Basic I/O System Design

Department of Computer Science & Engineering
BRAC University.

Course ID: CSE - 341
Course Title: Microprocessors

Basic I/O System

- A **Microprocessor** is a great tool for solving problem but is of little or no use if it can't communicate with other devices.

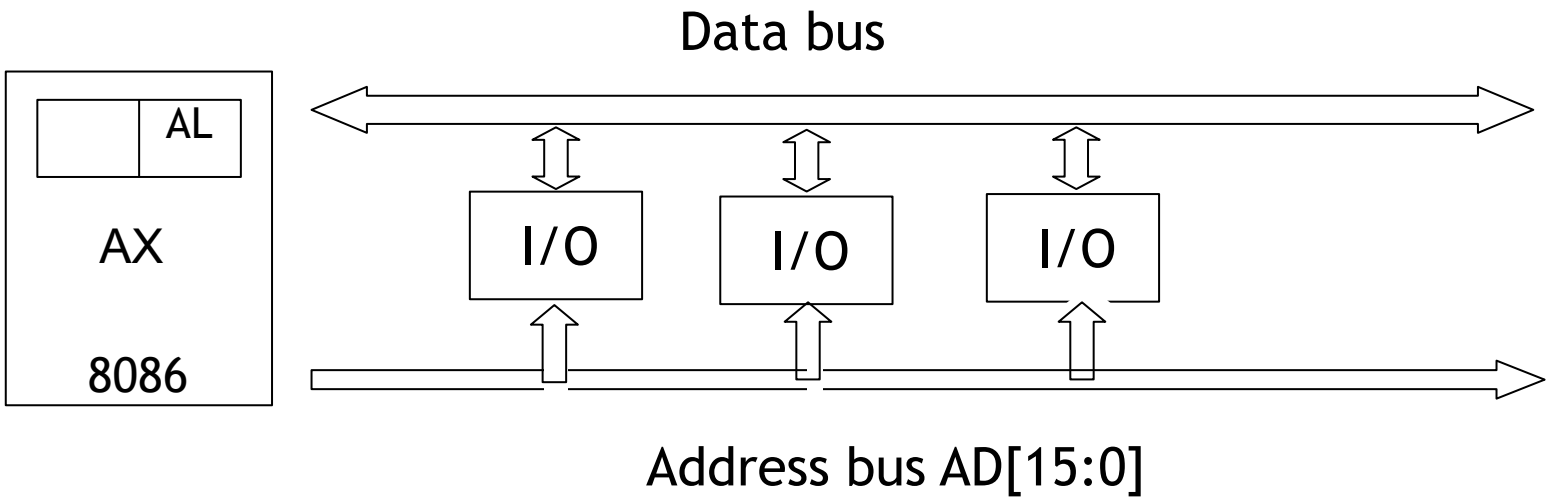
- **Input-Output devices (or peripherals)** such as Keyboards, Mouse, LEDs etc. are essential components of the microprocessor-based or microcontroller-based systems.

- **Input Devices**
  - Receive data from peripheral (i.e., device)
  - Send data to processor

- **Output Devices**
  - Receive data from processor
  - Send data to peripheral (i.e., device)

Basic I/O System

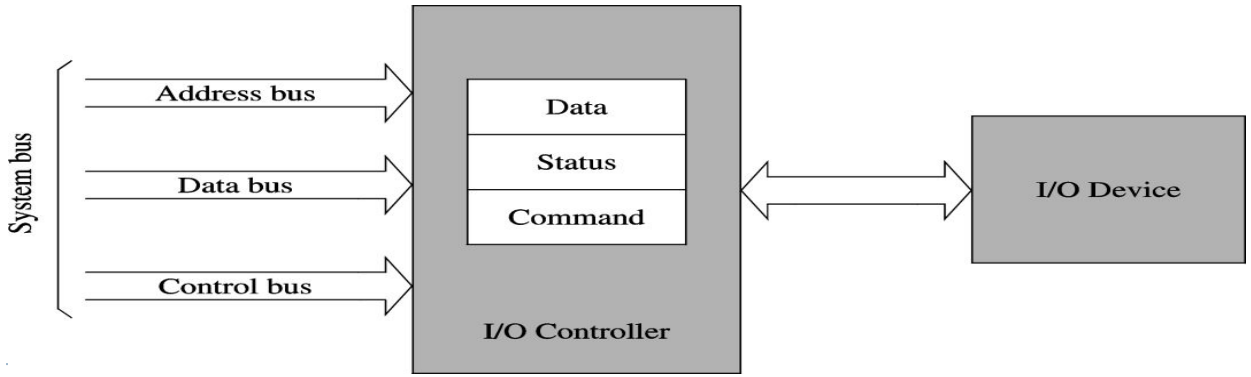
- 8086 processor uses address bus pins AD[15:0] to locate an I/O port
- 65,536 possible I/O ports
- Data transfer between ports and the processor occurs over data bus
- AL (or AX) is the processor register that takes input data (or provide output data)



Basic I/O System

- I/O devices serve two main purposes :
  - **To communicate with outside world**
  - **To store and transfer data**

- ***I/O Controller*** is a microchip that acts as an interface between the systems bus and I/O device and helps to facilitate the flow of data between these devices. For e.g., 8255
  - Relieve the processor of low-level details
  - Takes care of electrical interface
  - I/O Controller Chips have three types of registers : Data, Command and Status



Basic I/O System

- **Read Operation: To receive binary data from an input peripheral**
  - **MPU** places the address of an input port on the **address bus** to locate the I/O device, then it enables the input port by asserting the **RD signal**, and reads **data** using the **data bus**.

