LECTURE NOTES

ON

MICROPROCESSORS AND MICROCONTROLLERS (AECB24)CORE

B.Tech V semester
(IARE-R18)
(2020-2021)

Mrs. B.Lakshmi Prasanna, Assistant Professor, ECE

ELECTRICAL AND ELECTRONICS ENGINEERING

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)

DUNDIGAL, HYDERABAD - 500043

<h1 style="text-align:center; color:red;">MODULE-1
8086 MICROPROCESSORS</h1>

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.

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.

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"

Limitations of 8 Bit microprocessor:
- Low speed of execution
- Low memory addressing capability
- Limited number of general purpose registers
- Less power full instruction set

**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

What are 1st / 2nd / 3rd / 4th generation processor?

- The processor made of PMOS technology is called 1st generation processor, and it is made up of 4 bits
- The processor made of NMOS technology is called 2nd 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 4th 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
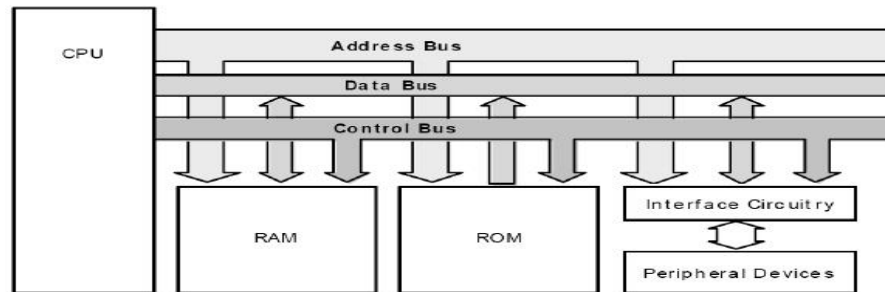
The Central Processing Unit (CPU):

This device coordinates all operations of a micro computer. It fetches programs stored in ROM"s or RAMs and executes the instructions depending one 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_0-A_{15}.
- The first 8 lines of address bus and 8 lines of data bus are multiplexed AD_0 - AD_7. Data bus is a group of 8 lines D_0 - D_7.
- 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:

Internal Architecture of 8085
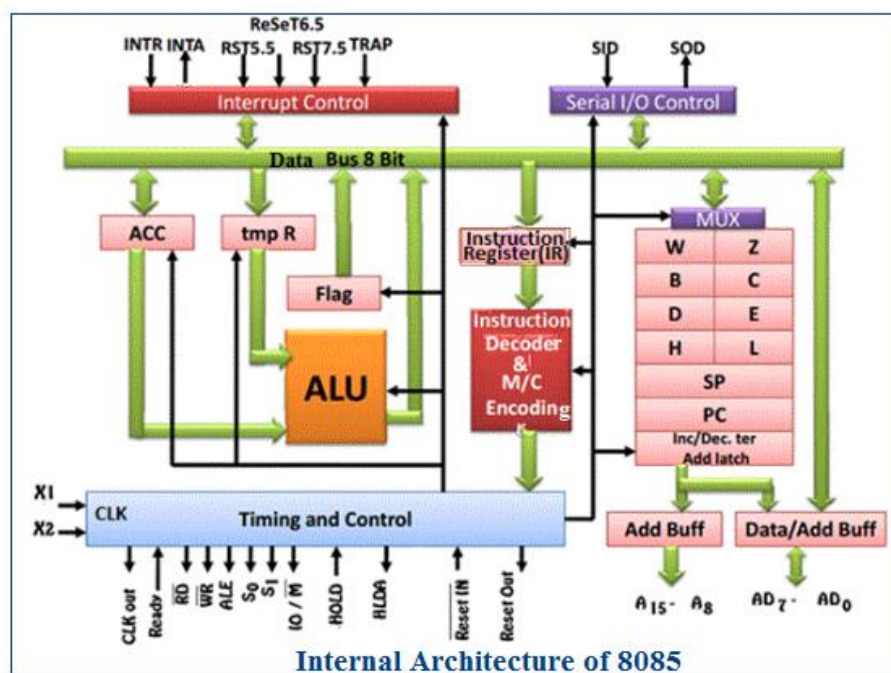
8085 Bus Structure: Address Bus:

- The address bus is a group of 16 lines generally identified as A0 to A15.

- 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.
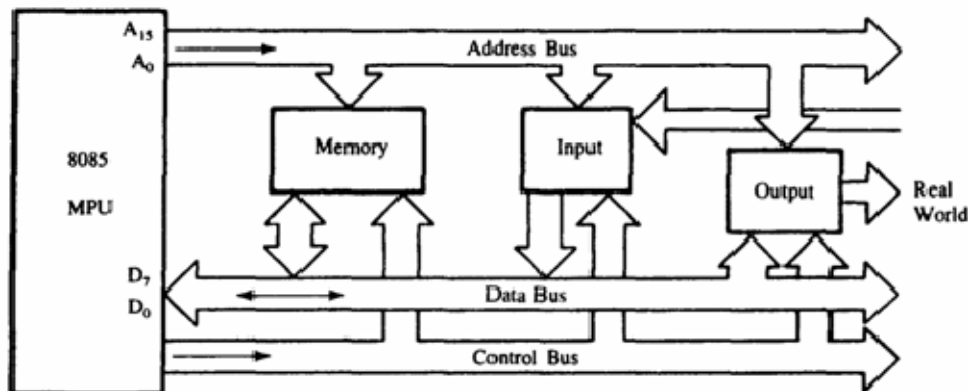

Data Bus:

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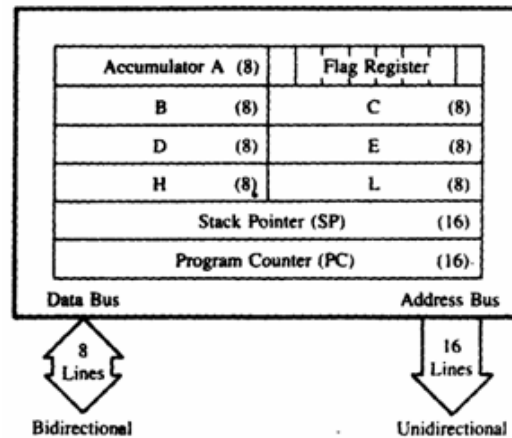
Accumulator (A):
- The accumulator is an 8-bit register that is part of the arithmetic/logic unit (ALU).
- This register is used to store 8-bit data and to perform arithmetic and logical operations.
- The result of an operation is stored in the accumulator.

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_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
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.

| 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

Functional pin diagram

Pin configuration
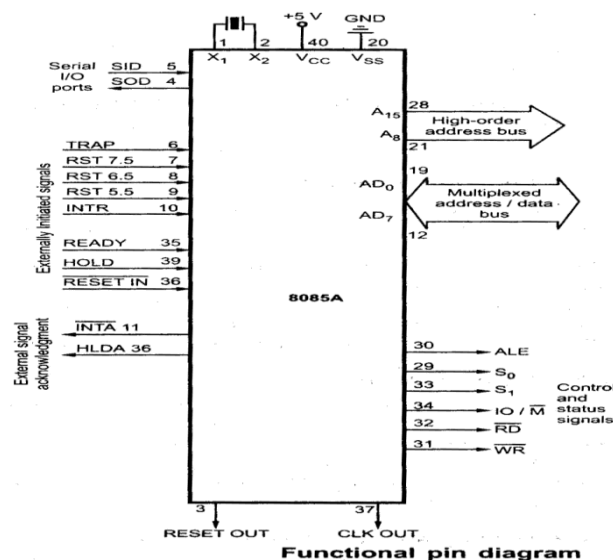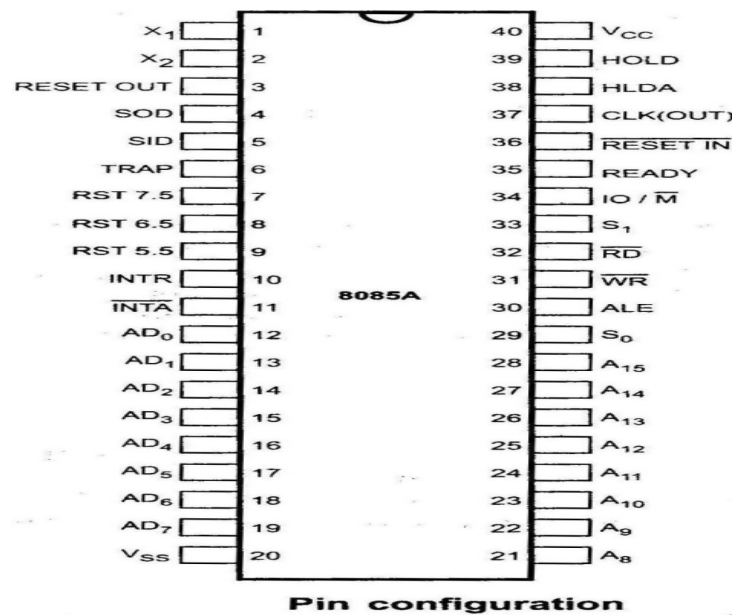
1. Power supply and clock frequency signals

- 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,

```
TRAP ─────┐
RST 7.5   │
RST 6.5   ├── (inputs)
RST 5.5   │
INTR ─────┘
INTA  ( active low output)
```
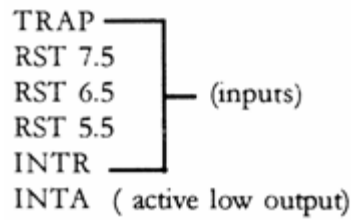
- On receipt of an interrupt, the microprocessor acknowledges the interrupt by the active low INTA (Interrupt Acknowledge) signal.
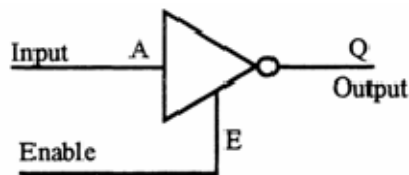
Reset In (input, active low)

- 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
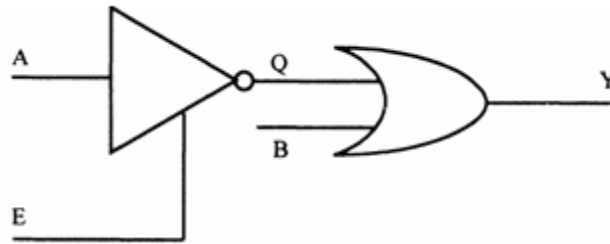
- For both high and low states, the output Q draws a current from the input of the OR gate.

- 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.
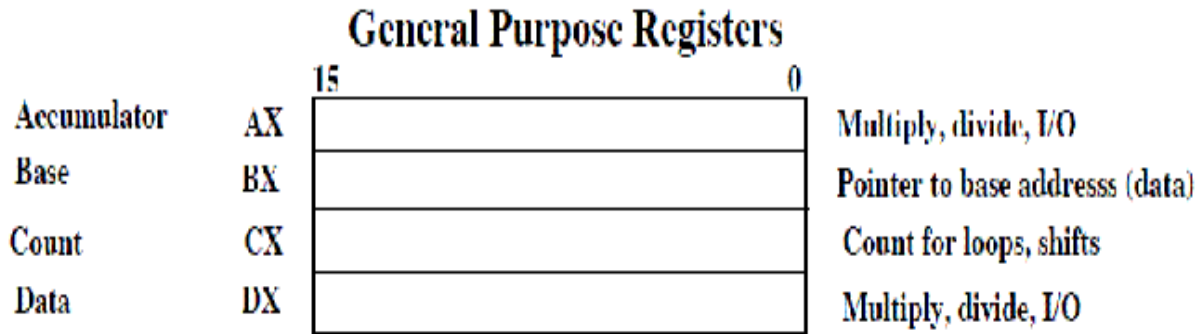
Register organization of 8086

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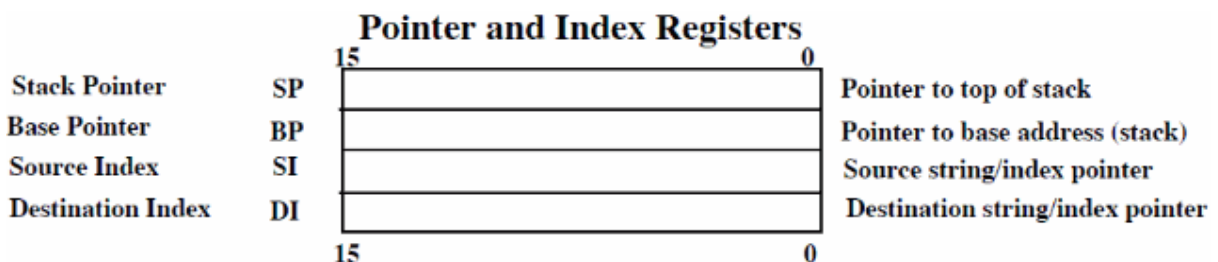
Base register consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. BL in this case contains the low-order byte of the word, and BH contains the high-order byte. BX register usually contains a data pointer used for based, based indexed or register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

Pointer and Index Registers

		15 ... 0	
Stack Pointer	SP		Pointer to top of stack
Base Pointer	BP		Pointer to base address (stack)
Source Index	SI		Source string/index pointer
Destination Index	DI		Destination string/index pointer
		15 ... 0	

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 of 8086:

It 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.

The Flags Register
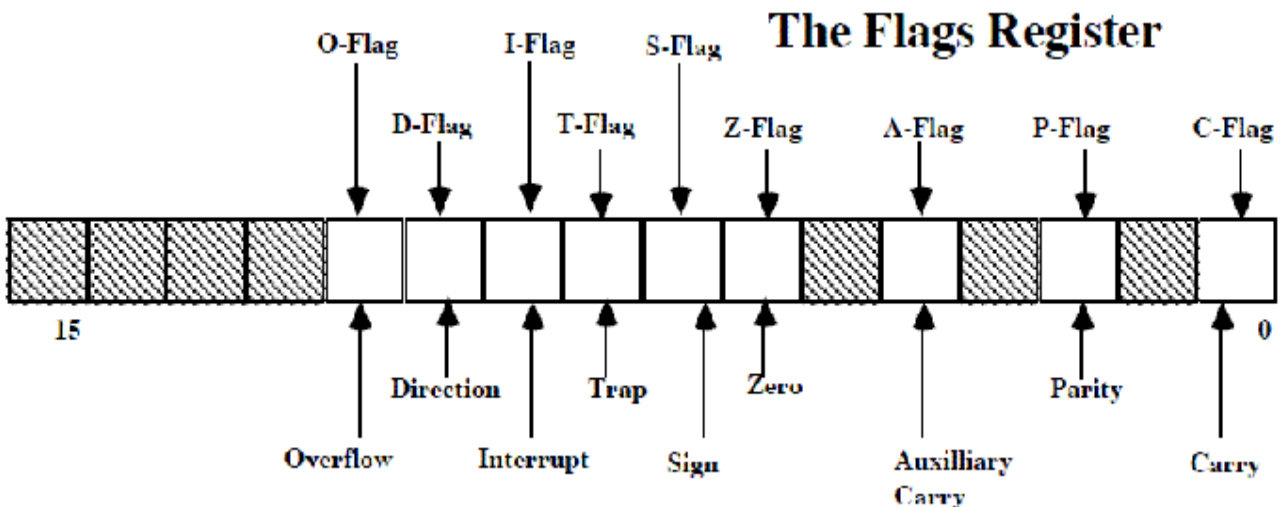
Control Flags: There are three control flags

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 flags 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 Microprocessor

Overview or Features of 8086
- It is a 16-bit Microprocessor (µp).It's ALU, internal registers works with 16bit binary word.
- 8086 has a 20 bit address bus can access up to 2^{20}= 1 MB memory locations.
- 8086 has a 16bit data bus. It can read or write data to a memory/port either 16bits 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 micro processors configuration.

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.
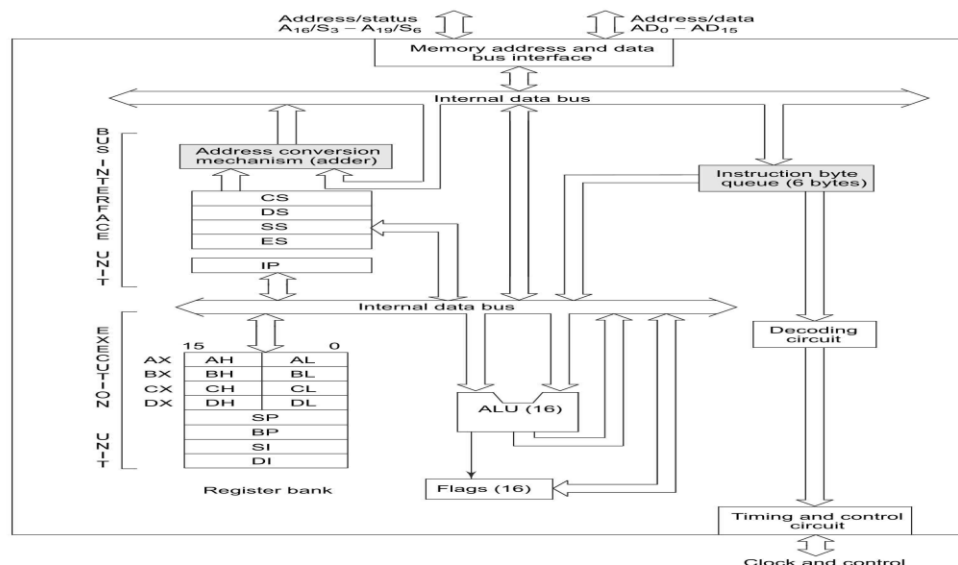

Figure: 8086 Architecture

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.

SIGNAL DESCRIPTION OF 8086

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


Pin Diagram of 8086
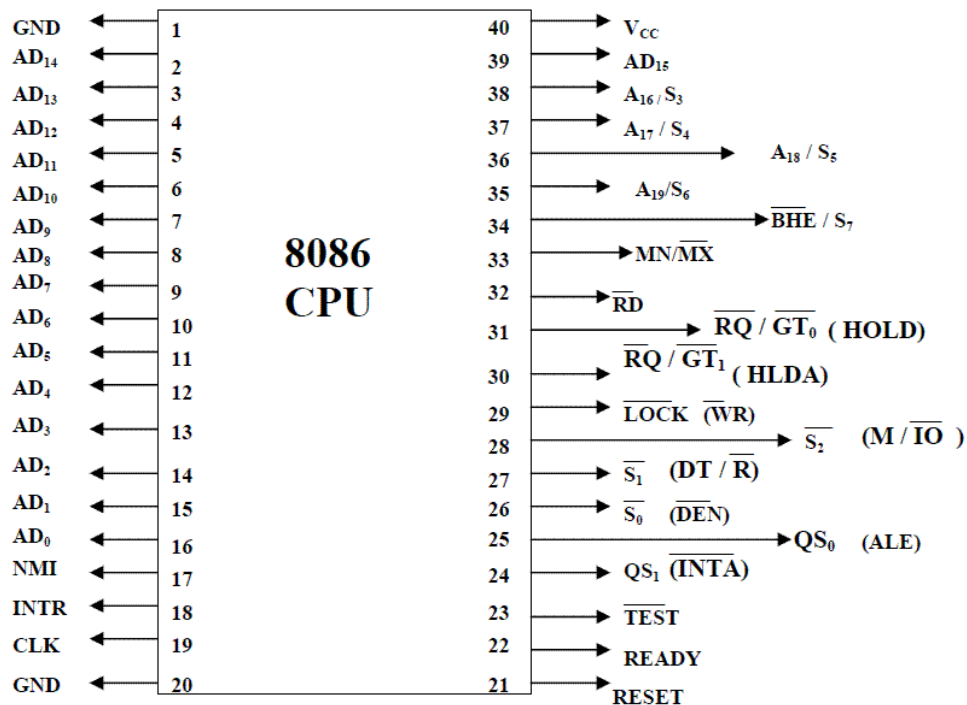
- The Microprocessor 8086 is a 16-bit CPU available in different clock rates and packaged in a 40 pin CERDIP or plastic package.

- 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.

9.

S_4	S_3	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 acknowledges 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.

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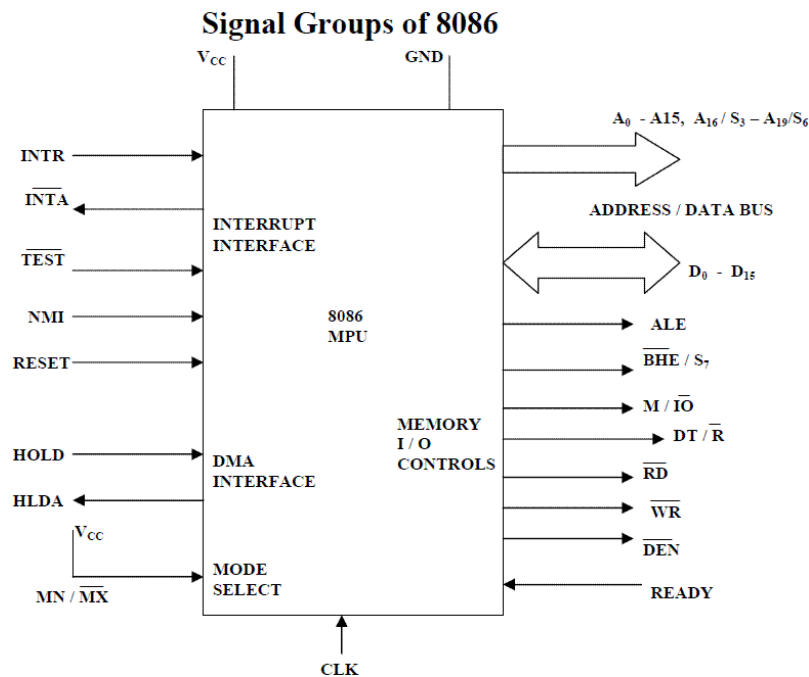
The following pin functions are for the minimum mode operation of 8086.

- **M/IO – Memory/IO:** 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	QS0	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.

PHYSICAL MEMORY ORGANIZATION

Two ways of interfacing memory with a processor

   i) linear memory   ii) Segmented memory

The memory in 8086 is organised as segmented memory. The complete 1MB memory is divided into 16 logical segments, each of 64KB size. To address specific memory location within a segment ,we need an offset address. Physical address is calculated by shifting the segment address left bitwise four times and to this result offset address is added to produce 20-bit address.

Non overlapping segments
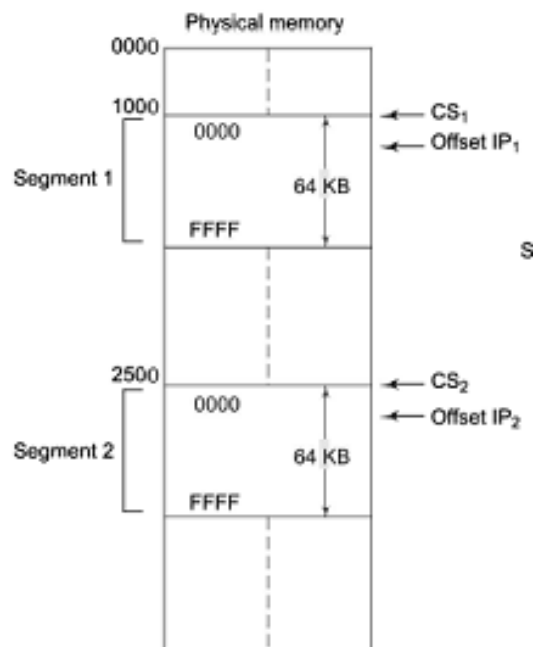
In this case if a segment starts at particular location and its maximum size is 64KB,after this only the next segment starts. In this case always each segment address varies from 0000H to FFFFH.

overlapping segments
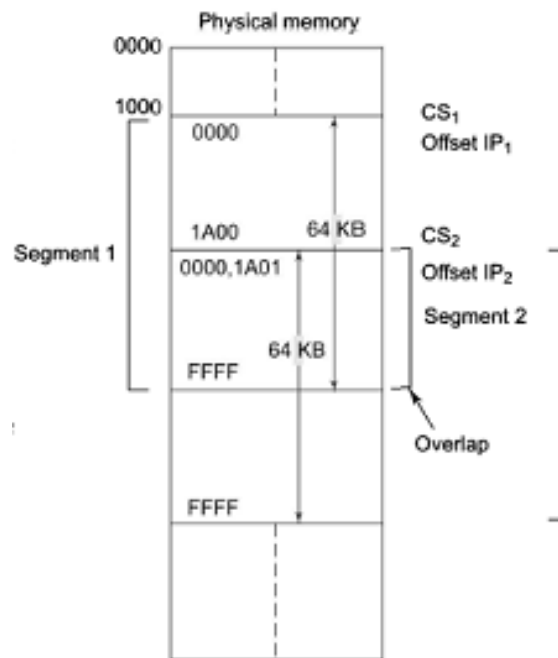
In this case if a segment starts at particular location and its maximum size is 64KB.But if another segment starts before this 64K bytes location of the first segment, the two segments are said to be overlapped segments.

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, whenever 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.
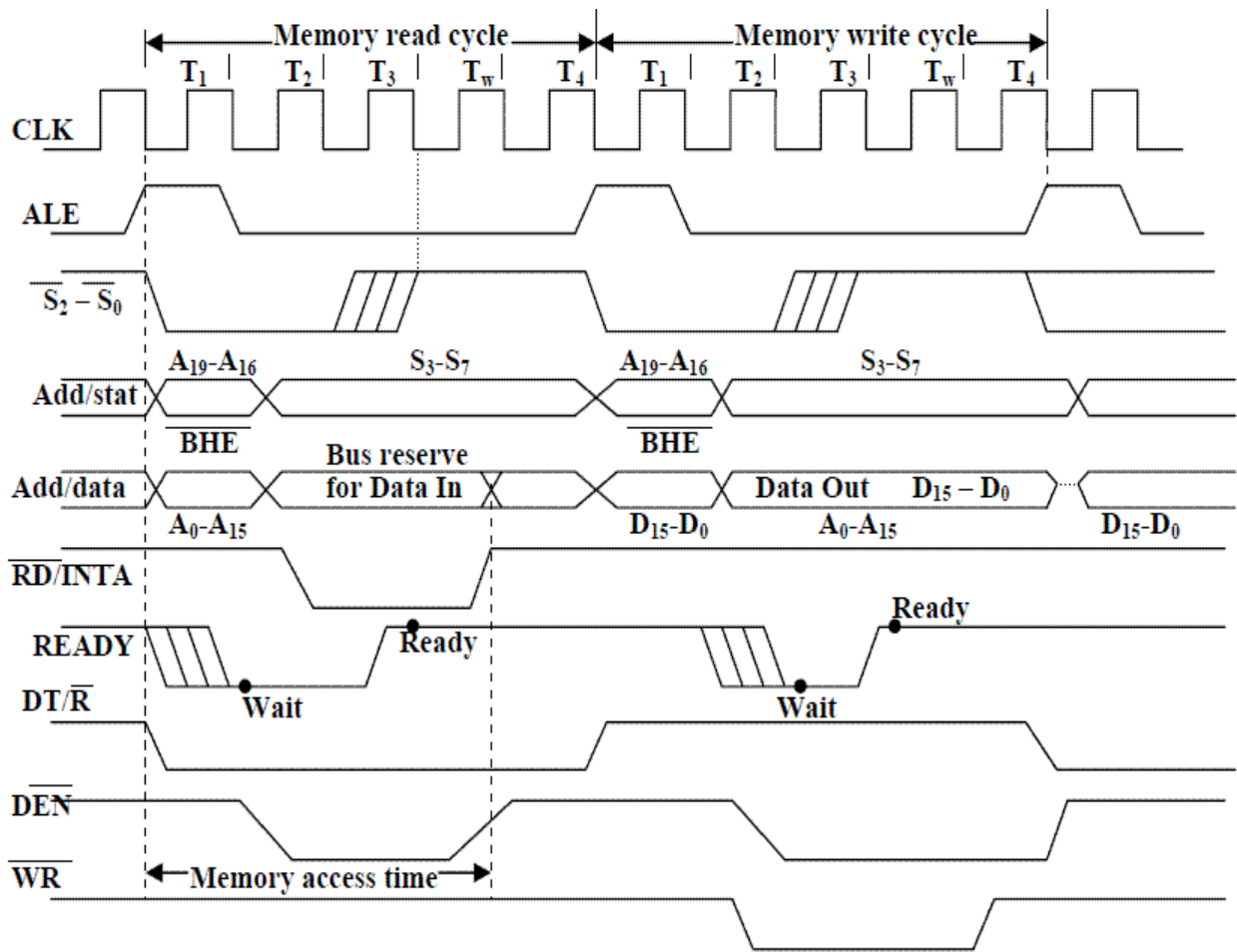
Maximum mode

- 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 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.

Minimum mode

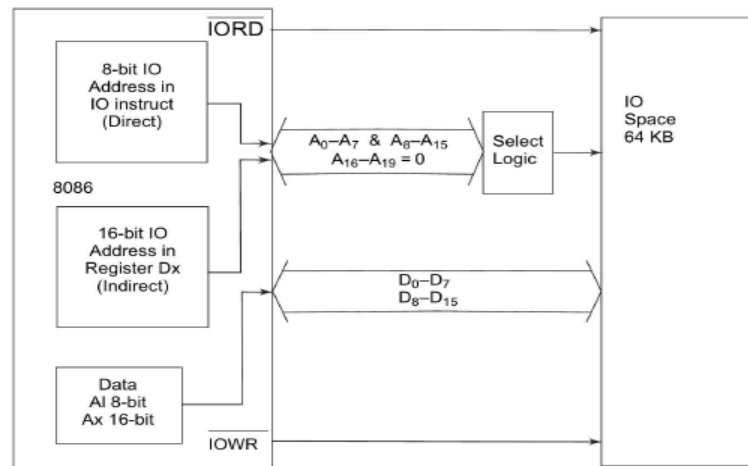- In a minimum mode 8086 system, the microprocessor 8086 is operated in minimum mode by strapping its MN/MX pin to logic 1.

- 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.

I/O addressing capability

- The 8086/8088 processor can address up to 64K byte I/O registers or 32K word registers.

- The I/O address appears on the address lines A0 to A15 for one clock cycle (T1). It may then be latched using the ALE signal.
- The upper address lines (A16 - A19) are at logic 0 level during the I/O operations.
- The 16-bitregister DX is used as 16-bit I/O address pointer, with full capability to address up to 64K devices.
- In memory mapped I/O interfacing, the I/O device addresses are treated as memory locations in page 0, i.e. segment address 0000H.Even addressed bytes are transferred on D7-Do and odd addressed bytes are transferred on D8-D15 lines.
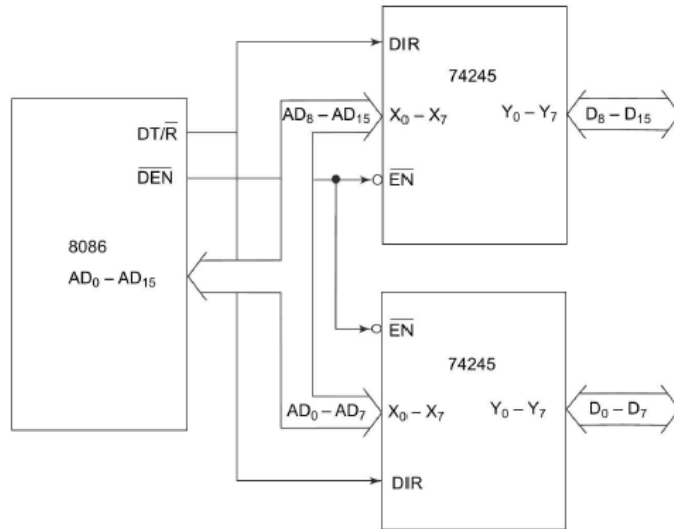

special purpose activities

- Processor Reset and Initialisation
- HALT
- TEST and Synchoronization with external signals
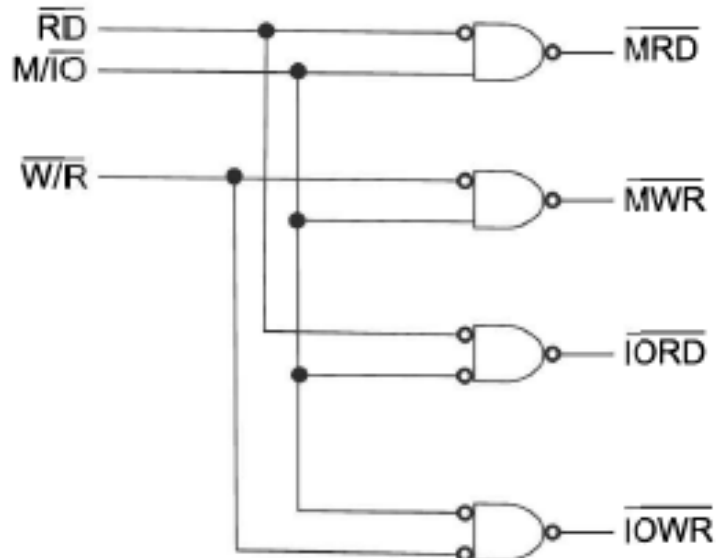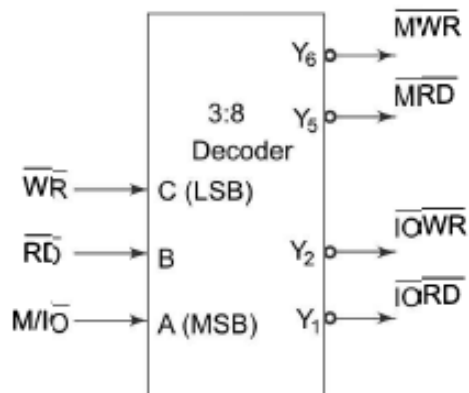- Deriving system bus

Latching 20-bit address of 8086
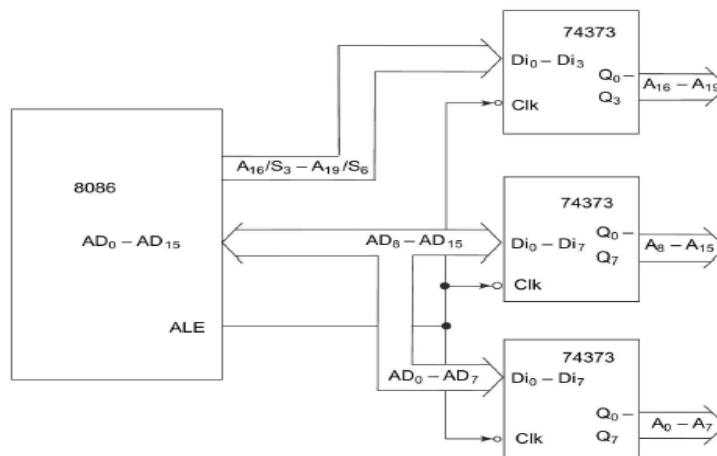
Buffering Data Bus 0f 8086

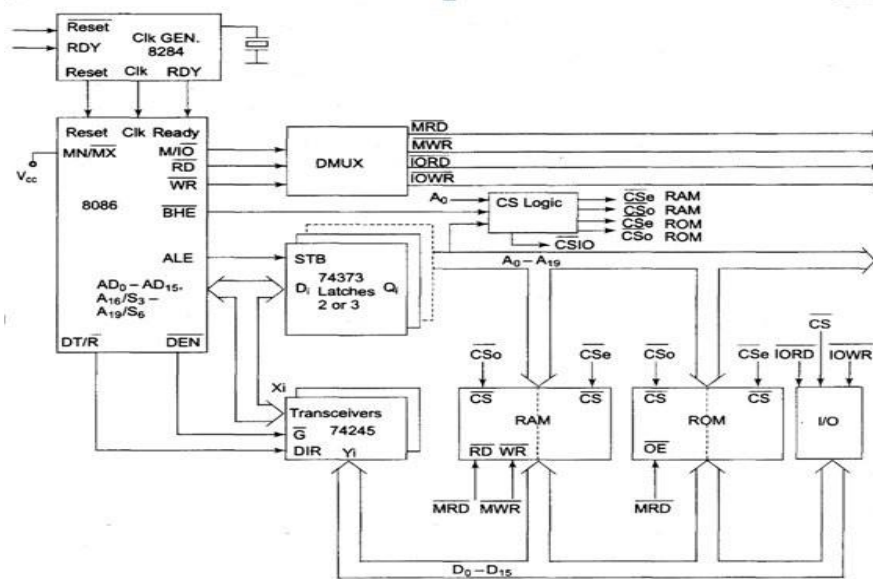**Deriving 8086 control signals**

Deriving 8086 control signals

MINIMUM MODE AND MAXIMUM MODE OF OPERATION WITH TIMING DIAGRAMS

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.
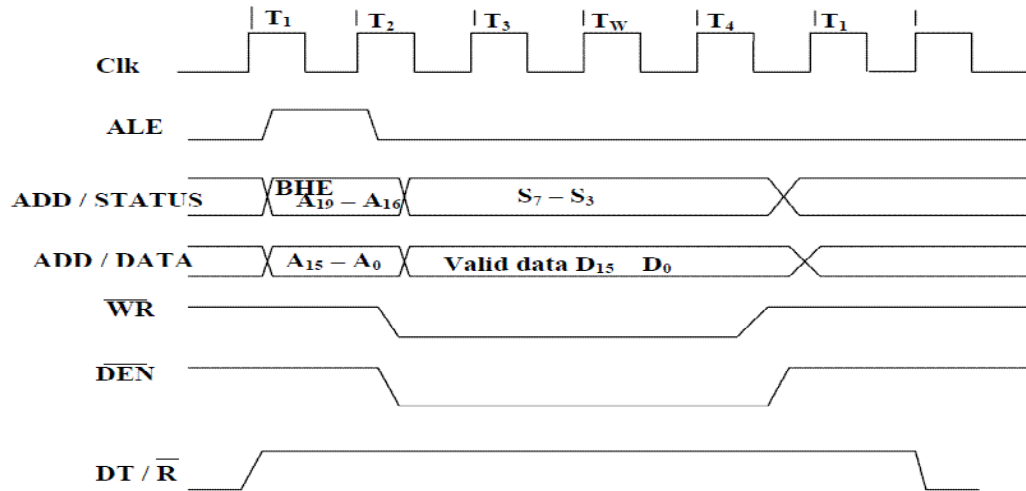
Write Cycle Timing Diagram for Minimum Mode

- 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

Bus Request and
Bus Grant Timings in Minimum Mode System
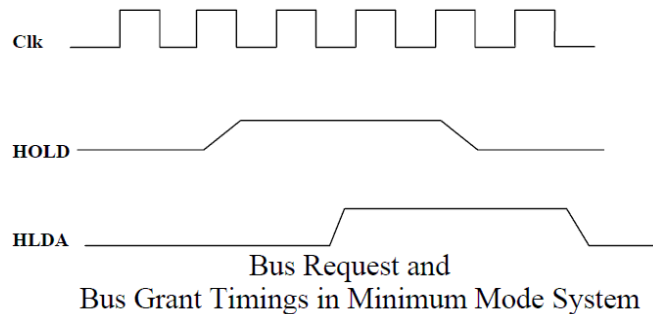
- Hold Response sequence: The HOLD pin is checked at leading edge of each clock pulse. If it is received active by the processor before T4 of the previous cycle or during T1 state of the current cycle, the CPU activates HLDA in the next clock cycle and for succeeding bus cycles, the bus will be given 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.
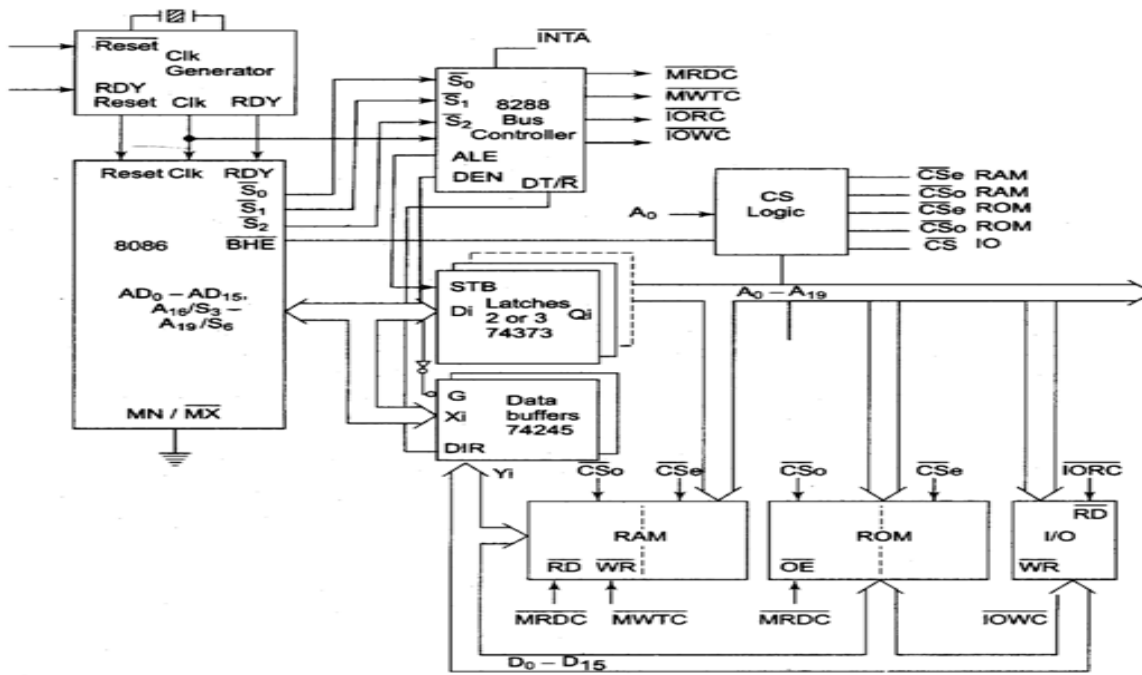
- It derives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC. The AEN, IOB and CEN pins are especially useful for multiprocessor systems.

