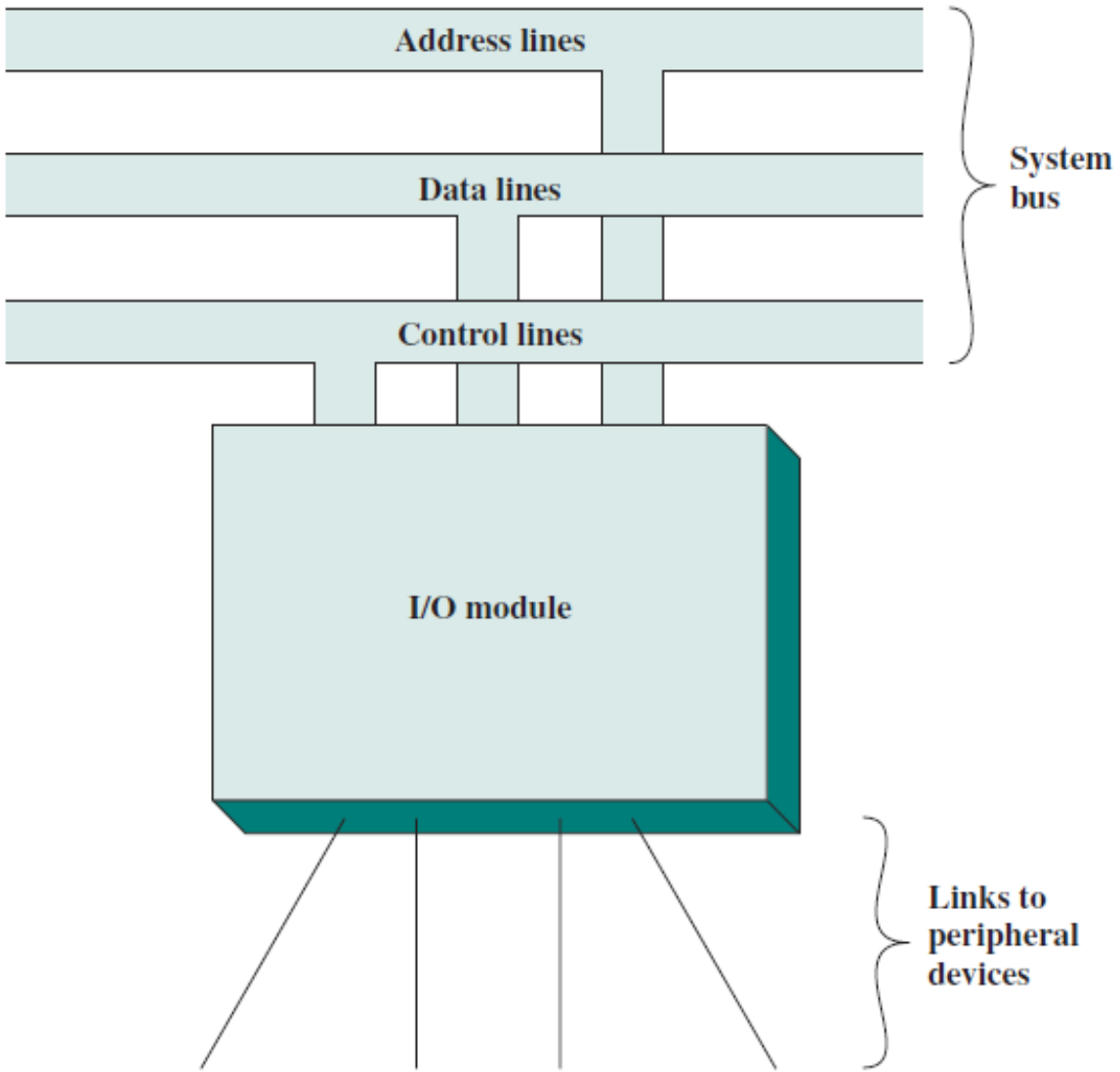
External Devices

- I/O operations are done using different external devices also called as **I/O Module**, that **provide a interface for exchanging data between the external environment and the computer.**

- An **external device attaches to the computer by a link to an I/O module**.

- The link is used to exchange control, status, and data between the I/O module and the external device.

- An external device connected to an I/O module is often referred to as a **Peripheral Device** or, simply, a **Peripheral**.


Address lines
Data lines
Control lines
System bus
I/O module
Links to peripheral devices

- Peripherals are of three categories:

1. **Human readable:** Suitable for communicating with the computer user.

   Eg. Printers, Video Display Terminals (VDTs).

2. **Machine readable:** Suitable for communicating with equipment.

   Eg. Magnetic Disks or tapes, Sensors, Actuators.

3. **Communication:** Suitable for communicating with remote devices.

   Eg. Devices which allow a computer to exchange data with a remote device, which may be a human-readable device, a machine readable device, or even another computer. (Routers, Servers)

How I/O Module works?

- Data transfer using I/O Module, involves following steps:

1. The processor tells the I/O module to check the status of the attached device.
2. The I/O module returns the device status.
3. If the device is operational and ready to transmit, the processor requests the transfer of data, by means of a command to the I/O module.
4. The I/O module obtains a unit of data (e.g., 8 or 16 bits) from the external device.
5. The data are transferred from the I/O module to the processor.

I/O Modules: Module Function

- The major functions or requirements for an I/O module fall into the following categories:

  - **Control and Timing**
  - **Processor Communication**
  - **Device Communication**
  - **Data Buffering**
  - **Error Detection**

- **Control and Timing:**
  - During any period of time, the **processor may communicate with one or more external devices**, depending on the program's need for I/O.
  - **The internal resources**, such as main memory and the system bus, **must be shared among a number of activities, including data I/O.**
  - It is **required to coordinate the flow of traffic between internal resources and external devices**.

- **Processor Communication:**

  It involves the following:

  - **Command Decoding:** The I/O module accepts commands from the processor, decodes command to appropriate signals, and send signals to peripheral through control bus. For example, an I/O module for a disk drive might accept the following commands: READ SECTOR, WRITE SECTOR, SEEK track number, and SCAN record ID.

  - **Data:** Data are exchanged between the processor and the I/O module over the data bus.

- **Status Reporting:** Because peripherals are so slow compared to processor, it is important to know the status of operation. For example, if an I/O module is asked to send data to the processor (read), it may not be ready to do so because it is still working on the previous I/O command.

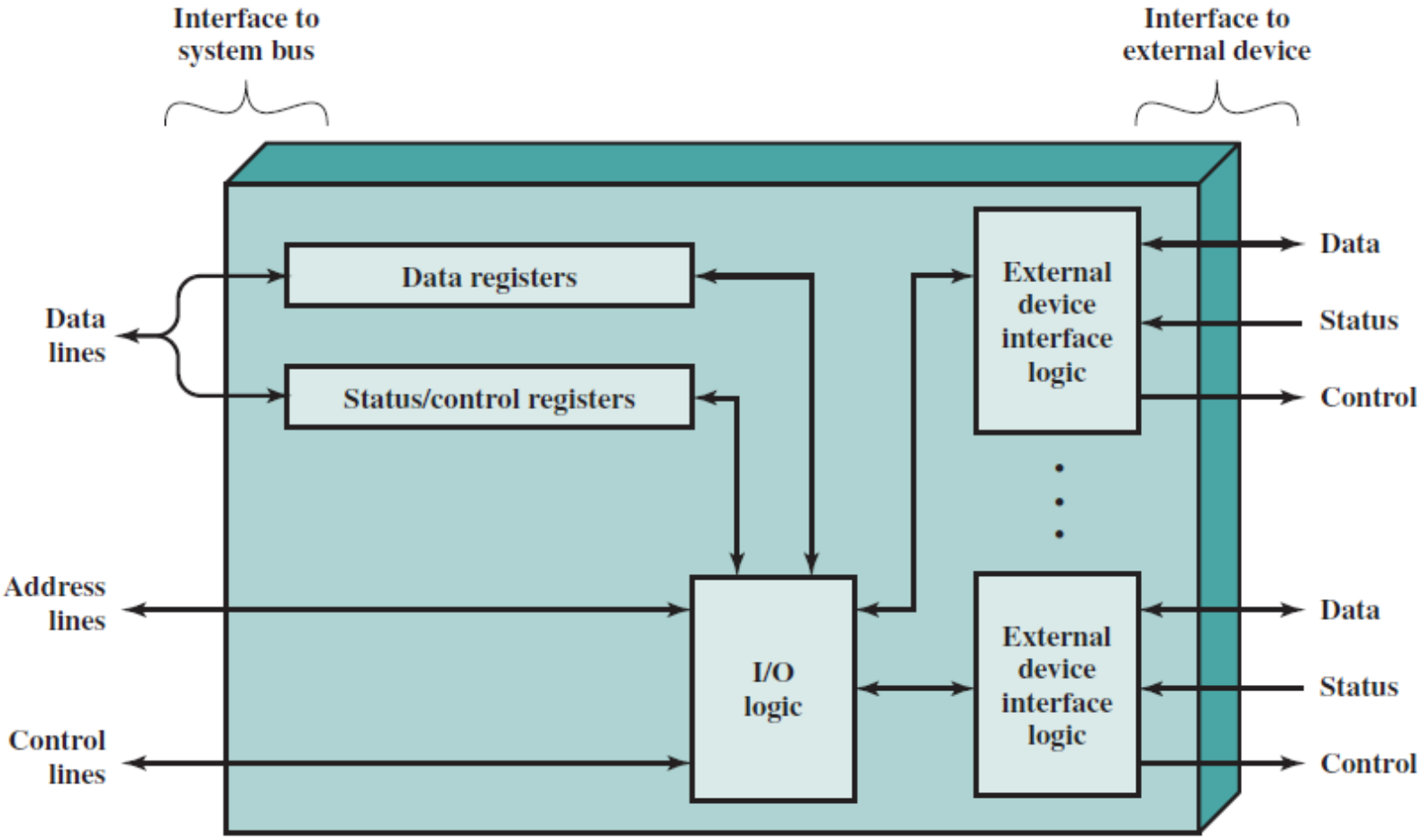
- **Address Recognition:** Each I/O device have a address which is used to recognize particular peripheral. Thus, an I/O module must recognize one unique address for each peripheral it controls.

- **Device Communication:** The I/O module must be able to perform Communication in between different peripherals. This communication involves commands, status information, and data.

- **Data Buffering:** The I/O module must be able to buffer data during data operations.

- **Error Detection:** An I/O module is often responsible for error detection and for subsequently reporting errors to the processor.