- **Write Operation: To send binary data to an output peripheral**
  - **MPU** places the address of an output port on the **address bus**, places **data** on **data bus**, and asserts the **WR signal** to enable the output port.

- **Important Points:**
  - Writing to the port : When the MPU sends out or transfers data to an output port
  - Reading from the port : When the MPU receives data from an input port
  - One port can be accessed at a time (Serial Interfacing)

I/O Instructions

- **I/O Instructions are commands given by the microprocessor to the I/O devices or peripherals in order to perform a specific task.**

  - **IN** is the instruction that reads information from an I/O device.
  - **OUT** is the instruction that writes/sends data to an I/O device.
  - Data transfer takes place between the microprocessor accumulator (AL or AX) and the I/O device .

- The I/O device may be identified using two methods:

  - **Fixed address –** A byte called **p8** immediately following the opcode stores an 8 bit I/O address. This is called fixed because this is stored with the opcode in the ROM.
  - **Variable address –** Register DX holds a 16 bit I/O address. Because this can be changed.

I/O Instructions

- **IN AX, 25 -** Here 25 (p8) refers to port 25 which is an 8 bit address (direct format)

- **IN AL, DX -** A byte input from the port addressed by DX into AL
- **IN AX, DX -** A word input from the port addressed by DX into AX (indirect format)

- If **p8** is used, only 8 bits are used thus the address or port number appears on lines A0-A7.
  - Hence, the first 2^8 or 256 I/O port addresses (00H – FFH) are accessed by both fixed and variable I/O instructions.

- If **DX** is used, then 16 bits are used thus the address or port number appears on lines A0-A15
  - Hence the addresses 0100H-FFFFH (257^{th} address to 65535^{th} address) are accessed only by variable I/O Address instructions.

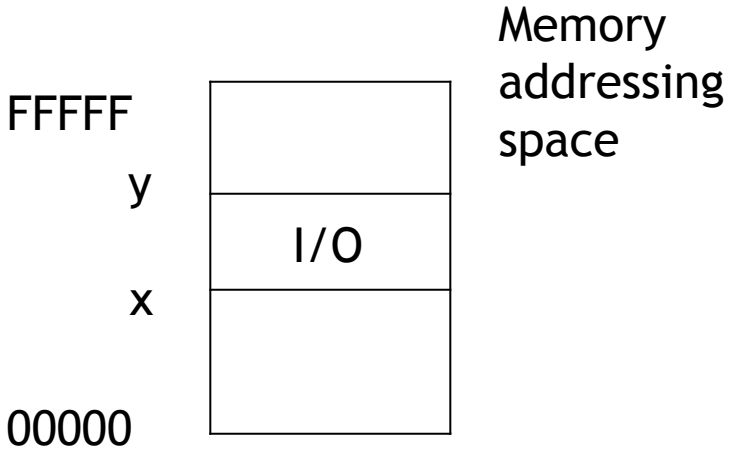
Accessing I/O Devices

To establish a stable passage of communication between the CPU and the I/O devices, the peripherals must first be mapped to specific addresses so that they can accessed. Also to enable the flow of data the help of the system bus (address, data and control) is also required. This allocation of the system bus and mapping the I/O devices to addresses is performed in two different ways :

- Memory-mapped I/O
- Isolated I/O

I/O Address Mapping

- Memory Mapped I/O
  - A part of the memory is used to map I/O devices
  - A device is mapped to a memory location. Sending data to the particular location causes interaction with the device.
  - Common address, data and control lines for both memory and I/O
  - Uses same memory read and write signals for memory and I/O
  - Most processors use this I/O mapping



I/O Address Mapping

Memory Mapped I/O

Advantages :

- Less complication
- Less circuitry
- Same instructions such as MOV can be used for both memory and I/O
- Less decoding

.

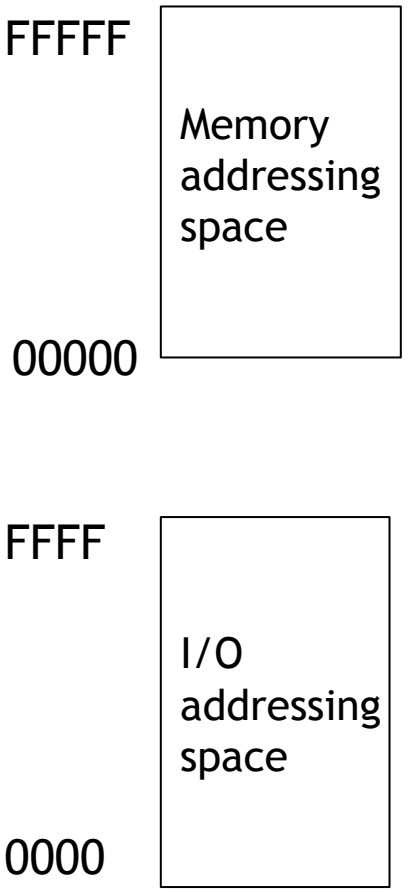
Disadvantages :

- A portion of the memory system is used as the I/O Map

I/O Address Mapping

Isolated I/O

- Separate I/O address space
- Common data and address bus for peripherals, CPU and memory
- But separate or isolated control signals for I/O and memory.
- $\overline{\text{IORC}}$ and $\overline{\text{IOWC}}$ signals are for I/O devices.
- $\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ signals are for memory
- This is the most common form of I/O transfer technique used with Intel processors and PC.
- Pentium supports isolated I/O of 64 KB address space.



