



**INSTITUTE OF AERONAUTICAL ENGINEERING  
(Autonomous)**

Dundigal, Hyderabad -500 043

**INFORMATION TECHNOLOGY**

**COURSE LECTURE NOTES**

Course Name	MICROPROCESSORS INTERFACING AND APPLICATIONS
Course Code	AEC023
Programme	B.Tech
Semester	VI
Course Coordinator	Ms. B Lakshmi Prasanna, Assistant Professor
Course Faculty	Ms. B Lakshmi Prasanna, Assistant Professor
Lecture Numbers	1-157
Topic Covered	All

**COURSE OBJECTIVES (COs):**

The course should enable the students to:	
I	Understand the concept of microprocessor and familiarize the architecture of 8085 and 8086 processor.
II	Analyze the assembly language programming using 8086 microprocessor.
III	Develop the knowledge of microprocessor based systems and interfacing techniques
IV	Understand the concept of Interrupts and their significance in 8086.
V	Impart the basic concepts of serial and parallel bus standards.
VI	Understand the basic concept of advanced processor architectures.

**COURSE LEARNING OUTCOMES (CLOs):**

Students, who complete the course, will have demonstrated the ability to do the following:

AEC023.01	Differentiate between 8085 and 8086 microprocessors architectures and its functionalities.
AEC023.02	Describe the internal Architecture of 8086 microprocessor and explain its functionalities.
AEC023.03	Describe in detail about functions of general purpose register and 8086 flag register with its functions.
AEC023.04	Explain various addressing modes and instruction set present in 8086 microprocessors and Describe in detail about the concept of interrupt, types of interrupts 8086 microprocessor.
AEC023.05	Understand and apply the fundamentals and procedures and assembler directives of assembly level programming of microprocessors.

AEC023.06	Develop low level languages like ALP in 8086 microprocessor systems for real time applications
AEC023.07	Describe Minimum mode and maximum mode of operation and timing diagram of 8086 microprocessor
AEC023.08	Explain various Assembly language programs involving logical, branch and call instructions.
AEC023.09	Evaluation of arithmetic expressions, string manipulation, sorting using various Assembly language programs.
AEC023.10	Identify the importance of various modes of 8255 operation and interfacing to 8086.
AEC023.11	Discuss the interfacing diagram of I/O devices with keyboard, stepper motor, 7-segment display, LCD and digital to analog and analog to digital converter.
AEC023.12	Explain in detail about the importance of DMA, interrupt and interrupt sub routines in 8086 microprocessor
AEC023.13	Analyze and understand various synchronous and asynchronous serial data transfer schemes in 8086.
AEC023.14	Develop and design the interfacing circuit diagram of 8251 USART with 8086 processor.
AEC023.15	Understand the high- speed serial communications standards, USB.
AEC023.16	Understand basic architecture of 16 bit and 32 bit microprocessors with the help of GDT, LDT and multitasking and addressing modes.
AEC023.17	Flag register 80386: Architecture, register organization, memory access in protected mode
AEC023.18	Analyze the various advanced microprocessors internal architectures for 80X86 by paging and technical features.

## SYLLABUS

<b>Unit-I</b>	<b>OVERVIEW OF 8086 MICROPROCESSOR</b>	<b>Classes: 11</b>
Introduction to 8085 microprocessor. RISC and CISC processors, architecture of 8086 microprocessor, special functions of general purpose register, 8086 flag register and function of 8086 flags, addressing modes of 8086, instruction set of 8086, assembler directives.		
<b>Unit -II</b>	<b>8086 ASSEMBLY LANGUAGE PROGRAMMING</b>	<b>Classes: 10</b>
Minimum mode and maximum mode of operation, timing diagram, Assembly language programs: Assembly language programs involving logical, branch and call instructions, sorting, evaluation of arithmetic expressions, string manipulation.		
<b>Unit -III</b>	<b>8255 PROGRAMMABLE PERIPHERAL INTERFACE (PPI)</b>	<b>Classes: 10</b>
Various modes of 8255 operation and interfacing to 8086; Interfacing keyboard, displays, 8279 Stepper motor and actuators, digital to analog and analog to digital converter interfacing.		
Interrupt structure of 8086: Interrupt structure of 8086, Vector interrupt table, interrupt service routines; Introduction to DOS and BIOS interrupts, need for DMA, DMA data transfer method, interfacing with 8237/8257.		

<b>Unit –I V</b>	<b>SERIAL DATA TRANSFER SCHEMES</b>	<b>Classes: 10</b>
Asynchronous and synchronous data transfer schemes, 8251 USART architecture and interfacing; TTL to RS 232C and RS232C to TTL conversion; Sample program of serial data transfer; Introduction to high-speed serial communications standards, USB.		
<b>Unit -V</b>	<b>ADVANCED MICROPROCESSORS</b>	<b>Classes: 09</b>
80286 microprocessor: Architecture, registers (Real/Protected mode), privilege levels, descriptor cache, memory access in GDT and LDT, multitasking, addressing modes; Flag register 80386: Architecture, register organization, memory access in protected mode, paging; 80486: Only the technical features.		
<b>Text Books:</b>		
<ol style="list-style-type: none"> <li>1. D. V. Hall, “Microprocessors and Interfacing”, Tata McGraw-Hill Education, 3<sup>rd</sup> Edition 2013.</li> <li>2. A.K Ray, K. M. Bhurchandani, “Advanced Microprocessors and Peripherals” Tata McGraw-Hill Education, 2<sup>nd</sup> Edition, 2006.</li> <li>3. Savaliya M. T, “8086 Programming and Advance Processor Architecture”, Wiley India Pvt., 1<sup>st</sup> Edition, 2012.</li> </ol>		
<b>Reference Books:</b>		
<ol style="list-style-type: none"> <li>1. N. Senthil Kumar, M. Saravanan, S.Jeevanathan, S. K. Shah, “Microprocessors and Interfacing”, Oxford University, 1<sup>st</sup> Edition, 2012.</li> <li>2. Lyla B. Das, “The x86 Microprocessors”, Pearson India, 2<sup>nd</sup> Edition, 2014.</li> <li>3. Daniel Tabak, –Advanced Microprocessors, Addison-Wesley, 2<sup>nd</sup> Edition, 1996.</li> <li>4. Triebel, Singh, –The 8088 and 8086 Microprocessors, PHI, 4<sup>th</sup> Edition 2003.</li> </ol>		
<b>Web References:</b>		
<ol style="list-style-type: none"> <li>1. <a href="http://www.daenotes.com/electronics/digital-electronics/Intel-8085-8-bitmicroprocessor# axzz">http://www.daenotes.com/electronics/digital-electronics/Intel-8085-8-bitmicroprocessor# axzz</a></li> <li>2. <a href="http://www.alljntuworld.in/wp-content/uploads/2015/12/Microprocessors-and-Interfacing-Devices.pdf">http://www.alljntuworld.in/wp-content/uploads/2015/12/Microprocessors-and-Interfacing-Devices. p df</a></li> <li>3. <a href="https://www.smartworld.com/notes/microprocessors-and-microcontrollers-mpmc/">https://www.smartworld.com/notes/microprocessors-and-microcontrollers-mpmc/</a></li> </ol>		
<b>E-Text Books:</b>		
<ol style="list-style-type: none"> <li>1. <a href="http://www.freepdfbook.com/micro-processors-and-interfacing/">http://www.freepdfbook.com/micro-processors-and-interfacing/</a></li> <li>2. <a href="http://engineersevanigam.blogspot.in/2013/07/microprocessors-and-interfacing-by.html">http://engineersevanigam.blogspot.in/2013/07/microprocessors-and-interfacing-by.html</a></li> <li>3. <a href="https://www.scribd.com/doc/153593067/Microprocessor-by-A-P-Godse-D-A-Godse">https://www.scribd.com/doc/153593067/Microprocessor-by-A-P-Godse-D-A-Godse</a></li> </ol>		

## UNIT-I

### OVERVIEW OF 8086 MICROPROCESSOR

#### Introduction to processor:

- A processor is the logic circuitry that responds to and processes the basic instructions that drives a computer.
- The term processor has generally replaced the term central processing unit (CPU). The processor in a personal computer or embedded in small devices is often called a microprocessor.
- The **processor** (CPU, for Central Processing Unit) is the computer's brain. It allows the processing of numeric data, meaning information entered in binary form, and the execution of instructions stored in memory.

#### Evolution of Microprocessor:

A microprocessor is used as the CPU in a microcomputer. There are now many different microprocessors available.

- Microprocessor is a program-controlled device, which fetches the instructions from memory, decodes and executes the instructions. Most Micro Processor are single- chip devices.
- Microprocessor is a backbone of computer system. which is called CPU
- Microprocessor speed depends on the processing speed depends on DATA BUS WIDTH.
- A common way of categorizing microprocessors is by the no. of bits that their ALU can Work with at a time
- The address bus is unidirectional because the address information is always given by the Micro Processor to address a memory location of an input / output devices.
- The data bus is Bi-directional because the same bus is used for transfer of data between Micro Processor and memory or input / output devices in both the direction.
- It has limitations on the size of data. Most Microprocessor does not support floating-point operations.
- Microprocessor contain ROM chip because it contain instructions to execute data.
- What is the primary & secondary storage device? - In primary storage device the
- Storage capacity is limited. It has a volatile memory. In secondary storage device the storage capacity is larger. It is a nonvolatile memory.
  - a) Primary devices are: RAM (Read / Write memory, High Speed, Volatile Memory) / ROM (Read only memory, Low Speed, Non Voliate Memory)
  - b) Secondary devices are: Floppy disc / Hard disk

**Compiler:** Compiler is used to translate the high-level language program into machine code at a time. It doesn't require special instruction to store in a memory, it stores automatically. The Execution time is less compared to Interpreter.

### **1.4-bit Microprocessor:**

- The first **microprocessor** (Intel 4004) was invented in 1971. It was a 4-bit calculation device with a speed of 108 kHz. Since then, microprocessor power has grown exponentially. So what exactly are these little pieces of silicone that run our computers(" Common Operating Machine Particularly Used For Trade Education And Research ")
- It has 3200 PMOS transistors.
- It is a 4-bit device used in calculator.

### **2.8-Bit microprocessor:**

- In 1972, Intel came out with the 8008 which is 8-bit.
- In 1974, Intel announced the 8080 followed by 8085 is a 8-bit processor Because 8085 processor has 8 bit ALU (Arithmetic Logic Review). Similarly 8086 processor has 16 bit ALU. This had a larger instruction set then 8080. used NMOS transistors, so it operated much faster than the 8008.

The 8080 is referred to as a "Second generation Microprocessor"

### **3. Limitations of 8 Bit microprocessor:**

- Low speed of execution
- Low memory addressing capability
- Limited number of general purpose registers
- Less power full instruction set

### **4. Examples for 4/ 8 / 16 / 32 bit Microprocessors:**

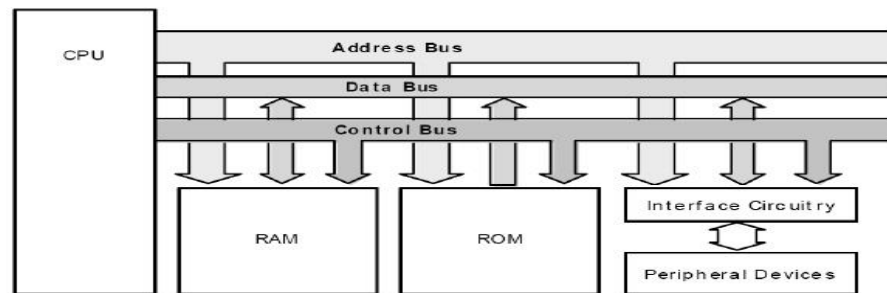
- 4-Bit processor – 4004/4040
- 8-bit Processor - 8085 / Z80 / 6800
- 16-bit Processor - 8086 / 68000 / Z8000
- 32-bit Processor - 80386 / 80486

### **5. What are 1st / 2nd / 3rd / 4th generation processor?**

- The processor made of PMOS technology is called 1<sup>st</sup> generation processor, and it is made up of 4 bits
- The processor made of NMOS technology is called 2<sup>nd</sup> generation processor, and it is made up of 8 bits
- The processor made of CMOS technology is called 3rd generation processor, and it is made up of 16 bits
- The processor made of HCMOS technology is called 4<sup>th</sup> generation processor, and it is made up of 32 bits (**HCMOS** : High-density n- type Complementary Metal Oxide Silicon field effect transistor)

## Block diagram of microprocessor:

### Microcomputer Block Diagram



#### The Central Processing Unit (CPU):

This device coordinates all operations of a micro computer. It fetches programs stored in ROM's or RAMs and executes the instructions depending on a specific Instructions set, which is characteristic of each type of CPU, and which is recognized by the CPU.

#### The Random Access Memory (RAM): Temporary or trail programs are written.

Besides the ROM area, every computer has some memory space for temporary storage of data as well as for programs under development. These memory devices are RAMs or Read – write memory. The contents of it are not permanent and are altered when power is turned off. So the RAM memory is considered to be volatile memory.

#### The Read Only Memory (ROM): Permanent programs are stored.

The permanent memory device/area is called ROM, because whatever be the memory contents of ROMs, they cannot be over written with some other information.

For a blank ROM, the manufacturer supplies the device without any inf. In it, information can be entered electrically into the memory space. This is called burning a ROM or PROM.

#### Data Lines/Data Bus:

The number of data lines, like add. Lines vary with the specific CPU. The set of data lines is database like the address bus unlike add. Bus, the data bus is bidirectional because while the information on the address Bus always flows out of the CPU; the data can flow both out of the CPU as well as into the CPU.

#### Control lines/ control Bus:

The no. of control lines also depends on the specific CPU one is using.

Ex: Read; Write lines are examples of control lines

**Clock:** The clock is a symmetrical square wave signal that drives the CPU

**Instructions:** An **instruction** is an elementary operation that the processor can accomplish. Instructions are stored in the main memory, waiting to be processed by the processor. An instruction has two fields:

- **Operation code**, which represents the action that the processor must execute;

**Operand code**, which defines the parameters of the action. The operand code depends on the operation. It can be data or a memory address

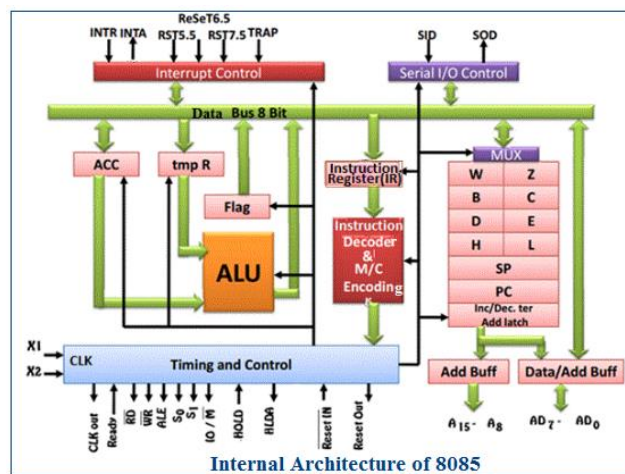
#### Introduction to 8085 Microprocessor:

##### The Salient Features of 8085 Microprocessor:

- 8085 is an 8 bit microprocessor, manufactured with N-MOS technology.

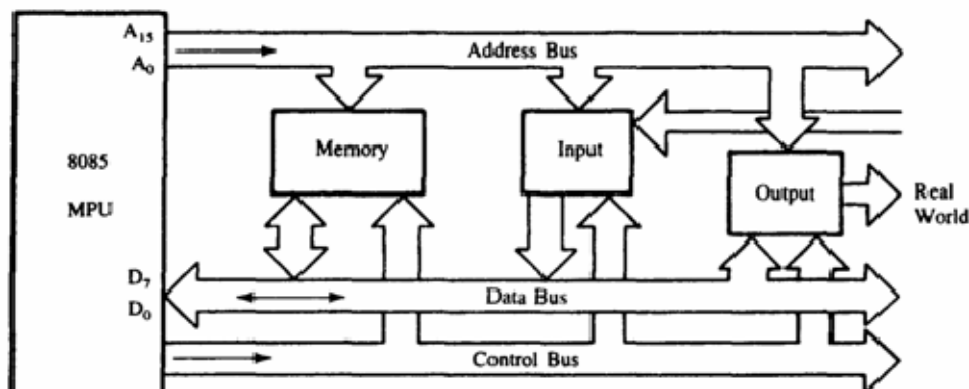
- It has 16-bit address bus and hence can address up to  $2^{16} = 65536$  bytes (64KB) memory locations through A<sub>0</sub>-A<sub>15</sub>.
- The first 8 lines of address bus and 8 lines of data bus are multiplexed AD<sub>0</sub> - AD<sub>7</sub>. Data bus is a group of 8 lines D<sub>0</sub> - D<sub>7</sub>.
- It supports external interrupt request. 8085 consists of 16 bit program counter (PC) and stack pointer (SP).
- Six 8-bit general purpose register arranged in pairs: BC, DE, HL.
- It requires a signal +5V power supply and can operate at 3 MHz, 5 MHz and 6 MHz Serial in/Serial out Port.
- It is enclosed with 40 pins DIP (Dual in line package).

### Internal Architecture of 8085:



### 8085 Bus Structure: Address Bus:

- The address bus is a group of 16 lines generally identified as A<sub>0</sub> to A<sub>15</sub>.
- The address bus is unidirectional: bits flow in one direction-from the MPU to peripheral devices.
- The MPU uses the address bus to perform the first function: identifying a peripheral or a memory location.



### Data Bus:

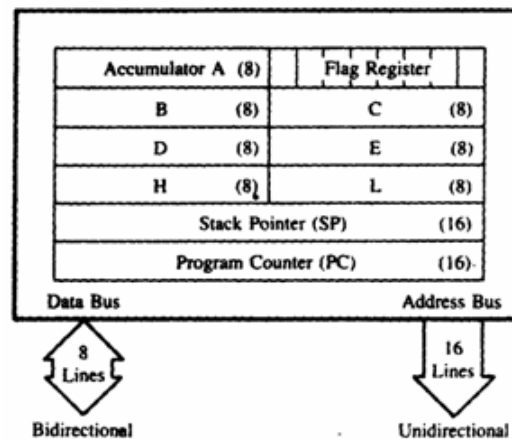
- The data bus is a group of eight lines used for data flow.
- These lines are bi-directional - data flow in both directions between the MPU and memory and peripheral devices.
- The MPU uses the data bus to perform the second function: transferring binary information.
- The eight data lines enable the MPU to manipulate 8-bit data ranging from 00 to FF (28 = 256 numbers).
- The largest number that can appear on the data bus is 11111111.

### Control Bus:

- The control bus carries synchronization signals and providing timing signals.
- The MPU generates specific control signals for every operation it performs. These signals are used to identify a device type with which the MPU wants to communicate.

### Registers of 8085:

- The 8085 have six general-purpose registers to store 8-bit data during program execution.
- These registers are identified as B, C, D, E, H, and L.
- They can be combined as register pairs-BC, DE, and HL-to perform some 16-bit operations.



### Accumulator (A):

- The accumulator is an 8-bit register that is part of the arithmetic/logic unit (ALU).
- This register is used to store 8-bit data and to perform arithmetic and logical operations.
- The result of an operation is stored in the accumulator.

### Flags:

- The ALU includes five flip-flops that are set or reset according to the result of an operation.
- The microprocessor uses the flags for testing the data conditions.
- They are Zero (Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags. The most commonly used flags are Sign, Zero, and Carry.



The bit position for the flags in flag register is,

D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>
S	Z		AC		P		CY

### 1. Sign Flag (S):

After execution of any arithmetic and logical operation, if D7 of the result is 1, the sign flag is set. Otherwise it is reset.

D7 is reserved for indicating the sign; the remaining is the magnitude of number.

If D7 is 1, the number will be viewed as negative number. If D7 is 0, the number will be viewed as positive number.

### 2. Zero Flag (z):

If the result of arithmetic and logical operation is zero, then zero flag is set otherwise it is reset.

### 3. Auxiliary Carry Flag (AC):

If D3 generates any carry when doing any arithmetic and logical operation, this flag is set. Otherwise it is reset.

### 4. Parity Flag (P):

If the result of arithmetic and logical operation contains even number of 1's then this flag will be set and if it is odd number of 1's it will be reset.

### 5. Carry Flag (CY):

If any arithmetic and logical operation result any carry then carry flag is set otherwise it is reset.

### Arithmetic and Logic Unit (ALU):

- It is used to perform the arithmetic operations like addition, subtraction, multiplication, division, increment and decrement and logical operations like AND, OR and EX-OR.
- It receives the data from accumulator and registers.
- According to the result it set or reset the flags.

### Program Counter (PC):

- This 16-bit register sequencing the execution of instructions.
- It is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register.
- The function of the program counter is to point to the memory address of the next instruction to be executed.
- When an opcode is being fetched, the program counter is incremented by one to point to the next memory location.

### Stack Pointer (SP):

- The stack pointer is also a 16-bit register used as a memory pointer.
- It points to a memory location in R/W memory, called the stack.
- The beginning of the stack is defined by loading a 16-bit address in the stack pointer (register).

**Temporary Register:** It is used to hold the data during the arithmetic and logical operations.

**Instruction Register:** When an instruction is fetched from the memory, it is loaded in the instruction register.

**Instruction Decoder:** It gets the instruction from the instruction register and decodes the instruction. It identifies the instruction to be performed.

**Serial I/O Control:** It has two control signals named SID and SOD for serial data transmission.

**Timing and Control unit:**

- It has three control signals ALE, RD (Active low) and WR (Active low) and three status signals IO/M(Active low), S0 and S1.
- ALE is used for provide control signal to synchronize the components of microprocessor and timing for instruction to perform the operation.
- RD (Active low) and WR (Active low) are used to indicate whether the operation is reading the data from memory or writing the data into memory respectively.
- IO/M(Active low) is used to indicate whether the operation is belongs to the memory or peripherals.
- If,

IO/M(Active Low)	S1	S2	Data Bus Status(Output)
0	0	0	Halt
0	0	1	Memory WRITE
0	1	0	Memory READ
1	0	1	IO WRITE
1	1	0	IO READ
0	1	1	Opcode fetch
1	1	1	Interrupt acknowledge

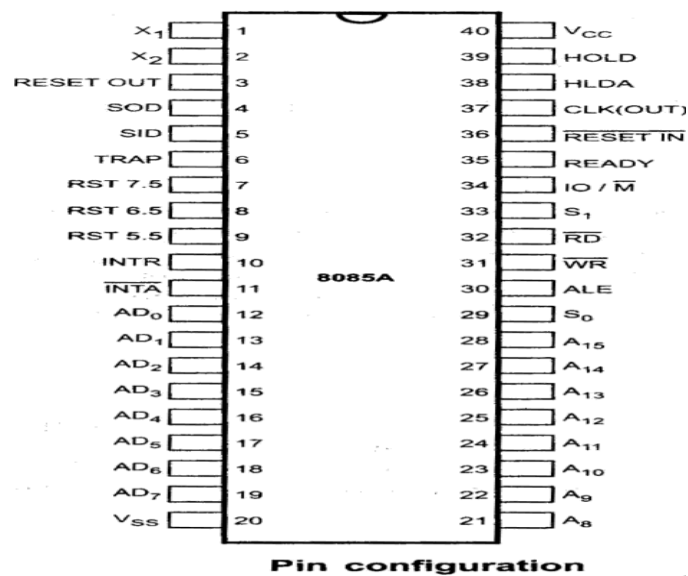
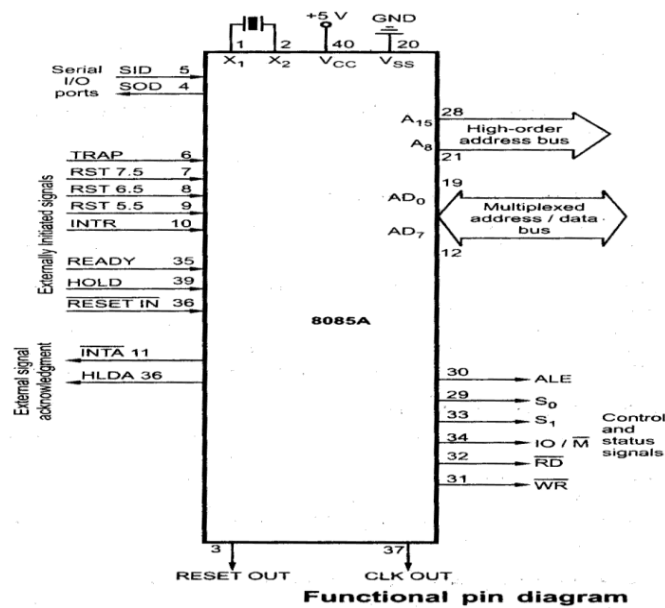
**Interrupt Control Unit:**

- It receives hardware interrupt signals and sends an acknowledgement for receiving the interrupt signal.

### Pin Diagram and Pin Description Of 8085

8085 is a 40 pin IC, DIP package. The signals from the pins can be grouped as follows

1. Power supply and clock signals
2. Address bus
3. Data bus
4. Control and status signals
5. Interrupts and externally initiated signals
6. Serial I/O ports



## 1. Power supply and clock frequency signals

- Vcc + 5 volt power supply
- Vss Ground
- X1, X2: Crystal or R/C network or LC network connections to set the frequency of internal clock generator.
- The frequency is internally divided by two. Since the basic operating timing frequency is 3 MHz, a 6 MHz crystal is connected externally.
- CLK (output)-Clock Output is used as the system clock for peripheral and devices interfaced with the microprocessor.

## **2. Address Bus:**

- A8 - A15 (output; 3-state)
- It carries the most significant 8 bits of the memory address or the 8 bits of the I/O address;

## **3. Multiplexed Address / Data Bus:**

- AD0 - AD7 (input/output; 3-state)
- These multiplexed set of lines used to carry the lower order 8 bit address as well as data bus.
- During the opcode fetch operation, in the first clock cycle, the lines deliver the lower order address A0 - A7.
- In the subsequent IO / memory, read / write clock cycle the lines are used as data bus.
- The CPU may read or write out data through these lines.

## **4. Control and Status signals:**

- ALE (output) - Address Latch Enable.
- This signal helps to capture the lower order address presented on the multiplexed address / data bus.
- RD (output 3-state, active low) - Read memory or IO device.
- This indicates that the selected memory location or I/O device is to be read and that the data bus is ready for accepting data from the memory or I/O device.
- WR (output 3-state, active low) - Write memory or IO device.
- This indicates that the data on the data bus is to be written into the selected memory location or I/O device.
- IO/M (output) - Select memory or an IO device.
- This status signal indicates that the read / write operation relates to whether the memory or I/O device.
- It goes high to indicate an I/O operation.
- It goes low for memory operations.

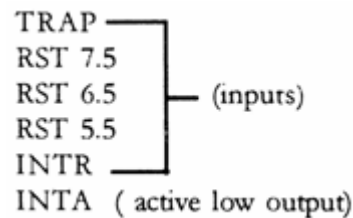
## **5. Status Signals:**

- It is used to know the type of current operation of the microprocessor.

IO/M (Active Low)	S1	S2	Data Bus Status (Output)
0	0	0	Halt
0	0	1	Memory WRITE
0	1	0	Memory READ
1	0	1	IO WRITE
1	1	0	IO READ
0	1	1	Opcode fetch
1	1	1	Interrupt acknowledge

## 6. Interrupts and externally initiated operations:

- They are the signals initiated by an external device to request the microprocessor to do a particular task or work.
- There are five hardware interrupts called,



- On receipt of an interrupt, the microprocessor acknowledges the interrupt by the active low INTA (Interrupt Acknowledge) signal.

### Reset In (input, active low)

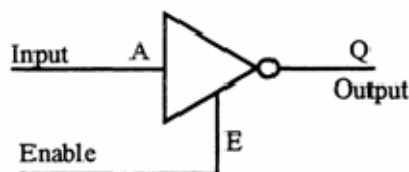
- This signal is used to reset the microprocessor.
- The program counter inside the microprocessor is set to zero.
- The buses are tri-stated.

### Reset Out (Output)

- It indicates CPU is being reset.
- Used to reset all the connected devices when the microprocessor is reset

## 7. Direct Memory Access (DMA):

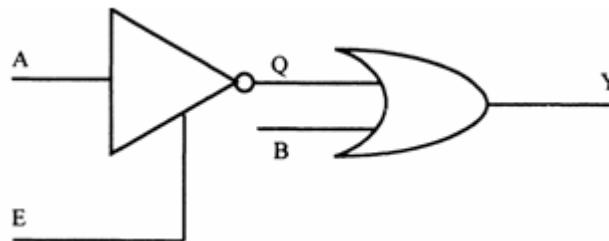
Tri state devices:



- 3 output states are high & low states and additionally a high impedance state.

- When enable E is high the gate is enabled and the output Q can be 1 or 0 (if A is 0, Q is 1, otherwise Q is 0). However, when E is low the gate is disabled and the output Q enters into a high impedance state.

E	A	Q	State
1(high)	0	1	High
1	1	0	Low
0(low)	0	0	High impedance
0	1	0	High impedance



- For both high and low states, the output Q draws a current from the input of the OR gate.
- When E is low, Q enters a high impedance state; high impedance means it is electrically isolated from the OR gate's input, though it is physically connected. Therefore, it does not draw any current from the OR gate's input.
- When 2 or more devices are connected to a common bus, to prevent the devices from interfering with each other, the tristate gates are used to disconnect all devices except the one that is communicating at a given instant.
- The CPU controls the data transfer operation between memory and I/O device. Direct Memory Access operation is used for large volume data transfer between memory and an I/O device directly.
- The CPU is disabled by tri-stating its buses and the transfer is effected directly by external control circuits.
- HOLD signal is generated by the DMA controller circuit. On receipt of this signal, the microprocessor acknowledges the request by sending out HLDA signal and leaves out the control of the buses. After the HLDA signal the DMA controller starts the direct transfer of data.

### READY (input)

- Memory and I/O devices will have slower response compared to microprocessors.
- Before completing the present job such a slow peripheral may not be able to handle further data or control signal from CPU.
- The processor sets the READY signal after completing the present job to access the data.
- The microprocessor enters into WAIT state while the READY pin is disabled.

## 8. Single Bit Serial I/O ports:

- SID (input) - Serial input data line
- SOD (output) - Serial output data line
- These signals are used for serial communication.

### Overview or Features of 8086

- It is a 16-bit Microprocessor ( $\mu p$ ). Its ALU, internal registers work with 16-bit binary word.
- 8086 has a 20-bit address bus can access up to  $2^{20} = 1$  MB memory locations.
- 8086 has a 16-bit data bus. It can read or write data to a memory/port either 16-bits or 8-bit at a time.
- It can support up to 64K I/O ports.
- It provides 14, 16-bit registers.
- Frequency range of 8086 is 6-10 MHz
- It has multiplexed address and data bus AD0- AD15 and A16 – A19.
- It requires single phase clock with 33% duty cycle to provide internal timing.
- It can prefetch up to 6 instruction bytes from memory and queues them in order to speed up instruction execution.
- It requires +5V power supply.
- A 40-pin dual in-line package.
- 8086 is designed to operate in two modes, Minimum mode and Maximum mode.
  - The minimum mode is selected by applying logic 1 to the MN / MX# input pin. This is a single microprocessor configuration.
  - The maximum mode is selected by applying logic 0 to the MN / MX# input pin. This is a multi-microprocessors configuration.

### Register Organization of 8086 General

#### purpose registers

The 8086 microprocessor has a total of fourteen registers that are accessible to the programmer. It is divided into four groups. They are:

- Four General purpose registers
- Four Index/Pointer registers
- Four Segment registers
- Two Other registers

General purpose registers:

General Purpose Registers		
		15 0
Accumulator	AX	Multiply, divide, I/O
Base	BX	Pointer to base address (data)
Count	CX	Count for loops, shifts
Data	DX	Multiply, divide, I/O

Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.

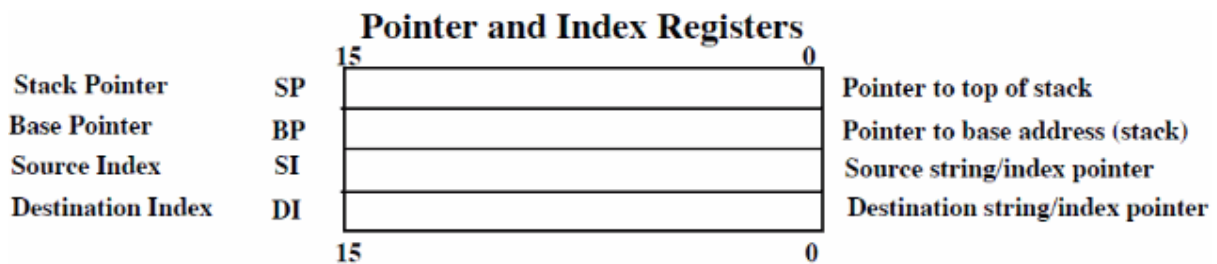
Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register indirect addressing.

Count register consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. When combined, CL register contains the low order byte of the word, and CH contains the high-order byte. Count register can be used in Loop, shift/rotate instructions and as a counter in string manipulation

Data register consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. When combined, DL register contains the low order byte of the word, and DH contains the high-order byte. Data register can be used as a port number in I/O operations. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

### Index or Pointer Registers

These registers can also be called as Special Purpose registers.



Stack Pointer (SP) is a 16-bit register pointing to program stack, i.e. it is used to hold the address of the top of stack. The stack is maintained as a LIFO with its bottom at the start of the stack segment (specified by the SS segment register). Unlike the SP register, the BP can be used to specify the offset of other program segments.

Base Pointer (BP) is a 16-bit register pointing to data in stack segment. It is usually used by subroutines to locate variables that were passed on the stack by a calling program. BP register is usually used for based, based indexed or register indirect addressing.

Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register indirect addressing, as well as a source data addresses in string manipulation instructions are used in conjunction with the DS register to point the data locations in the data segment.

Destination Index (DI) is a 16-bit register is used in conjunction with the ES register for string operations. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data addresses in string manipulation instructions. In short, Destination Index and SI Source Index registers are used to hold address.

### Segment Registers

Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers.



## Segment Registers

Code Segment	CS	
Data Segment	DS	
Stack Segment	SS	
Extra Segment	ES	

Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register. CS register cannot be changed directly. The CS register is automatically updated during far jump, far call and far return instructions.

Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions.

Extra segment (ES) used to hold the starting address of Extra segment. Extra segment is provided for programs that need to access a second data segment. Segment registers cannot be used in arithmetic operations.

### Other registers of 8086

## Other Registers

Flags	Flags
Instruction Pointer	IP

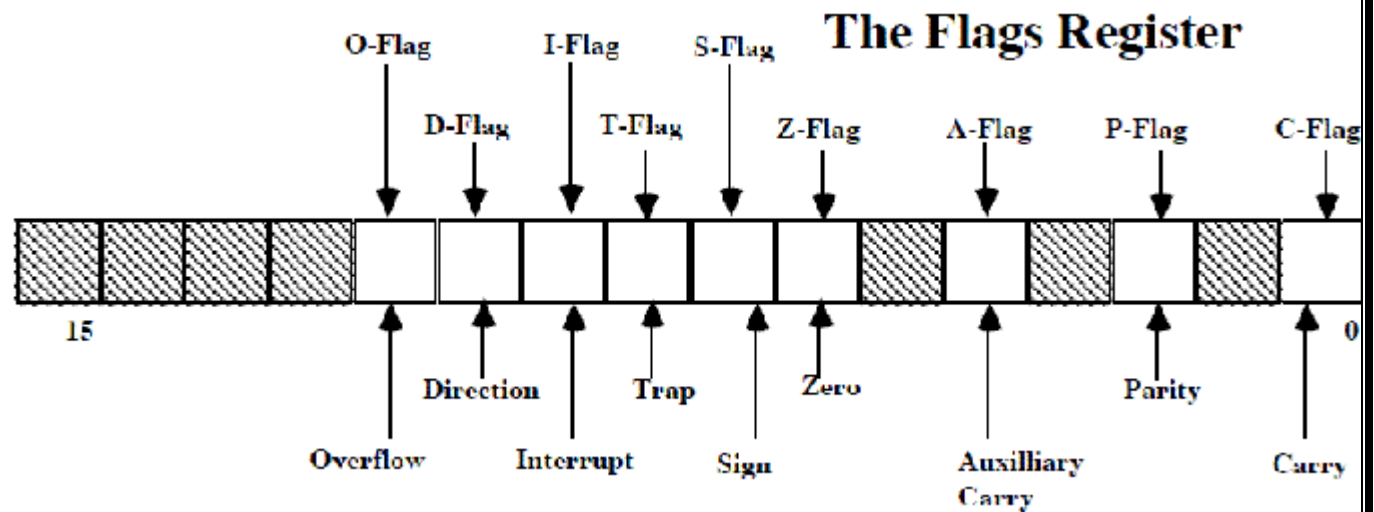
**Instruction Pointer (IP)** is a 16-bit register. This is a crucially important register which is used to control which instruction the CPU executes. The IP, or program counter, is used to store the memory location of the next instruction to be executed. The CPU checks the program counter to ascertain which instruction to carry out next. It then updates the program counter to point to the next instruction. Thus the program counter will always point to the next instruction to be executed.

**Flag Register** contains a group of status bits called flags that indicate the status of the CPU or the result of arithmetic operations. There are two types of flags:

1. The **status flags** which reflect the result of executing an instruction. The programmer cannot set/reset these flags directly.
2. The **control flags** enable or disable certain CPU operations. The programmer can set/reset these bits to control the CPU's operation.

Nine individual bits of the status register are used as control flags (3 of them) and status flags (6 of them). The remaining 7 are not used.

A flag can only take on the values 0 and 1. We say a flag is set if it has the value 1. The status flags are used to record specific characteristics of arithmetic and of logical instructions.



**Control Flags:** There are three control flags

1. **The Direction Flag (D):** Affects the direction of moving data blocks by such instructions as MOVS, CMPS and SCAS. The flag values are 0 = up and 1 = down and can be set/reset by the STD (set D) and CLD (clear D) instructions.

2. **The Interrupt Flag (I):** Dictates whether or not system interrupts can occur. Interrupts are actions initiated by hardware block such as input devices that will interrupt the normal execution of programs. The flag values are 0 = disable interrupts or 1 = enable interrupts and can be manipulated by the CLI (clear I) and STI (set I) instructions.

3. **The Trap Flag (T):** Determines whether or not the CPU is halted after the execution of each instruction. When this flag is set (i.e. = 1), the programmer can single step through his program to debug any errors. When this flag = 0 this feature is off. This flag can be set by the INT 3 instruction.

**Status Flags:** There are six status flags

1. **The Carry Flag (C):** This flag is set when the result of an unsigned arithmetic operation is too large to fit in the destination register. This happens when there is an end carry in an addition operation or there an end borrows in a subtraction operation. A value of 1 = carry and 0 = no carry.

2. **The Overflow Flag (O):** This flag is set when the result of a signed arithmetic operation is too large to fit in the destination register (i.e. when an overflow occurs). Overflow can occur when adding two numbers with the same sign (i.e. either positive or both negative). A value of 1 = overflow and 0 = no overflow.

3. **The Sign Flag (S):** This flag is set when the result of an arithmetic or logic operation is negative. This flag is a copy of the MSB of the result (i.e. the sign bit). A value of 1 means negative and 0 = positive.

4. **The Zero Flag (Z):** This flag is set when the result of an arithmetic or logic operation is equal to zero. A value of 1 means the result is zero and a value of 0 means the result is not zero.

5. **The Auxiliary Carry Flag (A):** This flag is set when an operation causes a carry from bit 3 to bit 4 (or a borrow from bit 4 to bit 3) of an operand. A value of 1 = carry and 0 = no carry.

6. **The Parity Flag (P):** This flag reflects the number of 1s in the result of an operation. If the number of 1s is even its value = 1 and if the number of 1s is odd then its value = 0.

### Architecture of 8086 or Functional Block diagram of 8086

- 8086 has two blocks Bus Interface Unit (BIU) and Execution Unit (EU).
- The BIU performs all bus operations such as instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The instruction bytes are transferred to the instruction queue.
- EU executes instructions from the instruction system byte queue.
- Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining. This results in efficient use of the system bus and system performance.
- BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder.
- EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.

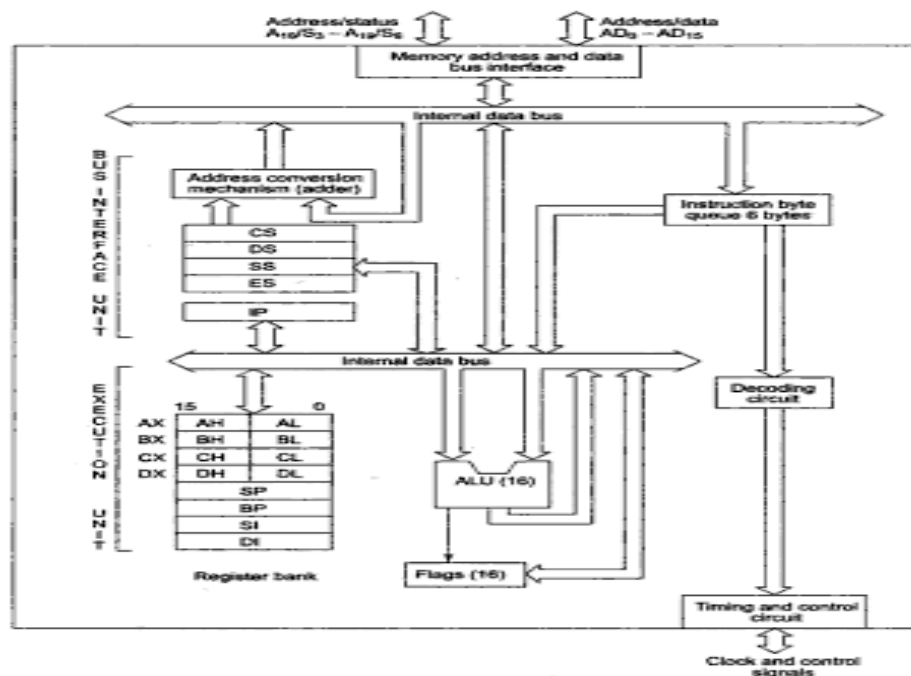


Figure: 8086 Architecture

## **Explanation of Architecture of 8086**

### **Bus Interface Unit:**

- It provides a full 16 bit bidirectional data bus and 20 bit address bus.
- The bus interface unit is responsible for performing all external bus operations.
- Specifically it has the following functions:
  - Instructions fetch Instruction queuing, Operand fetch and storage, Address relocation and Bus control.
  - The BIU uses a mechanism known as an instruction stream queue to implement pipeline architecture.
  - This queue permits prefetch of up to six bytes of instruction code. Whenever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by prefetching the next sequential instruction.
  - These prefetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle.
  - After a byte is loaded at the input end of the queue, it automatically shifts up through the FIFO to the empty location nearest the output.
  - The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory.
  - These intervals of no bus activity, which may occur between bus cycles, are known as idle state.
  - If the BIU is already in the process of fetching an instruction when the EU request it to read or write operands from memory or I/O, the BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle.
  - The BIU also contains a dedicated adder which is used to generate the 20bit physical address that is output on the address bus. This address is formed by adding an appended 16 bit segment address and a 16 bit offset address.
  - For example: The physical address of the next instruction to be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register.
  - The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write.

