MICROPROCESSOR AND ITS APPLICATIONS

Unit–I

Microcomputer – microprocessor architecture and its operations – memory input/output – addressing modes – instruction classification, format and timings.

Introduction to Microcomputer

The Microprocessor based system (single board microcomputer) consists of microprocessor as CPU, semiconductor memories like EPROM and RAM, input device, output device and interfacing devices. The memories, input device, output device and interfacing devices are called peripherals. The popular input devices are keyboard and floppy disk and the output devices are printer, LED/LCD displays, CRT monitor, etc.

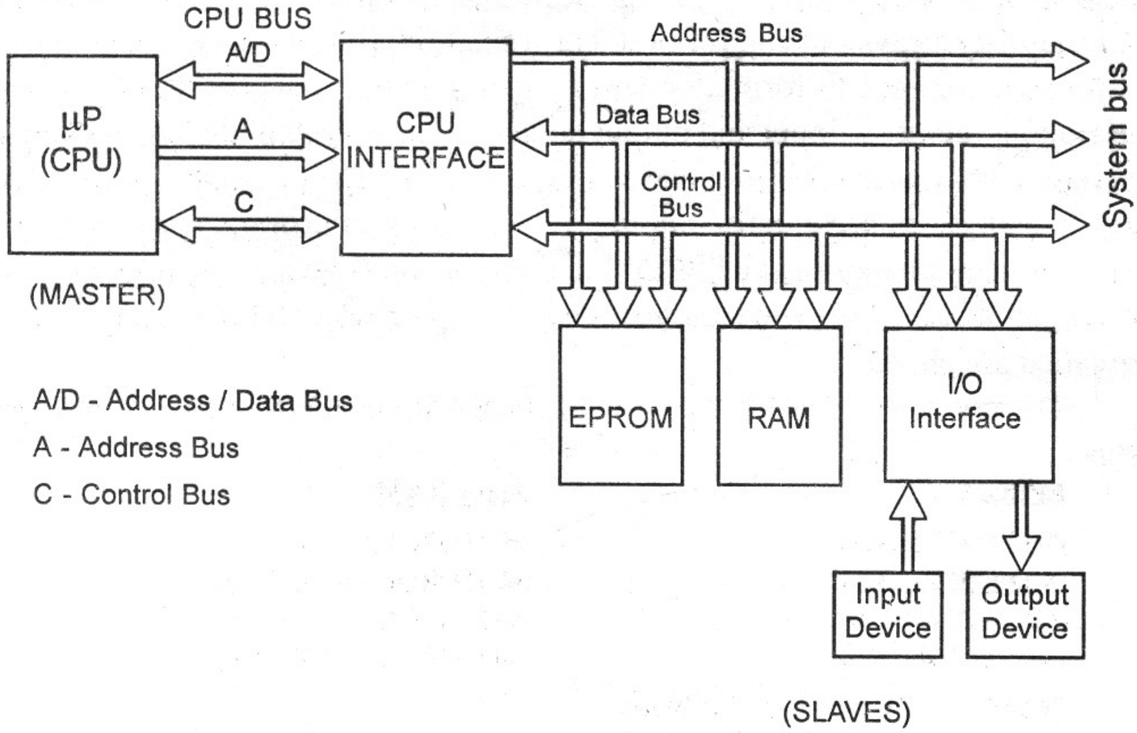


Fig : 1.1 Microprocessor Based System (organisation of microcomputer)

The above block diagram shows the organization of a microprocessor based system. In this system, the microprocessor is the master and all other peripherals are slaves. The master controls all the peripherals and initiates all operations.

The work done by the processor can be classified into the following three groups.

1. Work done internal to the processor
2. Work done external to the processor
3. Operations initiated by the slaves or peripherals.

The work done internal to the processors are addition, subtraction, logical operations, data transfer operations, etc. The work done external to the processor are reading/writing the memory and reading/writing the I/O devices or the peripherals. If the peripheral requires the attention of the master then it can interrupt the master and initiate an operation.

The microprocessor is the master, which controls all the activities of the system. To perform a specific job or task, the microprocessor has to execute a program stored in memory. The program consists of a set of instructions. It issues address and control signals and fetches the instruction and data from memory. The instruction is executed one by one internal to the processor and based on the result it takes appropriate action.

Input

The input section transfers data and instructions in binary from the outside world to the microprocessor. It includes devices such as keyboards, teletypes, and analog-to-digital converters. Typically, a microcomputer includes a keyboard as an input device. The key board has sixteen data keys (o to 9 and A to F) and some additional function keys to perform operations such as storing data and executing programs.

Output

The output section transfers data from the microprocessor to output devices such as light emitting diodes (LEDs), cathode-ray-tubes (CRTs), printers, magnetic tape, or another computer. Typically, single-board computers include LEDs and seven-segment LEDs as output devices.

Memory

Memory stores binary information such as instructions and data, and provides that information to the microprocessor whenever necessary. To execute programs, the microprocessor reads instructions and data from memory and performs the computing operations in its ALU section. Result are either transferred to the output section for display or stored in memory for later use. The memory block has two sections : Read - Only Memory (ROM) and Read / Write Memory (R/WM), popularly known as Random Access Memory (RAM).

The ROM is used to store programs that do not need alterations. The monitor program of a single - board microcomputer is generally stored in the ROM. Program stored in the ROM can only be read; they cannot be altered.

The Read / Write memory (R/WM) is also known as user memory. It is used to store user programs and data. The information stored in this memory can be read and altered easily.

System Bus

The system bus is a communication path between the microprocessor and the peripherals; it is nothing but a group of wires that carries bits. The microcomputer bus is in many ways similar to a one-track, express subway, the microcomputer bus carries bits between the microprocessor and only one peripheral at a time. The same bus is time - shared to communicate with various peripherals, with the timing provided by the control section of the microprocessor.

Address Bus

This is a unidirectional bus, because information flows over it in only one direction, from the CPU to the memory or I/O elements. The CPU alone can place logic levels on the lines of the address bus, thereby generating 2^{16} = 65,536 different possible addresses. Each of these addresses corresponds to one memory location or one I/O element.

When the CPU wants to communicate (read or write) with a certain memory location or I/O device, it places the appropriate 16-bit address code on its 16 address pin outputs, A_0 through A_{15}, and onto the address bus. These address bits are then decoded to select the desired memory location or I/O device.

Data Bus

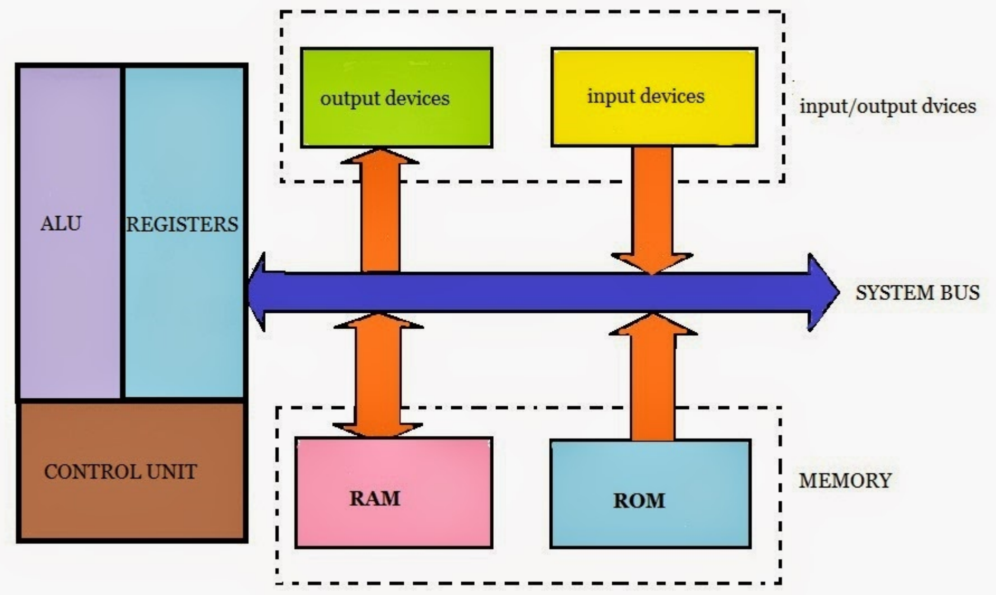
This is a bi-directional bus, because data can flow to or from the CPU. The CPU's eight data pins, D_0 through D_7,can be either inputs or outputs, depending on whether the CPU is performing a read or a write operation. During data bus by the memory or I/O element. During a write operation the CPU's data pins act as outputs and place data on the data bus, which are then sent to the selected memory or I/O element.