I/O Module Structure



Techniques for I/O Operations

- There are three different techniques

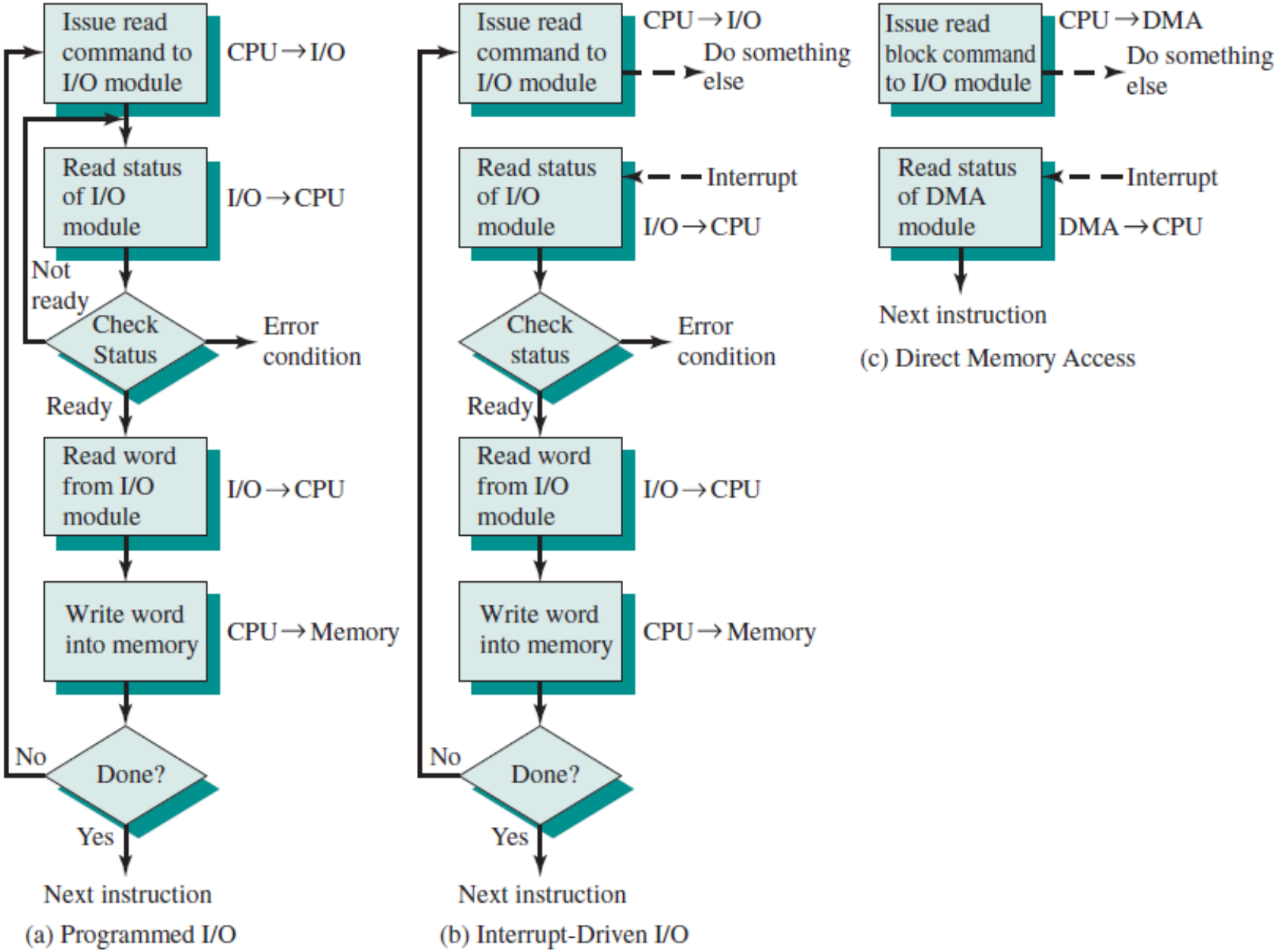
1. **Programmed I/O**
2. **Interrupt Driven I/O**
3. **Direct Memory Access**

- **Programmed I/O**
  - Data are exchanged between the processor and the I/O module.
  - The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data.
  - **When the processor issues a command to the I/O module, it must wait until the I/O operation is complete.** If the processor is faster than the I/O module, this is wasteful of processor time.

- **Interrupt-driven I/O**
  - The processor issues an I/O command, it continues doing its own work. When peripheral is ready, receive an interrupt and start working.
- **Disadvantage of Programmed I/O and Interrupt-driven I/O:**
  - With both programmed and interrupt I/O, The processor is responsible for data operations. So Processor is continuously used in such operations.

- **Direct Memory Access (DMA)**
  - In this mode, the data operations are done without 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)


Issue read command to I/O module
CPU → I/O
Read status of I/O module
I/O → CPU
Not ready
Check Status
Error condition
Ready
Read word from I/O module
I/O → CPU
Write word into memory
CPU → Memory
No
Done?
Yes
Next instruction
(a) Programmed I/O
Issue read command to I/O module
CPU → I/O
Do something else
Read status of I/O module
Interrupt
I/O → CPU
Check status
Error condition
Ready
Read word from I/O module
I/O → CPU
Write word into memory
CPU → Memory
No
Done?
Yes
Next instruction
(b) Interrupt-Driven I/O
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

Programmed I/O

- When the processor is executing a program and encounters an instruction relating to I/O, **it executes that instruction by issuing a command to the appropriate I/O module.**

- With programmed I/O, the **I/O module will perform the requested action and then set the appropriate bits in the I/O status register.**

- **The I/O module takes no further action to alert the processor.** In particular, it does not interrupt the processor.

- Thus, **it is the responsibility of the processor periodically to check the status** of the I/O module until it finds that the operation is complete.

- To explain the programmed I/O technique, we view it first from the point of view of the **I/O commands** issued by the processor to the I/O module, and then from the point of view of the **I/O instructions** executed by the processor.

Programmed I/O: I/O Commands

- **Control:** Used to activate a peripheral and tell it what to do.
- **Test:** Used to test various status conditions associated with an I/O module and its peripherals.
- **Read:** Causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer.
- **Write:** Causes the I/O module to take an item of data (byte or word) from the data bus and subsequently transmit that data item to the peripheral.

Programmed I/O: I/O Instructions

- Processor gives commands to I/O module through special instructions.
- Typically, there will be many I/O devices connected through I/O modules to the system.
- Each device is given a unique identifier or address.
- When the processor issues an I/O command, the command contains the address of the desired device.

- When the processor, main memory, and I/O share a common bus, two modes of addressing are possible:

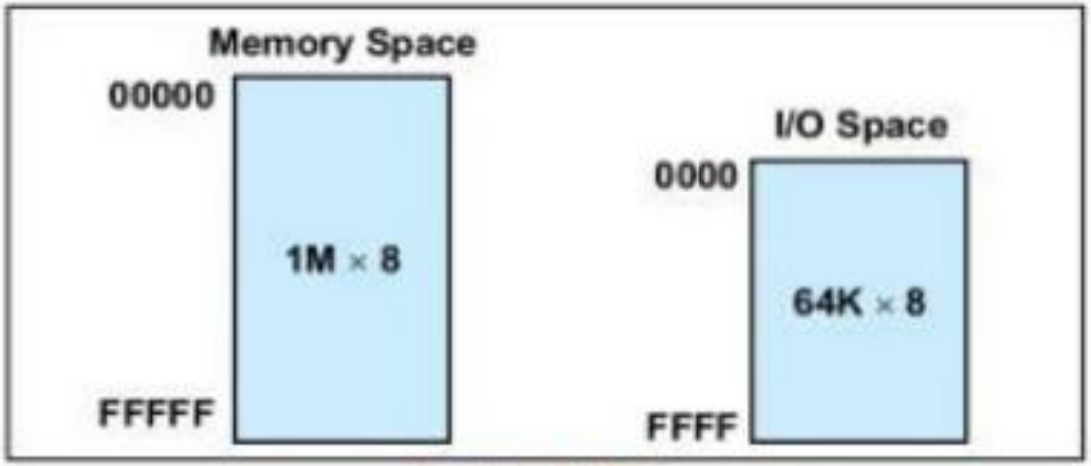
1. **Memory Mapped I/O**
2. **Isolated I/O**

- **Memory - Mapped I/O:**
  - In this configuration **same address space is used for both memory and I/O.**
  - In such scenarios **the devices (I/O) are treated as a part of the memory only.**
  - There are no specific I/O instructions. **It allows the computer to used the same instructions for both I/O transfers and memory transfers.**
  - Some instructions are memory reference instructions and others are I/O reference.
  - They are only **one set of read/write control signals.**

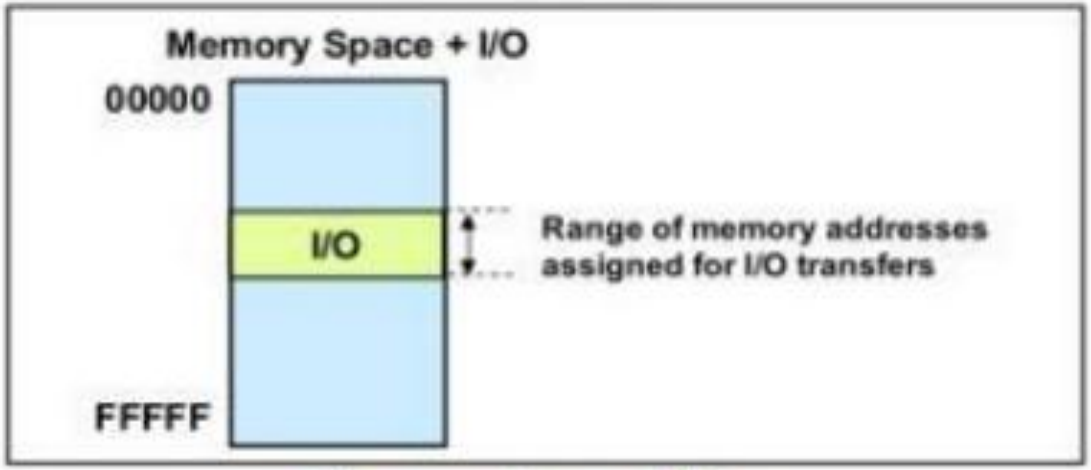
- **Isolated I/O:**
  - The isolated I/O configuration **separates all I/O interface addresses from the memory addresses.**
  - The devices of I/O are treated in a separate domain as compared to memory.
  - In the isolated I/O configuration, the CPU has distinct input and output instructions.
  - In isolated I/O configuration **the memory address and I/O address have its own address space.**

Isolated I/O	No.	Memory Mapped I/O
Isolated I/O uses separate memory space.	01	Memory mapped I/O uses memory from the main memory.
Limited instructions can be used. Those are IN, OUT, INS, OUTS.	02	Any instruction which references to memory can be used.
The addresses for Isolated I/O devices are called ports.	03	Memory mapped I/O devices are treated as memory locations on the memory map.
IORC & IOWC signals expands the circuitry.	04	*IORC & IOWC* signals has no functions in this case which reduces the circuitry.
Efficient I/O operations due to using separate bus	05	Inefficient I/O operations due to using single bus for data and addressing
Comparatively larger in size	06	Smaller in size
Uses complex internal logic	07	Common internal logic for memory and I/O devices
Slower operations	08	Faster operations

Isolated vs. Memory Mapped I/O



Isolated I/O



Memory-Mapped I/O

IN and OUT Instruction

Mnemonic	Description
IN AL,imm8	Input byte from imm8 I/O port address into AL.
IN AX,imm8	Input byte from imm8 I/O port address into AX.
IN EAX,imm8	Input byte from imm8 I/O port address into EAX.
IN AL,DX	Input byte from I/O port in DX into AL.
IN AX,DX	Input word from I/O port in DX into AX.
IN EAX,DX	Input doubleword from I/O port in DX into EAX.