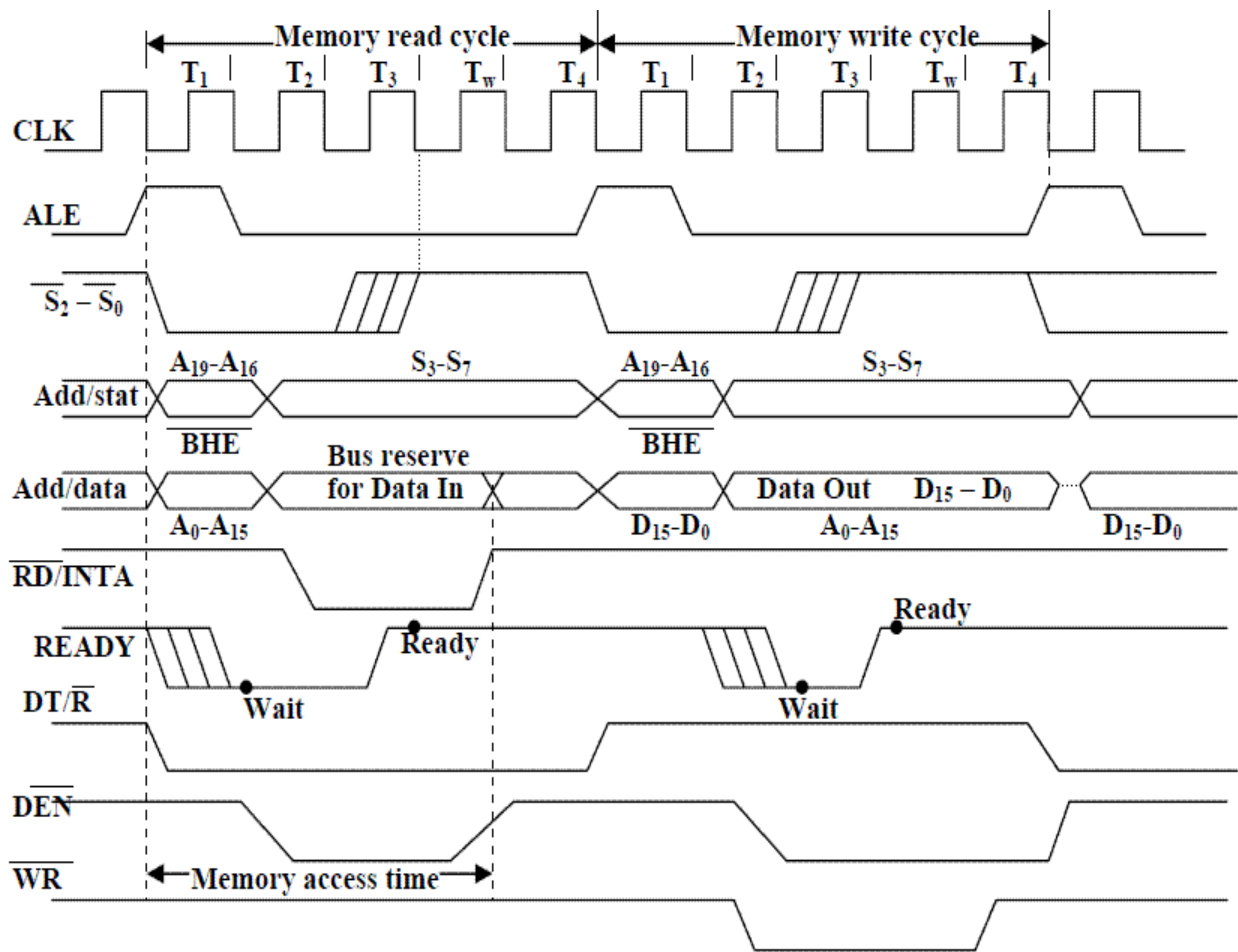
### **Execution Unit**

- The Execution unit is responsible for decoding and executing all instructions.

- The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operands if necessary, passes them to the BIU and requests it to perform the read or write bus cycles to memory or I/O and perform the operation specified by the instruction on the operands.
- During the execution of the instruction, the EU tests the status and control flags and updates them based on the results of executing the instruction.
- If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to top of the queue.
- When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions.
- Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.

### **General Bus Operation**

- The 8086 has a combined address and data bus commonly referred as a time multiplexed address and data bus.
- The main reason behind multiplexing address and data over the same pins is the maximum utilization of processor pins and it facilitates the use of 40 pin standard DIP package.
- The bus can be demultiplexed using a few latches and transceivers, when ever required.
- Basically, all the processor bus cycles consist of at least four clock cycles. These are referred to as T1, T2, T3, and T4. The address is transmitted by the processor during T1. It is present on the bus only for one cycle.
- The negative edge of this ALE pulse is used to separate the address and the data or status information. In maximum mode, the status lines S0, S1 and S2 are used to indicate the type of operation.
- Status bits S3 to S7 are multiplexed with higher order address bits and the BHE signal. Address is valid during T1 while status bits S3 to S7 are valid during T2 through T4.



### Maximum mode

- In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
- In this mode, the processor derives the status signal  $S_2$ ,  $S_1$ ,  $S_0$ . Another chip called bus controller derives the control signal using this status information.
- In the maximum mode, there may be more than one microprocessor in the system configuration.

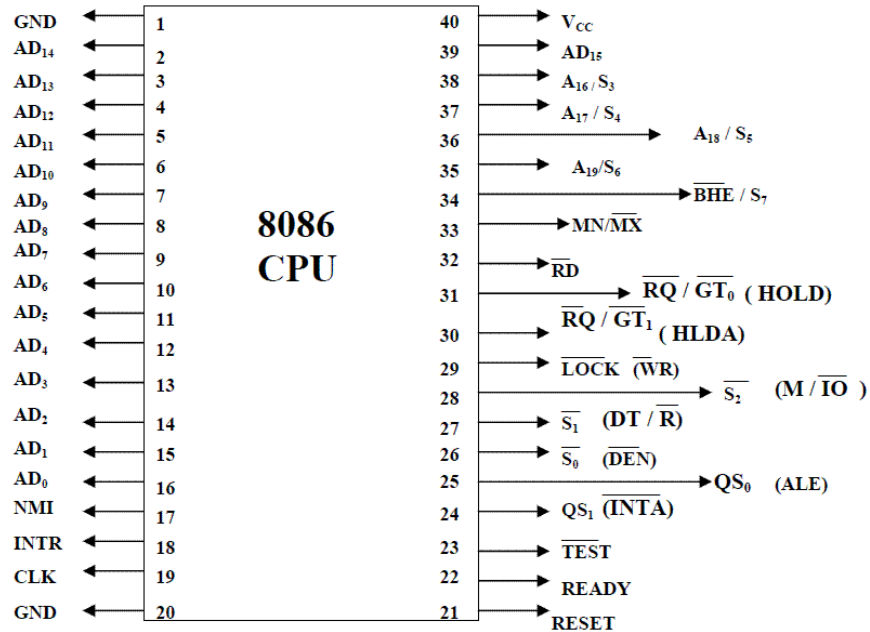
### Minimum mode

- In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.
- In this mode, all the control signals are given out by the microprocessor chip itself.
- There is a single microprocessor in the minimum mode system.

## Pin Diagram of 8086 and Pin description of 8086

Figure shows the Pin diagram of 8086. The description follows it.

**Pin Diagram of 8086**



- The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin Cerdip or plastic package.
- The 8086 operates in single processor or multiprocessor configuration to achieve high performance. The pins serve a particular function in minimum mode (single processor mode) and other function in maximum mode configuration (multiprocessor mode).
- The 8086 signals can be categorized in three groups.
  1. The first are the signal having common functions in minimum as well as maximum mode.
  2. The second are the signals which have special functions for minimum mode
  3. The third are the signals having special functions for maximum mode.
- The following signal descriptions are common for both modes.
- **AD15-AD0:** These are the time multiplexed memory I/O address and data lines.
  1. Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, Tw and T4. These lines are active high and float to a tristate during interrupt acknowledge and local bus hold acknowledge cycles.

- **A19/S6, A18/S5, A17/S4, and A16/S3:** These are the time multiplexed address and status lines.
  2. During T1 these are the most significant address lines for memory operations.
  3. During I/O operations, these lines are low.
  4. During memory or I/O operations, status information is available on those lines for T2, T3, Tw and T4.
  5. The status of the interrupt enable flag bit is updated at the beginning of each clock cycle.
  6. The S4 and S3 combine indicate which segment registers is presently being used for memory accesses as in below fig
  7. These lines float to tri-state off during the local bus hold acknowledge. The status line S6 is always low.
  8. The address bit is separated from the status bit using latches controlled by the ALE signal.

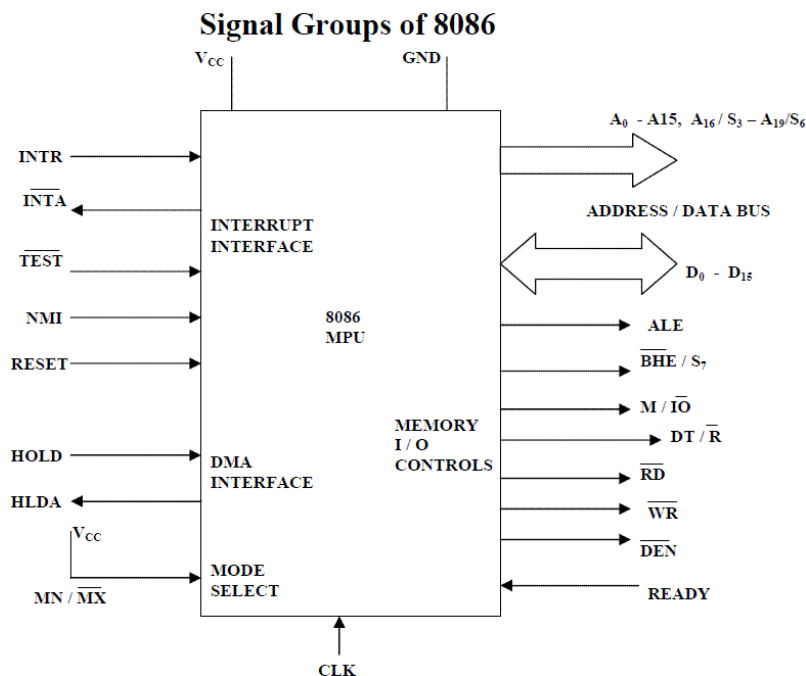
S4	S3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code or None
1	1	Data
0	0	Whole word
0	1	Upper byte from or to even address
1	0	Lower byte from or to even address

- **BHE/S7:** The bus high enable is used to indicate the transfer of data over the higher order (D15-D8) data bus as shown in table. It goes low for the data transfer over D15-D8 and is used to derive chip selects of odd address memory bank or peripherals. BHE is low during T1 for read, write and interrupt acknowledge cycles, whenever a byte is to be transferred on higher byte of data bus. The status information is available during T2, T3 and T4. The signal is active low and tristated during hold. It is low during T1 for the first pulse of the interrupt acknowledge cycle.
- **RD – Read:** This signal on low indicates the peripheral that the processor is performing memory or I/O read operation. RD is active low and shows the state for T2, T3, Tw of any read cycle. The signal remains tristated during the hold acknowledge.



- **READY:** This is the acknowledgement from the slow device or memory that they have completed the data transfer. The signal made available by the devices is synchronized by the 8284A clock generator to provide ready input to the 8086. the signal is active high.
- **INTR-Interrupt Request:** This is a triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the interrupt acknowledge cycle. This can be internally masked by resulting the interrupt enable flag. This signal is active high and internally synchronized.
- **TEST:** This input is examined by a 'WAIT' instruction. If the TEST pin goes low, execution will continue, else the processor remains in an idle state. The input is synchronized internally during each clock cycle on leading edge of clock.
- **CLK- Clock Input:** The clock input provides the basic timing for processor operation and bus control activity. It's an asymmetric square wave with 33% duty cycle.

Figure shows the Pin functions of 8086.



*The following pin functions are for the minimum mode operation of 8086.*

- **M/I0 – Memory/I0:** This is a status line logically equivalent to S2 in maximum mode. When it is low, it indicates the CPU is having an I/O operation, and when it is high, it indicates that the CPU is having a memory operation. This line becomes active high in the previous T4 and remains active till final T4 of the current cycle. It is tristated during local bus “hold acknowledge “.
- **INTA – Interrupt Acknowledge:** This signal is used as a read strobe for interrupt

acknowledge cycles. i.e. when it goes low, the processor has accepted the interrupt.

- **ALE – Address Latch Enable:** This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches. This signal is active high and is never tristated.
- **DT/R – Data Transmit/Receive:** This output is used to decide the direction of data flow through the transceivers (bidirectional buffers). When the processor sends out data, this signal is high and when the processor is receiving data, this signal is low.
- **DEN – Data Enable:** This signal indicates the availability of valid data over the address/data lines. It is used to enable the transceivers (bidirectional buffers) to separate the data from the multiplexed address/data signal. It is active from the middle of T2 until the middle of T4. This is tristated during ‘hold acknowledge’ cycle.
- **HOLD, HLDA- Acknowledge:** When the HOLD line goes high; it indicates to the processor that another master is requesting the bus access. The processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin, in the middle of the next clock cycle after completing the current bus cycle.
- At the same time, the processor floats the local bus and control lines. When the processor detects the HOLD line low, it lowers the HLDA signal. HOLD is an asynchronous input, and is should be externally synchronized. If the DMA request is made while the CPU is performing a memory or I/O cycle, it will release the local bus during T4 provided :
  1. The request occurs on or before T2 state of the current cycle.
  2. The current cycle is not operating over the lower byte of a word.
  3. The current cycle is not the first acknowledge of an interrupt acknowledge sequence.
  4. A Lock instruction is not being executed.

*The following pin functions are applicable for maximum mode operation of 8086.*

- **S2, S1, and S0 – Status Lines:** These are the status lines which reflect the type of operation, being carried out by the processor. These become activity during T4 of the previous cycle and active during T1 and T2 of the current bus cycles.
- **LOCK:** This output pin indicates that other system bus master will be prevented from gaining the system bus, while the LOCK signal is low. The LOCK signal is activated by the ‘LOCK’ prefix instruction and remains active until the completion of the next instruction. When the CPU is executing a critical instruction which requires the system bus, the LOCK prefix instruction ensures that other processors connected in the system will not gain the control of the bus.

The 8086, while executing the prefixed instruction, asserts the bus lock signal output, which may be connected to an external bus controller. By prefetching the instruction, there is a considerable speeding up in instruction execution in 8086. This is known as **instruction pipelining**.

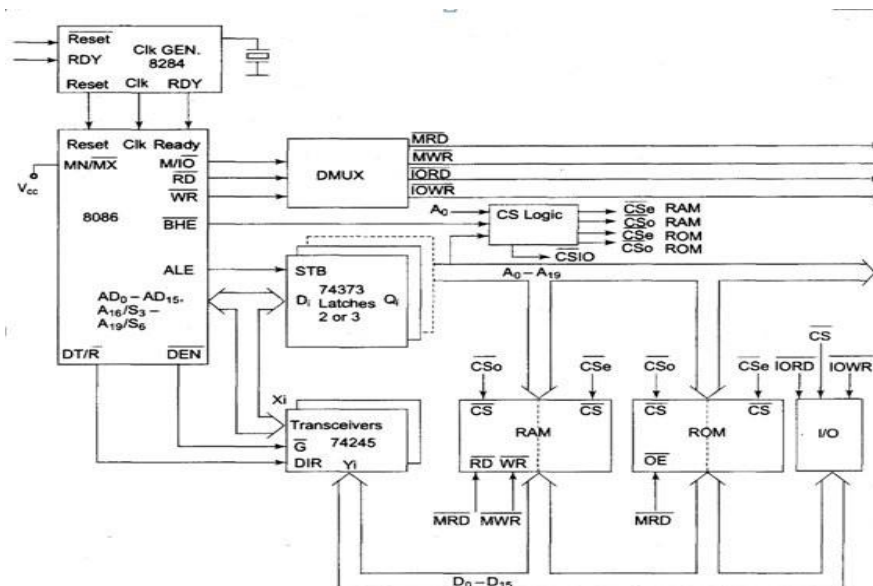
S2	S1	S0	Indication
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

- At the starting the CS: IP is loaded with the required address from which the execution is to be started. Initially, the queue will be empty and the microprocessor starts a fetch operation to bring one byte (the first byte) of instruction code, if the CS: IP address is odd or two bytes at a time, if the CS: IP address is even.
- The first byte is a complete opcode in case of some instruction (one byte opcode instruction) and is a part of opcode, in case of some instructions (two byte opcode instructions), the remaining part of code lie in second byte.
- The second byte is then decoded in continuation with the first byte to decide the instruction length and the number of subsequent bytes to be treated as instruction data. The queue is updated after every byte is read from the queue but the fetch cycle is initiated by BIU only if at least two bytes of the queue are empty and the EU may be concurrently executing the fetched instructions.
- The next byte after the instruction is completed is again the first opcode byte of the next instruction. A similar procedure is repeated till the complete execution of the program. The fetch operation of the next instruction is overlapped with the execution of the current instruction. As in the architecture, there are two separate units, namely Execution unit and Bus interface unit.
- While the execution unit is busy in executing an instruction, after it is completely decoded, the bus interface unit may be fetching the bytes of the next instruction from memory, depending upon the queue status.

QS1	QSo	Indication
0	0	No Operation
0	1	First Byte of the opcode from the queue
1	0	Empty Queue
1	1	Subsequent Byte from the Queue

- **RQ/GT0, RQ/GT1 – Request/Grant:** These pins are used by the other local bus master in maximum mode, to force the processor to release the local bus at the end of the processor current bus cycle.
- Each of the pin is bidirectional with RQ/GT0 having higher priority than RQ/GT1. RQ/GT pins have internal pull-up resistors and may be left unconnected. Request/Grant sequence is as follows:
  1. A pulse of one clock wide from another bus master requests the bus access to 8086.
  2. During T4(current) or T1(next) clock cycle, a pulse one clock wide from 8086 to the requesting master, indicates that the 8086 has allowed the local bus to float and that it will enter the 'hold acknowledge' state at next cycle. The CPU bus interface unit is likely to be disconnected from the local bus of the system.
  3. A one clock wide pulse from another master indicates to the 8086 that the hold request is about to end and the 8086 may regain control of the local bus at the next clock cycle. Thus each master to master exchange of the local bus is a sequence of 3 pulses. There must be at least one dead clock cycle after each bus exchange. The request and grant pulses are active low. For the bus request those are received while 8086 is performing memory or I/O cycle, the granting of the bus is governed by the rules as in case of HOLD and HLDA in minimum mode.

### Minimum Mode 8086 System



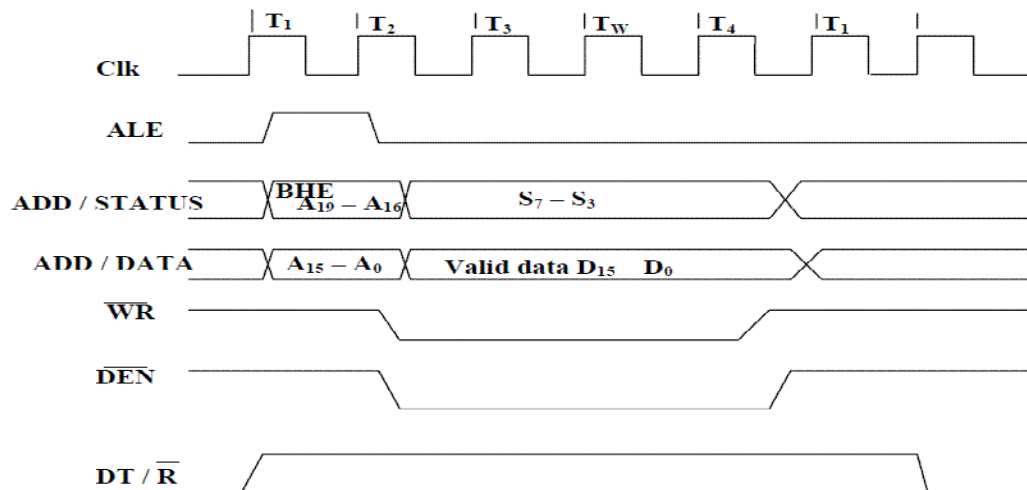
### Minimum mode 8086 system

- In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.

- In this mode, all the control signals are given out by the microprocessor chip itself. There is a single microprocessor in the minimum mode system.
- The remaining components in the system are latches, transceivers, clock generator, memory and I/O devices. Some type of chip selection logic may be required for selecting memory or I/O devices, depending upon the address map of the system.
- Latches are generally buffered output D-type flip-flops like 74LS373 or 8282. They are used for separating the valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.
- Transceivers are the bidirectional buffers and some times they are called as data amplifiers. They are required to separate the valid data from the time multiplexed address/data signals.
- They are controlled by two signals namely, DEN and DT/R.
- The DEN signal indicates the direction of data, i.e. from or to the processor. The system contains memory for the monitor and users program storage.
- Usually, EPROM is used for monitor storage, while RAM for users program storage. A system may contain I/O devices.

### Write Cycle Timing Diagram for Minimum Mode

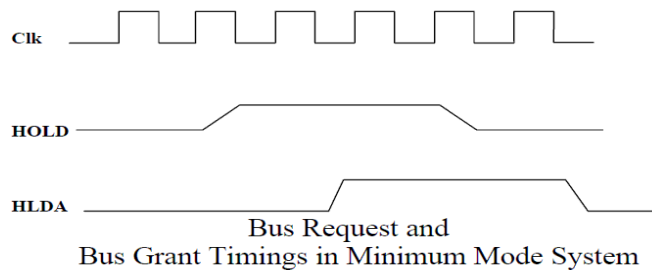
- The working of the minimum mode configuration system can be better described in terms of the timing diagrams rather than qualitatively describing the operations.
- The opcode fetch and read cycles are similar. Hence the timing diagram can be categorized in two parts, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.



Write Cycle Timing Diagram for Minimum Mode

- The read cycle begins in T1 with the assertion of address latch enable (ALE) signal and also M / IO signal. During the negative going edge of this signal, the valid address is latched on the local bus.
- The BHE and A0 signals address low, high or both bytes. From T1 to T4 , the M/IO signal indicates a memory or I/O operation.
- At T2, the address is removed from the local bus and is sent to the output. The bus is then tristated. The read (RD) control signal is also activated in T2.
- The read (RD) signal causes the address device to enable its data bus drivers. After RD goes low, the valid data is available on the data bus.
- The addressed device will drive the READY line high. When the processor returns the read signal to high level, the addressed device will again tristate its bus drivers.
- A write cycle also begins with the assertion of ALE and the emission of the address. The M/IO signal is again asserted to indicate a memory or I/O operation. In T2, after sending the address in T1, the processor sends the data to be written to the addressed location.
- The data remains on the bus until middle of T4 state. The WR becomes active at the beginning of T2 (unlike RD is somewhat delayed in T2 to provide time for floating).
- The BHE and A0 signals are used to select the proper byte or bytes of memory or I/O word to be read or write.
- The M/IO, RD and WR signals indicate the type of data transfer as specified in table below.

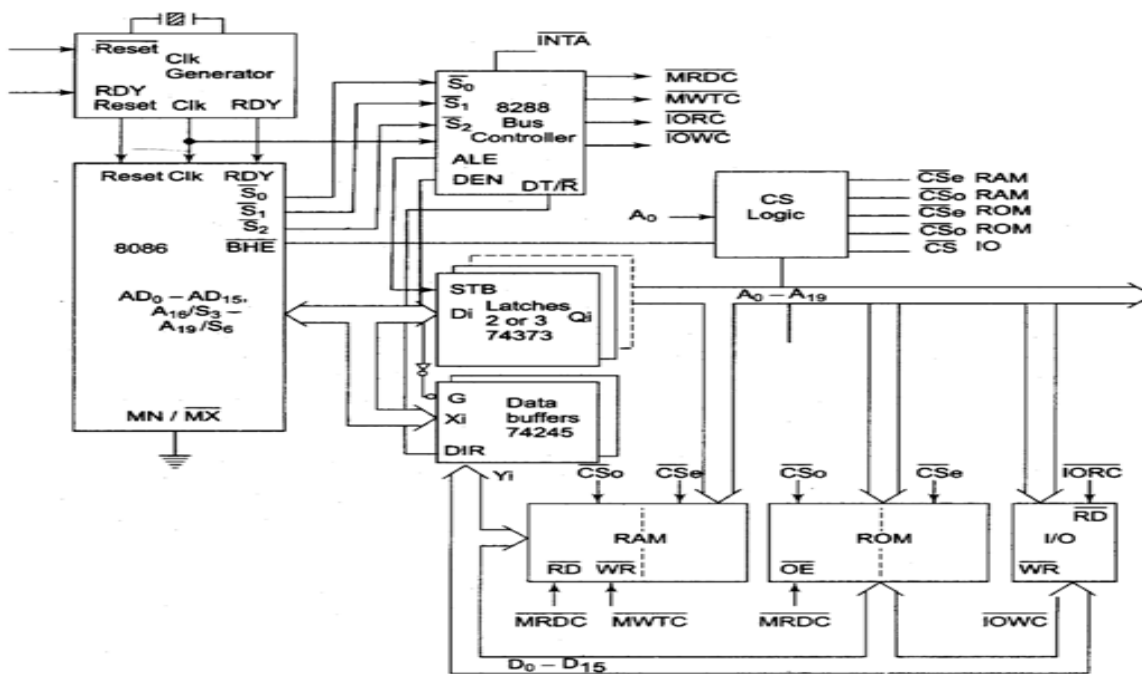
### Bus Request and Bus Grant Timings in Minimum Mode System of 8086



- Hold Response sequence: The HOLD pin is checked at leading edge of each clock pulse. If it is received active by the processor before T4 of the previous cycle or during T1 state of the current cycle, the CPU activates HLDA in the next clock cycle and for succeeding bus cycles, the bus will be given to another requesting master.
- The control of the bus is not regained by the processor until the requesting master does not drop the HOLD pin low. When the request is dropped by the requesting master, the HLDA is dropped by the processor at the trailing edge of the next clock.

## Maximum Mode 8086 System

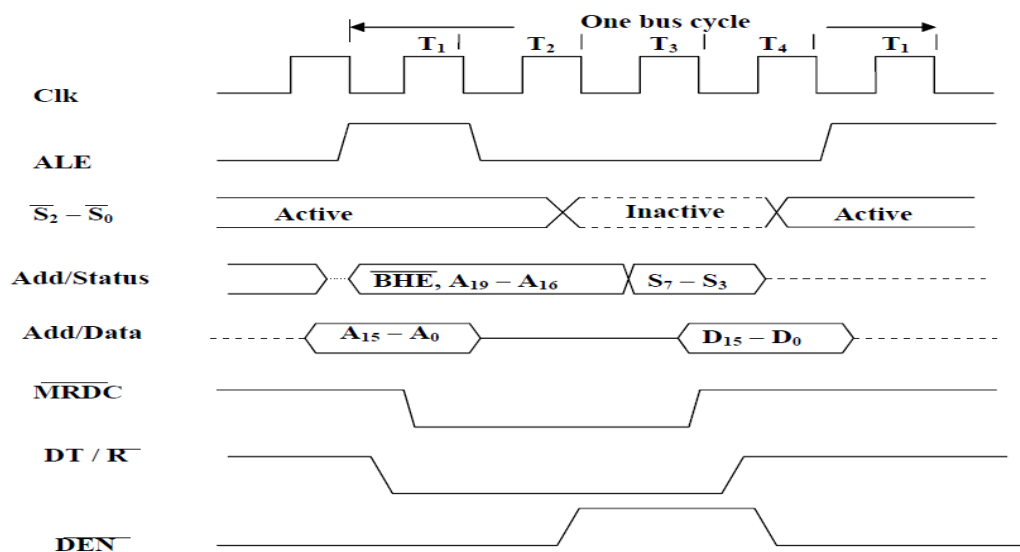
- In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.
- In this mode, the processor derives the status signal S2, S1, S0. Another chip called bus controller derives the control signal using this status information.
- In the maximum mode, there may be more than one microprocessor in the system configuration.
- The components in the system are same as in the minimum mode system.
- The basic function of the bus controller chip IC8288 is to derive control signals like RD and WR (for memory and I/O devices), DEN, DT/R, ALE etc. using the information by the processor on the status lines.
- The bus controller chip has input lines S2, S1, S0 and CLK. These inputs to 8288 are driven by CPU.



- It derives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC. The AEN, IOB and CEN pins are especially useful for multiprocessor systems.
- AEN and IOB are generally grounded. CEN pin is usually tied to +5V. The significance of the MCE/PDEN output depends upon the status of the IOB pin.
- If IOB is grounded, it acts as master cascade enable to control cascade 8259A, else it acts as peripheral data enable used in the multiple bus configurations.
- INTA pin used to issue two interrupt acknowledge pulses to the interrupt controller or to an interrupting device.

- IORC, IOWC are I/O read command and I/O write command signals respectively.
- These signals enable an IO interface to read or write the data from or to the address port.
- The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read or write signals.
- All these command signals instructs the memory to accept or send data from or to the bus.
- For both of these write command signals, the advanced signals namely AIOWC and AMWTC are available.
- Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.
- R0, S1, S2 are set at the beginning of bus cycle. 8288 bus controller will output a pulse as on the ALE and apply a required signal to its DT / R pin during T1.
- In T2, 8288 will set DEN=1 thus enabling transceivers, and for an input it will activate MRDC or IORC. These signals are activated until T4. For an output, the AMWC or AIOWC is activated from T2 to T4 and MWTC or IOWC is activated from T3 to T4.
- The status bit S0 to S2 remains active until T3 and become passive during T3 and T4.
- If reader input is not activated before T3, wait state will be inserted between T3 and T4.

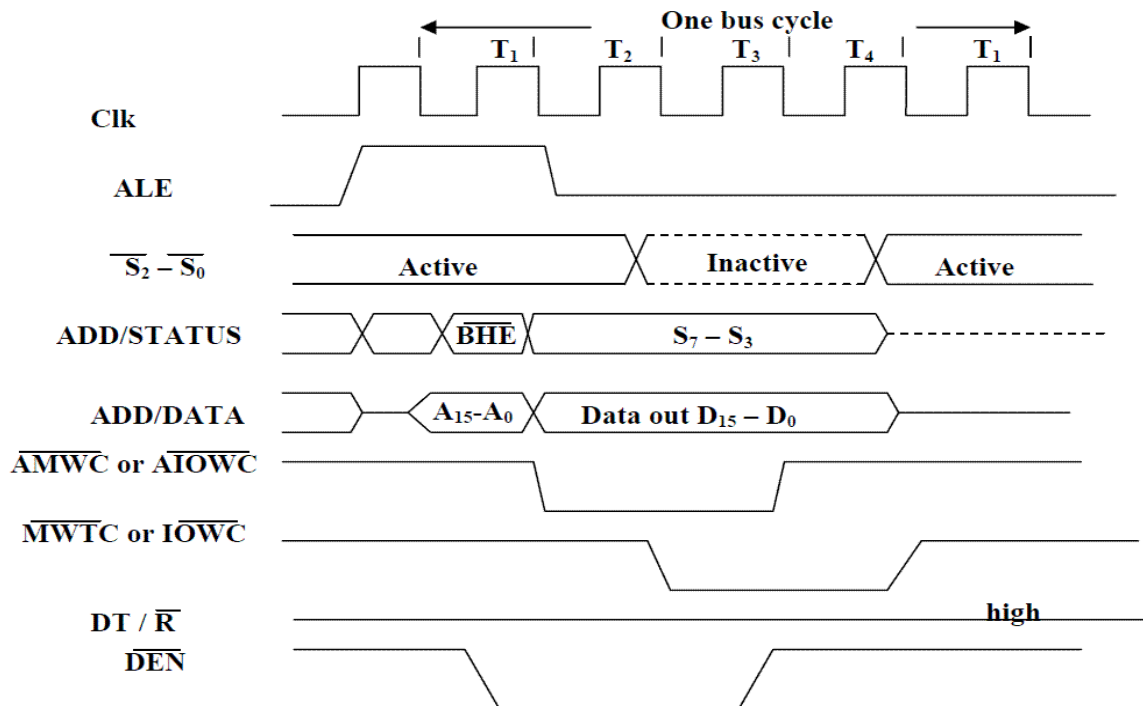
#### Memory Read Timing Diagram in Maximum Mode of 8086



Memory Read Timing in Maximum Mode

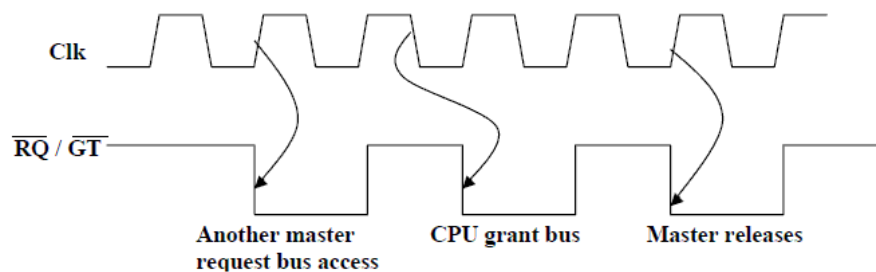


## Memory Write Timing in Maximum mode of 8086



Memory Write Timing in Maximum mode.

## RQ/GT Timings in Maximum Mode



$\overline{RQ}/\overline{GT}$  Timings in Maximum Mode

- The request/grant response sequence contains a series of three pulses. The request/grant pins are checked at each rising pulse of clock input.
- When a request is detected and if the condition for HOLD request is satisfied, the processor issues a grant pulse over the RQ/GT pin immediately during T4 (current) or T1 (next) state.
- When the requesting master receives this pulse, it accepts the control of the bus, it sends a release pulse to the processor using RQ/GT pin.

## **Minimum Mode Interface**

- When the Minimum mode operation is selected, the 8086 provides all control signals needed to implement the memory and I/O interface.
- The minimum mode signal can be divided into the following basic groups :

### **1. Address/data bus**

### **2. Status**

### **3. Control**

### **4. Interrupt and**

### **5. DMA.**

Each and every group is explained clearly.

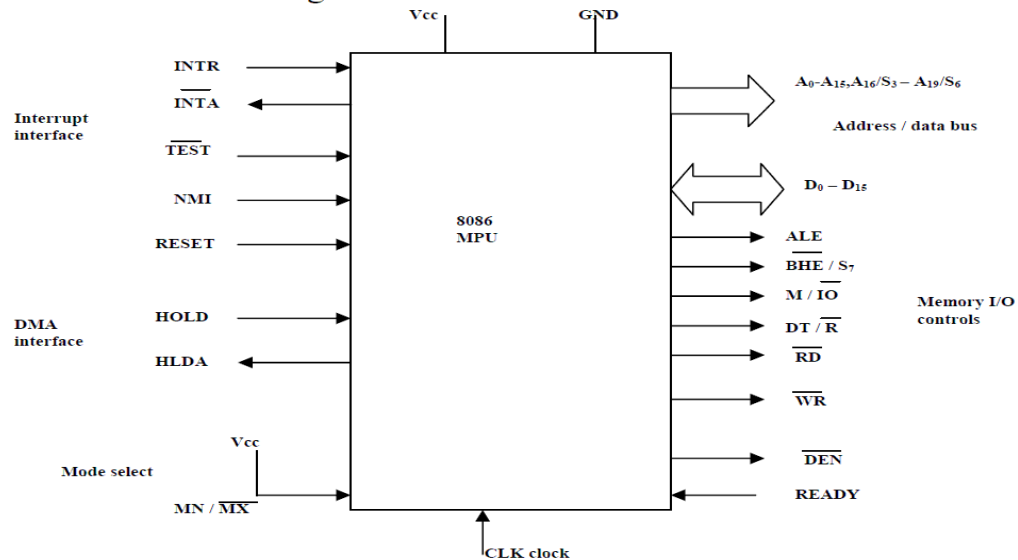
## **Address/Data Bus:**

- These lines serve two functions. As an address bus is 20 bits long and consists of signal lines A0 through A19. A19 represents the MSB and A0 LSB. A 20bit address gives the 8086 a 1Mbyte memory address space. More over it has an independent I/O address space which is 64K bytes in length.
- The 16 data bus lines D0 through D15 are actually multiplexed with address lines A0 through A15 respectively. By multiplexed we mean that the bus work as an address bus during first machine cycle and as a data bus during next machine cycles.
- D15 is the MSB and D0 LSB. When acting as a data bus, they carry read/write data for memory, input/output data for I/O devices, and interrupt type codes from an interrupt controller.

## **Status signal:**

- The four most significant address lines A19 through A16 are also multiplexed but in this case with status signals S6 through S3.
- These status bits are output on the bus at the same time that data are transferred over the other bus lines.

## Block Diagram of the Minimum Mode 8086 MPU



- Bit S<sub>4</sub> and S<sub>3</sub> together form a 2 bit binary code that identifies which of the 8086 internal segment registers is used to generate the physical address that was output on the address bus during the current bus cycle. Code S<sub>4</sub>S<sub>3</sub> = 00 identifies a register known as extra segment register as the source of the segment address.
- Status line S<sub>5</sub> reflects the status of another internal characteristic of the 8086. It is the logic level of the internal enable flag. The last status bit S<sub>6</sub> is always at the logic 0 level.

S <sub>4</sub>	S <sub>3</sub>	Segment Register
0	0	Extra
0	1	Stack
1	0	Code / none
1	1	Data

Memory segment status codes

### Control Signals:

- The control signals are provided to support the 8086 memory I/O interfaces. They control functions such as when the bus is to carry a valid address in which direction data are to be transferred over the bus, when valid write data are on the bus and when to put read data on the system bus.
- ALE is a pulse to logic 1 that signals external circuitry when a valid address word is on the bus. This address must be latched in external circuitry on the 1-to-0 edge of the pulse at ALE.
- Another control signal that is produced during the bus cycle is BHE bank high enable. Logic 0 on this used as a memory enable signal for the most significant byte half of the data bus D<sub>8</sub> through D<sub>15</sub>. These lines also serve a second function, which is as the S<sub>7</sub> status line.

- Using the M/I/O and DT/R lines, the 8086 signals which type of bus cycle is in progress and in which direction data are to be transferred over the bus. The logic level of M/I/O tells external circuitry whether a memory or I/O transfer is taking place over the bus. Logic 1 at this output signals a memory operation and logic 0 an I/O operation.
- The direction of data transfer over the bus is signaled by the logic level output at DT/R. When this line is logic 1 during the data transfer part of a bus cycle, the bus is in the transmit mode. Therefore, data are either written into memory or output to an I/O device. On the other hand, logic 0 at DT/R signals that the bus is in the receive mode. This corresponds to reading data from memory or input of data from an input port.
- The signals read RD and write WR indicates that a read bus cycle or a write bus cycle is in progress. The 8086 switches WR to logic 0 to signal external device that valid write or output data are on the bus.
- On the other hand, RD indicates that the 8086 is performing a read of data of the bus. During read operations, one other control signal is also supplied. This is DEN (data enable) and it signals external devices when they should put data on the bus. There is one other control signal that is involved with the memory and I/O interface. This is the READY signal.
- READY signal is used to insert wait states into the bus cycle such that it is extended by a number of clock periods. This signal is provided by an external clock generator device and can be supplied by the memory or I/O sub-system to signal the 8086 when they are ready to permit the data transfer to be completed.

#### **Interrupt signals:**

- The key interrupt interface signals are interrupt request (INTR) and interrupt acknowledge (INTA).
- INTR is an input to the 8086 that can be used by an external device to signal that it need to be serviced.
- Logic 1 at INTR represents an active interrupt request. When an interrupt request has been recognized by the 8086, it indicates this fact to external circuit with pulse to logic 0 at the INTA output.
- The TEST input is also related to the external interrupt interface. Execution of a WAIT instruction causes the 8086 to check the logic level at the TEST input.
- If the logic 1 is found, the MPU suspend operation and goes into the idle state. The 8086 no longer executes instructions; instead it repeatedly checks the logic level of the TEST input waiting for its transition back to logic 0.
- As TEST switches to 0, execution resume with the next instruction in the program. This feature can be used to synchronize the operation of the 8086 to an event in external hardware.
- There are two more inputs in the interrupt interface: the nonmaskable interrupt NMI and the reset interrupt RESET.
- On the 0-to-1 transition of NMI control is passed to a nonmaskable interrupt

service routine. The RESET input is used to provide a hardware reset for the 8086. Switching RESET to logic 0 initializes the internal register of the 8086 and initiates a reset service routine.