Control Bus

This is the set of signals that is used to synchronize the activities of the separate microcomputer elements. Some of these control signals, such as RD and WR are sent by the CPU to the other elements to tell them what type of operation is currently in progress. The I/O elements can send control signals to the CPU. An example is the rest input (RES) of the CPU which, when driven LOW, causes the CPU to reset to a particular starting stare.

Microprocessor

The microprocessor is a semiconductor device consisting of electronic logic circuits manufactured by using either a large-scale (LSI) or very large-scale integration (VLSI) technique. The microprocessor is capable of performing computing functions and making decisions to change the sequence of program execution. The microprocessor can be divided into three segments, arithmetic/logic unit (ALU), register unit, and control unit.



Block Diagram of Microprocessor

Arithmetic and Logic Unit : In this area of the microprocessor, computing functions are performed on data. The CPU performs arithmetic operations such as addition and subtraction, and logic operations such as AND, OR, and exclusive OR. Results are stored either in register or in memory or sent to output devices.

Register Unit : This area of the microprocessor consists of various registers. The register are used primarily to store data temporarily during the executing of a program. Some of the registers are accessible to the user through instructions.

Control Unit : The control unit provides the necessary timing and control signals to all the operations in the microcomputer. It controls the flow of data between the microprocessor and peripherals (including memory).

Microprocessor Architecture and its Operations

The MPU performs primarily four operations.

1. Memory Read: Reads data from memory.
2. Memory Write: Writes data into memory.
3. I/O read: Accepts data from input devices.
4. I/O Write: Sends data to output devices.

All these operations are part of the communication process between the MPU and peripheral devices (including memory). To communicate with a peripheral (or a memory location), the Microprocessor needs to perform the following steps:

Step 1 : Identify the peripheral or the memory location (with its address).
Step 2 : Transfer data
Step 3 : Provide timing or synchronization signals.

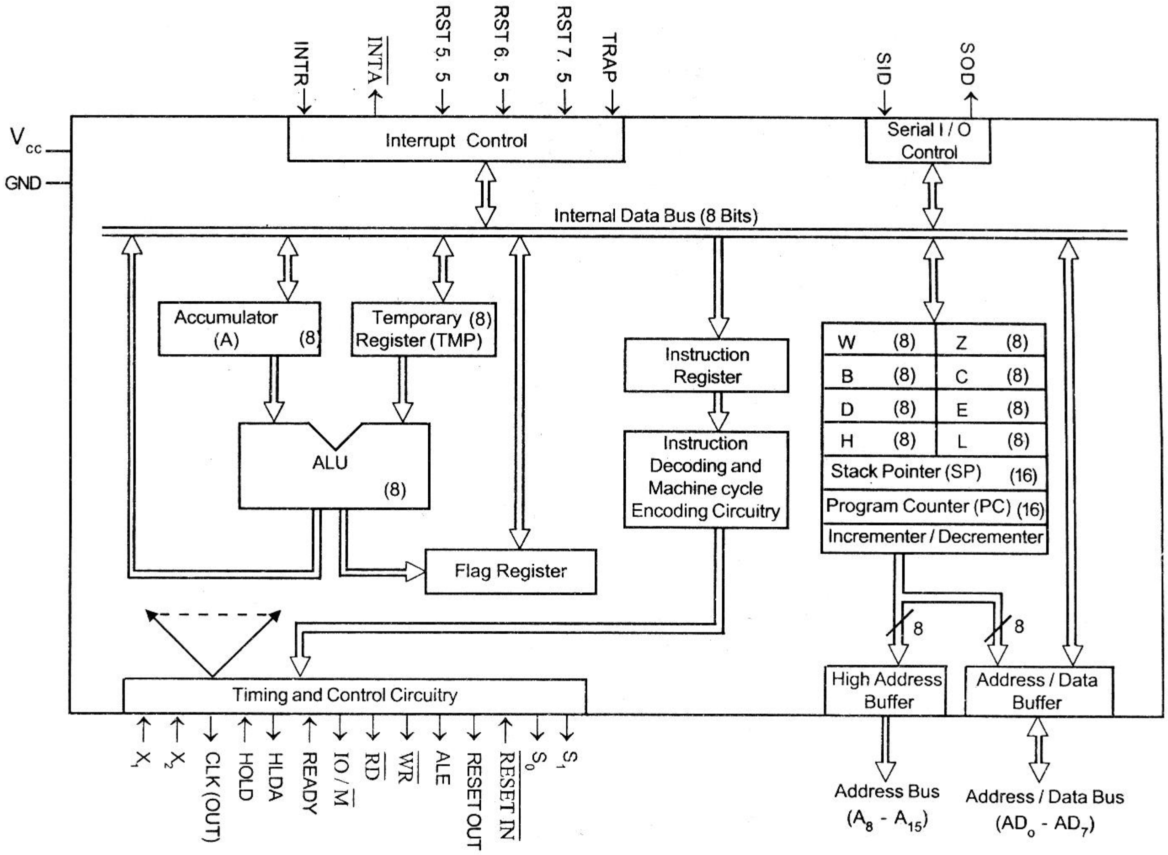
The internal operations can be performed by the microprocessor are

1. Store 8-bits data.
2. Perform arithmetic and logical operations.
3. Test for conditions.
4. Sequence the execution of instructions.
5. Store data temporarily during execution in the defined R/W memory locations called the stack.

To perform these operations, the microprocessor requires registers, an arithmetic logic unit (ALU) and control logic, and internal buses (path for information flow).

INTEL 8085 ARCHITECTURE

The architecture of.8085 is shown in figure given below. The internal architecture of 8085 includes the ALU, timing and control unit, instruction register and decoder, register array, interrupt control and serial I/O control.



Operations performed by 8085

The ALU performs the arithmetic and logical operations.

The operations performed by ALU of 8085 are addition, subtraction, increment, decrement, logical AND, OR, EXCLUIVE -OR, compare, complement and left / right shift. The accumulator and temporary register are used to hold the data during an arithmetic / logical operation. After an operation the result is stored in the accumulator and the flags are set or reset according to the result of the operation.

Registers

The 8085 has six general - purpose registers to perform the first operation listed above, that is, to store 8-bit data during a program execution. These registers are identified as B, C, E, H, and L. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operation.