- 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
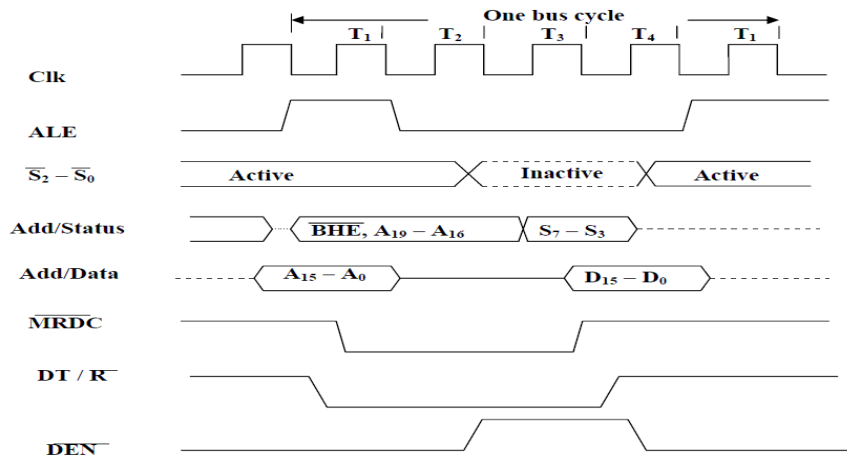
Memory Read Timing in Maximum Mode

Memory Write Timing in Maximum mode of 8086

Memory Write Timing in Maximum mode.
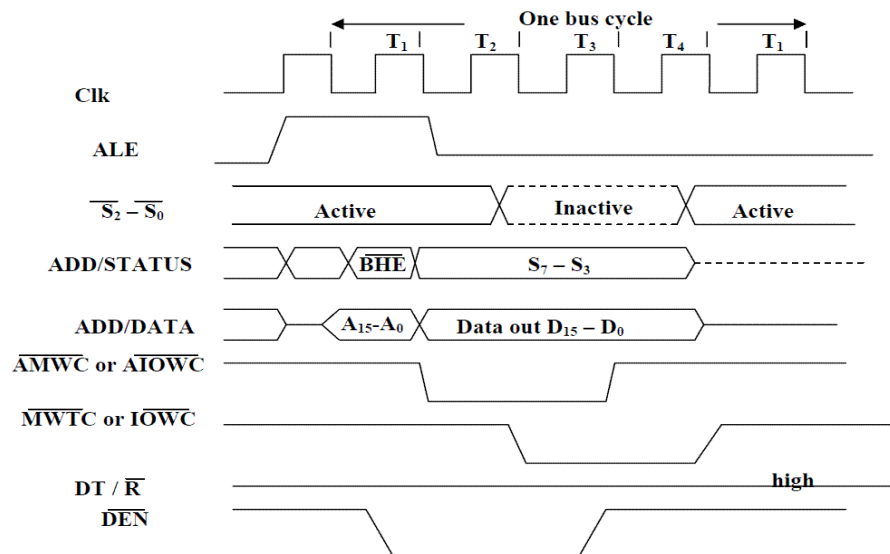
RQ/GT Timings in Maximum Mode

RQ/GT Timings in Maximum Mode
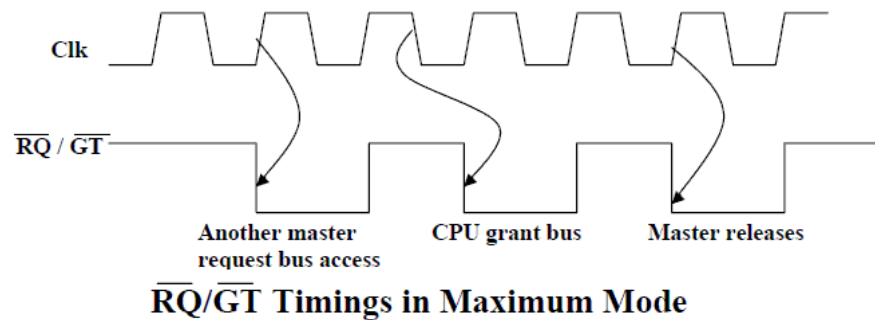
- The request/grant response sequence contains a series of three pulses. The request/grant pins are checked at each rising pulse of clock input.

- 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
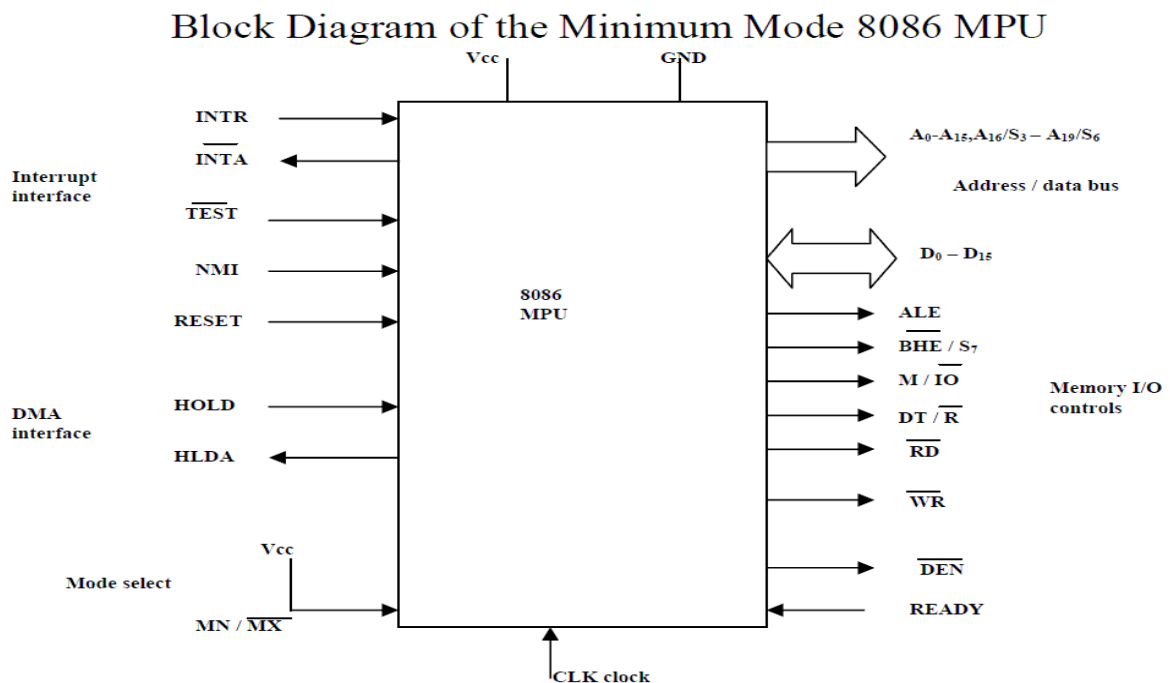
- Bit S4 and S3 together from a 2 bit binary code that identifies which of the 8086 internal segment registers is used to generate the physical address that was output on the address bus during the current bus cycle. Code S4S3 = 00 identifies a register known as extra segment register as the source of the segment address.

- Status line S5 reflects the status of another internal characteristic of the 8086. It is the logic level of the internal enable flag. The last status bit S6 is always at the logic 0 level.

S4	S3	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 D8 through D1. These lines also serve a second function, which is as the S7 status line.

- Using the M/IO 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/IO 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.

8086 Maximum mode Block Diagram
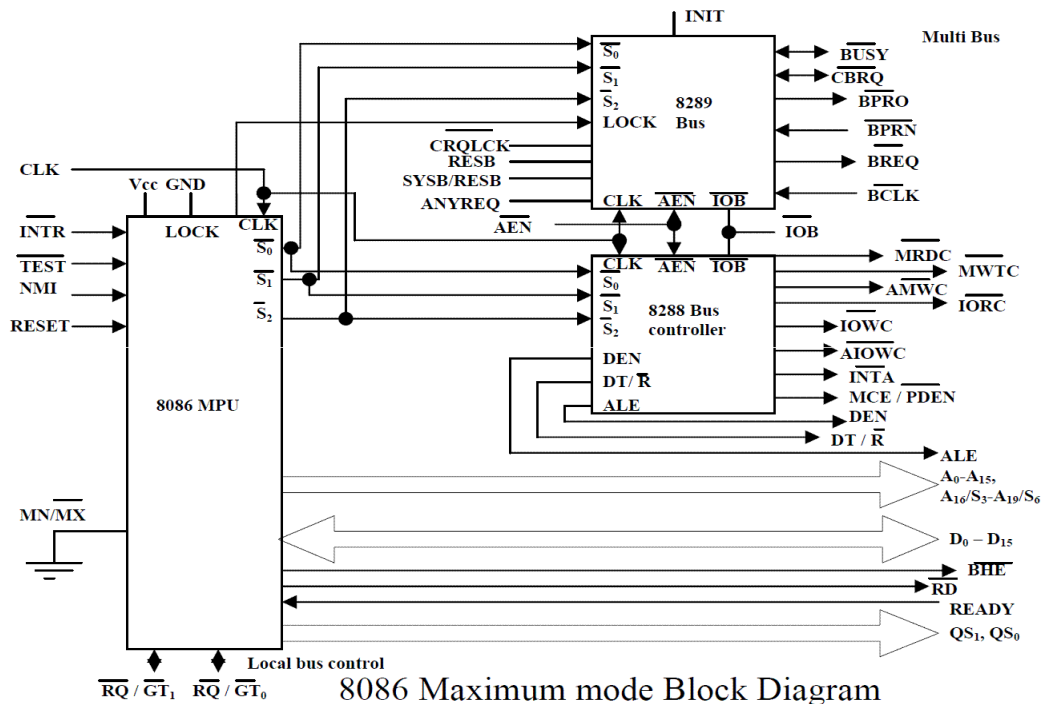
- Specially the WR, M/IO, DT/R, DEN, ALE and INTA, signals are no longer produced by the 8086. Instead it outputs three status signals S0, S1, S2 prior to the initiation of each bus cycle. This 3- bit bus status code identifies which type of bus cycle is to follow.

- 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	$\overline{INTA}$
0	0	1	Read I/O port	$\overline{IORC}$
0	1	0	Write I/O port	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	Halt	None
1	0	0	Instruction Fetch	$\overline{MRDC}$
1	0	1	Read Memory	$\overline{MRDC}$
1	1	0	Write Memory	$\overline{MWTC}$, $\overline{AMWC}$
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 (BREQ) 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.

Machine language instruction formats

The Opcode stands for Operation Code. Every Instruction has a unique 6-bit opcode.

For example, the opcode for MOV is 100010 .

D stands for direction

If D=0, then the direction is from the register

If D=1, then the direction is to the register

W stands for word

If W=0, then only a byte is being transferred, i.e. 8 bits

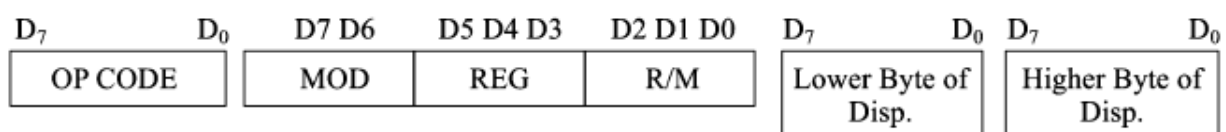If W=1, them a whole word is being transferred, i.e. 16 bits
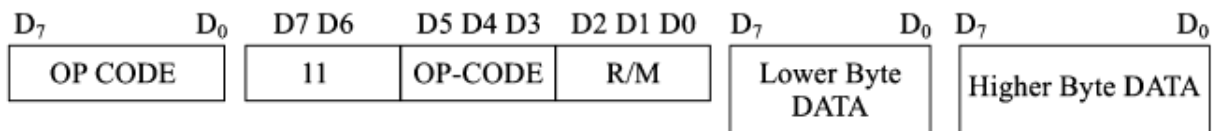
Register to Register

Register to/from Memory with no displacement:

Register to/from Memory with displacement:

Immediate Operand to Register:

D7 ... D0	D7 D6	D5 D4 D3	D2 D1 D0	D7 ... D0	D7 ... D0
OP CODE	11	OP-CODE	R/M	Lower Byte DATA	Higher Byte DATA

Immediate Operand to memory with 16-bit displacement

D7 ... D0	D7 D6	D5 D4 D3	D2 D1 D0	D7 ... D0
OP CODE	MOD	OP-CODE	R/M	Lower Byte of Disp.

D7 ... D0	D7 ... D0	D7 ... D0
Higher Byte of Disp.	Lower Byte of DATA	Higher Byte of DATA
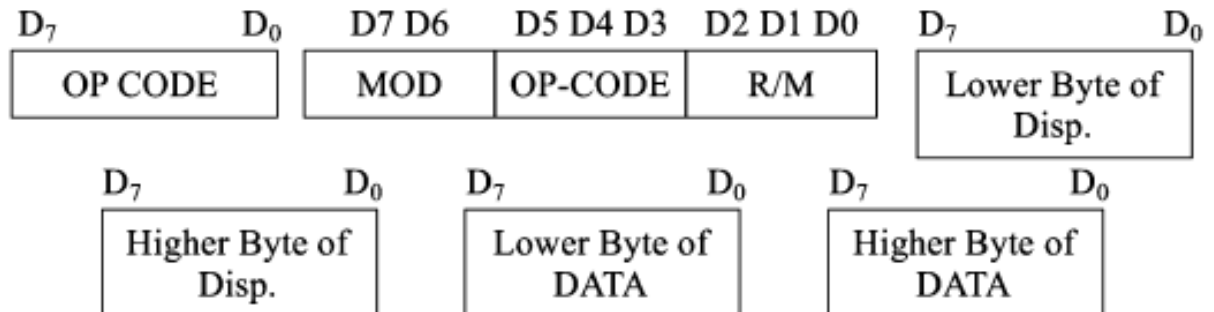
Addressing Modes of 8086:

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 as 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 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.

Memory stack segment
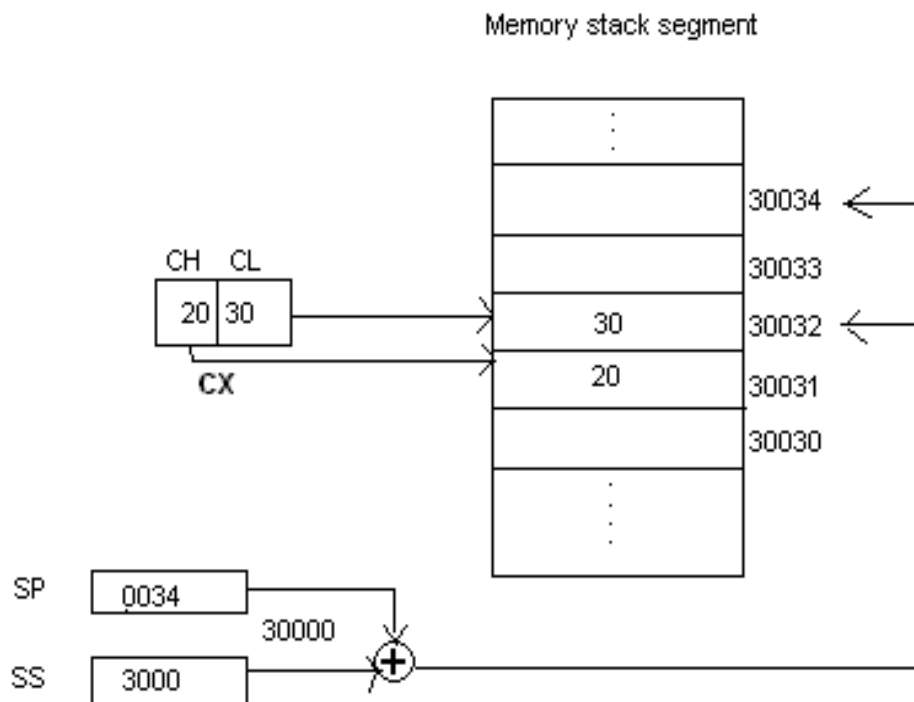
Example:-

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 <= 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 (IMPLICT)

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 IMPLICT)

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= $(1011)_2$= (B) $_{16}$ AFTER EXECUTION AX= $(0100)_2$= $(4)_{16}$).
- 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.
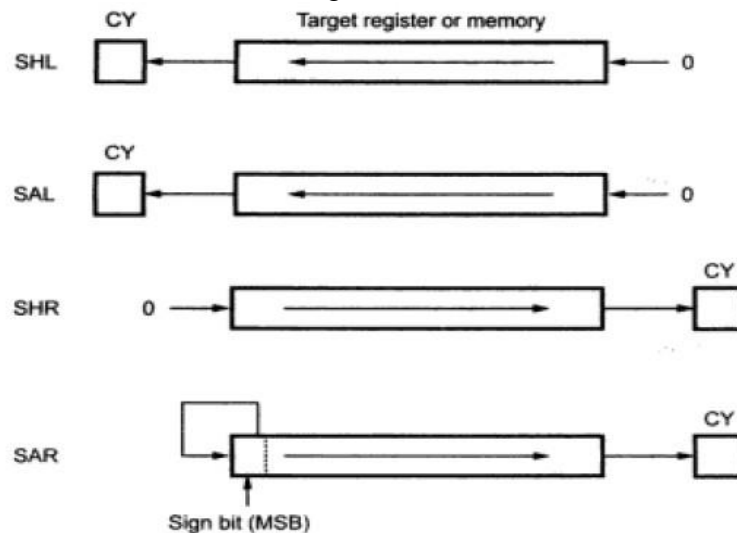

Fig.1 Shift operations

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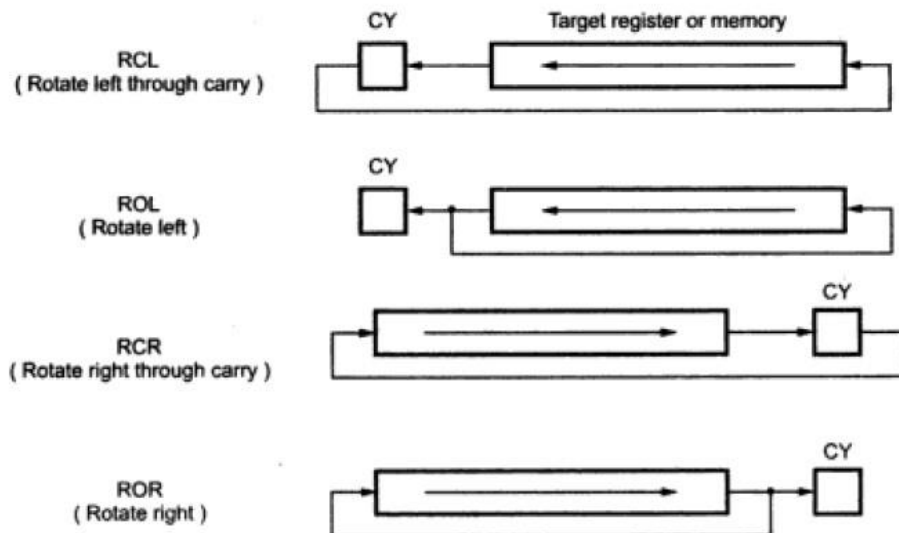

Fig.2 Rotate operations

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	1
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 B 1 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 B 1 B0 CY

 1   0   1   1   0   1   1   1   0

After execution,

B7 B6 B5 B4 B3 B2 B 1 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 AX, 1234h

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.

Example:

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/MOVSB/MOVSW

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. MOVSB is used for byte sized movements while MOVSW 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 DL, 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.

Example:		Comments
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 AND OPERATORS

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 vale.

    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:

A Macro is a named block of assembly language statements. Once defined, it can be invoked (called) as many times in a program as you wish. When you invoke a macro, a copy of its code is inserted directly into the program at the location where it was invoked. This type of automatic code insertion is also known as inline expansion. It is customary to refer to calling a macro, although technically there is no CALL instruction involved.

Declaring Macros are defined directly at the beginning of a source program, or they are placed in a separate file and copied into a program by an INCLUDE directive. Macros are expanded during the assembler's preprocessing step. In this step, the preprocessor reads a macro definition and scans the remaining source code in the program. At every point where the macro is called, the assembler inserts a copy of the macro's source code into the program. A macro definition must be found by the assembler before trying to assemble any calls of the macro. If a program defines a macro but never calls it, the macro code does not appear in the compiled program.

Defining Macros: A macro is defined using the MACRO and ENDM directives.

The syntax is:

macroname MACRO parameter-1, parameter-2...statement-list

.

.

.

ENDM

There is no fixed rule regarding indentation, but we recommend that you indent

statements between macroname and ENDM. You might also want to prefix macro names with the letter m, creating recognizable names such as **mPutChar**, **mWriteString**, and **mGotoxy.** The statements between the MACRO and ENDM directives are not assembled until the macro is called. There can be any number of parameters in the macro definition, separated by commas.

Parameters:

Macro parameters are named placeholders for text arguments passed to the caller. The arguments may in fact be integers, variable names, or other values, but the preprocessor treats them as text. Parameters are not typed, so the preprocessor does not check argument types to see whether they are correct. If a type mismatch occurs, it is caught by the assembler after the macro has been expanded. Example: The following Putchar macro receives a single input parameter called char and displays it on the console:

Putchar MACRO char

mov dl,char

mov ah,02h

int 21h

ENDM 2

Invoking Macros A macro is called (invoked) by inserting its name in the program, possibly followed by macro arguments. The syntax for calling a macro is: macroname argument-1, argument-2, ... Macroname must be the name of a macro defined prior to this point in the source code. Each argument is a text value that replaces a parameter in the macro. The order of arguments must correspond to the order of parameters, but the number of arguments does not have to match the number of parameters. If too many arguments are passed, the assembler issues a warning. If too few arguments are passed to a macro, the unfilled parameters are left blank.

PROCEDURES:

Defining a Procedure:Informally, we can define a procedure as a named block of statements that ends in a return statement.

A procedure is declared using the PROC and ENDP directives. It must be assigned a name (a valid identifier). A procedure must be written in the end of the code segment (before end directive), and it can't receive any parameter.

When you create a procedure other than your program's startup procedure, end it with a RET instruction. RET forces the CPU to return to the location from where the procedure was called. A procedure is defined using the PROC and ENDP directives.

The syntax is:

procedurename PROC

.

.

ret

procedurename ENDP

Example: Sum of Three Integers

Let's create a procedure named Sum Of that calculates the sum of three 16-bit integers. We will assume that relevant integers are assigned to AX, BX, and CX before the procedure is called. The procedure returns the sum in AX:

SumOf PROC

add ax,bx

add ax,cx

ret

SumOf ENDP

Invoking procedures A procedure is called (invoked) by inserting its name in the program Preceded by (call). The syntax for calling a procedure is:

Call procedurename

CALL and RET Instructions

The CALL instruction calls a procedure by directing the processor to begin execution at a new memory location. The procedure uses a RET (return from procedure) instruction to bring the processor back to the point in the program where the procedure was called. Mechanically speaking, the CALL instruction pushes its return address on the stack and copies the called procedure's address into the instruction pointer. When the procedure is ready to return, its RET instruction pops the return address from the stack into the instruction pointer. In 32-bit mode, the CPU executes 3 the instruction in memory pointed to by EIP (instruction pointer register). In 16-bit mode, IP points to the instruction.

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
```

MODULE-II

PROGRAMMING WITH

8086 MICROPROCESSOR

8086 ASSEMBLY LANGUAGE PROGRAMMING

Introduction:

The assembly programming language is a low-level language which is developed by using mnemonics. The microcontroller or microprocessor can understand only the binary language like 0"s or 1"s therefore the assembler convert the assembly language to binary language and store it the memory to perform the tasks. Before writing the program the embedded designers must have sufficient knowledge on particular hardware of the controller or processor, so first we required to know hardware of 8086 processor.

Machine Language:

Set of fundamental instructions the machine can execute Expressed as a pattern of 1"s and 0"s

Assembly Language:

Alphanumeric equivalent of machine language Mnemonics more human-oriented than 1"s and 0"s

Assembler:

Computer program that transliterates (one-to-one mapping) assembly to machine language Computer's native language is machine/assembly language

Why Assembly Language Programming

Faster and shorter programs: Compilers do not always generate optimum code.

Instruction set knowledge is important for machine designers.

Compiler writers must be familiar with details of machine

language. Small controllers embedded in many products

1. Have specialized functions,
2. Rely so heavily on input/output functionality,
3. HLLs inappropriate for product development.

8086 Assembly programming means develop programs in 8086 assembly programming language. 8086 Assembly is a low level programming language. The developer has to deal with object of the processor like segment and register. In this article, we will see what are the basic elements of this language and the structure of a simple program.

Basic elements of 8086 assembly programming language
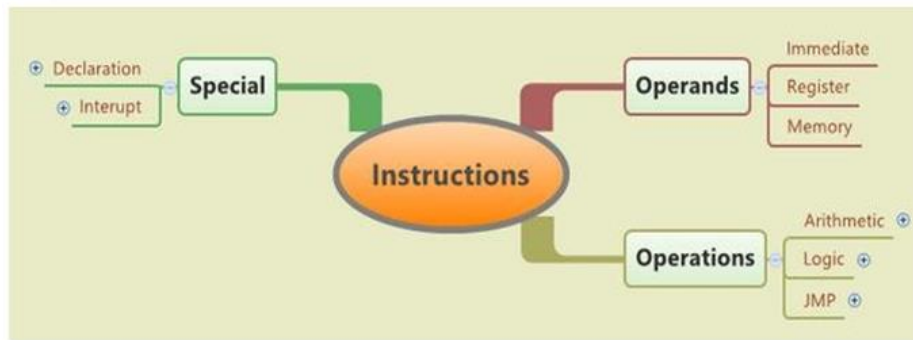
8086 assembly programming language instructions

Like we know instruction are the lines of a program that means an action for the computer to execute.
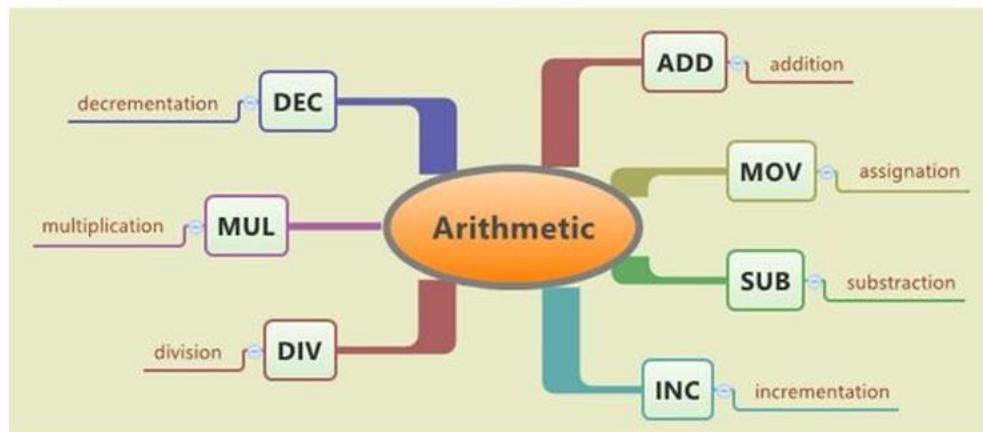
In 8086, a normal instruction is made by an operation code and sometimes operands.

Structure:

Operation Code [Operand1 [, Operand2]]

Operations

The operation is usually logic or arithmetic, but we can also find some special operation like the Jump (JMP) operation.

Operands

Operands are the parameters of the operation in the instruction. They can be use in 3 way:

• Immediate

This means a direct access of a variable that have been declared in the program.

• Register

Here we use the content of a register to be a parameter.

• Memory

Here we access to the content of a specific part of the memory using a pointer.

Special instructions

Now we will see how to declare a variable and some interrupt instruction.

Declaration

In 8086, we need to declare the name of the variable, the size of the variable and it value.

Structure:

Variable Name Size Directive value [dup (num)]

Variable Name : name of your variable.

Size Directive: word system to define the size, DB will define byte and DW will define a Word(2 byte).

Value is your variable value

dub: a system word that means duplicate used for vector. Num: number of time you duplicate, the size of your vector. Interrupt

Interrupt calls a sub routine of the system to be executed. We will see 2 simple interrupt:

Human-Readable Machine Language:

• Computers like ones and zeros…

001111101010000

• Humans like symbols…

ADD   R6, R2, R6   ; increment index register

Assembler is a program that turns symbols into machine instructions.

–   ISA-specific:

Close correspondence between symbols and instruction set

• Mnemonics for opcodes
• Labels for memory locations
–   Additional operations for allocating storage and initializing data

The assembly language programming 8086 has some rules such as

An Assembly Language Program:

• The assembly level programming 8086 code must be written in upper case letters
• The labels must be followed by a colon, for example: label:
• All labels and symbols must begin with a letter
• All comments are typed in lower case
• The last line of the program must be ended with the END directive

Op-code: A single instruction is called as an op-code that can be executed by the CPU. Here the 'MOV' instruction is called as an op-code.

Operands: A single piece data are called operands that can be operated by the op-code. Example, subtraction operation is performed by the operands that are subtracted by the operand. Syntax: SUB b, c

SYNTAX OF 8086/8088 ASSEMBLY LANGUAGE

• The language is not case sensitive.

• There may be only one statement per line. A statement may start in any column.

• A statement is either an instruction, which the assembler translates into machine code, or an assembler directive (pseudo-op), which instructs the assembler to perform some specific task.

• Syntax of a statement:

{name} mnemonic {operand(s)} {; comment}

(a) The curly brackets indicate those items that are not present or are optional in some statements.

(b) The name field is used for instruction labels, procedure names, segment names, macro names, names of variables, and names of constants.

(c) MASM 6.1 accepts identifier names up to 247 characters long. All characters are significant, whereas under MASM 5.1, names are significant to 31 characters only. Names may consist of letters, digits, and the following 6 special characters: ? . @ _ $ % .If a period is used; it must be the first character. Names may not begin with a digit.

(d) Instruction mnemonics, directive mnemonics, register names, operator names and other words are reserved.

• Syntax of an instruction:

{label:} mnemonic   {operand { , operand} }   {; comment}

The curly brackets indicate those items that are not present or are optional in some instructions. Thus an instruction may have zero, one, or two operands.

• Operators:

The 8086/8088 Assembly language has a number of operators. An operator acts on an operand or operands to produce a value at assembly time. Examples are: + , - , *, / , DUP, and OFFSET

- Comments:

A semicolon starts a comment. A comment may follow a statement or it may be on a separate line. Multiple-line comments can be written by using the COMMENT directive. The syntax is:

COMMENT delimiter   {comment} comment

. . .

delimiter {comment }

where delimiter is any non-blank character not appearing in comment. The curly brackets indicate an item that is optional.
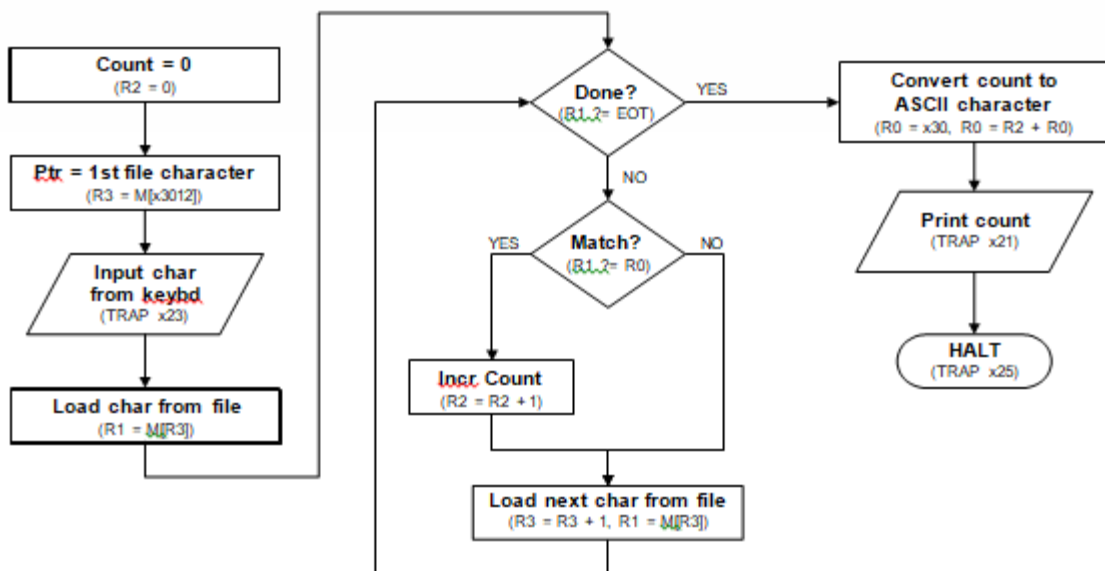
e.g.,

COMMENT *

This program finds

the maximum element in a byte array

Sample Program:

Count the occurrences of a character in a file

1. Find the .ORIG statement, which tells us the address of the first instruction.

–   Initialize   location   counter(LC), which keeps track of   the   current instruction.

2. For each non-empty line in the program:

– If line contains a label, add label and LC to symbol table.

– Increment LC.

– NOTE:If statement is .BLKW or .STRINGZ, increment LC by the number of words allocated.

3. Stop when .END statement is reached.

• NOTE: A line that contains only a comment is considered an empty line.

Second Pass: Generating Machine Language:

• For each executable assembly language statement, generate the corresponding machine language instruction.

– If operand is a label,

look up the address from the symbol table.

• Potential problems:

– Improper number or type of arguments

• Ex: NOT   R1,#7

ADD   R1, R2

ADD   R3, R3, NUMBER

– Immediate argument too large

• ex: ADD   R1,R2,#1023

– Address (associated with label) more than 256 from instruction

• can't use PC-relative addressing mode

Programming with an assembler

Using "assemble" (UNIX) or (Windows), generates several different output files.
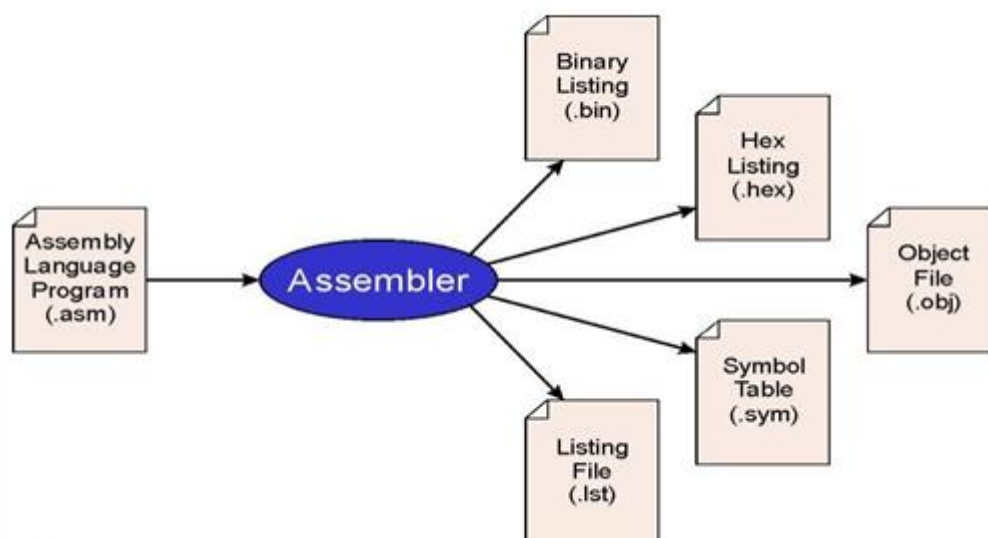
Multiple Object Files:

- An object file is not necessarily a complete program.
– system-provided library routines
– code blocks written by multiple developers
- For simulator, can load multiple object files into memory, then start executing at a desired address.
– system routines, such as keyboard input, are loaded automatically
- loaded into "system memory," below x3000
- user code should be loaded between x3000 and xFDFF
– each object file includes a starting address
– be careful not to load overlapping object files

Linking and Loading:

- Loading is the process of copying an executable image into memory.
– more sophisticated loaders are able to relocate images to fit into available memory
– must readjust branch targets, load/store addresses
- Linking is the process of resolving symbols between independent object files.
– suppose we define a symbol in one module, and want to use it in another
– some notation, such as .EXTERNAL, is used to tell assembler that a symbol is defined in another module
– linker will search symbol tables of other modules to resolve symbols and complete code generation before loading

Assembly language programs

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
```