#### **DMA Interface signals:**

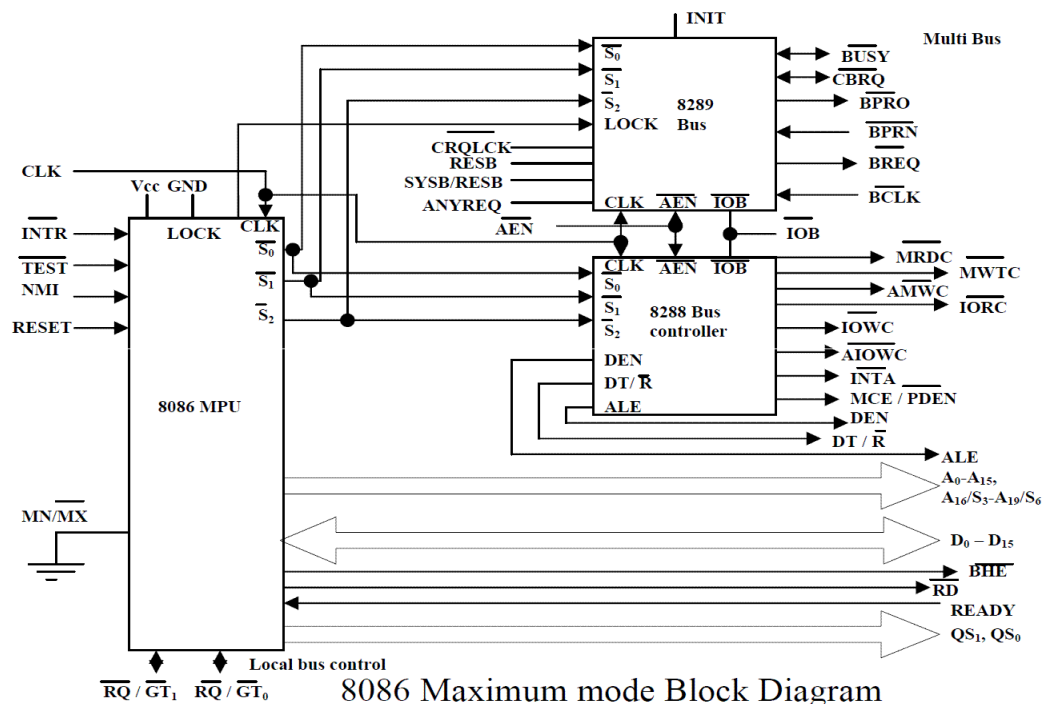
- The direct memory access DMA interface of the 8086 minimum mode consist of the HOLD and HLDA signals.
- When an external device wants to take control of the system bus, it signals to the 8086 by switching HOLD to the logic 1 level. At the completion of the current bus cycle, the 8086 enters the hold state. In the hold state, signal lines AD0 through AD15, A16/S3 through A19/S6, BHE, M/IO, DT/R, RD, WR, DEN and INTR are all in the high Z state.
- The 8086 signals external device that it is in this state by switching its HLDA output to logic 1 level.

#### **Maximum Mode Interface**

- When the 8086 is set for the maximum-mode configuration, it provides signals for implementing a multiprocessor / coprocessor system environment.
- By multiprocessor environment we mean that one microprocessor exists in the system and that each processor is executing its own program.
- Usually in this type of system environment, there are some system resources that are common to all processors. They are called as global resources. There are also other resources that are assigned to specific processors. These are known as local or private resources.
- Coprocessor also means that there is a second processor in the system. In these two processors does not access the bus at the same time. One passes the control of the system bus to the other and then may suspend its operation.
- In the maximum-mode 8086 system, facilities are provided for implementing allocation of global resources and passing bus control to other microprocessor or coprocessor.

#### **8288 Bus Controller – Bus Command and Control Signals:**

- 8086 does not directly provide all the signals that are required to control the memory, I/O and interrupt interfaces.



- Specially the WR, M/IO, DT/R, DEN, ALE and INTA, signals are no longer produced by the 8086. Instead it outputs three status signals S0, S1, S2 prior to the initiation of each bus cycle. This 3-bit bus status code identifies which type of bus cycle is to follow.
- S2S1S0 are input to the external bus controller device, the bus controller generates the appropriately timed command and control signals.
- The 8288 produces one or two of these eight command signals for each bus cycles. For instance, when the 8086 outputs the code S2S1S0 equals 001; it indicates that an I/O read cycle is to be performed.

S2	S1	S0	Indication	8288 Command
0	0	0	Interrupt Acknowledge	INTA
0	0	1	Read I/O port	IORC
0	1	0	Write I/O port	IOWC, AIOWC
0	1	1	Halt	None
1	0	0	Instruction Fetch	MRDC
1	0	1	Read Memory	MRDC
1	1	0	Write Memory	MWTC, AMWC
1	1	1	Passive	None

- In the code 111 is output by the 8086, it is signaling that no bus activity is to take place.

- The control outputs produced by the 8288 are DEN, DT/R and ALE. These 3 signals provide the same functions as those described for the minimum system mode.
- This set of bus commands and control signals is compatible with the Multibus and industry standard for interfacing microprocessor systems.
- The output of 8289 are bus arbitration signals:

Bus busy (BUSY), common bus request (CBRQ), bus priority out (BPRO), bus priority in (BPRN), bus request (BREX) and bus clock (BCLK).

- They correspond to the bus exchange signals of the Multibus and are used to lock other processor off the system bus during the execution of an instruction by the 8086.
- In this way the processor can be assured of uninterrupted access to common system resources such as global memory.
- Queue Status Signals: Two new signals that are produced by the 8086 in the maximum- mode system are queue status outputs QS0 and QS1. Together they form a 2-bit queue status code, QS1QS0.
- Following table shows the four different queue status.
- Local Bus Control Signal – Request / Grant Signals: In a maximum mode configuration, the minimum mode HOLD, HLDA interface is also changed

QS1	QS0	Queue Status
0 (Low)	0	Queue Empty. The queue has been reinitiated as a result of the execution of a transfer instruction.
0	1	First Byte. The byte taken from the queue was the first byte of the instruction.
1	0	Queue Empty. The queue has been reinitiated as a result of the execution of a transfer instruction.
1	1 (High)	Subsequent Byte. The byte taken from the queue was the subsequent byte of the instruction.

- . These two are replaced by request/grant lines RQ/ GT0 and RQ/ GT1, respectively. They provide a prioritized bus access mechanism for accessing the local bus.

## Interrupts

**Definition:** The meaning of ‘interrupts’ is to break the sequence of operation. While the CPU is executing a program, on ‘interrupt’ breaks the normal sequence of execution of instructions, diverts its execution to some other program called Interrupt Service Routine (ISR). After executing ISR, the control is transferred back again to the main program. Interrupt processing is an alternative to polling.

**Need for Interrupt:** Interrupts are particularly useful when interfacing I/O devices that provide or require data at relatively low data transfer rate.

**Types of Interrupts:** There are two types of Interrupts in 8086. They are: (i) Hardware Interrupts and  
(ii) Software Interrupts

(i) **Hardware Interrupts** (External Interrupts). The Intel microprocessors support hardware interrupts through:

- Two pins that allow interrupt requests, INTR and NMI
- One pin that acknowledges, INTA, the interrupt requested on INTR. INTR and NMI
- INTR is a maskable hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction.
- When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location  $4 * \text{<interrupt type>}$ . Interrupt processing routine should return with the IRET instruction.
- NMI is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority than the maskable interrupt.
- – Ex: NMI, INTR.

(ii) **Software Interrupts** (Internal Interrupts and Instructions) .Software interrupts can be caused by:

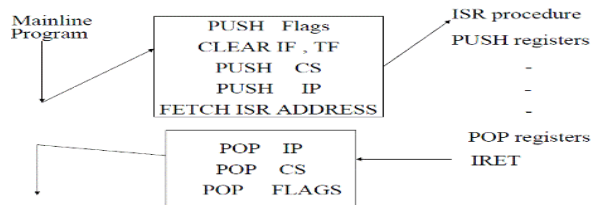
- INT instruction - breakpoint interrupt. This is a type 3 interrupt.
- INT <interrupt number> instruction - any one interrupt from available 256 interrupts.
- INTO instruction - interrupt on overflow
- Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.
- Processor exceptions: Divide Error (Type 0), Unused Opcode (type 6) and Escape opcode (type 7).
- Software interrupt processing is the same as for the hardware interrupts.
- - Ex: INT n (Software Instructions)
- Control is provided through:



i. IF and TF flag bits

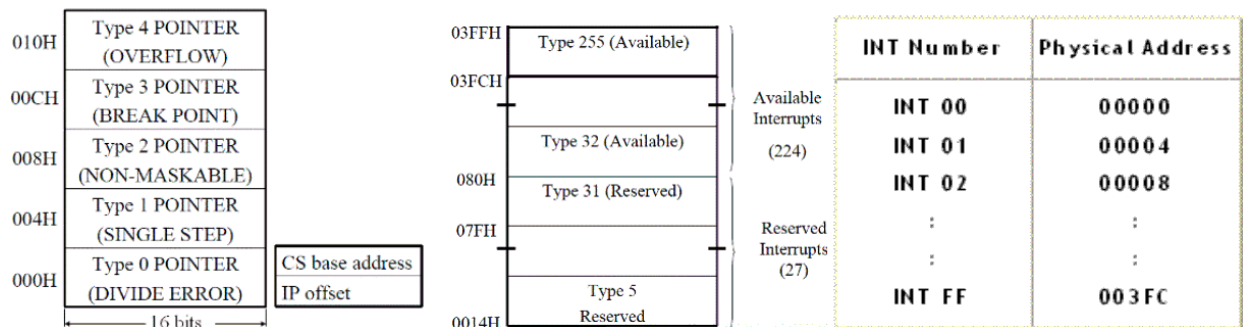
ii. IRET and IRETD

### Performance of Software Interrupts



1. It decrements SP by 2 and pushes the flag register on the stack.
2. Disables INTR by clearing the IF.
3. It resets the TF in the flag Register.
4. It decrements SP by 2 and pushes CS on the stack.
5. It decrements SP by 2 and pushes IP on the stack.
6. Fetch the ISR address from the interrupt vector table.

### Interrupt Vector Table



### Functions associated with INT00 to INT04

#### INT 00 (divide error)

- INT00 is invoked by the microprocessor whenever there is an attempt to divide a number by zero.
- ISR is responsible for displaying the message “Divide Error” on the screen

#### INT 01

- For single stepping the trap flag must be 1
- After execution of each instruction, 8086 automatically jumps to 00004H to fetch 4 bytes for CS: IP of the ISR.
- The job of ISR is to dump the registers on to the screen

### INT 02 (Non maskable Interrupt)

- Whenever NMI pin of the 8086 is activated by a high signal (5v), the CPU Jumps to physical memory location 00008 to fetch CS: IP of the ISR associated with NMI.

### INT 03 (break point)

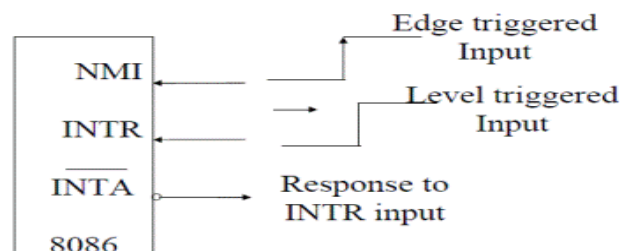
- A break point is used to examine the CPU and memory after the execution of a group of Instructions.
- It is one byte instruction whereas other instructions of the form “INT nn” are 2 byte instructions.

### INT 04 (Signed number overflow)

- There is an instruction associated with this INT 0 (interrupt on overflow).
- If INT 0 is placed after a signed number arithmetic as IMUL or ADD the CPU will activate INT 04 if OF = 1.
- In case where OF = 0, the INT 0 is not executed but is bypassed and acts as a NOP.

### Performance of Hardware Interrupts

- NMI : Non maskable interrupts - TYPE 2 Interrupt
- INTR : Interrupt request - Between 20H and FFH



### Interrupt Priority Structure

Interrupt	Priority
Divide Error, INT(n),INTO	Highest
NMI	↓
INTR	
Single Step	
	Lowest

## UNIT – II

### PIN DIAGRAM OF 8086 AND ASSEMBLY LANGUAGE PROGRAMMING

#### Addressing Modes of 8086:

Addressing mode indicates a way of locating data or operands. Depending up on the data type used in the instruction and the memory addressing modes, any instruction may belong to one or more addressing modes or same instruction may not belong to any of the addressing modes.

The addressing mode describes the types of operands and the way they are accessed for executing an instruction. According to the flow of instruction execution, the instructions may be categorized as

1. Sequential control flow instructions and
2. Control transfer instructions.

Sequential control flow instructions are the instructions which after execution, transfer control to the next instruction appearing immediately after it (in the sequence) in the program. For example the arithmetic, logic, data transfer and processor control instructions are Sequential control flow instructions.

The control transfer instructions on the other hand transfer control to some predefined address or the address somehow specified in the instruction, after their execution. For example INT, CALL, RET & JUMP instructions fall under this category.

The addressing modes for Sequential and control flow instructions are explained as follows.

#### 1. Immediate addressing mode:

In this type of addressing, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

**Example:** MOV AX, 0005H.

In the above example, 0005H is the immediate data. The immediate data may be 8-bit or 16-bit in size.

#### 2. Direct addressing mode:

In the direct addressing mode, a 16-bit memory address (offset) directly

specified in the instruction as a part of it.

**Example:** MOV AX, [5000H].

### **3. Register addressing mode:**

In the register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers, except IP, may be used in this mode.

**Example:** MOV BX, AX

### **4. Register indirect addressing mode:**

Sometimes, the address of the memory location which contains data or operands is determined in an indirect way, using the offset registers. The mode of addressing is known as register indirect mode.

In this addressing mode, the offset address of data is in either BX or SI or DI Register. The default segment is either DS or ES.

**Example:** MOV AX, [BX].

### **5. Indexed addressing mode:**

In this addressing mode, offset of the operand is stored one of the index registers. DS & ES are the default segments for index registers SI & DI respectively.

**Example:** MOV AX, [SI]

Here, data is available at an offset address stored in SI in DS.

### **6. Register relative addressing mode:**

In this addressing mode, the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the register BX, BP, SI & DI in the default (either in DS & ES) segment.

**Example:** MOV AX, 50H [BX]

### **7. Based indexed addressing mode:**

The effective address of data is formed in this addressing mode, by adding content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

**Example:** MOV AX, [BX][SI]

## **8. Relative based indexed:**

The effective address is formed by adding an 8 or 16-bit displacement with the sum of contents of any of the base registers (BX or BP) and any one of the index registers, in a default segment.

**Example:** MOV AX, 50H [BX] [SI]

For the control transfer instructions, the addressing modes depend upon whether the destination location is within the same segment or in a different one. It also depends upon the method of passing the destination address to the processor. Basically, there are two addressing modes for the control transfer instructions, viz. intersegment and intrasegment addressing modes.

If the location to which the control is to be transferred lies in a different segment other than the current one, the mode is called intersegment mode. If the destination location lies in the same segment, the mode is called intrasegment mode.

### **Addressing Modes for control transfer instructions:**

#### **1. Intersegment**

- Intersegment direct
- Intersegment indirect

#### **2. Intrasegment**

- Intrasegment direct
- Intrasegment indirect

#### **1. Intersegment direct:**

In this mode, the address to which the control is to be transferred is in a different segment. This addressing mode provides a means of branching from one code segment to another code segment. Here, the CS and IP of the destination address are specified directly in the instruction.

**Example:** JMP 5000H, 2000H;

Jump to effective address 2000H in segment 5000H.

## **2. Intersegment indirect:**

In this mode, the address to which the control is to be transferred lies in a different segment and it is passed to the instruction indirectly, i.e. contents of a memory block containing four bytes, i.e. IP(LSB), IP(MSB), CS(LSB) and CS(MSB) sequentially. The starting address of the memory block may be referred using any of the addressing modes, except immediate mode.

**Example:** JMP [2000H].

Jump to an address in the other segment specified at effective address 2000H in DS.

## **3. Intrasegment direct mode:**

In this mode, the address to which the control is to be transferred lies in the same segment in which the control transfers instruction lies and appears directly in the instruction as an immediate displacement value. In this addressing mode, the displacement is computed relative to the content of the instruction pointer.

The effective address to which the control will be transferred is given by the sum of 8 or 16 bit displacement and current content of IP. In case of jump instruction, if the signed displacement (d) is of 8-bits (i.e.  $-128 < d < +127$ ), it is a short jump and if it is of 16 bits (i.e.  $-32768 < d < +32767$ ), it is termed as long jump.

**Example:** JMP SHORT LABEL.

## **4. Intrasegment indirect mode:**

In this mode, the displacement to which the control is to be transferred is in the same segment in which the control transfer instruction lies, but it is passed to the instruction directly. Here, the branch address is found as the content of a register or a memory location.

This addressing mode may be used in unconditional branch instructions.

**Example:** JMP [BX]; Jump to effective address stored in BX.

## **INSTRUCTION SET OF 8086**

The Instruction set of 8086 microprocessor is classified into 7, they are:-

- Data transfer instructions
- Arithmetic & logical instructions
- Program control transfer instructions
- Machine Control Instructions
- Shift / rotate instructions
- Flag manipulation instructions
- String instructions

### **Data Transfer instructions**

Data transfer instruction, as the name suggests is for the transfer of data from memory to internal register, from internal register to memory, from one register to another register, from input port to internal register, from internal register to output port etc

#### **1. MOV instruction**

It is a general purpose instruction to transfer byte or word from register to register, memory to register, register to memory or with immediate addressing.

##### **General Form:**

MOV destination, source

Here the source and destination needs to be of the same size, that is both 8 bit or both 16 bit.

MOV instruction does not affect any flags.

##### **Example:-**

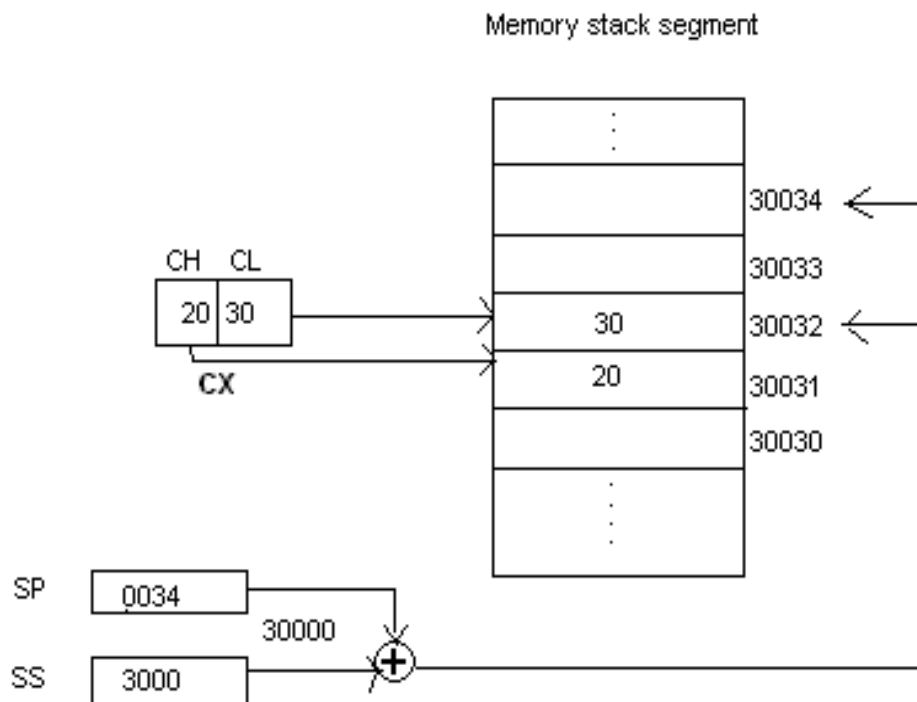
MOV BX, 00F2H	;	load the immediate number 00F2H in BX Register
MOV CL, [2000H]	;	Copy the 8 bit content of the memory Location, at a displacement of 2000H from data segment base to the CL register
MOV [589H], BX	;	Copy the 16 bit content of BX register on to the memory location, which at a displacement of 589H from the data segment base.
MOV DS, CX	;	Move the content of CX to DS

## 2. PUSH instruction

The PUSH instruction decrements the stack pointer by two and copies the word from source to the location where stack pointer now points. Here the source must be of word size data. Source can be a general purpose register, segment register or a memory location.

The PUSH instruction first pushes the most significant byte to sp-1, then the least significant to the sp-2.

Push instruction does not affect any flags.



### Example:-

PUSH CX ; Decrements SP by 2, copy content of CX to the stack (figure shows execution of this instruction)

PUSH DS ; Decrement SP by 2 and copy DS to stack

## 3. POP instruction

The POP instruction copies a word from the stack location pointed by the stack pointer to the destination. The destination can be a General purpose register, a segment register or a memory location. Here after the content is copied the stack pointer is automatically incremented by two.

The execution pattern is similar to that of the PUSH instruction.



**Example:**

POP CX ; Copy a word from the top of the stack to CX and increment SP by 2.

**4. IN & OUT instructions**

The IN instruction will copy data from a port to the accumulator. If 8 bit is read the data will go to AL and if 16 bit then to AX. Similarly OUT instruction is used to copy data from accumulator to an output port.

Both IN and OUT instructions can be done using direct and indirect addressing modes.

**Example:**

IN AL, 0F8H	;	Copy a byte from the port 0F8H to AL
MOV DX, 30F8H	;	Copy port address in DX
IN AL, DX	;	Move 8 bit data from 30F8H port
IN AX, DX	;	Move 16 bit data from 30F8H port
OUT 047H, AL	;	Copy contents of AL to 8 bit port 047H
MOV DX, 30F8H	;	Copy port address in DX
OUT DX, AL	;	Move 8 bit data to the 30F8H port
OUT DX, AX	;	Move 16 bit data to the 30F8H port

**5. XCHG instruction**

The XCHG instruction exchanges contents of the destination and source. Here destination and source can be register and register or register and memory location, but XCHG cannot interchange the value of 2 memory locations.

**General Format**

XCHG Destination, Source

**Example:**

XCHG BX, CX	;	exchange word in CX with the word in BX
XCHG AL, CL	;	exchange byte in CL with the byte in AL
XCHG AX, SUM[BX]	;	Here physical address, which is DS+SUM+[BX]. The content at physical address and the content of AX are interchanged.

## Arithmetic and Logic instructions

The arithmetic and logic logical group of instruction include,

### 1. ADD instruction

Add instruction is used to add the current contents of destination with that of source and store the result in destination. Here we can use register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected

#### General Format:

ADD Destination, Source

#### Example:

- ADD AL, 0FH ; Add the immediate content, 0FH to the content of AL and store the result in AL
- ADD AX, BX ;       $AX \leftarrow AX + BX$
- ADD AX, 0100H – IMMEDIATE
- ADD AX, BX – REGISTER
- ADD AX, [SI] – REGISTER INDIRECT OR INDEXED
- ADD AX, [5000H] – DIRECT
- ADD [5000H], 0100H – IMMEDIATE
- ADD 0100H – DESTINATION AX (IMPLICIT)

### 2. ADC: ADD WITH CARRY

This instruction performs the same operation as ADD instruction, but adds the carry flag bit (which may be set as a result of the previous calculation) to the result. All the condition code flags are affected by this instruction. The examples of this instruction along with the modes are as follows:

#### Example:

- ADC AX, BX – REGISTER
- ADC AX, [SI] – REGISTER INDIRECT OR INDEXED
- ADC AX, [5000H] – DIRECT
- ADC [5000H], 0100H – IMMEDIATE
- ADC 0100H – IMMEDIATE (AX IMPLICIT)

### 3. SUB instruction

SUB instruction is used to subtract the current contents of destination with that of source and store the result in destination. Here we can use

register and/or memory locations. AF, CF, OF, PF, SF, and ZF flags are affected

### **General Format:**

SUB Destination, Source

#### **Example:**

- SUB AL, 0FH ; subtract the immediate content, 0FH from the content of AL and store the result in AL
- SUB AX, BX ; AX <= AX-BX
- SUB AX, 0100H – IMMEDIATE (DESTINATION AX)
- SUB AX, BX – REGISTER
- SUB AX, [5000H] – DIRECT
- SUB [5000H], 0100H – IMMEDIATE

## **4. SBB: SUBTRACT WITH BORROW**

The subtract with borrow instruction subtracts the source operand and the borrow flag (CF) which may reflect the result of the previous calculations, from the destination operand. Subtraction with borrow, here means subtracting 1 from the subtraction obtained by SUB, if carry (borrow) flag is set.

The result is stored in the destination operand. All the flags are affected (condition code) by this instruction. The examples of this instruction are as follows:

#### **Example:**

- SBB AX, 0100H – IMMEDIATE (DESTINATION AX)
- SBB AX, BX – REGISTER
- SBB AX, [5000H] – DIRECT
- SBB [5000H], 0100H – IMMEDIATE

## **5. CMP: COMPARE**

The instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location. For comparison, it subtracts the source operand from the destination operand but does not store the result anywhere. The flags are affected depending upon the result of the subtraction. If both of the operands are equal, zero flag is set. If the source operand is greater than the destination operand, carry flag is set or else, carry flag is reset. The examples of this instruction are as follows:

**Example:**

- CMP BX,0100H – IMMEDIATE
- CMP AX,0100H – IMMEDIATE
- CMP [5000H], 0100H – DIRECT
- CMP BX,[SI] – REGISTER INDIRECT OR INDEXED
- CMP BX, CX – REGISTER

**6. INC & DEC instructions**

INC and DEC instructions are used to increment and decrement the content of the specified destination by one. AF, CF, OF, PF, SF, and ZF flags are affected.

**Example:**

INC AL	;	AL<= AL + 1
INC AX	;	AX<=AX + 1
DEC AL	;	AL<= AL – 1
DEC AX	;	AX<=AX – 1

**7. AND instruction**

This instruction logically ANDs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.

The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.

**General Format:**

AND Destination, Source

**Example:**

- AND BL, AL ; suppose BL=1000 0110 and AL = 1100 1010 then after the operation BL would be BL= 1000 0010.
- AND CX, AX ; CX <= CX AND AX
- AND CL, 08 ; CL<= CL AND (0000 1000)

**8. OR instruction**

This instruction logically ORs each bit of the source byte/word with the corresponding bit in the destination and stores the result in destination. The source can be an immediate number, register or memory location, register can be a register or memory location.

The CF and OF flags are both made zero, PF, ZF, SF are affected by the operation and AF is undefined.

### **General Format:**

OR Destination, Source

### **Example:**

- OR BL, AL ; suppose BL=1000 0110 and AL = 1100 1010 then after the operation BL would be BL= 1100 1110.
- OR CX, AX ; CX <= CX AND AX
- OR CL, 08 ; CL<= CL AND (0000 1000)

## **9. NOT instruction**

The NOT instruction complements (inverts) the contents of an operand register or a memory location, bit by bit. The examples are as follows:

### **Example:**

- NOT AX (BEFORE AX= (1011)<sub>2</sub>= (B)<sub>16</sub> AFTER EXECUTION AX= (0100)<sub>2</sub>= (4)<sub>16</sub>).
- NOT [5000H]

## **10. XOR instruction**

The XOR operation is again carried out in a similar way to the AND and OR operation. The constraints on the operands are also similar. The XOR operation gives a high output, when the 2 input bits are dissimilar. Otherwise, the output is zero. The example instructions are as follows:

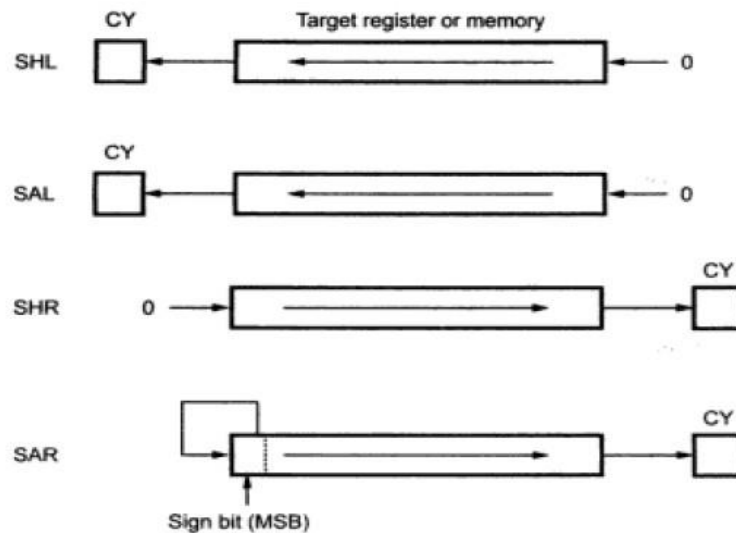
### **Example:**

- XOR AX,0098H
- XOR AX,BX
- XOR AX,[5000H]

## **Shift / Rotate Instructions**

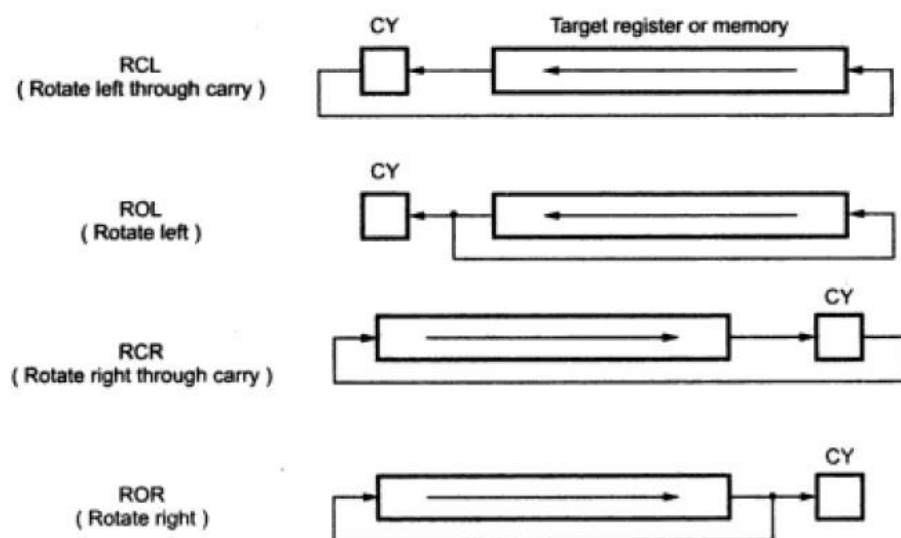
Shift instructions move the binary data to the left or right by shifting them within the register or memory location. They also can perform multiplication of powers of  $2^{+n}$  and division of powers of  $2^{-n}$ .

There are two type of shifts logical shifting and arithmetic shifting, later is used with signed numbers while former with unsigned.



**Fig.1 Shift operations**

Rotate on the other hand rotates the information in a register or memory either from one end to another or through the carry flag.



**Fig.2 Rotate operations**

### **SHL/SAL instruction**

Both the instruction shifts each bit to left, and places the MSB in CF and LSB is made 0. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected.

### **General Format:**

SAL/SHL destination, count

**Example:**

MOV BL, B7H ; BL is made B7H

SAL BL, 1 ; shift the content of BL register one place to left.

Before execution,

CY	B7	B6	B5	B4	B3	B2	B1	B0
0	1	0	1	1	0	1	1	1

After the execution,

CY	B7	B6	B5	B4	B3	B2	B1	B0
0	1	1	0	1	1	1	0	

**1. SHR instruction**

This instruction shifts each bit in the specified destination to the right and 0 is stored in the MSB position. The LSB is shifted into the carry flag. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected

**General Format: SHR**

destination, count

**Example:**

MOV BL, B7H ; BL is made B7H

SHR BL, 1 ; shift the content of BL register one place to the right.

Before execution,

B7	B6	B5	B4	B3	B2	B1	B0	CY
1	0	1	1	0	1	1	1	0

After execution,

B7	B6	B5	B4	B3	B2	B1	B0	CY
0	1	0	1	1	0	1	1	1

**2. ROL instruction**

This instruction rotates all the bits in a specified byte or word to the left some number of bit positions. MSB is placed as a new LSB and a new CF. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected

**General Format: ROL**

destination, count

**Example:**

MOV BL, B7H ; BL is made B7H

ROL BL, 1 ; rotates the content of BL register one place to the left.

Before execution,

CY	B7	B6	B5	B4	B3	B2	B1	B0
0	1	0	1	1	0	1	1	1

After the execution,

CY	B7	B6	B5	B4	B3	B2	B1	B0
1	0	1	1	0	1	1	1	1

**3. ROR instruction**

This instruction rotates all the bits in a specified byte or word to the right some number of bit positions. LSB is placed as a new MSB and a new CF. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected

**General Format: ROR**

destination, count

**Example:**

MOV BL, B7H ; BL is made B7H

ROR BL, 1 ; shift the content of BL register one place to the right.

Before execution,

B7	B6	B5	B4	B3	B2	B1	B0	CY
1	0	1	1	0	1	1	1	0

After execution,

B7	B6	B5	B4	B3	B2	B1	B0	CY
1	1	0	1	1	0	1	1	1

**4. RCR instruction**



This instruction rotates all the bits in a specified byte or word to the right some number of bit positions along with the carry flag. LSB is placed in a new CF and previous carry is placed in the new MSB. The destination can be of byte size or of word size, also it can be a register or a memory location. Number of shifts is indicated by the count.

All flags are affected

**General Format: RCR**

destination, count

**Example:**

MOV BL, B7H ; BL is made B7H

RCR BL, 1 ; shift the content of BL register one place to the right.

Before execution,

B7 B6 B5 B4 B3 B2 B1 B0 CY

1 0 1 1 0 1 1 1 0

After execution,

B7 B6 B5 B4 B3 B2 B1 B0 CY

0 1 0 1 1 0 1 1 1

**Program control transfer instructions**

There are 2 types of such instructions. They are:

1. Unconditional transfer instructions – CALL, RET, JMP
2. Conditional transfer instructions – J condition

**1. CALL instruction**

The CALL instruction is used to transfer execution to a subprogram or procedure. There are two types of CALL instructions, near and far.

A **near CALL** is a call to a procedure which is in the same code segment as the CALL instruction. 8086 when encountered a near call, it decrements the SP by 2 and copies the offset of the next instruction after the CALL on the stack. It loads the IP with

the offset of the procedure then to start the execution of the procedure.

A **far CALL** is the call to a procedure residing in a different segment. Here value of CS and offset of the next instruction both are backed up in the stack. And then branches to the procedure by changing the content of CS with the segment base containing procedure and IP with the offset of the first instruction of the procedure.

#### **Example:**

Near call

CALL PRO ; PRO is the name of the procedure

CALL CX ; Here CX contains the offset of the first instruction of the procedure, that is replaces the content of IP with the content of CX

Far call

CALL DWORD PTR[8X] ; New values for CS and IP are fetched from four memory locations in the DS. The new value for CS is fetched from [8X] and [8X+1], the new IP is fetched from [8X+2] and [8X+3].

## **2. RET instruction**

RET instruction will return execution from a procedure to the next instruction after the CALL instruction in the calling program. If it was a near call, then IP is replaced with the value at the top of the stack, if it had been a far call, then another POP of the stack is required. This second popped data from the stack is put in the CS, thus resuming the execution of the calling program. RET instruction can be followed by a number, to specify the parameters passed. RET instruction does not affect any flags.

#### **General format:**

RET

#### **Example:**

p1 PROC procedure declaration.

MOV  
Δ X

RET return to caller.

p1 ENDP

## **3 JMP instruction**

This is also called as unconditional jump instruction, because the processor jumps to

the specified location rather than the instruction after the JMP instruction. Jumps can be **short jumps** when the target address is in the same segment as the JMP instruction or **far jumps** when it is in a different segment.

**General Format:** JMP<target address> **Example:**

```

MOV AL,05H          ;
                    ;
JMP label1          ;      jump over to label

MOV
AL, 00H      ;
label1:      MOV
[2000H], AL;
RET          ;

```

#### 4. Conditional Jump (J cond)

Conditional jumps are always short jumps in 8086. Here jump is done only if the condition specified is true/false. If the condition is not satisfied, then the execution proceeds in the normal way.

**Ex  
am  
ple:**

```

There are many conditional
jump instructions like JC      :
                          Jump on carry (CF=set)
JNC      :      Jump on non carry (CF=reset)

JZ :      Jump on zero (ZF=set)

JNO      :      Jump on overflow (OF=set)

```

Etc

#### 5. Iteration control instructions

These instructions are used to execute a series of instructions some number of times. The number is specified in the CX register, which will be automatically decremented in course of iteration. But here the destination address for the jump must be in the range of -128 to 127 bytes.

**Example:**

Instructions here are:-

LOOP : loop through the set of instructions until CX is 0  
LOOPE/LOOPZ : here the set of instructions are repeated until CX=0 or ZF=0  
LOOPNE/LOOPNZ: here repeated until CX=0 or ZF=1

### **Machine Control Instructions**

#### **1. HLT instruction**

The HLT instruction will cause the 8086 microprocessor to fetching and executing instructions.

The 8086 will enter a halt state. The processor gets out of this Halt signal upon an interrupt signal in INTR pin/NMI pin or a reset signal on RESET input.

#### **General form:-**

HLT

#### **2. WAIT instruction**

When this instruction is executed, the 8086 enters into an idle state. This idle state is continued till a high is received on the TEST input pin or a valid interrupt signal is received. Wait affects no flags. It generally is used to synchronize the 8086 with a peripheral device(s).

#### **3. ESC instruction**

This instruction is used to pass instruction to a coprocessor like 8087. There is a 6 bit instruction for the coprocessor embedded in the ESC instruction. In most cases the 8086 treats ESC and a NOP, but in some cases the 8086 will access data items in memory for the coprocessor

#### **4. LOCK instruction**

In multiprocessor environments, the different microprocessors share a system bus, which is needed to access external devices like disks. LOCK

Instruction is given as prefix in the case when a processor needs exclusive access of the system bus for a particular instruction. It affects no flags.

#### **Example:**

LOCK XCHG SEMAPHORE, AL : The XCHG instruction requires two bus accesses. The lock prefix prevents another processor from taking control of the system bus between the 2 accesses

#### **5. NOP instruction**

At the end of NOP instruction, no operation is done other than the fetching and decoding of the instruction. It takes 3 clock cycles. NOP is used to fill in time delays or to provide space for instructions while trouble shooting. NOP affects no flags.

### **Flag manipulation instructions**

### **1. STC instruction**

This instruction sets the carry flag. It does not affect any other flag.

### **2. CLC instruction**

This instruction resets the carry flag to zero. CLC does not affect any other flag.

### **3. CMC instruction**

This instruction complements the carry flag. CMC does not affect any other flag.

### **4. STD instruction**

This instruction is used to set the direction flag to one so that SI and/or DI can be decremented automatically after execution of string instruction. STD does not affect any other flag.

### **5. CLD instruction**

This instruction is used to reset the direction flag to zero so that SI and/or DI can be incremented automatically after execution of string instruction. CLD does not affect any other flag.

### **6. STI instruction**

This instruction sets the interrupt flag to 1. This enables INTR interrupt of the 8086. STI does not affect any other flag.

### **7. CLI instruction**

This instruction resets the interrupt flag to 0. Due to this the 8086 will not respond to an interrupt signal on its INTR input. CLI does not affect any other flag.

## **String Instructions**

### **1. MOVS/MOVSMB/MOVSX**

These instructions copy a word or byte from a location in the data segment to a location in the extra segment. The offset of the source is in SI and that of destination is in DI. For multiple word/byte transfers the count is stored in the CX register.

When direction flag is 0, SI and DI are incremented and when it is 1, SI and DI are decremented.

MOVS affect no flags. MOVSMB is used for byte sized movements while MOVSX is for word sized.

### **Example:**

CLD ; clear the direction flag to auto increment SI and DI

```

MOV AX, 0000H;
MOV DS, AX   ; initialize data segment register to 0
MOV ES, AX   ; initialize extra segment register to 0
MOV SI, 2000H ; Load the offset of the string1 in SI
MOV DI, 2400H ; Load the offset of the string2 in DI
MOV CX, 04H  ; load length of the string in CX
REP MOVSB    ; decrement CX and MOVSB until CX will be 0

```

## 2. REP/REPE/REP2/REPNE/REPNZ

REP is used with string instruction; it repeats an instruction until the specified condition becomes false.

Example:	Comments
REP	CX=0
REPE/REPZ	CX=0 OR ZF=0
REPNE/REPNZ	CX=0 OR ZF=1

## 3. LODS/LODSB/LODSW

This instruction copies a byte from a string location pointed to by SI to AL or a word from a string location pointed to by SI to AX. LODS does not affect any flags. LODSB copies byte and LODSW copies word.

### Example:

```

CLD                ; clear direction flag to auto increment SI
MOV SI, OFFSET S_STRING ; point SI at string
LODS S_STRING      ;

```

## 4. STOS/STOSB/STOSW

The STOS instruction is used to store a byte/word contained in AL/AX to the offset contained in the DI register. STOS does not affect any flags. After copying the content DI is automatically incremented or decremented, based on the value of direction flag.

### Example:

```

MOV DI, OFFSET D_STRING; assign DI with destination address.

```

```

STOS D_STRING          ; assembler uses string name to determine byte or
                        ; Word, if byte then AL is used and if of word size, AX is used.

```

## 5. CMPS/CMPSB/CMPSW

CMPS is used to compare the strings, byte wise or word wise. The

comparison is affected by subtraction of content pointed by DI from that pointed by SI. The AF, CF, OF, PF, SF and ZF flags are affected by this instruction, but neither operand is affected.

<b>Example:</b>		<b>Comments</b>
MOV SI, OFFSET STRING_A	;	Point first string
MOV DI, OFFSET STRING_B	;	Point second string
MOV CX, 0AH	;	Set the counter as 0AH
CLD	;	Clear direction flag to auto increment
REPE CMPSB	;	Repeatedly compare till unequal or counter =0

## **ASSEMBLER DIRECTIVES**

There are some instructions in the assembly language program which are not a part of processor instruction set. These instructions are instructions to the assembler, linker and loader. These are referred to as pseudo-operations or as assembler directives. The assembler directives enable us to control the way in which a program assembles and lists. They act during the assembly of a program and do not generate any executable machine code.