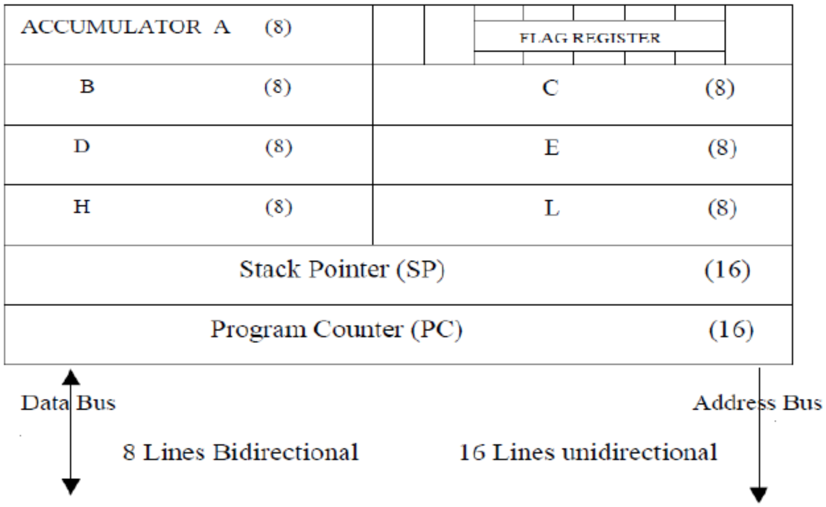


Fig. 3 Register organisation

These registers are programmable, meaning that a programmer can use them to load or transfer data from the registers by using instructions.

Accumulator

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. The accumulator is also identified as register A.

Flags

The ALU includes five flip-flops that are set or reset according to data conditions in the accumulator and other registers. The microprocessor uses them to perform the third operation; namely testing for data conditions.

The bit position of the flip flop in flag register is:

D7	D6	D5	D4	D3	D2	D1	D0
S	Z		AC		P		CY

All of the three flip flop set and reset according to the stored result in the accumulator.

For example, after an addition of two numbers, if the result in the accumulator is larger than 8-bit, the flip-flop uses to indicate a carry by setting CY flag to 1. When an arithmetic operation results in zero, Z flag is set to 1. The S flag is just a copy of the bit D7 of the accumulator. A negative number has a 1 in bit D7 and a positive number has a 0 in 2's complement representation. The AC flag is set to 1, when a carry result from bit D3 and passes to bit D4. The P flag is set to 1, when the result in accumulator contains even number of 1s.

Stack Pointer (SP)

The stack pointer SP, holds the address of the stack top. The stack is a sequence of RAM memory locations defined by the programmer. The stack is used to save the content of registers during the execution of a program.

Program Counter (PC):

The program counter (PC) keeps track of program execution. To execute a program the starting address of the program is loaded in program counter. The PC sends out an address to fetch a byte of instruction from memory and increment its content automatically. Hence, when a byte of instruction is fetched, the PC holds the address of the next byte of the instruction or next instruction.

Timing and Control Unit

It provides timing and control signal to the microprocessor to perform the various operation. It has three control signal. It controls all external and internal circuits. It operates with reference to clock signal. It synchronizes all the data transfers.

There are three control signal:

1.ALE-Address Latch Enable, It provides control signal to synchronize the components of microprocessor.

2.RD- This is active low used for reading operation.

3.WR-This is active low used for writing operation.

There are three status signal used in microprocessor S0, S1 and IO/M. It changes its status according the provided input to these pins.

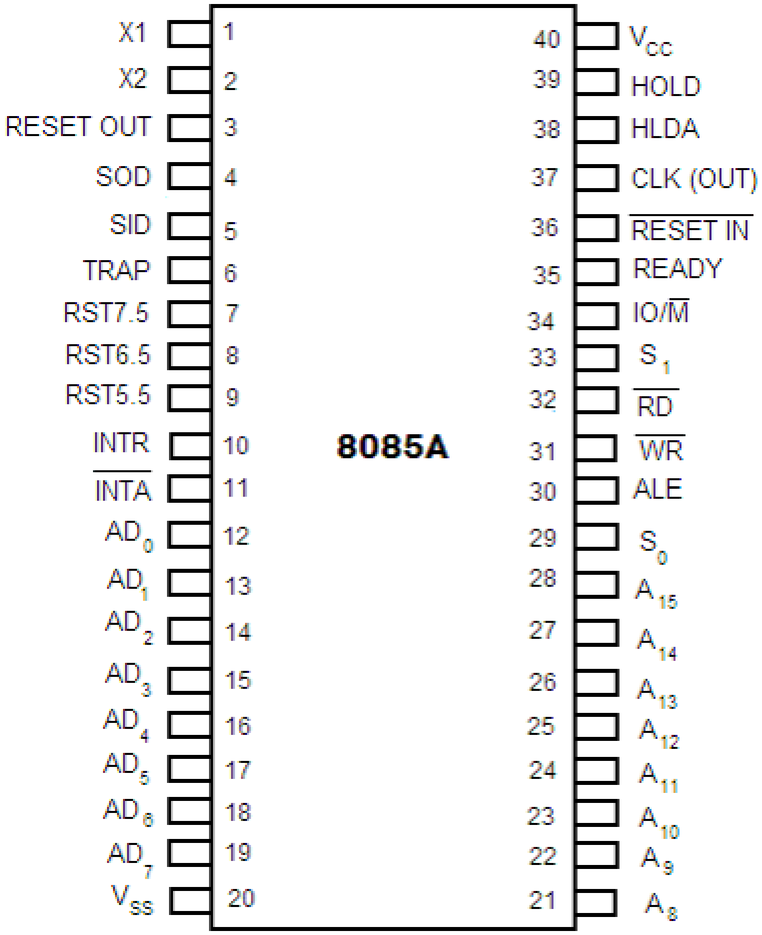
Serial Input Output Control

There are two pins in this unit. This unit is used for serial data communication.

Interrupt Unit

There are 6 interrupt pins in this unit. Generally an external hardware is connected to these pins. These pins provide interrupt signal sent by external hardware to microprocessor and microprocessor sends acknowledgement for receiving the interrupt signal. Generally INTA is used for acknowledgement.

Pin Diagram and Pin description of 8085



The 8085 microprocessor is available on a 40-pin Dual-in-Line package (DIP). The following describes the function of each pin:

A8 – A15 (Output 3 State): Address Bus

The most significant 8 bits of the memory address or the 8 bits of the I/0 addresses, 3 stated during Hold and Halt modes.

AD0 - AD7 (Input/ Output 3 state): Multiplexed Address/Data Bus

Lower 8 bits of the memory address (or I/0 address) appear on the bus during the first clock cycle of a machine state. It then becomes the data bus during the second and third clock cycles. 3 stated during Hold and Halt modes.

ALE (Output): Address Latch Enable

It occurs during the first clock cycle of a machine state and enables the address to get latched into the on chip latch of peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information. ALE can also be used to strobe the status information.

SO, S1 (Output): Data Bus Status

Encoded status of the bus cycle:

S1	S0	status
0	0	HALT
0	1	WRITE
1	0	READ
1	1	FETCH

RD (Output 3state): READ

Indicates the selected memory or 1/0 device is to be read and that the Data Bus is available or the data transfer.

WR (Output 3state): WRITE

Indicates the data on the Data Bus is to be written into the selected memory or 1/0 location.

READY (Input):

If Ready is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If Ready is low, the CPU will wait for Ready to go high before completing the read or write cycle.

HOLD (Input):

Indicates that another Master is requesting the use of the Address and Data Buses. The CPU, upon receiving the Hold request. will relinquish the use of buses as soon as the completion of the current machine cycle. Internal processing can continue. The processor can regain the buses only after the Hold is removed. When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines are 3stated.