Mnemonic	Description
OUT imm8, AL	Output byte in AL to I/O port address imm8.
OUT imm8, AX	Output word in AX to I/O port address imm8.
OUT imm8, EAX	Output doubleword in EAX to I/O port address imm8.
OUT DX, AL	Output byte in AL to I/O port address in DX.
OUT DX, AX	Output word in AX to I/O port address in DX.
OUT DX, EAX	Output doubleword in EAX to I/O port address in DX.

INS and OUTS Instructions

Opcode	Mnemonic
6C INS m8, DX	Input byte from I/O port specified in DX into memory location specified in ES:(E)DI.
6D INS m16, DX	Input word from I/O port specified in DX into memory location specified in ES:(E)DI.
6D INS m32, DX	Input doubleword from I/O port specified in DX into memory location specified in ES:(E)DI.
6C INSB	Input byte from I/O port specified in DX into memory location specified with ES:(E)DI.
6D INSW	Input word from I/O port specified in DX into memory location specified in ES:(E)DI.
6D INSD	Input doubleword from I/O port specified in DX into memory location specified in ES:(E)DI.

Mnemonic	Description
OUTS DX, m8	Output byte from memory location specified in DS:(E)SI to I/O port specified in DX.
OUTS DX, m16	Output word from memory location specified in DS:(E)SI to I/O port specified in DX.
OUTS DX, m32	Output doubleword from memory location specified in DS:(E)SI to I/O port specified in DX.
OUTSB	Output byte from memory location specified in DS:(E)SI to I/O port specified in DX.
OUTSW	Output word from memory location specified in DS:(E)SI to I/O port specified in DX.
OUTSD	Output doubleword from memory location specified in DS:(E)SI to I/O port specified in DX.

Programmed I/O: Problem

- The **processor has to wait a long time till I/O module gives status of peripheral** for either reception or transmission of data.
- The processor, **while waiting, must repeatedly check the status of the I/O module.**
- As a result, the level of the performance of the entire system is severely degraded.

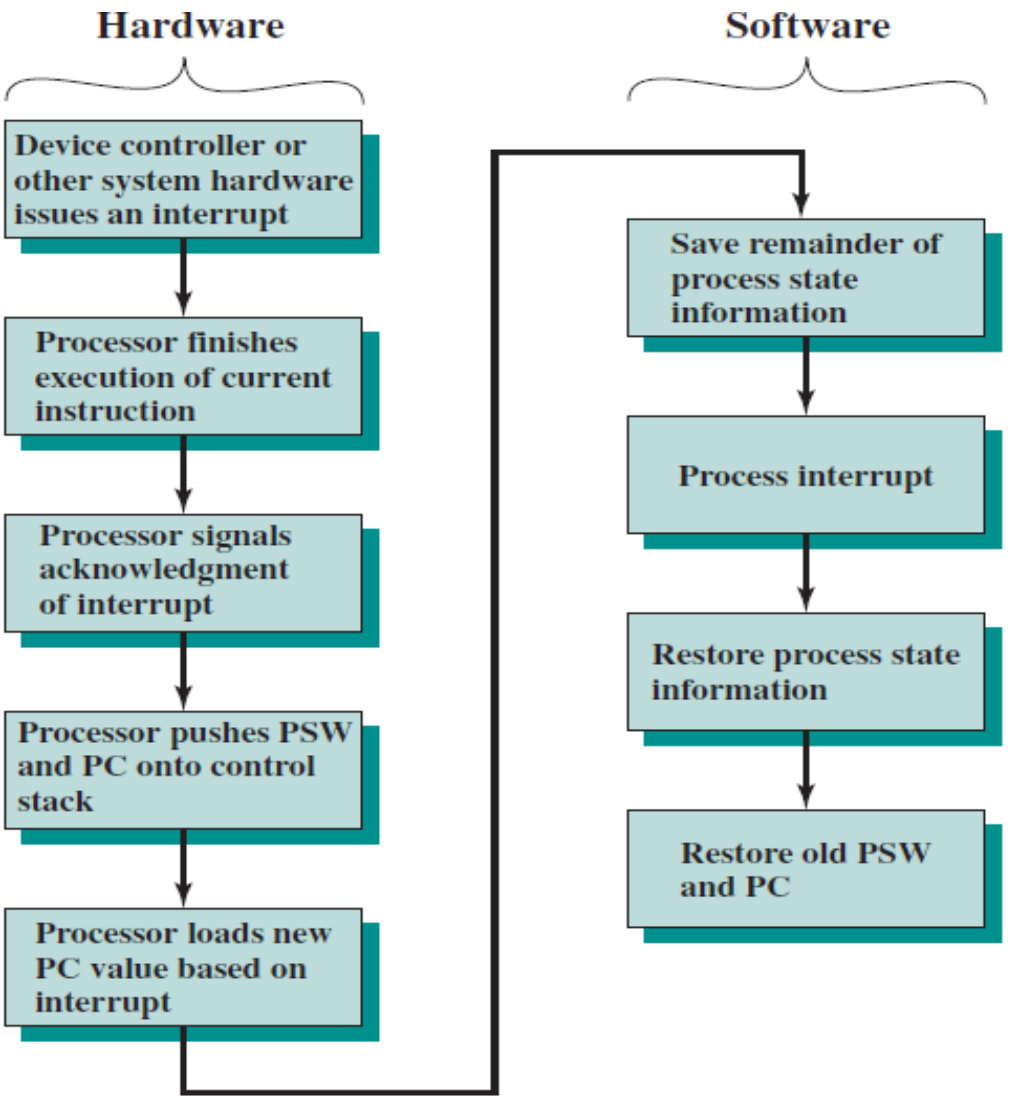
Interrupt-Driven I/O

- An alternative to programmed I/O is Interrupt-driven I/O.
- Here the processor to issue an I/O command to a module and then go on to do some other useful work.
- The **I/O module will then interrupt the processor to request service when it is ready** to exchange data with the processor.
- The processor then executes the data transfer.

How it works?

- Fig 7.4 (b), slide no 16.

Interrupt Processing



Interrupt-Driven I/O: Design Issues

1. Because there will almost invariably be multiple I/O modules, **how does the processor determine which device issued the interrupt?**

   – Solution:

      1. **Multiple Interrupt Lines**
      2. **Software Poll**
      3. **Daisy Chain (Hardware Poll, Vectored)**
      4. **Bus Arbitration (Vectored)**

2. **If multiple interrupts have occurred**, how does the processor decide which one to process?

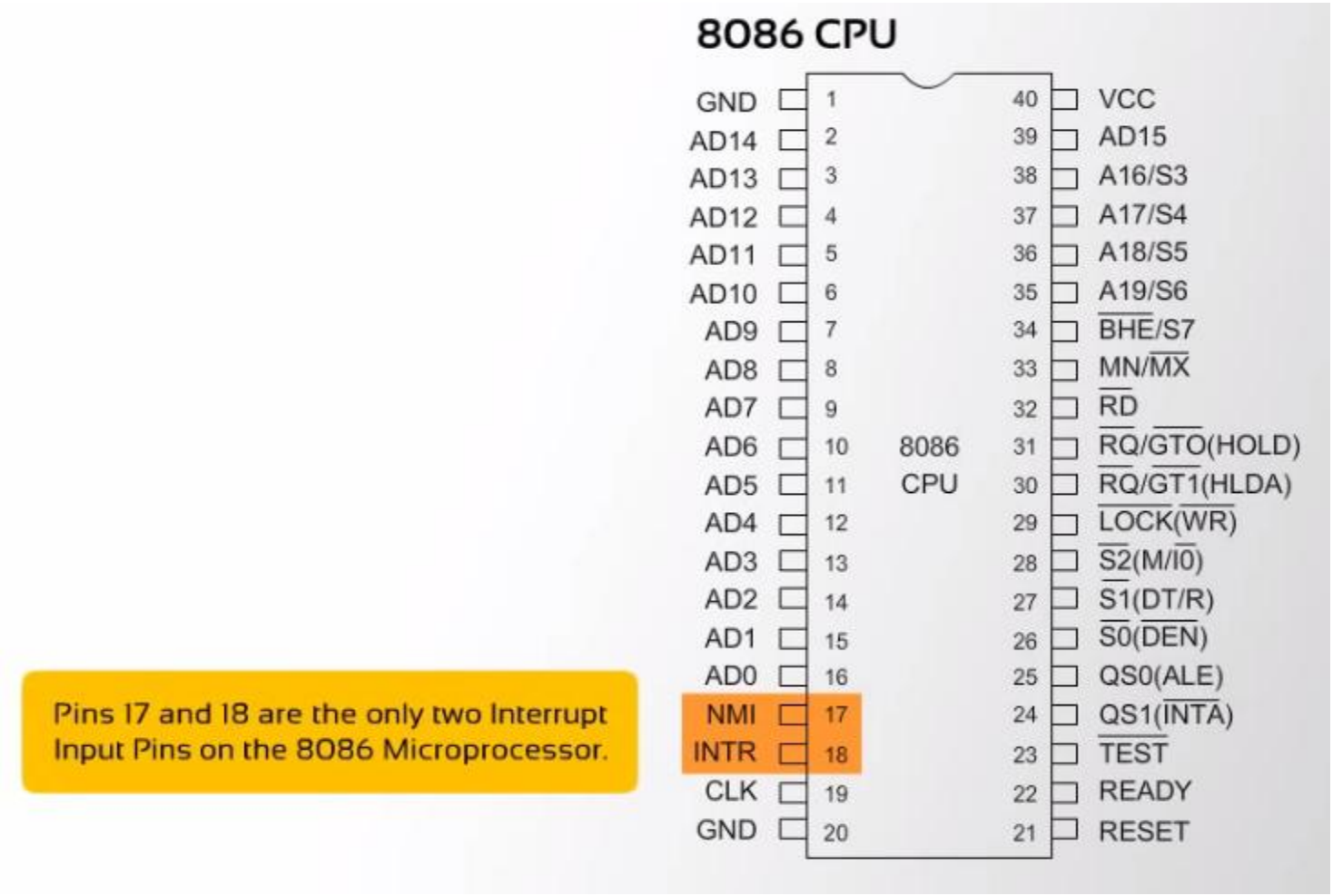
   - **Give priority to interrupt.**
   - **Methods:**
     - **Polling**
     - **Hardware can be used**