I/O Address Mapping

Isolated I/O

Advantages:

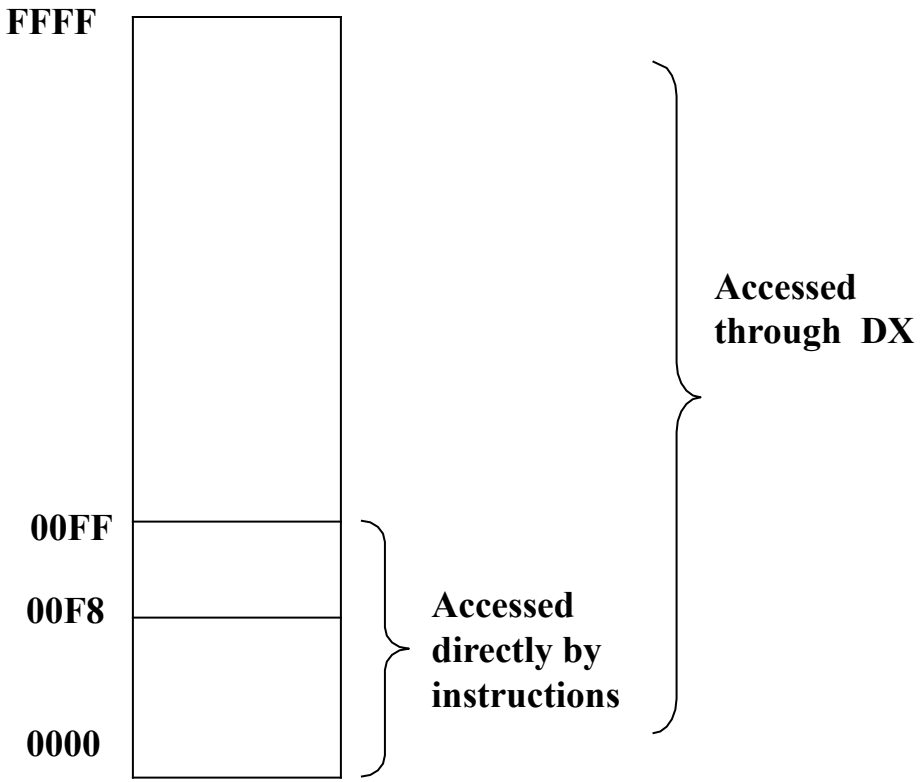
- In this system no memory is wasted for I/O mapping.

Disadvantages:

- Separate signals are also needed to interact with the I/O devices which separate it from normal memory access instructions.
- Instructions **IN** and **OUT** need to be used to perform data transfer.

Accessing I/O Devices in 8086

Addressing Space



- **Accessing directly by instructions**

```
IN      AL,     80 H
IN      AX,     6 H
OUT     3C H,   AL
OUT     A0 H,   AX
```

- **Accessing through DX**

```
IN    AX, DX
OUT   DX, AL
OUT   DX, AX
```

I/O Data Transfer Techniques

Different ways of transfer of data between the CPU and the I/O devices, namely:

- **Programmed I/O**
- **Interrupt-driven I/O**
- **Direct memory access (DMA)**

Programmed I/O

Transfer of data between CPU and I/O using programmed I/O:

- Invoke read / write command to peripheral
- Check ready status of I/O device
- If not ready, continue checking status until ready
- If ready, initiate operation
- Thus transfer of data takes place

Key points to remember:

- CPU waits for I/O device to complete the operation if its not ready
- Programmed I/O Wastes CPU time
- CPU performance is degraded

Now to improve this situation to some extent, we will use Interrupt-Driven I/O

Interrupt-driven I/O

Transfer of data between CPU and I/O using programmed I/O:

- Invoke read / write command to peripheral
- Now instead of waiting for I/O device, it continues it's other tasks
- I/O device when ready, gives an Interrupt signal to CPU signaling its ready for data transfer
- Since I/O device is ready ready, operation is initiated.
- Thus transfer of data takes place

Key points to remember:

- CPU does not continuously wait for I/O device
- I/O device itself alerts the CPU when it is ready
- CPU performance is better than programmed I/O

But there is still room for making CPU performance more efficient

Why do we require an alternate?

- Interrupt driven and programmed I/O require active CPU intervention.
- Makes the CPU wait
- Requires CPU's attention during data transfer
- CPU is tied up and processing time is being wasted

DMA is the answer

Direct Memory Access (DMA)

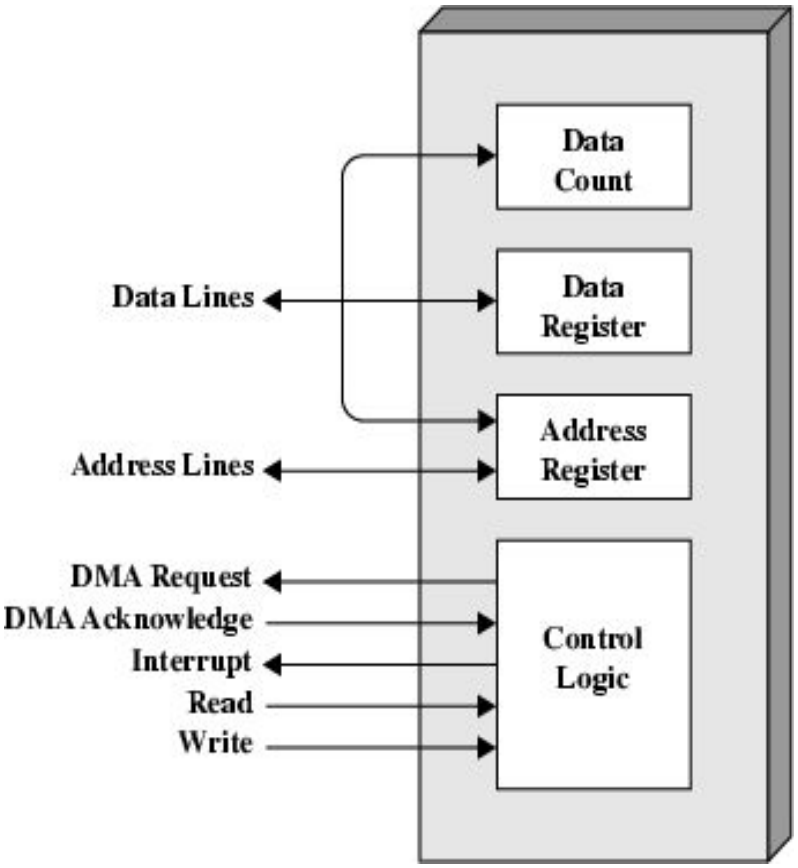
Direct Memory Access (DMA) is a process of data transfer between the memory and I/O devices. An external device takes over the control of system bus from the CPU and facilitates with the data transfer.