HLDA (Output): HOLD ACKNOWLEDGE

Indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed. The CPU takes the buses one half clock cycle after HLDA goes low.

INTR (Input): INTERRUPT REQUEST

It is used as a general purpose interrupt. It is sampled only during the next to the last clock cycle of the instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.

INTA (Output): INTERRUPT ACKNOWLEDGE

It is used instead of (and has the same timing as) RD during the Instruction cycle after an INTR is accepted. It can be used to activate the 8259 Interrupt chip or some other interrupt port.

RST 5.5, RST 6.5, RST 7.5 (Inputs): RESTART INTERRUPTS

These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted.

RST 7.5 --------- Highest Priority

RST 6.5

RST 5.5--------Lowest Priority

The priority of these interrupts is ordered as shown above. These interrupts have a higher priority than the INTR.

TRAP (Input):

Trap interrupt is a non-maskable restart interrupt. It is recognized at the same time as INTR. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt.

RESET IN (Input):

Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flip-flops. None of the other flags or registers (except the instruction register) are affected The CPU is held in the reset condition as long as Reset is applied.

RESET OUT (Output):

Indicates CPU is being reset. Can be used as a system RESET. The signal is synchronized to the processor clock.

X1, X2 (Input):

Crystal or RC network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 to give the internal operating frequency.

CLK (Output):

Clock Output for use as a system clock when a crystal or R/ C network is used as an input to the CPU. The period of CLK is twice the X1, X2 input period.

IO/M (Output):

IO/M indicates whether the Read/Write is to memory or l/O Tristated during Hold

and Halt modes.

SID (Input): Serial input data line

The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

SOD (output): Serial output data line

The output SOD is set or reset as specified by the SIM instruction.

Vcc:

+5 volts supply.

Vss:

Ground Reference.

Microcontroller: A highly integrated chip that contains all the components comprising a controller.

- Typically this includes a CPU, RAM, some form of ROM, I/O ports, and timers.
- Unlike a general-purpose computer, which also includes all of these components, a microcontroller is designed for a very specific task - to control a particular system.

• A microcontroller differs from a microprocessor, which is a general-purpose chip that is used to create a multi-function computer or device and requires multiple chips to handle various tasks.

• A microcontroller is meant to be more self-contained and independent, and functions as a tiny, dedicated computer.

• The great advantage of microcontrollers, as opposed to using larger microprocessors, is that the parts-count and design costs of the item being controlled can be kept to a minimum.

• They are typically designed using CMOS (complementary metal oxide semiconductor) technology, an efficient fabrication technique that uses less power and is more immune to power spikes than other techniques.

• Microcontrollers are sometimes called *embedded microcontrollers,* which just means that they are part of an embedded system that is, one part of a larger device or system.

Microprocessor	**Microcontroller**
Microprocessor is heart of Computer system.	Micro Controller is a heart of embedded system
General-Purpose	Single-Purpose (control-oriented)
It is just a processor. Memory and I/O components have to be connected externally	Micro controller has external processor along with internal memory and i/O components
Since memory and I/O has to be connected externally, the circuit becomes large.	Since memory and I/O are present internally, the circuit is small.
Cost of the entire system increases	Cost of the entire system is low
High Processing Power	Low Processing Power
Due to external components, the entire power consumption is high. Hence it is not suitable to used with devices running on stored power like batteries.	Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries.
Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower.	Since components are internal, most of the operations are internal instruction, hence speed is fast.
Microprocessor have less number of registers, hence more operations are memory based.	Micro controller have more number of registers, hence the programs are easier to write.
Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module	Micro controllers are based on Harvard architecture where program memory and Data memory are separate
Mainly used in personal computers	Used mainly in washing machine, MP3 players

The 8051 Microcontroller is one of the basic type of microcontroller, designed by Intel in 1980's. This microcontroller was based on Harvard Architecture and developed primarily for use in embedded systems technology. Normally, this microcontroller was developed using NMOS technology, which requires more power to operate. Therefore, Intel redesigned Microcontroller 8051 using CMOS technology and their updated versions came with a letter C in their name, for instance an 80C51 it is an 8 bit microcontroller. These latest Microcontrollers requires less power to operate as compared to their previous versions. The 8051 Microcontroller has two buses and two memory spaces of 64K X 8 size for program and data units. It has an 8 bit processing unit and 8 bit accumulator units.

Some of the features that have made the 8051 popular are:

- 4 KB on chip program memory.
- 128 bytes on chip data memory(RAM)
    - 32 bytes devoted to register banks
    - 16 bytes of bit-addressable memory
    - 80 bytes of general-purpose memory
- 4 register banks.
- 8-bit data bus
- 16-bit address bus
- 16 bit timers (usually 2, but may have more, or less).
- 3 internal and 2 external interrupts.
- Bit as well as byte addressable RAM area of 16 bytes.
- Four 8-bit ports, (short models have two 8-bit ports).
- 16-bit program counter and data pointer.
- 1 Microsecond instruction cycle with 12 MHz Crystal.

Applications of 8051 Microcontroller

Some of the applications of 8051 is mainly used in daily life & industrial applications also some of that applications are shown below

- Light sensing and controlling devices
- Temperature sensing and controlling devices
- Fire detections and safety devices
- Automobile applications
- Defense applications

Some industrial applications of micro controller and its applications

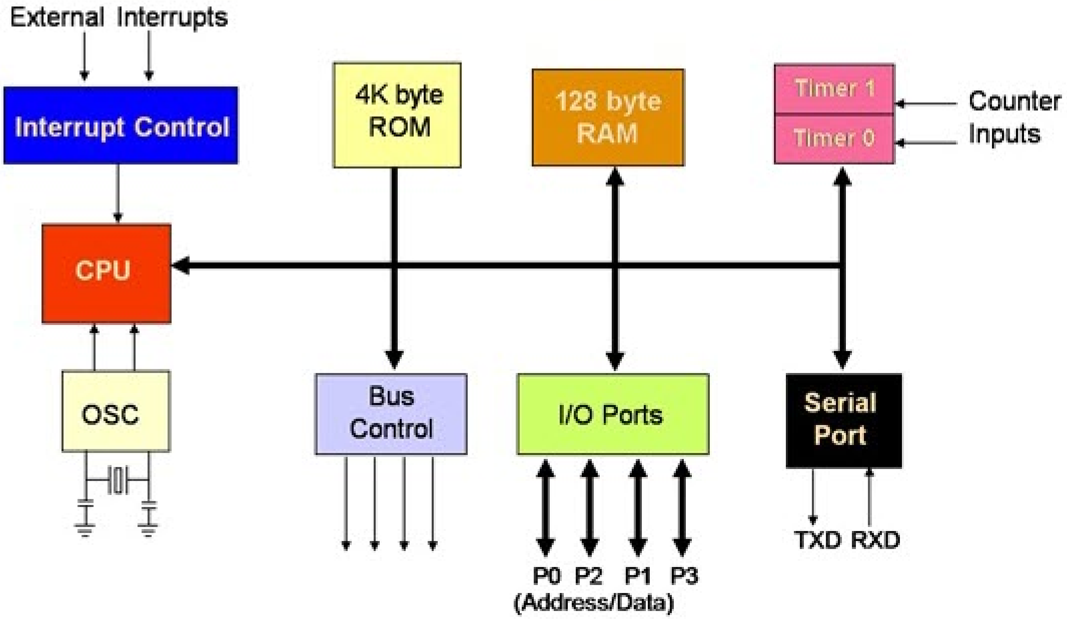
- Industrial instrumentation devices
- Process control devices

Some of 8051 microcontroller devices are used in measurement applications

- Voltmeter applications
- Measuring and revolving objects
- Current meter objects
- Hand held metering system

Block Diagram of 8051 Microcontroller

Following diagram is 8051 Microcontroller architecture . Let us have a look at each part or block of this Architecture of microcontroller.