Case Study: 8259

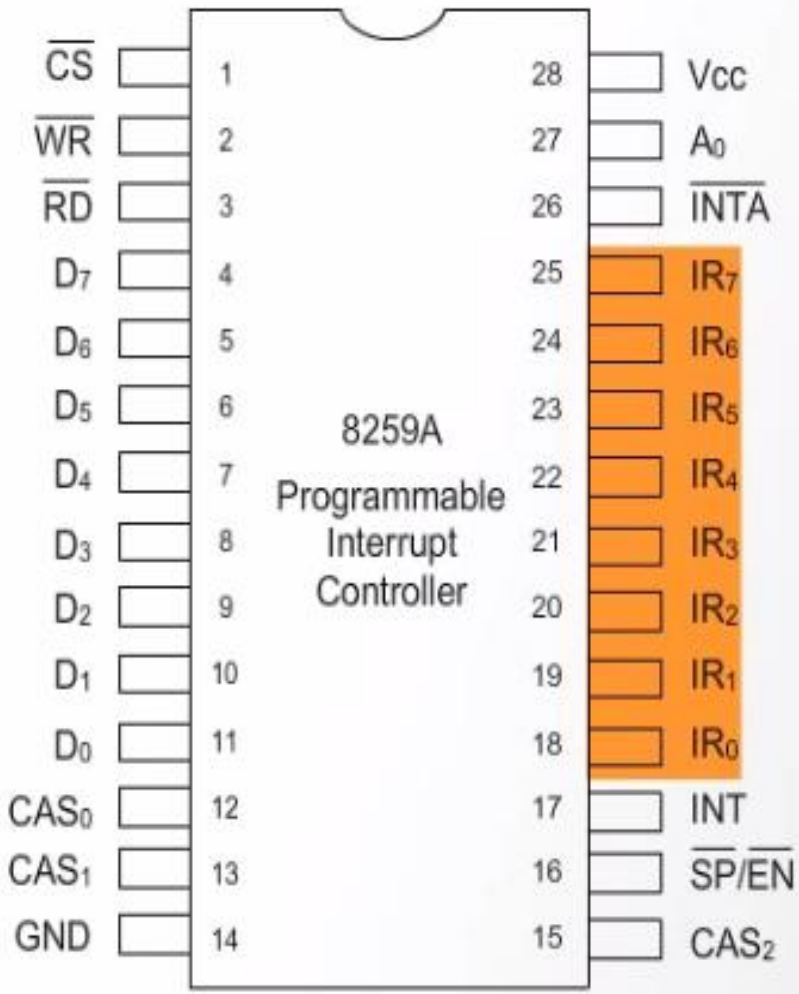
- 8259 is **Programmable Interrupt Controller (PIC).**
- It is connected to 8086 to **manage complex interrupt systems.**
- It **manages interrupt requests as per priority of interrupts.**
- **8259 act as a multiplexer**, combining multiple interrupt input sources into single interrupt request to the processor.

- 8259 is a very flexible peripheral controller chip:
  - It supports **eight input interrupt requests** from peripherals and issues a single interrupt signal to processor.
  - PIC can deal with up to **64 interrupt inputs** by cascading eight 8259's together.
  - Various priority schemes can also programmed.

Why 8259?

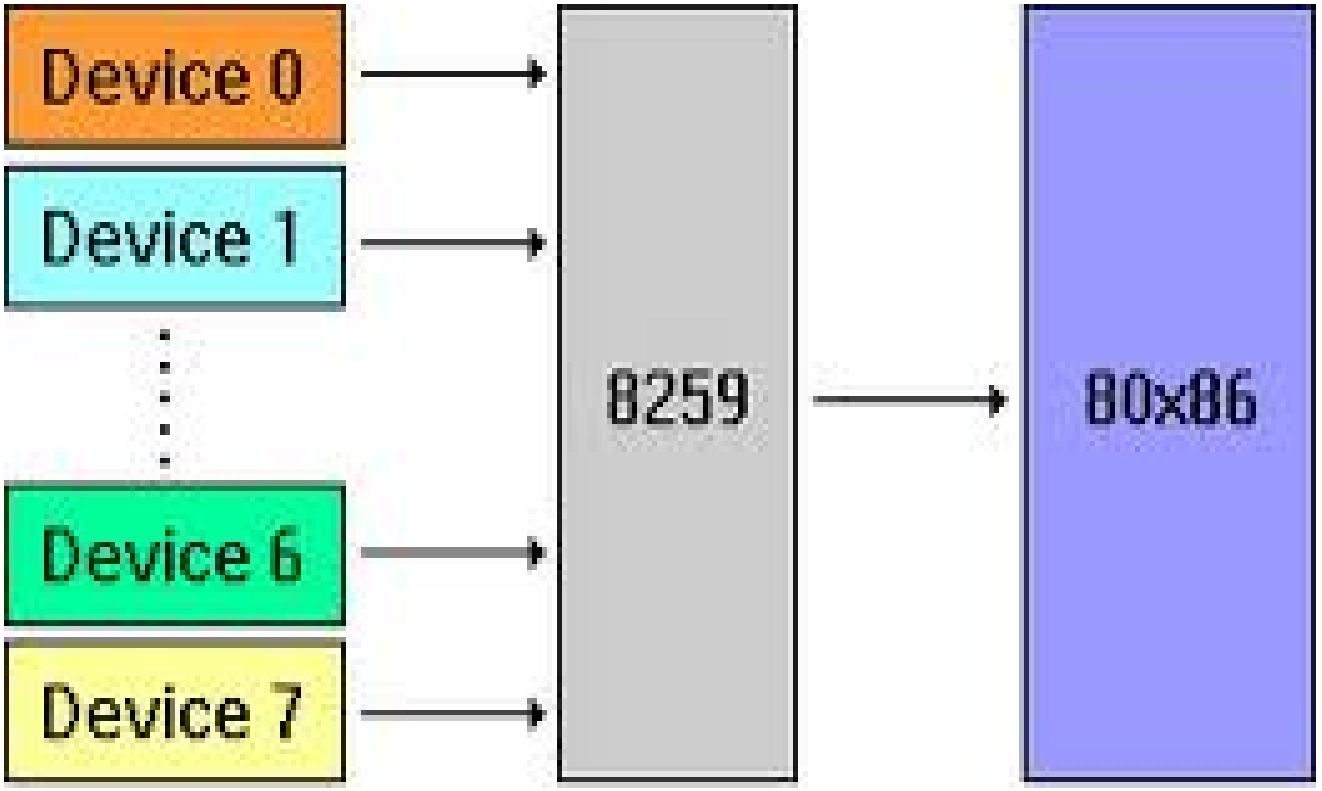

8086 CPU
GND 1
AD14 2
AD13 3
AD12 4
AD11 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD1 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
8086 CPU
40 VCC
39 AD15
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
34 BHE/S7
33 MN/MX
32 RD
31 RQ/GTO(HOLD)
30 RQ/GT1(HLDA)
29 LOCK(WR)
28 S2(M/IO)
27 S1(DT/R)
26 S0(DEN)
25 QS0(ALE)
24 QS1(INTA)
23 TEST
22 READY
21 RESET
Pins 17 and 18 are the only two Interrupt Input Pins on the 8086 Microprocessor.

8259 PIC



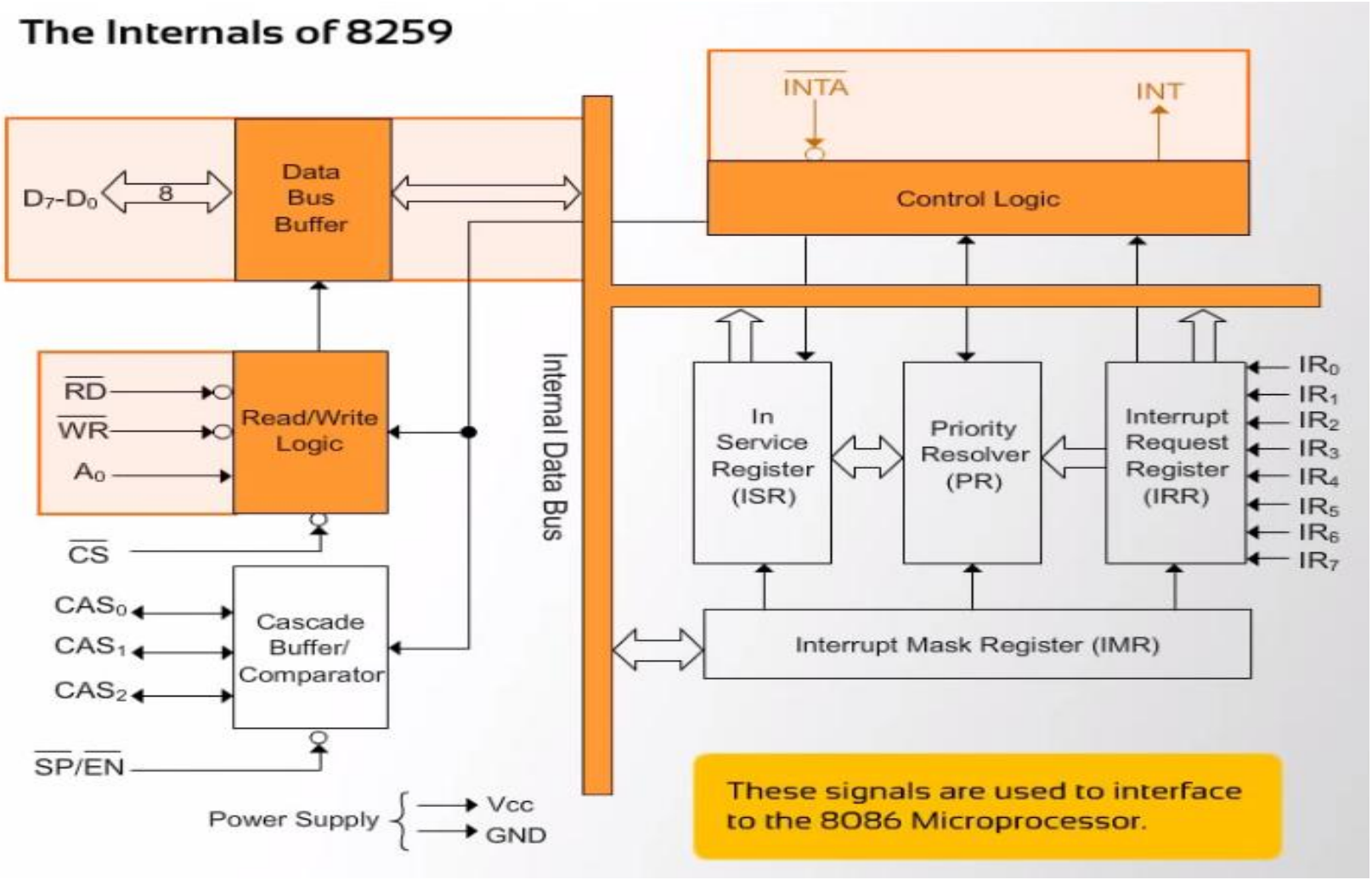
Upto eight Hardware Interrupting devices are supported.