There are many specialized assembler directives. Let us see the commonly used assembler directive in 8086 assembly language programming.

### **1. ASSUME:**

It is used to tell the name of the logical segment the assembler to use for a specified segment.

E.g.: ASSUME CS: CODE tells that the instructions for a program are in a logical segment named CODE.

### **2. DB -Define Byte:**

The DB directive is used to reserve byte or bytes of memory locations in the available memory. While preparing the EXE file, this directive directs the assembler to allocate the specified number of memory bytes to the said data type that may be a constant, variable, string, etc. Another option of this directive also initializes the reserved memory bytes with the ASCII codes of the characters specified as a string. The following examples show how the DB directive is used for different purposes.

- 1) RANKS DB 01H,02H,03H,04H

This statement directs the assembler to reserve four memory locations for a list named RANKS and initialize them with the above specified four values.

- 2) MESSAGE DB „GOOD MORNING“

This makes the assembler reserve the number of bytes of memory equal to the number of

characters in the string named MESSAGE and initializes those locations by the ASCII equivalent of these characters.

3) VALUE DB 50H

This statement directs the assembler to reserve 50H memory bytes and leave them uninitialized for the variable named VALUE.

**3. DD -Define Double word** - used to declare a double word type variable or to reserve memory locations that can be accessed as double word.

E.g.:           ARRAY       \_POINTER       DD       25629261H declares a  
                  double       word named ARRAY\_POINTER.

**4. DQ -Define Quad word**

This directive is used to direct the assembler to reserve 4 words (8 bytes) of memory for the specified variable and may initialize it with the specified values.

E.g.:           BIG\_NUMBER           DQ       2432987456292612H  
                  declares           a       quad       word   named  
BIG\_NUMBER.

**5. DT -Define Ten Bytes:**

The DT directive directs the assembler to define the specified variable requiring 10-bytes for its storage and initialize the 10-bytes with the specified values. The directive may be used in case of variables facing heavy numerical calculations, generally processed by numerical processors.

E.g.: PACKED\_BCD 11223344556677889900 declares an array that is 10 bytes in length.

**6. DW -Define Word:**

The DW directives serves the same purposes as the DB directive, but it now makes the assembler reserve the number of memory words (16-bit) instead of bytes. Some examples are given to explain this directive.

1) WORDS DW 1234H, 4567H, 78ABH, 045CH

This makes the assembler reserve four words in memory (8 bytes), and initialize the words with the specified values in the statements. During initialization, the lower bytes are stored at the lower memory addresses, while the upper bytes are stored at the higher addresses.

2) NUMBER1 DW 1245H

This makes the assembler reserve one word in memory.

**7. END-End of Program:**

The END directive marks the end of an assembly language program. When the assembler comes across this END directive, it ignores the source lines available later on. Hence, it should be ensured that the END statement should be the last



statement in the file and should not appear in between. Also, no useful program statement should lie in the file, after the END statement.

- 8. ENDP-End Procedure** - Used along with the name of the procedure to indicate the end of a procedure.

E.g.: SQUARE\_ROOT PROC: start of procedure  
SQUARE\_ROOT ENDP: End of procedure

**9. ENDS-End of Segment:**

This directive marks the end of a logical segment. The logical segments are assigned with the names using the ASSUME directive. The names appear with the ENDS directive as prefixes to mark the end of those particular segments. Whatever are the contents of the segments, they should appear in the program before ENDS. Any statement appearing after ENDS will be neglected from the segment. The structure shown below explains the fact more clearly.

DATA SEGMENT

-----

----- DATA

ENDS

ASSUME CS: CODE, DS: DATA

CODE SEGMENT

-----

----- CODE

ENDS

- 10. EQU-Equate** - Used to give a name to some value or symbol. Each time the assembler finds the given name in the program, it will replace the name with the value.

E.g.: CORRECTION\_FACTOR EQU 03H  
MOV AL, CORRECTION\_FACTOR

- 11. EVEN** - Tells the assembler to increment the location counter to the next even address if it is not already at an even address.

Used because the processor can read even addressed data in one clock cycle

- 12. EXTRN** - Tells the assembler that the names or labels following the directive are in some other assembly module.

For example if a procedure in a program module assembled at a different time from that which contains the CALL instruction, this directive is used to tell the assembler that the procedure is external

- 13. GLOBAL** - Can be used in place of a PUBLIC directive or in place of an EXTRN directive.

It is used to make a symbol defined in one module available to other modules.

E.g.: GLOBAL DIVISOR makes the variable DIVISOR public so that it can be accessed from other modules.

**14. GROUP-**Used to tell the assembler to group the logical statements named after the directive into one logical group segment, allowing the contents of all the segments to be accessed from the same group segment base.

E.g.: `SMALL_SYSTEM GROUP CODE, DATA, STACK_SEG`

**15. INCLUDE** - Used to tell the assembler to insert a block of source code from the named file into the current source module.

This will shorten the source code.

**16. LABEL-** Used to give a name to the current value in the location counter.

This directive is followed by a term that specifies the type you want associated with that name.

E.g.: `ENTRY_POINT LABEL FAR`

`NEXT: MOV AL, BL`

**17. NAME-** Used to give a specific name to each assembly module when programs consisting of several modules are written.

E.g.: `NAME PC_BOARD`

**18. OFFSET-** Used to determine the offset or displacement of a named data item or procedure from the start of the segment which contains it.

E.g.: `MOV BX, OFFSET PRICES`

**19. ORG-** The location counter is set to 0000 when the assembler starts reading a segment. The ORG directive allows setting a desired value at any point in the program.

E.g.: `ORG 2000H`

**20. PROC-** Used to identify the start of a procedure.

E.g.: `SMART_DIVIDE PROC FAR` identifies the start of a procedure named `SMART_DIVIDE` and tells the assembler that the procedure is far

**21. PTR-** Used to assign a specific type to a variable or to a label.

E.g.: `INC BYTE PTR[BX]` tells the assembler that we want to increment the byte pointed to by BX

**22. PUBLIC-** Used to tell the assembler that a specified name or label will be accessed from other modules.

E.g.: `PUBLIC DIVISOR, DIVIDEND` makes the two variables `DIVISOR` and `DIVIDEND` available to other assembly modules.

**23. SEGMENT-** Used to indicate the start of a logical segment.

E.g.: `CODE SEGMENT` indicates to the assembler the start of a logical segment called `CODE`

**24. SHORT-** Used to tell the assembler that only a 1 byte displacement is needed to code a jump instruction.

E.g.: `JMP SHORT NEARBY_LABEL`

**25. TYPE** - Used to tell the assembler to determine the type of a specified variable.

E.g.: `ADD BX, TYPE WORD_ARRAY` is used where we want to increment BX to point to the next word in an array of words.

#### **Macros:**

Macro is a group of instruction. The macro assembler generates the code in the program each time where the macro is “called”. Macros can be defined by `MACROP` and `ENDM` assembler directives. Creating macro is very similar to creating a new opcode that can be used in the program, as shown below.

Example:

```
INIT MACRO MOV
    AX,@DATA MOV DS
MOV ES, AX ENDM
```

It is important to note that macro sequences execute faster than procedures because there is no `CALL` and `RET` instructions to execute. The assembler places the macro instructions in the program each time when it is invoked. This procedure is known as Macro expansion.

#### **WHILE:**

In Macro, the `WHILE` statement is used to repeat macro sequence until the expression specified with it is true. Like `REPEAT`, end of loop is specified by `ENDM` statement. The `WHILE` statement allows to use relational operators in its expressions.

The table-1 shows the relational operators used with `WHILE` statements.

OPERATOR	FUNCTION
EQ	Equal
NE	Not equal
LE	Less than or equal
LT	Less than
GE	Greater than or equal
GT	Greater than
NOT	Logical inversion
AND	Logical AND
OR	Logical OR

Table-1: Relational operators used in `WHILE` statement.

#### **FOR statement:**

A `FOR` statement in the macro repeats the macro sequence for a list of data. For

example, if we pass two arguments to the macro then in the first iteration the FOR statement gives the macro sequence using first argument and in the second iteration it gives the macro sequence using second argument. Like WHILE statement, end of FOR is indicated by ENDM statement. The program shows the use of FOR statement in the macro.

Example1:

```
DISP MACRO CHR MOV AH,  
    02H FOR ARG, <CHR>  
    MOV DL, ARG INT 21H  
ENDM ENDM  
. MODEL SMALL  
  
. CODE  
  
START: DISP „M“, „A“, „C“, „R“, „O“ END START
```

### **CODE FOR 8 BIT ADDER**

```
DATA SEGMENT  
    A1 DB 50H  
    A2 DB 51H  
    RES DB ?  
DATA ENDS  
CODE SEGMENT  
ASSUME CS: CODE, DS:DATA  
START: MOV AX,DATA  
        MOV DS,AX  
        MOV AL,A1  
        MOV BL,A2  
        ADD AL,BL  
        MOV RES,AL  
        MOV AX,4C00H  
        INT 21H  
        CODE ENDS  
        END START
```

### **CODE FOR 16 BIT ADDER**

```
DATA SEGMENT  
    A1 DW 0036H  
    A2 DW 0004H  
    SUM DW ?  
DATA ENDS  
CODE SEGMENT  
ASSUME CS:CODE,DS:DATA
```

```

START: MOV AX,DATA
      MOV DS,AX
      MOV AX,A1
      MOV BX,A2
      DIV BX
      MOV SUM,AX
      MOV AX,0008H
      INT 21H
      CODE ENDS
      END START

```

### **ADD33 MATRIX**

```

.MODEL SMALL
.DATA
M1 DB 10H,20H,30H,40H,50H,60H,70H,80H,90H
M2 DB 10H,20H,30H,40H,50H,60H,70H,80H,90H
RESULT DW 9 DUP (0)
.CODE
START: MOV AX,@DATA
      MOV DS,AX
      MOV CX,9
      MOV DI,OFFSET M1
      MOV BX,OFFSET M2
      MOV SI,OFFSET
      RESULT
BACK:  MOV AH,00
      MOV AL,[DI]
      ADD AL,[BX]
      ADC AH,00
      MOV [SI],AX
      INC DI
      INC BX
      INC SI
      INC SI
      LOOP BACK
      MOV AH,4CH
      INT 21H
      END START
      END

```

### **ARRAY SUM**

```

.MODEL SMALL
.DATA
ARRAY DB 12H, 24H, 26H, 63H, 25H, 86H, 2FH, 33H, 10H, 35H

```

```

SUM DW 0
.CODE
START:MOV AX, @DATA
      MOV DS, AX
      MOV CL, 10
      XOR DI, DI
      LEA BX, ARRAY
BACK: MOV AL, [BX+DI]
      MOV AH, 00H
      ADD SUM, AX
      INC DI
      DEC CL
      JNZ BACK
      END START

```

### **ASCIITOHX**

```

DATA SEGMENT
  A DB 41H
  R DB ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV AL,A
        SUB AL,30H
        CMP AL,39H
        JBE L1
        SUB AL,7H
        L1: MOV R,AL
            INT 3H
CODE ENDS
END START

```

### **AVERAGE**

```

.MODEL SMALL
.STACK 100
.DATA
  NO1 DB 63H
  NO2 DB 2EH
  AVG DB ?
.CODE
START: MOV AX,@DATA
      MOV DS,AX

```

```
    MOV AL,NO1
    ADD AL,NO2
    ADC AH,00H
    SAR AX,1
    MOV AVG,AL
END START
```

### **16 BIT SUB**

```
DATA SEGMENT
    A1 DW 1001H
    A2 DW 1000H
    SUB DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
        MOV DS,AX
        MOV AX,A1
        MOV BX,A2
        SBB AX,BX
        MOV SUB,AX
        MOV AX,4C00H
        INT 21H
CODE ENDS
END START
```

### **16BIT SUM**

```
DATA SEGMENT
    A1 DW 1000H
    A2 DW 1001H
    SUM DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
        MOV DS,AX
        MOV AX,A1
        MOV BX,A2
        ADC AX,BX
        MOV SUM,AX
        MOV AX,4C00H
        INT 21H
```

```
CODE ENDS
END START
```

### **8BMUL**

```
DATA SEGMENT
    A1 DB 25H
    A2 DB 25H
    A3 DB ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS:DATA
START:MOV AX,DATA
        MOV DS,AX
        MOV AL,A1
        MOV BL,A2
        MUL BL
        MOV A3,AL
        MOV AX,4C00H
        INT 21H
CODE ENDS
END START
```

### **16BIT MUL**

```
DATA SEGMENT
    A1 DW 1000H
    A2 DW 1000H
    A3 DW ?
    A4 DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS:DATA
START:MOV AX,DATA
        MOV DS,AX
        MOV AX,A1
        MOV BX,A2
        MUL BX
        MOV A3,DX
        MOV A4,AX
        MOV AX,4C00H
        INT 21H
CODE ENDS
```



END START

### **EVENODD**

```
DATA SEGMENT
    ORG 2000H
    FIRST DW 3H
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
        MOV DS,AX
        MOV AX,FIRST
        SHR AX,1
        JC L1
        MOV BX,00
        INT 3H
    L1: MOV BX,01
        INT 3H
CODE ENDS
END START
```

### **FACTORIAL**

```
DATA SEGMENT
    ORG 2000H
    FIRST DW 3H
    SEC DW 1H
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
        MOV DS,AX
        MOV AX,SEC
        MOV CX,FIRST
    L1:  MUL CX
        DEC CX
        JCXZ L2
        JMP L1
    L2:  INT 3H
CODE ENDS
END START
```

## **FIBONOCCHI**

```
DATA SEGMENT
    ORG 2000H
    FIRST DW 0H
    SEC DW 01H
    THIRD DW 50H
    RESULT DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS:DATA
START:  MOV AX,DATA
        MOV DS,AX
        MOV SI,OFFSET RESULT
        MOV AX,FIRST
        MOV BX,SEC
        MOV CX,THIRD
        MOV [SI],AX
    L1:  INC SI
        INC SI
        MOV [SI],BX
        ADD AX,BX
        XCHG AX,BX
        CMP BX,CX
        INT 3H
CODE ENDS
END START
```

## **FIND NUMBER**

```
.MODEL SMALL
.STACK 100
.DATA
ARRAY DB 63H,32H,45H,75H,12H,42H,09H,14H,56H,38H
SER_NO DB 09H
SER_POS DB ?
.CODE
START:MOV AX,@DATA
    MOV DS,AX
    MOV ES,AX
    MOV CX,000AH
    LEA DI,ARRAY
    MOV AL,SER_NO
    CLD
```

```

    REPNE SCAS ARRAY
    MOV AL,10
    SUB AL CL
    MOV SER_POS,AL
END START

```

### **GREATER**

```

DATA SEGMENT
ORG 2000H
FIRST  DW  5H,2H,3H,1H,4H
    COUNT EQU (($-FIRST)/2)-1
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS:DATA
START: MOV AX,DATA
        MOV DS,AX
        MOV CX,COUNT
        MOV SI,OFFSET FIRST
        MOV AX,[SI]
L2: INC SI
        INC SI
        MOV BX,[SI]
        CMP AX,BX
        JGE L1
        XCHG AX,BX
        JMP L1
L1: DEC CX
        JCXZ L4
        JMP L2
L4: INT 3H
CODE ENDS
END START

```

### **HEX TO ASCII**

```

DATA SEGMENT
A DB 08H
C DB ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START: MOV AX,DATA
        MOV DS,AX
        MOV AL,A
        ADD AL,30H

```

```

        CMP AL,39H
        JBE L1
        ADD AL,7H
L1:     MOV C,AL
        INT 3H
CODE ENDS
END START

```

### **MAX**

```

.MODEL SMALL
.STACK 100
.DATA
ARRAY DB 63H,32H,45H,75,12H,42H,09H,14H,56H,38H
MAX DB 0
.CODE
START:MOV AX,@DATA
        MOV DS,AX
        XOR DI,DI
        MOV CL,10
        LEA BX,ARRAY
        MOV AL,MAX
BACK:   CMP AL,[BX+DI]
        JNC SKIP
        MOV DL,[BX+DI]
        MOV AL,DL
SKIP:   INC DI
        DEC CL
        JNZ BACK
        MOV MAX,AL
        MOV AX,4C00H
        INT 21H
END START

```

### **NO OF 1S**

```

DATA SEGMENT
    ORG 2000H
    FIRST DW 7H
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START: MOV AX,DATA
        MOV DS,AX
        MOV AX,FIRST
        MOV BX,00
        MOV CX,16

```

```

L2: SHR AX,1
    JC L1
L4: DEC CX
    JCXZ L3
    JMP L2
L1: INC BX
    JMP L4
L3: INT 3H
CODE ENDS
END START

```

### **SMALLER**

```

DATA SEGMENT
ORG 2000H
    FIRST DW 5H,2H,3H,1H,4H
    COUNT EQU (( $\$$ -FIRST)/2)-1
    RESULT DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS:DATA
START: MOV AX,DATA
    MOV DS,AX
    MOV CX,COUNT
    MOV SI,OFFSET FIRST
    MOV AX,[SI]
L2: INC SI
    INC SI
    MOV BX,[SI]
    CMP AX,BX
    JB L1
    XCHG AX,BX
    JMP L1
L1: DEC CX
    JCXZ L4
    JMP L2
L4: MOV RESULT,AX
CODE ENDS
END START

```

### **SUM OF CUBES**

```

DATA SEGMENT
    ORG 2000H
    NUM DB 1H

```

```

        RES DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START: MOV DX,DATA
        MOV DS,AX
        MOV CL,NUM
        MOV BX,00
L1:     MOV AL,CL
        MOV CH,CL MUL AL
        MUL CH
        ADD BX,AX
        DEC CL
        JNZ L1
        MOV RES,BX
        INT 3H
CODE ENDS
END START

```

### **SUM OF SQUARES**

```

DATA SEGMENT
NUM DW 5H
RES DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START:  MOV AX,DATA
        MOV DS,AX MOV
        CX,NUM MOV BX,00
L1:     MOV AX,CX
        MUL CX
        ADD BX,AX
        DEC CX
        JNZ L1
        MOV RES,BX
        INT 3H
CODE ENDS
END START

```

## UNIT-III

### 8255 PROGRAMMABLE PERIPHERAL INTERFACE (PPI)

#### Introduction:

Any application of a microprocessor based system requires the transfer of data between external circuitry to the microprocessor and microprocessor to the external circuitry. User can give information to the microprocessor based system using keyboard and user can see the result or output information from the microprocessor based system with the help of display device. The transfer of data between keyboard and microprocessor, and microprocessor and display device is called input/output data transfer or I/O data transfer. This data transfer is done with the help of I/O ports.

#### Input port:

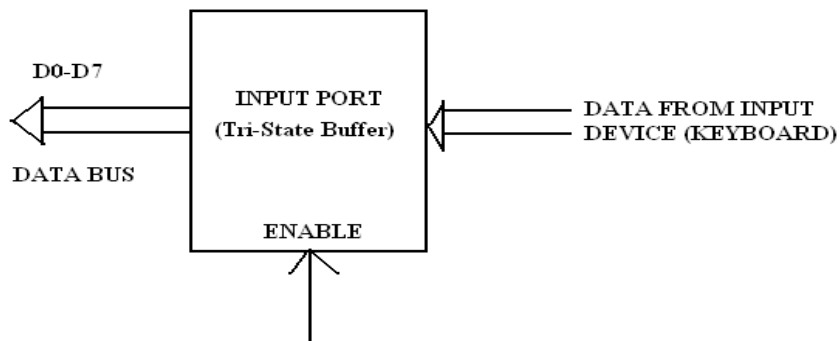


FIG.1 INPUT PORT

It is used to read data from the input device such as keyboard. The simplest form of input port is a buffer. The input device is connected to the microprocessor through buffer, as shown in the fig.1. This buffer is a tri-state buffer and its output is available only when enable signal is active. When microprocessor wants to read data from the input device (keyboard), the control signals from the microprocessor activates the buffer by asserting enable input of the buffer. Once the buffer is enabled, data from the input device is available on the data bus. Microprocessor reads this data by initiating read command.

#### Output port:

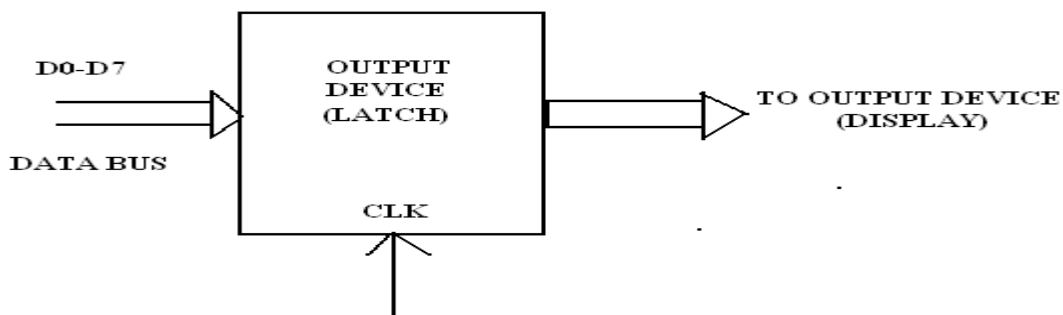


FIG.2 OUTPUT PORT

It is used to send data to the output device such as display from the microprocessor. The simplest form of output port is a latch. The output device is connected to the microprocessor through latch, as shown in the fig.2. When microprocessor wants to send data to the output device it puts the data on the data bus and activates the clock signal of the latch, latching the data from the data bus at the output of latch. It is then available at the output of latch for the output device.

### **Serial and Parallel Transmission:**

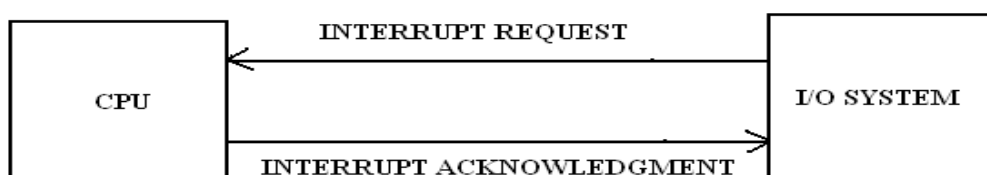
In telecommunications, serial transmission is the sequential transmission of signal elements of a group representing a character or other entity of data. Digital serial transmissions are bits sent over a single wire, frequency or optical path sequentially. Because it requires less signal processing and less chance for error than parallel transmission, the transfer rate of each individual path may be faster. This can be used over longer distances as a check digit or parity bit can be sent along it easily.

In telecommunications, parallel transmission is the simultaneous transmission of the signal elements of a character or other entity of data. In digital communications, parallel transmission is the simultaneous transmission of related signal elements over two or more separate paths. Multiple electrical wires are used which can transmit multiple bits simultaneously, which allows for higher data transfer rates than can be achieved with serial transmission. This method is used internally within the computer, for example the internal buses, and sometimes externally for such things as printers, The major issue with this is "skewing" because the wires in parallel data transmission have slightly different properties (not intentionally) so some bits may arrive before others, which may corrupt the message. A parity bit can help to reduce this. However, electrical wire parallel data transmission is therefore less reliable for long distances because corrupt transmissions are far more likely.

### **Interrupt driven I/O:**

In this technique, a CPU automatically executes one of a collection of special routines whenever certain condition exists within a program or a processor system. Example CPU gives response to devices such as keyboard, sensor and other components when they request for service. When the CPU is asked to communicate with devices, it services the devices. Example each time you type a character on a keyboard, a keyboard service routine is called. It transfers the character you typed from the keyboard I/O port into the processor and then to a data buffer in memory.

The interrupt driven I/O technique allows the CPU to execute its main program and only stop to service I/O device when it is told to do so by the I/O system as shown in fig.3. This method provides an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is completed, the processor would resume exactly where it left off.



**FIG.3 INTERRUPT DRIVEN I/O**



An analogy to the interrupt concept is in the classroom, where the professor serves as CPU and the students as I/O ports. The classroom scenario for this interrupt analogy will be such that the professor is busy in writing on the blackboard and delivering his lecture.

The student raises his finger when he wants to ask a question (student requesting for service). The professor then completes his sentence and acknowledges student's request by saying "YES" (professor acknowledges the interrupt request). After acknowledgement from the professor, student asks the question and professor gives answer to the question (professor services the interrupt). After that professor continues its remaining lecture from where it was left.

### **PIO 8255:**

The parallel input-output port chip 8255 is also called as programmable **peripheral input-output port**. The Intel's 8255 are designed for use with Intel's 8-bit, 16-bit and higher capability microprocessors. It has 24 input/output lines which may be individually programmed in two groups of twelve lines each, or three groups of eight lines.

The two groups of I/O pins are named as Group A and Group B. Each of these two groups contains a subgroup of eight I/O lines called as 8-bit port and another subgroup of four lines or a 4-bit port. Thus Group A contains an 8-bit port Along with a 4-bit port C upper. Group B contains an 8-bit port B and a 4-bit port C lower.

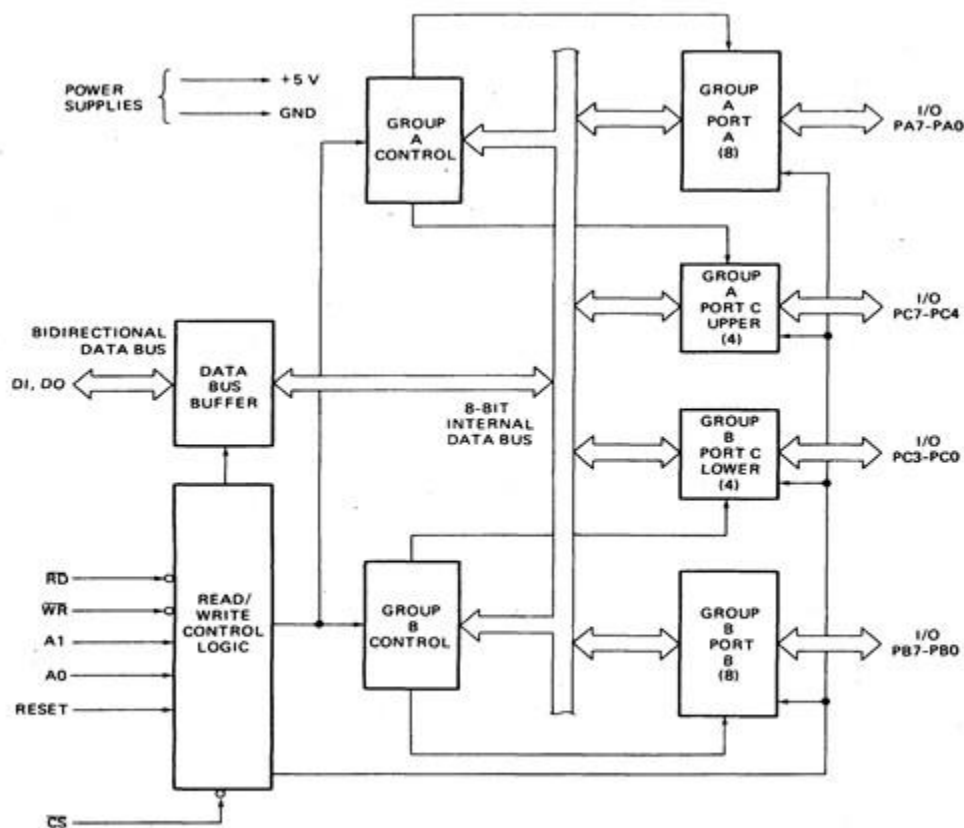


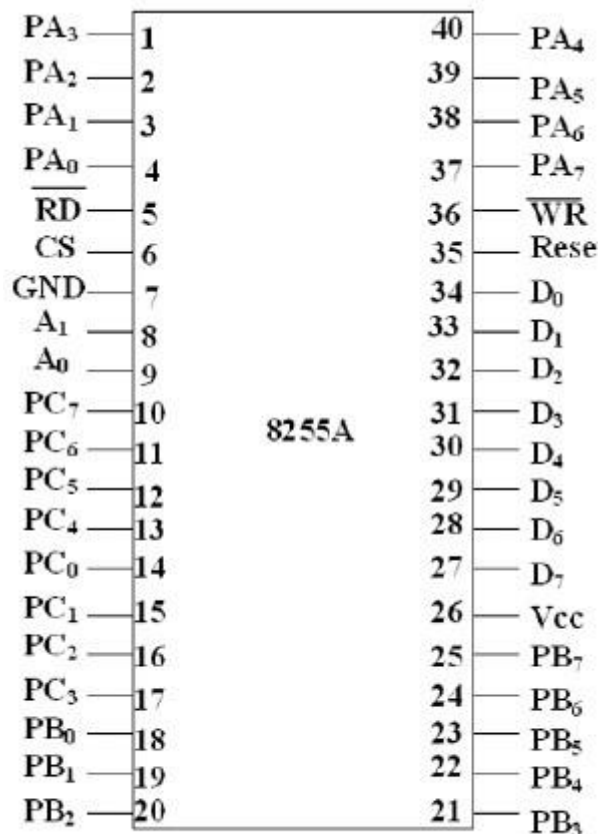
FIGURE Internal block diagram of 8255A programmable parallel port device. (Intel Corporation)

The port A lines are identified by symbols PA0-PA7 while the port C lines are

identified as PC4-PC7 similarly. Group B contains an 8-bit port B, containing lines PB0- PB7 and a 4-bit port C with lower bits PC0-PC3. The port C upper and port C lower can be used in combination as an 8-bit port C. Both the port Cs is assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit I/O ports from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR). The internal block diagram and the pin configuration of 8255 are shown in figs.

The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfer of both data and control words. RD, WR, A1, A0 and RESET are the inputs, provided by the microprocessor to READ/WRITE control logic of 8255. The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus. This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.

### Pin Diagram of 8255A



**8255A Pin Configuration**

The pin configuration of 8255 is shown in fig.

- The port A lines are identified by symbols PA0-PA7 while the port C lines are
- Identified as PC4-PC7. Similarly, Group B contains an 8-bit port B, containing lines PB0-PB7 and a 4-bit port C with lower bits PC0- PC3. The port C upper and

port C lower can be used in combination as an 8-bit port C.

- Both the port C is assigned the same address. Thus one may have either three 8-bit I/O ports or two 8-bit and two 4-bit ports from 8255. All of these ports can function independently either as input or as output ports. This can be achieved by programming the bits of an internal register of 8255 called as control word register (CWR).
- The 8-bit data bus buffer is controlled by the read/write control logic. The read/write control logic manages all of the internal and external transfers of both data and control words.
- RD, WR, A1, A0 and RESET are the inputs provided by the microprocessor to the READ/ WRITE control logic of 8255. The 8-bit, 3-state bidirectional buffer is used to interface the 8255 internal data bus with the external system data bus.
- This buffer receives or transmits data upon the execution of input or output instructions by the microprocessor. The control words or status information is also transferred through the buffer.

**The signal description of 8255 is briefly presented as follows:**

- **PA7-PA0:** These are eight port A lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.
- **PC7-PC4:** Upper nibble of port C lines. They may act as either output latches or input buffers lines.
- This port also can be used for generation of handshake lines in mode1 or mode2.
- **PC3-PC0:** These are the lower port C lines; other details are the same as PC7-PC4 lines.
- **PB0-PB7:** These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.
- **RD:** This is the input line driven by the microprocessor and should be low to indicate read operation to 8255.
- **WR:** This is an input line driven by the microprocessor. A low on this line indicates write operation.
- **CS:** This is a chip select line. If this line goes low, it enables the 8255 to respond to RD and WR signals, otherwise RD and WR signal are neglected.
- **D0-D7:** These are the data bus lines those carry data or control word to/from the microprocessor.
- **RESET:** Logic high on this line clears the control word register of 8255. All ports are set as input ports by default after reset.
- **A1-A0:** These are the address input lines and are driven by the microprocessor.
- These lines A1-A0 with RD, WR and CS from the following operations for 8255. These address lines are used for addressing any one of the four registers, i.e. three ports and a control word register as given in table below.

In case of 8086 systems, if the 8255 is to be interfaced with lower order data bus, the A0 and A1 pins of 8255 are connected with A1 and A2 respectively.

$\overline{\text{RD}}$	$\overline{\text{WR}}$	$\overline{\text{CS}}$	A <sub>1</sub>	A <sub>0</sub>	Input (Read) cycle
0	1	0	0	0	Port A to Data bus
0	1	0	0	1	Port B to Data bus
0	1	0	1	0	Port C to Data bus
0	1	0	1	1	CWR to Data bus

$\overline{\text{RD}}$	$\overline{\text{WR}}$	$\overline{\text{CS}}$	A <sub>1</sub>	A <sub>0</sub>	Output (Write) cycle
1	0	0	0	0	Data bus to Port A
1	0	0	0	1	Data bus to Port B
1	0	0	1	0	Data bus to Port C
1	0	0	1	1	Data bus to CWR

$\overline{\text{RD}}$	$\overline{\text{WR}}$	$\overline{\text{CS}}$	A <sub>1</sub>	A <sub>0</sub>	Function
X	X	1	X	X	Data bus tristated
1	1	0	X	X	Data bus tristated

Control Word Register

### Modes of Operation of 8255

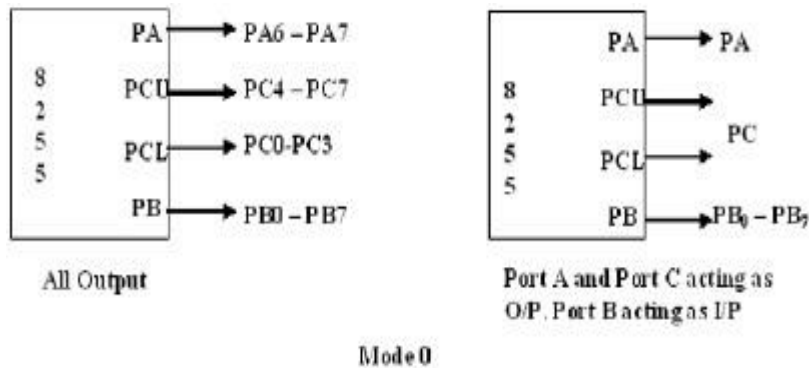
- These are two basic modes of operation of 8255. I/O mode and Bit Set-Reset mode (BSR).
- In I/O mode, the 8255 ports work as programmable I/O ports, while in BSR mode only port C (PC0-PC7) can be used to set or reset its individual port bits.
- Under the I/O mode of operation, further there are three modes of operation of 8255, so as to support different types of applications, mode 0, mode 1 and mode 2.
- **BSR Mode:** In this mode any of the 8-bits of port C can be set or reset depending on D0 of the control word. The bit to be set or reset is selected by bit select flags D3, D2 and D1 of the CWR as given in table.

### I/O Modes:

**a) Mode 0 (Basic I/O mode):** This mode is also called as basic input/output Mode. This mode provides simple input and output capabilities using each of the three ports. Data can be simply read from and written to the input and output ports respectively, after appropriate initialization.

D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	Selected bits of port C
0	0	0	D <sub>0</sub>
0	0	1	D <sub>1</sub>
0	1	0	D <sub>2</sub>
0	1	1	D <sub>3</sub>
1	0	0	D <sub>4</sub>
1	0	1	D <sub>5</sub>
1	1	0	D <sub>6</sub>
1	1	1	D <sub>7</sub>

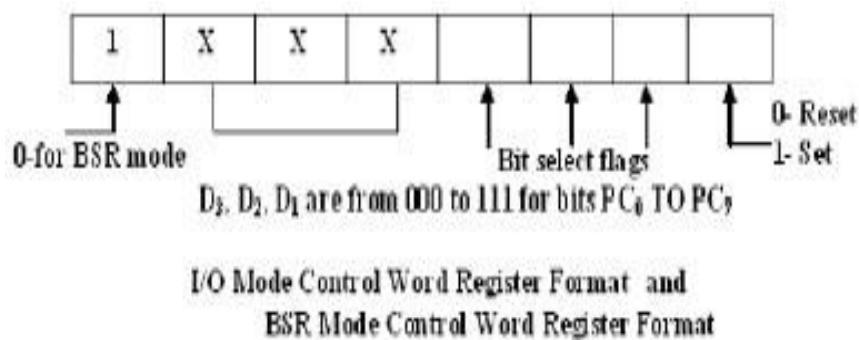
**BSR Mode : CWR Format**

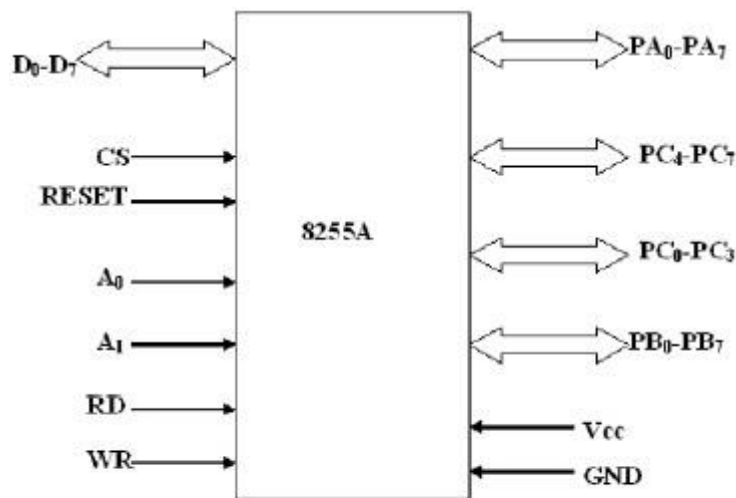


The salient features of this mode are as listed below:

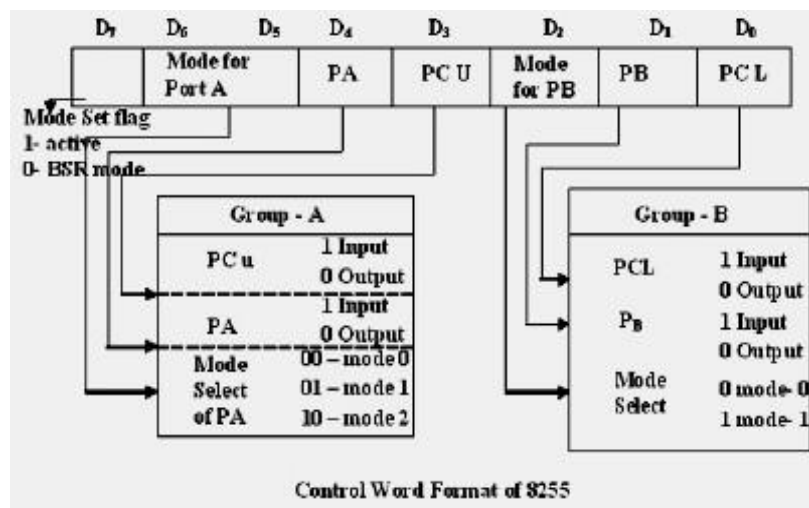
1. Two 8-bit ports (port A and port B) and two 4-bit ports (port C upper and lower) are available. The two 4-bit ports can be combined used as a third 8-bit port.
  2. Any port can be used as an input or output port.
  3. Output ports are latched. Input ports are not latched.
  4. A maximum of four ports are available so that overall 16 I/O configurations are possible.
- All these modes can be selected by programming a register internal to 8255 known as CWR.
  - The control word register has two formats. The first format is valid for I/O modes of operation, i.e. modes 0, mode 1 and mode 2 while the second format is valid for bit set/reset (BSR) mode of operation.

These formats are shown in following fig.





Signals of 8255



Control Word Format of 8255

**b) Mode 1: (Strobed input/output mode)** in this mode the handshaking control the input and output action of the specified port. Port C lines PC0-PC2, provide strobe or handshake lines for port B. This group which includes port B and PC0-PC2 is called as group B for Strobed data input/output. Port C lines PC3-PC5 provides strobe lines for port A. This group including port A and PC3-PC5 from group A. Thus port C is utilized for generating handshake signals.

The salient features of mode 1 are listed as follows:

1. Two groups – group A and group B are available for strobed data transfer.
2. Each group contains one 8-bit data I/O port and one 4-bit control/data port.

3. The 8-bit data port can be either used as input and output port. The inputs and outputs both are latched.
4. Out of 8-bit port C, PC0-PC2 are used to generate control signals for port B and PC3-PC5 are used to generate control signals for port A. the lines PC6, PC7 may be used as independent data lines.

**The control signals for both the groups in input and output modes are explained as follows:**

**Input control signal definitions (mode 1):**

- **STB** (Strobe input) – If this line falls to logic low level, the data available at 8-bit input port is loaded into input latches.
- **IBF** (Input buffer full) – If this signal rises to logic 1, it indicates that data has been loaded into latches, i.e. it works as an acknowledgement. IBF is set by a low on STB and is reset by the rising edge of RD input.
- **INTR** (Interrupt request) – This active high output signal can be used to interrupt the CPU whenever an input device requests the service. INTR is set by a high STB pin and a high at IBF pin. INTE is an internal flag that can be controlled by the bit set/reset mode of either PC4 (INTEA) or PC2 (INTEB) as shown in fig.
- INTR is reset by a falling edge of RD input. Thus an external input device can be request the service of the processor by putting the data on the bus and sending the strobe signal.