Block Diagram of 8051 Microcontroller Architecture

Central Processor Unit (CPU)

As we know that the CPU is the brain of any processing device of the microcontroller. It monitors and controls all operations that are performed on the Microcontroller units. The User has no control over the work of the CPU directly . It reads program written in ROM memory and executes them and do the expected task of that application.

Interrupts

As its name suggests, Interrupt is a subroutine call that interrupts of the microcontrollers main operations or work and causes it to execute any other program, which is more important at the time of operation. The feature of Interrupt is very useful as it helps in case of emergency operations. An Interrupts gives us a mechanism to put on hold the ongoing operations, execute a subroutine and then again resumes to another type of operations.

The Microcontroller 8051 can be configured in such a way that it temporarily terminates or pause the main program at the occurrence of interrupts. When a subroutine is completed, Then the execution of main program starts. Generally five interrupt sources are there in 8051 Microcontroller. There are 5 vectored interrupts are shown in below

- INTO
- TFO
- INT1
- TF1
- R1/T1

Out of these, (INT0)̅ and (INT1)̅ are external interrupts that could be negative edge triggered or low level triggered. When All these interrupts are activated, set the corresponding flogs except for serial interrupt,.The interrupt flags are cleared when the processor branches to the interrupt service routine (ISR). The external interrupt flags are cleared when the processor branches to the interrupt service routine, provides the interrupt is a negative edge triggered whereas the timers and serial port interrupts two of them are external interrupts, two of them are timer interrupts and one serial port interrupt terminal in general.

Memory

Microcontroller requires a program which is a collection of instructions. This program tells microcontroller to do specific tasks. These programs require a memory on which these can be saved and read by Microcontroller to perform specific operations of a particular task. The memory which is used to store the program of the microcontroller is known as code memory or Program memory of applications. It is known as ROM memory of microcontroller also requires a memory to store data or operands temporarily of the micro controller. The data memory of the 8051 is used to store data temporarily for operation is known RAM memory. 8051 microcontroller has 4K of code memory or program memory,that has 4KB ROM and also 128 bytes of data memory of RAM.

BUS

Basically Bus is a collection of wires which work as a communication channel or medium for transfer of Data. These buses consists of 8, 16 or more wires of the microcontroller. Thus, these can carry 8 bits,16 bits simultaneously. Hire two types of buses that are shown in below

- Address Bus
- Data Bus

Address Bus: Microcontroller 8051 has a 16 bit address bus for transferring the data. It is used to address memory locations and to transfer the address from CPU to Memory of the microcontroller. It has four addressing modes that are

- Immediate addressing modes.
- Bank address (or) Register addressing mode.
- Direct Addressing mode.
- Register indirect addressing mode.

Data Bus: Microcontroller 8051 has 8 bits of the data bus, which is used to carry data of particular applications.

Oscillator

Generally, we know that the microcontroller is a device, therefore it requires clock pulses for its operation of microcontroller applications. For this purpose, microcontroller 8051 has an on-chip oscillator which works as a clock source for Central Processing Unit of the microcontroller. The output pulses of oscillator are stable. Therefore, it enables synchronized work of all parts of the 8051 Microcontroller.

Input/Output Port

Normally microcontroller is used in embedded systems to control the operation of machines in the microcontroller. Therefore, to connect it to other machines, devices or peripherals we require I/O interfacing ports in the microcontroller interface. For this purpose microcontroller 8051 has 4 input, output ports to connect it to the other peripherals

Program Counter

16bit PC contains the address of the next instruction to be executed. On reset PC will set 0000h. After fetching every instruction PC will increment by one

Stack Pointer

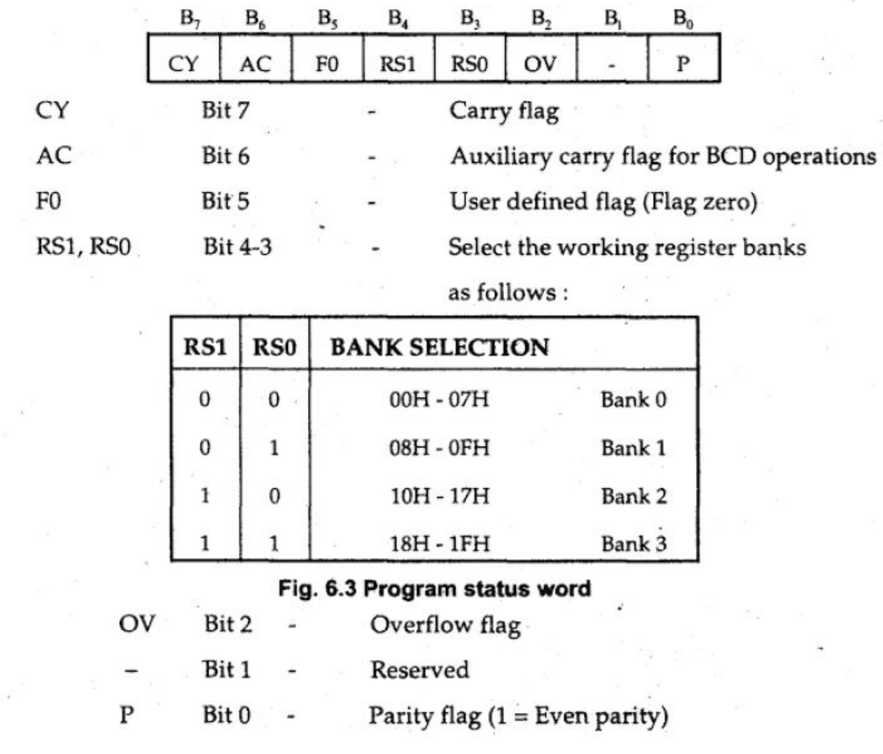
It Contains the address of the data item on the top of the stack. Stack may reside anywhere on the internal RAM. On reset, SP is initialized to 07h so that default stack will start from the address 08h onwards

Data Pointer (DPTR)

DPH (Data Pointer higher byte), DPL (Data Pointer lower byte). This is a 16 bit register which is used to furnish address information for internal and external program memory and for external data memory

Program Status Word

Many instructions implicitly or explicitly affect (or are affected by) several status flags, which are grouped together to form the Program Status Word. It also used to select the memory bank



B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
CY	AC	F0	RS1	RS0	OV	-	P

CY	Bit 7	-	Carry flag
AC	Bit 6	-	Auxiliary carry flag for BCD operations
F0	Bit 5	-	User defined flag (Flag zero)
RS1, RS0	Bit 4-3	-	Select the working register banks as follows :

RS1	RS0	BANK SELECTION	
0	0	00H - 07H	Bank 0
0	1	08H - 0FH	Bank 1
1	0	10H - 17H	Bank 2
1	1	18H - 1FH	Bank 3

Fig. 6.3 Program status word

OV	Bit 2	-	Overflow flag
–	Bit 1	-	Reserved
P	Bit 0	-	Parity flag (1 = Even parity)