Interfacing 8259 with Microprocessor and other devices

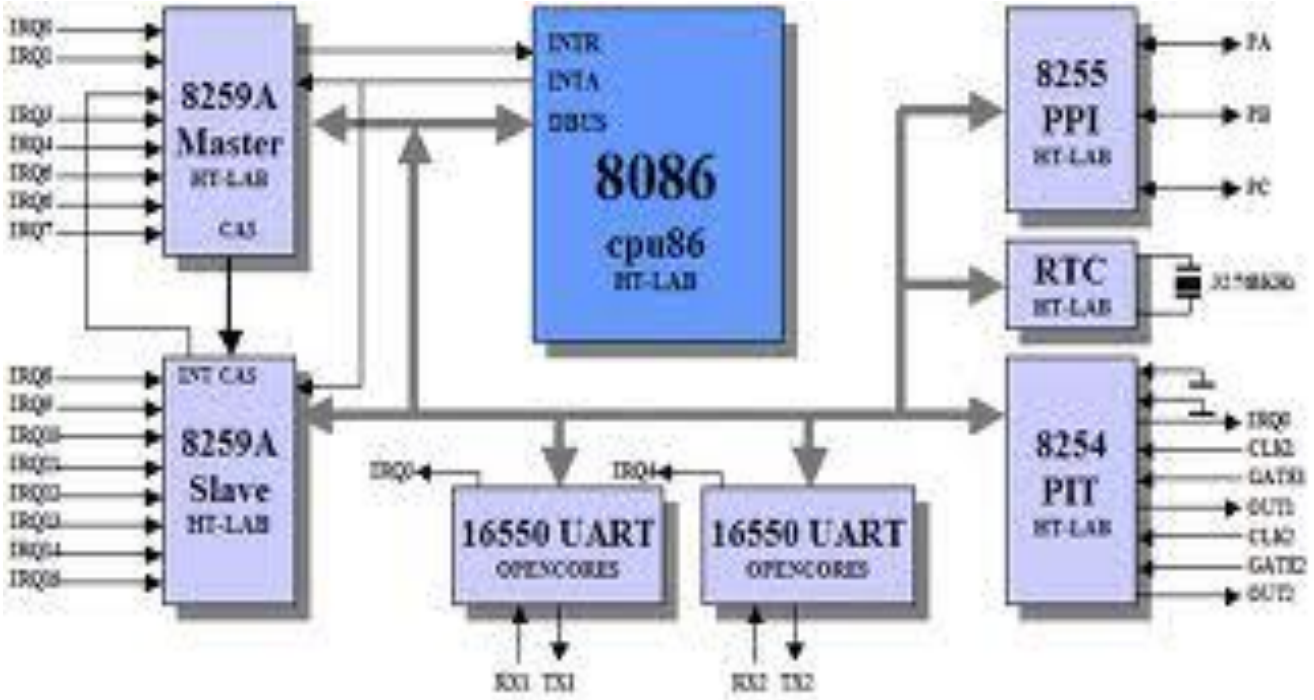


Devices may be Slave 8259's.

Block Diagram : 8259



- **Data bus buffer:** bidirectional D0-D7 bus, it is connected to the processor.
- **R/W Logic:** Using these pins 8259 is interfaced with processors.
- **Cascade buffer/ comparator:** These logic allows for cascading multiple 8259 controller in Master-Slave configuration.



- Three types of Registers:

1. **IRR (Interrupt Request Register):** It stores interrupt requests. Interrupt lines (IR0-IR7) goes high, when it accepts interrupt, and store it accordingly.

2. **IMR (Interrupt Mask Register):** This logic block masks the interrupt lines based on programming by the processor. It prevents masked interrupt lines by interrupting the processor.

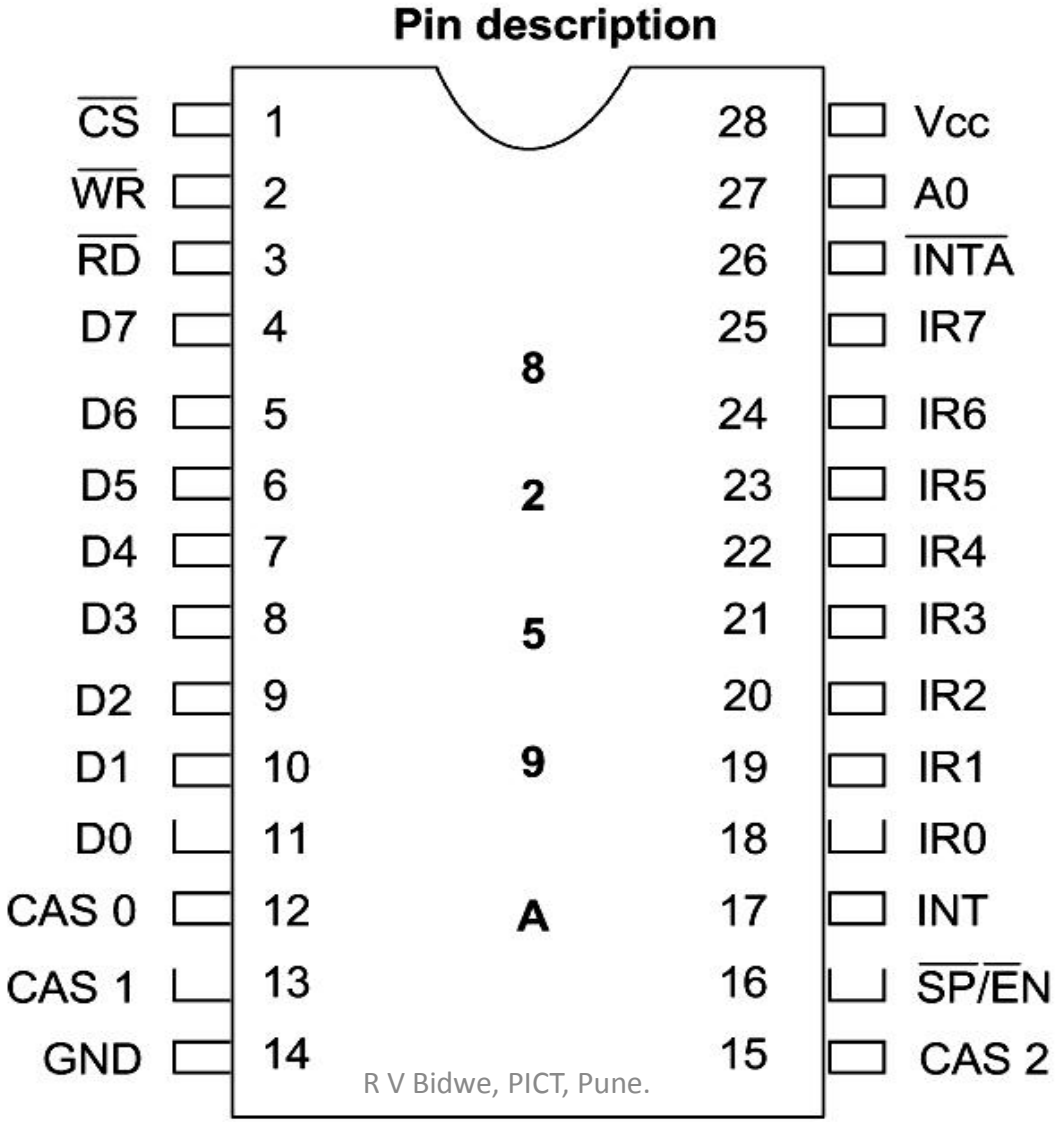
3. **ISR (In Service Register):** It stores all interrupt levels that are currently being serviced.

- **Priority resolver:** Determine the interrupt priority of the active interrupt inputs.

- **Control Logic:** It contains two pins,

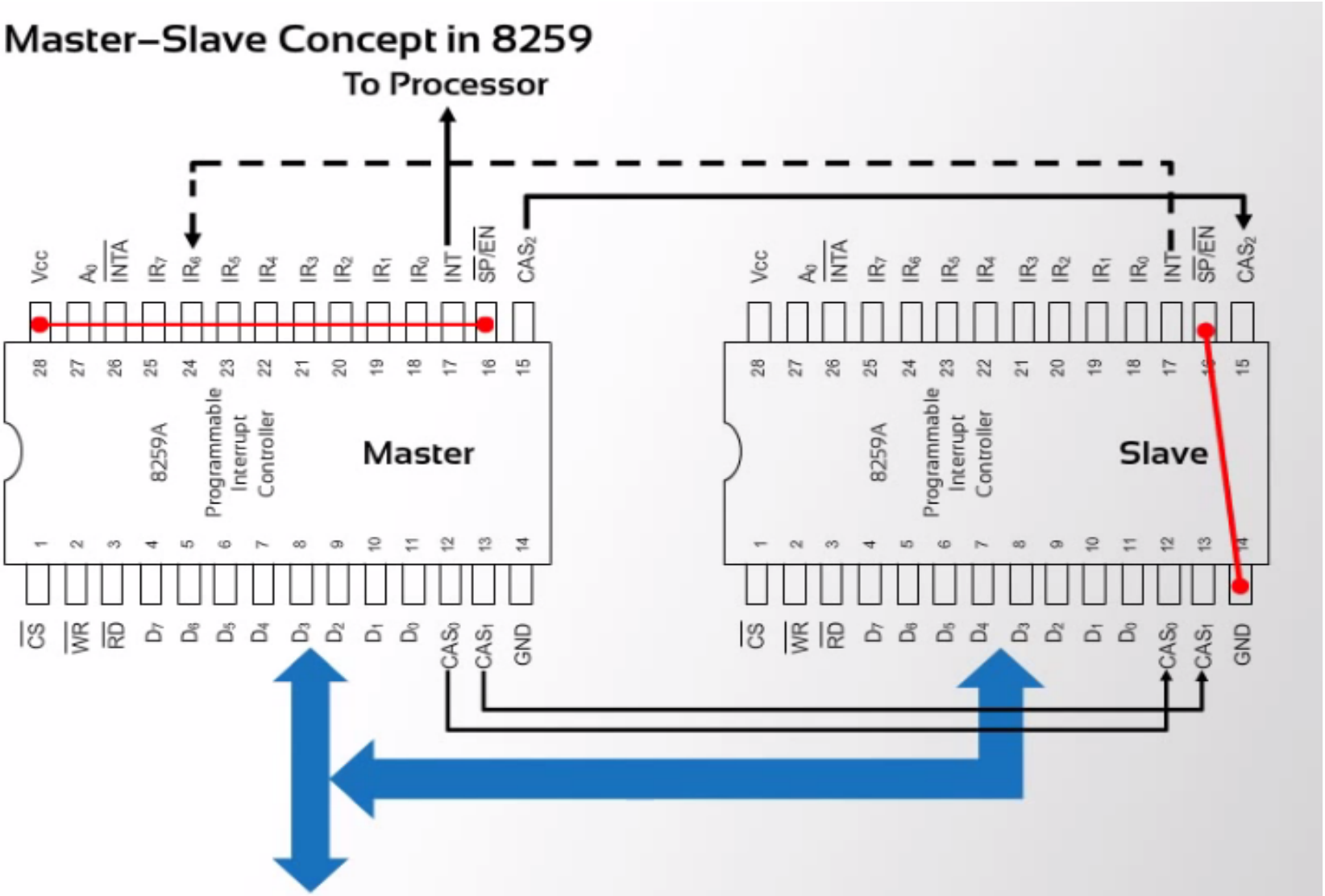
1. **INTR :** It is output pin used to interrupt the CPU.
2. **INTA(#):** Through this pin 8259 get acknowledgement for INT signal.