- The basic idea of **DMA** is to transfer blocks of data directly between memory and peripherals. The data does not have to go through the microprocessor.
- DMA controller can perform this task, relieving the CPU from the burden of transferring data.
- The CPU just issues the command for data transfer and shift its focus on actual data processing while the DMA handles the transfer of data
- Can be used for **high-speed data transfer** between memory and peripherals such as HDD, magnetic tape etc.

Nowadays, DMA can transfer data as fast as 40-60 M byte per second.

DMA Controller

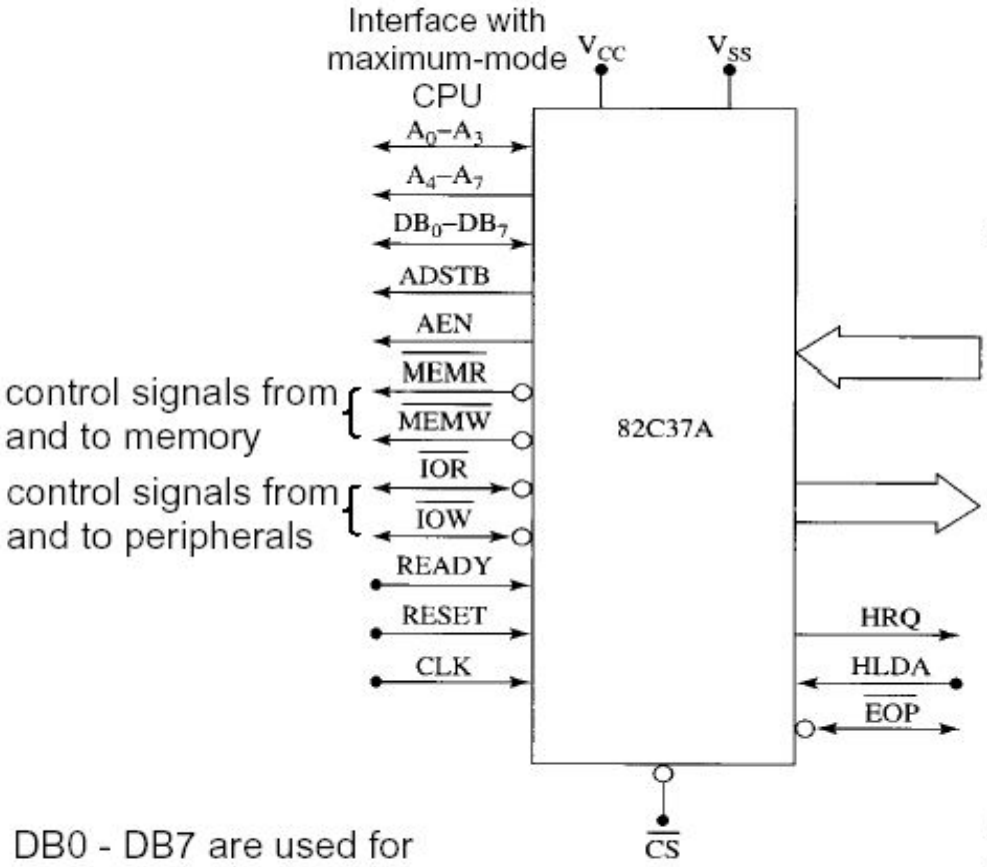
- A DMA controller is connected with several peripherals that may request DMA.
- The controller can decides the priority of the DMA requests, can communicate with the I/O devices and the CPU, and provides memory addresses for data transfer.
- For e.g. the Intel 8237 is a DMA controller
- It is a 4-channel device. Each channel is dedicated to a specific I/O device
- It is capable of addressing 64KB section of memory



Typical DMA Module Diagram

8237 DMA Controller

8237 DMA controller



DB0 - DB7 are used for

1) transfer of data
2) 8237 programming

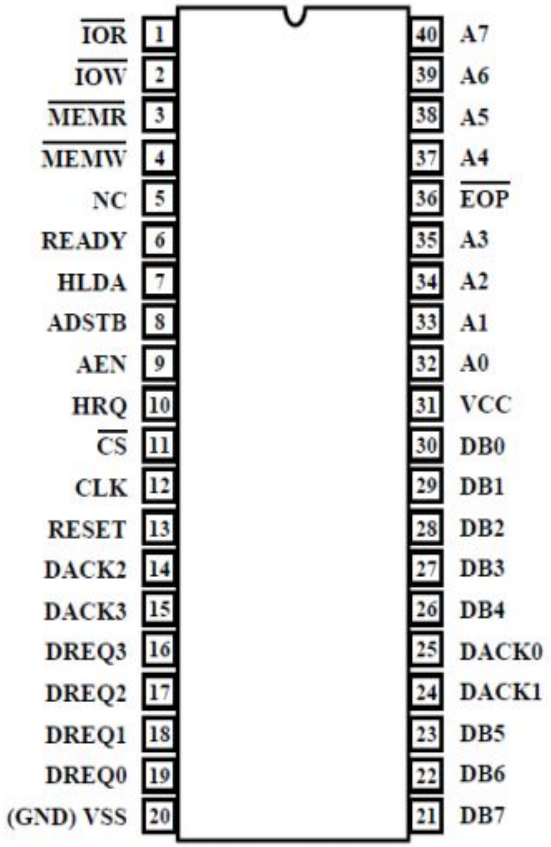
A0 - A3 are used for

1) accessing 8237 internal ports
2) carrying memory address in DMA read and write operations