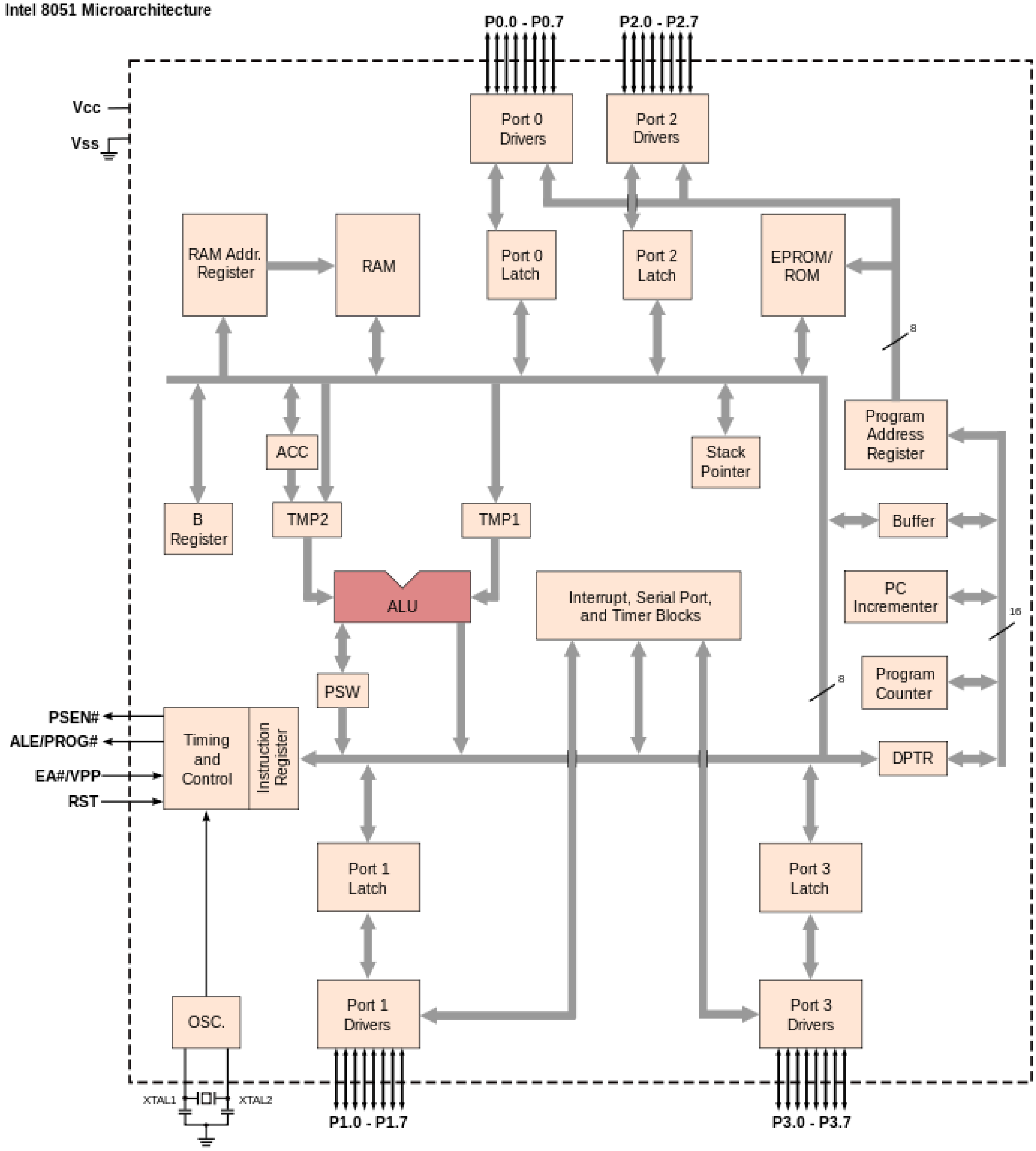
Instruction Register (IR) & timing and control unit:

The 8051 has 8-bit ALU, which performs arithmetic and logical operations on binary data. The A and B registers are used to hold the input data and the result of ALU operation. The controller will fetch the instructions one by one, starting from the address stored in PC and store in IR, which decodes the instructions and give information to timing and control unit. Using the information supplied by the IR unit the control signals necessary for internal and external operations are generated by the timing and control unit.

Timer/Counter & Serial port

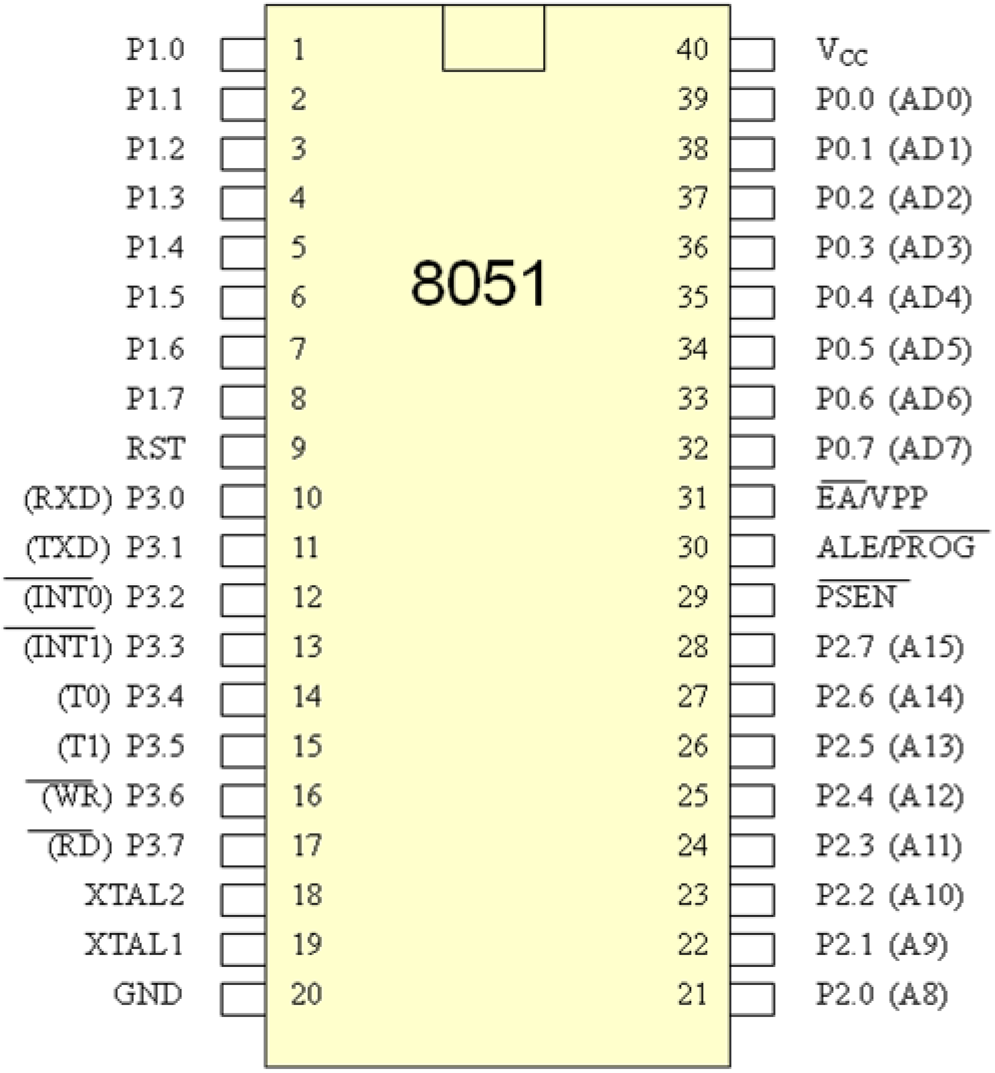
The 8031/8051 has two 16-bit programmable timer/counter namely timer-1 and timer 0. In the counter mode of operation they can count the number of high to low transitions of the signal applied to the timer pins. In timer mode of operation they can be independently programmed to work in any one of the four operation modes. They are called mode-0, mode-1, mode-2 and mode-3. In mode-0 the serial port can either receive or transmit at fixed baud rate. In mode-2 it can simultaneously transmit and receive at any one of the two selectable baud rate. In mode-1 and mode-3 it can work as full duplex serial port with variable baud rate, which is programmed using timer-1.