ADD 3x3 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
```

ASCII TO HEX

```
    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
```

16 BIT 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
```

8 Bit MUL

```
    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
```

16 BIT 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
```

EVEN ODD

```
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
```

FIBONOCCI

```
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


  SORTING:

ASCENDING ORDER

            MOV AX,0000H
            MOV  CH, 0004H
            DEC CH
UP1:  MOV CL, CH
            MOV SI,2000H
UP:     MOV AL,[SI]
            INC SI
            CMP AL,[SI]
            JC DOWN
            XCHG AL,[SI]
            DEC SI
            MOV [SI], AL
            INC SI
DOWN: DEC CL
            JNZ UP
            DEC CH
            JNZ UP1
            INT 03


DESCENDING ORDER

            MOV AX,0000H
            MOV  CH, 0004H
            DEC CH
UP1:  MOV CL, CH
            MOV SI,2000H
UP:     MOV AL,[SI]
            INC SI
            CMP AL,[SI]
            JNC DOWN
            XCHG AL,[SI]
            DEC SI
            MOV [SI], AL
            INC SI
DOWN: DEC CL
            JNZ UP
            DEC CH
            JNZ UP1
            INT 03
```

EVALUATION OF ARITHMETIC EXPRESSIONS

An Assembly program for performing the following operation Z= ((A-B)/10*C)

```
DATA SEGMENT
            A DB 60
            B DB 20
            C DB 5
            Z DW?
            ENDS
            CODE SEGMENT
            ASSUME DS: DATA CS: CODE
START:          MOV AX, DATA
                MOV DS, AX
                MOV AH, 0    ; Clear content of AX
                MOV AL, A    ; Move A to register AL
                SUB AL, B    ;Subtract AL and B
                MUL C                 ; Multiply C to AL
                MOV BL, 10   ; Move 10 to register BL
                DIV BL               ; Divide AL content by BL
                MOV Z, AX    ; Move content of AX to Z
                MOV AH, 4CH
                INT 21H
            ENDS
            END START
```

An Assembly program for performing the following operation C = (A + B)^2

```
DATA SEGMENT
      A DB 60
      B DB 20
      C EQU 2000H
      ENDS
      CODE SEGMENT
      ASSUME DS: DATA CS: CODE
      START:     MOV AX, DATA
                 MOV DS, AX
                 MOV AX, A   ; Move A to Register Ax
                 ADD AX, B   ; Add B to A
                 IMUL AX     ; Square (A+B)
                 MOV [C], AX ; Move (A+B) ^2 to C in BX.
                 INT 21H
      ENDS
      END START
```

STRING MANIPULATIONS

Inserting a string

```
MOV SI,2000
MOV DI,3000
MOV BX,5000
MOV CX,0005
```

```
        CLD
L1:     MOV AL,[SI]
        CMP AL,[BX]
        JZ L2
        MOVSB
        LOOP L1
        JMP L3
L2:     MOVSB
        MOV BX,7000
        MOV AL,[BX]
        MOV [DI],AL
        DEC CX
        INC DI
        REP
        MOVSB
L3:     INT 3
```

Deleting a string

```
        MOV SI,2000
        MOV DI,3000
        MOV BX,5000
        MOV CX,0005
        CLD
L1:     MOV AL,[SI]
        CMP AL,[BX]
        JZ L2
        MOVSB
        LOOP L1
        JMP L3
L2:     INC SI
        DEC CX
        REP
        MOVSB
L3:     INT 03
```

Moving A Block Of Data From One Memory Location To Another Memory Location

```
        MOV SI, 2000
        MOV DI, 2010
        MOV CX, 0008
        REP
         MOVSB
        INT 03
```

Reverse a given string

```
        MOV SI, 2000
        MOV DI, 2008
        MOV CX, 0008
        ADD SI, 07
```

```
UP:         MOV AL,[SI]
            MOV [DI], AL
            DEC SI
            INC DI
            DEC CX
            JNZ UP
            INT 03
```

Search a number from String

```
            MOV CX,0004
            MOV AX,0000
            MOV SI,2000
            MOV BX,3000
UP:         MOV AL,[SI]
            CMP AL,[BX]
            JZ DOWN
            INC SI
            DEC CL
            JNZ UP
            MOV AH,00
            JMP L3
DOWN:       DEC CL
            MOV AH,01
            MOV [DI],AH
    L3:     INT 03
```

Introduction to stack

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last In First Out) or FILO (First In Last Out).
Mainly the following three basic operations are performed in the stack:
Push: Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
Pop: Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.
stack structure of 8086/8088
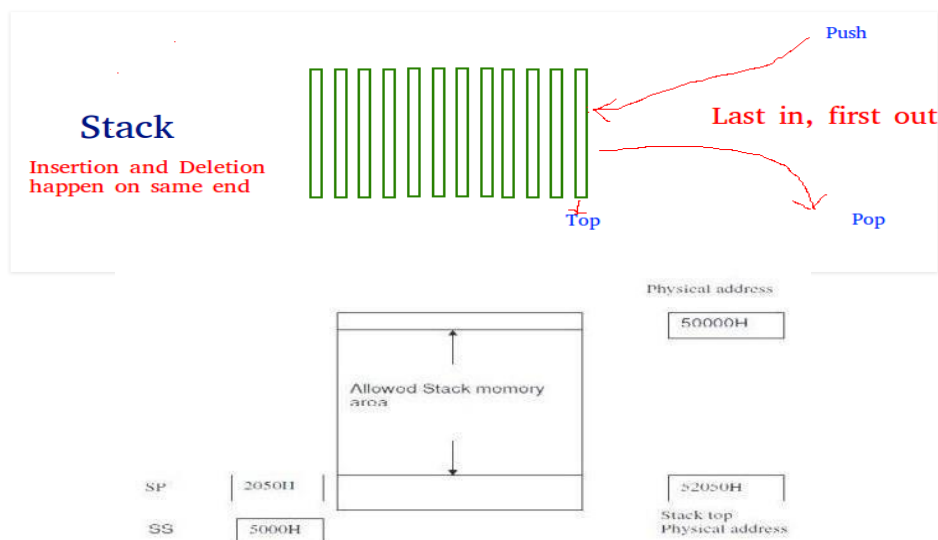
If the stack top points to a memory location 52050H, it means that the location 52050H is already occupied with the previously pushed data. The next 16 bit push operation will decrement the stack pointer by two, so that it will point to the new stack-top 5204EH and the decremented contents of SP will be 204EH. This location will now be occupied by the recently pushed data.

Thus for a selected value of SS, the maximum value of SP=FFFFH and the segment can have maximum of 64K locations. If the SP starts with an initial value of FFFFH, it will be decremented by two whenever a 16-bit data is pushed onto the stack.

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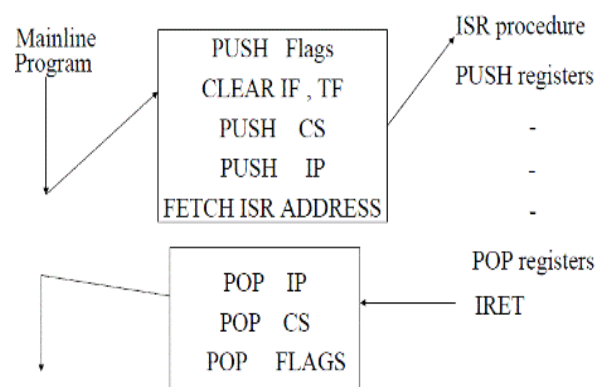
1.It decrements SP by 2 and pushes the flag register on the stack.
2.Disables INTR by clearing the IF.
3.It resets the TF in the flag Register.
4.It decrements SP by 2 and pushes CS on the stack.
5.It decrements SP by 2 and pushes IP on the stack.
6.Fetch the ISR address from the interrupt vector table.

Interrupt Vector Table
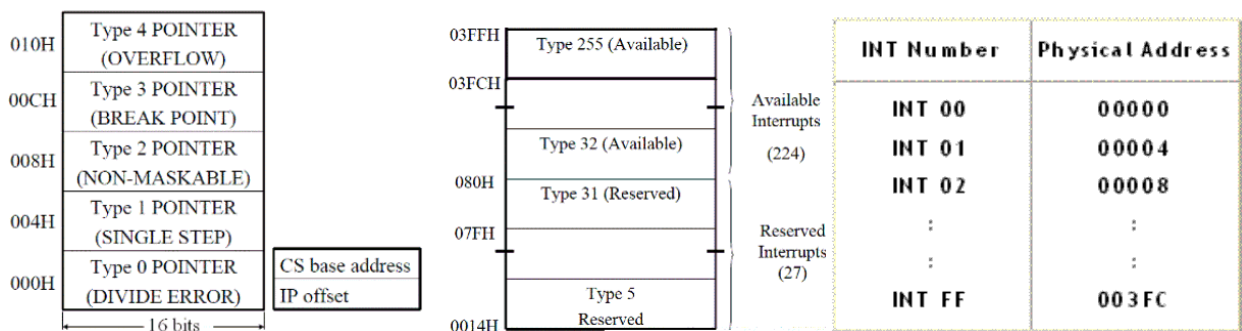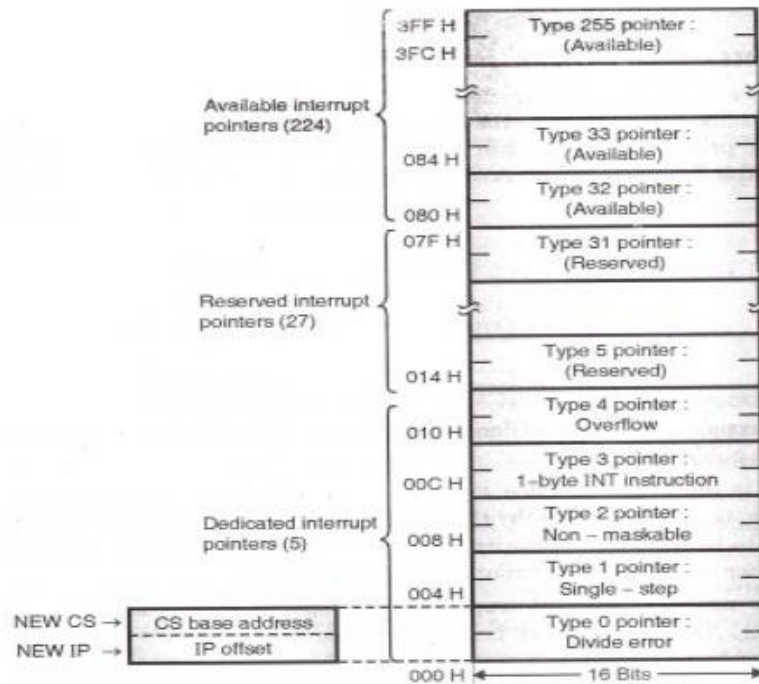
Functions associated with INT00 to INT04

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 0F = 1.

- In case where 0F = 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
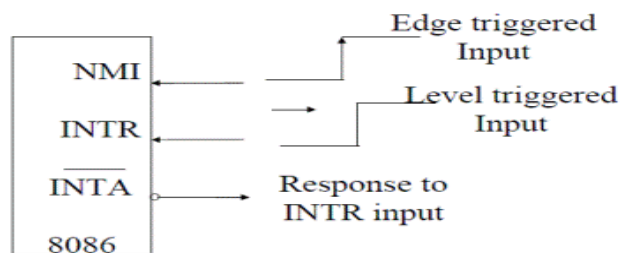
Interrupt Priority Structure

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

INTERRUPT SERVICE ROUTINE

ISR Stands for "Interrupt Service Routine." An ISR (also called an interrupt handler) is a software process invoked by an interrupt request from a hardware device. It handles the request and sends it to the CPU, interrupting the active process. When the ISR is complete, the process is resumed.

Module-III
INTERFACING WITH 8086/88

MEMORY INTERFACING TO 8086 (STATIC RAM & EPROM)

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.
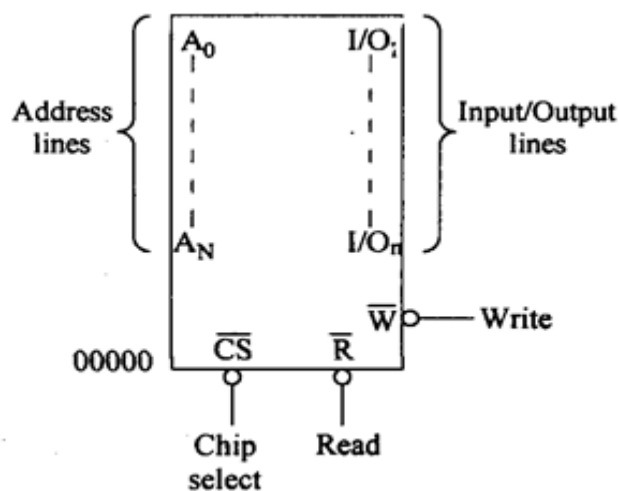
Simple Memory Device

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_O ...... 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_n or D_O ....D_n. The size of a memory location is dependent upon the number of data bits. If the numbers of data lines are eight D_0 - D_7 then 8 bits or 1 byte of data can be stored in each location. Similarly if numbers of data bits are 16 (D_0 - D_{15}) 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.

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.

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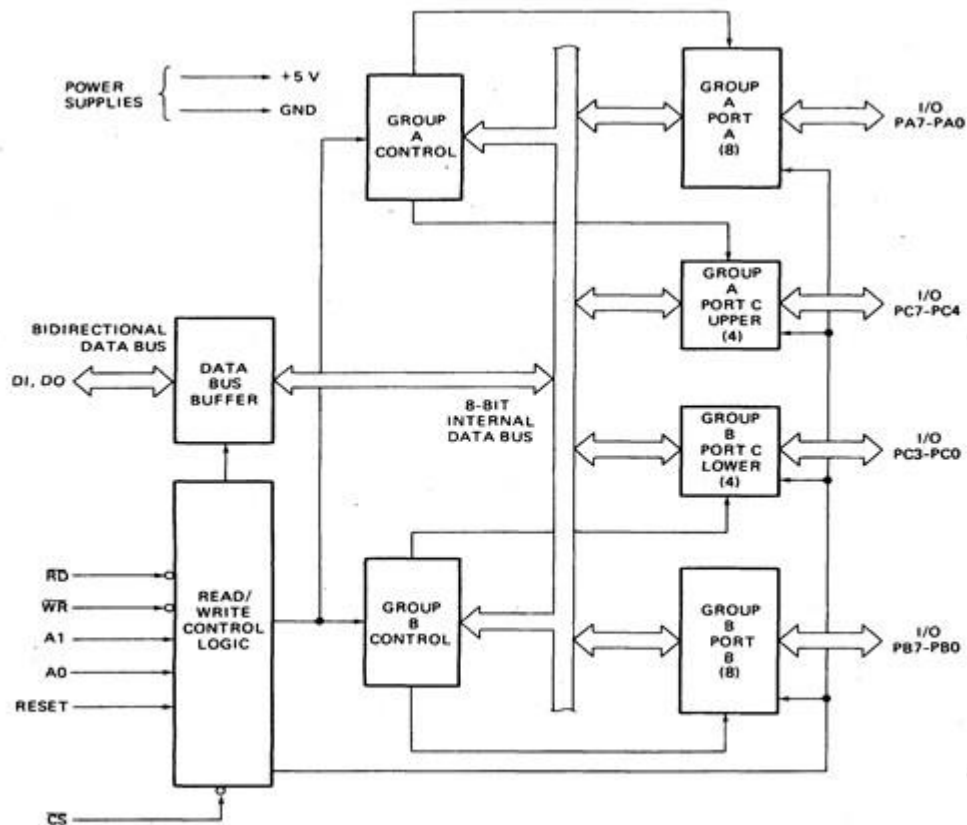
FIGURE Internal block diagram of 8255A programmable parallel port
device. (*Intel Corporation*)

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
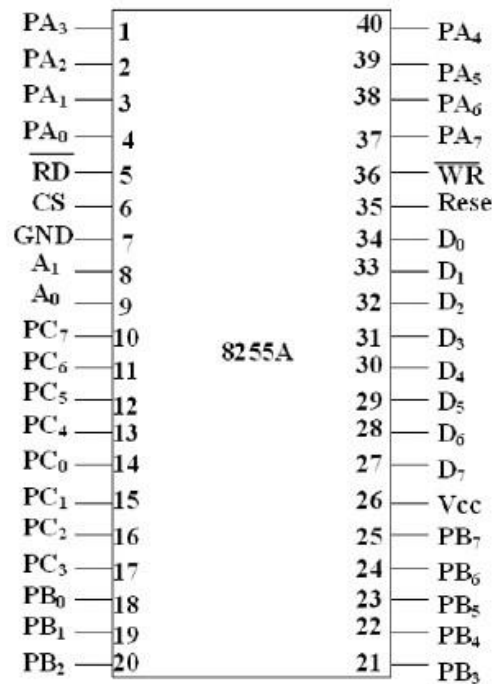
8255A Pin Configuration

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.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	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

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	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

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A_1	A_0	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_3	D_2	D_1	Selected bits of port C
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

BSR Mode : CWR Format
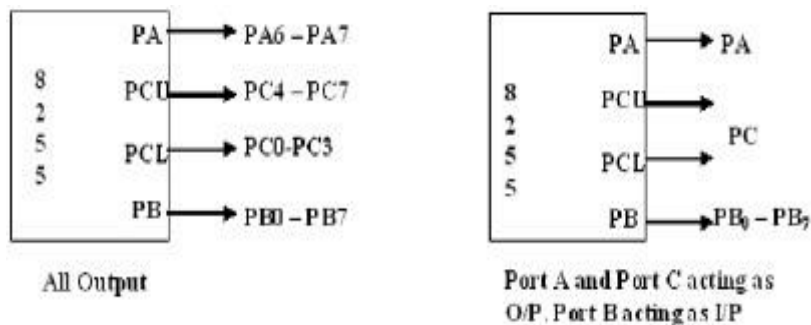
Mode 0

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 8255known 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.

| 1 | X | X | X | | | | |

0-for BSR mode

Bit select flags

0- Reset
1- Set

D_3, D_2, D_1 are from 000 to 111 for bits PC_0 TO PC_7
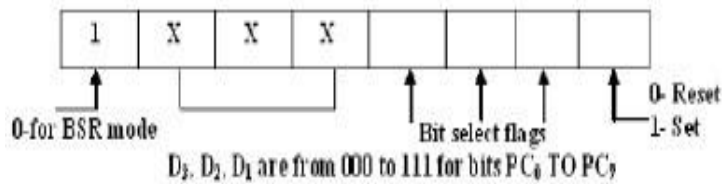
I/O Mode Control Word Register Format and
BSR Mode Control Word Register Format
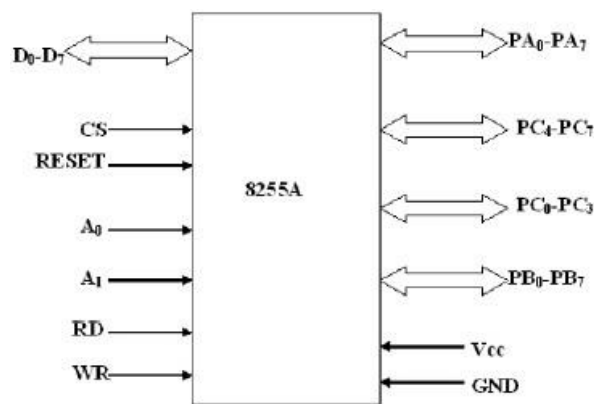
Signals of 8255
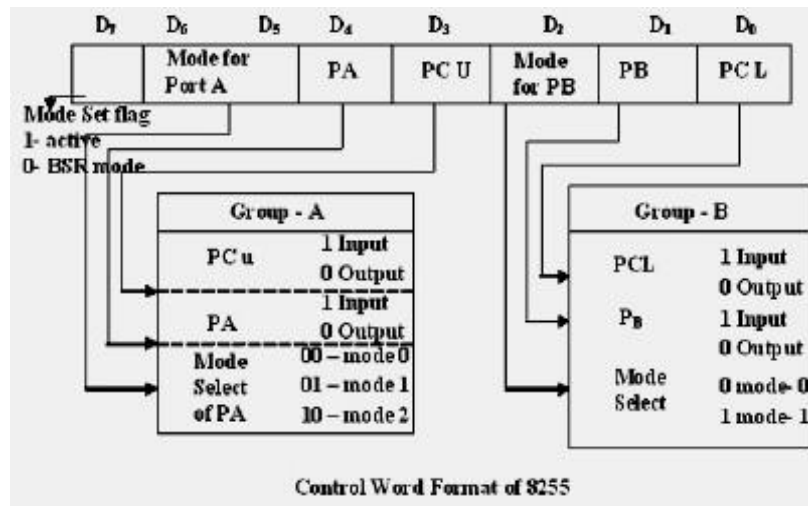
Control Word Format of 8255

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 andPC3-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 lines 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 beset 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 ofPC6 and PC2 respectively.

Input control signal definitions in Mode 1

Mode 1 Control Word Group A I/P

Mode 1 Control Word Group B I/P

Mode 1 Strobed Input Data Transfer

Mode 1 Strobed Data Output

Output control signal definitions Mode 1

1	0	1	0	1/0	X	X	X
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

1 - Input
0 - Output
For $PC_4 - PC_5$

1	X	X	X	X	1	0	X
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Mode 1 Control Word Group A   Mode 1 Control Word Group B
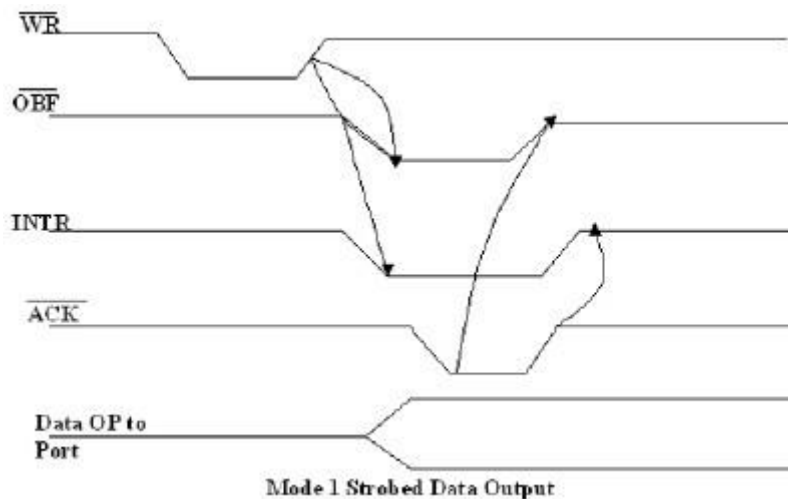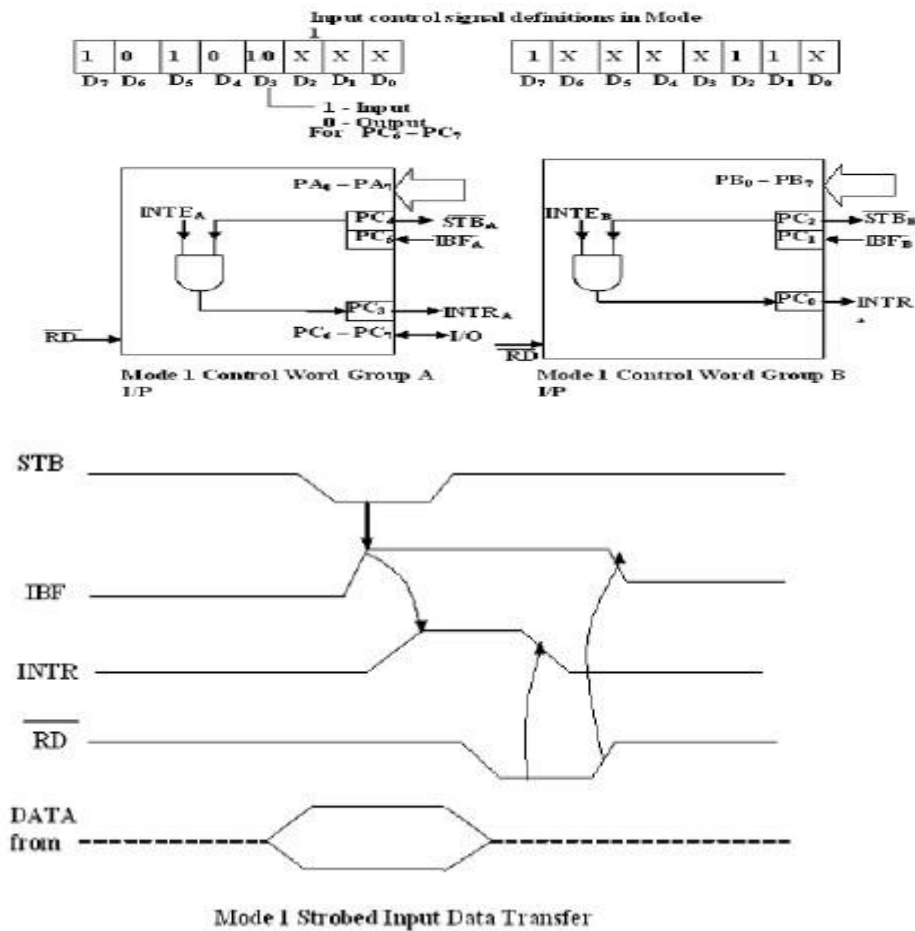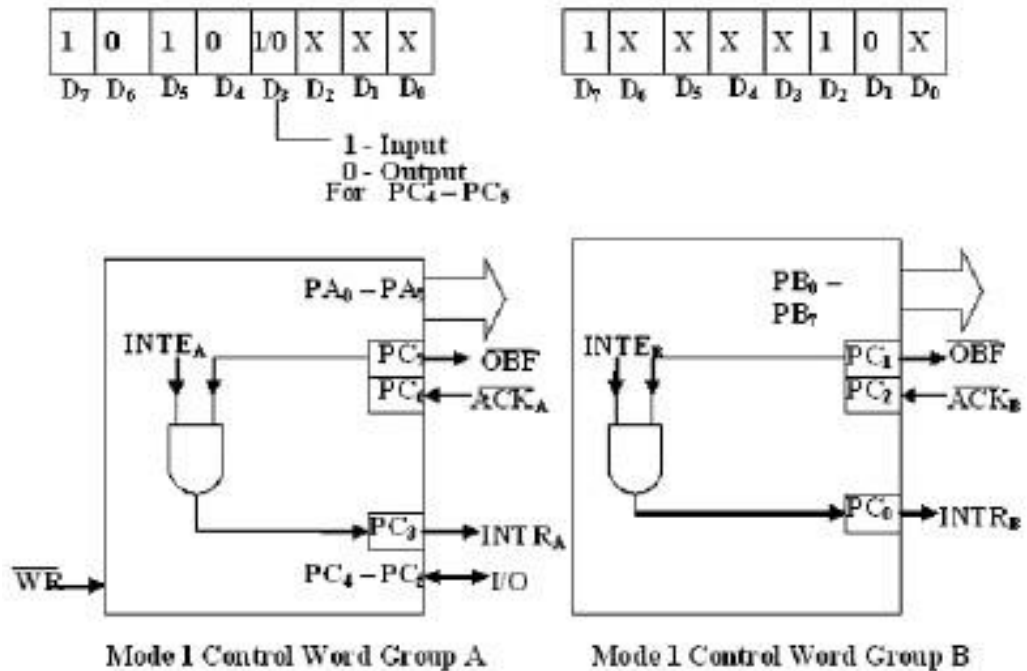
c) **Mode 2 (Strobed bidirectional I/O):** This mode of operation of 8255 is also called as strobed bidirectional I/O. This mode of operation provides 8255 with additional features for communicating with a peripheral device on an 8-bit data bus. Handshaking signals are provided to maintain proper data 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.

Mode 2 Bidirectional Data Transfer
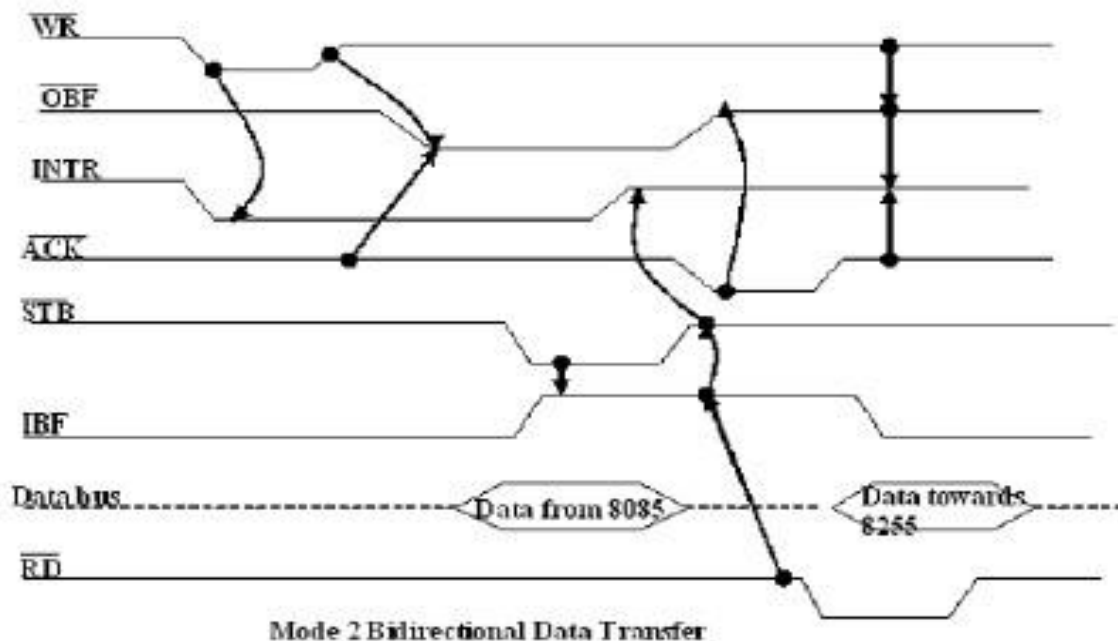
- The following fig shows a schematic diagram containing an 8-bit bidirectional port, 5-bit control port and the relation of INTR with the control pins. Port B can either be set to Mode 0 or 1 with port A(Group A) is in Mode 2.
- Mode 2 is not available for port B. The following fig shows the control 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.

Mode 2 control word

Mode 2 pins
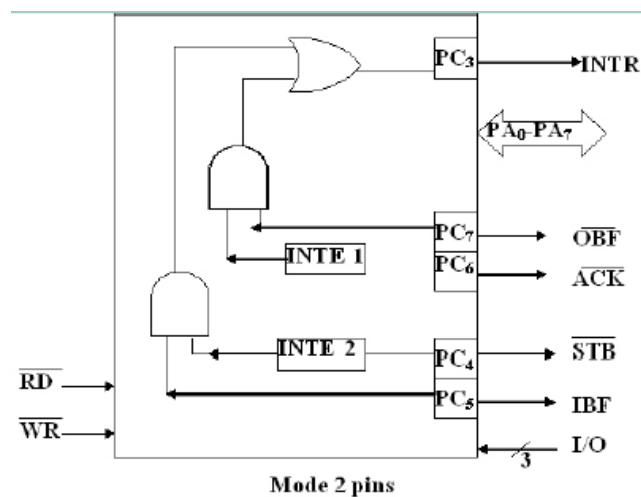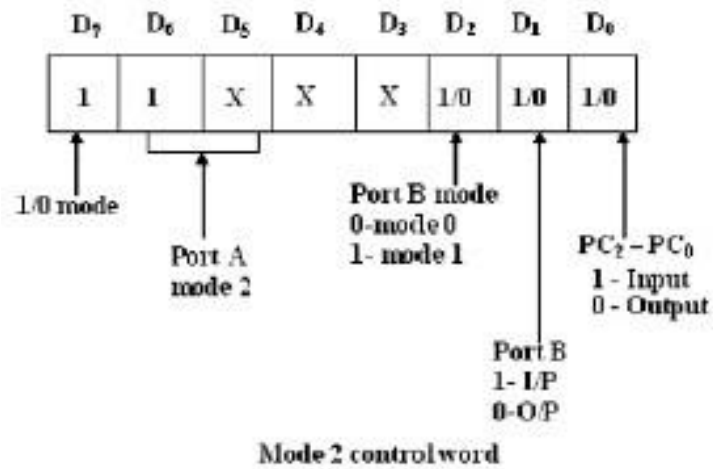
Analog to Digital Data Converters Interfacing:

- 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 100µs 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
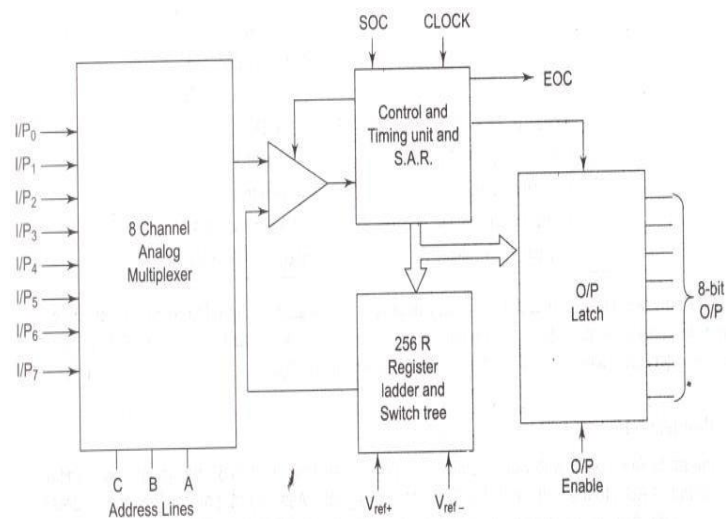
Fig.1 Block Diagram of ADC 0808/0809
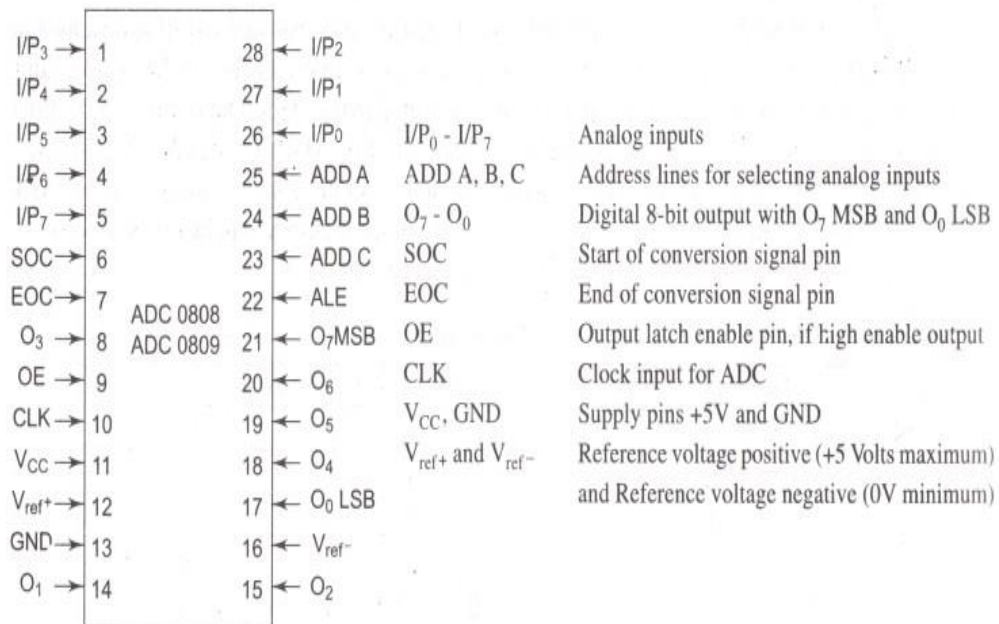
Fig.2 Pin Diagram of ADC 0808/0809

I/P$_0$ - I/P$_7$	Analog inputs
ADD A, B, C	Address lines for selecting analog inputs
O$_7$ - O$_0$	Digital 8-bit output with O$_7$ MSB and O$_0$ LSB
SOC	Start of conversion signal pin
EOC	End of conversion signal pin
OE	Output latch enable pin, if high enable output
CLK	Clock input for ADC
V$_{CC}$, GND	Supply pins +5V and GND
V$_{ref+}$ and V$_{ref-}$	Reference voltage positive (+5 Volts maximum) and Reference voltage negative (0V minimum)

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
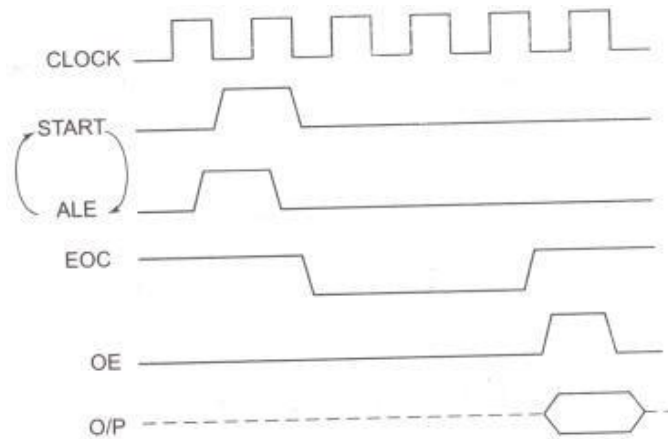
Fig.3 Timing Diagram Of ADC 0808.
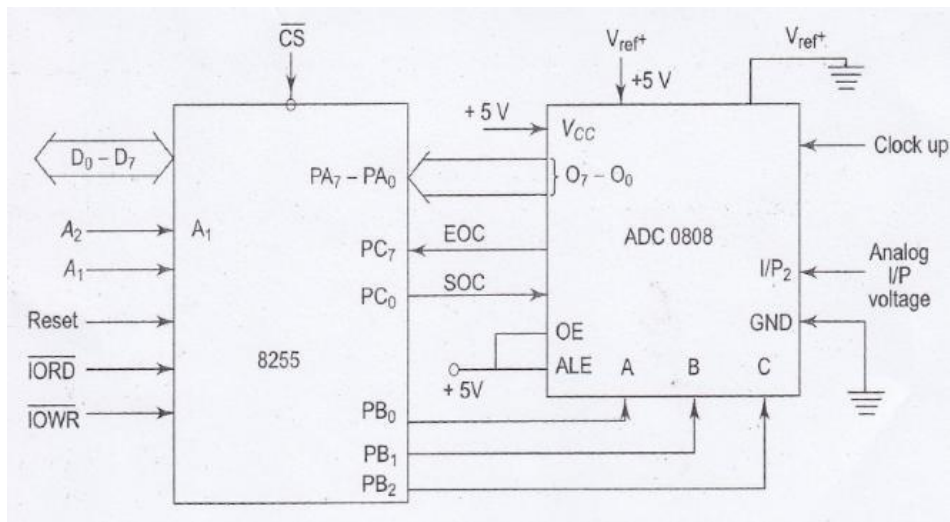
Interfacing ADC0808 with 8086

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.
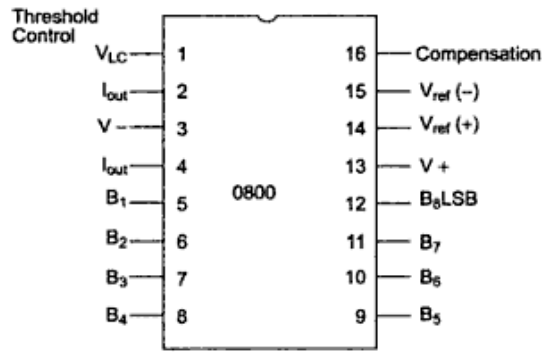
Pin Diagram of DAC 0800
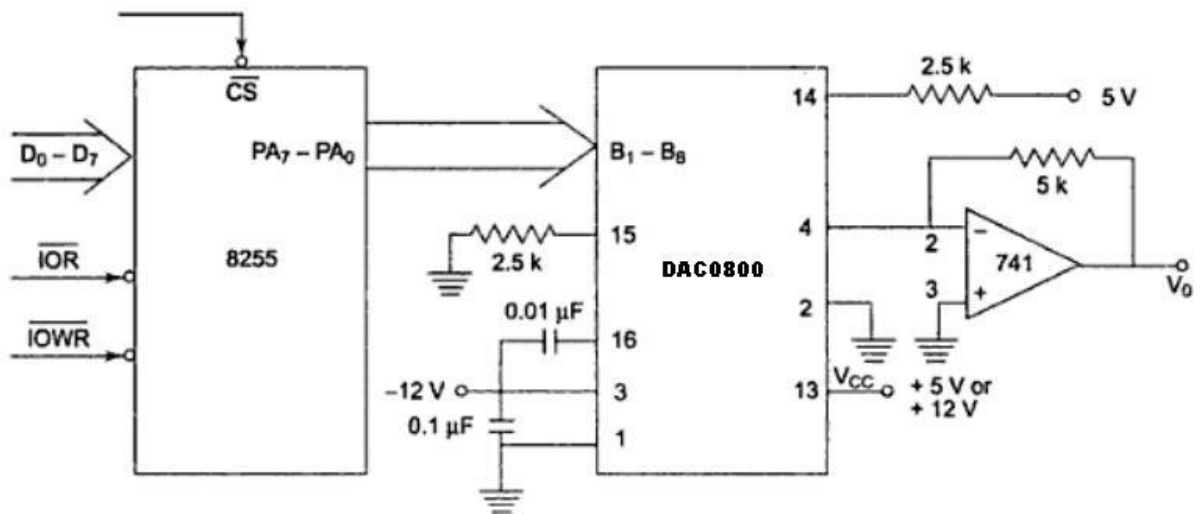
Digital to Analog Converter Interfacing

- 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.
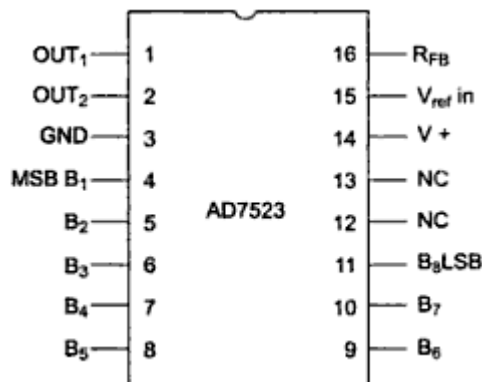
Pin Diagram of AD7523

- The supply range extends from +5V to +15V , while Vref 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.
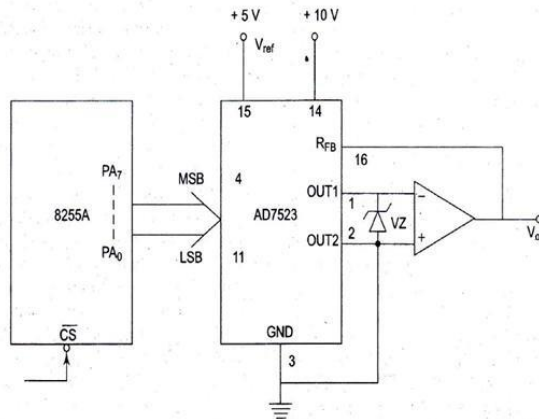
Interfacing AD7523 with 8086 Stepper

8279Stepper Motor and Actuators:
- 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:

$$X=360^{0}/\text{no. of rotor teeth}$$

- After the rotation of the shaft through angel 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.

116

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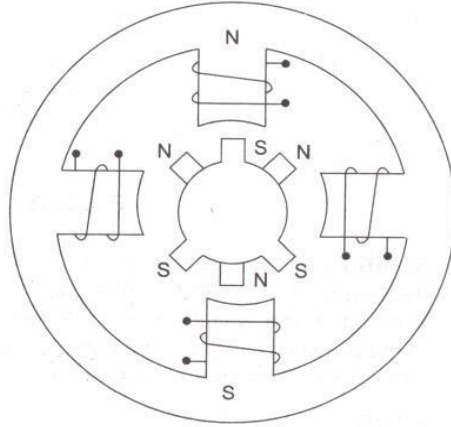
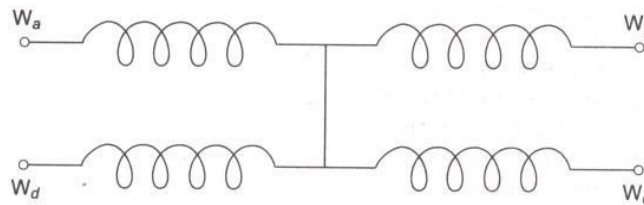**Fig.1 Internal schematic of a four winding stepper motor**

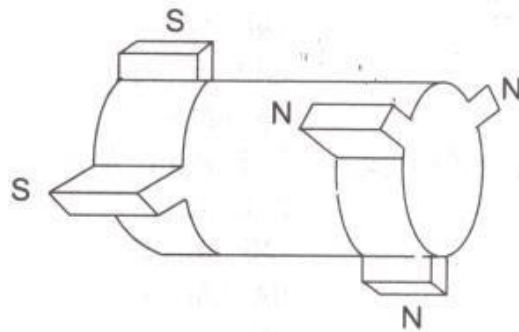**Fig.2 Winding arrangement of a stepper motor.**

Fig.3 Stepper motor rotor

- The circuit for interfacing a winding Wn 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 1.8^0 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 Wa, Wb, Wc and Wd 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.
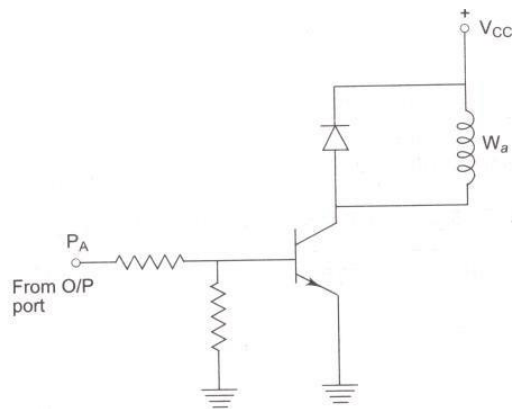
Fig.4 interfacing stepper motor winding.

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

PROGRAMMABLE INTERRUPT CONTROLLER 8259A

8259 microprocessor is defined as **Programmable Interrupt Controller (PIC)** microprocessor. There are 5 hardware interrupts and 2 hardware interrupts in 8085 and 8086 respectively. But by connecting 8259 with CPU, we can increase the interrupt handling capability. 8259 combines the multi interrupt input sources into a single interrupt output. Interfacing of single PIC provides 8 interrupts inputs from IR0-IR7.

For example, interfacing of 8085 and 8259 increases the interrupt handling capability of 8085 microprocessor from 5 to 8 interrupt levels.

Features of 8259 PIC microprocessor –

1. It is a LSI chip which manages 8 levels of interrupts i.e. it is used to implement 8 level interrupt systems.
2. It can be cascaded in a master slave configuration to handle up to 64 levels of interrupts.
3. It can identify the interrupting device.
4. It can resolve the priority of interrupt requests i.e. it does not require any external priority resolver.
5. It can be operated in various priority modes such as fixed priority and rotating priority.
6. The interrupt requests are individually mask-able.
7. The operating modes and masks may be dynamically changed by the software at any time during execution of programs.
8. It accepts requests from the peripherals, determines priority of incoming request, checks whether the

119

incoming request has a higher priority value than the level currently being serviced and issues an interrupt signal to the microprocessor.

9. It provides 8 bit vector number as an interrupt information.
10. It does not require clock signal.
11. It can be used in polled as well as interrupt modes.
12. The starting address of vector number is programmable.
13. It can be used in buffered mode.

BLOCK DIAGRAM OF 8259 PIC MICROPROCESSOR
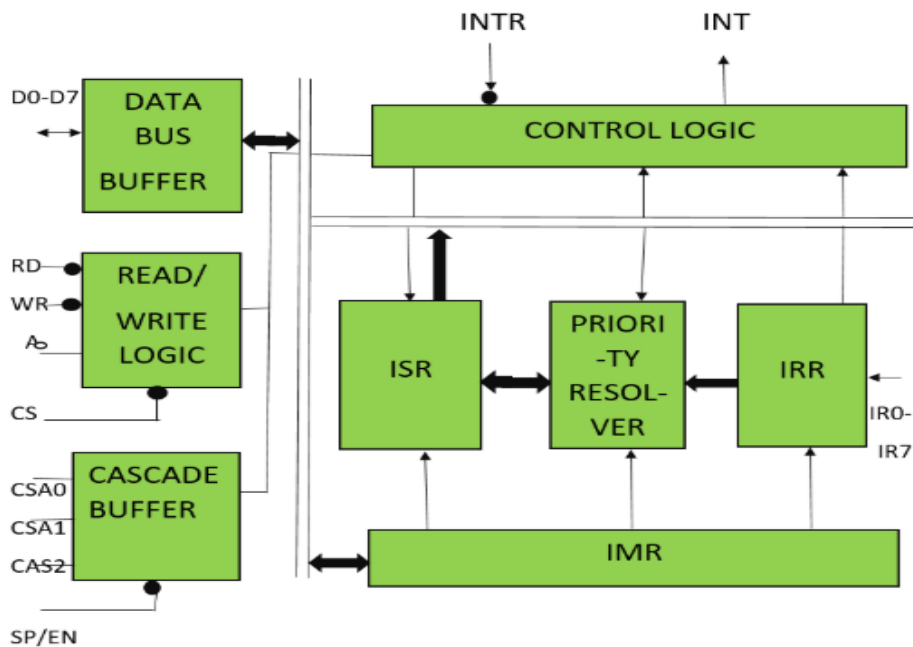
The Block Diagram consists of 8 blocks which are – Data Bus Buffer, Read/Write Logic, Cascade Buffer Comparator, Control Logic, Priority Resolver and 3 registers- ISR, IRR, IMR.

1. **Data bus buffer –**

   This Block is used as a mediator between 8259 and 8085/8086 microprocessor by acting as a buffer. It takes the control word from the 8085 (let say) microprocessor and transfer it to the control logic of 8259 microprocessor. Also, after selection of Interrupt by 8259 microprocessor, it transfer the opcode of the selected Interrupt and address of the Interrupt service sub routine to the other connected microprocessor. The data bus buffer consists of 8 bits represented as D0-D7 in the block diagram. Thus, shows that a maximum of 8 bits data can be transferred at a time.

2. **Read/Write logic –**

   This block works only when the value of pin CS is low (as this pin is active low). This block is

responsible for the flow of data depending upon the inputs of RD and WR. These two pins are active low pins used for read and write operations.

3. **Control logic –**

It is the centre of the microprocessor and controls the functioning of every block. It has pin INTR which is connected with other microprocessor for taking interrupt request and pin INT for giving the output. If 8259 is enabled, and the other microprocessor Interrupt flag is high then this causes the value of the output INT pin high and in this way 8259 responds to the request made by other microprocessor.

4. **Interrupt request register (IRR) –**

It stores all the interrupt level which are requesting for Interrupt services.

5. **Interrupt service register (ISR) –**

It stores the interrupt level which are currently being executed.

6. **Interrupt mask register (IMR) –**

It stores the interrupt level which have to be masked by storing the masking bits of the interrupt level.

7. **Priority resolver –**

It examines all the three registers and set the priority of interrupts and according to the priority of the interrupts, interrupt with highest priority is set in ISR register. Also, it reset the interrupt level which is already been serviced in IRR.

8. **Cascade buffer –**

To increase the Interrupt handling capability, we can further cascade more number of pins by using cascade buffer. So, during increment of interrupt capability, CSA lines are used to control multiple interrupt structure.

SP/EN (Slave program/Enable buffer) pin is when set to high, works in master mode else in slave mode. In Non Buffered mode, SP/EN pin is used to specify whether 8259 work as master or slave and in Buffered mode, SP/EN pin is used as an output to enable data bus.