8237 DMA Controller

Pin Description:

- **DREQ3 – DREQ0 (DMA channel request):** Used to request a DMA transfer from a particular DMA channel.
- **DACK3 – DACK0 (DMA channel acknowledge):** Acknowledges a channel DMA request from a device.
- **HRQ (Hold request):** Requests control of the system bus to the CPU.
- **HLDA (Hold acknowledge):** Signals the 8237 that the microprocessor has acknowledged the hold request and relinquished control of the system bus.



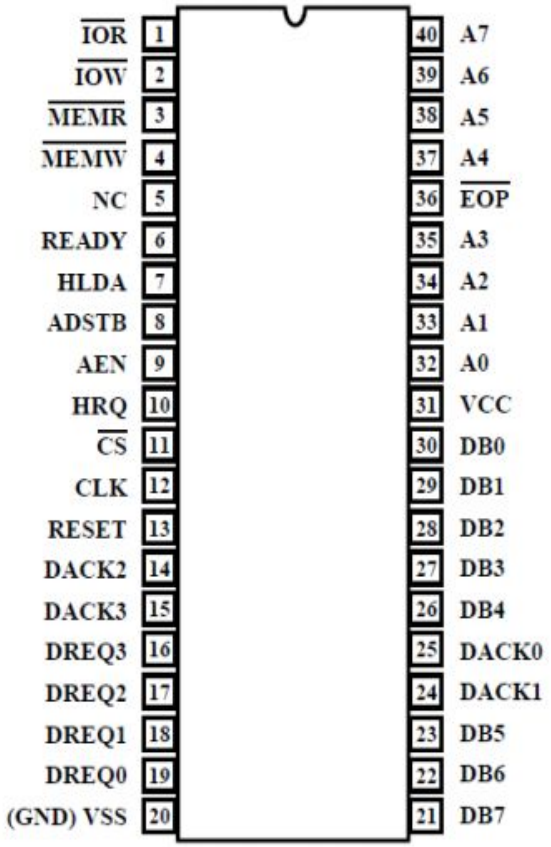
8237 DMA Controller

- **AEN (Address enable):** Enables the DMA address latch connected to the 8237 (Use to take the control of the address bus from the microprocessor)

- **ADSTB (Address strobe):** It is used to latch the higher 8 bits of the address during DMA operation.

- **EOP (End of process):** It is a bi directional pin that signals the end of the DMA process.

- **IOR (I/O read):** A bi-directional pin which the CPU can use to read data from the internal registers of 8237 and can be used as an output to read data from a peripheral during a DMA write cycle.



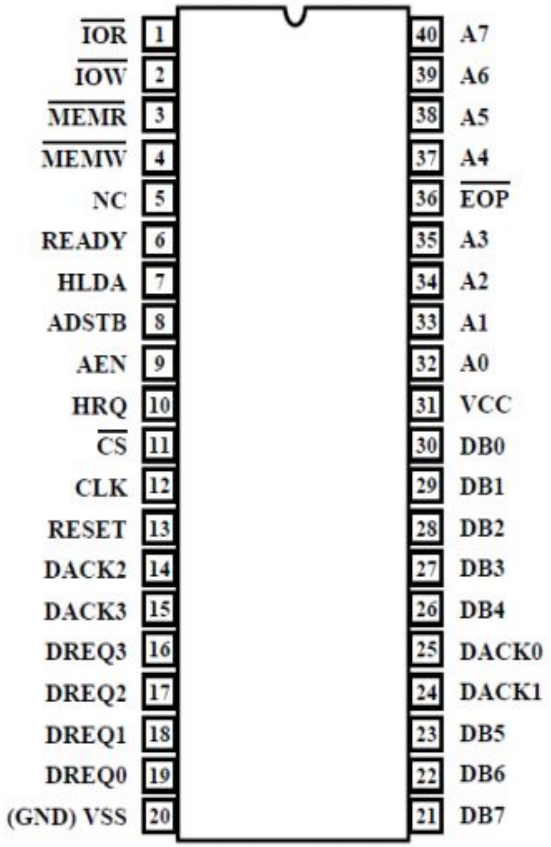
8237 DMA Controller

- **$\overline{\text{IOW}}$ (I/O write** A bi-directional pin which the CPU can use to write data to the internal registers of 8237 and can be used as an output strobe to write data to a peripheral during a DMA read cycle.

- **$\overline{\text{MEMW}}$ (Memory write):** Used to write data to the ~~selected~~ memory during a DMA write cycle.

- **$\overline{\text{MEMR}}$ (Memory read):** Used to read data from the selected memory during a DMA read cycle.