Intel 8051 Microarchitecture



- 8051 has 4 K Bytes of internal ROM. The address space is from 0000 to 0FFFh. If the program size is more than 4 K Bytes 8051 will fetch the code automatically from external memory.

- Accumulator is an 8 bit register widely used for all arithmetic and logical operations. Accumulator is also used to transfer data between external memory. B register is used along with Accumulator for multiplication and division. A and B registers together is also called MATH registers.



Pinout Description ***Pins 1-8***	**PORT 1**. Each of these pins can be configured as an input or an output.
Pin 9	**RESET**. A logic one on this pin disables the microcontroller and clears the contents of most registers. In other words, the positive voltage on this pin resets the microcontroller. By applying logic zero to this pin, the program starts execution from the beginning.
Pins10-17	**PORT 3**. Similar to port 1, each of these pins can serve as general input or output. Besides, all of them have alternative functions

Pin 10	RXD. Serial asynchronous communication input or Serial synchronous communication input.
Pin 11	TXD. Serial asynchronous communication output or Serial synchronous communication clock output.
Pin 12	INT0.External Interrupt 0 input
Pin 13	INT1. External Interrupt 1 input
Pin 14	T0. Counter 0 clock input
Pin 15	T1. Counter 1 clock input
Pin 16	WR. Write to external (additional) RAM
Pin 17	RD. Read from external RAM
Pin 18, 19	XTAL2, XTAL1. Internal oscillator input and output. A quartz crystal which specifies operating frequency is usually connected to these pins.
Pin 20	GND. Ground.
Pin 21-28	Port 2. If there is no intention to use external memory then these port pins are configured as general inputs/outputs. In case external memory is used, the higher address byte, i.e. addresses A8-A15 will appear on this port. Even though memory with capacity of 64Kb is not used, which means that not all eight port bits are used for its addressing, the rest of them are not available as inputs/outputs.
Pin 29	PSEN. If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.
Pin 30	ALE. Prior to reading from external memory, the microcontroller puts the lower address byte (A0-A7) on P0 and activates the ALE output. After receiving signal from the ALE pin, the external latch latches the state of P0 and uses it as a memory chip address. Immediately after that, the ALE pin is returned its previous logic state and P0 is now used as a Data Bus.
Pin 31	EA. By applying logic zero to this pin, P2 and P3 are used for data and address transmission with no regard to whether there is internal memory or not. It means that even there is a program written to the microcontroller, it will not be executed. Instead, the program written to external ROM will be executed. By applying logic one to the EA pin, the microcontroller will use both memories, first internal then external (if exists).
Pin 32-39	PORT 0. Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs. Otherwise, P0 is configured as address output (A0-A7) when the ALE pin is driven high (1) or as data output (Data Bus) when the ALE pin is driven low (0).
Pin 40	VCC. +5V power supply.

MEMORY AND REGISTER ORGANISATION

The 8051 has a separate memory space for code (programs) and data. We will refer here to on-chip memory and external memory as shown in figure 1.5. In an actual implementation the external memory may, in fact, be contained within the microcomputer chip. However, we will use the definitions of internal and external memory to be consistent with 8051 instructions which operate on memory. Note, the separation of the code and data memory in the 8051 architecture is a little unusual. The separated memory architecture is referred to as *Harvard* architecture whereas *Von Neumann* architecture defines a system where code and data can share common memory.

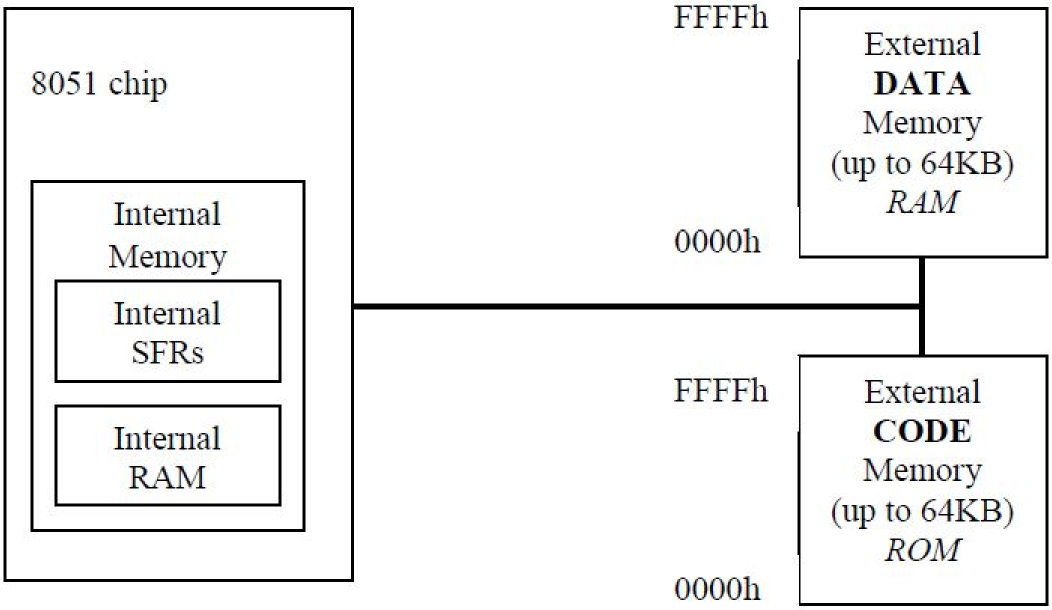

8051 chip
Internal Memory
Internal SFRs
Internal RAM
FFFFh
0000h
External **DATA** Memory (up to 64KB) *RAM*
FFFFh
0000h
External **CODE** Memory (up to 64KB) *ROM*

Figure 1.5 8051 Memory representation

External Code Memory

The executable program code is stored in this code memory. The code memory size is limited to 64KBytes (in a standard 8051). The code memory is *read-only* in normal operation and is programmed under special conditions e.g. it is a PROM or a Flash RAM type of memory.

External RAM Data Memory

This *is read-write* memory and is available for storage of data. Up to 64KBytes of external RAM data memory is supported (in a standard 8051).