```
CS    → 1          28 ← Vcc
WR    → 2          27 ← A₀
RD    → 3          26 ← INTA
D₇    ↔ 4          25 ← IR₇
D₆    ↔ 5          24 ← IR₆
D₅    ↔ 6          23 ← IR₅
D₄    ↔ 7          22 ← IR₄
D₃    ↔ 8          21 ← IR₃
D₂    ↔ 9          20 ← IR₂
D₁    ↔ 10         19 ← IR₁
D₀    ↔ 11         18 ← IR₀
CAS₀  ↔ 12         17 → INT
CAS₁  ↔ 13         16 ↔ SP/EN
Gnd   → 14         15 ↔ CAS₂
```
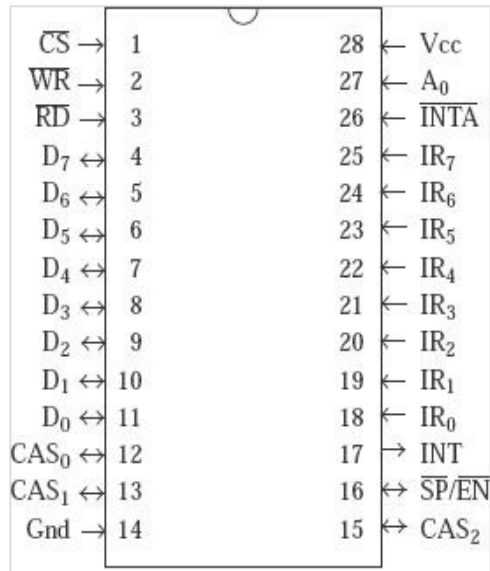
V$_{cc}$ and Gnd:It is the Power supply and ground pins. +5V power supply is used in this chip.

D7-D0:For communication with the processor, there are Eight bi-directional data pins.

RD*:It is active low-input pin activated by the processor to read the information status from the 8259.

WR*: It is an active low-input pin which is activated by the processor to write the control information to 8259.

CS*: For selecting the chip it is used .It is an active low input pin.

A0: An address input pin used along with RD* and WR* which is used to identify the various command words within 8259.

IR0-IR7: There are Eight asynchronous interrupt request inputs. These interrupt requests can be programmed for level-trigger or edge-triggered mode.

INT: A strong active high-output pin which interrupts the processor. Always connected to the INTR interrupt input of 8086. The process is interrupted whenever the interrupting device delivers a signal to 8259.

INTA*:It is termed as an active low-input pin. The 8259 receives the signal from INTA* to the output of 8086. 8086 sends the three consecutive INTA* signals, the 8259 sends a 3-byte CALL instruction to the 8086 via D$_{7-0}$ pins(Data Bus). The two bytes termed as second and third bytes of the CALL instruction contains the ISR address which depends on the IR input of 8259 that is going to be serviced.

CAS2-0:These are cascaded lines. Used only when there are multiple 8259s in the system. The interrupt control system might have a master 8259 and maximum eight Slave 8259s.

SP*/EN*:SP*/EN* stands for "slave program/enable buffer". This pin serves dual function. When

122

the 8259A is in buffered mode, this pin is an output that controls the data bus transceivers in a large microprocessor-based system. When the 8259A is not in buffered mode, this pin programs the device as a master (1) or a slave (0).

Command Words of 8259A

⦿ **Initialization Command Words(ICWs)**

⦿ **Operation Command Words(OCWs)**

The Programming 8259 requires two types of command words. Initialization Command Words (ICWs) and Operational Command Words (OCWs).

The Programming 8259 can be initialized with four ICWs; the first two are compulsory, and the other two are optional based on the modes being used. These words must be issued in a given sequence. If we want to change any 1 bit, all the four ICWs should be initialised again.

After initialization, the 8259A can be set up to operate in various modes by using three different OCWs; however, they are not necessary to be issued in a specific sequence.

Initialization sequence (ICWs) of 8259A
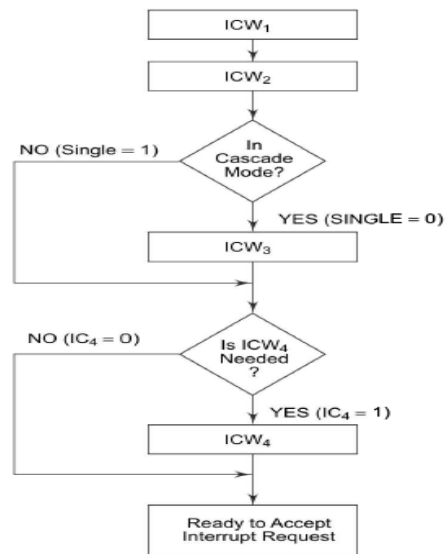
ICWs

ICW1 : A write command issued to the 8259 with $A_0 = 0$ and $D_4 = 1$ is interpreted as ICW1, which starts the initialization sequence.
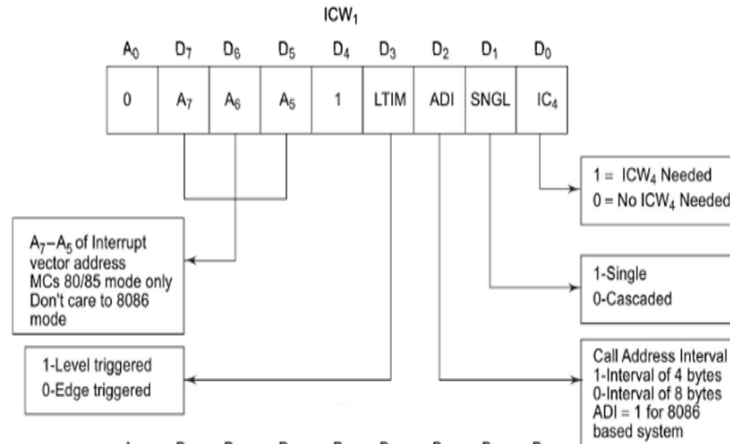
ICW₁

ICW2: A write command following ICW1, with $A_0 = 1$ is interpreted as ICW2. ICW2 is used to initialise interrupt vector number. A8, A9, A10 should be compulsory 000
IR0 selected should be like 40,48,50,58 etc.Hence last 3 bits will be zero and most significant bits are filled in T3 to T7.

ICW₂
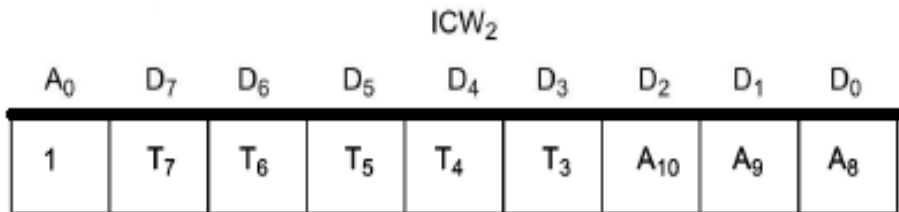
T_7–T_3 – For 8085 system they are filled by A_{15}–A_{11} of the interrupt vector address and the least significant 3 bits are same as the respective bits of vector address. For 8086 system they are filled by most significant 5 bits of interrupt type and the least significant 3 bits are 0, pointing to Ir₀

ICW3:ICW3 is read only when there are more than one 8259As in the system.i.e; cascading is used .In ICW1 if SNGL=0 cascading is used .
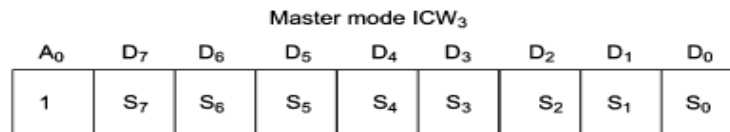
124
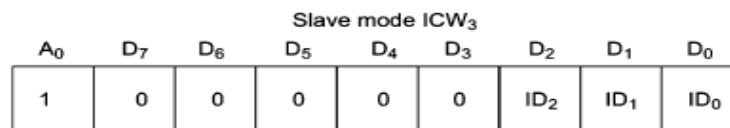
Master: This says to the master where the slave is present.
If S0=I; To IR0 slave is connected.
If S5=I; To IR5 slave is connected.

Slave: Slave identification
Since 8 slaves are possible ID0, ID1, ID2 are required.
ID0, ID1, ID2=000; Slave 0 is identified.

ICW4: The use of this command word depends on the IC4 bit of ICW1.If IC4=1, ICW4 is used. In 8086 ICW4 is compulsory. In 8085 it is optional.

SFNM: If SFNM=1 Special Fully Nested Mode is selected.

BUF: If BUF=1 the buffered mode is selected.
If BUF=0 non buffered mode is selected.

M/S: If M/S=1 8259A is a master.
If M/S=0 8259A is a slave.
AEOI: If AEOI=1 Automatic end of interrupt mode is selected.
If AEOI=0 end of interrupt mode is selected.

mPM: If mPM=1 8086/88 operation is selected.
If mPM=0 8085 operation is selected.

ICW4

	ICW$_4$							
A$_0$	D$_7$	D$_6$	D$_5$	D$_4$	D$_3$	D$_2$	D$_1$	D$_0$
1	0	0	0	SFNM	BUF	M/$\overline{S}$	AEOI	mPM

OCWs

OCW1: It says which interrupt should be masked. If M3=1Interrupt connected to this pin will be masked.i.e;not accepted. If M3=0 unmask.

OCW$_1$

1-Mask Set
0-Mask Reset

(a)
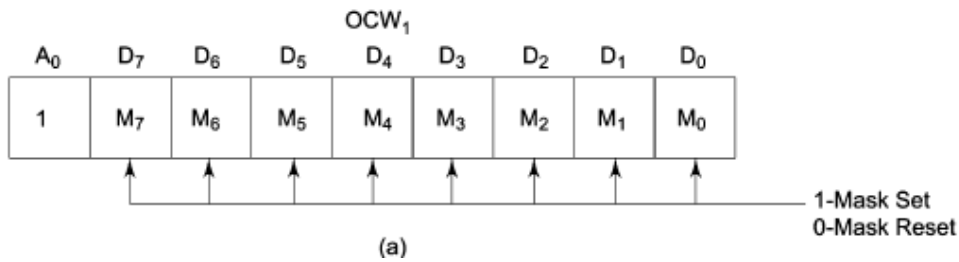
OCW2: If L0, L1, L2=000 IR0 is selected.
R (Rotate), SL (Select-Level), EOI (End Of Interrupt Modes)

OCW$_2$

END OF INTERRUPT	0	0	1	NON-SPECIFIC EOI COMMAND
	0	1	1	SPECIFIC EOI COMMAND
AUTOMATIC ROTATION	1	0	1	ROTATE ON NONSPECIFIC EOI COMMAND
	1	0	0	ROTATE IN AUTOMATIC EOI MODE (SET)
	0	0	0	ROTATE IN AUTOMATIC EOI (CLEAR)
SPECIFIC ROTATION	1	1	1	ROTATE ON SPECIFIC EOI COMMAND
	1	1	0	SET PRIORITY COMMAND*
	0	1	0	NO OPERATION
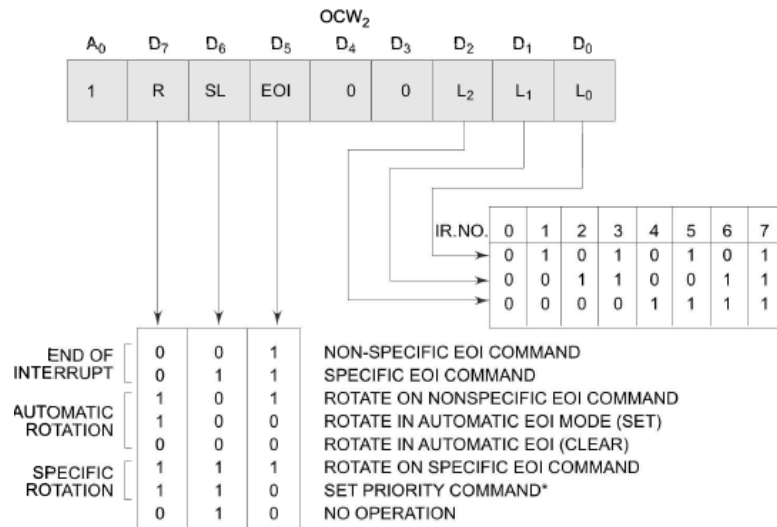
OCW3:

SMM: Special Mask mode
ESMM: Enable Special Mask mode
RR: Read IRR
RIS: Read ISR
P: Polling mode

OCW$_3$

(c)
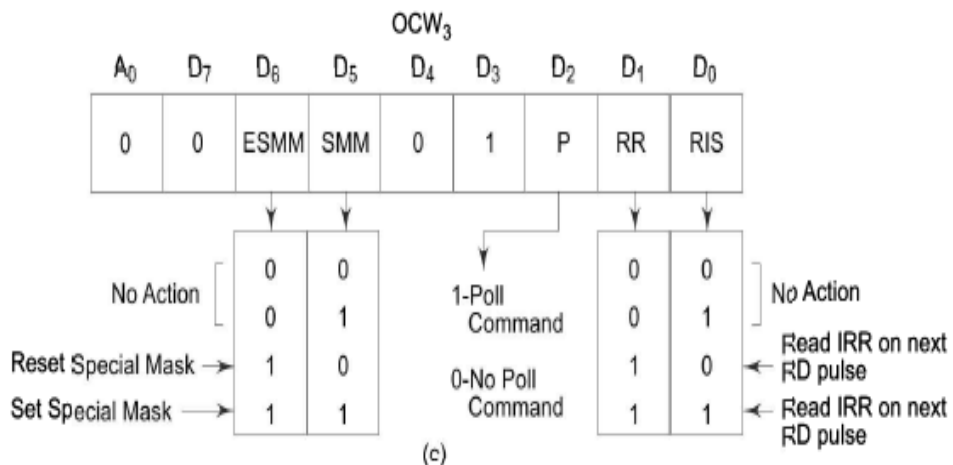
CASCADING OF INTERRUPT CONTROLLER AND ITS IMPORTANCE

231468-24
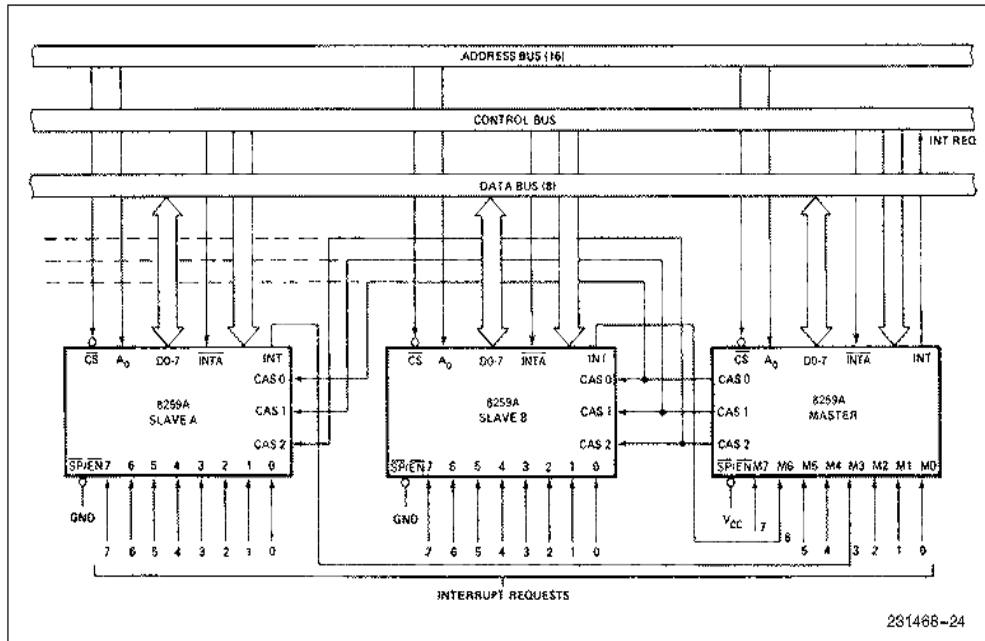
Cascading is necessary to connect more number of Devices which produces interrupts. The cascade pins (CAS0, CAS1 and CAS2) from the master are connected to the corresponding pins of the slave. For the slave 8259, the SP (low) / EN (low) pin is tied low to let the device know that it is a slave. The SP (low) / EN (low) pin can be used as input or output signal.

In non-buffered mode it is used as input signal and tied to logic-I in master 8259 and logic-0 in slave 8259.In buffered mode it is used as output signal to disable the data buffers while data is transferred from 8259A to the CPU.

CAS2-CAS0 (Cascade lines):
The CAS2-0 lines form a local 8259A bus to control multiple 8259As in master-slave configuration, i.e., to identify a particular slave 8259A to be accessed for transfer of vector information. These pins are automatically set as output pins for master 8259A and input pins for a slave 8259A once the chips are programmed as master or slave

Cascading of interrupt controller importance. The 8259A can be easily interconnected in a system of one master with up to eight slaves to handle up to 64 priority levels. The master controls the slaves through the 3 line cascade bus. The cascade bus acts like chip selects to the slaves during the INTA sequence. In a cascade configuration, the slave interrupt outputs are connected to the master interrupt request inputs.

Interfacing 8259A with 8086
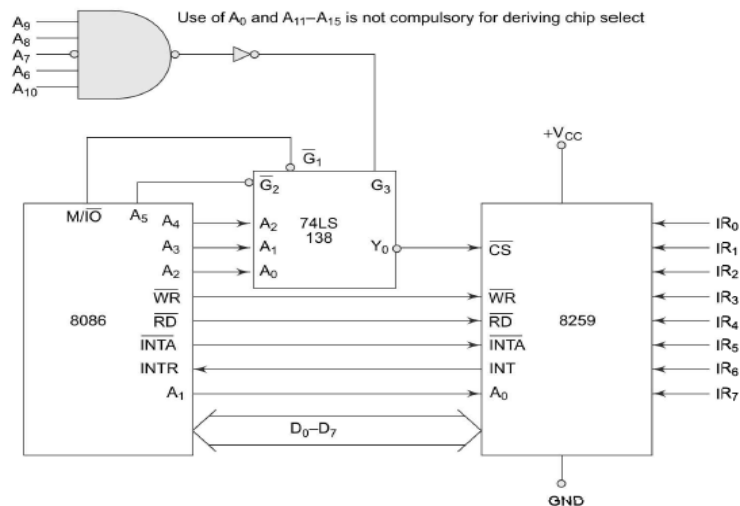
Operating modes of 8259A

FULLY NESTED MODE:
General purpose mode / default mode.
IR0 to IR7 are arranged from highest to lowest.
IR0 -> Highest ;IR7 -> Lowest

AUTOMATIC ROTATION MODE:
In this mode, a device after being serviced, receives the lowest priority.

SPECIFIC ROTATION MODE:
Similar to automatic rotation mode, except that the user can select any IR for the lowest priority, thus fixing all other priorities

END OF INTERRUPT (EOI):
After the completion of an interrupt service, the corresponding ISR bits needs to be reset to update the information in the ISR. This is called EOI command.
NON SPECIFIC EOI COMMAND:(manually)
When this command is sent to 8259A, it resets the highest priority ISR bit.

SPECIFIC EOI COMMAND:(manually)
This command specifies which ISR bit is to reset.

AUTOMATIC EOI:
In this mode, no command is necessary.
During the end of interrupt acknowledge cycle, the ISR bit is reset.
Used for master only

SPECIAL MASK MODE

When a mask bit is set in OCW, it inhibits further interrupts at that level and enables interrupt from other levels, which are not mastered.

EDGE AND LEVEL TRIGGERED MODE

Decides whether the interrupt should be edge triggered or level triggered.
If bit LTIM of ICW1=0, they are edge triggered, otherwise level triggered.

READING 8259A STATUS

Used to read the status of the internal registers of 8259A.
Reading is possible only in no polled mode.
OCW3, is used to read IRR and ISR and OCW1 for IMR.

POLL COMMAND

The INT output is neglected, though it functions normally by not connecting INT output or by masking INT input of the microprocessor.
This mode is entered by setting p=1 in OCW3.
A poll command may give more than 64 priority levels.

SPECIAL FULLY NESTED MODE

Used in more complicated systems.
Similar to, normal nested mode.

BUFFERED MODE

When the 8259A is used in the system in which bus driving buffers are used on the data buses, the problem of enabling the buffers arises.
The 8259A sends a buffer enable signal on SP'/EN' pin.

CASCADE MODE

The slave INT outputs are connected with master IR inputs. When a slave request line is activated and acknowledged, the master will enable the slave to release the vector addresses during the second pulses of INTA sequence.
The cascade lines are normally low and contain slave addresses codes from the trailing edge of the first INTA pulse to the trailing edge of the second INTA pulse.

SERIAL DATA TRANSFER SCHEMES
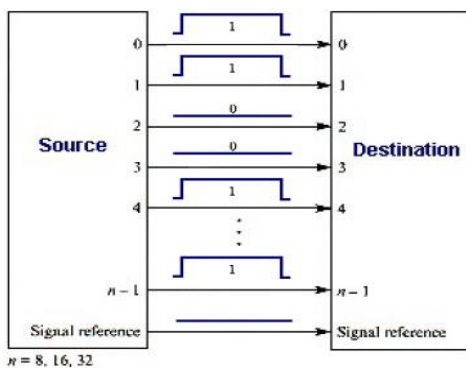## SERIAL COMMUNICATION
INTRODUCTION:

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

129

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**

130
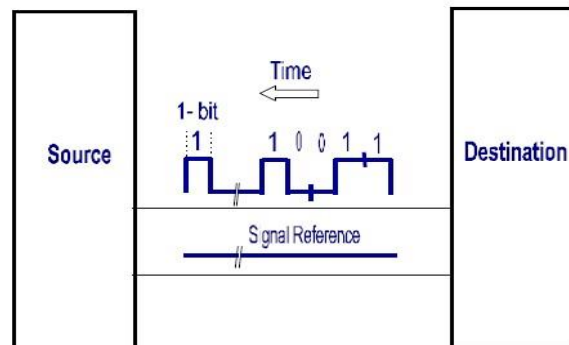
Even in shorter distance communications, serial computer buses are becoming more common because of a tipping point where the disadvantages of parallel busses (clock skew, interconnect density) outweigh their advantage of simplicity. The serial port on your PC is a full-duplex device meaning that it can 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 IC s 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,

131

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
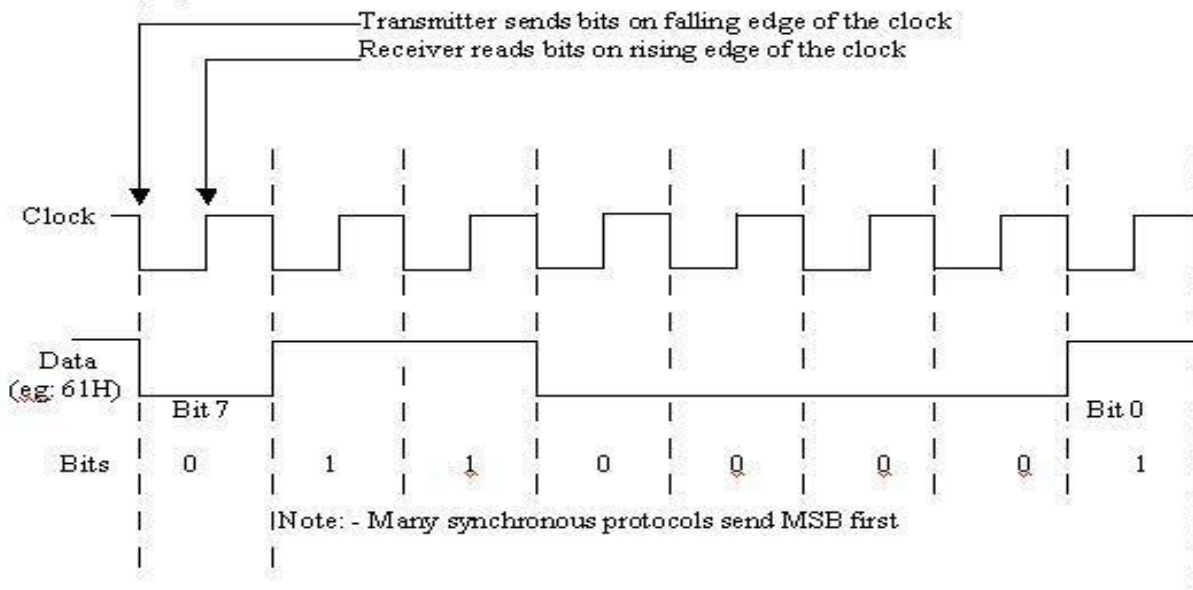
2. Asynchronous data transmission

ASYNCHRONOUS AND SYNCHRONOUS DATA TRANSFER SCHEMES

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.

1) Synchronous Transmission: -

Transmitter sends bits on falling edge of the clock
Receiver reads bits on rising edge of the clock

Clock

Data
(eg: 61H)

Bit 7

Bit 0

Bits   0   1   1   0   0   0   0   1

Note: - Many synchronous protocols send MSB first

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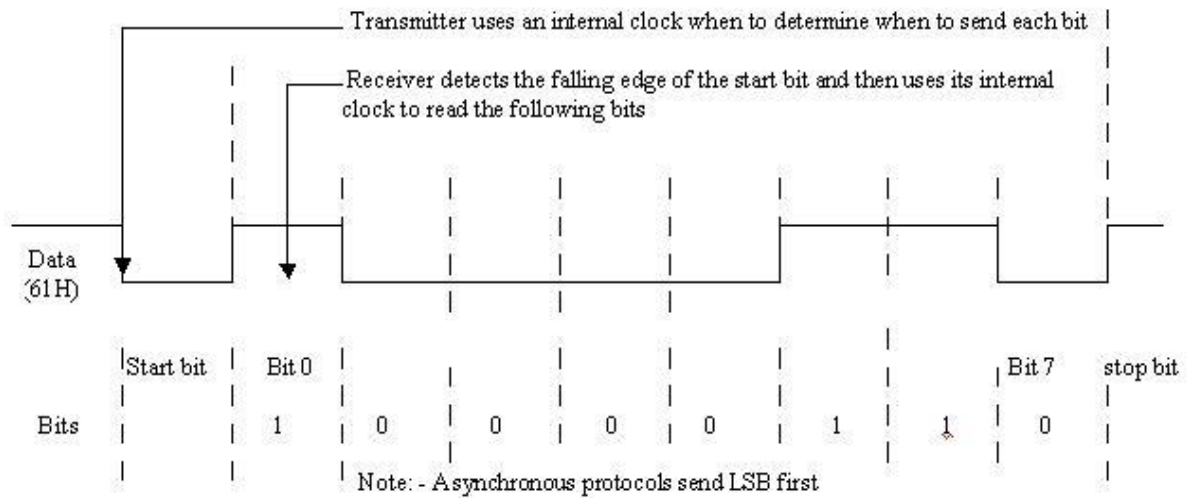
Disadvantages:

- 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 are 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.

133

Transmitter uses an internal clock when to determine when to send each bit

Receiver detects the falling edge of the start bit and then uses its internal clock to read the following bits

Data (61 H)

	Start bit	Bit 0							Bit 7	stop bit
Bits		1	0	0	0	0	1	1	0	

Note: - Asynchronous protocols send LSB first

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.

8251 USART ARCHITECTURE AND INTERFACING
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

134

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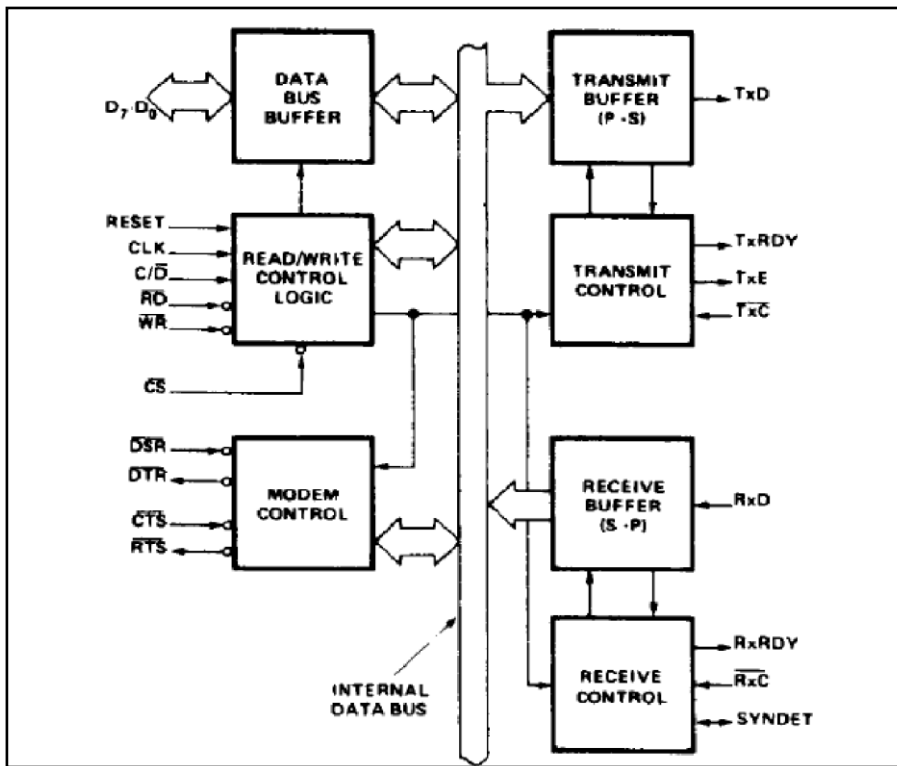
Data Bus Buffer: This bidirectional, 8-bit buffer used to interface the 8251A to the system data bus and also used to read or write status, command word or data from or to the 8251A.

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.

- 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 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 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 OF 8251A
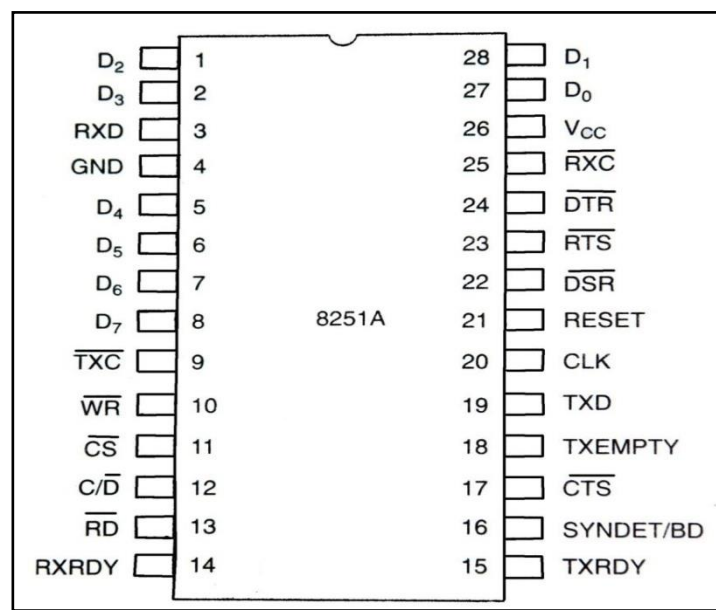
D0 to D7 (l/O terminal): This is bidirectional data bus which receives control words and transmits data from the CPU and sends status words and received data to CPU.

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

137

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.

$\overline{WR}$ **(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.

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

C/ $\overline{D}$ (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.

$\overline{CS}$ **(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 8251is 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 or 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
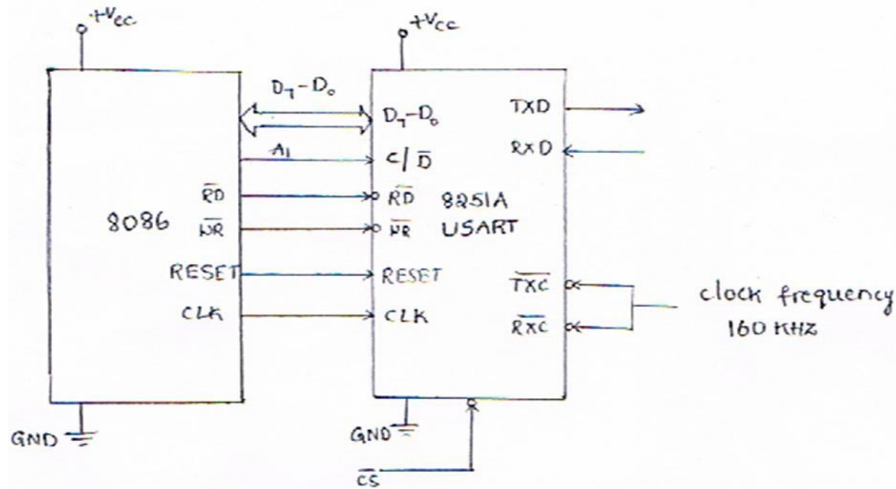
PROGRAMMING THE 8251A

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.
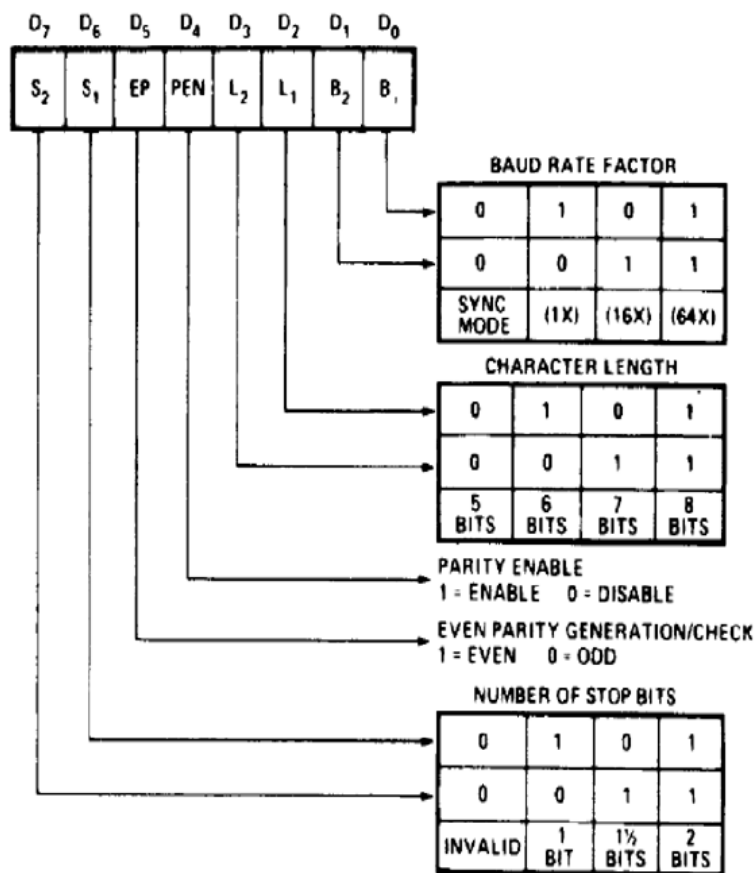

Fig. Mode instruction format, Asynchronous mode
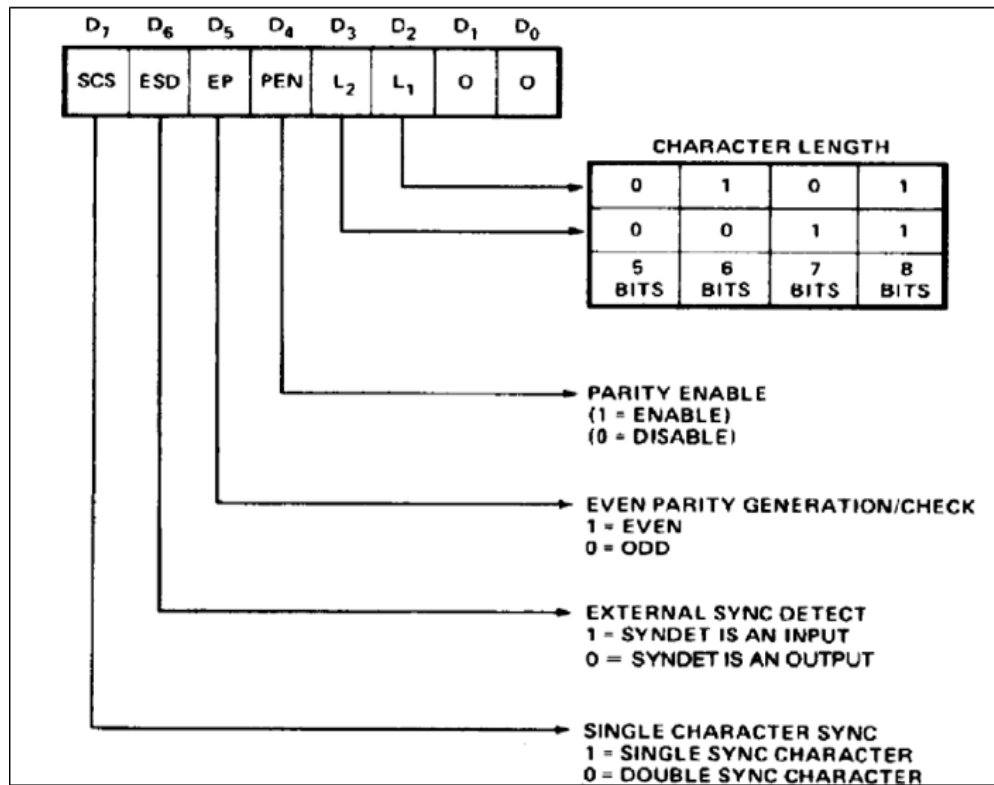
Fig. Mode instruction format, Synchronous mode

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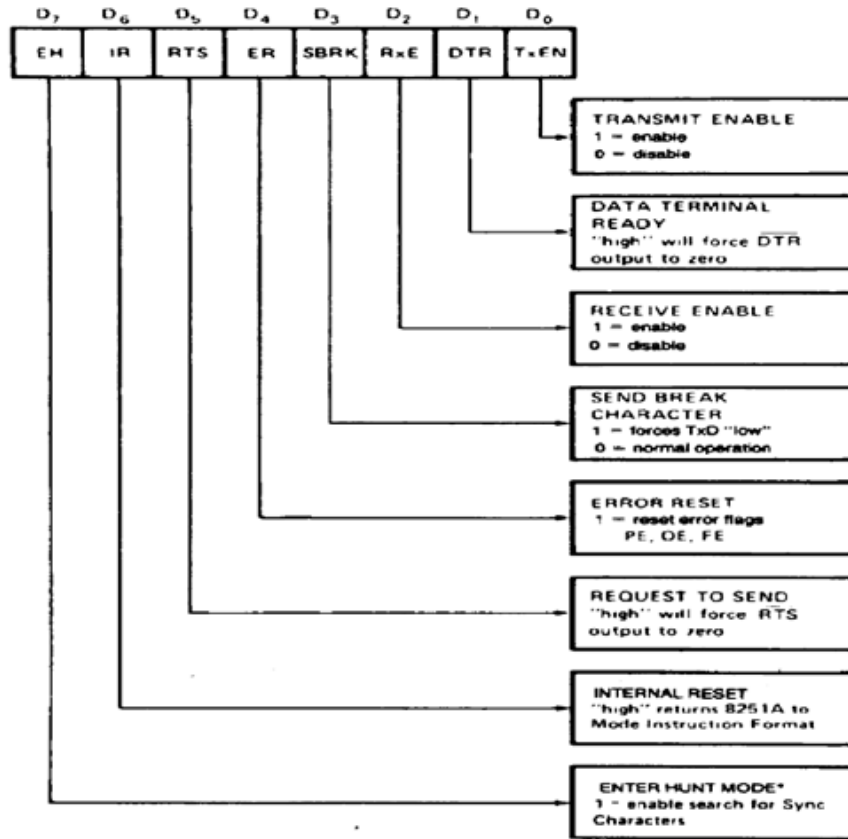
Status Word: It is possible to see the internal status of the 8251 by reading a status word. The format of status word is shown below.
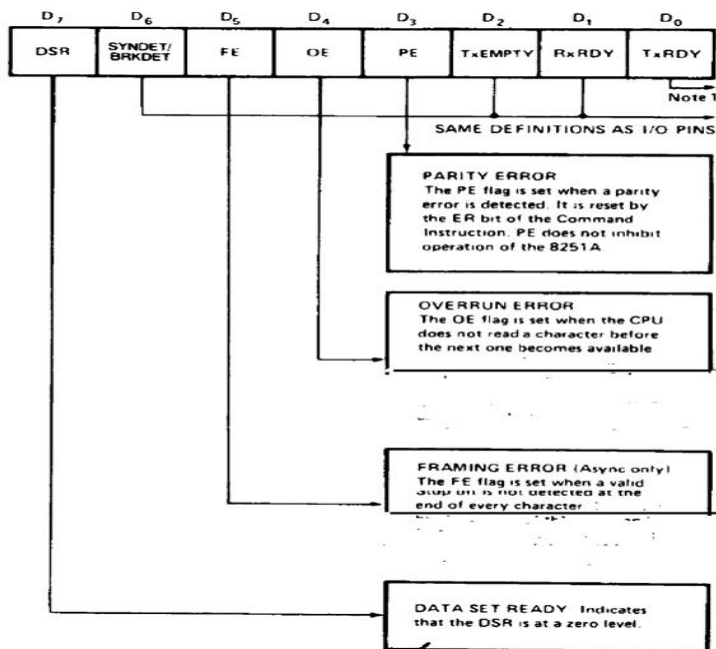

Fig. Status word

TTL TO RS 232C AND RS232C TO TTL CONVERSION.

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.
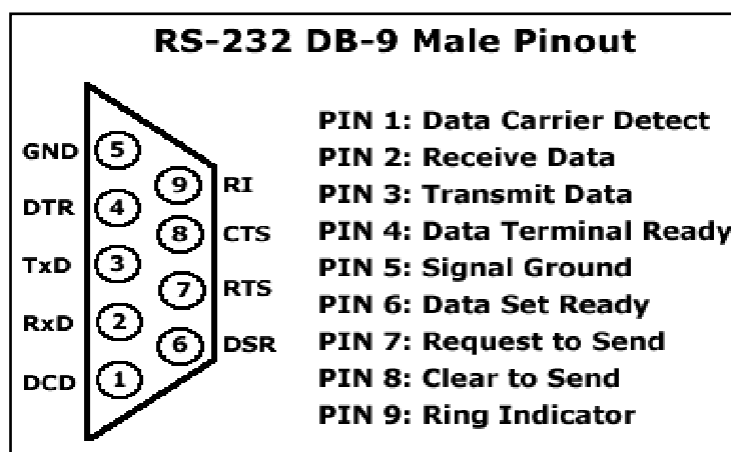
RS-232C is defined as the "Interface between data terminal equipment and data communications equipment using serial binary data exchange." This definition defines data terminal equipment (DTE) as the computer, while data communications equipment (DCE) is the modem. A modem cable has pin-to-pin connections, 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.

RS-232 DB-9 Male Pinout

PIN 1: Data Carrier Detect
PIN 2: Receive Data
PIN 3: Transmit Data
PIN 4: Data Terminal Ready
PIN 5: Signal Ground
PIN 6: Data Set Ready
PIN 7: Request to Send
PIN 8: Clear to Send
PIN 9: Ring Indicator

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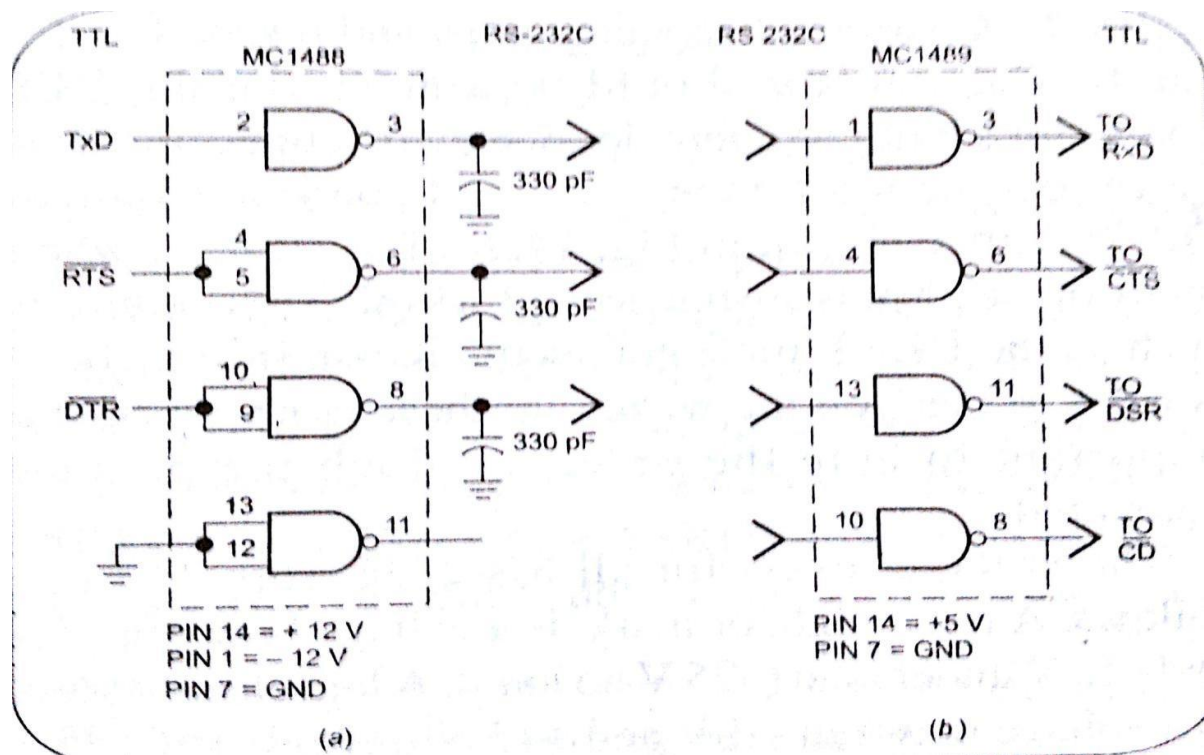
Fig. (a). TTL to Rs-232C and Fig. (b). RS-232C to TTL Conversion

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

- I²C

- 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

147

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.
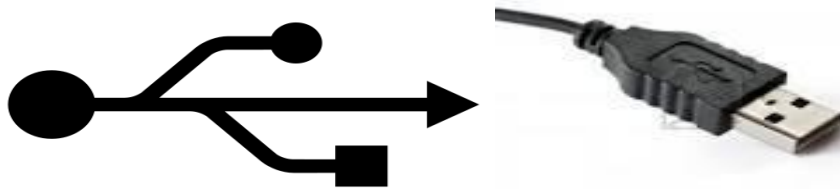
HISTORY

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.

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

148

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
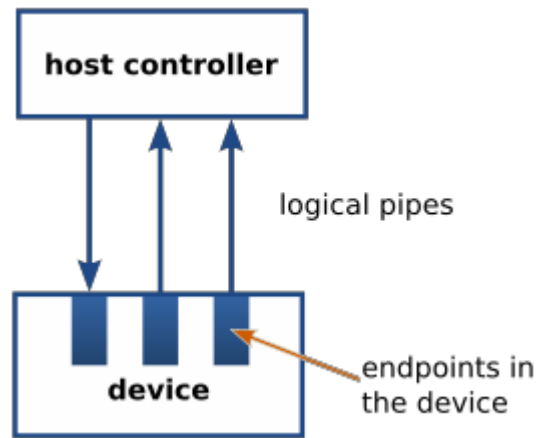
The design architecture of USB is asymmetrical in its topology, consisting of a host, a multitude of downstream USB ports, and multiple peripheral devices connected in a tiered-star topology. Additional USB hubs may be included in the tiers, allowing branching into a tree structure with up to five tier 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.
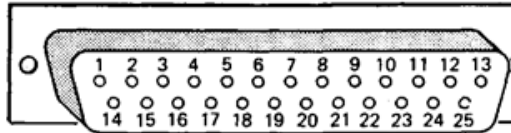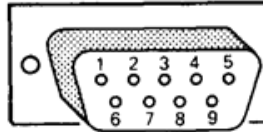
FIGURE   Connectors often used for RS-232C connections. (a) DB-25P 25-pin male. (b) DE-9P 9-pin male DIN connector.

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 ±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.
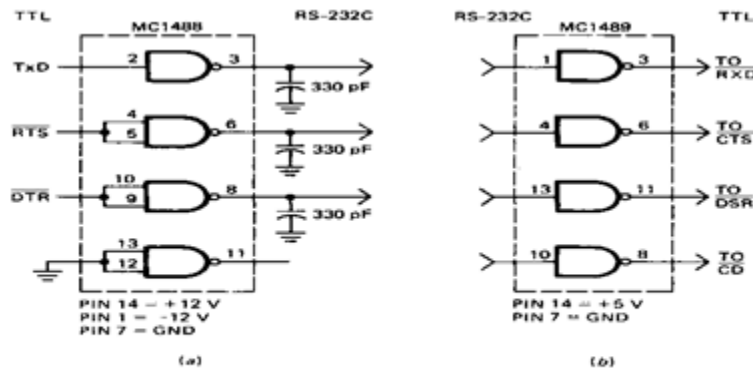

FIGURE   TTL to RS-232C to TTL signal conversion. (a). MC1488 used to convert TTL to RS-232C. (b) MC1489 used to convert RS-232C to TTL.

The MCI488s require + and - supplies, but the MCI489s require only+ 5 V. Note the capacitor

151

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/μs.

RS-232C SIGNAL DEFINITIONS

PIN NUMBERS FOR 9 PINS	PIN NUMBERS FOR 25 PINS	COMMON NAME	RS-232C NAME	DESCRIPTION	SIGNAL DIRECTION ON DCE
	1		AA	PROTECTIVE GROUND	—
3	2	TXD	BA	TRANSMITTED DATA	IN
2	3	RXD	BB	RECEIVED DATA	OUT
7	4	RTS	CA	REQUEST TO SEND	IN
8	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	OUT
	9		—	(RESERVED FOR DATA SET TESTING)	—
	10		—	(RESERVED FOR DATA SET TESTING)	—
	11			UNASSIGNED	—
	12		SCF	SECONDARY RECEIVED LINE SIGNAL DETECTOR	OUT
	13		SCB	SECONDARY CLEAR TO SEND	OUT
	14		SBA	SECONDARY TRANSMITTED DATA	IN
	15		DB	TRANSMISSION SIGNAL ELEMENT TIMING (DCE SOURCE)	OUT
	16		SBB	SECONDARY RECEIVED DATA	OUT
	17		DD	RECEIVER SIGNAL ELEMENT TIMING (DCE SOURCE)	OUT
	18			UNASSIGNED	—
	19		SCA	SECONDARY REQUEST TO SEND	IN
4	20	DTR	CD	DATA TERMINAL READY	IN
	21		CG	SIGNAL QUALITY DETECTOR	OUT
9	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   RS-232C pin names and signal directions.

Figure 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.

NEED FOR DMA

DMA stands for Direct Memory Access. It is designed by Intel to transfer data at the fastest rate.

If large amount of data is involved in I/O transfer these delays are going to be large, entire process will be slowed down if I/O transfer takes place through CPU.This difficulty can be overcome by isolating CPU and allow memory and I/O device perform the data transfer directly. Such an operation is called DMA.Since CPU is not involved in DMA it requires separate controller to perform these operations. This controller is called DMA controller.

DMA DATA TRANSFER METHOD

Initially, when any device has to send data between the device and the memory, the device has to send DMA request (DRQ) to DMA controller.

The DMA controller sends Hold request (HRQ) to the CPU and it waits for the CPU to assert the HLDA.

Then the microprocessor tri-states all the data bus, address bus, and control bus. The CPU leaves the control over bus and acknowledges the HOLD request through HLDA signal to DMA controller.

After receiving HLDA signal DMA controller gives DACK to the peripheral device which has requested DMA
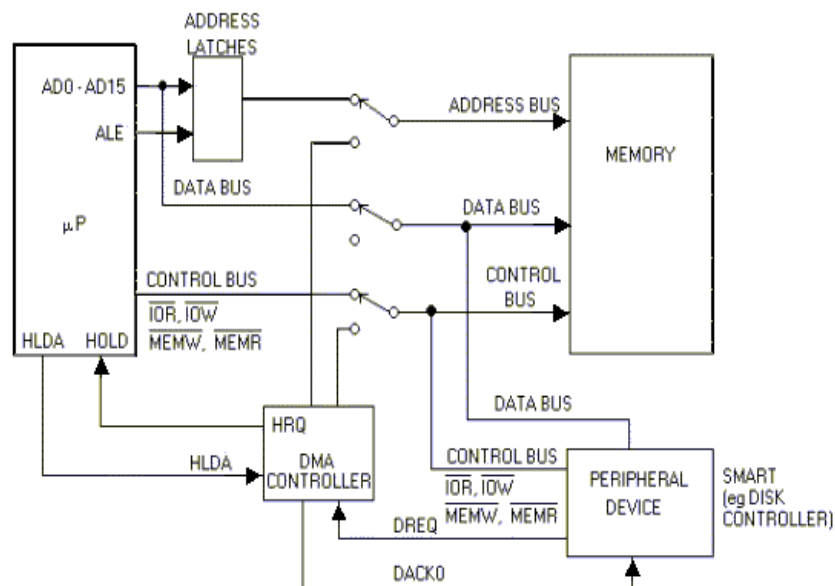
When DMA is not performed Switches are connected such that Microprocessor is connected to Address bus, Data Bus and control bus.

When DMA is performed the switches changes the position where address bus and control bus are connected to DMA controller and data bus gets isolated by CPU and is connected only in between I/O and Memory. Such process is called DMA.

Features of 8257
- It has four channels which can be used over four I/O devices.

153

- Each channel has 16-bit address register and 14-bit counter register.
- Each channel can transfer data up to 64kb.
- Each channel can be programmed independently.
- Each channel can perform read transfer, write transfer and verify transfer Operations.
- It generates MARK signal to the peripheral device that 128 bytes have been transferred.
- Its frequency ranges from 250Hz to 3MHz.
- It operates in 2 modes, i.e., Master mode and Slave mode.
- Slave mode: Where CPU and DMA controller Interact.
- Master mode: Where DMA controller connects memory and I/O device.
- The DMA controller which is widely used in microprocessor is Intel 8257 DMA controller with 40 pin Dual in line package.

Block diagram of Intel 8257
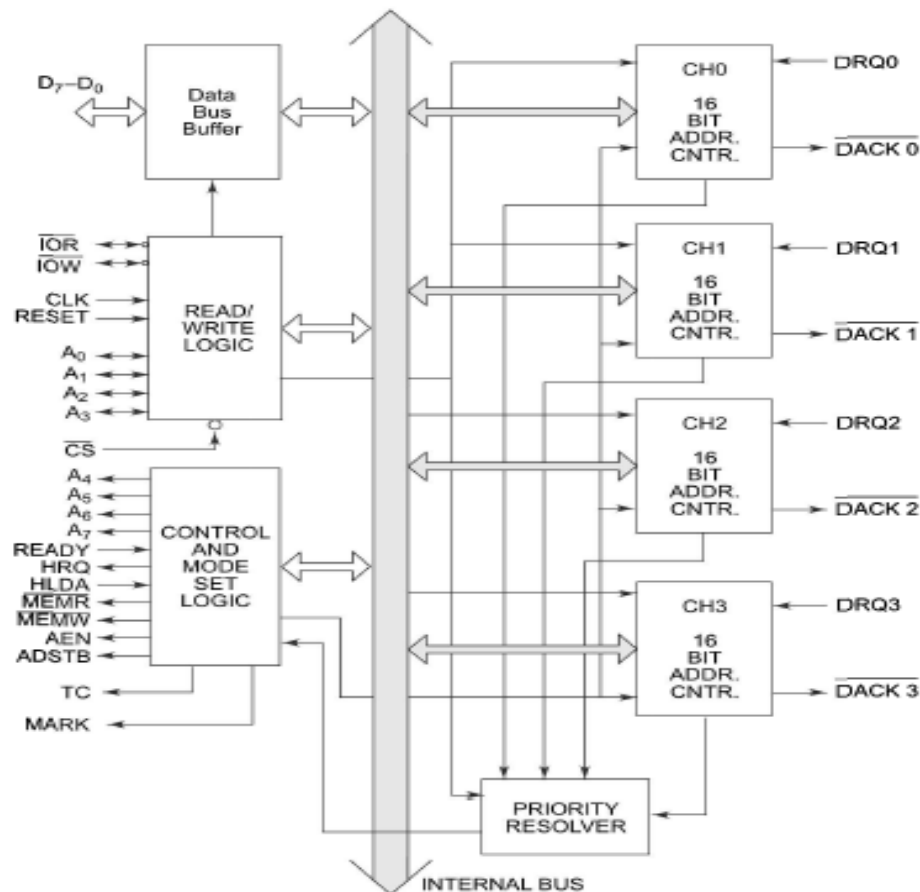
8257 is programmable DMA controller.

It consists of 4 DMA channels, control logic, data bus buffer, Read/write logic, Priority Resolver. The 8257 performs the DMA operation over four independent DMA channels. Each of four channels of 8257 has DMA address register and terminal count register. DMA address register is 16-bit

154

Terminal count register has 14-*bit* count number and 2-bit code indicating DMA transfer. There are two common registers for all the channels; namely, mode set register and status register. Thus there are a total of ten registers.

Register Organization of 8257

Register	Byte	A_3	A_2	A_1	A_0	F/L	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CH-0 DMA Address	LSB	0	0	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	0	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-0 Terminal Count	LSB	0	0	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	0	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-1 DMA Address	LSB	0	0	1	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	0	1	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-1 Terminal Count	LSB	0	0	1	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	0	1	1	1	Rd	Wr	C_{11}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-2 DMA Address	LSB	0	1	0	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	1	0	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-2 Terminal Count	LSB	0	1	0	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	1	0	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
CH-3 DMA Address	LSB	0	1	1	0	0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
	MSB	0	1	1	0	1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8
CH-3 Terminal Count	LSB	0	1	1	1	0	C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0
	MSB	0	1	1	1	1	Rd	Wr	C_{13}	C_{12}	C_{11}	C_{10}	C_9	C_8
MODE SET (Programme only)	—	1	0	0	0	0	AL	TCS	EW	RP	EN3	EN2	EN1	EN0
STATUS (Read only)	—	1	0	0	0	0	0	0	0	UP	TC3	TC2	TC1	TC0

DMA Address Register

Each DMA channel has one DMA address register. The function of this register is to store the address of the starting memory location, which will be accessed by the DMA channel. The device that wants to transfer data over a DMA channel, will access the block of the memory with the starting address stored in the DMA Address Register. Terminal Count Register: Each of the four DMA channels of 8257 has one terminal count register (TC).The low order 14-bits of the terminal count register are initialised with the binary equivalent of the number of required DMA cycles. After each DMA cycle, the terminal count register content will be decremented by one and finally it becomes zero after the required number of DMA cycles are over. In 1 DMA operation maximum number of bytes that can transfer are 2^{14} which is 16KB of information.
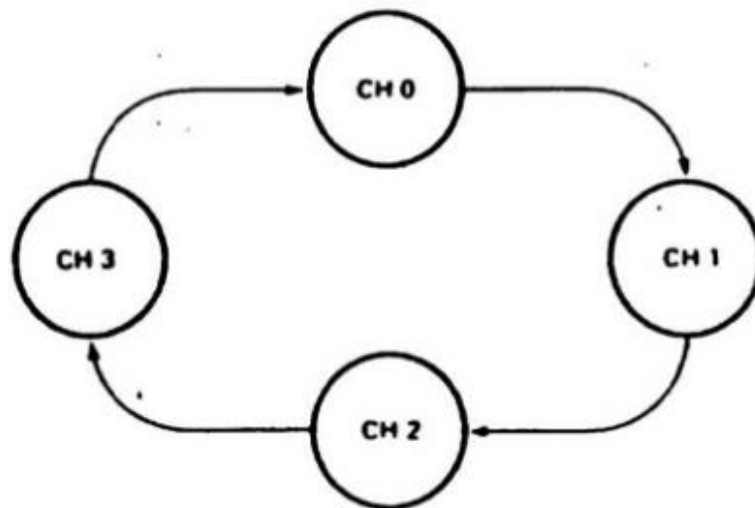
Mode Set register bit Format
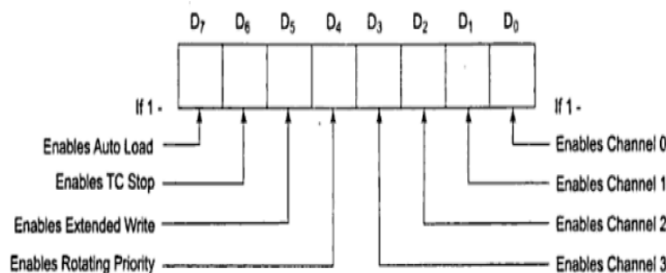
The mode set register is used for programming the 8257 as per the requirements of the system.

The function of the mode set register is to enable the DMA channels individually and also to set the various modes of operation.