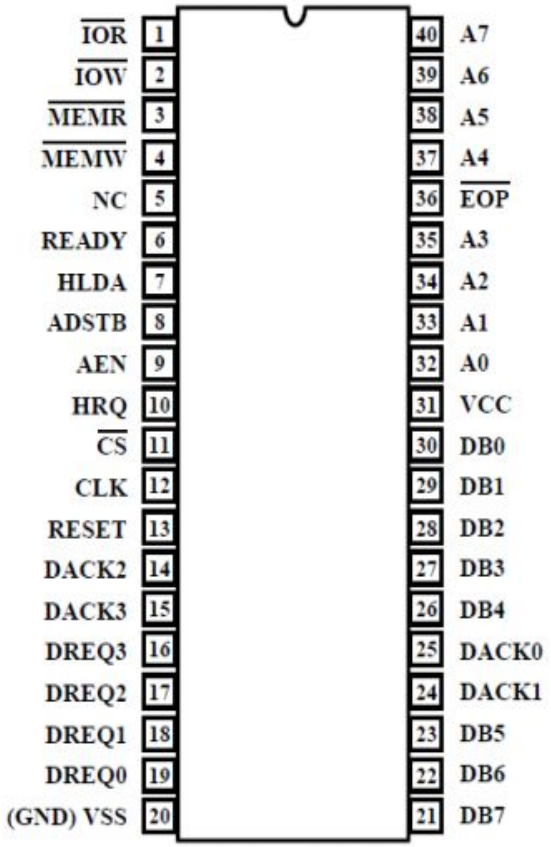
- **A3 – A0:** They are bi directional pins. They can be used to select the internal registers of the 8237 by the CPU . Otherwise holds the lower nibble of the lower 8 bits of the memory address from where data is to be read / write during DMA operation.



8237 DMA Controller

- **A7 – A4:** Used to provide the higher nibble of the lower 8 bits of the memory address from where data is to be read / write during DMA operation.

- **DB0 – DB7:** Carries data during data transfer. Otherwise stores the higher 8 bits of the memory address from where data is to be read / write during DMA operation.



DMA Operation

- CPU invokes a read/write command to the peripherals
- The peripherals when ready sends a DMA request (DREQ) signal to the DMA controller
- The DMA controller sends a Hold Request (HRQ) signal to the CPU to gain control of the system bus
- The CPU replies with a Hold Acknowledgement signal to the DMA giving the DMA controller total control over the system bus
- The DMA controller then sends a DMA Acknowledgement signal to the peripherals in reply to the previous DMA request signal
- Data can now be transferred between Memory and I/O with the help of the DMA controller
- DMA controller sends interrupt signal to the CPU when finished with data transfer.
- The DMA disables the DMA Acknowledgement signal and the CPU disables the Hold Acknowledgement signal taking back control of the system bus.