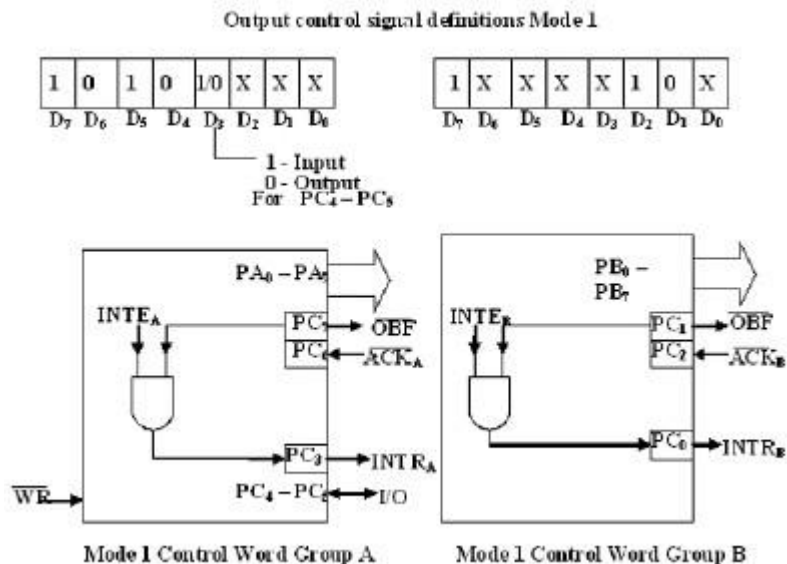
**Output control signal definitions (mode 1):**

- **OBF** (Output buffer full) – This status signal, whenever falls to low, indicates that CPU has written data to the specified output port. The OBF flip-flop will be set by a rising edge of WR signal and reset by a low going edge at the ACK input.
- **ACK** (Acknowledge input) – ACK signal acts as an acknowledgement to be given by an output device. ACK signal, whenever low, informs the CPU that the data transferred by the CPU to the output device through the port is received by the output device.
- **INTR** (Interrupt request) – Thus an output signal that can be used to interrupt the CPU when an output device acknowledges the data received from the CPU. INTR is set when ACK, OBF and INTE are 1. It is reset by a

Falling edge on WR input. The INTEA and INTEB flags are controlled by the bit set-reset mode of PC6 and PC2 respectively.







c) **Mode 2 (Strobed bidirectional I/O):** This mode of operation of 8255 is also called as strobed bidirectional I/O. This mode of operation provides 8255 with additional features for communicating with a peripheral device on an 8-bit data bus. Handshaking signals are provided to maintain proper data flow and synchronization between the data transmitter and receiver. The interrupt generation and other functions are similar to mode 1.

In this mode, 8255 is a bidirectional 8-bit port with handshake signals. The Rd and WR signals decide whether the 8255 is going to operate as an input port or output port.

The Salient features of Mode 2 of 8255 are listed as follows:

1. The single 8-bit port in group A is available.
2. The 8-bit port is bidirectional and additionally a 5-bit control port is available.
3. Three I/O lines are available at port C.( PC2 – PC0 )
4. Inputs and outputs are both latched.
5. The 5-bit control port C (PC3-PC7) is used for generating / accepting handshake signals for the 8-bit data transfer on port A.

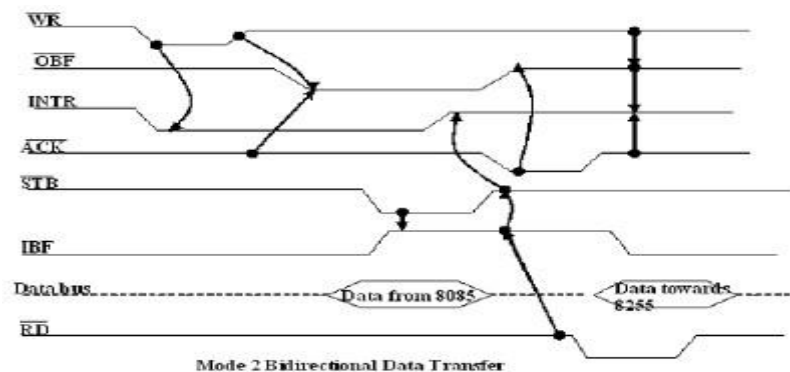
#### Control signal definitions in mode 2:

- **INTR** – (Interrupt request) As in mode 1, this control signal is active high and is used to interrupt the microprocessor to ask for transfer of the next data byte to/from it. This signal is used for input (read) as well as output (write) operations.
- **Control Signals for Output operations:**
- **OBF** (Output buffer full) – This signal, when falls to low level, indicates that the CPU has written data to port A.
- **ACK** (Acknowledge) This control input, when falls to logic low level, Acknowledges that the previous data byte is received by the destination and next byte may be sent by the processor. This signal enables the internal tristate buffers to send the next data byte on port A.
- **INTE1** ( A flag associated with OBF ) This can be controlled by bit set/reset mode

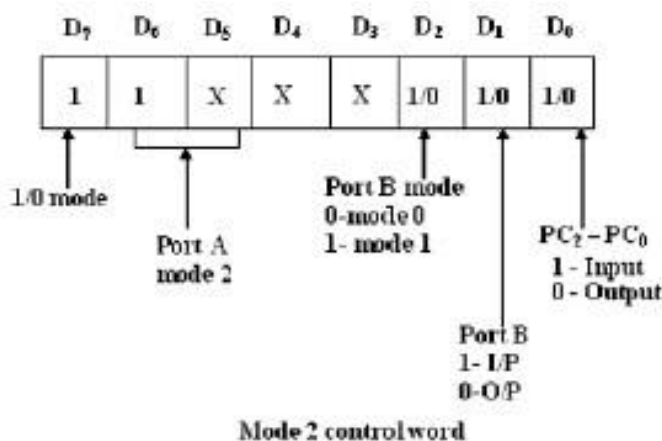
with PC6.

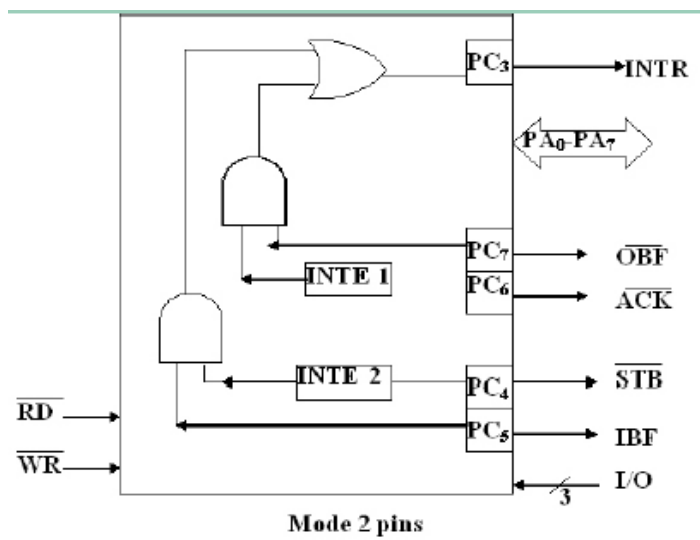
### Control signals for input operations:

- STB (Strobe input) a low on this line is used to strobe in the data into the input Latches of 8255.
- **IBF** (Input buffer full) when the data is loaded into input buffer, this signal rises to logic „1“. This can be used as an acknowledge that the data has been received by the receiver.
- The waveforms in fig show the operation in Mode 2 for output as well as input port.
- Note: WR must occur before ACK and STB must be activated before RD.



- The following fig shows a schematic diagram containing an 8-bit bidirectional port, 5-bit control port and the relation of INTR with the control pins. Port B can either be set to Mode 0 or 1 with port A (Group A) is in Mode 2.
- Mode 2 is not available for port B. The following fig shows the control word.
- The INTR goes high only if IBF, INTE2, STB and RD go high or OBF,
- INTE1, ACK and WR go high. The port C can be read to know the status of the peripheral device, in terms of the control signals, using the normal I/O instructions.





### **Interfacing Analog to Digital Data Converters:**

- In most of the cases, the PIO 8255 is used for interfacing the analog to digital converters with microprocessor.
- We have already studied 8255 interfacing with 8086 as an I/O port, in previous section. This section we will only emphasize the interfacing techniques of analog to digital converters with 8255.
- The analog to digital converters is treated as an input device by the microprocessor that sends an initializing signal to the ADC to start the analogy to digital data conversation process. The start of conversation signal is a pulse of a specific duration.
- The process of analog to digital conversion is a slow
- Process and the microprocessor have to wait for the digital data till the conversion is over. After the conversion is over, the ADC sends end of conversion EOC signal to inform the microprocessor that the conversion is over and the result is ready at the output buffer of the ADC. The set asks of issuing an SOC pulse to ADC, reading EOC signal from the ADC and reading the digital output of the ADC are carried out by the CPU using 8255 I/O ports.
- The time taken by the ADC from the active edge of SOC pulse till the active edge of EOC signal is called as the conversion delay of the ADC.
- It may range anywhere from a few microseconds in case of fast ADC to even a few hundred milliseconds in case of slow ADCs.
- The available ADC in the market use different conversion techniques for conversion of analog signal to digitals. Successive approximation techniques and dual slope integration techniques are the most popular techniques used in the integrated ADC chip.
- General algorithm for ADC interfacing contains the following steps:
- Ensure the stability of analog input, applied to the ADC.
- Issue start of conversion pulse to ADC
- Read end of conversion signal to mark the end of conversion processes.
- Read digital data output of the ADC as equivalent digital output.
- Analog input voltage must be constant at the input of the ADC right from the start of conversion till the end of the conversion to get correct results. This may be ensured by as ample and hold circuit which samples the analog signal and holds it constant for specific time duration. The microprocessor may issue a hold signal to the sample and hold circuit.
- If the applied input changes before the complete conversion process is over, the digital equivalent of the analog input calculated by the ADC may not be correct.

### **ADC 0808/0809:**

- The analog to digital converter chips 0808 and 0809 are 8-bit CMOS, successive approximation converters. This technique is one of the fast techniques for analog to digital conversion. The conversion delay is 100 $\mu$ s at a clock frequency of 640 KHz, which is quite low as compared to other converters. These converters do not need any external zero or full scale adjustments as they are already taken care of by internal circuits.
- These converters internally have a 3:8 analog multiplexer so that at a time eight different analog conversion by using address lines - ADD A, ADD B, ADD C, as

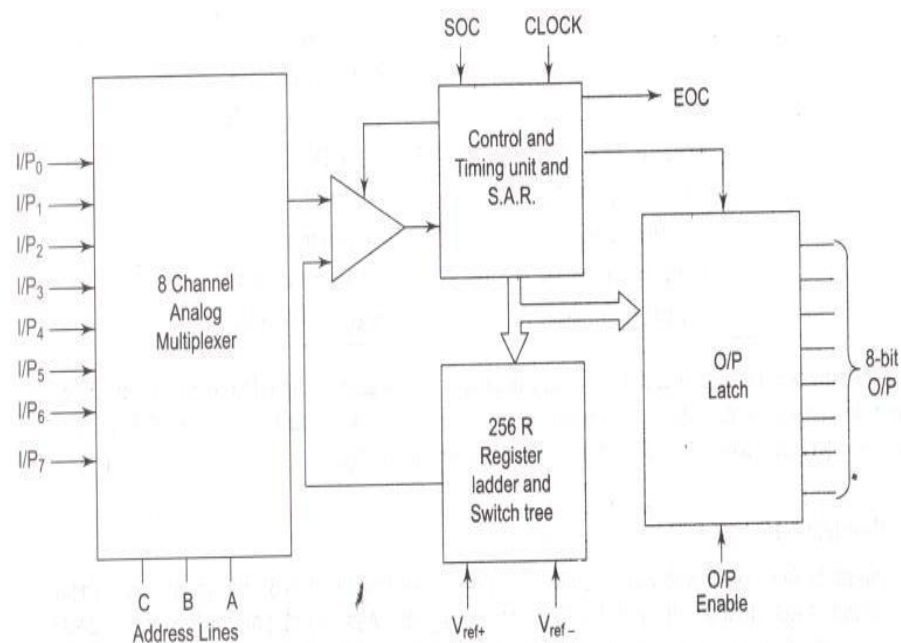
shown. Using these address inputs, multichannel data acquisition system can be designed using a single ADC. The CPU may drive these lines using output port lines in case of multichannel applications. In case of single input applications, these may be hardwired to select the proper input.

- There are unipolar analog to digital converters, i.e. they are able to convert only positive analog input voltage to their digital equivalent. These chips do not contain any internal sample and hold circuit.
- If one needs a sample and hold circuit for the conversion of fast signal into equivalent digital quantities, it has to be externally connected at each of the analog inputs.

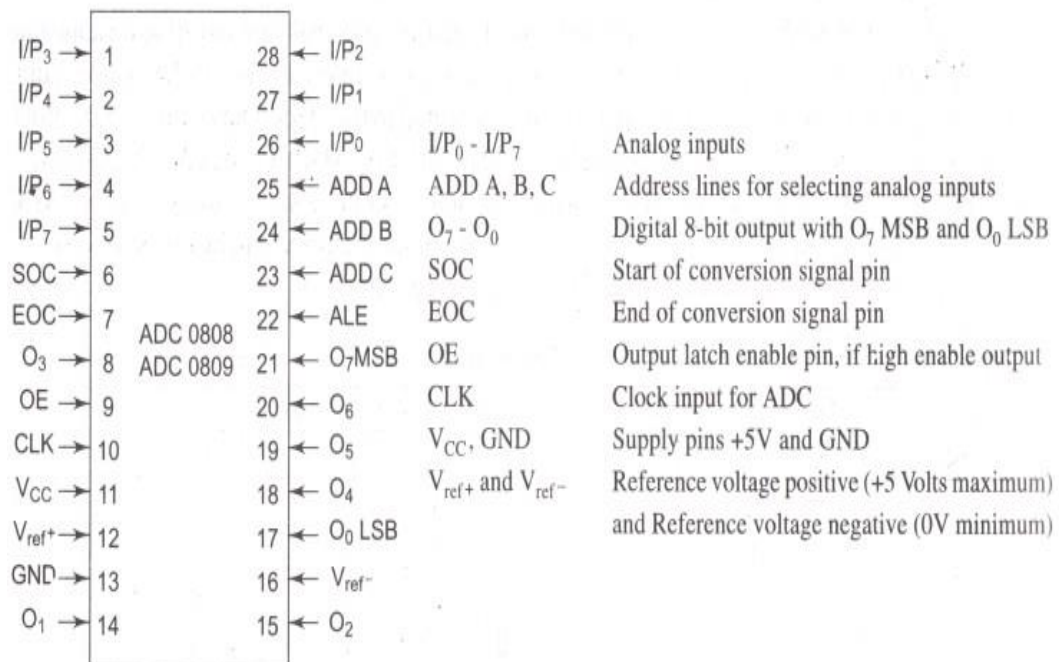
Fig (1) and Fig (2) show the block diagrams and pin diagrams for ADC 0808/0809.

**Table.1**

Analog I/P selected	Address lines		
	C	B	A
I/P 0	0	0	0
I/P 1	0	0	1
I/P 2	0	1	0
I/P 3	0	1	1
I/P 4	1	0	0
I/P 5	1	0	1
I/P 6	1	1	0
I/P 7	1	1	1



**Fig.1 Block Diagram of ADC 0808/0809**



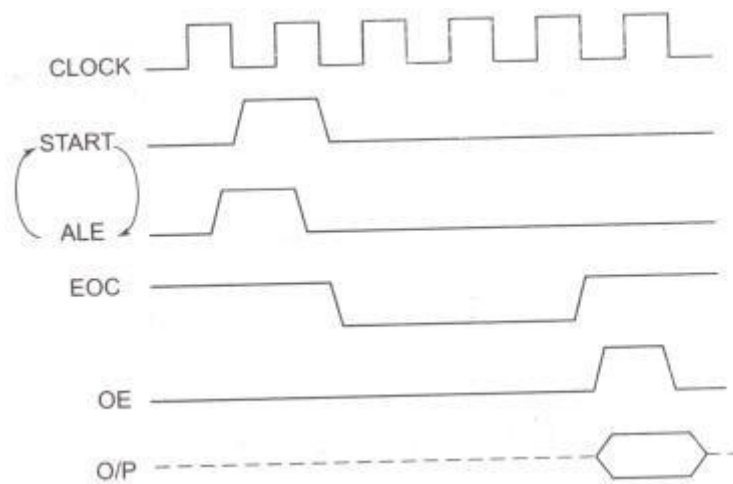
**Fig.2 Pin Diagram of ADC 0808/0809**

Some Electrical Specifications Of The ADC 0808/0809 Are Given In Table.2.

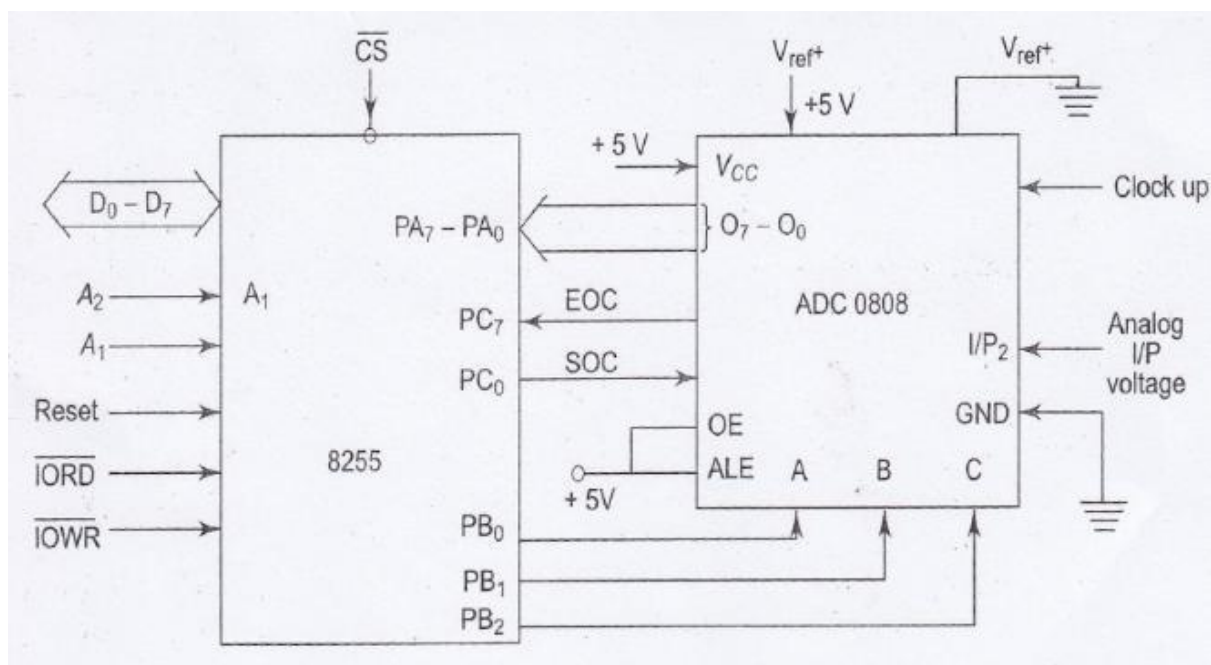
**Table.2**

Minimum SOC pulse width	100 ns
Minimum ALE pulse width	100 ns
Clock frequency	10 to 1280 kHz
Conversion time	100 ms at 640 kHz
Resolution	8-bit
Error	+/-1 LSB
$V_{ref+}$	Not more than +5V
$V_{ref-}$	Not less than GND
+ $V_{cc}$ supply	+ 5 V DC
Logical 1 i/p voltage	minimum $V_{cc} - 1.5$ V
Logical 0 i/p voltage	maximum 1.5 V
Logical 1 o/p voltage	minimum $V_{cc} - 0.4$ V
Logical 0 o/p voltage	maximum 0.45 V

**The Timing Diagram Of Different Signals Of Adc0808 Is Shown In Fig.3**



**Fig.3 Timing Diagram Of ADC 0808.**



**Interfacing ADC0808 with 8086**

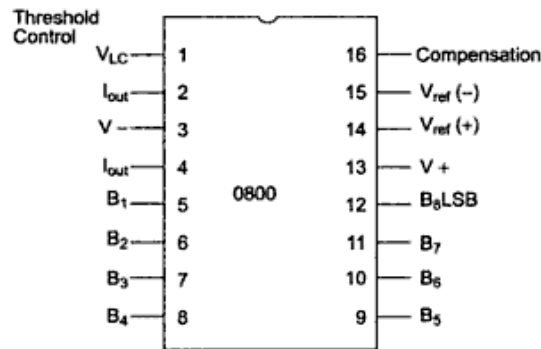
### **Interfacing Digital To Analog Converters:**

The digital to analog converters convert binary numbers into their analog equivalent voltages. The DAC find applications in areas like digitally controlled gains, motor speed controls, programmable gain amplifiers, etc.

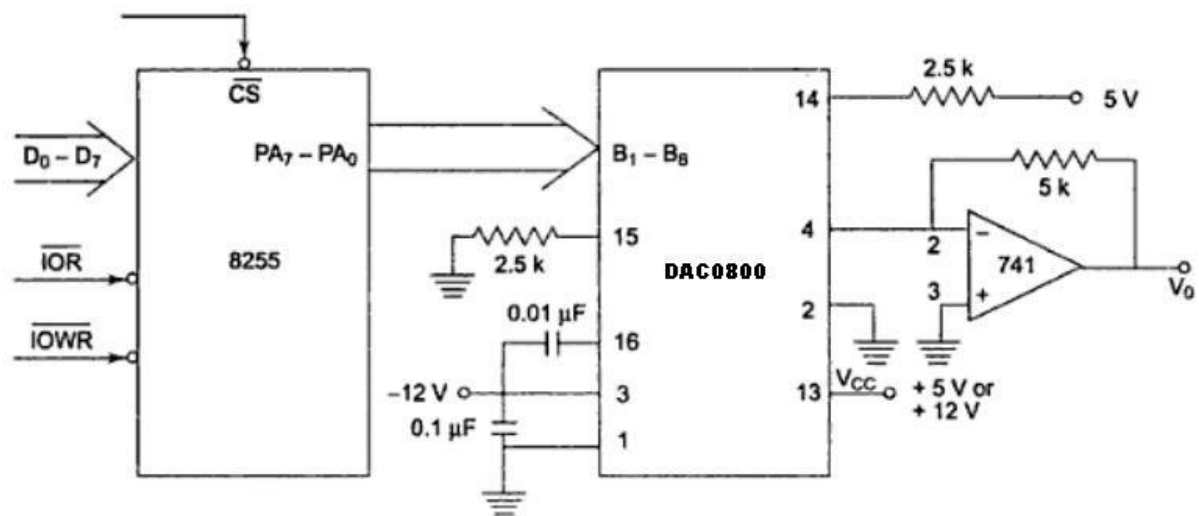
### **DAC0800 8-bit Digital to Analog Converter**

- The DAC 0800 is a monolithic 8-bit DAC manufactured by National Semiconductor.
- It has settling time around 100ms and can operate on a range of power supply voltages i.e. from 4.5V to +18V.
- Usually the supply V+ is 5V or +12V.

- The V-pin can be kept at a minimum of -12V.

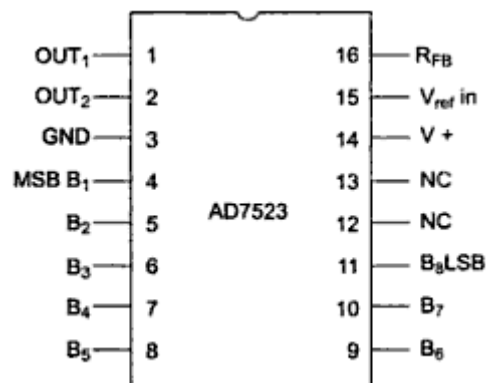


Pin Diagram of DAC 0800



### Interfacing DAC0800 with 8086 Ad 7523 8-Bit Multiplying DAC:

- Intersil's AD 7523 is a 16 pin DIP, multiplying digital to analog converter, containing R-2R ladder (R=10KΩ) for digital to analog conversion along with single pole double through NMOS switches to connect the digital inputs to the ladder.



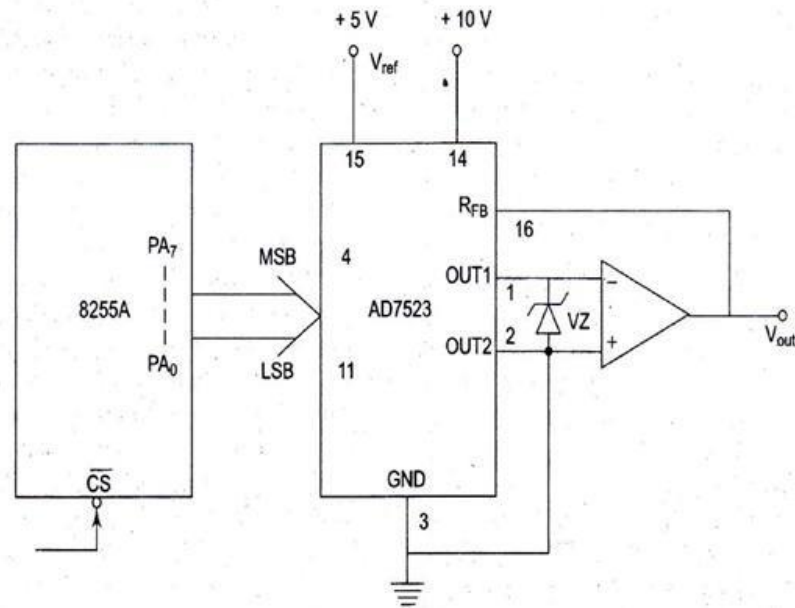
Pin Diagram of AD7523

- The supply range extends from +5V to +15V, while V<sub>ref</sub> may be anywhere



between -10V to +10V. The maximum analog output voltage will be +10V, when all the digital inputs are at logic high state. Usually a Zener is connected between OUT1 and OUT2 to save the DAC from negative transients.

- An operational amplifier is used as a current to voltage converter at the output of AD 7523 to convert the current output of AD7523 to a proportional output voltage.
- It also offers additional drive capability to the DAC output. An external feedback resistor acts to control the gain. One may not connect any external feedback resistor, if no gain control is required.



### Interfacing AD7523 with 8086 Stepper

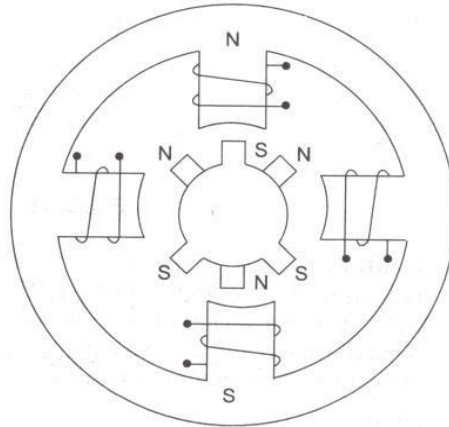
#### Motor Interfacing:

- A stepper motor is a device used to obtain an accurate position control of rotating shafts. It employs rotation of its shaft in terms of steps, rather than continuous rotation as in case of AC or DC motors. To rotate the shaft of the stepper motor, a sequence of pulses is needed to be applied to the windings of the stepper motor, in a proper sequence.
- The number of pulses required for one complete rotation of the shaft of the stepper motor is equal to its number of internal teeth on its rotor. The stator teeth and the rotor teeth lock with each other to fix a position of the shaft.
- With a pulse applied to the winding input, the rotor rotates by one teeth position or an angle  $x$ . The angle  $x$  may be calculated as:

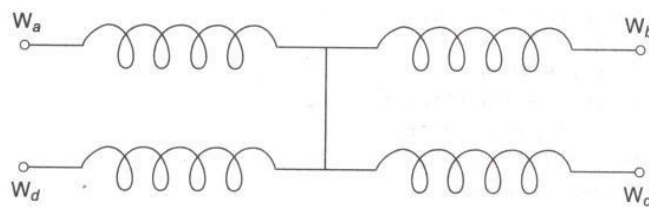
$$X = 360^\circ / \text{no. of rotor teeth}$$

- After the rotation of the shaft through angle  $x$ , the rotor locks itself with the next tooth in the sequence on the internal surface of stator.
- The internal schematic of a typical stepper motor with four windings is shown in fig.1.
- The stepper motors have been designed to work with digital circuits. Binary level pulses of 0-5V are required at its winding inputs to obtain the rotation of shafts. The sequence of the pulses can be decided, depending upon the required motion of the shaft.

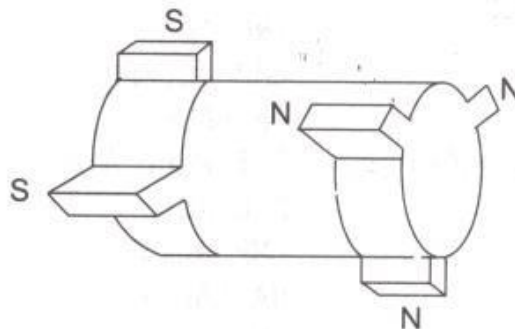
- Fig.2 shows a typical winding arrangement of the stepper motor.
- Fig.3 shows conceptual positioning of the rotor teeth on the surface of rotor, for a six teeth rotor.



**Fig.1 Internal schematic of a four winding stepper motor**



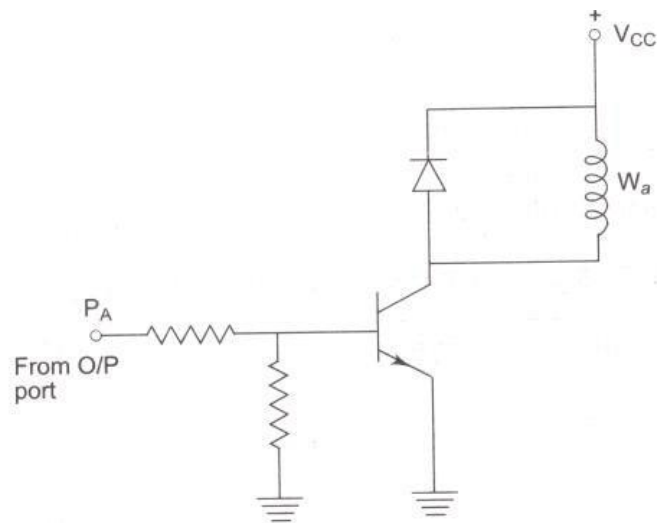
**Fig.2 Winding arrangement of a stepper motor.**



**Fig.3 Stepper motor rotor**

- The circuit for interfacing a winding  $W_n$  with an I/O port is given in fig.4. Each of the windings of a stepper motor needs this circuit for its interfacing with the output port. A typical stepper motor may have parameters like torque 3 Kg-cm, operating voltage 12V, current rating 0.2 A and a step angle  $1.8^\circ$  i.e. 200 steps/revolution (number of rotor teeth).

- A simple schematic for rotating the shaft of a stepper motor is called a wave scheme. In this scheme, the windings  $W_a$ ,  $W_b$ ,  $W_c$  and  $W_d$  are applied with the required voltages pulses, in a cyclic fashion. By reversing the sequence of excitation, the direction of rotation of the stepper motor shaft may be reversed.
- Table.1 shows the excitation sequences for clockwise and anticlockwise rotations. Another popular scheme for rotation of a stepper motor shaft applies pulses to two successive windings at a time but these are shifted only by one position at a time. This scheme for rotation of stepper motor shaft is shown in table2.



**Fig.4 interfacing stepper motor winding.**

**Table.1 Excitation sequence of a stepper motor using wave switching scheme.**

Motion	step	A	B	C	D
Clock Wise Direction	1	1	0	0	0
	2	0	1	0	0
	3	0	0	1	0
	4	0	0	0	1
	5	1	0	0	0
Anti clock wise Direction	1	1	0	0	0
	2	0	0	0	1
	3	0	0	1	0
	4	0	1	0	0
	5	1	0	0	0

**Table.2 An alternative scheme for rotating stepper motor shaft**

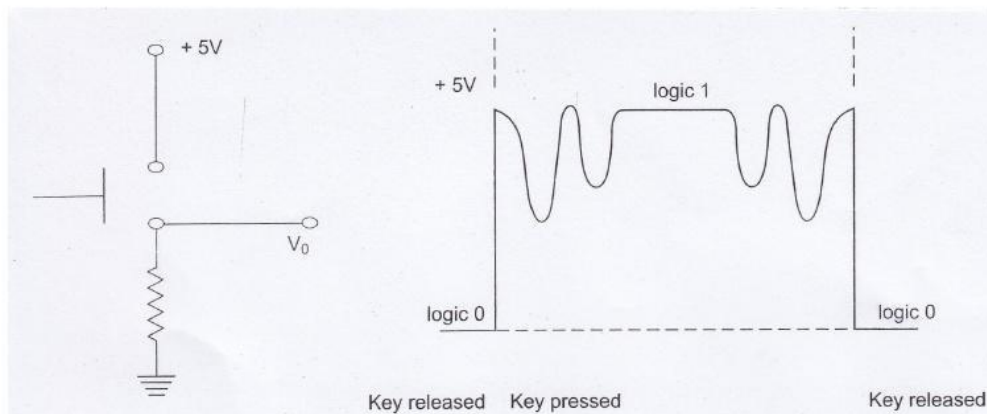
Motion	step	A	B	C	D
Clock wise Direction	1	0	0	1	1
	2	0	1	1	0
	3	1	1	0	0
	4	1	0	0	1
	5	0	0	1	1
Anti clock wise Direction	1	0	0	1	1
	2	1	0	0	1
	3	1	1	0	0
	4	0	1	1	0
	5	0	0	0	0

### **Keyboard Interfacing**

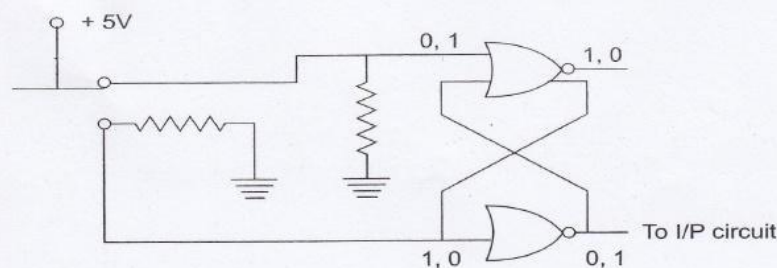
- In most keyboards, the key switches are connected in a matrix of Rows and Columns.
- Getting meaningful data from a keyboard requires three major tasks:
  1. detect a keypress
  2. Debounce the keypress.
  3. Encode the keypress (produce a standard code for the pressed key).
- Logic „0“ is read by the microprocessor when the key is pressed.

### **Key Debounce:**

Whenever a mechanical push-button is pressed or released once, the mechanical components of the key do not change the position smoothly; rather it generates a transient response. These may be interpreted as the multiple pressures and responded accordingly.

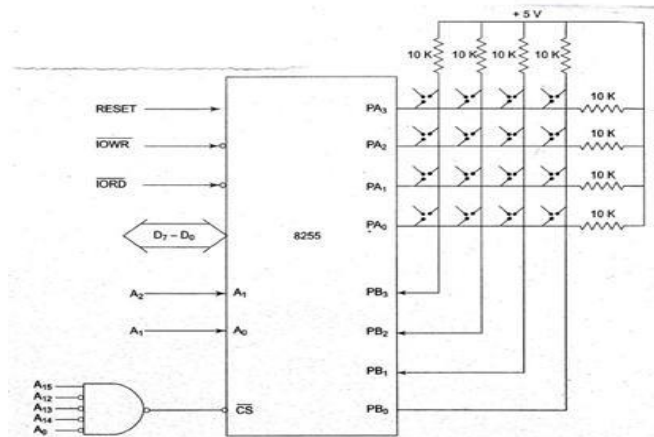


**Fig. 5.23** A Mechanical Key and Its Response



**Fig. 5.24** Hardware Debouncing Circuit

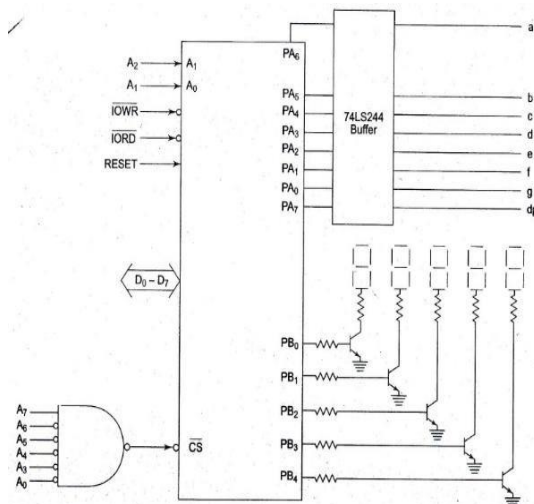
- The rows of the matrix are connected to four output Port lines, & columns are connected to four input Port lines.
- When no keys are pressed, the column lines are held high by the pull-up resistors connected to +5v.
- Pressing a key connects a row & a column.
- To detect if any key is pressed is to output 0's to all rows & then check columns to see if a pressed key has connected a low (zero) to a column.
- Once the columns are found to be all high, the program enters another loop, which waits until a low appears on one of the columns i.e indicating a key press.
- A simple 20/10 m sec delay is executed to debounce task.
- After the debounce time, another check is made to see if the key is still pressed. If the columns are now all high, then no key is pressed & the initial detection was caused by a noise pulse.
- To avoid this problem, two schemes are suggested:
  1. Use of Bistable multivibrator at the output of the key to debounce it.
  2. The microprocessor has to wait for the transient period (at least for 10 ms), so that the transient response settles down and reaches a steady state.
- If any of the columns are low now, then the assumption is made that it was a valid key press.
- The final task is to determine the row & column of the pressed key & convert this information to Hex-code for the pressed key.
- The 4-bit code from I/P port & the 4-bit code from O/P port (row & column) are converted to Hex-code.



## Interfacing 4x4 keyboard

### Display Interface

Number to be displayed	PA7d	PA6	PA5	PA4	PA3	PA2	PA1	PA0	Code
1	p	a	b	c	d	e	f	g	CF
2	1	0	0	1	0	0	1	0	92
3	1	0	0	0	0	1	1	0	86
4	1	1	0	0	1	1	0	0	CC
5	1	0	1	0	0	1	0	0	A4



## Interfacing multiplexed 7-segment display

## Interfacing with Advanced devices

### 4.1 MEMORY AND I/O INTERFACING

#### 4.1.1 I/O Interface

Any application of a microprocessor system requires the transfer of data between microprocessor and external environment and also within the microprocessor. This is known as Input/output. There are three different ways that the data transfer can take place. They are

- (1) Program controlled I/O
- (2) Interrupt Program Controlled I/O
- (3) Hardware controlled I/O

In program controlled I/O data transfer scheme the transfer of data is completely under the control of the microprocessor program. In this case an I/O operation takes place only when an I/O transfer instruction is executed.

In an interrupt program controlled I/O an external device indicates directly to the microprocessor its readiness to transfer data by a signal at an interrupt input of the microprocessor. When microprocessor receives this signal the control is transferred to ISS (Interrupt service subroutine) which performs the data transfer.

Hardware controlled I/O is also known as direct memory access DMA. In this case the data transfer takes place directly between an I/O device and memory but not through microprocessors. Microprocessor only initializes the process of data transfer by indicating the starting address and the number of words to be transferred.

The instruction set of any microprocessor contains instructions that transfer information to an I/O device and to read information from an I/O device. In 8086 we have IN, OUT instructions for this purpose. OUT instruction transfers information to an I/O device whereas IN instruction is used to read information from an I/O device. Both the instructions perform the data transfer using accumulator AL or AX. The I/O address is stored in register DX.

The port number is specified along with IN or OUT instruction. The external I/O interface decodes to find the address of the I/O device. The 8 bit fixed port number appears on address bus  $A_0 - A_7$  with  $A_8 - A_{15}$  all zeros. The address connections above  $A_{15}$  are undefined for an I/O instruction. The 16 bit variable port number appears on address connections  $A_0 - A_{15}$ . The above

notation indicates that first 256 I/O port addresses 00 to FF are accessed by both the fixed and variable I/O instructions. The I/O addresses from 0000 to FFFF are accessed by the variable I/O address.

I/O devices can be interfaced to the microprocessors using two methods. They are I/O mapped I/O and memory mapped I/O. The I/O mapped I/O is also known as isolated I/O or

direct I/O. In I/O mapped I/O the IN and OUT instructions transfer data between the accumulator or memory and I/O device. In memory mapped I/O the instruction that refers memory can perform the data transfer.

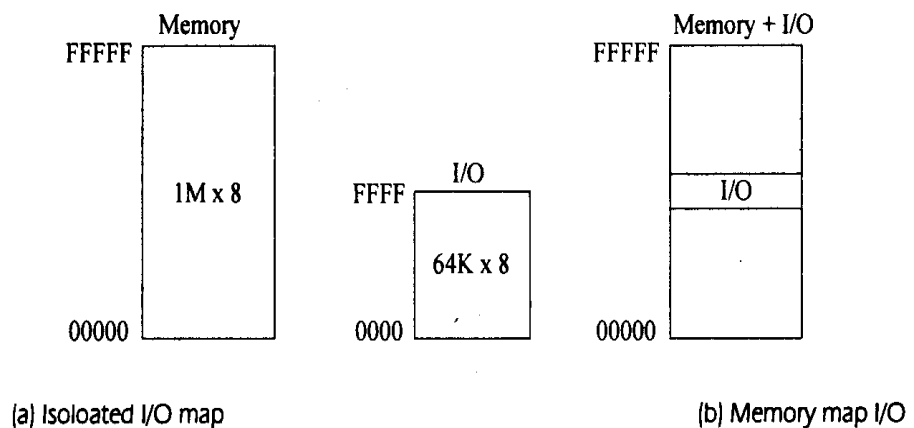


Fig. 3.12 Memory and I/O map of 8086

I/O mapped I/O is the most commonly used I/O transfer technique. In this method I/O locations are placed separately from memory. The addresses for isolated I/O devices are separate from memory. Using this method user can use the entire memory. This method allows data transfer only by using instructions IN, OUT. The pins  $M/\overline{IO}$  and  $W/R$  are used to indicate I/O read or an I/O write operations. The signals on these lines indicate that the address on the address bus is for I/O devices.

Memory mapped I/O does not use the IN, OUT instruction it uses only the instruction that transfers data between microprocessor and memory. A memory mapped I/O device is treated as memory location. The disadvantage in this system is the overall memory is reduced. The advantage of this system is that any memory transfer instruction can be used for data transfer and control signals like I/O read and I/O write are not necessary which simplify the hardware.

#### 4.1.2 Memory interfacing



Memory is an integral part of a microcomputer system. There are two main types of memory.

- (i) **Read only memory (ROM):** As the name indicates this memory is available only for reading purpose. The various types available under this category are PROM, EPROM, EEPROM which contain system software and permanent system data.
- (ii) **Random Access memory (RAM):** This is also known as Read Write Memory. It is a volatile memory. RAM contains temporary data and software programs generally for different applications.