The bits **Do-D3** enable one of the four DMA channels of 8257. for example, if Do is '1', channel 0 is enabled.

If **D6** TC STOP bit is set, the selected channel is disabled after the terminal count condition is reached, and it further prevents any DMA cycle on the channel.

If **D7** Autoload bit is set, Count will be once again loaded into count register and DMA transfer will be permitted for same number of bytes once again. This autoload is available only for channel 2.

If **D4** is set, rotating priority is enabled, otherwise, fixed priority is enabled. Rotating priority means the channels has a circular sequence. After each DMA cycle the priority of the channel changes. The channel which has just been serviced will have the lowest priority.

If **D5** Extended Write bit is set, Memory write and I/O write will happen one clock ahead of operation. So that both memory device and I/O device would know that Write operation is being performed.

Status register bit Format:
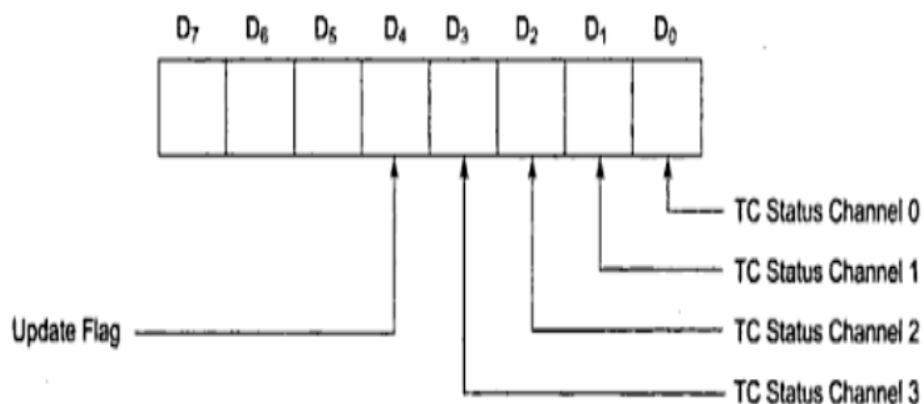
If 1, the respective channel has reached the terminal count condition.

156

The lower order 4-bits of this register contain the terminal count status for the four individual channels. If any of these bits is set, it indicates that the specific channel has reached the terminal count condition. The update flag is set every time, the channel 2 registers are loaded with contents of the channel 3 registers. This register can only read.

Data Bus Buffer, Read/Write Logic, Control Unit, Priority Resolver

In the slave mode, the read/write logic accepts the I/O Read or I/O Write signals, decodes the Ao-A3 lines and either writes the contents of the data bus to the addressed internal register or reads the contents of the selected register depending upon whether IOW or IOR signal is activated. In master mode, the read/write logic generates the IOR and IOW signals to control the data flow to or from the selected peripheral.

The control logic controls the sequences of operations and generates the required control signals like AEN, ADSTB, MEMR, MEMW, TC and MARK along with the address lines A4-A7, in master mode. The priority resolver resolves the priority of the four DMA channels depending upon whether normal priority or rotating priority is programmed.

Pin Diagram of 8257 DMA Controller
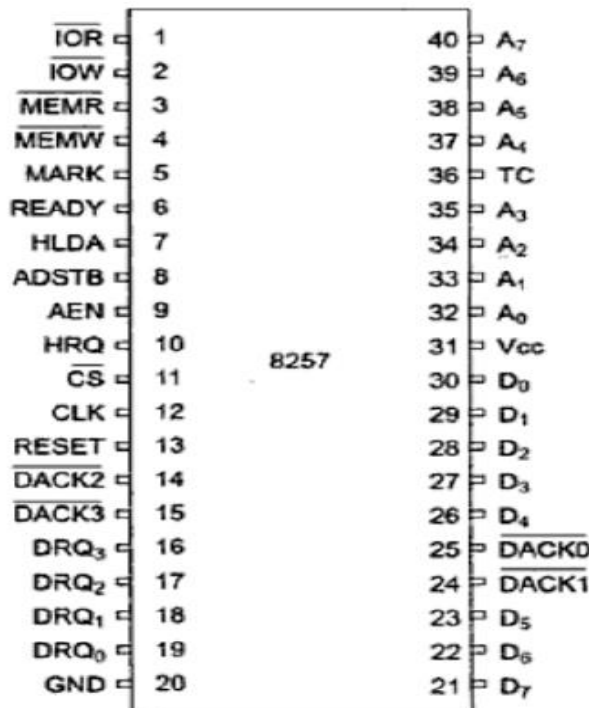
Signal Description of 8257

DRQo-DRQ3 : DMA request

These are the four individual channel DMA request inputs, used by the peripheral devices for requesting the DMA services. The DRQo has the highest priority while DRQ3 has the lowest one,

if the fixed priority mode is selected.

DACKo'-DACK3':DMA Acknowledgement

These are the active-low DMA acknowledge lines, which updates the requesting peripheral about the status of their request by the CPU. These lines can also act as strobe lines for the requesting devices.

Do-D7:

These are bidirectional, data lines used to interface the system bus with the internal data bus of 8257.The address is transferred over Do-D7 during the first clock cycle of the DMA cycle. During the rest of the period, data is available on the data bus.

IOR':I/O Read

This is an active-low bidirectional input line . In slave mode, this acts as input signal.It is used by the CPU to read internal registers of 8257. In master mode it acts as output.It is used to read data from a peripheral during a memory write cycle.

IOW' :I/O Write

This is an active low bidirectionl line that acts as input in slave mode to load the contents of the data bus to the 8-bit mode register or upper/lower byte of a 16-bit DMA address register or terminal count register. In the master mode, it is a control output that loads the data to a peripheral during DMA memory read cycle (write to peripheral).

CLK:

It is a clock frequency signal which is required for the internal operation of 8257.

RESET :

This signal is used to RESET the DMA controller by disabling all the DMA channels.

Ao-A3: (Address bus)

These are the four least significant address lines. In the slave mode, they act as an input, which selects one of the registers to be read or written. In the master mode, they are the four least significant memory address output lines generated by 8257.

CS: Chip Select

It is an active-low chip select line. In the Slave mode, it enables the read/write operations to/from 8257. In the master mode, it disables the read/write operations to/from 8257.

A4-A7 :

This is the higher nibble of the lower byte address generated by 8257 during the master mode of DMA operation.

READY:

It is an active-high asynchronous input signal, which makes DMA ready by inserting wait states.

HRQ: Hold Request

This signal is used to receive the hold request signal from the output device. In the slave mode, it is connected with a DRQ input line 8257. In Master mode, it is connected with HOLD input of the CPU.

HLDA: Hold Acknowledgement

It is the hold acknowledgement signal which indicates the DMA controller that the bus has been granted to the requesting peripheral by the CPU when it is set to 1.

MEMR': Memory Read

This active –low memory read output is used to read data from the addressed memory locations during DMA read cycles.

MEMW': Memory Write

This active-low three state output is used to write data to the addressed memory location during DMA write operation.

V_{cc}

It is the power signal which is required for the operation of the circuit.

ADSTB: Address Strobe

This signal is used to convert the higher byte of the memory address generated by the DMA controller into the latches.

AEN: Address Enable

This signal is used to disable the address bus/data bus.

TC: Terminal Count

It stands for 'Terminal Count', which indicates the present DMA cycle to the present peripheral devices.

MARK

The mark will be activated after each 128 cycles or integral multiples of it from the beginning. It indicates the current DMA cycle is the 128th cycle since the previous MARK output to the selected peripheral device.

INTERFACING WITH 8237/8257

Interfacing 8257 with 8086
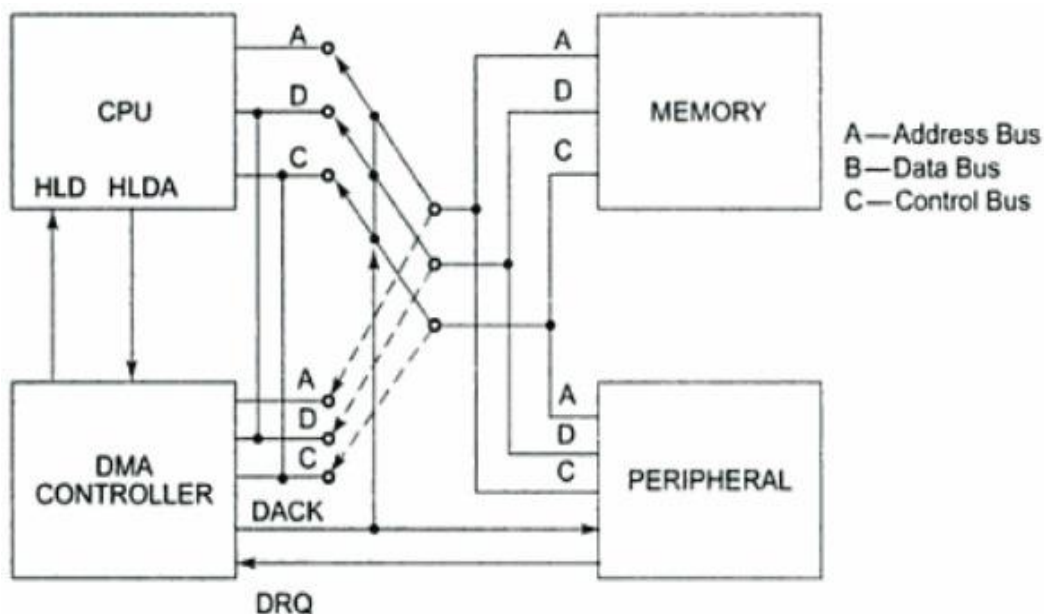
Once a DMA controller is initialised by a CPU property, it is ready to take control of the system bus on a DMA request, either from a peripheral or itself (in case of memory-to-memory transfer). The DMA controller sends a HOLD request to the CPU and waits for the CPU to assert the HLDA signal.

159

The CPU relinquishes the control of the bus before asserting the HLDA signal. Once the HLDA signal goes high, the DMA controller activates the DACK signal to the requesting peripheral and gains the control of the system bus. The DMA controller is the sole master of the bus, till the DMA operation is over. The CPU remains in the HOLD status (all of its signals are tristate except HOLD and HLDA), till the DMA controller is the master of the bus.
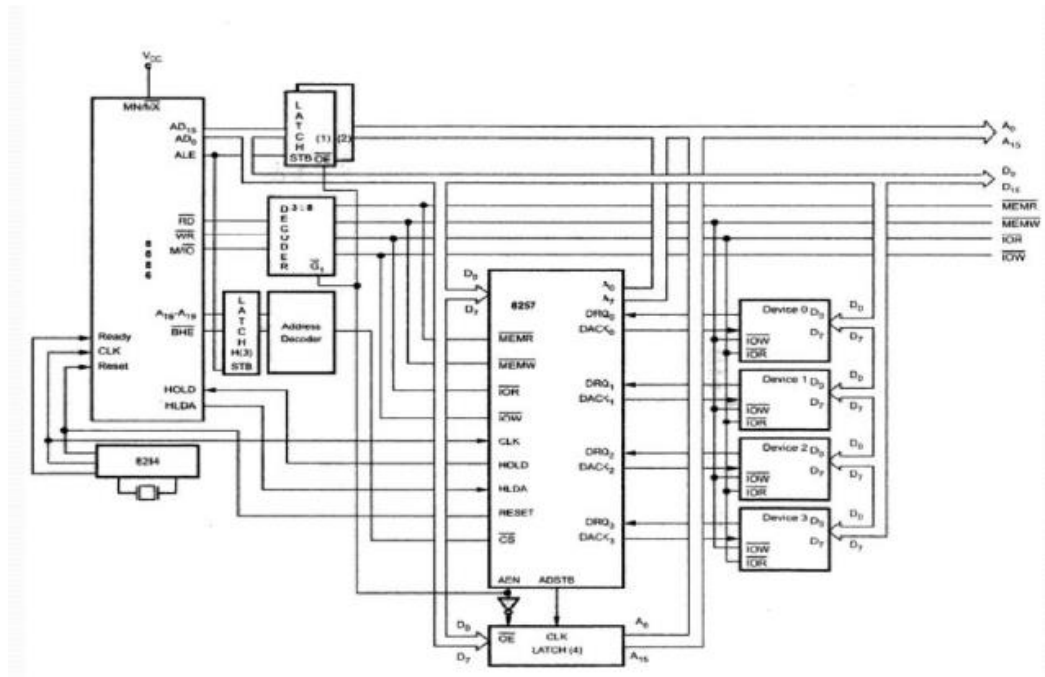
MODULE –IV
8051 MICROCONTROLLER

THE 8051 ARCHITECTURE

Different aspects of a microprocessor/controller:

- Hardware :Interface to the real world

- Software :order how to deal with inputs

The necessary tools for a microprocessor/controller:

- CPU: Central Processing Unit

- I/O: Input /Output

- Bus: Address bus & Data bus

- Memory: RAM & ROM

- Timer

- Interrupt

- Serial Port

- Parallel Port

Microprocessors:

General-purpose microprocessor:

- CPU for Computers

- No RAM, ROM, I/O on CPU chip itself

- Example : Intel's x86, Motorola's 680x0
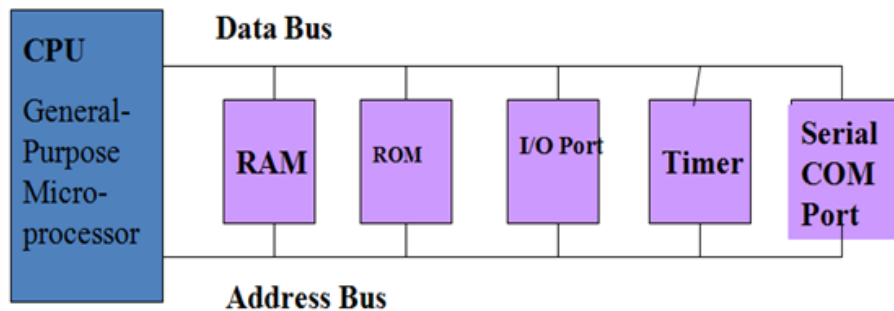
General purpose of microprocessor or Block diagram

Microcontroller:

- A smaller computer
- On-chip RAM, ROM, I/O ports...
- Example : Motorola"s 6811, Intel's 8051, Zilog's Z8 and PIC 16X

Difference between Microprocessor and Microcontroller

Microcontroller	Microprocessor
Microcontrollers are used to execute a single task within an application.	Microprocessors are used for big applications.
Its designing and hardware cost is low.	Its designing and hardware cost is high.
Easy to replace.	Not so easy to replace.
It is built with CMOS technology, which requires less power to operate.	Its power consumption is high because it has to control the entire system.
It consists of CPU, RAM, ROM, I/O ports.	It doesn't consist of RAM, ROM, I/O ports. It uses its pins to interface to peripheral devices.

Types of Microcontrollers

Microcontrollers are divided into various categories based on memory, architecture, bits and instruction sets. Following is the list of their types −Bit

Based on bit configuration, the microcontroller is further divided into three categories.

- **8-bit microcontroller** − This type of microcontroller is used to execute arithmetic and logical operations like addition, subtraction, multiplication division, etc. For example, Intel 8031 and 8051 are 8 bits microcontroller.

- **16-bit microcontroller** − This type of microcontroller is used to perform arithmetic and logical operations where higher accuracy and performance is required. For example, Intel 8096 is a 16-bit microcontroller.

- **32-bit microcontroller** − This type of microcontroller is generally used in automatically controlled appliances like automatic operational machines, medical appliances, etc.

Memory

Based on the memory configuration, the microcontroller is further divided into two categories.

- **External memory microcontroller** − This type of microcontroller is designed in such a way that they do not have a program memory on the chip. Hence, it is named as external memory microcontroller. For example: Intel 8031 microcontroller.

- **Embedded memory microcontroller** − This type of microcontroller is designed in such a way that the microcontroller has all programs and data memory, counters and timers, interrupts, I/O ports are embedded on the chip. For example: Intel 8051 microcontroller.

Instruction Set

Based on the instruction set configuration, the microcontroller is further divided into two categories.

- **CISC** − CISC stands for complex instruction set computer. It allows the user to insert a single instruction as an alternative to many simple instructions.

- **RISC** − RISC stands for Reduced Instruction Set Computers. It reduces the operational time by shortening the clock cycle per instruction.

Applications of Microcontrollers

Microcontrollers are widely used in various different devices such as −

- Light sensing and controlling devices like LED.

163

- Temperature sensing and controlling devices like microwave oven, chimneys.

- Fire detection and safety devices like Fire alarm.

- Measuring devices like Volt Meter.

Three criteria in Choosing a Microcontroller:

1. meeting the computing needs of the task efficiently and cost effectively
   - speed, the amount of ROM and RAM, the number of I/O ports and timers, size, packaging, power consumption
   - easy to upgrade
   - cost per unit
2. availability of software development tools
   - assemblers, debuggers, C compilers, emulator, simulator, technical support
3. wide availability and reliable sources of the microcontrollers.

Block Diagram:
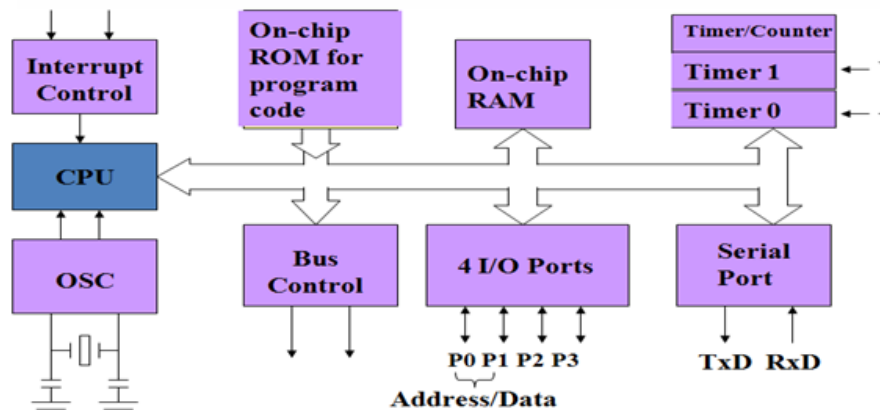
The basic features 8051 Core:

- 8-bit CPU optimized for control applications
- Capability for single bit Boolean operations.
- Supports up to 64K of program memory.
- Supports up to 64K of program memory.
- 4 K bytes of on-chip program memory.

- Newer devices provide more.

- 128 or 256 bytes of on-chip data RAM.

- Four 8 bit ports.

- Two 16-bit timer/counters.

- UART.

- Interrupts.

- On-chip clock oscillator

Inside Microcontroller
(Block Diagram of Original 8051)
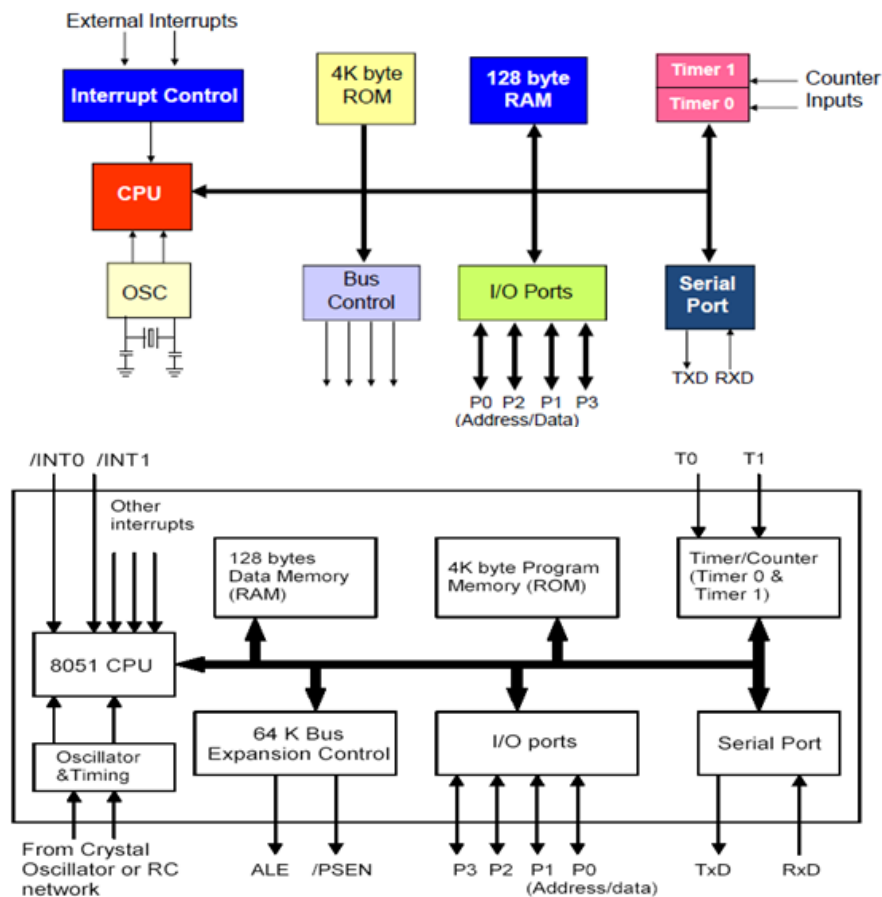
8051 Microcontroller Architecture

Let us now discuss the architecture of 8051 Microcontroller.

In the following diagram, the system bus connects all the support devices to the CPU. The system bus consists of an 8-bit data bus, a 16-bit address bus and bus control signals. All other devices like program memory, ports, data memory, serial interface, interrupt control, timers, and the CPU are all interfaced together through the system bus.
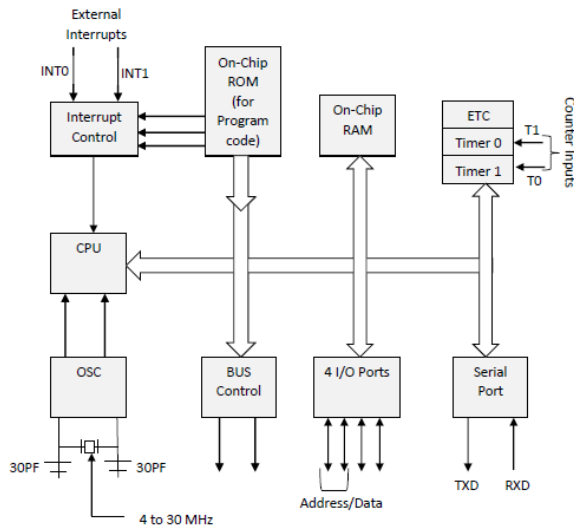
The pin diagram of 8051 microcontroller looks as follows −
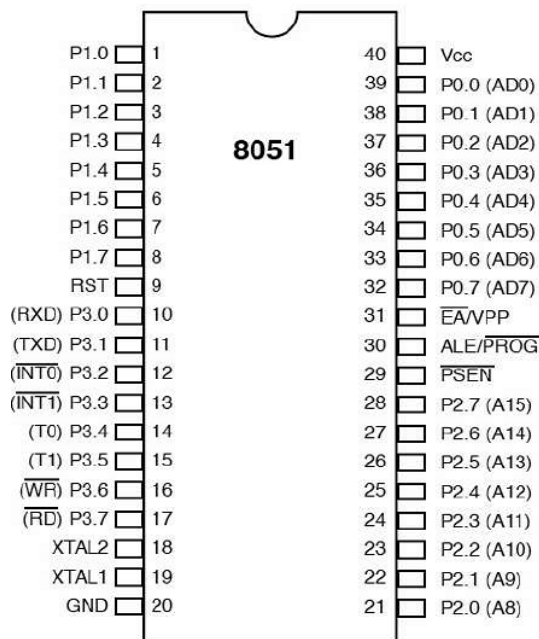
- **Pins 1 to 8** − These pins are known as Port 1. This port doesn't serve any other functions. It is internally pulled up, bi-directional I/O port.

- **Pin 9** − It is a RESET pin, which is used to reset the microcontroller to its initial values.

- **Pins 10 to 17** − These pins are known as Port 3. This port serves some functions like interrupts, timer input, control signals, serial communication signals RxD and TxD, etc.

- **Pins 18 & 19** − These pins are used for interfacing an external crystal to get the system clock.

- **Pin 20** − This pin provides the power supply to the circuit.

166

- **Pins 21 to 28** − These pins are known as Port 2. It serves as I/O port. Higher order address bus signals are also multiplexed using this port.

- **Pin 29** − This is PSEN pin which stands for Program Store Enable. It is used to read a signal from the external program memory.

- **Pin 30** − This is EA pin which stands for External Access input. It is used to enable/disable the external memory interfacing.

- **Pin 31** − This is ALE pin which stands for Address Latch Enable. It is used to demultiplex the address-data signal of port.

- **Pins 32 to 39** − These pins are known as Port 0. It serves as I/O port. Lower order address and data bus signals are multiplexed using this port.

- **Pin 40** − This pin is used to provide power supply to the circuit.

  8051 microcontrollers have 4 I/O ports each of 8-bit, which can be configured as input or output. Hence, total 32 input/output pins allow the microcontroller to be connected with the peripheral devices.

- **Pin configuration**, i.e. the pin can be configured as 1 for input and 0 for output as per the logic state.

o **Input/output (I/O) pin** − All the circuits within the microcontroller must be connected to one of its pins except P0 port because it does not have pull-up resistors built-in.

o **Input pin** − Logic 1 is applied to a bit of the P register. The output FE transistor is turned off and the other pin remains connected to the power supply voltage over a pull-up resistor of high resistance.

- **Port 0** − The P0 (zero) port is characterized by two functions −

o When the external memory is used then the lower address byte (addresses A0A7) is applied on it, else all bits of this port are configured as input/output.

o When P0 port is configured as an output then other ports consisting of pins with built-in pull-up resistor connected by its end to 5V power supply, the pins of this port have this resistor left out.

Input Configuration

If any pin of this port is configured as an input, then it acts as if it "floats", i.e. the input has unlimited input resistance and in-determined potential.

Output Configuration

When the pin is configured as an output, then it acts as an "open drain". By applying logic 0 to a port bit, the appropriate pin will be connected to ground (0V), and applying logic 1, the external output will keep on "floating".

In order to apply logic 1 (5V) on this output pin, it is necessary to build an external pullup resistor.

Port 1

P1 is a true I/O port as it doesn't have any alternative functions as in P0, but this port can be configured as general I/O only. It has a built-in pull-up resistor and is completely compatible with TTL circuits.

Port 2

P2 is similar to P0 when the external memory is used. Pins of this port occupy addresses intended for the external memory chip. This port can be used for higher address byte with addresses A8-A15. When no memory is added then this port can be used as a general input/output port similar to Port 1.

Port 3

In this port, functions are similar to other ports except that the logic 1 must be applied to appropriate bit of the P3 register.

Pins Current Limitations

- When pins are configured as an output (i.e. logic 0), then the single port pins can receive a current of 10mA.

- When these pins are configured as inputs (i.e. logic 1), then built-in pull-up resistors provide very weak current, but can activate up to 4 TTL inputs of LS series.

- If all 8 bits of a port are active, then the total current must be limited to 15mA (port P0: 26mA).

- If all ports (32 bits) are active, then the total maximum current must be limited to 71mA.

Interrupts are the events that temporarily suspend the main program, pass the control to the external sources and execute their task. It then passes the control to the main program where it had left off.

8051 has 5 interrupt signals, i.e. INT0, TFO, INT1, TF1, RI/TI. Each interrupt can be enabled or disabled by setting bits of the IE register and the whole interrupt system can be disabled by clearing the EA bit of the same register.

IE (Interrupt Enable) Register

This register is responsible for enabling and disabling the interrupt. EA register is set to one for enabling interrupts and set to 0 for disabling the interrupts. Its bit sequence and their meanings are shown in the following figure.

EA	-	-	ES	ET1	EX1	ET0	EX0

EA	IE.7	It disables all interrupts. When EA = 0 no interrupt will be acknowledged and EA = 1 enables the interrupt individually.
-	IE.6	Reserved for future use.
-	IE.5	Reserved for future use.
ES	IE.4	Enables/disables serial port interrupt.
ET1	IE.3	Enables/disables timer1 overflow interrupt.
EX1	IE.2	Enables/disables external interrupt1.
ET0	IE.1	Enables/disables timer0 overflow interrupt.
EX0	IE.0	Enables/disables external interrupt0.

IP (Interrupt Priority) Register

We can change the priority levels of the interrupts by changing the corresponding bit in the Interrupt Priority (IP) register as shown in the following figure.

- A low priority interrupt can only be interrupted by the high priority interrupt, but not interrupted by another low priority interrupt.

- If two interrupts of different priority levels are received simultaneously, the request of higher priority level is served.

- If the requests of the same priority levels are received simultaneously, then the internal polling sequence determines which request is to be serviced.

-		-	PT2	PS	PT1	PX1	PT0	PX0
bit7			bit6	bit5	bit4	bit3	bit2	bit1

-	IP.6	Reserved for future use.
-	IP.5	Reserved for future use.
PS	IP.4	It defines the serial port interrupt priority level.
PT1	IP.3	It defines the timer interrupt of 1 priority.
PX1	IP.2	It defines the external interrupt priority level.
PT0	IP.1	It defines the timer0 interrupt priority level.
PX0	IP.0	It defines the external interrupt of 0 priority level.

TCON Register

TCON register specifies the type of external interrupt to the microcontroller.

REGISTER SET OF 8051

A register is a small place in a CPU that can store small amounts of the data used for performing various operations such as addition and multiplication and loads the resulting data on the main memory. Registers contain the address of the memory location where the data is to be stored. The size of the register is very

important for modern controllers. For instance, for a 64-bit register, a CPU tries to add two 32-bit numbers and gives a 64-bit result.

Types of Registers

The 8051 microcontroller contains mainly two types of registers:
- General-purpose registers (Byte addressable registers)
- Special function registers (Bit addressable registers)

8051 RAM Memory
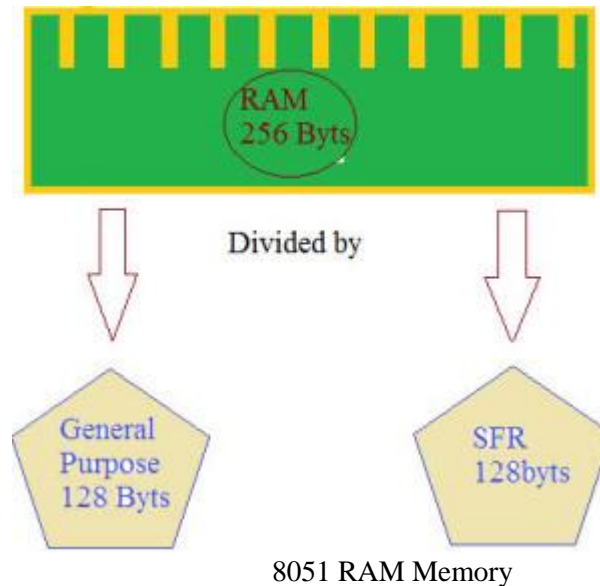
The 8051 microcontroller consists of 256 bytes of RAM, which is divided into two ways, such as 128 bytes for general purpose and 128 bytes for special function registers (SFR) memory. The memory which is used for general purpose is called as RAM, and the memory used for SFR contains all the peripheral related registers like Accumulator, 'B' register, Timers or Counters, and interrupt related registers.

General Purpose Registers

As we discussed earlier in this article that there are four different bank registers with each bank having 8 addressable 8-bit registers, and only one bank register can be accessed at a time. But, by changing the bank register's number in the flag register, we can access other bank registers, which have been discussed earlier on this paper along with interrupt concept in 8051.

Special Function Registers

The special function registers including the Accumulator, Register B, Data pointer, PCON, PSW, etc., are designed for a predetermined purpose during manufacturing with the address 80H to FFH,

and this area cannot be used for the data or program storage purpose. These registers can be implemented by bit address and byte address registers.

The most widely used registers of the 8051 are A (accumulator), B, R0-R7, DPTR (data pointer), and PC (program counter). All these registers are of 8-bits, except DPTR and PC.

Storage Registers in 8051

We will discuss the following types of storage registers here −

- Accumulator
- R register
- B register
- Data Pointer (DPTR)
- Program Counter (PC)
- Stack Pointer (SP)

Accumulator

The accumulator, register A, is used for all arithmetic and logic operations. If the accumulator is not present, then every result of each calculation (addition, multiplication, shift, etc.) is to be stored into the main memory. Access to main memory is slower than access to a register like the accumulator because the technology used for the large main memory is slower (but cheaper) than that used for a register.

The "R" Registers

The "R" registers are a set of eight registers, namely, R0, R1 to R7. These registers function as auxiliary or temporary storage registers in many operations. Consider an example of the sum of 10 and 20. Store a variable 10 in an accumulator and another variable 20 in, say, register R4. To process the addition operation, execute the following command −

ADD A,R4

After executing this instruction, the accumulator will contain the value 30. Thus "R" registers are very important auxiliary or **helper registers**. The Accumulator alone would not be very useful if it were not for these "R" registers. The "R" registers are meant for temporarily storage of values.

Let us take another example. We will add the values in R1 and R2 together and then subtract the values of R3 and R4 from the result.

MOV A,R3   ;Move the value of R3 into the accumulator