Pin Diagram: 8259

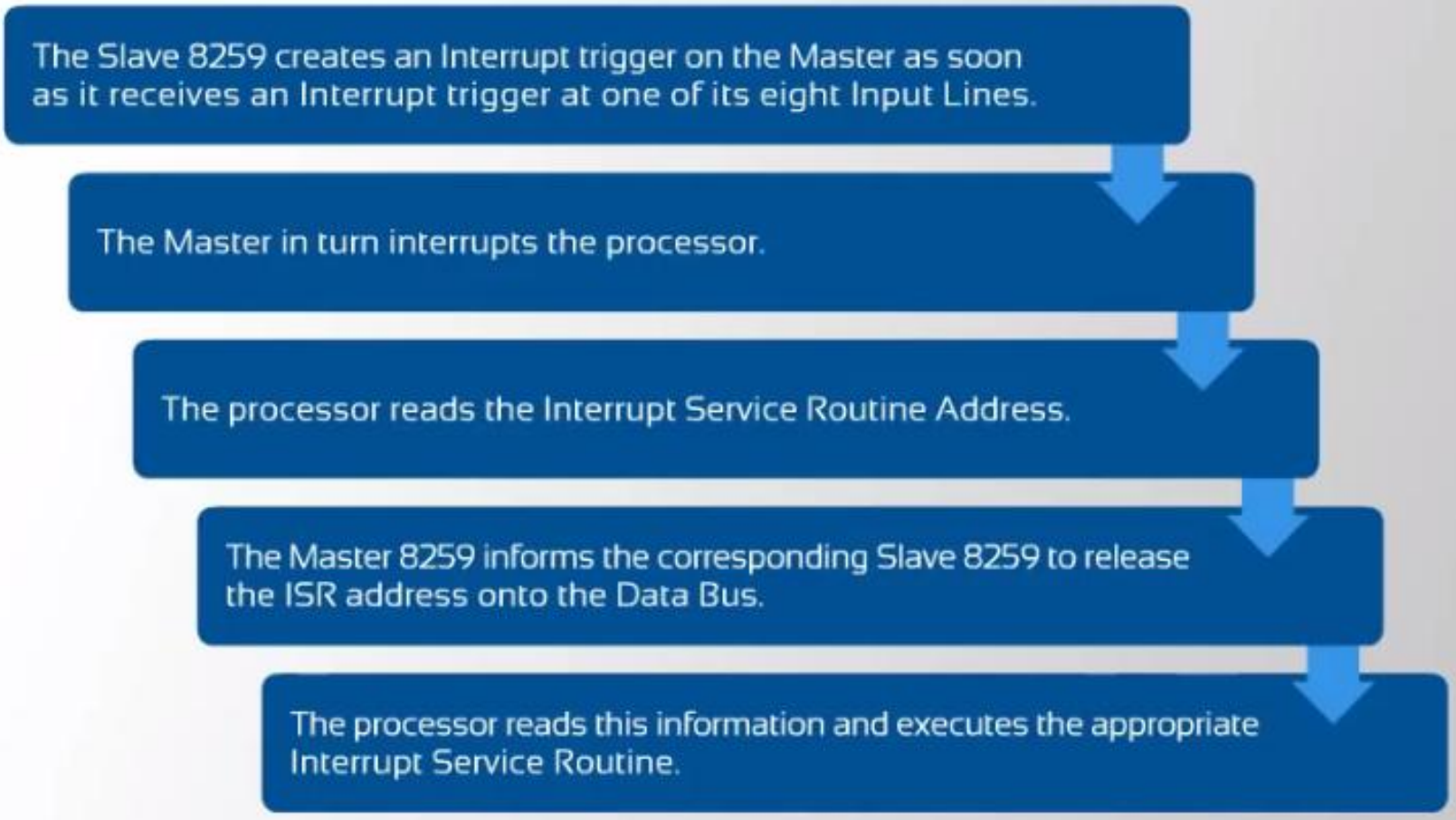


Symbol	Pin No.	Type	Name and Function
V_{CC}	28	I	SUPPLY: +5V Supply
GND	14	I	GROUND
$\overline{CS}$	1	I	CHIP SELECT: A LOW on this pin enables $\overline{RD}$ and $\overline{WR}$ communication between the CPU and the 8259A. INTA functions are independent of $\overline{CS}$.
$\overline{WR}$	2	I	WRITE: A LOW on this pin when $\overline{CS}$ is low enables the 8259A to accept command words from the CPU.
$\overline{RD}$	3	I	READ: A LOW on this pin when $\overline{CS}$ is low enables the 8259A to release data onto the data bus for the CPU.
D_7-D_0	4 -11	I/O	BIDIRECTIONAL DATA BUS: Control, status and interrupt-vector information is transferred via this bus.
CAS_0-CAS_2	12,13,15	I/O	CASCADE LINES: The CAS lines form a private 8259A bus to control a multiple 8259A structure. These pins are outputs for a master 8259A and inputs for a slave 8259A.

SP/$\overline{EN}$	16	I/O	SLAVE PROGRAM/ENABLE BUFFER: This is a dual function pin. When in the Buffered Mode it can be used as an output to control buffer transceivers (EN). When not in the buffered mode it is used as an input to designate a master (SP - 1) or slave (SP - 0).
INT	17	O	INTERRUPT: This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU, thus it is connected to the CPU's interrupt pin.
IR_0-IR_7	18-25	I	INTERRUPT REQUESTS: Asynchronous inputs. An interrupt request is executed by raising an IR input (low to high), and holding it high until it is acknowledged (*Edge Triggered Mode*), or just by a high level on an IR input (*Level Triggered Mode*).
$\overline{INTA}$	26	I	INTERRUPT ACKNOWLEDGE: This pin is used to enable 8259A interrupt-vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the CPU
A_0	27	I	A0 ADDRESS LINE: This pin acts in conjunction with the $\overline{CS}$, $\overline{WR}$, and $\overline{RD}$ pins. It is used by the 8259A to decipher various Command Words the CPU writes and status the CPU wishes to read.


Master–Slave Concept in 8259
To Processor
Master
Slave
8259A
Programmable
Interrupt
Controller
Vcc
A0
INTA
IR7
IR6
IR5
IR4
IR3
IR2
IR1
IR0
INT
SP/EN
CAS2
CS
WR
RD
D7
D6
D5
D4
D3
D2
D1
D0
CAS0
CAS1
GND

How does the Master–Slave concept work?



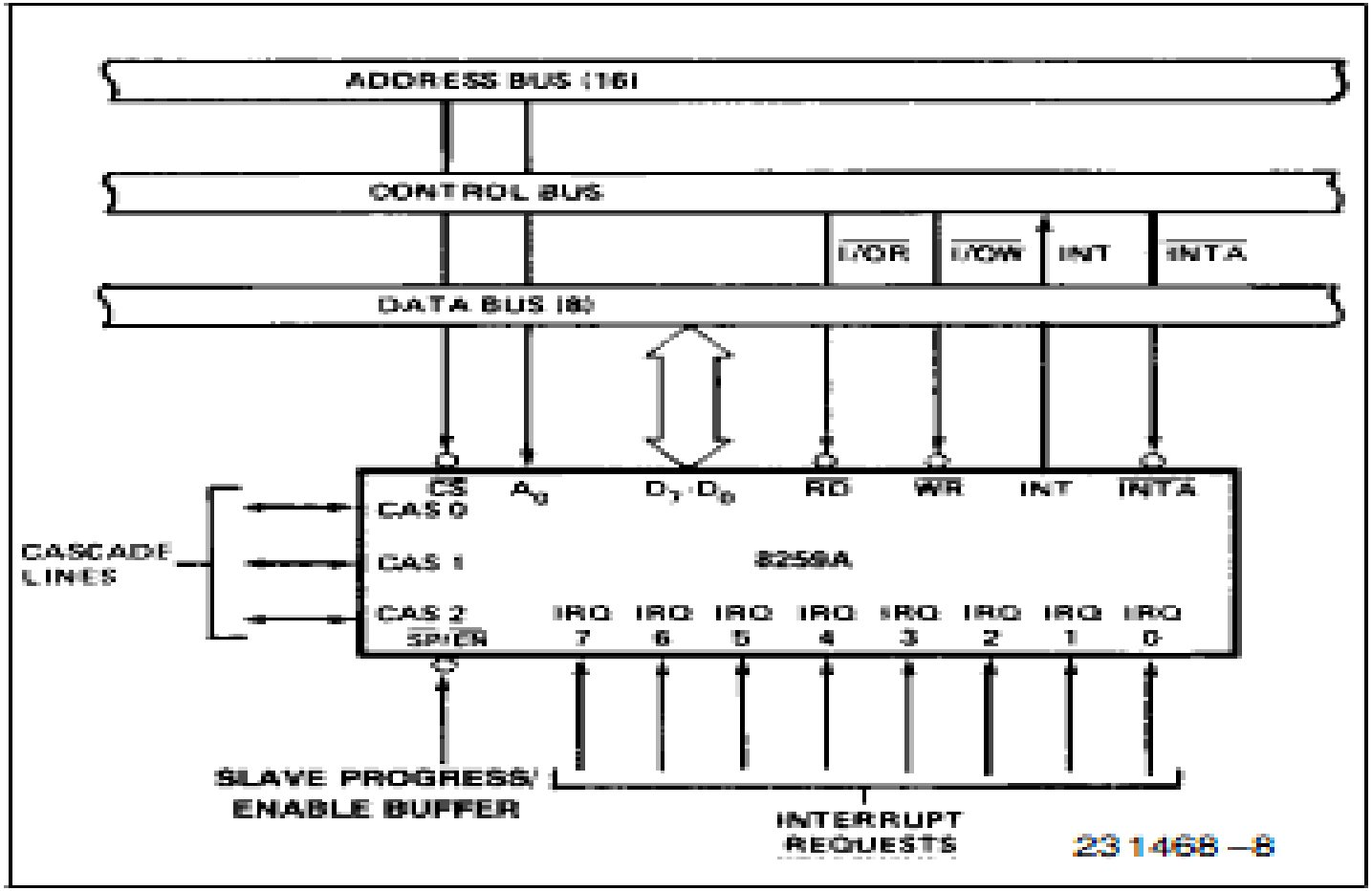


Figure 5. 8259A Interface to Standard System Bus

Drawbacks of Programmed and Interrupt-Driven I/O

- Interrupt-driven I/O, simple programmed I/O **both requires active participation of Processor** in data transfer process.

- Thus, both these forms of I/O suffer from two inherent drawbacks:
  1. The **I/O transfer rate is limited by the speed** with which the processor can test and service a device.

  2. The **processor is tied up in managing an I/O transfer,** as number of instructions must be executed for each I/O transfer.