While executing particular task it is necessary to access memory to get instruction codes and data stored in memory. Microprocessor initiates the necessary signals when read or write operation is to be performed. Memory device also requires some signals to perform read and write operations using various registers. To do the above job it is necessary to have a device and a circuit, which performs this task is known as interfacing device and as this is involved with memory it is known as memory interfacing device.

The basic concepts of memory interfacing involve three different tasks. The microprocessor should be able

to read from or write into the specified register. To do this it must be able to select the required chip, identify the required register and it must enable the appropriate buffers.

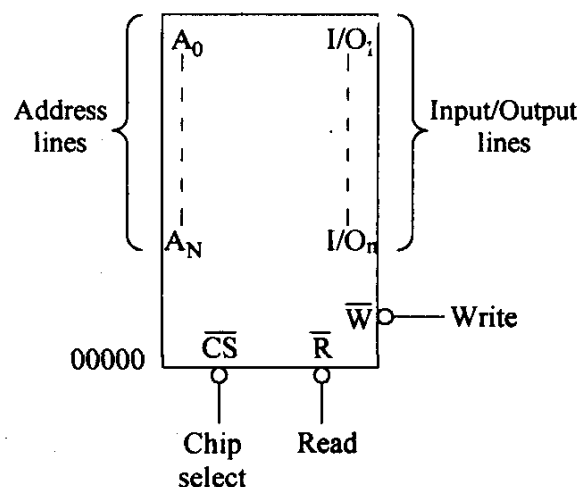


Fig. 3.13 Simple memory device

Any memory device must contain address lines and Input, output lines, selection input, control input to perform read or write operation. All memory devices have address inputs

that select memory location within the memory device. These lines are labeled as  $A_0$  .....  $A_N$ . The number of address lines indicates the total memory capacity of the memory device. A 1K memory requires 10 address lines  $A_0$ - $A_9$ . Similarly a 1MB requires 20 lines  $A_0$ - $A_{19}$  (in the case of 8086). The memory devices may have separate I/O lines or a common set of bidirectional I/O lines. Using these lines data can be transferred in either direction. Whenever output buffer is activated the operation is read whenever input buffers are activated the operation is write. These lines are labelled as I/O ... I/O<sub>n</sub> or D<sub>0</sub> .....D<sub>n</sub>. The size of a memory location is dependent upon the number of data bits. If the numbers of data lines are eight D<sub>0</sub> - D<sub>7</sub> then 8 bits or 1 byte of data can be stored in each location. Similarly if numbers of data bits are 16 (D<sub>0</sub> - D<sub>15</sub>) then the memory size is 2 bytes. For example 2K x 8 indicates there are 2048 memory locations and each memory location can store 8 bits of data.

Memory devices may contain one or more inputs which are used to select the memory device or to enable the memory device. This pin is denoted by  $\overline{CS}$  (Chip select) or  $\overline{CE}$  (Chip enable). When this pin is at logic '0' then only the memory device performs a read or a write operation. If this pin is at logic '1' the memory chip is disabled. If there are more than one  $\overline{CS}$  input then all these pins must be activated to perform read or write operation.

All memory devices will have one or more control inputs. When ROM is used we find  $\overline{OE}$  output enable pin which allows data to flow out of the output data pins. To perform this task both  $\overline{CS}$  and  $\overline{OE}$  must be active. A RAM contains one or two control inputs. They are  $\overline{R} / \overline{W}$  or  $\overline{RD}$  and  $\overline{WR}$ . If there is only one input  $R / \overline{W}$  then it performs read operation when  $R / \overline{W}$  pin is at logic 1. If it is at logic 0 it performs write operation. Note that this is possible only when  $\overline{CS}$  is also active.

#### **4.4 Memory Interface using RAMS, EPROMS and EEPROMS**

##### **Semiconductor Memory Interfacing:**

Semiconductor memories are of two types, viz. RAM (Random Access Memory) and ROM (Read Only Memory).

##### **Static RAM Interfacing:**

The semiconductor RAMs are of broadly two types-static RAM and dynamic RAM. The

semiconductor memories are organized as two dimensional arrays of memory locations. For example, 4K x 8 or 4K byte memory contains 4096 locations, where each location contains 8-bit data and only one of the 4096 locations can be selected at a time. Obviously, for addressing 4K bytes of memory, twelve address lines are required. In general, to address a memory location out of  $N$  memory locations, we will require at least  $n$  bits of address, i.e.  $n$  address lines where  $n = \log_2 N$ . Thus if the microprocessor has  $n$  address lines, then it is able to address at the most  $N$  locations of memory, where  $2^n = N$ . However, if out of  $N$  locations only  $P$  memory locations are to be interfaced, then the least significant  $p$  address lines out of the available  $n$  lines can be directly connected from the microprocessor to the memory chip while the remaining  $(n-p)$  higher order address lines may be used for address decoding (as inputs to the chip selection logic). The memory address depends upon the hardware circuit used for decoding the chip select ( $\overline{CS}$ ). The output of the decoding circuit is connected with the  $\overline{CS}$  pin of the memory chip. The general procedure of static memory interfacing with 8086 is briefly described as follows:

1. Arrange the available memory chips so as to obtain 16-bit data bus width. The upper 8-bit bank is called 'odd address memory bank' and the lower 8-bit bank is called 'even address memory bank'.
2. Connect available memory address lines of memory chips with those of the microprocessor and also connect the memory  $\overline{RD}$  and  $\overline{WR}$  inputs to the corresponding processor control signals. Connect the 16-bit data bus of the memory bank with that of the microprocessor 8086.
3. The remaining address lines of the microprocessor,  $\overline{BHE}$  and  $A_0$  are used for decoding the required chip select signals for the odd and even memory banks.  $\overline{CS}$  of memory is derived from the O/P of the decoding circuit.

As a good and efficient interfacing practice, the address map of the system should be continuous as far as possible, i.e. there should be no windows in the map. A memory location should have a single address corresponding to it, i.e. absolute decoding should be preferred, and minimum hardware should be used for decoding. In a number of cases, linear decoding may be used to minimise the required hardware. Let us now consider a few example problems on memory interfacing with 8086.

### Problem 5.1

Interface two  $4K \times 8$  EPROMs and two  $4K \times 8$  RAM chips with 8086. Select suitable maps.

**Solution** We know that, after reset, the IP and CS are initialised to form address FFFF0H. Hence, this address must lie in the EPROM. The address of RAM may be selected any where in the 1MB address space of 8086, but we will select the RAM address such that the address map of the system is continuous, as shown in Table 5.1.

**Table 5.1** Memory Map for Problem 5.1

Address	$A_{19}$	$A_{18}$	$A_{17}$	$A_{16}$	$A_{15}$	$A_{14}$	$A_{13}$	$A_{12}$	$A_{11}$	$A_{10}$	$A_{09}$	$A_{08}$	$A_{07}$	$A_{06}$	$A_{05}$	$A_{04}$	$A_{03}$	$A_{02}$	$A_{01}$	$A_{00}$
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
EPROM								8K $\times$ 8												
FE000H	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
FDFFFH	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
RAM								8K $\times$ 8												
FC000H	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

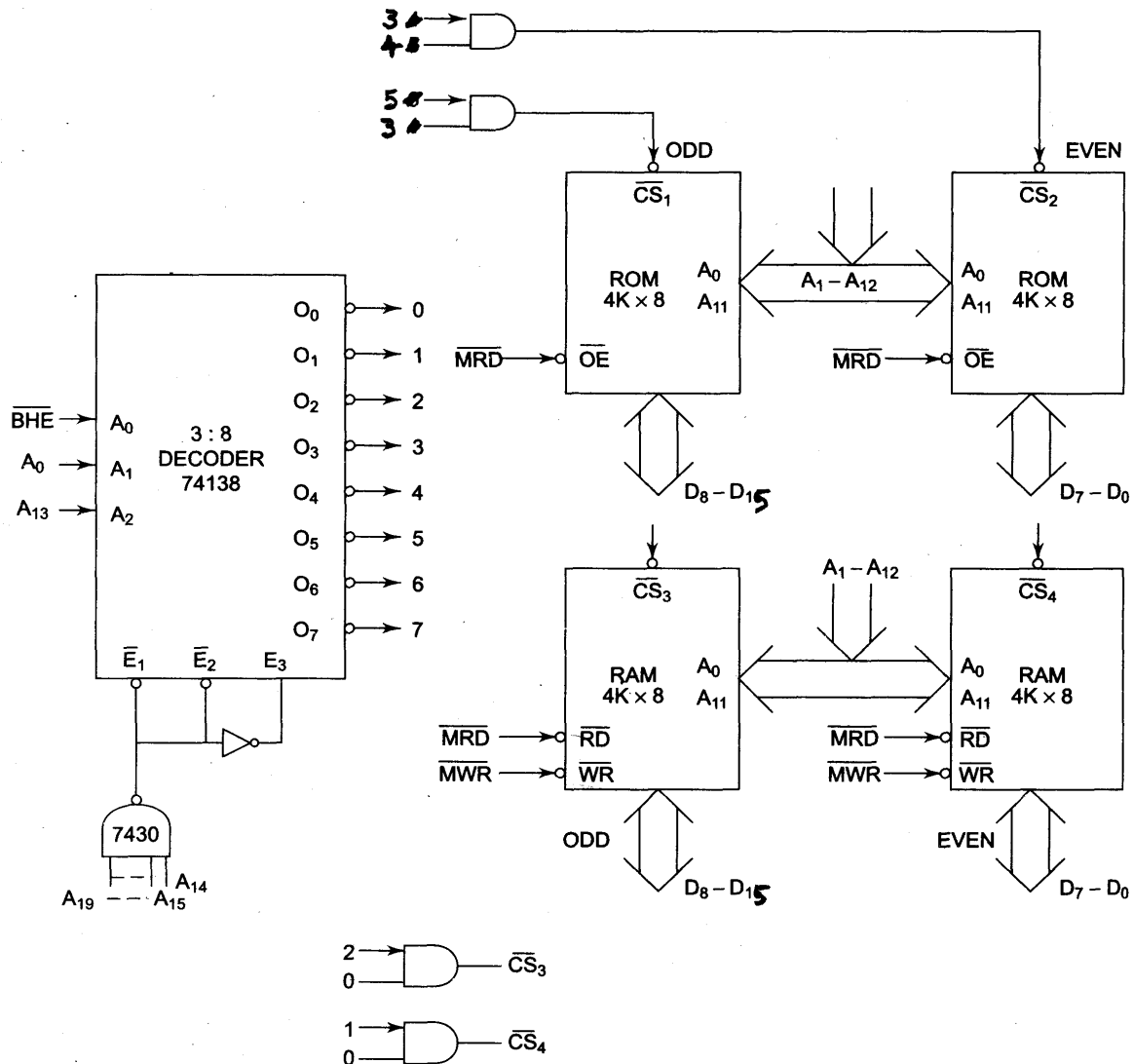
Total 8K bytes of EPROM need 13 address lines  $A_0 - A_{12}$  (since  $2^{13} = 8K$ ). Address lines  $A_{13} - A_{19}$  are used for decoding to generate the chip select. The  $\overline{BHE}$  signal goes low when a transfer is at odd address or higher byte of data is to be accessed. Let us assume that the latched address,  $\overline{BHE}$  and demultiplexed data lines are readily available for interfacing. Figure 5.1 shows the interfacing diagram for the memory system.

The memory system in this example contains in total four  $4K \times 8$  memory chips.

The two  $4K \times 8$  chips of RAM and ROM are arranged in parallel to obtain 16-bit data bus width. If  $A_0$  is 0, i.e. the address is even and is in RAM, then the lower RAM chip is selected indicating 8-bit transfer at an even address. If  $A_0$  is 1, i.e. the address is odd and is in RAM, the  $\overline{BHE}$  goes low, the upper RAM chip is selected, further indicating that the 8-bit transfer is at an odd address. If the selected addresses are in ROM, the respective ROM chips are selected. If at a time  $A_0$  and  $\overline{BHE}$  both are 0, both the RAM or ROM chips are selected, i.e. the data transfer is of 16 bits. The selection of chips here takes place as shown in Table 5.2.

**Table 5.2** Memory Chip Selection for Problem 5.1

Decoder I/P $\rightarrow$ Address/ $\overline{\text{BHE}}$ $\rightarrow$	$A_2$ $A_{13}$	$A_1$ $A_0$	$A_0$ $\overline{\text{BHE}}$	Selection/ Comment
Word transfer on $D_0 - D_{15}$	0	0	0	Even and odd addresses in RAM
Byte transfer on $D_7 - D_0$	0	0	1	Only even address in RAM
Byte transfer on $D_8 - D_{15}$	0	1	0	Only odd address in RAM
Word transfer on $D_0 - D_{15}$	1	0	0	Even and odd addresses in ROM
Byte transfer on $D_0 - D_7$	1	0	1	Only even address in ROM
Byte transfer on $D_8 - D_{15}$	1	1	0	Only odd address in ROM



**Fig. 5.1** Interfacing Problem 5.1

### Problem 5.2

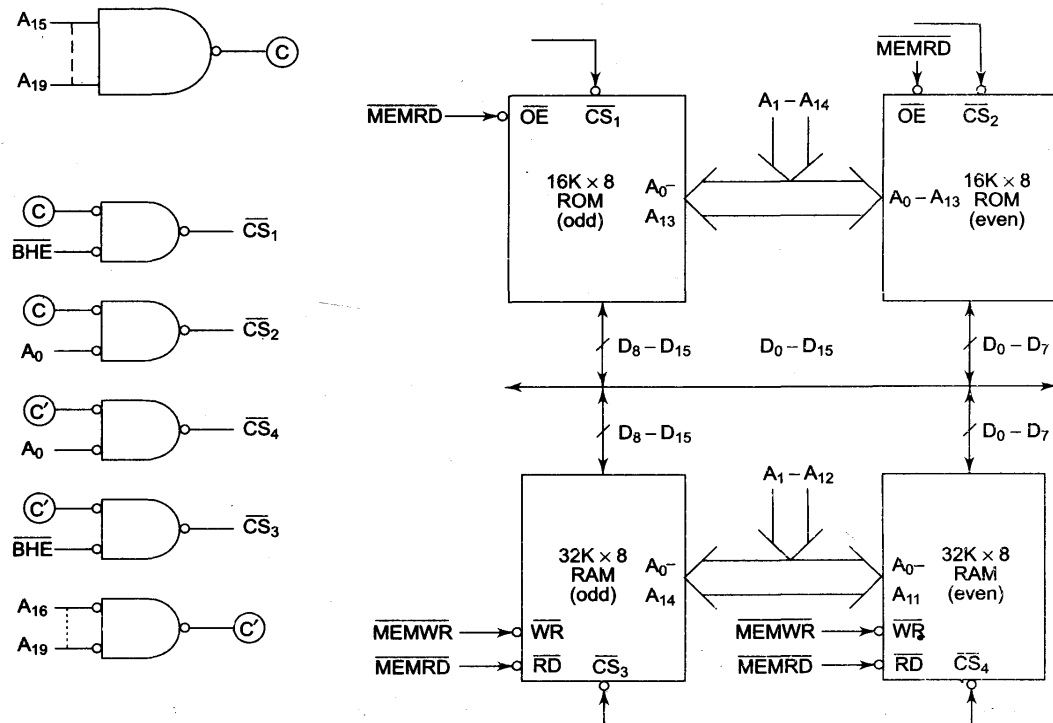
Design an interface between 8086 CPU and two chips of  $16K \times 8$  EPROM and two chips of  $32K \times 8$  RAM. Select the starting address of EPROM suitably. The RAM address must start at 00000H.

**Solution:** The last address in the map of 8086 is FFFFFH. After resetting, the processor starts from FFFF0H. Hence this address must lie in the address range of EPROM. Figure 5.2 shows the interfacing diagram, and Table 5.3 shows complete map of the system.

**Table 5.3** Address Map for Problem 5.2

Addresses	$A_{19}$	$A_{18}$	$A_{17}$	$A_{16}$	$A_{15}$	$A_{14}$	$A_{13}$	$A_{12}$	$A_{11}$	$A_{10}$	$A_{09}$	$A_{08}$	$A_{07}$	$A_{06}$	$A_{05}$	$A_{04}$	$A_{03}$	$A_{02}$	$A_{01}$	$A_{00}$
FFFFFH	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	32KB EPROM																			
F8000H	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	64KB RAM																			
00000H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

It is better not to use a decoder to implement the above map because it is not continuous, i.e. there is some unused address space between the last RAM address (0FFFFH) and the first EPROM address (F8000H). Hence the logic is implemented using logic gates, as shown in Fig. 5.2.



**Fig. 5.2** Interfacing Problem 5.2

### Problem 5.3

It is required to interface two chips of  $32K \times 8$  ROM and four chips of  $32K \times 8$  RAM with 8086, according to the following map.

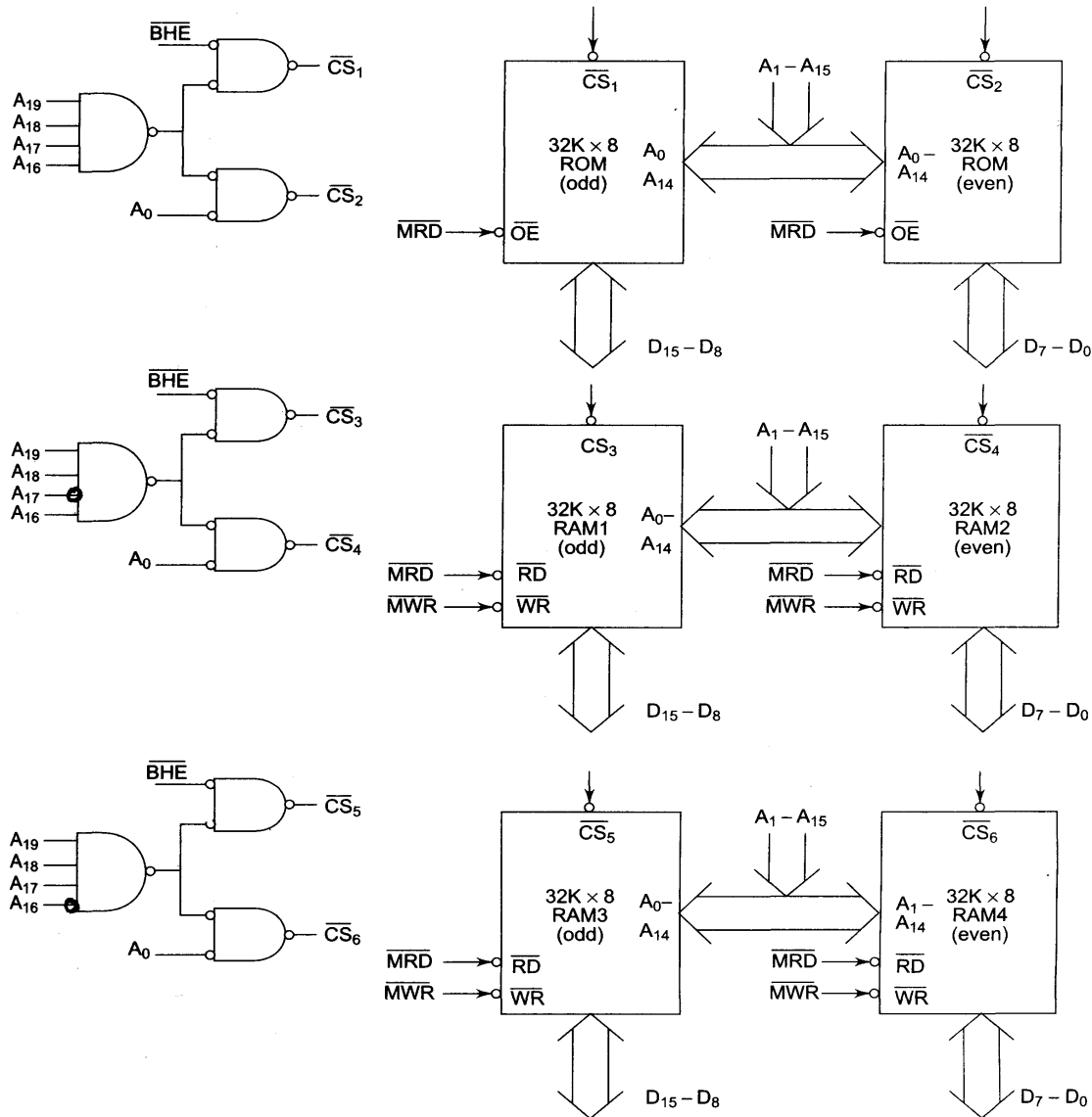
ROM 1 and 2 F0000H - FFFFFH, RAM 1 and 2 D0000H - DFFFFH

RAM 3 and 4 E0000H - EFFFFH

Show the implementation of this memory system.

**Solution** Let us write the memory map of the system as shown in Table 5.6.

The implementation of the above map is shown in Fig. 5.3 using the same technique as in Problem 5.1 and Problem 5.2. All the address, data and control signals are assumed to be readily available.



**Fig. 5.3** Interfacing Problem 5.3

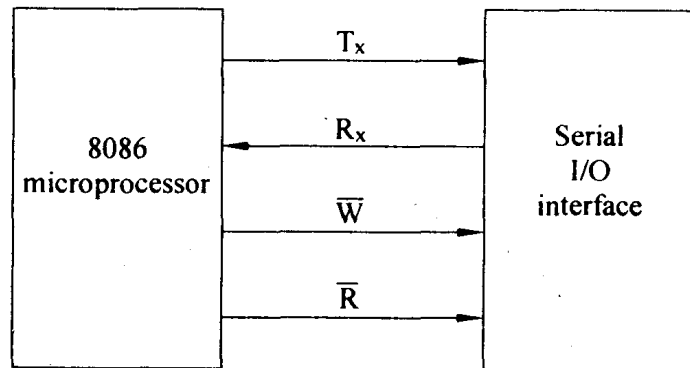
## SERIAL COMMUNICATION STANDARDS

Most of devices are parallel in nature. These devices transfer data simultaneously on data lines. But parallel data transfer process is very complicated and expensive. Hence in some situations the serial I/O mode is used where one bit is transferred over a single line at a time. In this type of transmission parallel word is converted into a stream of serial bits which is known as

parallel to serial conversion. The rate of transmission in serial mode is BAUD, i.e., bits per second. The serial data transmission involves starting, end of transmission, error verification bits along with the data. Any serial I/O involves the following concepts.

(a) Interfacing requirements (b) Alphanumeric codes (c) Transmission format (d) Error checks in data communication (e) Data communication over lines (f) Standards in serial I/O

The microprocessor has to identify the port address to perform read or write operation. Serial I/O uses only one data line, chip select, read, write control signals.



**Fig. 5.1** Block diagram of serial I/O interfacing

Data transfer takes place using ASCII code (American standard code for Information Interchange) which is 7 bit code with 128 combinations. The data can be transmitted by taking various parameters into consideration such as synchronization or synchronization, direction of data flow speed, errors, medium of data transmission etc. In synchronous transmission both transmitter and receiver operate, in synchronous to each other.

Synchronization used for high speed operations. In asynchronous data transmission data is transmitted between Start and Stop bits with logic 1 as mark logic 0 as space. In asynchronous we get around 11 bits for data transmission one start, 8 bits of data, 2 stop bits. A synchronous data transmission is used for less than 20 Kbits /second transmission.



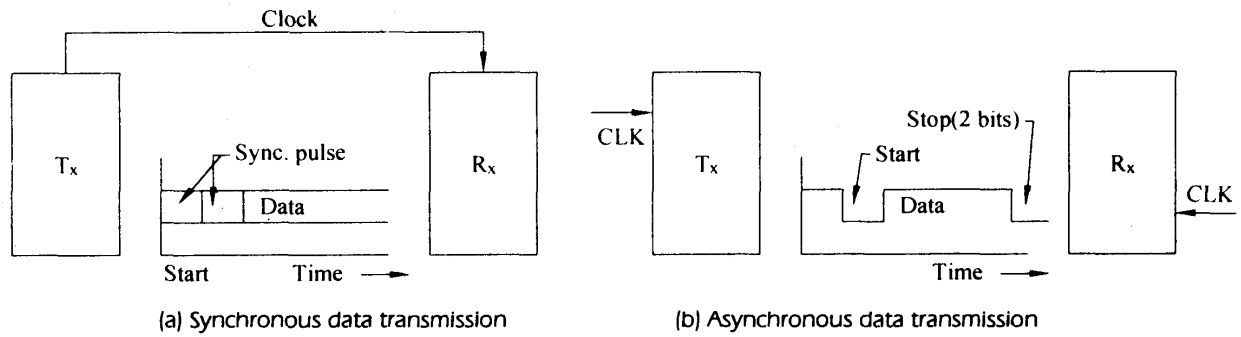


Fig. 5.2 Data communication

### DIFFERENCE BETWEEN SYNCHRONOUS AND ASYNCHRONOUS TRANSMISSION:

S.No.	Synchronous	Asynchronous
1.	Same clock pulse is applied to both Tx & Rx simultaneously	Different clock pulses are applied to Tx & Rx separately
2.	Only hardware is required to implement this.	Both hardware and software are required for this.
3.	Group or a set of characters can be transmitted at a time.	Only one character is transmitted at a time.
4.	Synchronous pulses are required	Synchronous pulses are not required but uses start and stop bits.
5.	Uses for high speed Tx.	Used for low speed Tx.

## **UNIT-IV**

### **SERIAL DATA TRANSFER SCHEMES**

#### **SERIAL COMMUNICATION**

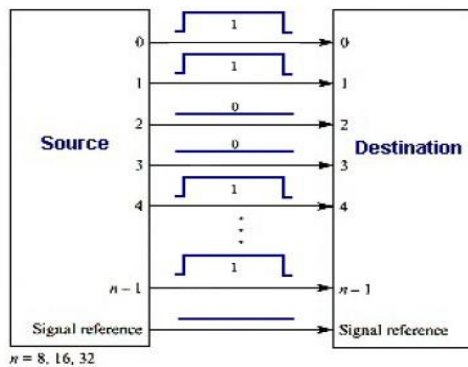
##### **INTRODUCTION**

Serial communication is common method of transmitting data between a computer and a peripheral device such as a programmable instrument or even another computer. Serial communication transmits data one bit at a time, sequentially, over a single communication line to a receiver. Serial is also a most popular communication protocol that is used by many devices for instrumentation. This method is used when data transfer rates are very low or the data must be transferred over long distances and also where the cost of cable and synchronization difficulties makes parallel communication impractical. Serial communication is popular because most computers have one or more serial ports, so no extra hardware is needed other than a cable to connect the instrument to the computer or two computers together.

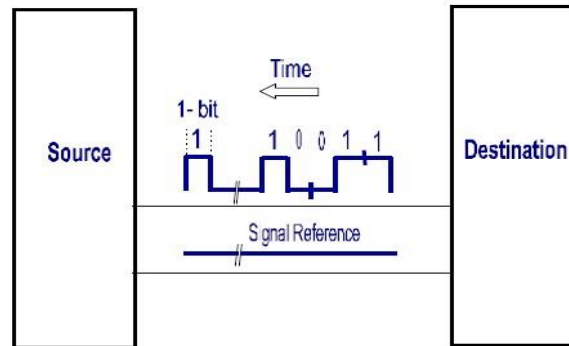
##### **SERIAL AND PARALLEL TRANSMISSION**

Let us now try to have a comparative study on parallel and serial communications to understand the differences and advantages & disadvantages of both in detail.

We know that parallel ports are typically used to connect a PC to a printer and are rarely used for other connections. A parallel port sends and receives data eight bits at a time over eight separate wires or lines. This allows data to be transferred very quickly. However, the setup looks more bulky because of the number of individual wires it must contain. But, in the case of a serial communication, as stated earlier, a serial port sends and receives data, one bit at a time over one wire. While it takes eight times as long to transfer each byte of data this way, only a few wires are required. Although this is slower than parallel communication, which allows the transmission of an entire byte at once, it is simpler and can be used over longer distances. So, at first sight it would seem that a serial link must be inferior to a parallel one, because it can transmit less data on each clock tick. However, it is often the case that, in modern technology, serial links can be clocked considerably faster than parallel links, and achieves a higher data rate.



**Parallel Transmission**



**Serial Transmission**

Even in shorter distance communications, serial computer buses are becoming more common because of a tipping point where the disadvantages of parallel busses (clock skew, interconnect density) outweigh their advantage of simplicity. The serial port on your PC is a full-duplex device meaning that it can send and receive data at the same time. In order to be able to do this, it uses separate lines for transmitting and receiving data.

From the above discussion we could understand that serial communications have many advantages over parallel one like:

- Requires fewer interconnecting cables and hence occupies less space.
- "Cross talk" is less of an issue, because there are fewer conductors compared to that of parallel communication cables.
- Many ICs and peripheral devices have serial interfaces.
- Clock skew between different channels is not an issue.
- Cheaper to implement.

#### **Clock skew:**

Clock skew is a phenomenon in synchronous circuits in which the clock signal sent from the clock circuit arrives at different components at different times, which can be caused by many things, like:

- Wire-interconnect length
- Temperature variations
- Variation in intermediate devices
- capacitive coupling
- Material imperfections

## **SERIAL DATA TRANSMISSION MODES**

When data is transmitted between two pieces of equipment, three communication modes of operation can be used.

**Simplex:** In a simple connection, data is transmitted in one direction only. For example, from a computer to printer that cannot send status signals back to the computer.

**Half-duplex:** In a half-duplex connection, two-way transfer of data is possible, but only in one direction at a time.

**Full duplex:** In a full-duplex configuration, both ends can send and receive data simultaneously, which technique is common in our PCs.

## **SERIAL DATA TRANSFER SCHEMS**

Like any data transfer methods, Serial Communication also requires coordination between the sender and receiver. For example, when to start the transmission and when to end it, when one particular bit or byte ends and another begins, when the receiver's capacity has been exceeded, and so on. Here comes the need for synchronization between the sender and the receiver. A protocol defines the specific methods of coordinating transmission between a sender and receiver. For example a serial data signal between two PCs must have individual bits and bytes that the receiving PC can distinguish. If it doesn't, then the receiving PC can't tell where one byte ends and the next one begin or where one bit ends and begins. So the signal must be synchronized in such a way that the receiver can distinguish the bits and bytes as the transmitter intends them to be distinguished.

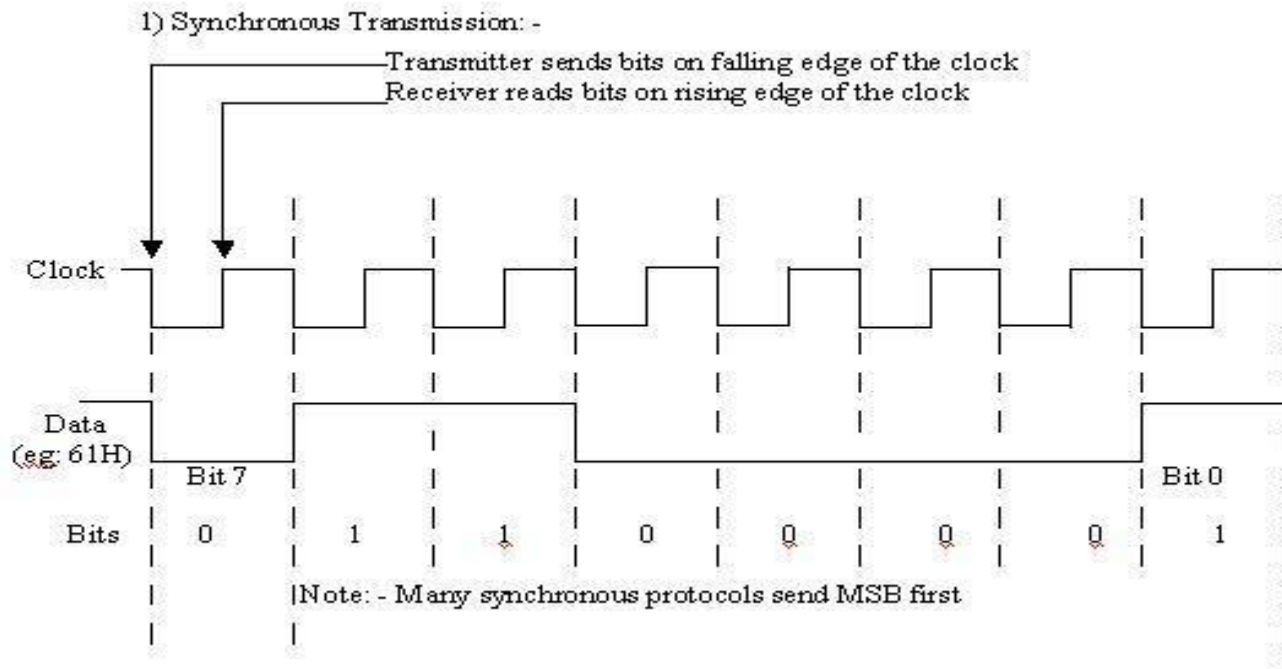
There are two ways to synchronize the two ends of the communication.

1. Synchronous data transmission
2. Asynchronous data transmission

### **Synchronous Data Transmission**

The synchronous signaling methods use two different signals. A pulse on one signal line indicates when another bit of information is ready on the other signal line.

In synchronous transmission, the stream of data to be transferred is encoded and sent on one line, and a periodic pulse of voltage which is often called the "clock" is put on another line, that tells the receiver about the beginning and the ending of each bit.



**Advantages:** The only advantage of synchronous data transfer is the Lower overhead and thus, greater throughput, compared to asynchronous one.

**Disadvantages:**

- Slightly more complex
- Hardware is more expensive

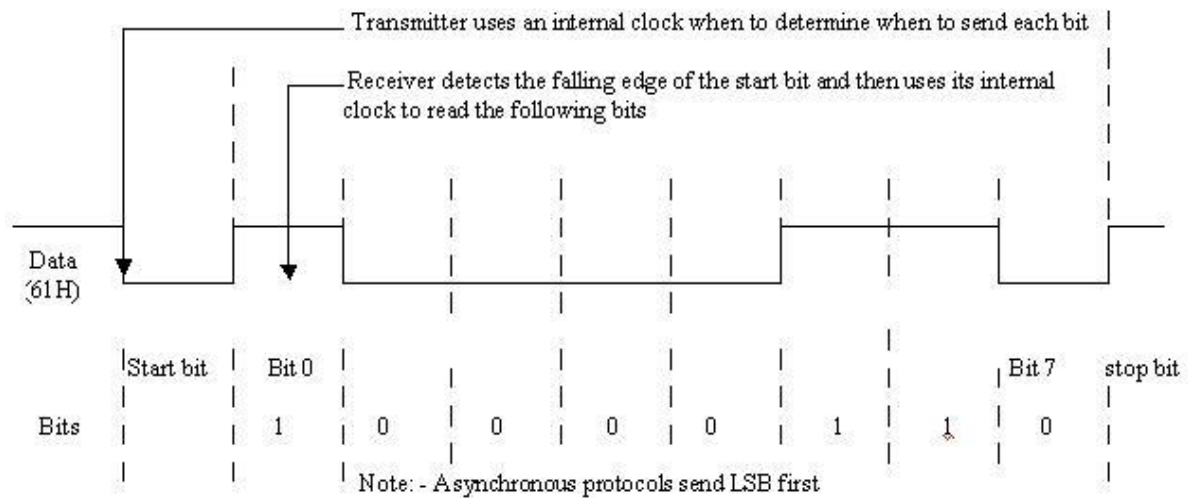
**Asynchronous data transmission**

The asynchronous signaling methods use only one signal. The receiver uses transitions on that signal to figure out the transmitter bit rate (known as auto baud) and timing. A pulse from the local clock indicates when another bit is ready. That means synchronous transmissions use an external clock, while asynchronous transmissions use special signals along the transmission medium. Asynchronous communication is the commonly prevailing communication method in the personal computer industry, due to the reason that it is easier to implement and has the unique advantage that bytes can be sent whenever they are ready,

a no need to wait for blocks of data to accumulate.

-

## 2) Asynchronous Transmission: -



### Advantages:

- Simple and doesn't require much synchronization on both communication sides.
- The timing is not as critical as for synchronous transmission; therefore hardware can be made cheaper.
- Set-up is very fast, so well suited for applications where messages are generated at irregular intervals, for example data entry from the keyboard.

### Disadvantages:

One of the main disadvantages of asynchronous technique is the large relative overhead, where a high proportion of the transmitted bits are uniquely for control purposes and thus carry no useful information.

## **8251A-PROGRAMMABLE COMMUNICATION INTERFACE** **(8251A-USART-Universal Synchronous/Asynchronous Receiver/Transmitter)\_**

### **INTRODUCTION**

A USART is also called a programmable communications interface (PCI). When information is to be sent by 8086 over long distances, it is economical to send it on a single line. The 8086 has to convert parallel data to serial data and then output it. Thus lot of microprocessor time is required for such a conversion.

Similarly, if 8086 receives serial data over long distances, the 8086 has to internally convert this into parallel data before processing it. Again, lot of time is required for such a conversion. The 8086 can delegate the job of conversion from serial to parallel and vice versa to the 8251A USART used in the system.

The Intel8251A is the industry standard Universal Synchronous/Asynchronous Receiver/Transmitter (USART), designed for data communications with Intel microprocessor families such as 8080, 85, 86 and

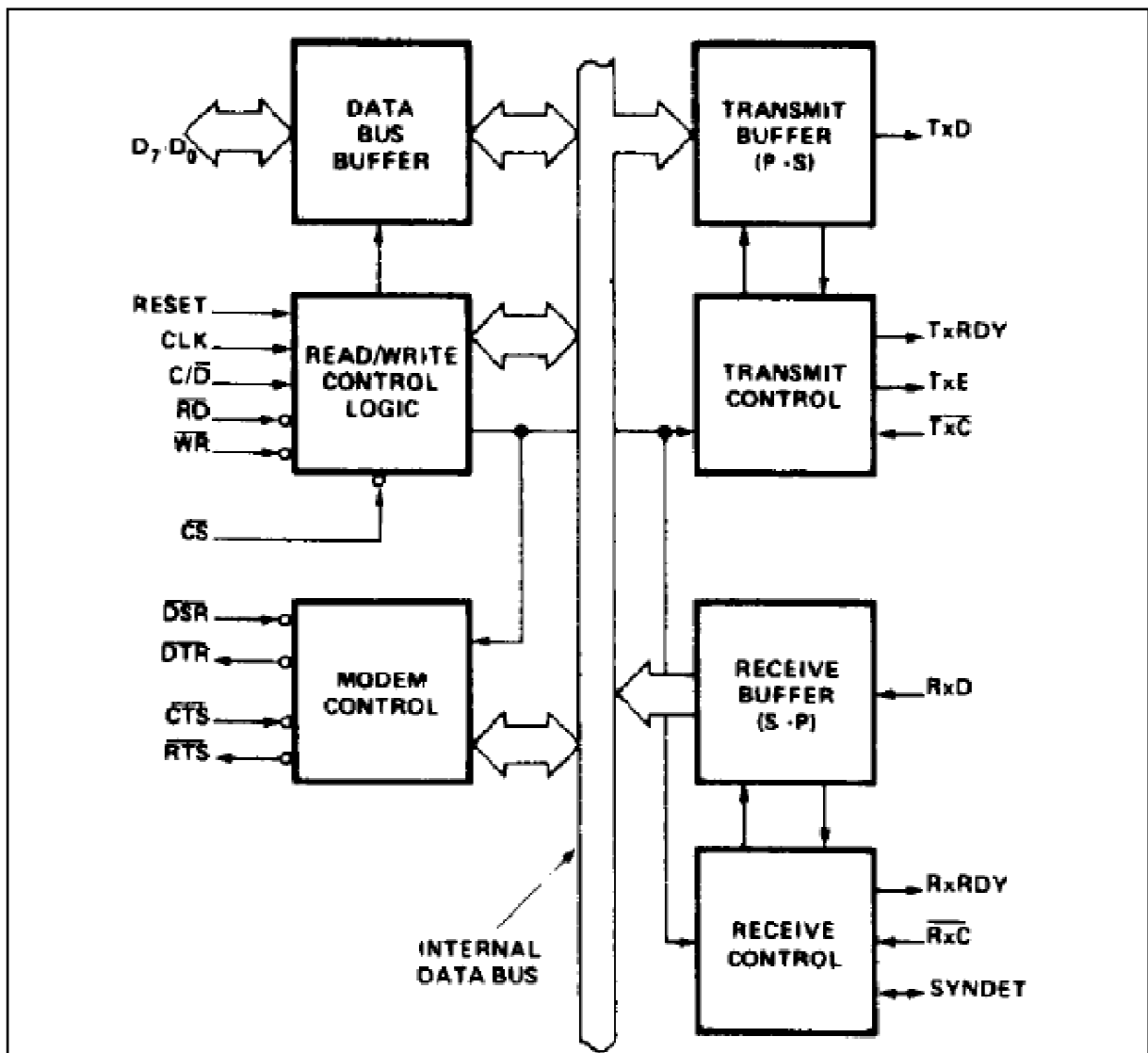
88. The 8251A converts the parallel data received from the processor on the D7-0 data pins into serial data, and transmits it on TxD (transmit data) output pin of 8251A. Similarly, it converts the serial data received on RxD (receive data) input into parallel data, and the processor reads it using the data pins D7-0.

### **FEATURES**

- Compatible with extended range of Intel microprocessors.
- It provides both synchronous and asynchronous data transmission.
- Synchronous 5-8 bit characters.
- Asynchronous 5-8 bit characters.
- It has full duplex, double buffered transmitter and receiver.
- Detects the errors-parity, overrun and framing errors.
- All inputs and outputs are TTL compatible.
- Available in 28-pin DIP package.

## ARCHITECTURE

The 8251A is a USART (Universal Synchronous Asynchronous Receiver Transmitter) for serial data communication. As a peripheral device of a microcomputer system, the 8251 receives parallel data from the CPU and transmits serial data after conversion. This device also receives serial data from the outside and transmits parallel data to the CPU after conversion. The internal block diagram of 8251A is shown in fig below.





**Data Bus Buffer:** This bidirectional, 8-bit buffer used to interface the 8251A to the system data bus and also used to read or write status, command word or data from or to the 8251A.