```
ADD A,R4   ;Add the value of R4
MOV R5,A   ;Store the resulting value temporarily in R5
MOV A,R1   ;Move the value of R1 into the accumulator
ADD A,R2   ;Add the value of R2
SUBB A,R5  ;Subtract the value of R5 (which now contains R3 + R4)
```

As you can see, we used R5 to temporarily hold the sum of R3 and R4. Of course, this is not the most efficient way to calculate (R1 + R2) – (R3 + R4), but it does illustrate the use of the "R" registers as a way to store values temporarily.

16-bit Registers of 8051

8-bit Registers of 8051
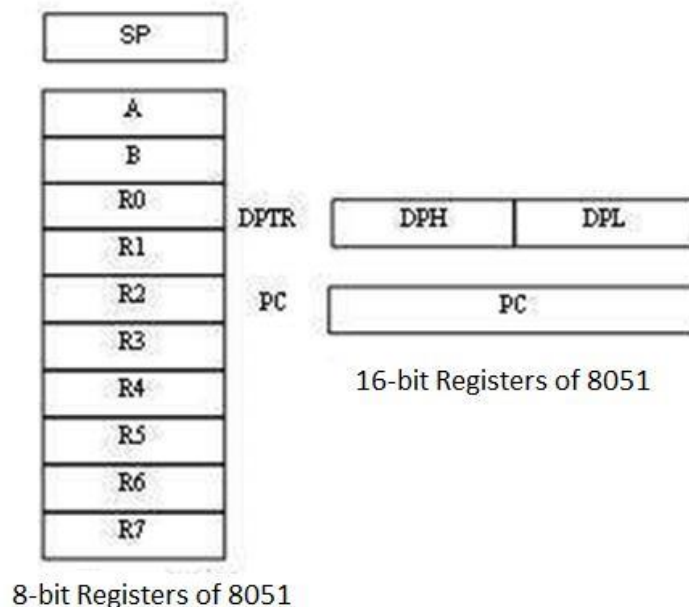
The "B" Register

The "B" register is very similar to the Accumulator in the sense that it may hold an 8-bit (1-byte) value. The "B" register is used only by two 8051 instructions: **MUL AB** and **DIV AB**. To quickly and easily multiply or divide A by another number, you may store the other number in "B" and make use of these two instructions. Apart from using MUL and DIV instructions, the "B" register is often used as yet another temporary storage register, much like a ninth R register.

The Data Pointer

The Data Pointer (DPTR) is the 8051's only user-accessible 16-bit (2-byte) register. The Accumulator, R0–R7 registers and B register are 1-byte value registers. DPTR is meant for pointing to data. It is used by the 8051 to access external memory using the address indicated by DPTR. DPTR is the only 16-bit register available and is often used to store 2-byte values.

The Program Counter

The Program Counter (PC) is a 2-byte address which tells the 8051 where the next instruction to execute can be found in the memory. PC starts at 0000h when the 8051 initializes and is incremented every time after an instruction is executed. PC is not always incremented by 1. Some instructions may require 2 or 3 bytes; in such cases, the PC will be incremented by 2 or 3.

Branch, jump, and **interrupt** operations load the Program Counter with an address other than the next sequential location. Activating a power-on reset will cause all values in the register to be lost. It means the value of the PC is 0 upon reset, forcing the CPU to fetch the first opcode from the ROM location 0000. It means we must place the first byte of up code in ROM location 0000 because that is where the CPU expects to find the first instruction.

The Stack Pointer (SP)

The Stack Pointer, like all registers except DPTR and PC, may hold an 8-bit (1-byte) value. The Stack Pointer tells the location from where the next value is to be removed from the stack. When a value is pushed onto the stack, the value of SP is incremented and then the value is stored at the resulting memory location. When a value is popped off the stack, the value is returned from the memory location indicated by SP, and then the value of SP is decremented.

This order of operation is important. SP will be initialized to 07h when the 8051 is initialized. If a value is pushed onto the stack at the same time, the value will be stored in the internal RAM address 08h because the 8051 will first increment the value of SP (from 07h to 08h) and then will store the pushed value at that memory address (08h). SP is modified directly by the 8051 by six instructions: PUSH, POP, ACALL, LCALL, RET, and RETI.

PSW Register

The program status word (PSW) register is an 8-bit register, also known as **flag register**. It is of 8-bit wide but only 6-bit of it is used. The two unused bits are **user-defined flags**. Four of the flags are called **conditional flags**, which mean that they indicate a condition which results after an instruction is executed. These four are **CY** (Carry), **AC** (auxiliary carry), **P** (parity), and **OV** (overflow). The bits RS0 and RS1 are used to change the bank registers. The following figure shows the program status word register.

The PSW Register contains that status bits that reflect the current status of the CPU.

174

CY	CA	F0	RS1	RS0	OV	-	P

CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry Flag
F0	PSW.5	Flag 0 available to user for general purpose.
RS1	PSW.4	Register Bank selector bit 1
RS0	PSW.3	Register Bank selector bit 0
OV	PSW.2	Overflow Flag
-	PSW.1	User definable FLAG
P	PSW.0	Parity FLAG. Set/ cleared by hardware during instruction cycle to indicate even/odd number of 1 bit in accumulator.

We can select the corresponding Register Bank bit using RS0 and RS1 bits.

RS1	RS2	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

- **CY, the carry flag** − This carry flag is set (1) whenever there is a carry out from the D7 bit. It is affected after an 8-bit addition or subtraction operation. It can also be reset to 1 or 0 directly by an instruction such as "SETB C" and "CLR C" where "SETB" stands for set bit carry and "CLR" stands for clear carry.

- **AC, auxiliary carry flag** − If there is a carry from D3 and D4 during an ADD or SUB operation, the AC bit is set; otherwise, it is cleared. It is used for the instruction to perform binary coded decimal arithmetic.

- **P, the parity flag** − The parity flag represents the number of 1's in the accumulator register only. If the A register contains odd number of 1's, then P = 1; and for even number of 1's, P = 0.

- **OV, the overflow flag** − This flag is set whenever the result of a signed number operation is too large causing the high-order bit to overflow into the sign bit. It is used only to detect errors in signed arithmetic operations.

What is an Addressing Mode?

An Addressing Mode is a way to locate a target Data, which is also called as Operand. The 8051 Family of Microcontrollers allows five types of Addressing Modes for addressing the Operands. They are:

- Immediate Addressing
- Register Addressing
- Direct Addressing
- Register – Indirect Addressing
- Indexed Addressing

Immediate Addressing

In Immediate Addressing mode, the operand, which follows the Opcode, is a constant data of either 8 or 16 bits. The name Immediate Addressing came from the fact that the constant data to be stored in the memory immediately follows the Opcode.

The constant value to be stored is specified in the instruction itself rather than taking from a register. The destination register to which the constant data must be copied should be the same size as the operand mentioned in the instruction.

Example: MOV A, #030H

Here, the Accumulator is loaded with 30 (hexadecimal). The # in the operand indicates that it is a data and not the address of a Register.

Immediate Addressing is very fast as the data to be loaded is given in the instruction itself.

Register Addressing

In the 8051 Microcontroller Memory Organization Tutorial, we have seen the organization of RAM and four banks of Working Registers with eight Registers in each bank.

In Register Addressing mode, one of the eight registers (R0 – R7) is specified as Operand in the Instruction.

It is important to select the appropriate Bank with the help of PSW Register. Let us see a example of Register Addressing assuming that Bank0 is selected.

Example: MOV A, R5

Here, the 8-bit content of the Register R5 of Bank0 is moved to the Accumulator.

Direct Addressing

In Direct Addressing Mode, the address of the data is specified as the Operand in the instruction. Using Direct Addressing Mode, we can access any register or on-chip variable. This includes general purpose RAM, SFRs, I/O Ports, Control registers.

Example: MOV A, 47H

Here, the data in the RAM location 47H is moved to the Accumulator.

Register Indirect Addressing

In the Indirect Addressing Mode or Register Indirect Addressing Mode, the address of the Operand is specified as the content of a Register. This will be clearer with an example.

Example: MOV A, @R1

The @ symbol indicates that the addressing mode is indirect. If the contents of R1 is 56H, for example, then the operand is in the internal RAM location 56H. If the contents of the RAM location 56H is 24H, then 24H is moved into accumulator.

Only R0 and R1 are allowed in Indirect Addressing Mode. These register in the indirect addressing mode are called as Pointer registers.

Indexed Addressing Mode

With Indexed Addressing Mode, the effective address of the Operand is the sum of a base register and an offset register. The Base Register can be either Data Pointer (DPTR) or Program Counter (PC) while the Offset register is the Accumulator (A).

In Indexed Addressing Mode, only MOVC and JMP instructions can be used. Indexed Addressing Mode is useful when retrieving data from look-up tables.

Example: MOVC A, @A+DPTR

Instruction set of 8051

The following table shows the 8051 Instruction Groups and Instructions in each group. There are 49 Instruction Mnemonics in the 8051 Microcontroller Instruction Set and these 49 Mnemonics are divided into five groups

8051 has about 111 instructions. These can be grouped into the following categories

1. Arithmetic Instructions
2. Logical Instructions
3. Data Transfer instructions
4. Boolean Variable Instructions
5. Program Branching Instructions

The following nomenclatures for register, data, address and variables are used while write instructions. A: Accumulator
B: "B" register C: Carry bit
Rn: Register R0 - R7 of the currently selected register bank
Direct: 8-bit internal direct address for data. The data could be in lower 128bytes of RAM (00 - 7FH) or it could be in the special function register (80 - FFH).

Ri: 8-bit external or internal RAM address available in register R0 or R1. This is used for indirect addressing mode.

data8: Immediate 8-bit data available in the instruction. #data16: Immediate 16-bit data available in the instruction.

Addr11: 11-bit destination address for short absolute jump. Used by instructions AJMP & ACALL. Jump range is 2 Kbyte (one page).
Addr16: 16-bit destination address for long call or long jump.
Rel: 2's complement 8-bit offset (one - byte) used for short jump (SJMP) and all conditional jumps. bit: Directly addressed bit in internal RAM or SFR

Arithmetic Instructions

Mnemonics	Description	Bytes	Instruction Cycles
ADD A, Rn	A A + Rn	1	1
ADD A, direct	A A + (direct)	2	1
ADD A, @Ri	A A + @Ri	1	1
ADD A, #data	A A + data	2	1
ADDC A, Rn	A A + Rn + C	1	1

Mnemonics	Description	Bytes	Instruction Cycles
ADDC A, direct	A ← A + (direct) + C	2	1
ADDC A, @Ri	A ← A + @Ri + C	1	1
ADDC A, #data	A ← A + data + C	2	1
DA A	Decimal adjust accumulator	1	1
DIV AB	Divide A by B A ← quotient B ← remainder	1	4
DEC A	A ← A -1	1	1
DEC Rn	Rn ← Rn - 1	1	1
DEC direct	(direct) ← (direct) - 1	2	1
DEC @Ri	@Ri ← @Ri - 1	1	1
INC A	A ← A+1	1	1
INC Rn	Rn ← Rn + 1	1	1
INC direct	(direct) ← (direct) + 1	2	1
INC @Ri	@Ri ← @Ri +1	1	1
INC DPTR	DPTR ← DPTR +1	1	2
MUL AB	Multiply A by B A ← low byte (A*B) B ← high byte (A* B)	1	4
SUBB A, Rn	A ← A - Rn - C	1	1
SUBB A, direct	A ← A - (direct) - C	2	1
SUBB A, @Ri	A ← A - @Ri - C	1	1
SUBB A, #data	A ← A - data - C	2	1

Logical Instructions

Mnemonics	Description	Bytes	Instruction Cycles
ANL A, Rn	A ← A AND Rn	1	1
ANL A, direct	A ← A AND (direct)	2	1
ANL A, @Ri	A ← A AND @Ri	1	1
ANL A, #data	A ← A AND data	2	1
ANL direct, A	(direct) ← (direct) AND A	2	1
ANL direct, #data	(direct) ← (direct) AND data	3	2
CLR A	A ← 00H	1	1
CPL A	A ← A	1	1
ORL A, Rn	A ← A OR Rn	1	1
ORL A, direct	A ← A OR (direct)	1	1
ORL A, @Ri	A ← A OR @Ri	2	1
ORL A, #data	A ← A OR data	1	1
ORL direct, A	(direct) ← (direct) OR A	2	1
ORL direct, #data	(direct) ← (direct) OR data	3	2
RL A	Rotate accumulator left	1	1

RLC A	Rotate accumulator left through carry	1	1
RR A	Rotate accumulator right	1	1
RRC A	Rotate accumulator right through carry	1	1
SWAP A	Swap nibbles within Acumulator	1	1
XRL A, Rn	A A EXOR Rn	1	1
XRL A, direct	A A EXOR (direct)	1	1
XRL A, @Ri	A A EXOR @Ri	2	1
XRL A, #data	A A EXOR data	1	1
XRL direct, A	(direct) (direct) EXOR A	2	1
XRL direct, #data	(direct) (direct) EXOR data	3	2

Data Transfer Instructions

Mnemonics	Description	Bytes	Instruction Cycles
MOV A, Rn	A Rn	1	1
MOV A, direct	A (direct)	2	1
MOV A, @Ri	A @Ri	1	1
MOV A, #data	A data	2	1
MOV Rn, A	Rn A	1	1
MOV Rn, direct	Rn (direct)	2	2
MOV Rn, #data	Rn data	2	1
MOV direct, A	(direct) A	2	1
MOV direct, Rn	(direct) Rn	2	2
MOV direct1, direct2	(direct1) (direct2)	3	2
MOV direct, @Ri	(direct) @Ri	2	2
MOV direct, #data	(direct) #data	3	2
MOV @Ri, A	@Ri A	1	1
MOV @Ri, direct	@Ri (direct)	2	2
MOV @Ri, #data	@Ri data	2	1
MOV DPTR, #data16	DPTR data16	3	2
MOVC A, @A+DPTR	A Code byte pointed by A + DPTR	1	2
MOVC A, @A+PC	A Code byte pointed by A + PC	1	2
MOVC A, @Ri	A Code byte pointed by Ri 8-bit address)	1	2
MOVX A, @DPTR	A External data pointed by DPTR	1	2
MOVX @Ri, A	@Ri A (External data - 8bit address)	1	2

MOVX @DPTR, A	@DPTR → A(External data - 16bit address)	1	2
PUSH direct	(SP) → (direct)	2	2
POP direct	(direct) → (SP)	2	2
XCH Rn	Exchange A with Rn	1	1
XCH direct	Exchange A with direct byte	2	1
XCH @Ri	Exchange A with indirect RAM	1	1
XCHD A, @Ri	Exchange least significant nibble of A with that of indirect RAM	1	1

Boolean Variable Instructions

Mnemonics	Description	Bytes	Instruction Cycles
CLR C	C-bit → 0	1	1
CLR bit	bit → 0	2	1
SET C	C → 1	1	1
SET bit	bit → 1	2	1
CPL C	C	1	1
CPL bit	bit	2	1
ANL C, /bit	C → C .	2	1
ANL C, bit	C → C. bit	2	1
ORL C, /bit	C → C +	2	1
ORL C, bit	C → C + bit	2	1
MOV C, bit	C → bit	2	1
MOV bit, C	bit → C	2	2

Program Branching Instructions

Mnemonics	Description	Bytes	Instruction Cycles
ACALL addr11	PC + 2 → (SP) ; addr 11 → PC	2	2
AJMP addr11	Addr11 → PC	2	2
CJNE A, direct, rel	Compare with A, jump (PC + rel) if not equal	3	2
CJNE A, #data, rel	Compare with A, jump (PC + rel) if not equal	3	2
CJNE Rn, #data, rel	Compare with Rn, jump (PC + rel) if not equal	3	2
CJNE @Ri, #data, rel	Compare with @Ri A, jump (PC + rel) if	3	2

	not equal		
DJNZ Rn, rel	Decrement Rn, jump if not zero	2	2
DJNZ direct, rel	Decrement (direct), jump if not zero	3	2
JC rel	Jump (PC + rel) if C bit = 1	2	2
JNC rel	Jump (PC + rel) if C bit = 0	2	2
JB bit, rel	Jump (PC + rel) if bit = 1	3	2
JNB bit, rel	Jump (PC + rel) if bit = 0	3	2
JBC bit, rel	Jump (PC + rel) if bit = 1	3	2
JMP @A+DPTR	A+DPTR PC	1	2
JZ rel	If A=0, jump to PC + rel	2	2
JNZ rel	If A $\neq$ 0 , jump to PC + rel	2	2
LCALL addr16	PC + 3 (SP), addr16 PC	3	2
LJMP addr 16	Addr16 PC	3	2
NOP	No operation	1	1
RET	(SP) PC	1	2
RETI	(SP) PC, Enable Interrupt	1	2
SJMP rel	PC + 2 + rel PC	2	2
JMP @A+DPTR	A+DPTR PC	1	2
JZ rel	If A = 0. jump PC+ rel	2	2
JNZ rel	If A $\neq$ 0, jump PC + rel	2	2
NOP	No operation	1	1

Assembly Language Programming Using Data Transfer

An assembly language is a low-level programming language used to write program code in terms of mnemonics. Even though there are many high-level languages that are currently in demand, assembly programming language is popularly used in many applications. It can be used for direct hardware manipulations. It is also used to write the 8051 programming code efficiently with less number of clock cycles by consuming less memory compared to the other high-level languages.
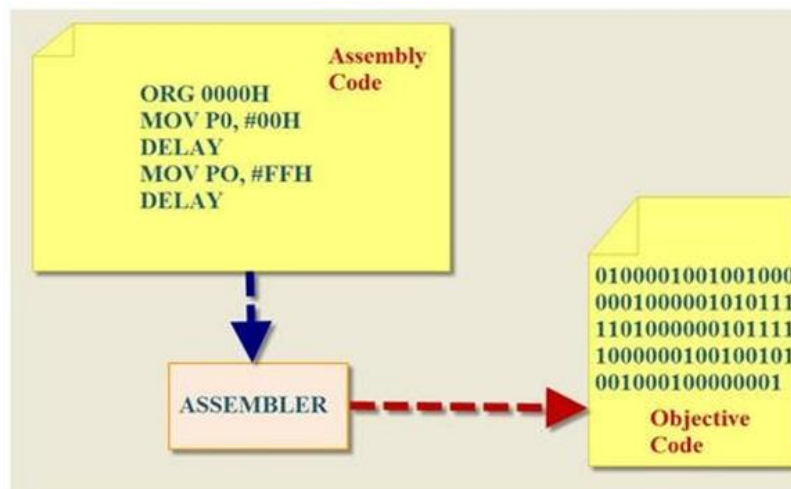
MODULE –V

SYSTEM DESIGN USING MICROCONTROLLER

MODES OF TIMER OPERATION:

8051 Timers/Counters

Timers/Counters of the 8051 micro controller. The 8051 has two counters/timers which can be used either as timer to generate a time delay or as counter to count events happening outside the microcontroller.

Many of the microcontroller applications require counting of external events such as frequency of the pulse trains and generation of precise internal time delays between computer actions. Both these tasks can be implemented by software techniques, but software loops for counting, and timing will not give the exact result rather more important functions are not done. To avoid these problems, timers and counters in the micro-controllers are better options for simple and low-cost applications. These timers and counters are used as interrupts in 8051 microcontroller.

There are two 16-bit timers and counters in 8051 microcontroller: timer 0 and timer 1. Both timers consist of 16-bit register in which the lower byte is stored in TL and the higher byte is stored in TH. Timer can be used as a counter as well as for timing operation that depends on the source of clock pulses to counters

A **timer** is a specialized type of clock which is used to measure time intervals. A timer that counts from zero upwards for measuring time elapsed is often called a **stopwatch**. It is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer.

A **counter** is a device that stores (and sometimes displays) the number of times a particular event or process occurred, with respect to a clock signal. It is used to count the events happening outside the microcontroller. In electronics, counters can be implemented quite easily using register-type circuits such as a flip-flop.

Difference between a Timer and a Counter

The points that differentiate a timer from a counter are as follows −

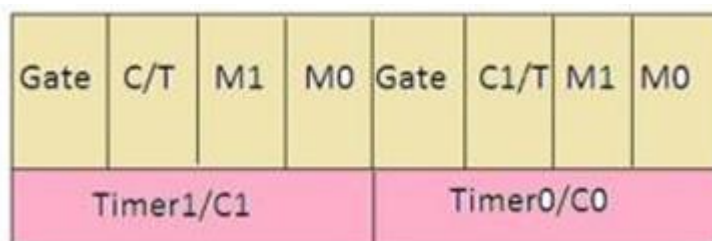| Timer	Counter
The register incremented for every machine cycle.	The register is incremented considering 1 to 0 transitions at its corresponding to an external input pin (T0, T1).

Maximum count rate is 1/12 of the oscillator frequency.	Maximum count rate is 1/24 of the oscillator frequency.
A timer uses the frequency of the internal clock, and generates delay.	A counter uses an external signal to count pulses.
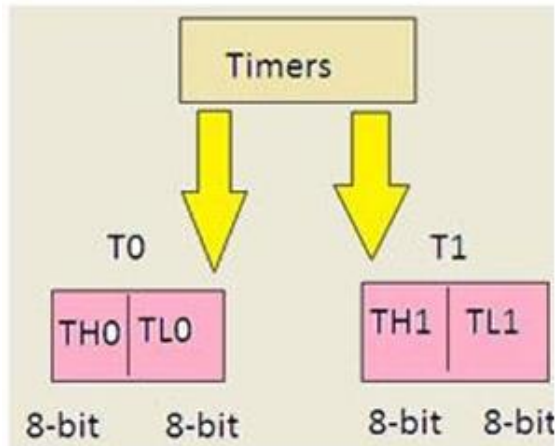
Timers & Counters

Counters and Timers in 8051 microcontroller contain two special function registers: TMOD (Timer Mode Register) and TCON (Timer Control Register), which are used for activating and configuring timers and counters.

Timer Mode Control (TMOD): TMOD is an 8-bit register used for selecting timer or counter and mode of timers. Lower 4-bits are used for control operation of timer 0 or counter0, and remaining 4-bits are used for control operation of timer1 or counter1.This register is present in SFR register, the address for SFR register is 89th.

Gate	C/T	M1	M0	Gate	C1/T	M1	M0
Timer1/C1				Timer0/C0			

Timer mode Control (TMOD)

Gate: If the gate bit is set to „0", then we can start and stop the "software" timer in the same way. If the gate is set to „1", then we can perform hardware timer.

C/T: If the C/T bit is „1", then it is acting as a counter mode, and similarly when set C+ =/T bit is „0"; it is acting as a timer mode.

Mode select bits: The M1 and M0 are mode select bits, which are used to select the timer operations. There are four modes to operate the timers.

Mode 0: This is a 13-bit mode that means the timer operation completes with "8192" pulses.

Mode 1: This is a16-bit mode, which means the timer operation completes with maximum clock pulses that "65535".

Mode 2: This mode is an 8-bit auto reload mode, which means the timer operation completes with only "256" clock pulses.

Mode 3: This mode is a split-timer mode, which means the loading values in T0 and automatically starts the T1.

M0	M1	Mode	Timer Pulses
0	0	M0	13-bit-2^13-8192
0	1	M1	16-bit-2^16-65535 pulses
1	0	M2	8-bit-autoreload mode-2^8= 256 pulses
1	1	M3	Split mode(load the values in T0 automatically start the T1

Mode selection Bits

Timer Control Register (TCON): TCON is another register used to control operations of counter and timers in microcontrollers. It is an 8-bit register wherein four upper bits are responsible for timers and counters and lower bits are responsible for interrupts.

TF1	TR1	TF0	TR0	IE1	IT0	IE0	IT0
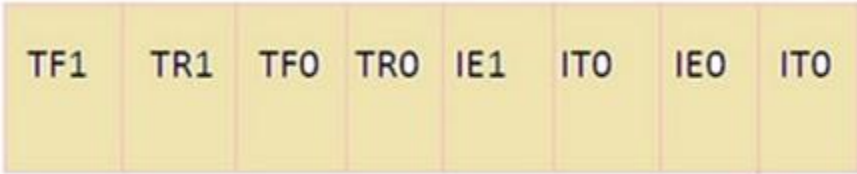
Timer Control Register (TCON)

TF1: The TF1 stands for „timer1" flag bit. Whenever calculating the time-delay in timer1, the TH1 and TL1 reaches to the maximum value that is "FFFF" automatically.

EX: while (TF1==1)

Whenever the TF1=1, then clear the flag bit and stop the timer.

TR1: The TR1 stands for timer1 start or stop bit. This timer starting can be through software instruction or through hardware method.

EX: gate=0(start timer 1 through software instruction) TR1=1; (Start timer)

TF0: The TF0 stands for „timer0" flag-bit. Whenever calculating the time delay in timer1, the TH0 and TL0 reaches to a maximum value that is „FFFF", automatically.

EX: while (TF0==1) Whenever the TF0=1, then clear the flag bit and stop the timer.

TR0: The TR0 stands for „timer0" start or stop bit; this timer starting can be through software instruction or through hardware method.
EX: gate=0 (start   timer 1 through software instruction) TR0=1; (Start timer)

SERIAL PORT OPERATION

8051 Serial Communication:
Sections

1 Basics of serial communication

2 8051 connection to RS232

3 8051 serial communication programming

8051 and PC:
- The 8051 module connects to PC by using RS232.
- RS232 is a protocol which supports half-duplex, synchronous/asynchronous, serial communication.
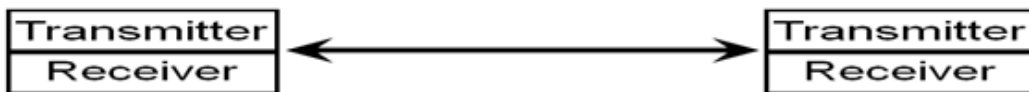- We discuss these terms in following sections.
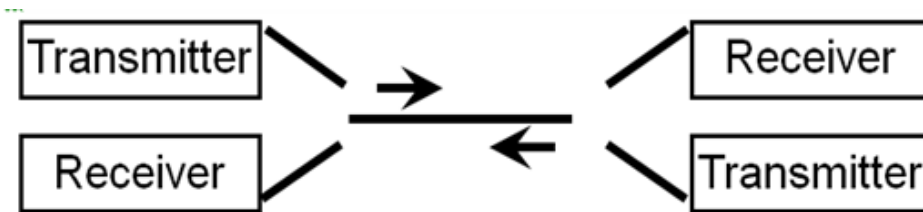
Simplex vs. Duplex Transmission

• Simplex transmission: the data can sent in one direction.
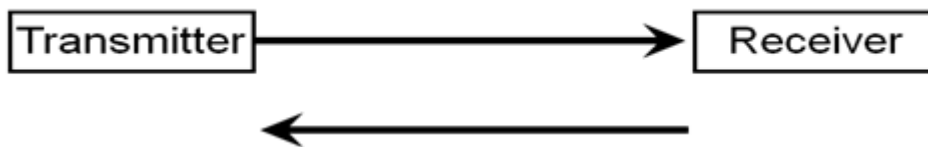
Example: the computer only sends data to the printer.