DMA Operation

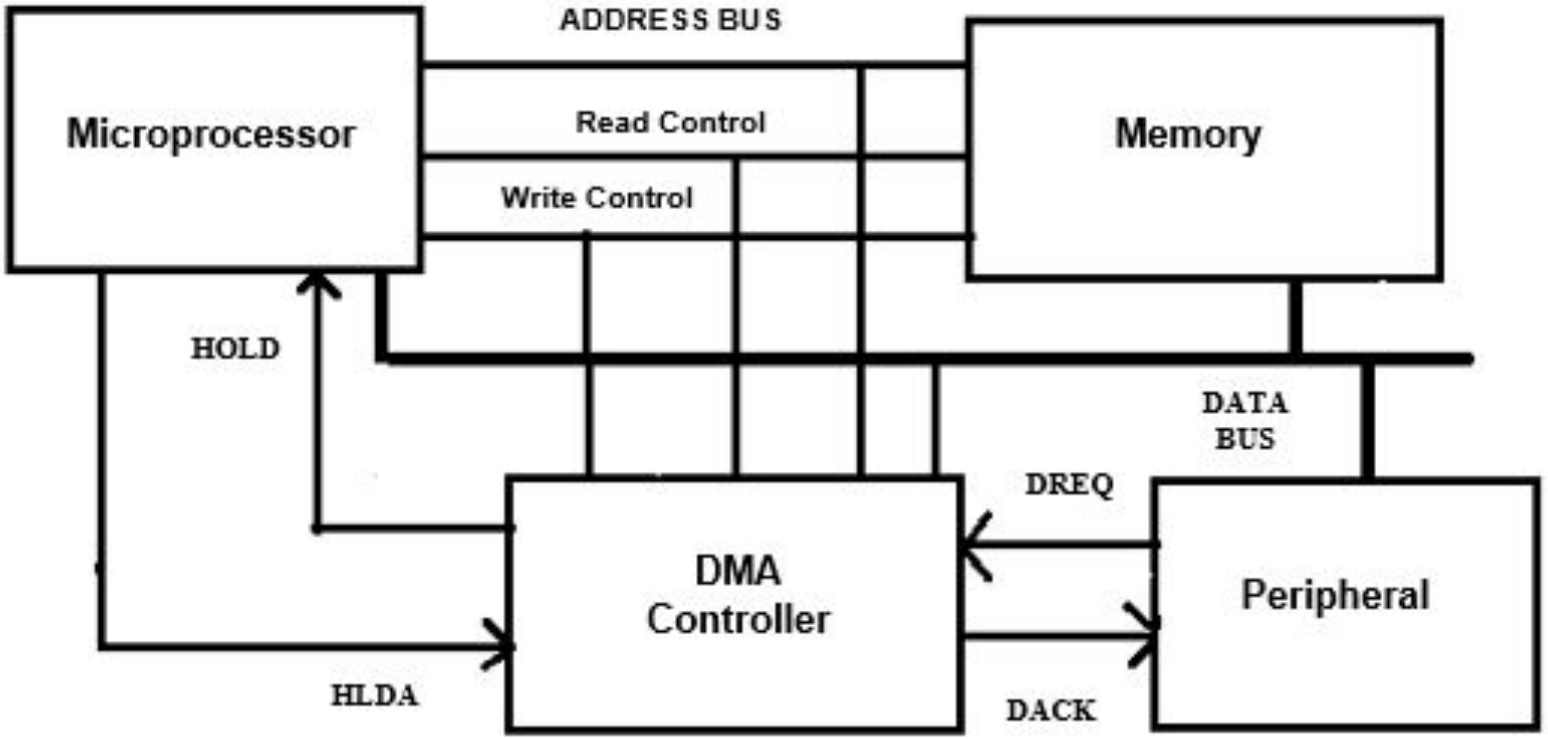
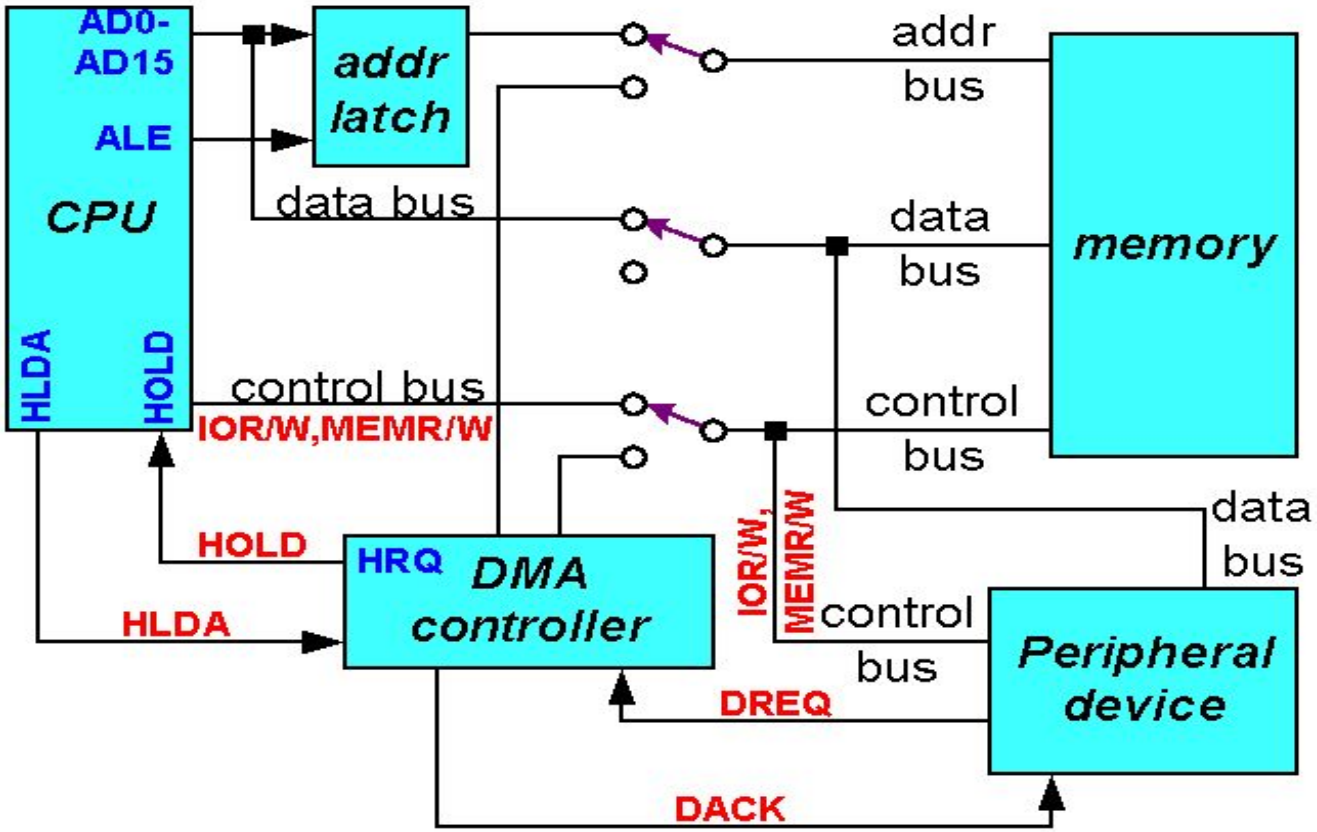