**Read/Write control logic:** The Read/Write Control logic interfaces the 8251A with microprocessor, determines the functions of the 8251A according to the control word written into its control register and monitors the data flow. This section has three registers and they are control register, status register and data buffer.

- The active low signals  $\overline{CS}$ ,  $\overline{W}$  and  $\overline{RD}$  are used for read/write operations with these three registers.
- When  $\overline{RD}$  is high, the control register is selected for writing control word or reading status word. When  $\overline{RD}$  is low, the data buffer is selected for read/write operation.
- When the reset is high, it forces 8251A into the idle mode.
- The clock input is necessary for 8251A for communication with microprocessor and this clock does not control either the serial transmission or the reception rate.

**Transmitter section:** The transmitter section accepts parallel data from microprocessor and converts them into serial data. The transmitter section is double buffered, i.e., it has a buffer register to hold an 8-bit parallel data and another register called output register to convert the parallel data into serial bits. When output register is empty, the data is transferred from buffer to output register. Now the processor can again load another data in buffer register.

- If buffer register is empty, then TxRDY is goes to high.
- If output register is empty then TxEMPTY goes to high.
- The clock signal  $\overline{CLK}$  controls the rate at which the bits are transmitted by the USART.
- The clock frequency can be 1,16 or 64 times the baud rate.

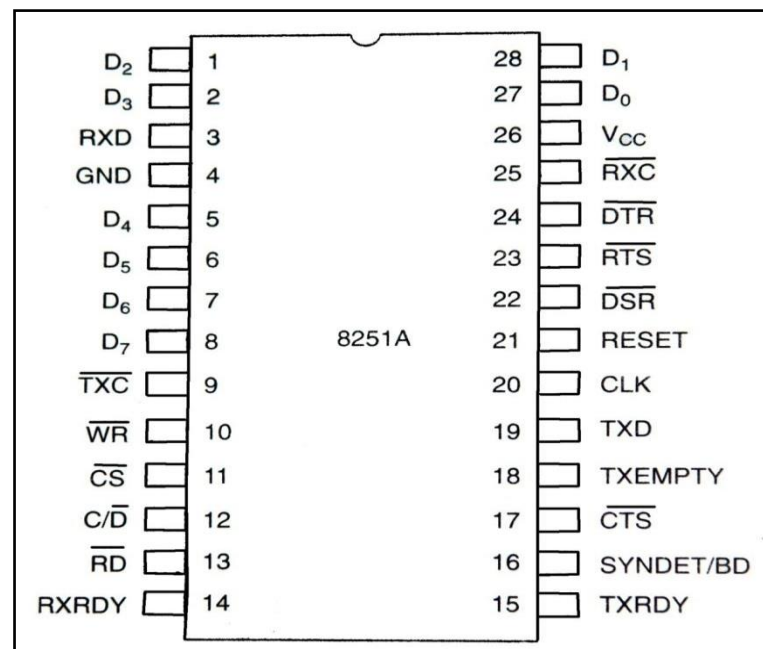
**Receiver Section:** The receiver section accepts serial data and converts them into parallel data. The receiver section is double buffered, i.e., it has an input register to receive serial data and convert to parallel, and a buffer register to hold the parallel data. When the RxD line goes low, the control logic assumes it as a START bit, waits for half a bit time and samples the line again. If the line is still low, then the input register accepts the following bits, forms a character and loads it into the buffer register. The microprocessor reads the parallel data from the buffer

register.

- When the input register loads a parallel data to buffer register, the RxRDY line goes high.
- The clock signal  $\phi$  controls the rate at which bits are received by the USART.
- During asynchronous mode, the signal SYNDET/BRKDET will indicate the break in the data transmission. During synchronous mode, the signal SYNDET/BRKDET will indicate the reception of synchronous character.

**MODEM Control:** The MODEM control unit allows to interface a MODEM to 8251A and to establish data communication through MODEM over telephone lines. This unit takes care of handshake signals for MODEM interface.

### PIN DIAGRAM



**D0 to D7 (I/O terminal):** This is bidirectional data bus which receives control words and transmits data from the CPU and sends status words and received data to CPU.

**RESET (Input terminal):** A "High" on this input forces the 8251 into "reset status." The device waits for the writing of "mode instruction." The min. reset width is six clock inputs during the operating status of CLK.

**CLK (Input terminal):** CLK signal is used to generate internal device timing. CLK signal is independent of RXC or TXC. However, the frequency of CLK must be greater than 30 times the RXC and TXC at Synchronous mode and Asynchronous "x1" mode, and must be greater than 5 times at Asynchronous "x16" and "x64" mode.

**$\overline{\text{WR}}$  (Input terminal):** This is the "active low" input terminal which receives a signal for writing transmit data and control words from the CPU into the 8251.

**$\overline{\text{RD}}$  (Input terminal):** This is the "active low" input terminal which receives a signal for reading receive data and status words from the 8251.

**C/ D (Input terminal):** This is an input terminal which receives a signal for selecting data or command words and status words when the 8251 is accessed by the CPU. If C/D = low, data will be accessed. If C/D = high, command word or status word will be accessed.

**$\overline{\text{CS}}$  (Input terminal):** This is the "active low" input terminal which selects the 8251 at low level when the CPU accesses. Note: The device won't be in "standby status"; only setting CS = High.

**TXD (output terminal):** This is an output terminal for transmitting data from which serial-converted data is sent out. The device is in "mark status" (high level) after resetting or during a status when transmit is disabled. It is also possible to set the device in "break status" (low level) by a command.

**TXRDY (output terminal):** This is an output terminal which indicates that the 8251 is ready to accept a transmitted data character. But the terminal is always at low level if CTS = high or the device was set in "TX disable status" by a command. Note: TXRDY status word indicates that transmit data character is receivable, regardless of CTS or command. If the CPU writes a data character, TXRDY will be reset by the leading edge of WR signal.

**TXEMPTY (Output terminal):** This is an output terminal which indicates that the 8251 has transmitted all the characters and had no data character. In "synchronous mode," the terminal is at high level, if transmit data characters are no longer remaining and sync characters are automatically transmitted. If the CPU writes a data character, TXEMPTY will be reset by the leading edge of WR signal. Note : As the transmitter is disabled by setting CTS "High" or command, data written before disable will be sent out. Then TXD and TXEMPTY will be

"High". Even if a data is written after disable, that data is not sent out and TXE will be "High". After the transmitter is enabled, it sent out. (Refer to Timing Chart of Transmitter Control and Flag Timing)

**$\overline{\text{TXC}}$  (Input terminal):** This is a clock input signal which determines the transfer speed of transmitted data. In "synchronous mode," the baud rate will be the same as the frequency of TXC. In "asynchronous mode", it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16 or 1/64 the TXC. The falling edge of TXC sifts the serial data out of the 8251.

**RXD (input terminal):** This is a terminal which receives serial data.

**RXRDY (Output terminal):** This is a terminal which indicates that the 8251 contains a character that is ready to READ. If the CPU reads a data character, RXRDY will be reset by the leading edge of RD signal. Unless the CPU reads a data character before the next one is received completely, the preceding data will be lost. In such a case, an overrun error flag status word will be set.

**RXC (Input terminal):** This is a clock input signal which determines the transfer speed of received data. In "synchronous mode," the baud rate is the same as the frequency of RXC. In "asynchronous mode," it is possible to select the baud rate factor by mode instruction. It can be 1, 1/16, 1/64 the RXC.

**SYNDET/BD (Input or output terminal):** This is a terminal whose function changes according to mode. In "internal synchronous mode." this terminal is at high level, if sync characters are received and synchronized. If a status word is read, the terminal will be reset. In "external synchronous mode, "this is an input terminal. A "High" on this input forces the 8251 to start receiving data characters.

In "asynchronous mode," this is an output terminal which generates "high level "output upon the detection of a "break" character if receiver data contains a "low-level" space between the stop bits of two continuous characters. The terminal will be reset, if RXD is at high level. After Reset is active, the terminal will be output at low level.

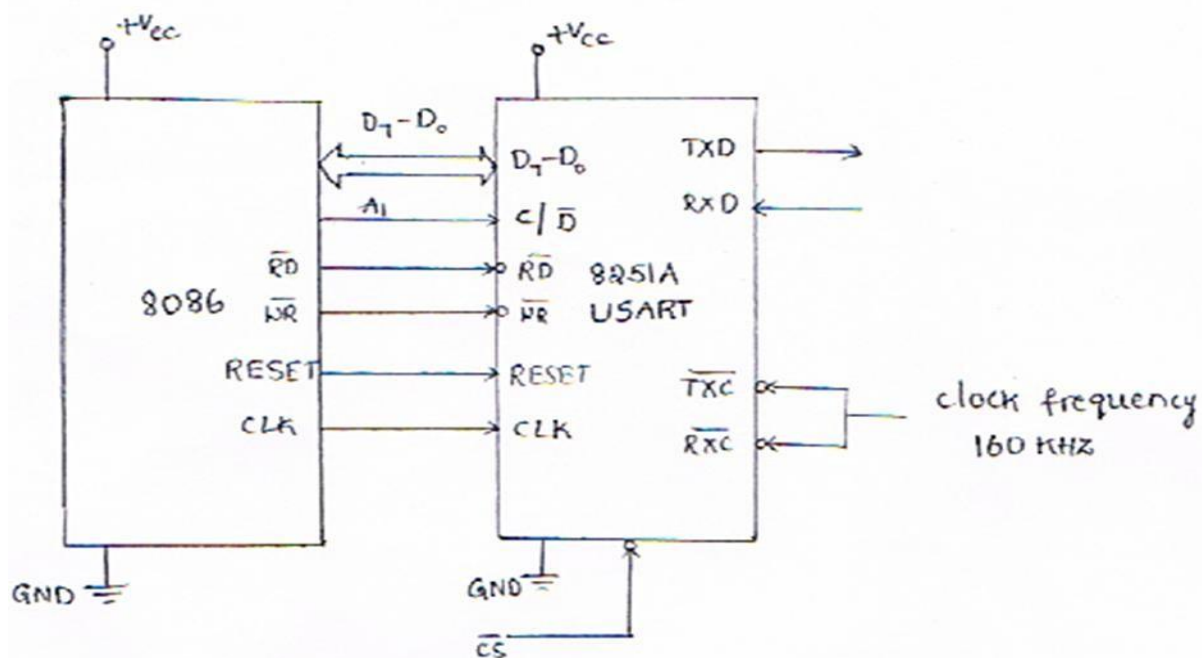
**$\overline{\text{DSR}}$  (Input terminal):** This is an input port for MODEM interface. The input status of the terminal can be recognized by the CPU reading status words.

**$\overline{\text{DTR}}$  (Output terminal):** This is an output port for MODEM interface. It is possible to set the status of DTR by a command.

**$\overline{\text{CTS}}$  (Input terminal):** This is an input terminal for MODEM interface which is used for controlling a transmit circuit. The terminal controls data transmission if the device is set in "TX Enable" status by a command. Data is transmittable if the terminal is at low level.

**$\overline{\text{RTS}}$  (Output terminal):** This is an output port for MODEM interface. It is possible to set the status RTS by a command.

### 8251A USART INTERFACING WITH 8086



### PROGRAMMING THE 8251A

Prior to starting a data transmission or reception, the 8251A must be loaded with a set of control words generated by the microprocessor. These control signals define the complete functional definition of the 8251A and must immediately follow a reset operation (internal or external). The control words are split into two formats.

#### 1. Mode instruction

## 2. Command instruction

**Mode instruction:** Mode instruction is used for setting the function of the 8251A. Mode instruction will be in "wait for write" at either internal reset or external reset. That is, the writing of a control word after resetting will be recognized as a "mode instruction."

Items set by mode instruction are as follows:

- Synchronous/asynchronous mode
- Stop bit length (asynchronous mode)
- Character length
- Parity bit
- Baud rate factor (asynchronous mode)
- Internal/external synchronization (synchronous mode)
- Number of synchronous characters (Synchronous mode)

The bit configuration of mode instruction format is shown in Figures below. In the case of synchronous mode, it is necessary to write one-or two byte sync characters. If sync characters were written, a function will be set because the writing of sync characters constitutes part of mode instruction.

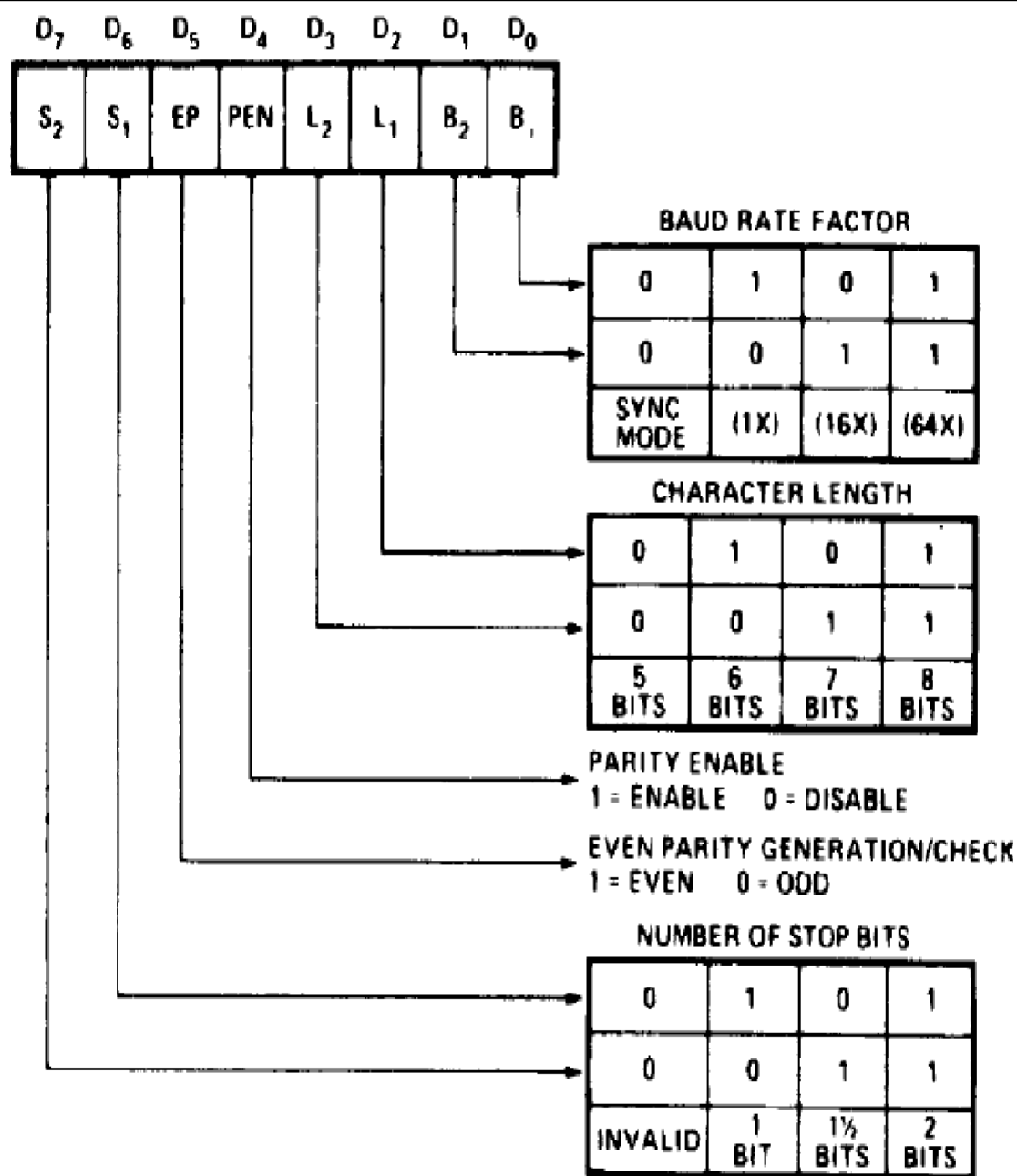
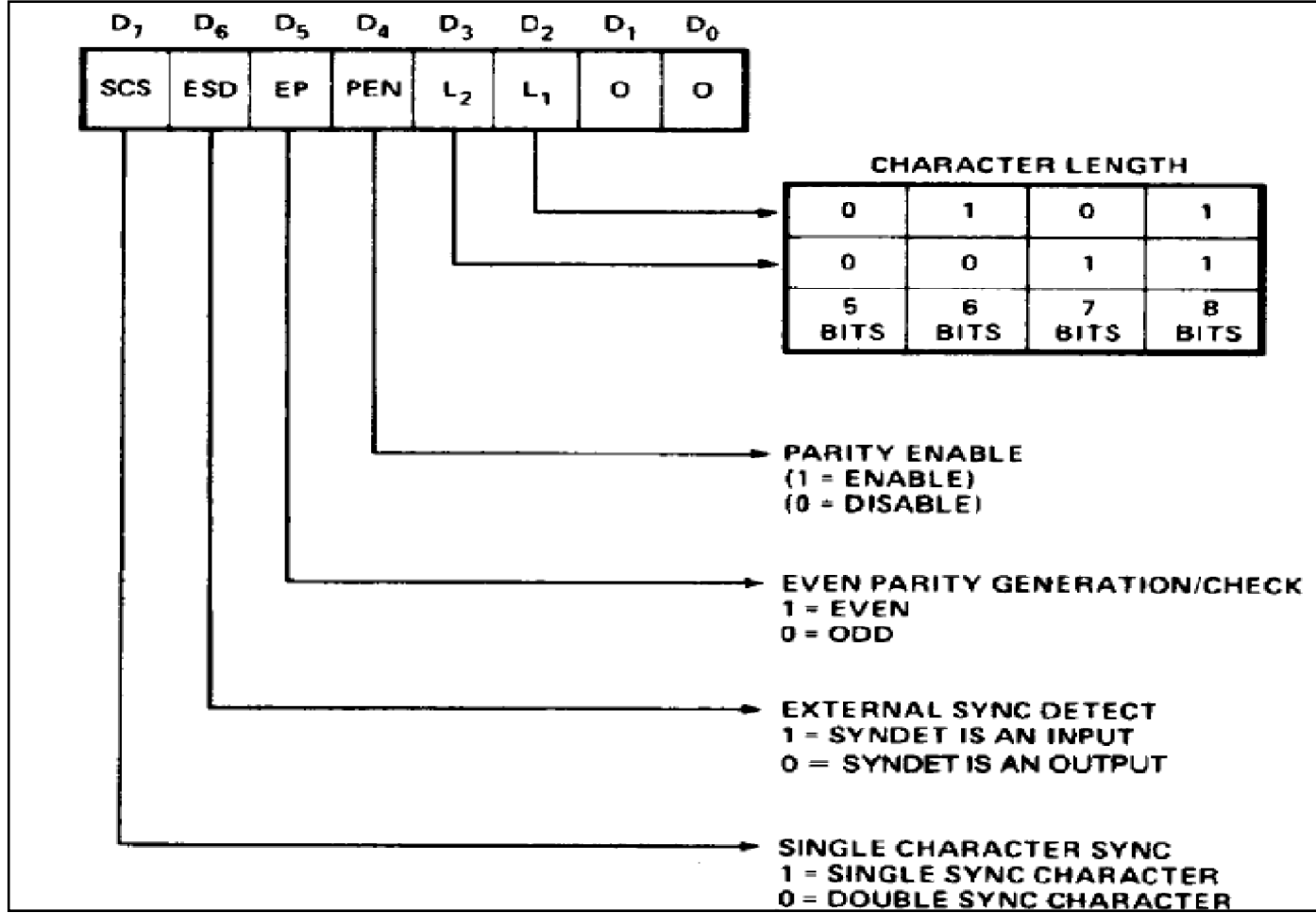


Fig. Mode instruction format, Asynchronous mode



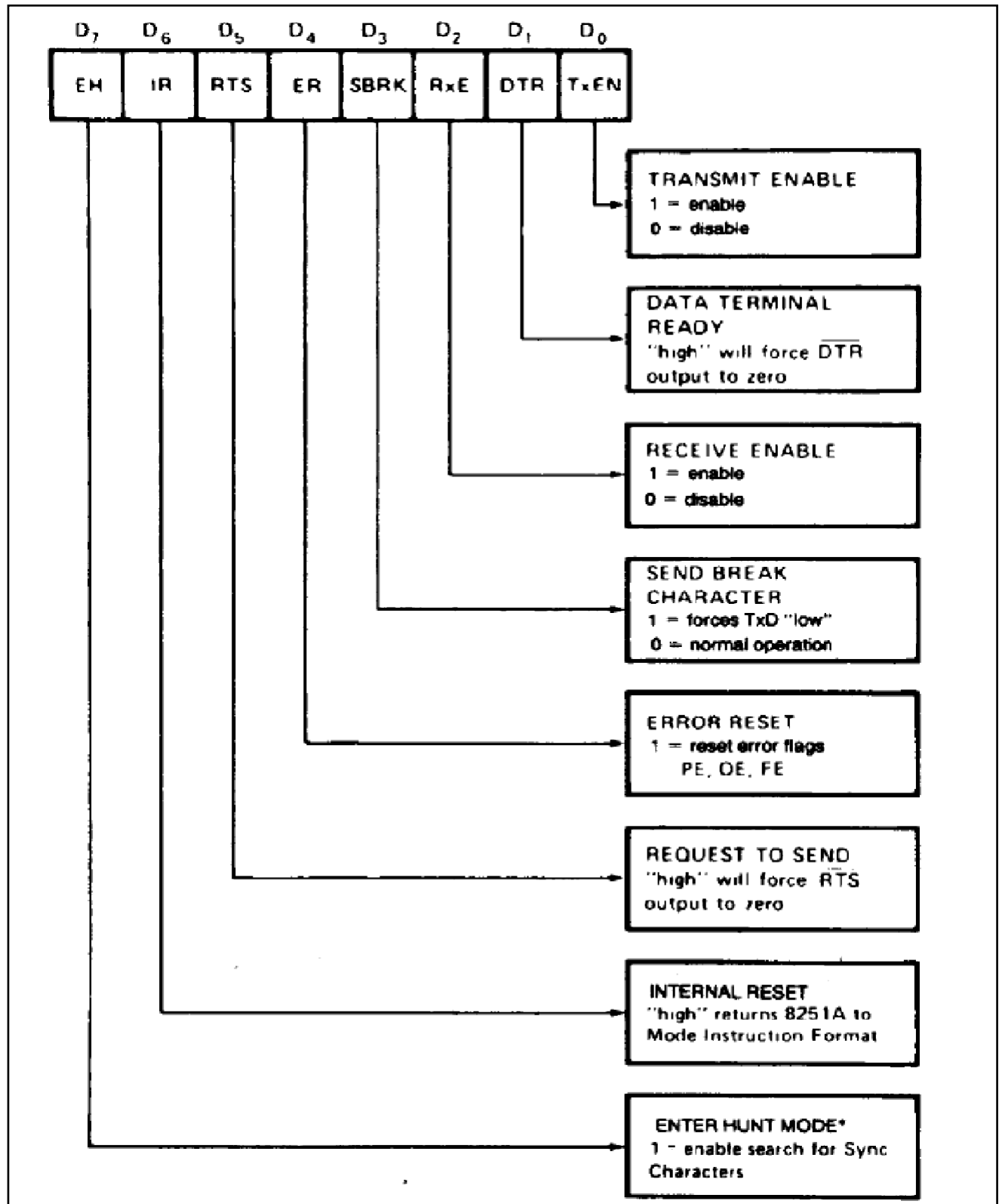
**Fig. Mode instruction format, Synchronous mode**

**Command Instruction:** Command is used for setting the operation of the 8251. It is possible to write a command whenever necessary after writing a mode instruction and sync characters.

Items to be set by command are as follows:

- Transmit Enable/Disable
- Receive Enable/Disable
- DTR, RTS Output of data.
- Resetting of error flag.
- Sending to break characters
- Internal resetting
- Hunt mode (synchronous mode)





**Status Word:** It is possible to see the internal status of the 8251 by reading a status word. The format of status word is shown below.

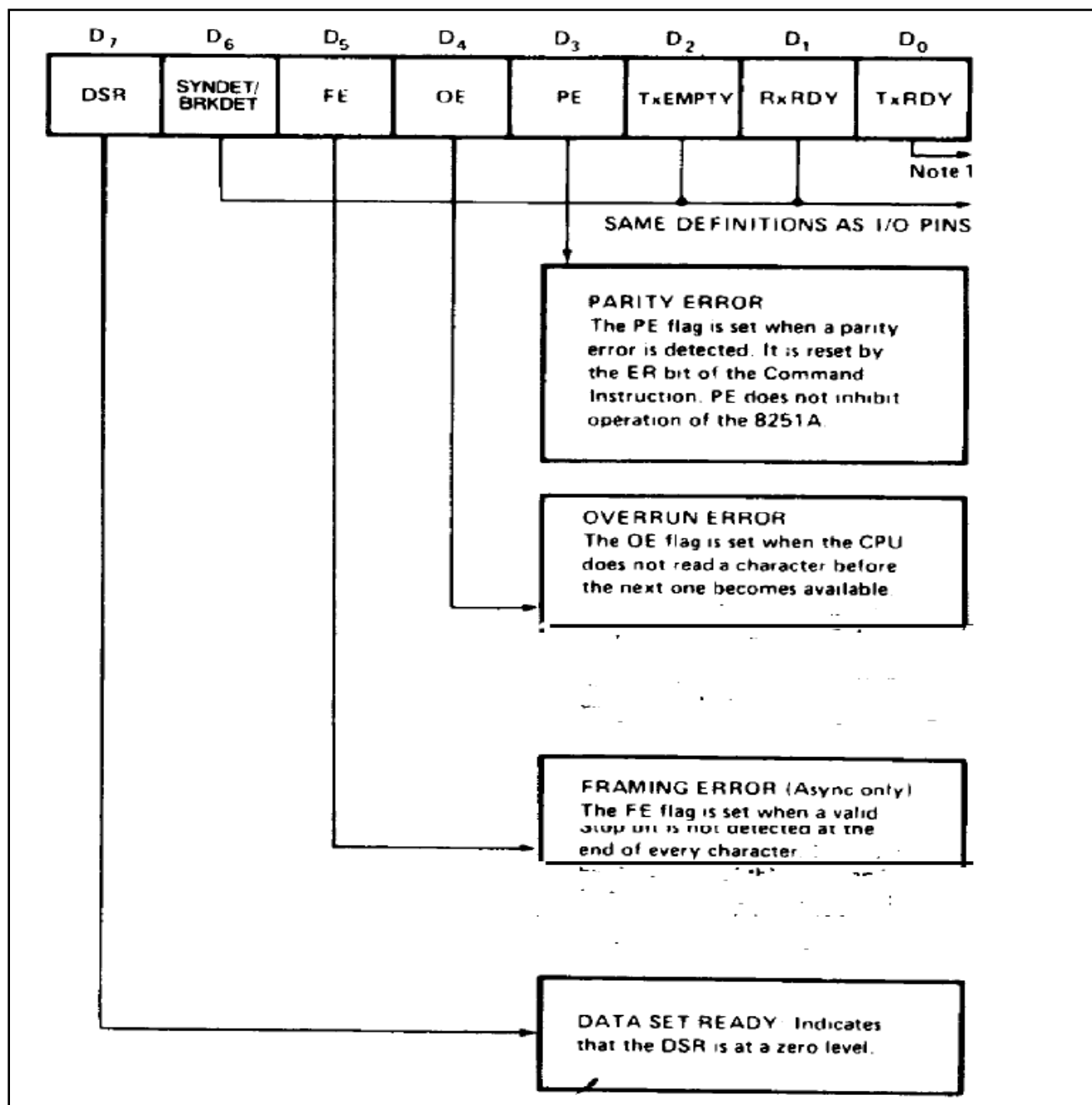


Fig. Status word

## **RECOMMENDED STANDARD -232C (RS-232C)**

RS-232 was first introduced in 1962 by the *Radio Sector* of the Electronic Industries Association ([EIA](#)). RS-232 (Recommended standard-232) is a standard interface approved by the Electronic Industries Association (EIA) for connecting serial devices. In other words, RS-232 is a long-established standard that describes the physical interface and protocol for relatively low-speed serial data communication between computers and related devices. An industry trade group, the Electronic Industries Association (EIA), defined it originally for teletypewriter devices. In 1987, the EIA released a new version of the standard and changed the name to EIA-232-D. Many people, however, still refer to the standard as RS-232C, or just RS-232. RS-232 is the interface that your computer uses to talk to and exchange data with your modem and other serial devices. The serial ports on most computers use a subset of the RS-232C standard.

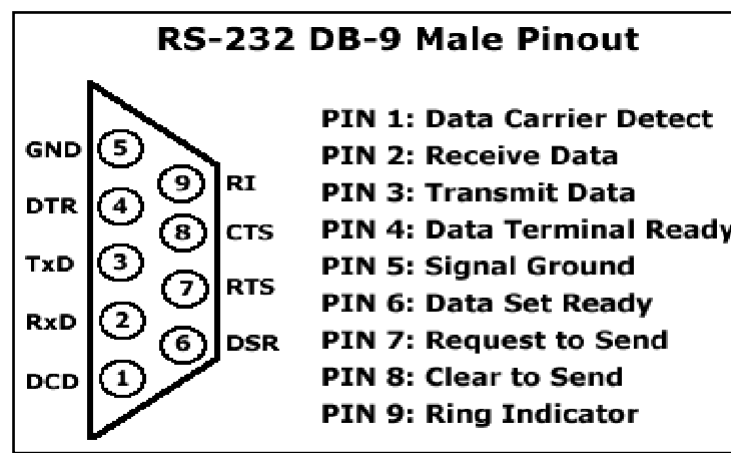
RS-232C is defined as the “Interface between data terminal equipment and data communications equipment using serial binary data exchange.” This definition defines data terminal equipment (DTE) as the computer, while data communications equipment (DCE) is the modem. A modem cable has pin-to-pin connections, and is designed to connect a DTE device to a DCE device. In addition to communications between computer equipment over telephone lines, RS-232C is now widely used for direct connections between data acquisition devices and computer systems. As in the definition of RS-232, the computer is data transmission equipment (DTE). RS-232C cables are commonly available with 4, 9 or 25-pin wiring. The 25-pin cable connects every pin; the 9-pin cables do not include many of the uncommonly used connections; 4-pin cables provide the bare minimum connections, and have jumpers to provide “handshaking” for those devices that require it.

An RS-232 serial port was once a standard feature of a personal computer, used for connections to modems, printers, mice, data storage, uninterruptible power supplies, and other peripheral devices. However, the low transmission speed, large voltage swing, and large standard connectors motivated development of the Universal Serial Bus, which has displaced RS-232 from most of its peripheral interface roles.

In RS-232, user data is sent as a time-series of bits. Both synchronous and asynchronous transmissions are supported by the standard. In addition to the data circuits, the standard defines a

number of control circuits used to manage the connection between the DTE and DCE. Each data or control circuit only operates in one direction, which is, signaling from a DTE to the attached DCE or the reverse. Since transmit data and receive data are separate circuits, the interface can operate in a full duplex manner, supporting concurrent data flow in both directions.

The RS-232 standard defines the voltage levels that correspond to logical one and logical zero levels for the data transmission and the control signal lines. Valid signals are either in the range of +3 to +15 volts for logic 0 or the range -3 to -15 volts for logic 1, the range between -3 to +3 volts is not a valid RS-232 level. For data transmission lines (TxD, RxD and their secondary channel equivalents) logic one is defined as a negative voltage, the signal condition is called "mark." Logic zero is positive and the signal condition is termed "space." The 9-pin RS-232C standard is shown in figure below.



### TTL TO RS-232C AND RS-232C TO TTL CONVERSION

The RS-232C standard does not define elements as the character encoding or the framing of characters, or error detection protocols. Details of character format and transmission bit rate are controlled by the serial port hardware, often a single integrated circuit called a USART that converts data from parallel to asynchronous start-stop serial form. Details of voltage levels, slew rate, and short-circuit behavior are typically controlled by a line driver (MC 1488) that converts from the USART's logic levels (TTL levels) to RS-232 compatible

signal levels, and a receiver (MC 1489) that converts RS-232 compatible signal levels to the USART's logic levels (TTL levels). The figure shows the conversion of TTL to RS-232C and Rs-232C to TTL levels.

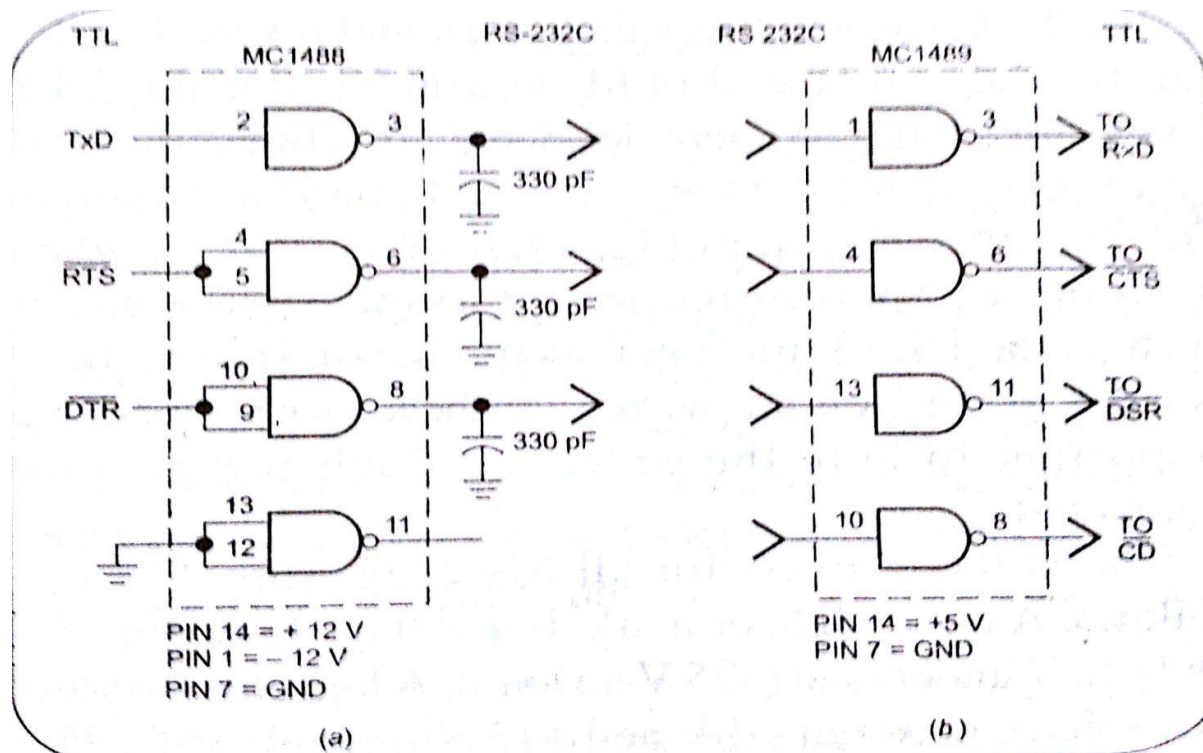


Fig. (a). TTL to Rs-232C and Fig. (b). RS-232C to TTL Conversion

## INTRODUCTION TO HIGH SPEED SERIAL COMMUNICATION STANDARDS

In telecommunication and computer science, serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus. This is in contrast to parallel communication, where several bits are sent as a whole, on a link with several parallel channels. Serial communication is used for all long-haul communication and most computer networks, where the cost of cable and synchronization difficulties make parallel communication impractical. Serial computer buses are becoming more common even at shorter distances, as improved signal integrity and transmission speeds in newer serial technologies have begun to outweigh the parallel bus's advantage of simplicity.

Examples of serial communication standards are:

- Morse code telegraphy

- RS-232 (low-speed, implemented by serial ports)
- RS-422
- RS-423
- RS-485
- I<sup>2</sup>C
- SPI
- ARINC 818 Avionics Digital Video Bus
- Atari SIO (Joe Decuir credits his work on Atari SIO as the basis of USB)
- Universal Serial Bus (moderate-speed, for connecting peripherals to computers)
- FireWire
- Ethernet
- Fibre Channel (high-speed, for connecting computers to mass storage devices)
- InfiniBand (very high speed, broadly comparable in scope to PCI)
- MIDI control of electronic musical instruments
- DMX512 control of theatrical lighting
- SDI-12 industrial sensor protocol
- CoaXPress industrial camera protocol over Coax
- Serial Attached SCSI
- Serial ATA
- Space Wire Spacecraft communication network
- Hyper Transport
- PCI Express
- SONET and SDH (high speed telecommunication over optical fibers)
- T-1, E-1 and variants (high speed telecommunication over copper pairs)
- MIL-STD-1553A/B

## **UNIVERSAL SERIAL BUS**

### **INTRODUCTION**

Universal Serial Bus (USB) is an industry standard developed in the mid-1990s that defines the

cables, connectors and communications protocols used in a bus for connection, communication and power supply between computers and electronic devices.

USB was designed to standardize the connection of computer peripherals (including keyboards, pointing devices, digital cameras, printers, portable media players, disk drives and network adapters) to personal computers, both to communicate and to supply electric power. It has become commonplace on other devices, such as smart phones, PDAs and video game consoles. USB has effectively replaced a variety of earlier interfaces, such as serial and parallel ports, as well as separate power chargers for portable devices.

As of 2008, approximately 6 billion USB ports and interfaces were in the global marketplace, and about 2 billion were being sold each year. The icon and cable of USB is shown in fig below.



## **HISTORY**

A group of seven companies began the development of USB in 1994: Compaq, DEC, IBM, Intel, Microsoft, NEC and Nortel. The goal was to make it fundamentally easier to connect external devices.

## **HISTORY**

A group of seven companies began the development of USB in 1994: Compaq, DEC, IBM, Intel, Microsoft, NEC and Nortel. The goal was to make it fundamentally easier to connect external devices to PCs by replacing the multitude of connectors at the back of PCs, addressing the usability issues of existing interfaces, and simplifying software configuration of all devices connected to USB, as well as permitting greater data rates for external devices. A team including Ajay Bhatt worked on the standard at Intel. The first

integrated circuits supporting USB were produced by Intel in 1995.

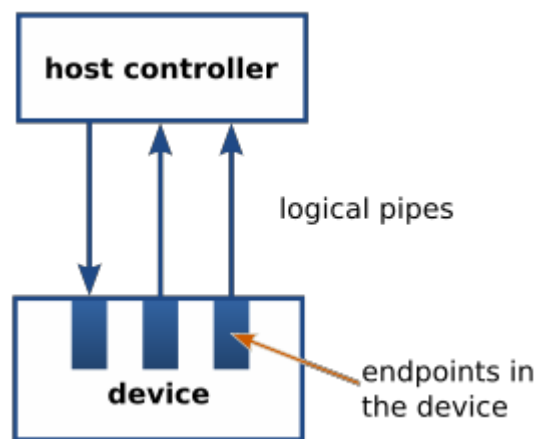
## **VERSIONS**

**USB 1 (Full Speed):** Released in January 1996, USB 1 specified data rates of 1.5 Mbit/s (Low-Bandwidth) and 12 Mbit/s (Full-Bandwidth).

**USB 2.0 (High Speed):** USB 2.0: Released in April 2000. Added higher maximum signaling rate of 480 Mbit/s (effective throughput up to 35 MB/s or 280 Mbit/s) (now called "Hi-Speed").

**USB 3.0 (Super Speed):** USB 3.0 was released in November 2008. The standard claims a theoretical "maximum" transmission speed of up to 5 Gbit/s (625 MB/s). USB 3.0 reduces the time required for data transmission, reduces power consumption, and is backward compatible with USB 2.0.

## **SYSTEM DESIGN**



The design architecture of USB is asymmetrical in its topology, consisting of a host, a multitude of downstream USB ports, and multiple peripheral devices connected in a tiered-star topology. Additional USB hubs may be included in the tiers, allowing branching into a tree structure with up to five tier levels. A USB host may implement multiple host controllers and each host controller may provide one or more USB ports. Up to 127 devices, including hub devices if present may be connected to a single host controller. USB devices are linked in series through hubs. One hub—built into the host controller—is the root hub. USB device communication is based on *pipes* (logical channels). A pipe is a connection from the host



controller to a logical entity, found on a device, and named an *endpoint*. Because pipes correspond 1- to-1 to endpoints, the terms are sometimes used interchangeably. A USB device can have up to 32 endpoints, though USB devices seldom have this many endpoints. An endpoint is built into the USB device by the manufacturer and therefore exists permanently, while a pipe may be opened and closed.

## **INTERFACING STANDARDS**

Serial I/O is used to interface various devices or for connecting various equipment to the system. Common understanding is necessary among various manufacturers such that a standard notation is followed for interfacing these components. These standards may be provided by IEEE or by any standard professional organization. The serial I/O standards must specify clearly voltage levels, speed of data transfer, length of cables etc. In serial I/O data can be transmitted as either current or voltage 20 mA or 60 mA current loops are used if data is transmitted using current. Current flow takes place when the system is at logic 1. The current flow is stopped when the system is at logic 0. In the current loop method the signals are relatively noise-free and they are best suited for long distance transmission.

RS-232 is developed long before which is used for communication between terminals and modems. Using RS-232C data can be transmitted as voltage. The data terminals equipment and data communication equipment are used to communicate using RS-232C cable. RS-232C is not compatible with TTL logic and cannot be used for long distance transmission.

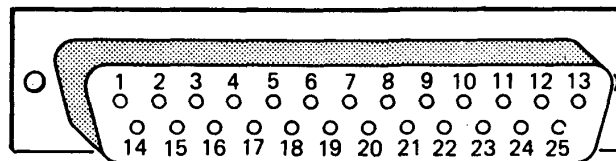
### **RS-232C Serial Data Standard**

#### **OVERVIEW:**

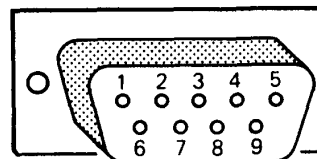
Modems were developed so that terminals could use phone lines to communicate with distant computers. As we stated earlier, modems and other devices used to send serial data are often referred to as *data communication equipment* or DCE. The terminals or computers that are sending or receiving the data are referred to as *data terminal equipment* or DTE. In response to the need for signal and handshake standards between DTE and DCE, the Electronic Industries Association (EIA) developed EIA standard RS-232C. This standard describes the function of 25 signal and handshake pins for serial-data transfer. It also describes the voltage levels, impedance

levels, rise and fall times, maximum bit rate, and maximum capacitance for these signal lines.