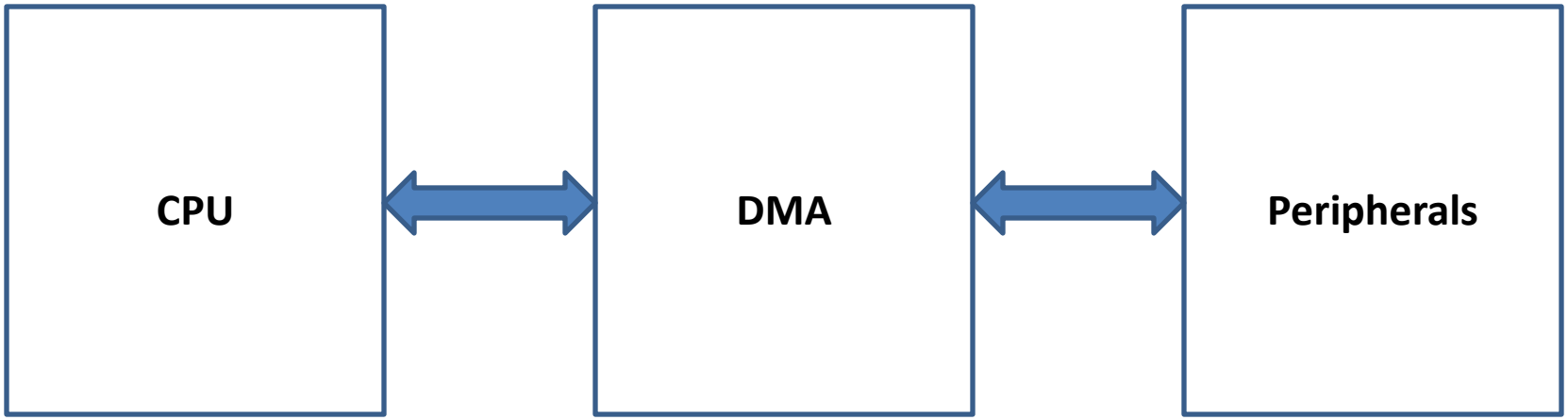
DMA Introduction

- **Direct Memory Access (DMA)** is a method of allowing data to be moved from one location to another in a computer without involving the central processor (CPU).

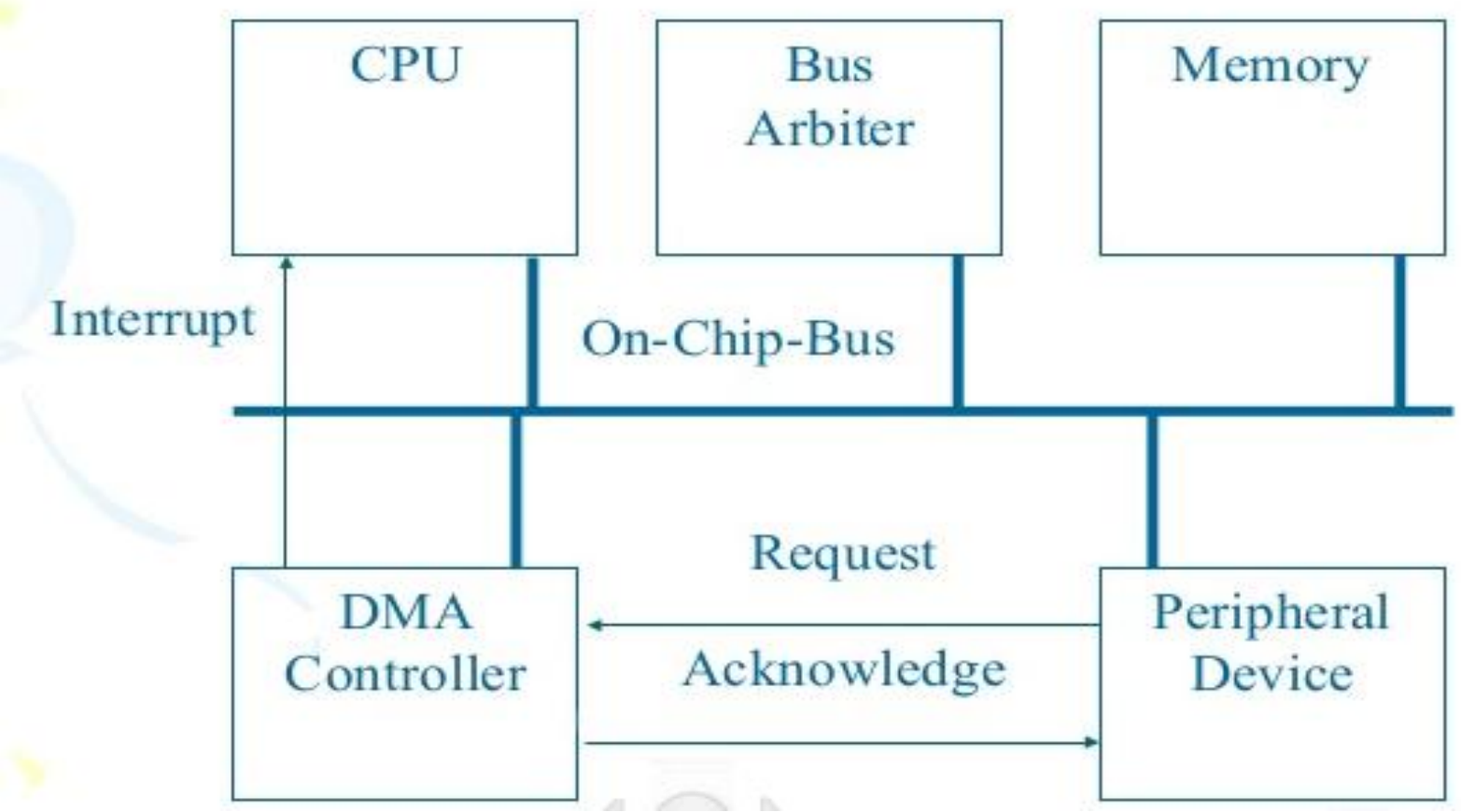
- It is also **a fast way of transferring data within (and sometimes between) computer.**

- The device requests CPU (through a DMA controller) to **Hold** its data, address bus and control bus, so that device may transfer data to/from memory.

- DMA transfer is only initiated only after receiving **HLDA** signal from CPU.


CPU
DMA
Peripherals

How DMA works



DMA Terminologies

- **DMA channel:** System pathway used by a device to transfer information directly to and from memory. There are usually 8 channels in a computer system.
- **DMA controller:** Dedicated hardware used for controlling the DMA operation.
- **Single-cycle mode:** DMA data transfer is done one byte at a time.
- **Burst-mode:** DMA transfer is finished when all data has been moved.

How DMA Works?

- It is a **Different Module** on a system.
- In data transfers, we **can use this without involving Processor**.
- **DMA uses system bus to do data transfer** operations. So, the DMA module must use the bus only when
  - The processor does not need it.
  - It must force the processor to suspend operation temporarily.

- Eg. The processor wishes to read or write a block of data through DMA module. Then processor issues DMA module the following information:

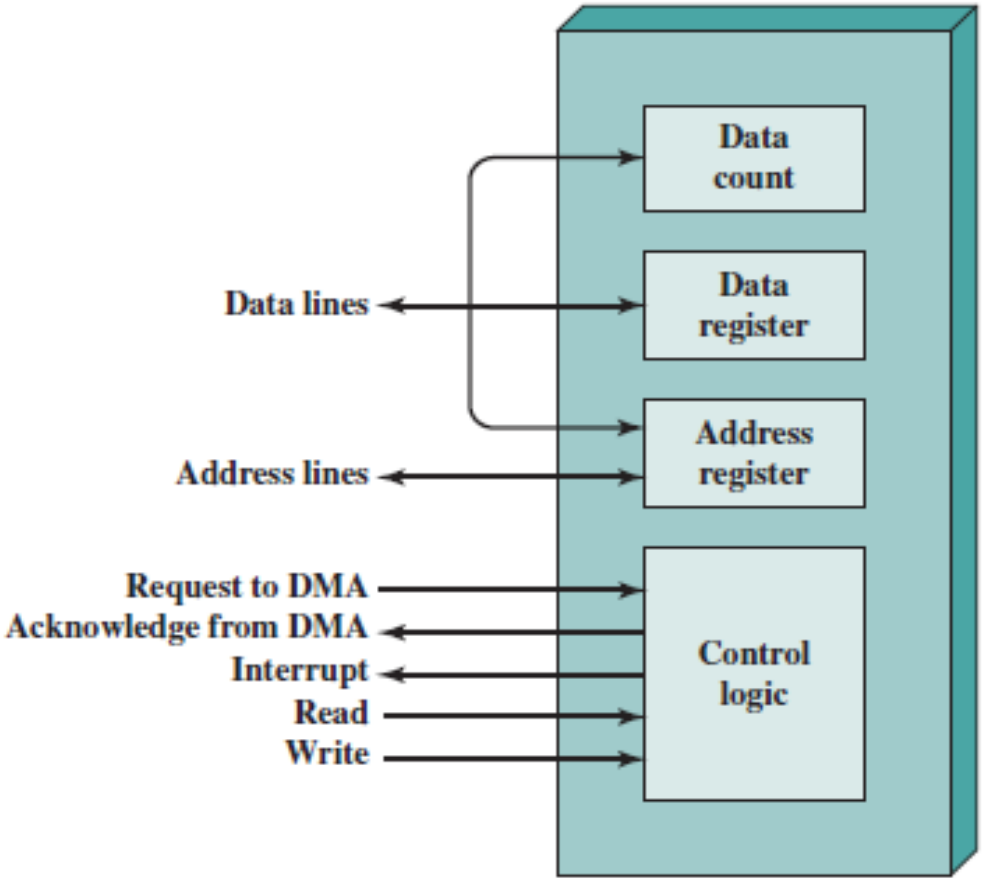
  - Whether a read or write is requested, using the **RD(#) or WR(#)** line between the processor and the DMA module.

  - The **address of the I/O device** involved, communicated on the **Data Lines**.

  - The **starting location in memory** to read from or write to, communicated on the data lines and stored by the DMA module in its **Address Register**.

  - The **number of words to be read or written**, again communicated via the data lines and stored in the **Data Count Register**.

Typical DMA Block Diagram



DMA with Processor for System Bus

- Two control signals are used to request and acknowledge a Direct Memory Access (DMA) transfer in the microprocessor-based system.
  - The **HOLD** signal as an input(to the processor) is used to request a System bus.
  - The **HLDA** signal as an output that acknowledges the availability of system bus.
- When the processor recognizes the HOLD, it stops its execution and enters hold cycles.