```
┌─────────────┐                    ┌──────────┐
│ Transmitter ├───────────────────►│ Receiver │
└─────────────┘                    └──────────┘
```

• Duplex transmission: the data can be transmitted and receive

```
┌─────────────┐                    ┌─────────────┐
│ Transmitter │◄──────────────────►│ Transmitter │
│  Receiver   │                    │  Receiver   │
└─────────────┘                    └─────────────┘
```

Half duplex: if the data is transmitted one way at a time

```
┌─────────────┐           ┌──────────┐
│ Transmitter │     →     │ Receiver │
└─────────────┘           └──────────┘
┌─────────────┐     ←     ┌─────────────┐
│  Receiver   │           │ Transmitter │
└─────────────┘           └─────────────┘
```

Full duplex: if the data can go both ways at the same time. Two wire conductors

```
┌─────────────┐                    ┌──────────┐
│ Transmitter ├───────────────────►│ Receiver │
└─────────────┘                    └──────────┘

        ◄───────────────────────
```

Parallel vs. Serial:

 • Computers transfer data in two ways:

   – Parallel
     • data is sent a byte or more a time (fast)
     • Only short distance between two systems
     • The 8-bit data path is expensive
   • Example: printer, hard disks

   – Serial

- The data is sent one bit at a time (slow)
- Long distance (rarely distortion)
- cheap
- The data can be transferred on the telephone line (by using modem.)

## Serial versus Parallel Data Transfer (1/2):
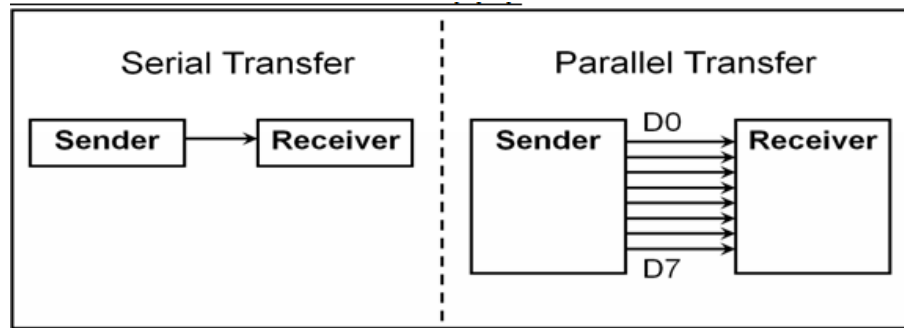
Serial Communication:

- How to transfer data?
    - Sender:
        - The byte of data must be converted to serial bits using a parallel-in-serial- out shift register.

        - The bit is transmitted over a single data line.
    - Receiver
        - The receiver must be a serial-in-parallel-out shift register to receive the serial data and pack them into a byte.
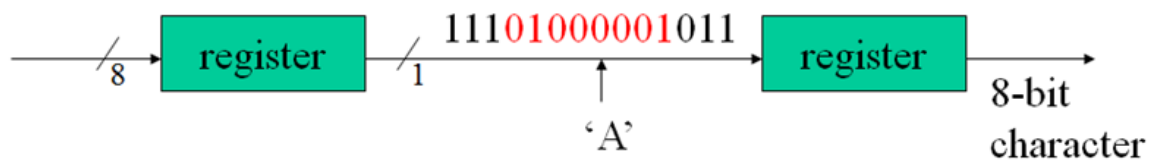
Serial communication uses two methods:

- **In synchronous communication**, data is sent in blocks of bytes.
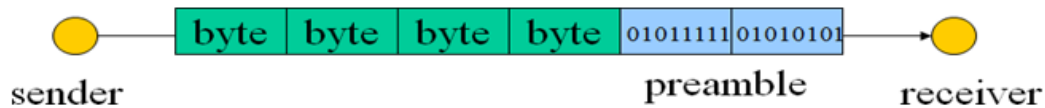
– **In asynchronous communication**, data is sent in bytes.
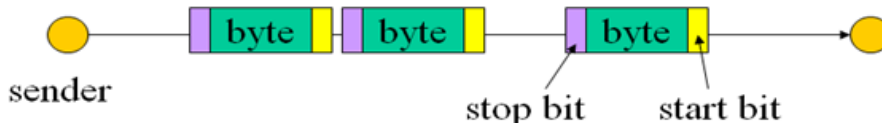
UART & USART:

- It is possible to write software to use both methods, but the programs can be tedious and long.

- Special IC chips are made for serial communication:
  – USART (universal synchronous-asynchronous receiver-transmitter)
  – UART (universal asynchronous receiver-transmitter)
- The 8051 chip has a built-in UART.

8051 Serial Communication:

- The 8051 has serial communication capability built into it.
  – Half-duplex
  – Asynchronous mode only.
- How to detect that a character is sent via the line in the asynchronous mode?
  – Answer: Data framing!

RS232 Standard:

- RS232 is an interfacing standard which is set by the Electronics Industries Association (EIA) in 1960.

  – RS232 is the most widely used serial I/O interfacing standard.

  – RS232A (1963), RS232B (1965) and RS232C (1969), now is RS232E
- Define the voltage level, pin functionality, baud rate, signal meaning, communication distance.
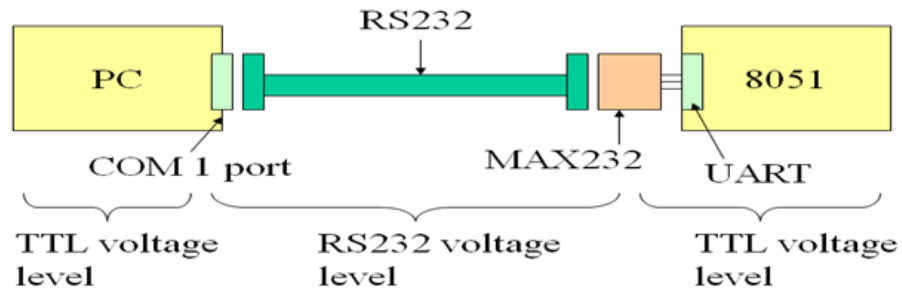
RS232 Voltage Level:

- The input and output voltage of RS232 is not of the TTL compatible.
  – RS232 is older than TTL.
- We must use voltage converter (also referred to as line driver) such as MAX232 to convert the TTL logic levels to the RS232 voltage

level, and vice versa.

    – MAX232, TSC232, ICL232

MAX232:

- MAX232 IC chips are commonly referred to as line drivers.

RS232 pins:

- Figure shows the RS232 connector DB-25.

- Table shows the pins and their labels for the RS232 cable.
  - DB-25P : plug connector (male)
  - DB-25S: socket connector (female)
- Figure shows DB9 connector and Table shows the signals.
  - IBM version for PC.
- All the RS 232 pin function definitions of Tables and are from the DTE point of view.

RS232 Connector DB-25:

RS232 Connector DB-25:
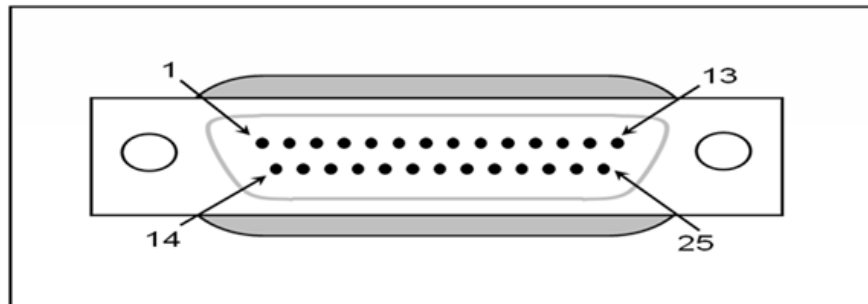
RS232 Pins (DB-25) for DTE (1/2):

| Pin	Description		Pin	Description
1	Protective ground		14	Secondary transmitted data
2	Transmitted data (TxD)		15	Transmit signal element timing
3	Received data (RxD)		16	Secondary received data
4	Request to send (RTS)		17	Receive signal element timing
5	Clear to send (CTS)		18	Unassigned
6	Data set ready (DSR)		19	Secondary request to sent
7	Signal ground (GND)		20	Data terminal ready (DTR)
8	Data carrier detect (DCD)		21	Signal quality detector
9/10	Reserved for data testing		22	Ring indicator
11	Unassigned		23	Data signal rate select
12	Secondary data carrier detect		24	Transmit signal element timing
13	Secondary clear to send		25	Unassigned

TxD and RxD pins in the 8051:

- In 8051, the data is received from or transmitted to
    - RxD: received data (Pin 10, P3.0)
    - TxD: transmitted data (Pin 11, P3.1)
- TxD and RxD of the 8051 are TTL compatible.
- The 8051 requires a line driver to make them RS232 compatible.
    - One such line driver is the MAX232 chip.

8051 Pin Diagram:
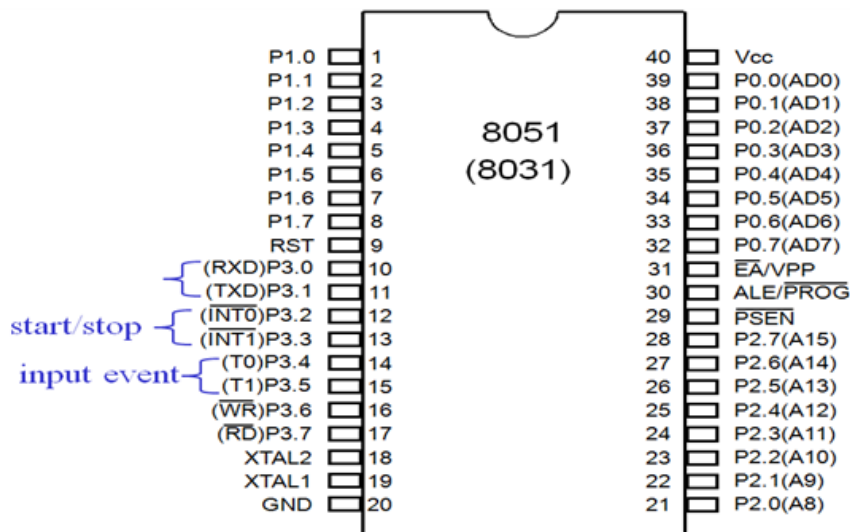
MAX232 (1/2):
- MAX232 chip converts from RS232 voltage levels to TTL voltage levels, and vice versa. MAX232 uses a +5V power source which is the same as the source voltage for the 8051.
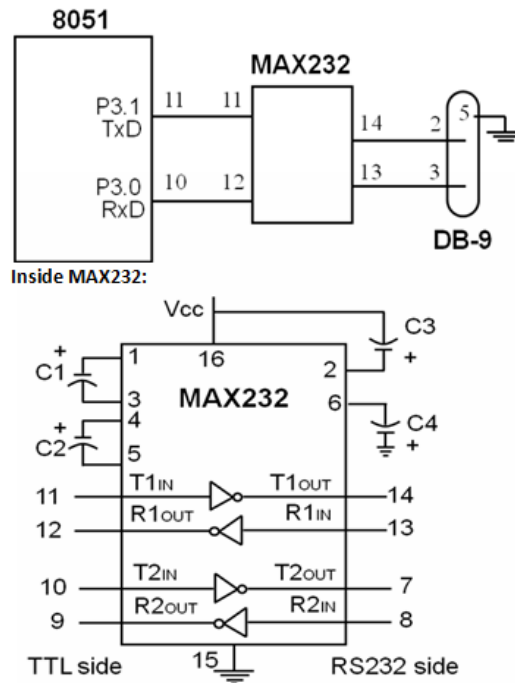
8051 Serial Communication Programming:

PC Baud Rates:

- PC supports several baud rates.

– You can use netterm, terminal.exe, stty, ptty to send/receive data.
- Hyper terminal supports baud rates much higher than the ones list in the Table.

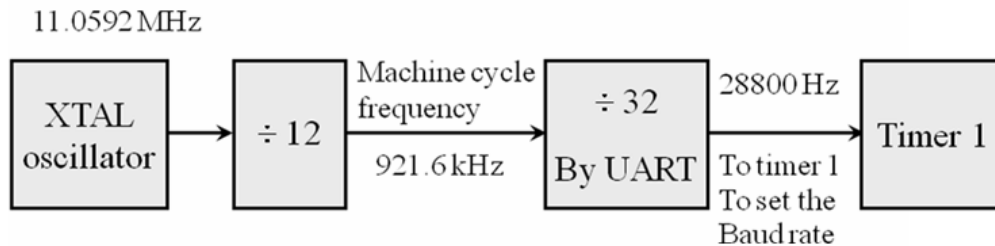| 110 bps |
| 150 |
| 300 |
| 600 |
| 1200 |
| 2400 |
| 4800 |
| 9600 (default) |
| 19200 |

Note: Baud rates supported by
486/Pentium IBM PC BIOS.

Baud Rates in the 8051 (1/2):

- The 8051 transfers and receives data serially at many different baud rates by using

UART.

- UART divides the machine cycle frequency by 32 and sends it to Timer 1 to set the baud rate.

- Signal change for each roll over of timer 1

Baud Rates in the 8051:

- Timer 1, mode 2 (8-bit, auto-reload)
- Define TH1 to set the baud rate.

    - XTAL = 11.0592 MHz
        - The system frequency = 11.0592 MHz / 12 = 921.6 kHz
        - Timer 1 has 921.6 kHz/ 32 = 28,800 Hz as source.

        - TH1=FDH means that UART sends a bit every 3 timer source.

    - Baud rate = 28,800/3= 9,600 Hz

Example:

With XTAL = 11.0592 MHz, find the TH1 value needed to have the following baud rates. (a) 9600 (b) 2400 (c) 1200

Solution:

With XTAL = 11.0592 MHz, we have:

The frequency of system clock = 11.0592 MHz / 12 = 921.6 kHz The frequency sent to timer 1 = 921.6 kHz/ 32 = 28,800 Hz

(a) 28,800 / 3 = 9600   where -3 = FD (hex) is loaded into TH1
(b) 28,800 / 12 = 2400 where -12 = F4 (hex) is loaded into TH1
(c) 28,800 / 24 = 1200 where -24 = E8 (hex) is loaded into TH1 Notice that dividing 1/12th of the crystal frequency by 32 is the default value upon activation of the 8051 RESET pin.

Registers Used in Serial Transfer Circuit:

- SUBF (Serial data buffer)
- SCON (Serial control register)
- PCON (Power control register)
- You can see Appendix H (pages 416-417) for details.
- PC has several registers to control COM1, COM2.

SBUF Register:

- Serial data register: **SBUF**

MOV SBUF,#'A' ;put char 'A' to transmit MOV SBUF,A ;send data from A

MOV A,SUBF ;receive and copy to A

- – An 8-bit register
- – Set the usage mode for two timers
    - For a byte of data to be transferred via the TxD line, it must be placed in the SBUF.

    - SBUF holds the byte of data when it is received by the 8051;s RxD line.
- – Not bit-addressable

SCON Register:

Serial control register: **SCON**

SM0, **SM1** Serial port mode specifier

REN(Receive enable) set/cleared by software to enable/disable reception.

TI Transmit interrupt flag.

RI Receive interrupt flag.

SM2 = **TB8** = **TB8** =0 (not widely used)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

* SCON is bit-addressable.

<u>SM0, SM1:</u>
- SM1 and SM0 determine the framing of data.
    - – SCON.6 (SM1) and SCON.7 (SM0)

- – Only mode 1 is compatible with COM port of IBM PC.
- SM2 enables the multiprocessor communication for mode 2 & 3.
- We make it 0 since we are not using the 8051 in a multiprocessor environment.
  - – SM2=0 : Single processor environment
  - – SM2=1 : multiprocessor environment

REN (Receive Enable):

- SCON.4
- Set/cleared by software to enable/disable reception.

  - – REN=1
    - It enable the 8051 to receive data on the RxD pin of the 8051.
    - If we want the 8051 to both transfer and receive data, REN must be set to 1.

    - **SETB SCON.4**

  - – REN=0
    - The receiver is **disabled.**
    - The 8051 can not receive data.

    - **CLR SCON.4 TB8 (Transfer Bit 8):**
- SCON.3

- TB8 is used for serial modes 2 and 3.
- The 9th bit that will be transmitted in mode 2 & 3.

- Set/Cleared by software.
- SCON.2
- In serial mode 1, RB8 gets a copy of the stop bit when an 8-bit data is received.

TI (Transmit Interrupt Flag):

- SCON.1
- When the 8051 finishes the transfer of the 8-bit character, it raises the TI flag.
- TI is raised by hardware at the beginning of the stop bit in mode 1.
- Must be cleared by software.

RI (Receive Interrupt):

- SCON.0
- Receive interrupt flag. Set by hardware halfway through the stop bit time in mode 1. Must be cleared by software.

- When the 8051 receives data serially via RxD, it gets rid of the start and stop bits and place the byte in the SBUF register.

- Then 8051 rises RI to indicate that a byte.

- RI is raised at the beginning of the stop bit.

SCON Serial Port Control Register (Bit Addressable):

| SM0 | SM1 | SM2 | REN | TB8 | RB8 | TI | RI |

SM0	SCON.7	Serial port mode specifier
SM1	SCON.6	Serial port mode specifier
SM2	SCON.5	Used for multiprocessor communication. (Make it 0)
REN	SCON.4	Set/cleared by software to enable/disable reception.
TB8	SCON.3	Not widely used.
RB8	SCON.2	Not widely used.
TI	SCON.1	Transmit interrupt flag. Set by hardware at the beginning of the stop bit in mode 1. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware halfway through the stop bit time in mode 1. Must be cleared by software.

Note: Make SM2, TB8, and RB8 = 0.

INTERRUPT STRUCTURE OF 8051

8051 interrupts

Interrupts are the events that temporarily suspend the main program, pass the control to the external sources and execute their task. It then passes the control to the main program where it had left off.

8051 has 5 interrupt signals, i.e. INT0, TFO, INT1, TF1, RI/TI. Each interrupt can be enabled or disabled by setting bits of the IE register and the whole interrupt system can be disabled by clearing the EA bit of the same register.

E (Interrupt Enable) Register

This register is responsible for enabling and disabling the interrupt. EA register is set to one for enabling interrupts and set to 0 for disabling the interrupts. Its bit sequence and their meanings

are shown in the following figure.

EA	-	-	ES	ET1	EX1	ET0	EX0

EA	IE.7	It disables all interrupts. When EA = 0 no interrupt will be acknowledged and EA = 1 enables the interrupt individually.
-	IE.6	Reserved for future use.
-	IE.5	Reserved for future use.
ES	IE.4	Enables/disables serial port interrupt.
ET1	IE.3	Enables/disables timer1 overflow interrupt.
EX1	IE.2	Enables/disables external interrupt1.
ET0	IE.1	Enables/disables timer0 overflow interrupt.
EX0	IE.0	Enables/disables external interrupt0.

IP (Interrupt Priority) Register

We can change the priority levels of the interrupts by changing the corresponding bit in the Interrupt Priority (IP) register as shown in the following figure.

- A low priority interrupt can only be interrupted by the high priority interrupt, but not interrupted by another low priority interrupt.
- If two interrupts of different priority levels are received simultaneously, the request of higher priority level is served.
- If the requests of the same priority levels are received simultaneously, then the internal polling sequence determines which request is to be serviced.

-	-	PT2	PS	PT1	PX1	PT0	PX0
bit7	bit6	bit5	bit4	bit3	bit2	bit1

-	IP.6	Reserved for future use.
-	IP.5	Reserved for future use.
PS	IP.4	It defines the serial port interrupt priority level.
PT1	IP.3	It defines the timer interrupt of 1 priority.
PX1	IP.2	It defines the external interrupt priority level.
PT0	IP.1	It defines the timer0 interrupt priority level.
PX0	IP.0	It defines the external interrupt of 0 priority level.

Interrupt programming in 8051

1. **Timer Interrupt Programming**: In microcontroller Timer 1 and Timer 0 interrupts are generated by time register bits TF0 AND TF1. This timer interrupts programming by C code involves:
   - Selecting the configuration of TMOD register and their mode of operation.
   - Enables the IE registers and corresponding timer bits in it.
   - Choose and load the initial values of TLx and THx by using appropriate mode of operation.
   - Set the timer run bit for starting the timer.
   - Write the subroutine for a timer and clears the value of TRx at the end of the subroutine.

Let's see the timer interrupt programming using Timer0 model for blinking LED using interrupt method:

- #include< reg51 .h>
- sbit Blink Led = P2^0; // LED is connected to port 2 Zeroth pin
- void timer0_ISR (void) interrupt 1 //interrupt no. 1 for Timer0
- {
- Blink Led=~Blink Led; // Blink LED on interrupt
- TH0=0xFC; // loading initial values to timer
- TL0=0x66;
- }
- void main()
- {
- TMOD=0x0l;   // mode 1 of Timer0
- TH0 = 0xFC: // initial value is loaded to timer
- TL0 = 0x66:
- ET0 =1;   // enable timer 0 interrupt
- TR0 = 1;   // start timer
- while (1);   // do nothing

- }

- Interrupts are basically the events that temporarily suspend the main program, pass the control to the external sources and execute their task. It then passes the control to the main program where it had left off.

- 8051 has five interrupts. These interrupts are INT0, INT1,TO ,T1 , TI/RI. All of the interrupts can be enabled or disabled by using the IE (interrupt enable) register.

- The interrupt addresses of these interrupts are like below:

Interrupt	Address
INT0	0003H
INT1	000BH
T0	0013H
T1	001BH
TI/RI	0023H

- **Interrupt Enable (IE)Register**

- This register can be used to enable or disable interrupts programmatically. This register is an SFR. The address is A8H. This byte is bit addressable. So it can be programmed by the user. The bits in this register has a different meaning. The register structure is looking like this:

Bit Address	AF	AE	AD	AC	AB	AA	A9	A8
Bit Details	EA	X	X	ES	ET1	EX1	ET0	EX0

- Now, let us see the bit details and different operations when the value is low (0) and high(1).

Bit Details	High Value(1)	Low Value(0)
EA	Least significant 5 bits can decide enable or disable of these five interrupts.	Disable all five interrupts. It just ignores the rest five bits.
ES	Enable Serial Port Interrupt	Disable Serial Port Interrupt
ET1	Enable Timer1 interrupt	Disable Timer1 interrupt
EX1	Enable external interrupt 1 (INT1)	Disable external interrupt 1 (INT1)
ET0	Enable Timer0 interrupt	Disable Timer0 interrupt
EX0	Enable external interrupt 0 (INT0)	Disable external interrupt 0 (INT0)

- **Interrupt Priority (IP) Register**
- All of these five interrupts can be in one or two interrupt level. The priority levels are level 1 and level 0. Priority level 1 indicates the higher priority, and level 0 indicates lower priority. This IP register can be used to store the priority levels for each interrupt. This is also a bit addressable SFR. Its address is B8H.

Bit Address	BF	BE	BD	BC	BB	BA	B9	B8
Bit Details	X	X	X	PS	PT1	PX1	PT0	PX0

- Now, let us see the bit details and different operations when the value is low (0) and high(1).

Bit Details	High Value(1)	Low Value(0)
PS	Set 1 level priority of Serial port interrupt	Set 0 level priority of Serial port interrupt
PT1	Set 1 level priority of Timer1 interrupt	Set 0 level priority of Timer1 interrupt
PX1	Set 1 level priority of external interrupt 1 (INT1)	Set 0 level priority of external interrupt 1 (INT1)
PT0	Set 1 level priority of Timer0 interrupt	Set 0 level priority of Timer0 interrupt
PX0	Set 1 level priority of external interrupt 0 (INT0)	Set 0 level priority of external interrupt 0 (INT0)

- When all of the five interrupts are in same priority level, and if all of the interrupts are enabled, then the sequence of interrupts will be INT0, T0, INT1, T1, TI/R I.

- Some specific priority register value can be used to maintain the priorities of the interrupts. Let the value of Priority register is xxx00101 indicates the sequence INT0, INT1, TI/RI, T1, T0. But all of the sequences are not feasible. Like INT0, INT1, TI/RI, T1, T0 is not valid.

- **External Interrupt**

- The external interrupts of 8051 are INT0and. INT1 These interrupts can be programmed to either edge-triggered or level triggered. The TCON register can be used top program external interrupts to edge or level triggered. The TCON is Timer Control. TCON is another bit addressable SFR. Here the address is 88H.

Bit Address	8F	8E	8D	8C	8B	8A	89	88
Bit Details	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

- Now, let us see the bit details and different operations when the value is low (0) and high (1).

Bit Details	High Value(1)	Low Value(0)
IT0	Set (INT0) as negative edge triggered input.	Set (INT0) as active low level triggered input.