RS-232C specifies 25 signal pins, and it specifies that the DTE connector should be a male and the DCE connector should be a female. A specific connector is not given, but the most commonly used connectors are the DB-25P male shown in Figure 14-7a. For systems where many of the 25 pins are not needed, a 9-pin DIN connector such as the DE-9P male connector shown in Figure 14-7b is used.



(a)



(b)

**FIGURE 14-7** Connectors often used for RS-232C connections. (a) DB-25P 25-pin male. (b) DE-9P 9-pin male DIN connector.

The voltage levels for all RS-232C signals are as follows. A logic high, or mark, is a voltage between -3V and -15 V under load (-25 V no load). A logic low or space is a voltage between +3 V and +15 V under load (+ 25 V no load). Voltages such as  $\pm 12$  V are commonly used.

#### **RS-232C to TTL INTERFACING:**

Obviously a USART such as the 8251A is not directly compatible with RS-232C signal levels. The standard way to interface between RS-232C and TTL levels is with MCI488 quad TTL-to-RS-232C drivers and MCI489 quad RS-232C-to-TTL receivers shown in Figure 14-8.

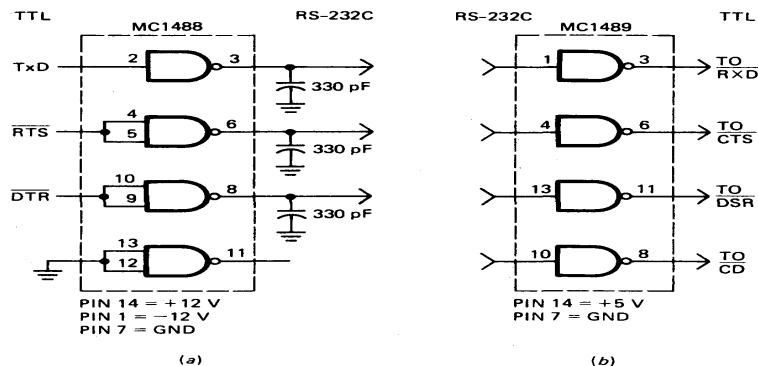


FIGURE 14-8 TTL to RS-232C to TTL signal conversion. (a) MC1488 used to convert TTL to RS-232C. (b) MC1489 used to convert RS-232C to TTL.

The MCI488s require + and - supplies, but the MCI489s require only + 5 V. Note the capacitor to ground on the outputs of the MCI488 drivers is to reduce cross talk between adjacent wires, the rise and fall times for RS-232C signals are limited to 30 V/ $\mu$ s.

## RS-232C SIGNAL DEFINITIONS

PIN NUMBERS FOR 9 PINS	PIN NUMBERS FOR 25 PINS	COMMON NAME	RS-232C NAME	DESCRIPTION	SIGNAL DIRECTION ON DCE
3	1		AA	PROTECTIVE GROUND	-
2	2	TxD	BA	TRANSMITTED DATA	IN
7	3	RxD	BB	RECEIVED DATA	OUT
8	4	RTS	CA	REQUEST TO SEND	IN
	5	CTS	CB	CLEAR TO SEND	OUT
6	6	DSR	CC	DATA SET READY	OUT
5	7	GND	AB	SIGNAL GROUND (COMMON RETURN)	-
1	8	CD	CF	RECEIVED LINE SIGNAL DETECTOR (RESERVED FOR DATA SET TESTING)	OUT
	9		-	(RESERVED FOR DATA SET TESTING)	-
	10		-		-
	11		SCF	UNASSIGNED	-
	12		SCB	SECONDARY RECEIVED LINE SIGNAL DETECTOR	OUT
	13		SBA	SECONDARY CLEAR TO SEND	OUT
	14		DB	SECONDARY TRANSMITTED DATA	IN
	15			TRANSMISSION SIGNAL ELEMENT TIMING (DCE SOURCE)	OUT
	16		SBB	SECONDARY RECEIVED DATA	OUT
	17		DD	RECEIVER SIGNAL ELEMENT TIMING (DCE SOURCE)	OUT
	18			UNASSIGNED	-
4	19	DTR	SCA	SECONDARY REQUEST TO SEND	IN
	20		CD	DATA TERMINAL READY	IN
9	21		CG	SIGNAL QUALITY DETECTOR	OUT
	22		CE	RING INDICATOR	OUT
	23		CH/CI	DATA SIGNAL RATE SELECTOR (DTE/DCE SOURCE)	IN/OUT
	24		DA	TRANSMIT SIGNAL ELEMENT TIMING (DTE SOURCE)	IN
	25			UNASSIGNED	-

FIGURE 14-9 RS-232C pin names and signal directions.

Figure 14-9 shows the signal names, signal direction, and a brief description for each of the 25 pins denned for RS-232C. For most applications only a few of these pins are used. Note that the signal direction is specified with respect to the DGE, this convention is part of the standard. Note that there is both a chassis ground (pin 1) and a signal ground (pin 7). To prevent large ac-induced ground currents in the signal ground, these two should be connected together

only at the power supply in the terminal or the computer.

The TxD, RxD, and handshake signals shown with common names in Figure 14-9 are the ones most often used for simple systems. These signals control what is called the *primary or forward* communications channel of the modem. Some modems allow communication over a secondary or *backward* channel, which operates in the reverse direction from the forward channel and at a much lower baud rate. Pins 12, 13, 14, 16, and 19 are the data and handshake lines for this backward channel. Pins 15, 17, 21, and 24 are used for synchronous data communication.

## **UNIT-V**

### **ADVANCED MICROPROCESSORS**

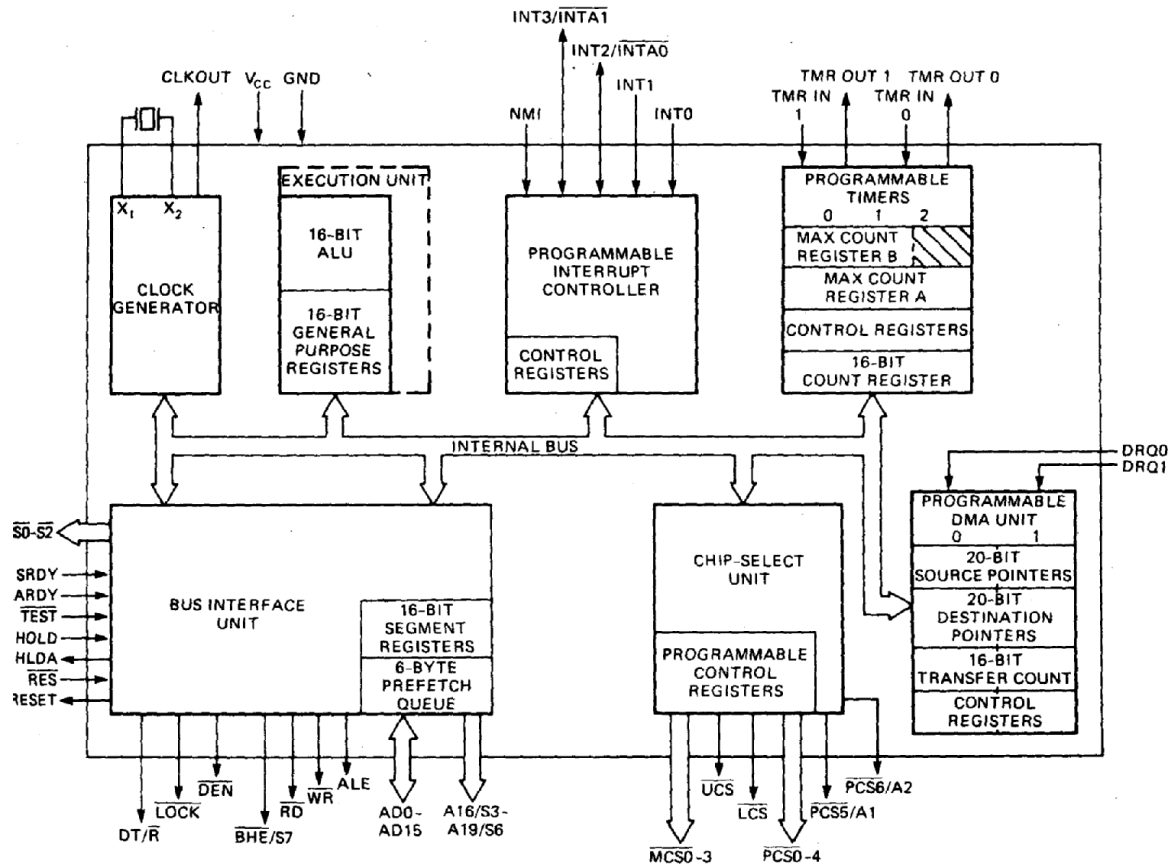
#### **INTEL 80186**

80186 contain a 16-bit data bus. The internal register structure of the 80186 is virtually identical to the 8086. The only difference is that the 80186 contain additional reserved interrupt vectors and some very powerful built in I/O features. The 80186 and 80188 are often called embedded controllers because of their application as a controller, not as a microprocessor-based computer.

#### **Features**

- The 80186 contains 16 bit data bus.
- The internal register structure of 80186 is virtually identical to the 8086.
- Enhanced 8086-2 CPU.
- Clock Generator.
- 2 Independent DMA Channels.
- Programmable Interrupt Controller.
- 3 Programmable 16-bit Timers.
- Programmable Memory and Peripheral Chip-Select Logic.
- Available in 10 MHz and 8 MHz Versions.
- Direct Addressing Capability to 1 Mbyte of Memory and 64 Kbyte I/O.

# ARCHITECTURE



In addition to the BIU and EU, the 80186 family contains a clock generator, a programmable interrupt controller, programmable timers, a programmable DMA controller and a programmable chip selection unit. These enhancements greatly increase the utility of the 80186 and reduce the number of peripheral components required to implement a system.

**Clock Generator:** The internal clock generator replaces the external 8284A clock generator used with the 8086/8088 microprocessors. This reduces the component count in a system. The internal clock generator has three pin connections: X1, X2, and CLKOUT. The X1 (CLKIN) and X2 (OSCOUT) pins are connected to a crystal that resonates at twice the operating frequency of the microprocessor. The CLKOUT pin provides a system clock signal that is one half the crystal frequencies. The CLKOUT pin drives other devices in a system and provides a timing source to additional microprocessors in the system.

**Programmable Interrupt Controller:** The programmable interrupt controller (PIC) arbitrates the internal and external interrupts and controls up to two external 8259A PICs. When an external 8259 is attached, the 80186 microprocessors function as the master and the 8259 functions as the slave. If the PIC is operated without the external 8259, it has five interrupt inputs: INT0–INT3 and NMI.

**Timers.** The timer section contains three fully programmable 16-bit timers. Timers 0 and 1 generate waveforms for external use and are driven by either the master clock of the 80186 or by an external clock. They are also used to count external events. The third timer, timer 2, is internal and clocked by the master clock. The output of timer 2 generates an interrupt after a specified number of clocks and can provide a clock to the other timers.

**Programmable DMA Unit:** The programmable DMA unit contains two DMA channels or four DMA channels in some models. Each channel can transfer data between memory locations, between memory and I/O, or between I/O devices.

**Programmable Chip Selection Unit:** The chip selection is a built-in programmable memory and I/O decoder. It has six output lines to select memory, seven lines to select I/O on some models, and 10 lines that select either memory or I/O on the some other models.

**Power Save/Power down Feature.** The power save feature allows the system clock to be divided by 4, 8, or 16 to reduce power consumption. The power-saving feature is started by software and exited by a hardware event such as an interrupt. The power down feature stops the clock completely. The power down mode is entered by execution of an HLT instruction and is

exited by any interrupt.

**Refresh Control Unit.** The refresh control unit generates the refresh row address at the interval programmed. The refresh control unit does not multiplex the address for the DRAM—this is still the responsibility of the system designer. The refresh address is provided to the memory system at the end of the programmed refresh interval, along with the control signal. The memory system must run a refresh cycle during the active time of the control signal.

## **INTEL 80286**

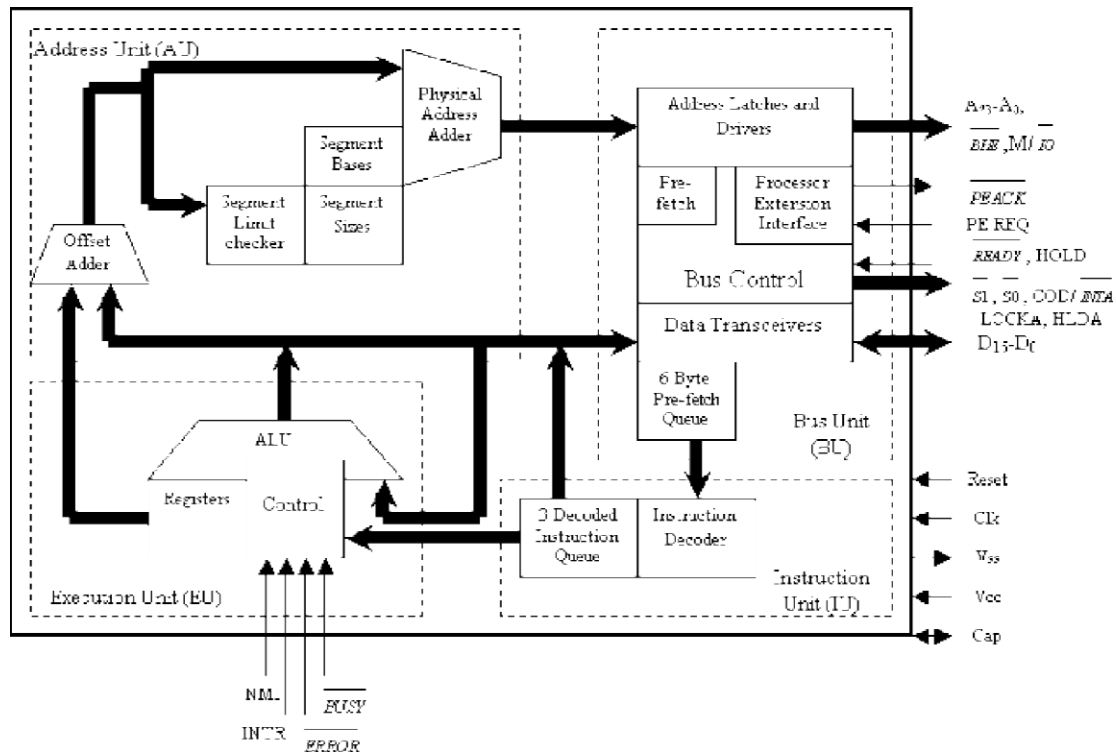
The 80286 is the first member of the family of advanced microprocessors with memory management and protection abilities.

### **Features**

- The 80286 microprocessor is an advanced version of the 8086 microprocessor that is designed for multi user and multitasking environments.
- The 80286 addresses 16 M Byte of physical memory and 1G Bytes of virtual memory by using its memory-management system.
- The 80286 is basically an 8086 that is optimized to execute instructions in fewer clocking periods than the 8086.
- Various versions of 80286 are available that runs on 12.5 MHz , 10 MHz and 8 MHz clock frequencies.
- Like the 80186, the 80286 doesn't incorporate internal peripherals; instead it contains a memory management unit (MMU).
- The 80286 operates in both the real and protected modes
- In the real mode, the 80286 addresses a 1MByte memory address space and is virtually identical to 8086.
- In the protected mode, the 80286 addresses a 16MByte memory space.



## 80286-ARCHITECTURE



The CPU contains four functional blocks

1. Address Unit (AU)
2. Bus Unit (BU)
3. Instruction Unit (IU)
4. Execution Unit (EU)

**Address Unit:** The address unit is responsible for calculating the physical address of instructions and data that the CPU wants to access. Also the address lines derived by this unit may be used to address different peripherals. The physical address computed by the address unit is handed over to the bus unit (BU) of the CPU.

**Bus Unit:** Major function of the bus unit is to fetch instruction bytes from the memory. Instructions are fetched in advance and stored in a queue to enable faster execution of the instructions. The bus unit also contains a bus control module that controls the prefetcher module.

These prefetched instructions are arranged in a 6-byte instructions queue. The 6- byte prefetch queue forwards the instructions arranged in it to the instruction unit (IU).

**Instruction Unit:** The instruction unit (IU) accepts instructions from the prefetch queue and an instruction decoder decodes them one by one. The decoded instructions are latched onto a decoded instruction queue. The output of the decoding circuit drives a control circuit in the execution unit.

**Execution Unit:** The execution unit (EU) is responsible for executing the instructions received from decoded instruction queue. The decoded instruction queue sends the data part of the instruction over the data bus. The EU contains the register bank used for storing the data as scratch pad, or used as special purpose registers. The ALU, the heart of the EU, carries out all the arithmetic and logical operations and sends the results over the data bus or back to the register bank.

### **Register Organization of 80286**

The 80286 CPU contains almost the same set of registers, as in 8086, namely

1. Eight 16-bit general purpose registers
2. Four 16-bit segment registers
3. Status and control registers
4. Instruction Pointer

The flag register reflects the results of logical and arithmetic instructions.

-	NT	IOPL	OF	DF	IF	TF	SF	ZF	-	AF	-	PF	-	CF
---	----	------	----	----	----	----	----	----	---	----	---	----	---	----

D2, D4, D6, D7 and D11 are called as status flag bits. The bits D8 (TF) and D9 (IF) are used for controlling machine operation and thus they are called control flags. The additional fields available in 80286 flag registers are:

1. **IOPL - I/O Privilege Field** (bits D12 and D13)
2. **NT - Nested Task flag** (bit D14)
3. **PE - Protection Enable** (bit D16, is set to select the protected mode of operation. The 80286

could not return to real mode without a hardware reset)

4. **MP - Monitor Processor Extension** (bit D17, is set to indicate that the arithmetic coprocessor is present in the system)
5. **EM - Processor Extension Emulator** (bit D18, Is set to cause a type 7 interrupt for each ESC instruction. We often use this interrupt to emulate, with software, the function of the coprocessor. Emulation reduces the system cost, but it often requires at least 100 times longer executing the emulated coprocessor instructions.)
6. **TS – Task Switch** (bit D19, If TS = 1, a numeric coprocessor instruction causes a type 7 (coprocessor not available) interrupt.)

### **Machine Status Word (MSW)**

The machine status word consists of four flags – PE, MP, EM and TS of the four lower order bits D19 to D16 of the upper word of the flag register. The LMSW and SMSW instructions are available in the instruction set of 80286 to write and read the MSW in real address mode.

### **Real Address Mode**

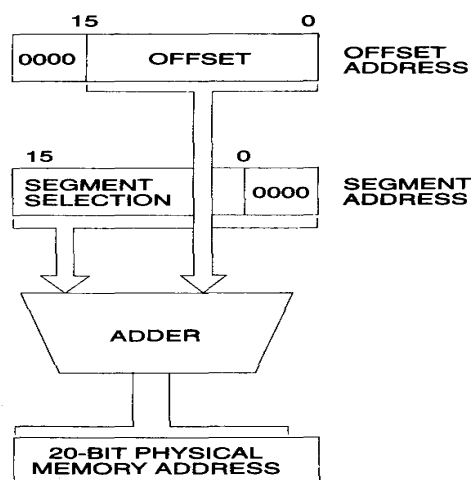
- Act as a fast 8086
- Instruction set is upwardly compatible
- It addresses only 1 M byte of physical memory using A0-A19.
- In real addressing mode of operation of 80286, it just acts as a fast 8086. The instruction set is upward compatible with that of 8086.

The 80286 addresses only 1Mbytes of physical memory using A0- A19. The lines A20-A23 are not used by the internal circuit of 80286 in this mode. In real address mode, while addressing the physical memory, the 80286 uses BHE along with A0- A19. The 20-bit physical address is again formed in the same way as that in 8086. The contents of segment registers are used as segment base addresses.

As in 8086, the physical memory is organized in terms of segments of 64Kbyte maximum size.

An exception is generated, if the segment size limit is exceeded by the instruction or the data. The overlapping of physical memory segments is allowed to minimize the memory requirements for a task. The 80286 reserves two fixed areas of physical memory for system initialization and interrupt vector table. In the real mode the first 1Kbyte of memory starting from address 0000H to 003FFH is reserved for interrupt vector table.

Also the addresses from FFFF0H to FFFFFH are reserved for system initialization. The program execution starts from FFFFH after reset and initialization. The interrupt vector table of 80286 is organized in the same way as that of 8086. Some of the interrupt types are reserved for exceptions, single-stepping and processor extension segment overrun, etc. When the 80286 is reset, it always starts the execution in real address mode.



**Fig: Real address mode Address calculation**

### **PROTECTED VIRTUAL ADDRESS MODE (PVAM)**

80286 is the first processor to support the concepts of virtual memory and memory management. The virtual memory does not exist physically it still appears to be available within the system. The concept of VM is implemented using Physical memory that the CPU can directly access and secondary memory that is used as storage for data and program, which are stored in secondary memory initially.

The Segment of the program or data required for actual execution at that instant is fetched from

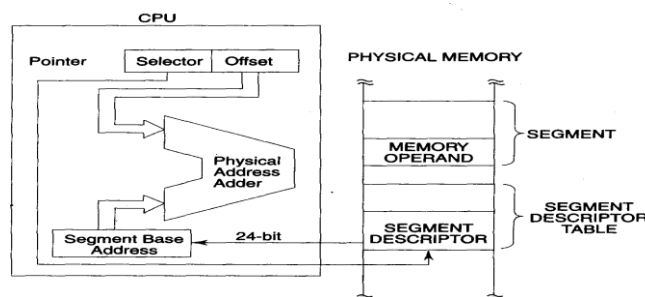
the secondary memory into physical memory. After the execution of this fetched segment, the next segment required for further execution is again fetched from the secondary memory, while the results of the executed segment are stored back into the secondary memory for further references. This continues till the complete program is executed.

During the execution the partial results of the previously executed portions are again fetched into the physical memory, if required for further execution. The procedure of fetching the chosen program segments or data from the secondary storage into physical memory is called *swapping*. The procedure of storing back the partial results or data back on the secondary storage is called *unswapping*. The virtual memory is allotted per task.

The 80286 is able to address 1 G byte (230 bytes) of virtual memory per task. The complete virtual memory is mapped on to the 16Mbyte physical memory. If a program larger than 16Mbyte is stored on the hard disk and is to be executed, if it is fetched in terms of data or program segments of less than 16Mbyte in size into the program memory by swapping sequentially as per sequence of execution.

Whenever the portion of a program is required for execution by the CPU, it is fetched from the secondary memory and placed in the physical memory is called *swapping in* of the program. A portion of the program or important partial results required for further execution may be saved back on secondary storage to make the PM free for further execution of another required portion of the program is called *swapping out* of the executable program.

80286 uses the 16-bit content of a segment register as a selector to address a descriptor stored in the physical memory. The descriptor is a block of contiguous memory locations containing information of a segment, like segment base address, segment limit; segment type, privilege level, segment availability in physical memory; descriptor type and segment use another task.



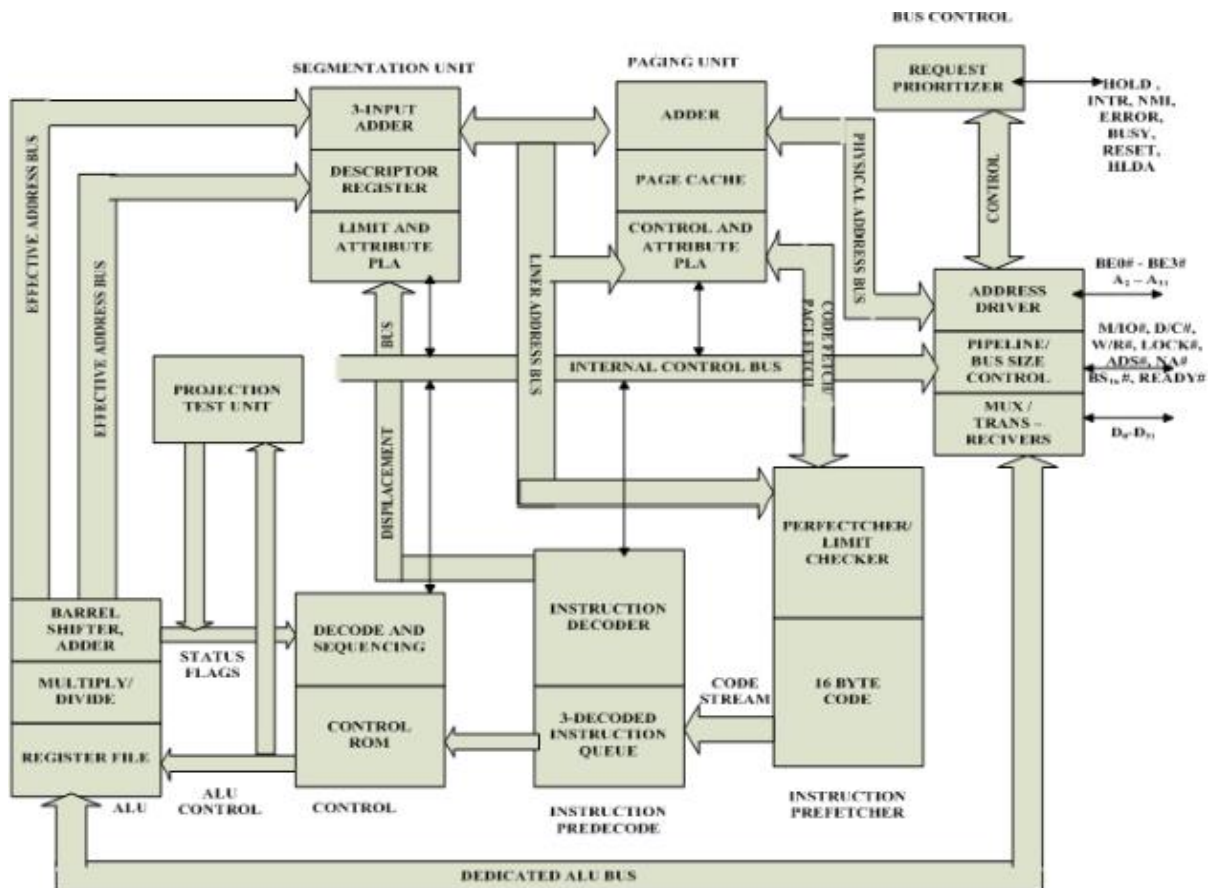
**Physical address calculation in PVAM**

## INTEL 80386

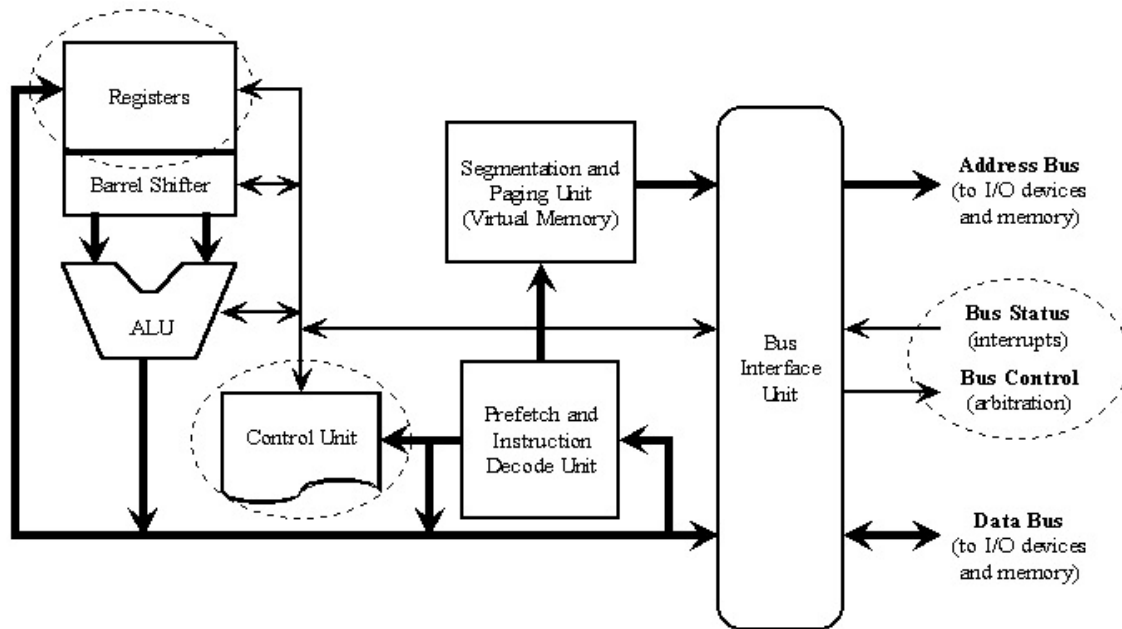
### Features:

- 32-bit general and offset registers.
- 16-byte prefetch queue.
- Memory Management Unit with a Segmentation Unit and a Paging Unit.
- 32-bit Address and Data Bus.
- 4-Gbyte Physical address space.
- 64-Tbyte virtual address space.
- i387 numerical coprocessor with IEEE standard-754-1985 for floating point arithmetic.
- 64K 8-, 16-, or 32-bit ports.
- Implementation of real, protected and virtual 8086 modes.
- The protection mode operation provides paging, virtual addressing, multilevel protection and multitasking capabilities.

### ARCHITECTURE:



OR



The Internal Architecture of 80386 is divided into 3 sections.

- Central processing unit
- Memory management unit
- Bus interface unit

**Central processing unit** is further divided into Execution unit and Instruction unit. **Execution unit** has 8 General purpose and 8 Special purpose registers which are either used for handling data or calculating offset addresses. The **Instruction unit** decodes the op code bytes received from the 16-byte instruction code queue and arranges them in a 3- instruction decoded instruction queue. After decoding them pass it to the control section for deriving the necessary control signals.

- The **barrel shifter** increases the speed of all shifts and rotate operations.
- The **multiply / divide logic** implements the bit-shift-rotate algorithms to complete the operations in minimum time. Even 32- bit multiplications can be executed within one microsecond by the multiply /divide logic.

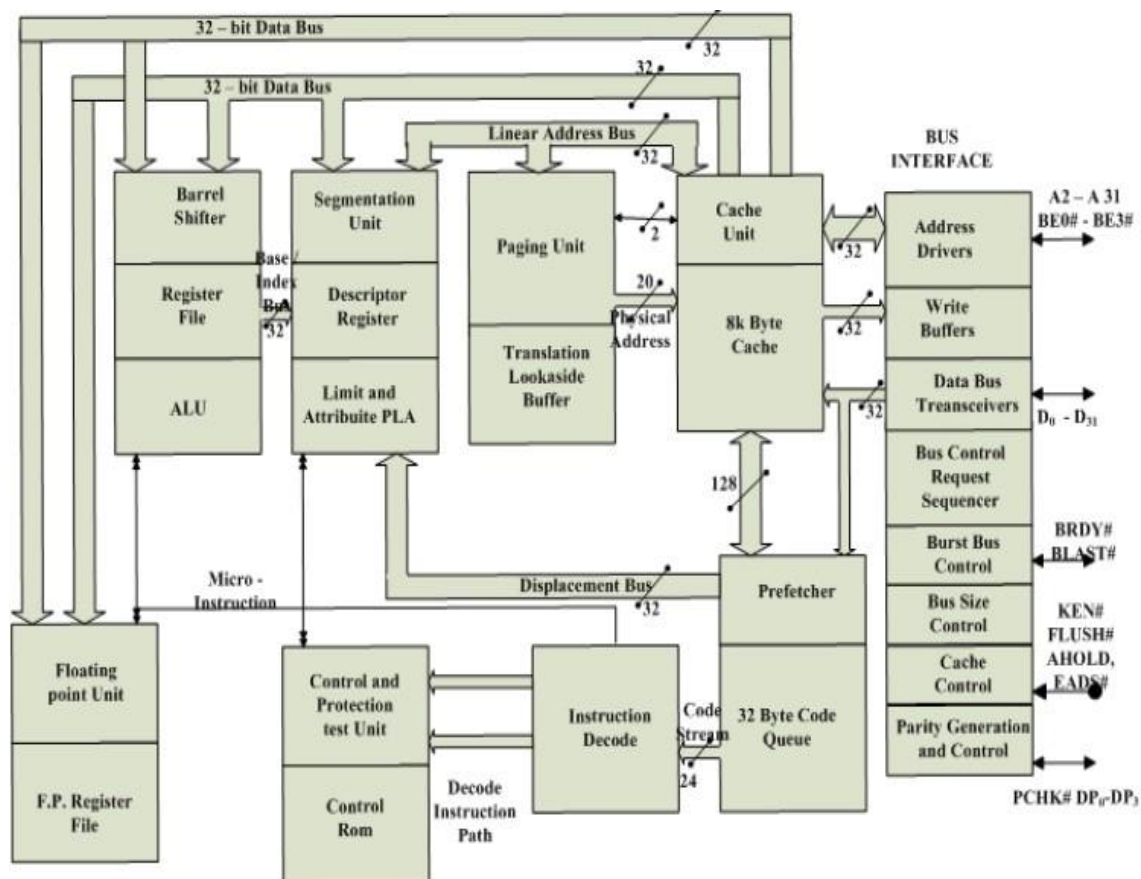
- The **Memory management unit** consists of a Segmentation unit and a Paging unit.
- **Segmentation unit** allows the use of two address components, viz. segment and offset for revocability and sharing of code and data. Segmentation unit allows segments of size 4Gbytes at max. The Segmentation unit provides a 4 level protection mechanism for protecting and isolating the system code and data from those of the application program.
- The **Paging unit** organizes the physical memory in terms of pages of 4kbytes size each. Paging unit works under the control of the segmentation unit, i.e. each segment is further divided into pages. The virtual memory is also organized in terms of segments and pages by the memory management unit. Paging unit converts linear addresses into physical addresses. The advantage of paging scheme is that the complete segment of a task need not be in the physical memory at any time. Only a few pages of the segments, which are required currently for the execution, need to be available in the physical memory. Thus the memory requirement of the task is substantially reduced, relinquishing the available memory for other tasks. Whenever the other pages of task are required for execution, they may be fetched from the secondary storage. The previous page which is executed need not be available in the memory, and hence the space occupied by them may be relinquished for other tasks. Thus paging mechanism provides an effective technique to manage the physical memory for multitasking systems.
- The **control and attribute PLA** checks the privileges at the page level. Each of the pages maintains the paging information of the task. The limit and attribute PLA checks segment limits and attributes at segment level to avoid invalid accesses to code and data in the memory segments.
- The **Bus control unit** has a prioritizer to resolve the priority of the various bus requests. This controls the access of the bus. The address driver drives the bus enable and address signal A0 – A31. The pipeline and dynamic bus sizing unit handle the related control signals.
- The **data buffers** interface the internal data bus with the system bus.



## INTEL 80486

### Features

- Improved 80386 CPU (6 extra instructions)
- Hard-wired implementation of frequently used instructions (as in RISCs)
- A 5 stage instruction pipeline
- An 8K Cache Memory + cache controller (previously a separate device)
- An on-chip Floating Point coprocessor
- Longer Prefetch Queue (32-bytes as opposed to 16 on the 80386)
- Higher frequency operation
- About a million transistors
- Like the 80386 it uses real, protected and virtual 8086 modes and its Memory Management Unit include a Segmentation Unit and a Paging Unit.



## The Pentium Processors

The Pentium microprocessor signals an improvement to the architecture found in the 80486 microprocessor. The changes include an improved cache structure, a wider data bus width, a faster numeric coprocessor, a dual integer processor, and branch prediction logic. The cache has been reorganized to form two caches that are each 8K bytes in size, one for caching data, and the other for instructions. The data bus width has been increased from 32 bits to 64 bits. The numeric coprocessor operates at about five times faster than the numeric 80486 coprocessor. A dual-integer processor often allows two instructions per clock. Finally, the branch prediction logic allows programs that branch to execute more efficiently.

### **Features of Pentium Processors:**

- It is a highly integrated device containing about 1.2 million transistors.
- **Wider Data Bus Width:** The Pentium processors have a wider data bus width. The data bus width has been increased from 32-bit to 64-bit to improve the data transfer rate.
- **Faster Floating Point Unit:** Faster algorithm provides up to ten times speed-up for common operations including add, multiply and load.
- **Improved Cache Structure:** Pentium processors include separate code and data caches integrated on-chip to meet performance goals.
- **Dual Integer Processor:** Pentium processor has integer processor. It allows execution of two instructions per clock.
- **Branch Prediction Logic:** The Pentium uses the technique called branch prediction to check whether a branch will be valid or invalid.
- **Data Integrity and Error Detection:** The Pentium processors have added significant data integrity and error detection capability.
- **Super Scalar Processor:** Processors capable to parallel instruction execution of multiple instructions are known as super scalar processors. The Pentium is capable, under special circumstances of executing two integer or two floating point instructions simultaneously and thus it support super scalar architecture.
- The Pentium MMX version contained an additional MMX unit that speeds up multimedia and 3D applications. Processing multimedia data involved instructions operating on large volumes of packetized data.

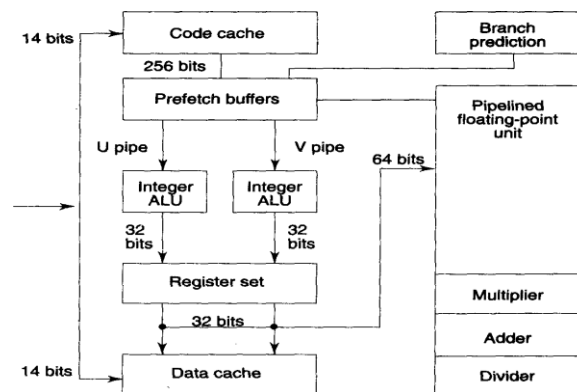
### **SALIENT FEATURES OF 80586 (PENTIUM)**

A salient feature of Pentium is its superscalar, super pipelined architecture. It has two integer

pipelines U and V, where each one is a 4-stage pipeline. This enhances the speed of integer arithmetic of Pentium to a large extent. Moreover, it has an on-chip floating-point unit, which has increased the floating-point performance manifold compared to the floating-point performances of 80386/486 processors.

Another feature of Pentium is that it contains two separate caches, viz. data cache and instruction cache. In 80486 there was a single unified data/instruction cache.

The Intel CPU architectures handle 80486 issues but only one instruction is to be there in the execution unit per cycle. This obviously leads to a comparatively slow process of decoding and execution. For enhancement of processor performance beyond one instruction per cycle, the computer architects employ the technique of multiple instruction issue (MII). Thus a microprocessor which is capable of issuing more than one instruction per single processor cycle will be termed as MII microprocessor. Obviously executing more than one instruction in a cycle, the microprocessor must have more than execution channels.



**Fig. 5 .1 Pentium CPU Architecture**

## **Pentium II**

Pentium II is also a 32-bit processor with 64-bit data bus and 36-bit address bus to address up to 64GB of physical memory space. It is actually a Pentium pro processor with on-chip MMX (Multi Media Extension). It is available with maximum internal ratings of 233 MHz to 450 MHz.

The features of Pentium II processor are;

- (i) Supports the INTEL architecture with dynamic execution.
- (ii) Integrated primary (L1) 16-kb instruction cache and 16-kb write back data cache.

- (iii) Integrated 256kb second level (L2) cache.
- (iv) Fully compatible with previous microprocessors.
- (v) Supports MMX technology.
- (vi) Quick start and Deep sleep modes provide extremely low power dissipation.
- (vii) Integrated math co-processor.

### **The Pentium III**

The Pentium III microprocessor is an improved version of the Pentium II microprocessor. Even though it is newer than the Pentium II, it is still based on the Pentium Pro architecture.

The salient architectural features are:

1. P-III CPU has been developed using 0.25 micron technology and includes over 9.5 million transistors. It has three versions operating at 450 MHz, 500 MHz and 550 MHz which are commercially available.
2. P-III incorporates multiple branch prediction algorithms.
3. Seventy new instructions have been added to Pentium III. These instructions are useful in advanced imaging, speech processing and multimedia applications.
4. Dual independent bus architecture increases bandwidth.
5. P-III employs dynamic execution technology.
6. A 512Kbyte unified, non-blocking level 2 cache has been used.
7. Eight 64-bit wide Intel MMX registers along with a set of 57 instructions for multimedia applications are available

### **Pentium IV**

The most recent version of the Pentium Pro architecture microprocessor is the Pentium 4 microprocessor from Intel. The Pentium 4 was released initially in November 2000 with a speed of 1.3 GHz. It is currently available in speeds up to 2.0 GHz. There are two packages available for this integrated microprocessor, the 423-pin PGA and the 478-pin FC- PGA2. Both versions use the 1.8 micron technology for fabrication. As with earlier versions of the Pentium, the Pentium 4 uses a 100-MHz memory bus speed.

<b>Name</b>	<b>Date</b>	<b>Internal Reg:</b>	<b>Clock Speed</b>	<b>Data Width</b>	<b>Address Lines</b>	<b>Max: Memory Space</b>
8086	1974	16 Bit	2 MHZ	16 bits	20 Bit	1MB
80286	1982	16 Bit	6 MHZ	16 bits	24 Bit	16 MB
80386	1985	32 Bit	16 MHZ	32 bits	32 Bit	4 GB
80486	1989	32 Bit	25 MHZ	32 bits	32 Bit	4 GB
Pentium	1993	32 Bit	60 MHZ	32 bits, 64 bit bus	32 Bit	4 GB
Pentium II	1997	32 Bit	233 MHZ	32 bits, 4 bit bus	32 Bit	64 GB
Pentium III	1999	32 Bit	450 MHZ	32 bits , 64 bit bus	32 Bit	64 GB
Pentium IV	2000	32 Bit	1.5 MHZ	32 bits, 64 bit bus	32 Bit	64 GB

### **Comparison Between different Processors**