The 8237 DMA Controller

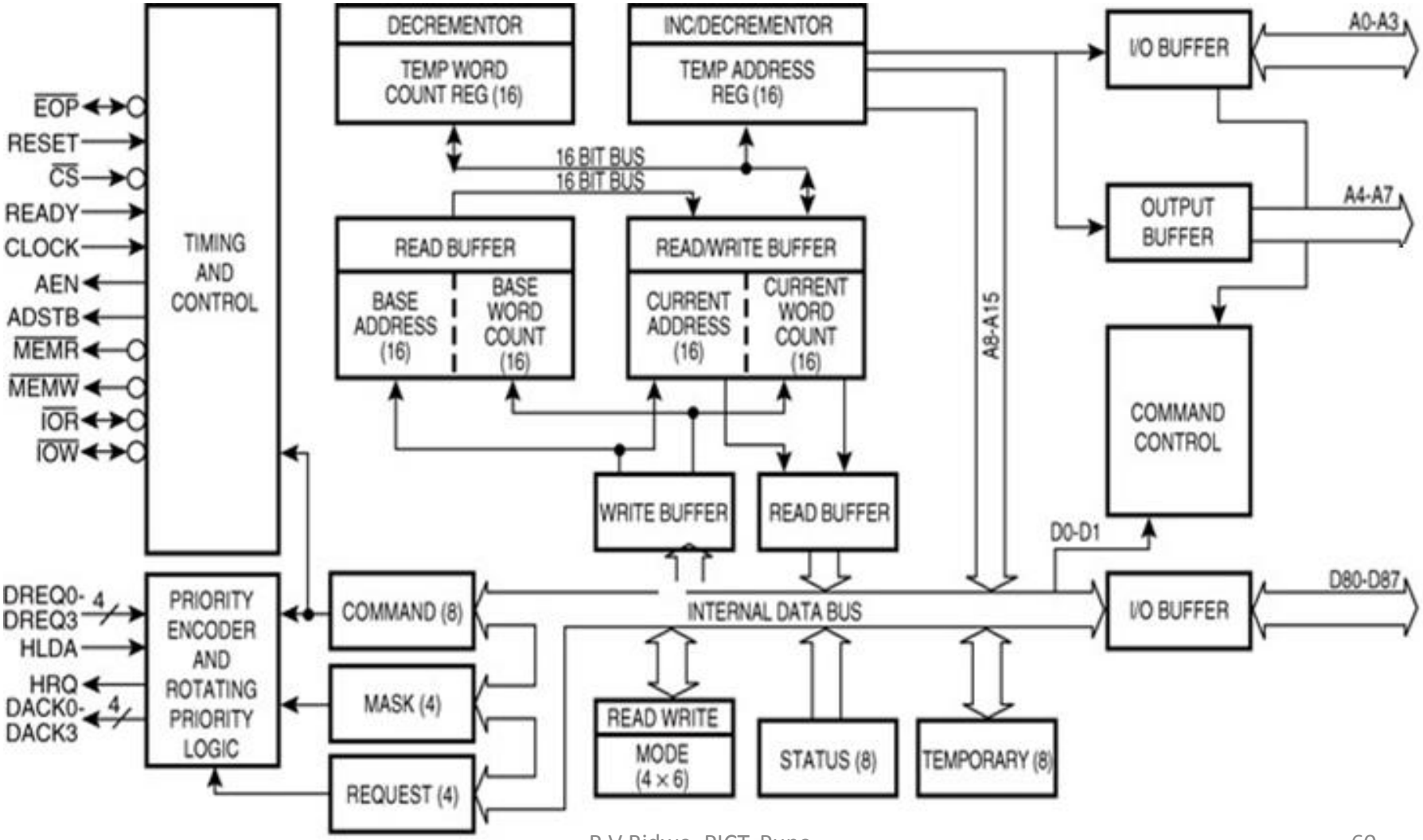
- The 8237 **supplies memory & I/O, the Control signals, Memory address information and Data** during the DMA transfer.

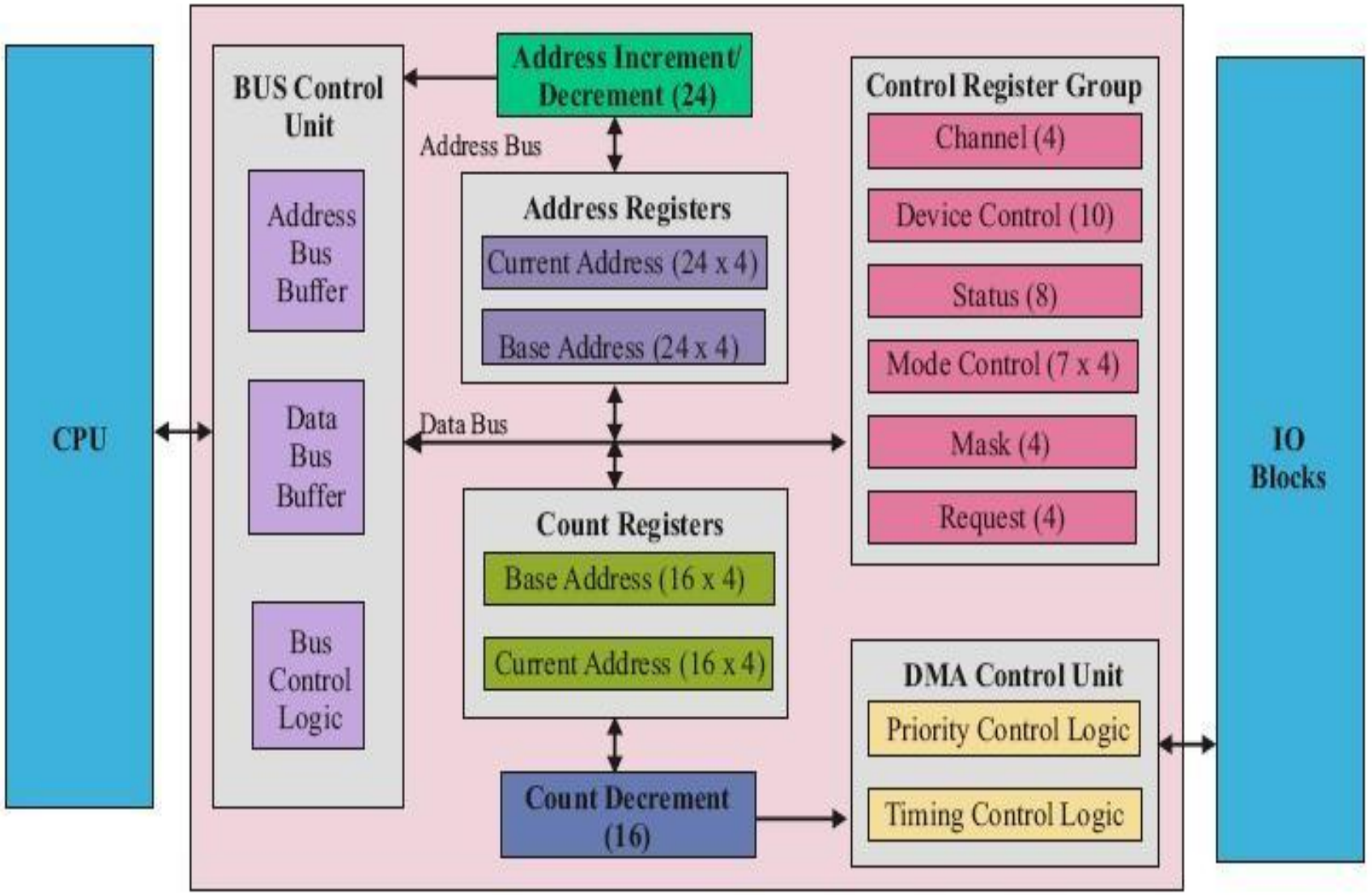
- It is actually a **Special-purpose Microprocessor whose job is high-speed data transfer** between memory and I/O.

- It **gives better performance than 8257.**

- It is **able to transmit bulk of data** between system memory and peripherals and vice versa.
- Memory to memory data transfer is also available.
- It supports **four independent DMA channels which may be expanded to any number by cascading more number of 8237.**
- Distinctive feature is that, **it provides many Programmable Control and ability of Dynamic Reconfiguration features which enhances data transfer rate of system.**

Block Diagram 8237




CPU
BUS Control Unit
Address Bus Buffer
Data Bus Buffer
Bus Control Logic
Address Increment/ Decrement (24)
Address Bus
Address Registers
Current Address (24 x 4)
Base Address (24 x 4)
Data Bus
Count Registers
Base Address (16 x 4)
Current Address (16 x 4)
Count Decrement (16)
Control Register Group
Channel (4)
Device Control (10)
Status (8)
Mode Control (7 x 4)
Mask (4)
Request (4)
DMA Control Unit
Priority Control Logic
Timing Control Logic
IO Blocks

8237 Internal Registers

CAR

- The **Current Address Register** holds a 16-bit memory address used for the DMA transfer.

CWCR

- The **Current Word Count Register** programs a channel for the number of bytes to transferred during a DMA action.

CR

- The **Command Register** programs the operation of the 8237 DMA controller.

- The register uses **Bit Position 0** to select the memory-to-memory DMA transfer mode.

  - Memory-to-Memory DMA transfers use DMA channel 0 to hold the source address.
  - DMA channel 1 holds the destination address.

BA and BWC

- The **Base Address (BA)** and **Base Word Count (BWC)** registers are used to hold the Base Address value and Initial Count value of a transfer. Also used when auto-initialization is selected for a channel.

- **In Auto-initialization Mode**, these registers are used to reload the CAR and CWCR Values.

MR

- The **Mode Register** programs the mode of operation for a channel.

- Each channel has its own mode register as selected by **Bit Positions 1 And 0**.
  - Bits of the mode register select Operation, Auto-initialization, Increment/Decrement, and Mode for the channel.

BR

- The **Bus Request Register** is used to request a DMA transfer via software.
  - very useful in memory-to-memory transfers, where an external signal is not available to begin the DMA transfer.

MRSR

- The **Mask Register Set/Reset** sets or clears the channel mask.
  - if the mask is set, the channel is disabled.
  - the RESET signal sets all channel masks to disable them.

MSR

- The **Mask Register** clears or sets all of the masks with one command instead of individual channels, as with the MRSR.

SR

- The **Status Register** shows status of each DMA channel. The **TC bits indicate if the channel has reached its Terminal Count** (transferred all its bytes).

- When the terminal count is reached, the DMA controller will enter in a Burst Mode.

Master clear

- Acts exactly the same as the RESET signal to the 8237.
  - As with the RESET signal, this command disables all channels

Clear Mask Register

- Enables all four DMA channels.

Clear The First/Last Flip-flop

- Clears the first/last (F/L) flip-flop within 8237.

- **The F/L flip-flop selects which byte (low or high order) is read/written in the current address and current count registers.**
  - if F/L = 0, the low-order byte is selected.
  - if F/L = 1, the high-order byte is selected.

- Any read or write to the address or count register automatically toggles the F/L flip-flop.

I/O Channels and Processors

- **The Evolution of the I/O Function**

The Evolution of the I/O Function

1. **The CPU directly controls a peripheral device.** This is seen in simple microprocessor- controlled devices.

2. **A controller or I/O module is added.** The CPU uses Programmed I/O without interrupts. With this step, the CPU becomes somewhat divorced from the specific details of external device interfaces. **(Programmable I/O)**

3. The same configuration as in step 2 is used, but now interrupts are employed. The CPU need not spend time waiting for an I/O operation to be performed, thus increasing efficiency. **(Interrupt-driven I/O)**

4. The I/O module is given direct access to memory via **DMA**. It can now move a block of data to or from memory without involving the CPU, except at the beginning and end of the transfer.

5. **The I/O module is enhanced to become a processor** in its own right, with a specialized instruction set tailored for I/O.

   The CPU directs the I/O processor to execute an I/O program in memory. The I/O processor fetches and executes these instructions without involving CPU.

   This allows the CPU to specify a sequence of I/O activities and to be interrupted only when the entire sequence has been performed.

6. **The I/O module has a local memory of its own** and is, in fact, a computer in its own right.

With this architecture, a large set of I/O devices can be controlled, with minimal CPU involvement.

A common use for such an architecture has been to control communication with interactive terminals. The I/O processor takes care of most of the tasks involved in controlling the terminals.