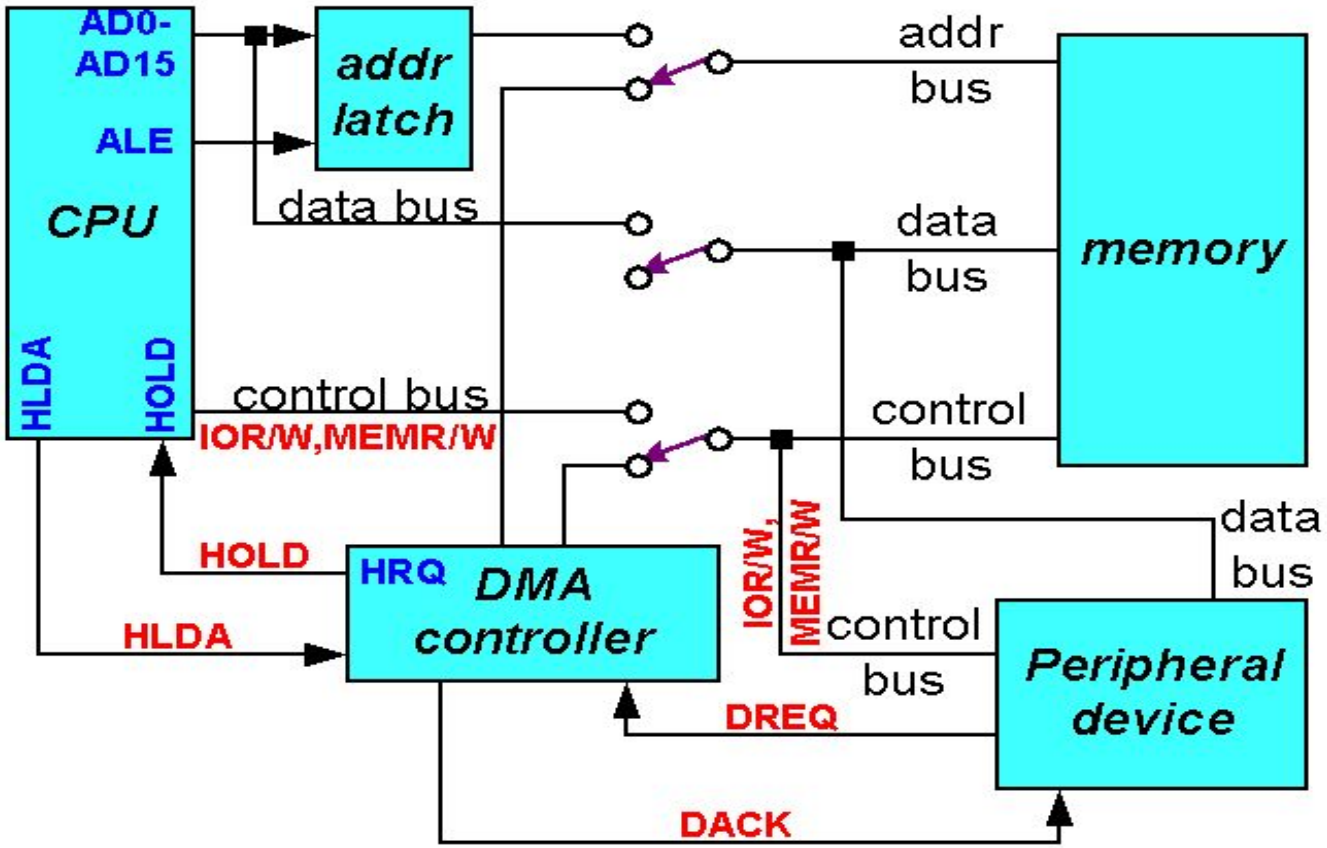
Fig : Simplified Block Diagram of a DMA Operation

DMA Operation



When the CPU is Bus Master

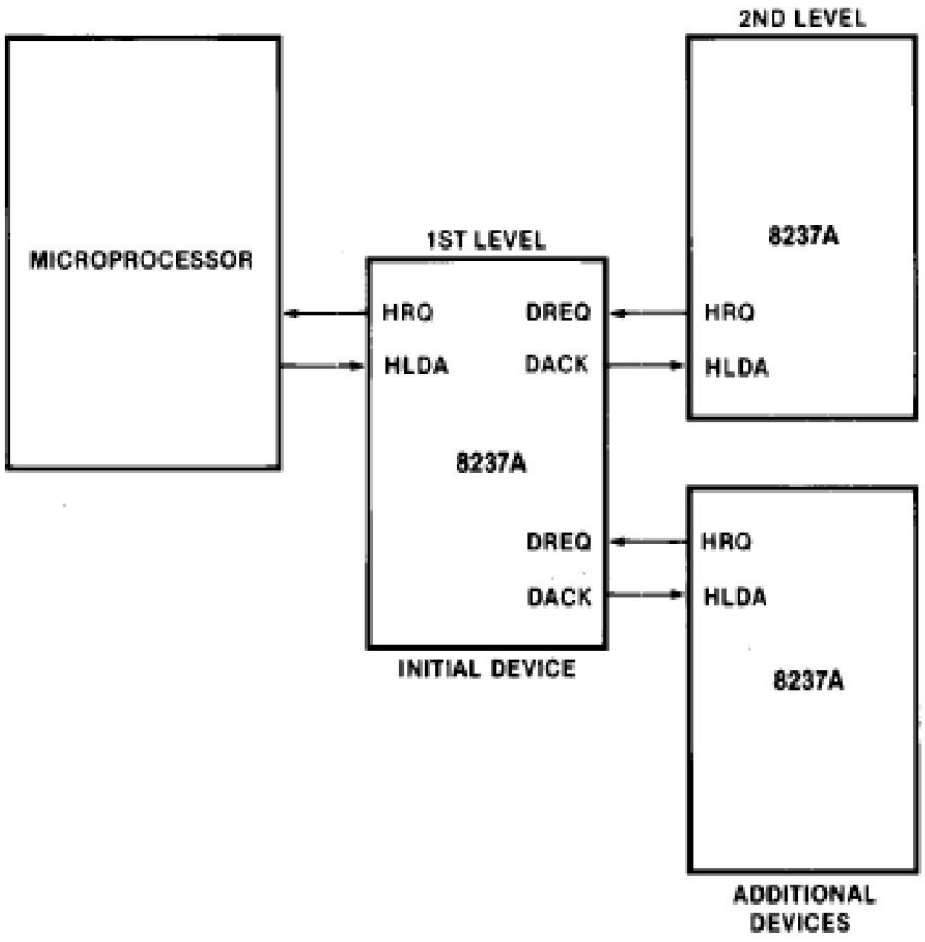
DMA Operation



When DMA takes control

DMA Cascade Mode

- Using this method more than one 8237 DMA Controller can be connected.
- This can be adopted to get the utility of more than 4 channels
- The HRQ and HLDA signals from the additional controllers are connected with the DREQ and DACK pins of the primary or “Host” controller
- It is the responsibility of the host to prioritize and co ordinate the secondary controllers.



Thank You !!