IT1	Set (INT1) as negative edge triggered input.	Set (INT1) as active low level triggered input.
IE0	This will be 1, when INT0is activated as level triggered.	This will be 0, when INT0is activated as edge triggered.
IE1	This will be 1, when INT1 is activated as level triggered.	This will be 0, when INT1 is activated as edge triggered.
TR0	Set Timer0 as run mode	Set Timer0 as stop mode.
TR1	Set Timer1 as run mode	Set Timer1 as stop mode.
TF0	High when Timer T0 overflow occurs.	After resetting the timer T0 thiswill also be changed to 0 state
TF1	High when Timer T1 overflows occurs.	After resetting the timer T1 this will also be changed to 0 state.

- The IT0 and IT1 are stands for Interrupt Type. These bits are used to decide whether the INT0 and INT1 will be level trigged or edge triggered.

- IE0 and IE1 bits are used to indicate the status of external interrupts. This bit can be set or reset by the microcontroller itself.

- The first four bits are the status information about timers. When TR0 and TR1 are 1, it indicates the running mode of the timers. These bits provide software control over the running of timers. Timers can also be controlled by the hardware. The priority of hardware mode is higher than the software mode.

- The TF0 and TF1 are used to indicate the overflow of timer T0 and T1 respectively. When over flow occurs these flags are set to 1. When the interrupt is handled by some interrupt service subroutine (ISS), these will be 0.

- **Serial Port Interrupt**

- The serial ports can be used either Transmitting mode or reception mode. The interrupt status for the Transmission is provided by TI, and status for Reception is provided by RI. These are two bits of SCON (Serial Control). This is also a bit addressable SFR. The address is98H

Bit Address	9F	9E	9D	9C	9B	9A	99	98
Bit Details	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

- The significance of these bits are as follows

Bit Details	Description
SM0	This is Serial Port Mode 0 shift register
SM1	This is Serial Port Mode 1 (8-bit UAR + variable)
SM2	Enable multiprocessor communication in the mode 2 or 3

Bit Details	Description
REN	Set or reset by the software to enable or disable the Reception
TB8	It indicates the 9th bit that will be transmitted in mode 2 or 3. It can be set or reset by the software
RB8	In mode 2 or 3, the 9th bit was received in mode 1.
TI	The transmission interrupt flag. It can be set by hardware.
RI	The receiver interrupt flag. It can be set by hardware but must be reset by software.

- The interrupt control system of 8051 is like below:
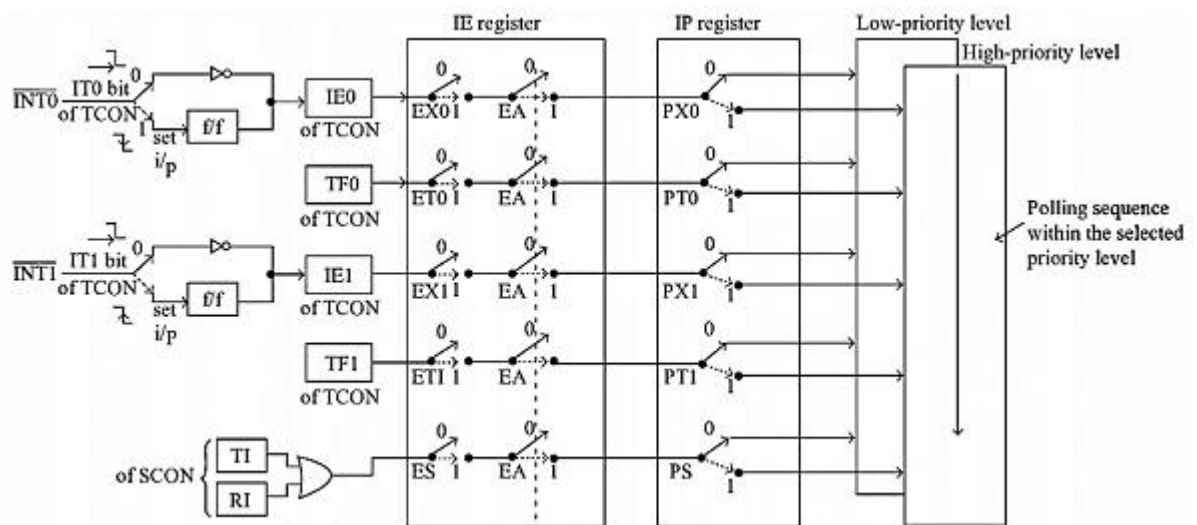
MEMORY AND I/O INTERFACING WITH 8051

Real world interfacing of 8051 with external memory

A single microcontroller can serve several devices. There are two ways to do that is interrupts or polling. In the interrupt method, whenever any device needs its services, the device notifies the micro controller interrupts whatever it is doing and serves the device. The program which is

associated with the interrupt is called the interrupt service routine (ISR) or Interrupt handler. In polling, the microcontrollers continuously monitor the status of several devices and serve each of them as certain conditions are met. The advantage of interrupts is that microcontroller can serve many devices. Each device can get the attention of microcontroller based on the priority assigned to it. For the polling method; it is not possible to assign priority. In interrupt method the microcontroller can also ignore (mask) a device request for service. This is not possible in polling method. The polling method wastes much of microcontrollers" time by polling devices that do not need service, so interrupts are preferred.

In 8051 TL is compatible. When its need to interface with Input/output device RS232 use interface circuit MAX 232.
We have seen that a typical 8051 Microcontroller has 4KB of ROM and 128B of RAM (most modern 8051 Microcontroller variants have 8K ROM and 256B of RAM).

The designer of an 8051 Microcontroller based system is not limited to the internal RAM and ROM present in the 8051 Microcontroller. There is a provision of connecting both external RAM and ROM i.e. Data Memory and Program.

The reason for interfacing external Program Memory or ROM is that complex programs written in high – level languages often tend to be larger and occupy more memory.

Another important reason is that chips like 8031 or 8032, which doesn"t have any internal ROM, have to be interfaced with external ROM.

A maximum of 64B of Program Memory (ROM) and Data Memory (RAM) each can be interface with the 8051 Microcontroller.

The following image shows the block diagram of interfacing 64KB of External RAM and 64KB of External ROM with the 8051 Microcontroller.
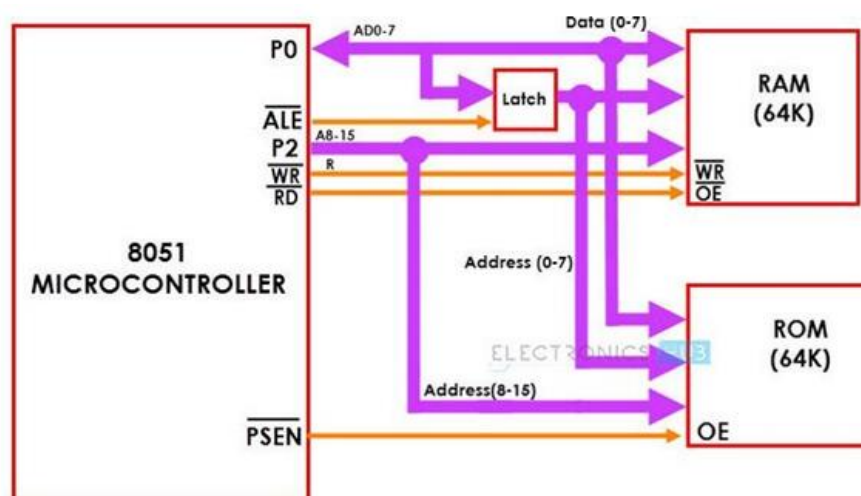
The 8051 Microcontroller Memory Organization, Internal ROM and RAM and how to

interface external ROM and RAM with 8051 Microcontroller.

Expansion of I/O ports

To perform input/output operation Microcontrollers 8051 possess 4 I/O ports each consisting of 8-bit, which can be configured as input or output. Hence, out of 40 pins only 32 input/output pins are allowed for the microcontrollers that are connected with the peripheral devices. As microprocessor does not have inbuilt input/output operation, it can be rectified by using microcontrollers 8051.

Pin configuration, i.e. the pin can be configured as 1 for input and 0 for output as per the logic state.

Input/output (I/O) pin – excluding port P0 which does not have pull-up resistors built in, all other circuits within the microcontroller should be connected to one of its pins.

Input–pin To a bit of P register logic 1 is applied. The output FE transistor is turned off and the other pin duly remains connected to the power supply voltage with the help of a pull-up resistor possessing high resistance.

Port-0 The P0 (zero) port performs two functions −Whenever external memory is used then the lower address byte (addresses A0A7) is applied on it, otherwise all bits of this port are configured as input/output.

When P0 port is configured as an output then the remaining ports consisting of pins with built-in pull-up resistor are connected to its end of 5V power supply, the port consisting of pins had left off this resistor.

Input Configuration

If the port consisting of pin is configured as an input, then it needs to act as "floats", i.e. the input is possessed with unlimited input resistance and in-determined potential.

Output Configuration

When the pin is configured as an output, then it acts as an "open drain". When the logic 0 to a port bit is applied then the appropriate pin will be connected to ground (0V), and when the logic 1 is applied, the external output would continue to float.

In this output configuration if the logic 1 (5V) power supply is applied then it is very essential to build an external pull-up resistor for this configuration.

Port 1

P1 is regarded as a true I/O port as it does not consist of any alternative functions as in P0, but this port could be configured as general I/O only. It has a built-in pull-up resistor and is completely compatible with TTL circuits.

Port 2

P2 is also called as "quasi-bidirectional port" because of its output pull-up resistors. It acts similar to P0 when the external memory is applied. Pins of this port used as input/output which occupy addresses intended for the external memory chip. This port can be used for upper order address byte with addresses A8-A15 (for external memory). This port acts as general input/output port when memory is not added then it functions similar to Port 1.

Port 3

Port 3 functions are similar to other ports except that the logic 1 must be applied to bit of the P3 register which should be appropriate. It is multifunctional port and can be used as simple input/output port.

Pins Current Limitations

When configuration of pins takes place as an output (i.e. logic 0), then the single port pins can receive a current of 10mA.

When these pins are configured as inputs (i.e. logic 1), very weak current is generated by the built- in pull-up resistors, but it can be activated up to 4 TTL inputs of LS series.

When all the 8 bits of a port is said to be active, then the total current must be limited to 15mA (port P0: 26mA).

When all the ports (32 bits) are said to be active, then the total maximum current must be limited to 71mA.

Interfacing I/O Devices

Every electrical and electronics project designed to develop electronic gadgets that are frequently used in our day-to-day life utilizes microcontrollers with appropriate interfacing devices. There are different types of applications that are designed using microcontroller based projects. In maximum number of applications, the microcontroller is connected with some external devices called as interfacing devices for performing some specific tasks. For example, consider security system with a user changeable password project, in which an interfacing device, keypad is interfaced with microcontroller to enter the password.
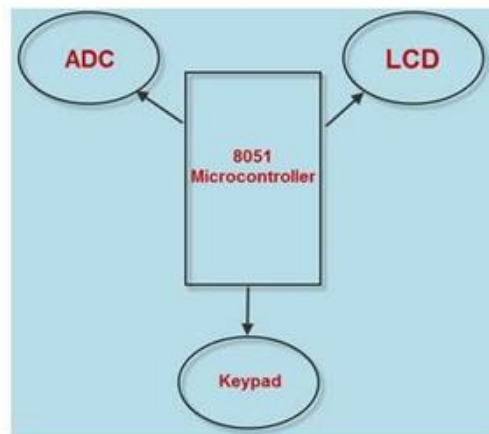
Interfacing Devices

Interfacing can be defined as transferring data between microcontrollers and interfacing peripherals such as sensors, keypads, microprocessors, analog to digital converters or ADC, LCD displays, motors, external memories, even with other microcontrollers, some other interfacing peripheral devices and so on or input devices and output devices. These devices that are interfacing with 8051 microcontroller are used for performing special tasks or functions are called as interfacing devices.

Interfacing is a technique that has been developed and being used to solve many composite problems in circuit designing with appropriate features, reliability, availability, cost, power consumption, size, weight, and so on. To facilitate multiple features with simple circuits, microcontroller is interfaced with devices such as ADC, keypad, LCD display and so on.

Intelligent LCD display:

- We examine an intelligent LCD display of two lines,20 char's. per line, that is interfaced to the 8051

Pin out of 16*2 LCD
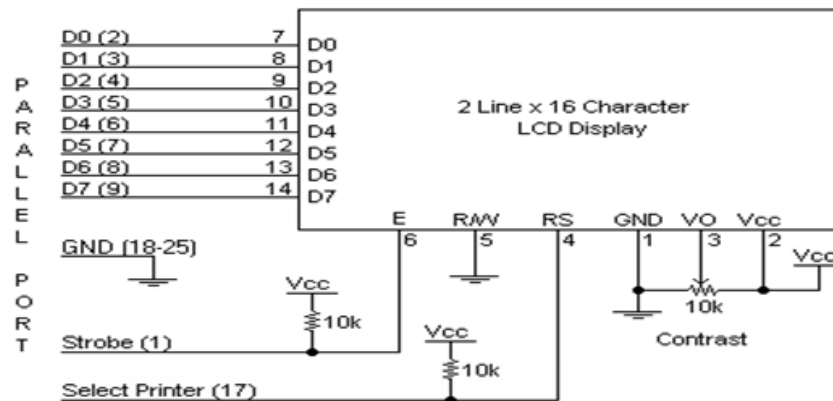
•The displays contains two internal byte wide registers one for commands (RS=0) and the second for char. To be displayed(RS=1)
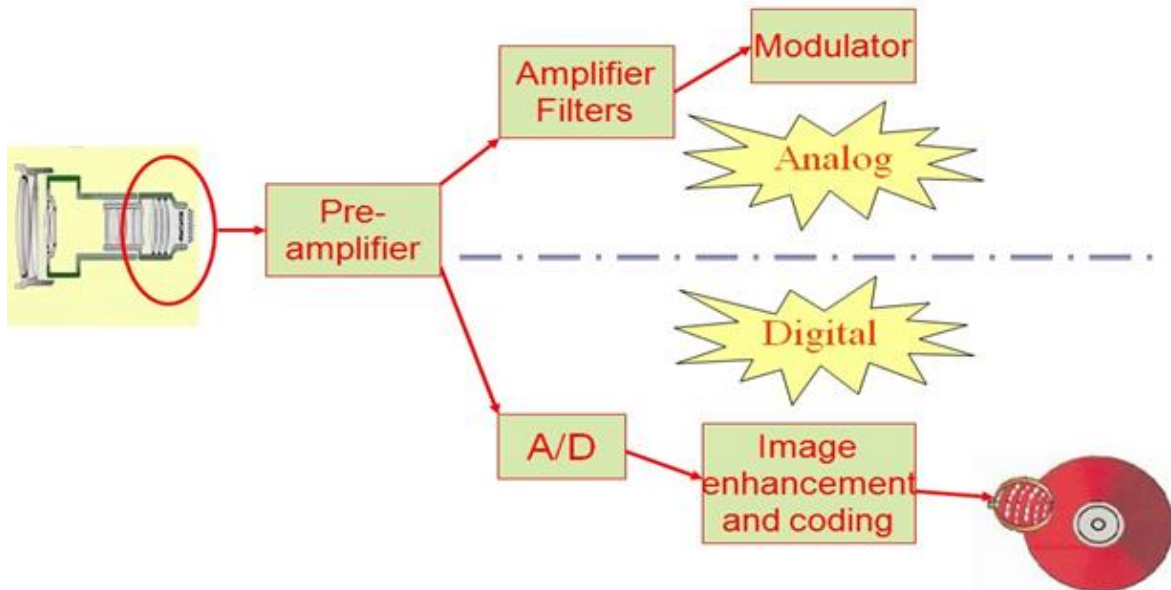
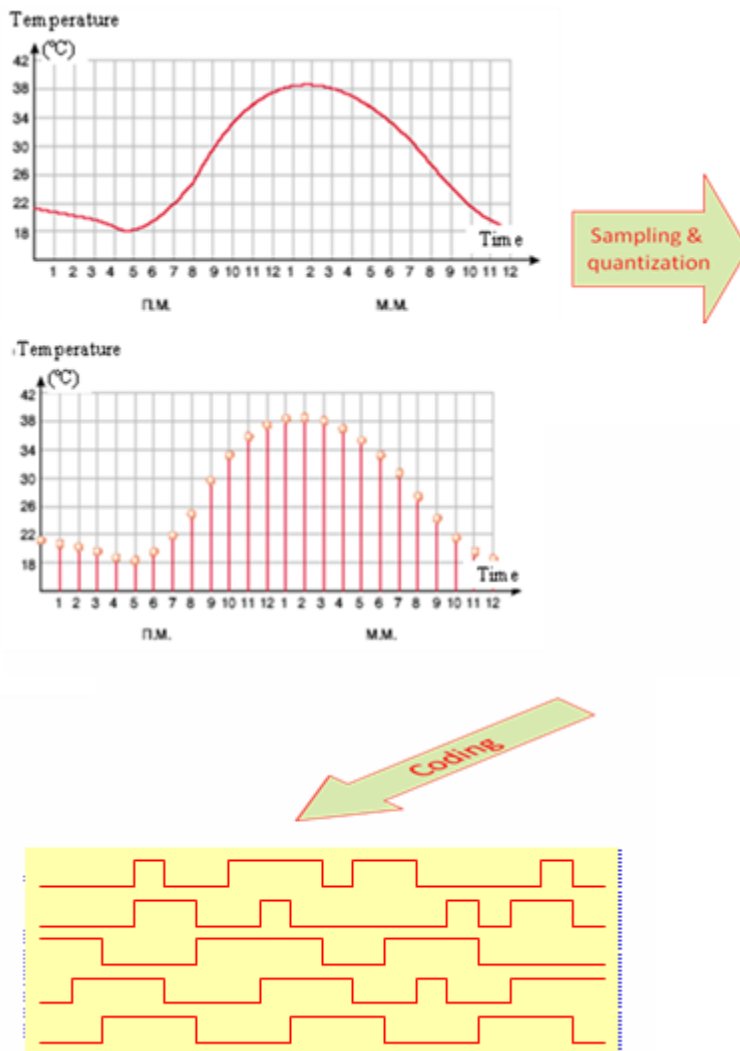**Interface Intelligent LCD Circuit with 8051**
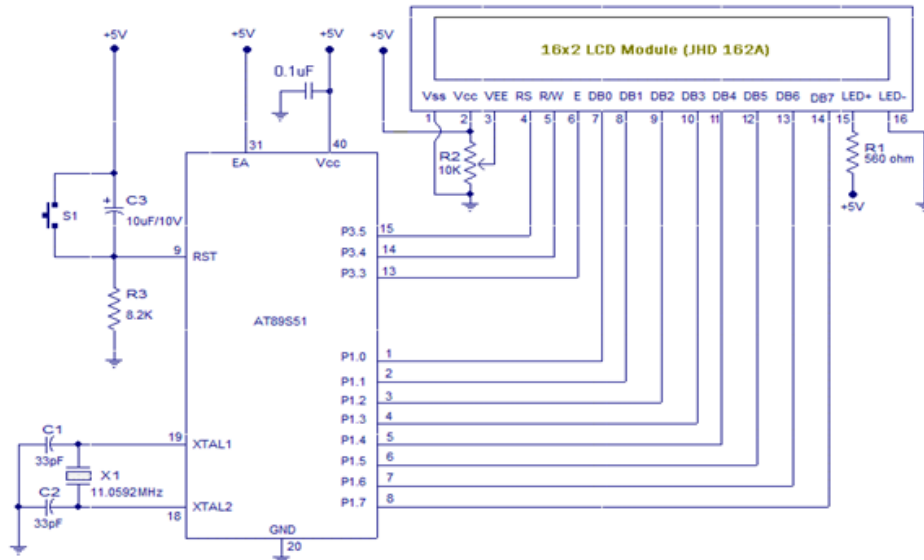
Data converters:

- Analog to Digital Converters (ADC)
  - Convert an analog quantity (voltage, current) into a digital code
- Digital to Analog Converters (DAC)
  - Convert a digital code into an analog quantity (voltage, current)

Video (Analog - Digital)

Temperature Recording by a Digital System :

Need for Data Converters:

Digital processing and storage of physical quantities (sound, temperature, pressure etc) exploits the advantages of digital electronics

– Better and cheaper technology compared to the analog

– More reliable in terms of storage, transfer and processing

• Not affected by noise

– Processing using programs (software)

• Easy to change or upgrade the system

– (e.g. Media Player 7 ⬜ Media Player 8 ή Real Player)

Digital to Analog Converters:

• The analog signal at the output of a D/A converter is linearly proportional to the binary code at the input of the converter.

- If the binary code at the input is 0001 and the output voltage is 5mV, then
- If the binary code at the input becomes 1001, the output voltage will become 45mv.....
- If a D/A converter has 4 digital inputs then the analog signal at the output can have one out of 16 values
- If a D/A converter has N digital inputs then the analog signal at the output can have one out of 2^N values.

D3	D2	D1	D0	Vout (mV)
0	0	0	0	0
0	0	0	1	5
0	0	1	0	10
0	0	1	1	15
0	1	0	0	20
0	1	0	1	25
0	1	1	0	30
0	1	1	1	35
1	0	0	0	40
1	0	0	1	45
1	0	1	0	50
1	0	1	1	55
1	1	0	0	60
1	1	0	1	65
1	1	1	0	70
1	1	1	1	75

Characteristics of Data Converters:

1. Number of digital lines
   - The number bits at the input of a D/A (or output of an A/D) converter.
   - Typical values: 8-bit, 10-bit, 12-bit and 16-bit
   - Can be parallel or serial
2. Microprocessor Compatibility
   - Microprocessor compatible converters can be connected directly on the microprocessor bus as standard I/O devices
   - They must have signals like CS, RD, and WR
     - Activating the WR signal on an A/D converter starts the conversion process.
3. Polarity
   - Polar: the analog signals can have only positive values
   - Bipolar: the analog signals can have either a positive or a negative value
4. Full-scale output

- The maximum analog signal (voltage or current)
- Corresponds to a binary code with all bits set to 1 (for polar converters)
- Set externally by adjusting a variable resistor that sets the Reference Voltage (or current)

5. Resolution
   - The analog voltage (or current) that corresponds to a change of 1LSB in the binary code
   - It is affected by the number of bits of the converter and the Full Scale voltage (VFS)
   - For example if the full-scale voltage of an 8-bit D/A converter is 2.55V the the resolution is:

   $$VFS/(2^N-1) = 2.55 / (2^8-1)\ 2.55/255 = 0.01\ V/LSB = 10mV/LSB$$

6. Conversion Time
   - The time from the moment that a "Start of Conversion" signal is applied to an A/D converter until the corresponding digital value appears on the data lines of the converter.

ADC RESPONSE TYPES:

- Linear
  - Most common
- Non-linear
  - Used in telecommunications, since human voice carries more energy in the low frequencies than the high.
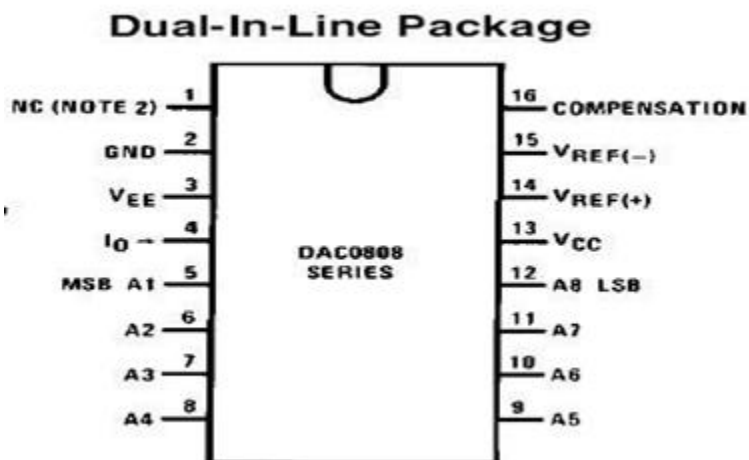
ADC TYPES:

- Direct Conversion
  - Fast
  - Low resolution
- Successive approximation
  - Low-cost
  - Slow
  - Not constant conversion delay
- Sigma-delta
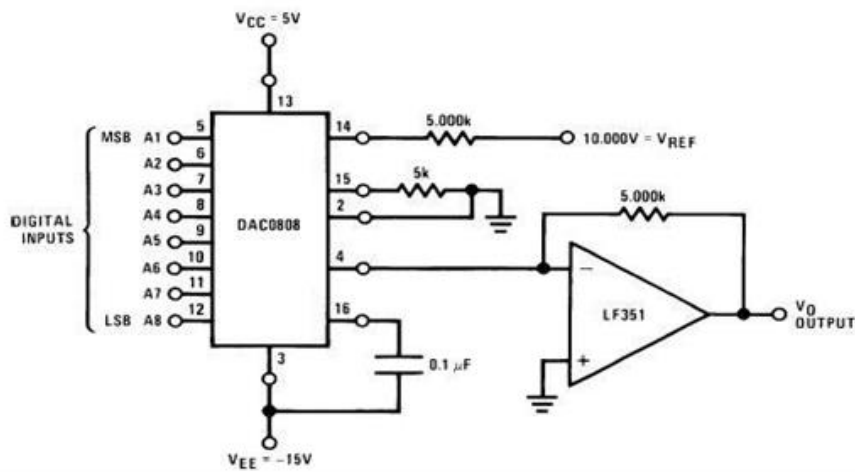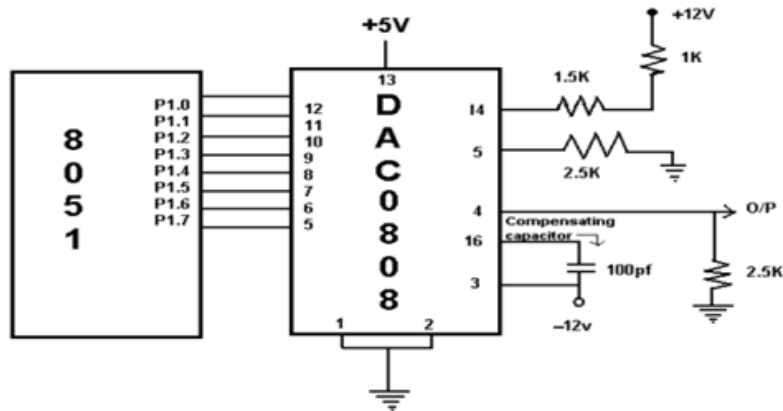 - High resolution,
  - low-cost,
  - high accuracy
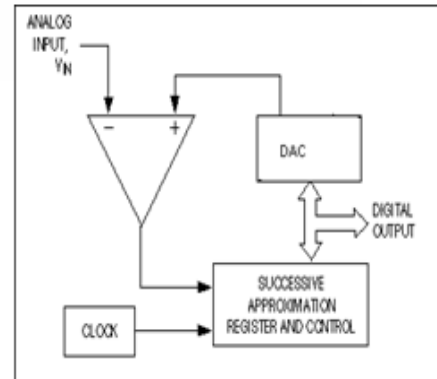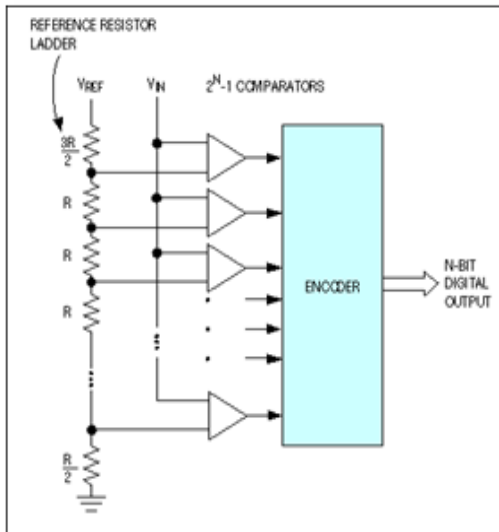
D/A Converter:

- Conversion between the analog and digital world requires the use of integrated circuits that have been designed to interface with computer

- D/A converter is a generic R-2R type ,based on several commercial models, is connected to ports 1 to 3
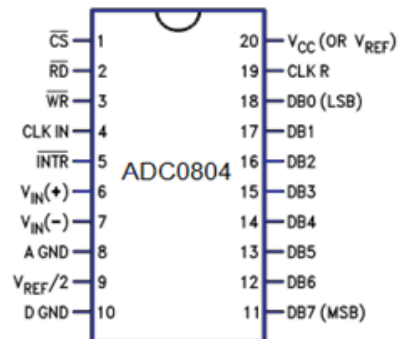
Pin diagram of DAC 0808

Dual-In-Line Package

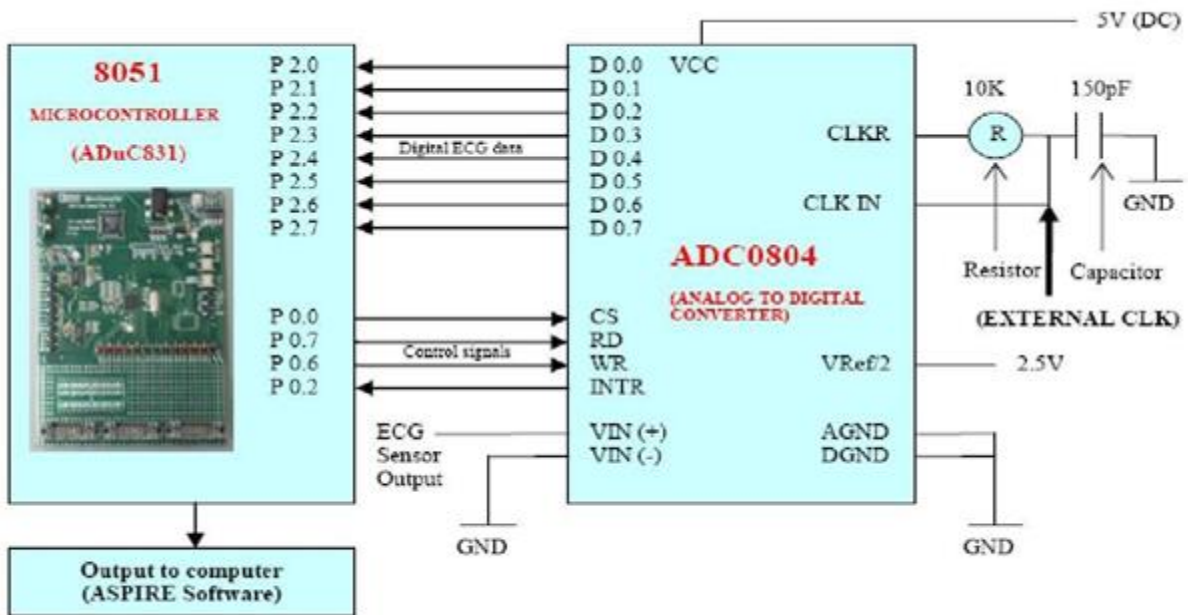| Pin	Signal		Pin	Signal
1	NC (NOTE 2)		16	COMPENSATION
2	GND		15	$V_{REF(-)}$
3	V_{EE}		14	$V_{REF(+)}$
4	$I_O \rightarrow$		13	V_{CC}
5	MSB A1		12	A8 LSB
6	A2		11	A7
7	A3		10	A6
8	A4		9	A5

Interfacing DAC with 8051:

Analog to digital converters:

The easiest A/D converters to use are the flash types which make conversions based on an array of internal comparators.

Pin out of ADC0804:

ADC Interfacing with 8051:

Stepper Motor interacting with 8051

Stepper motors are basically two types: Unipolar and Bipolar. **Unipolar stepper** motor generally has five or six wire, in which four wires are one end of four stator coils, and other end of the all four coils is tied together which represents fifth wire, this is called common wire (common point). Generally there are two common wire, formed by connecting one end of the two-two coils as shown in below figure. Unipolar stepper motor is very common and popular because of its ease of use.

Unipolar Stepper Motor
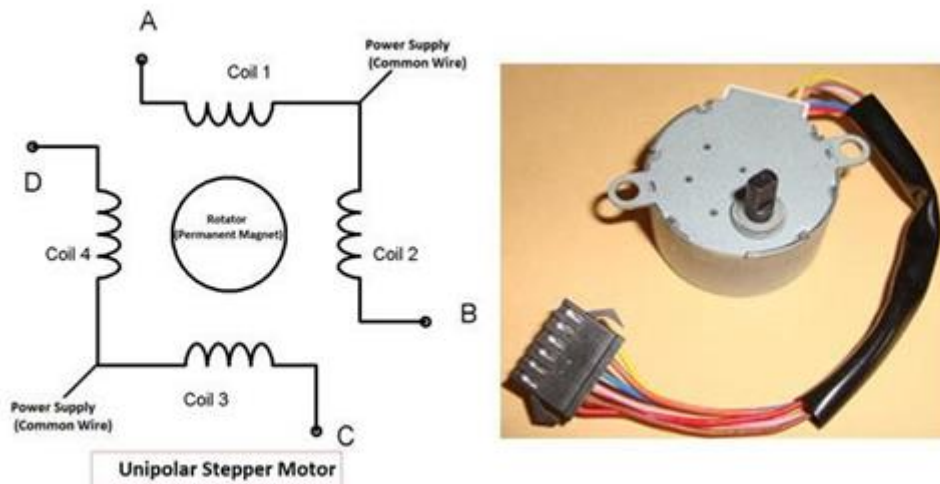
In **Bipolar stepper** motor there is just four wires coming out from two sets of coils, means there are no common wire.

Stepper motor is made up of a stator and a rotator. Stator represents the four electromagnet coils which remain stationary around the rotator, and rotator represents permanent magnet which rotates. Whenever the coils energized by applying the current, the electromagnetic field is created, resulting the rotation of rotator (permanent magnet). Coils should be energized in a particular sequence to make the rotator rotate. On the basis of this "sequence" we can divide the working method of **Unipolar stepper motor** in three modes: Wave drive mode, full step drive mode and half step drive mode.
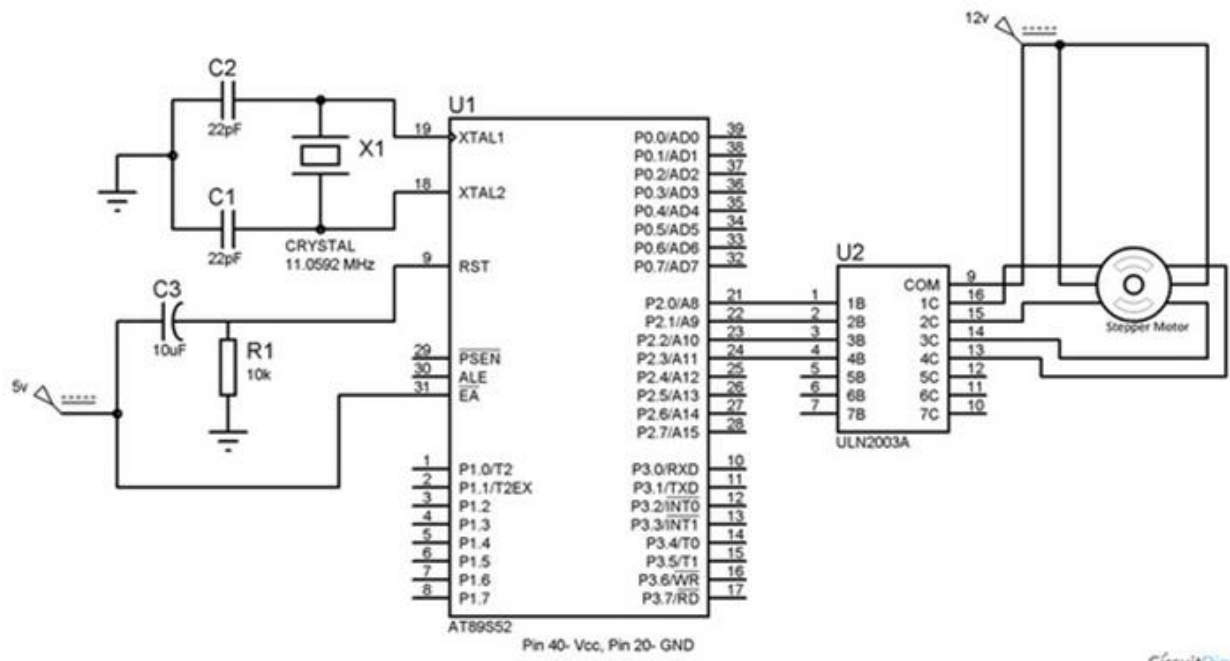
Wave drive mode: In this mode one coil is energized at a time, all four coil are energized one after another. It produces less torque in compare with Full step drive mode but power consumption is less. Following is the table for producing this mode using microcontroller, means we need to give Logic 1 to the coils in the sequential manner.

Steps	A	B	C	D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Full Drive mode: In this, two coil are energized at the same time producing high torque. Power consumption is higher. We need to give Logic 1 to two coils at the same time, then to the next two coils and so on.

Steps	A	B	C	D
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

8051 doesn't provide enough current to drive the coils so we need to use a **current driver IC that is ULN2003A**. ULN2003A is the array of seven NPN Darlington transistor pairs. Darlington pair is constructed by connecting two bipolar transistors to achieve high current amplification. In ULN2003A, 7 pins are input pins and 7 pins are output pins, two pins are for Vcc (power supply) and Ground. Here we are using four input and four output pins. We can also use L293D IC in place of ULN2003A for current amplification.