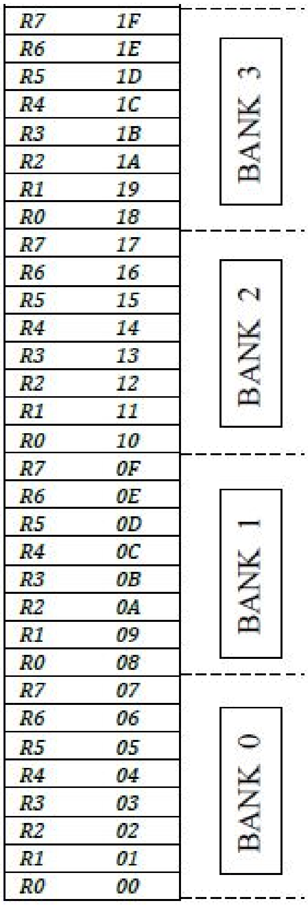
Internal Memory

The 8051's on-chip memory consists of 256 memory bytes organized as follows:

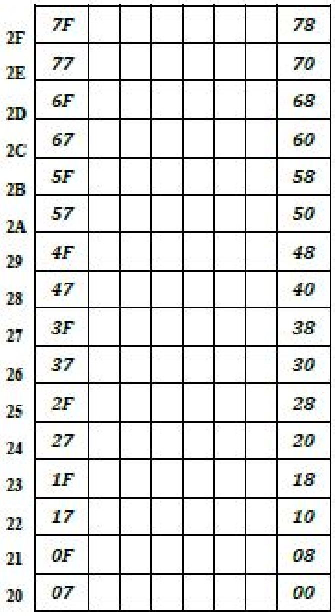
First 128 bytes:	00h to 1Fh Register Banks
	20h to 2Fh Bit Addressable RAM
	30 to 7Fh General Purpose RAM
Next 128 bytes:	80h to FFh Special Function Registers

The first 128 bytes of internal memory is organized as shown in figure 1.6, and is referred to as Internal RAM, or IRAM.

Internal RAM organization

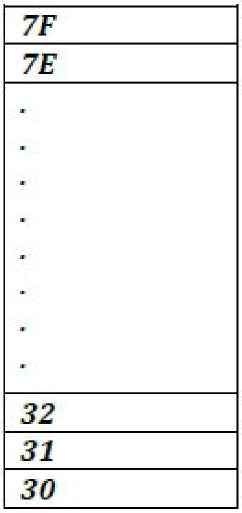


R7	1F	BANK 3
R6	1E	
R5	1D	
R4	1C	
R3	1B	
R2	1A	
R1	19	
R0	18	
R7	17	BANK 2
R6	16	
R5	15	
R4	14	
R3	13	
R2	12	
R1	11	
R0	10	
R7	0F	BANK 1
R6	0E	
R5	0D	
R4	0C	
R3	0B	
R2	0A	
R1	09	
R0	08	
R7	07	BANK 0
R6	06	
R5	05	
R4	04	
R3	03	
R2	02	
R1	01	
R0	00	



2F	7F							78
2E	77							70
2D	6F							68
2C	67							60
2B	5F							58
2A	57							50
29	4F							48
28	47							40
27	3F							38
26	37							30
25	2F							28
24	27							20
23	1F							18
22	17							10
21	0F							08
20	07							00

Bit addressable memory



7F
7E
. . . .
32
31
30

General purpose memory

Working With Registers

Register Banks: 00h to 1Fh. The 8051 uses 8 general-purpose registers R0 through R7 (R0, R1, R2, R3, R4, R5, R6, and R7). There are four such register banks. Selection of register bank can be done through RS1,RS0 bits of PSW. On reset, the default Register Bank 0 will be selected.

Bit Addressable RAM: 20h to 2Fh . The 8051 supports a special feature which allows access to bit variables. This is where individual memory bits in Internal RAM can be set or cleared. In all there are 128 bits numbered 00h to 7Fh. Being bit variables any one variable can have a value 0 or 1. A bit variable can be set with a command such as SETB and cleared with a command such as CLR.
Example instructions are:
SETB 25h ; sets the bit 25h (becomes 1)
CLR 25h ; clears bit 25h (becomes 0)
Note, bit 25h is actually bit 5 of Internal RAM location 24h.
The Bit Addressable area of the RAM is just 16 bytes of Internal RAM located between 20h and 2Fh.

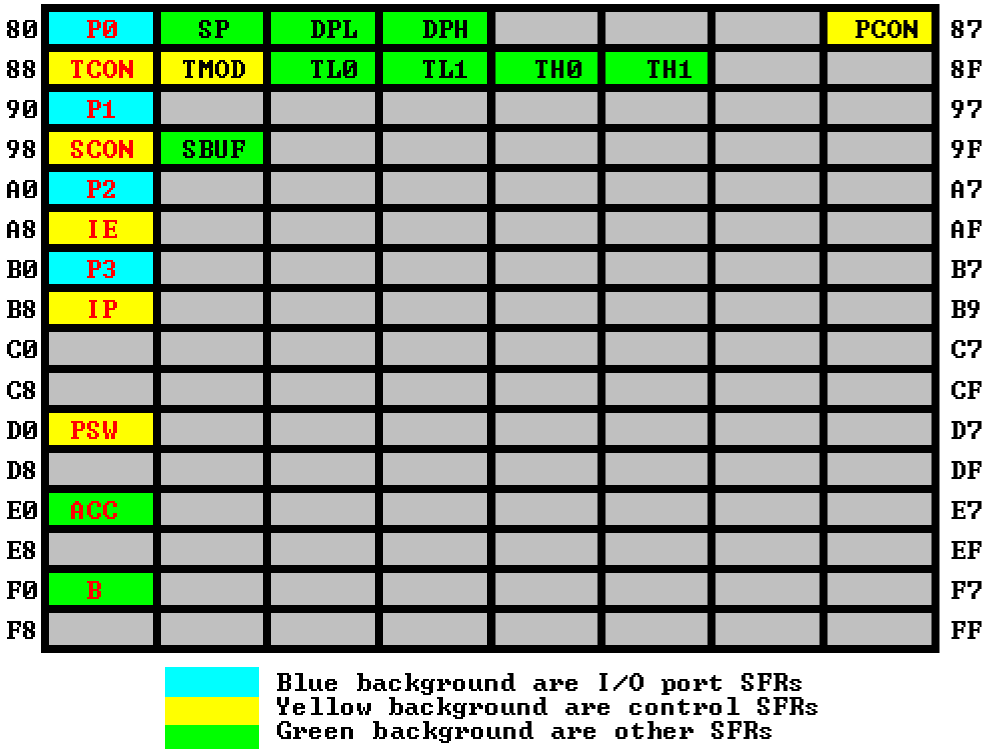
General Purpose RAM: 30h to 7Fh. Even if 80 bytes of Internal RAM memory are available for general-purpose data storage, user should take care while using the memory location from 00 -2Fh since these locations are also the default register space, stack space, and bit addressable space. It is a good practice to use general purpose memory from 30 – 7Fh. The general purpose RAM can be accessed using direct or indirect addressing modes.

What Are SFRs?

The 8051 is a flexible microcontroller with a relatively large number of modes of operations. Your program may inspect and/or change the operating mode of the 8051 by manipulating the values of the 8051's Special Function Registers (SFRs).

SFRs are accessed as if they were normal Internal RAM. The only difference is that Internal RAM is from address 00h through 7Fh whereas SFR registers exist in the address range of 80h through FFh.

Each SFR has an address (80h through FFh) and a name. The following chart provides a graphical presentation of the 8051's SFRs, their names, and their address.



80	P0	SP	DPL	DPH				PCON	87
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
90	P1								97
98	SCON	SBUF							9F
A0	P2								A7
A8	IE								AF
B0	P3								B7
B8	IP								B9
C0									C7
C8									CF
D0	PSW								D7
D8									DF
E0	ACC								E7
E8									EF
F0	B								F7
F8									FF

🟦 Blue background are I/O port SFRs
🟨 Yellow background are control SFRs
🟩 Green background are other SFRs

As you can see, although the address range of 80h through FFh offer 128 possible addresses, there are only 21 SFRs in a standard 8051. All other addresses in the SFR range (80h through FFh) are considered invalid. Writing to or reading from these registers may produce undefined values or behavior.

SFR Types

As mentioned in the chart itself, the SFRs that have a blue background are SFRs related to the I/O ports. The 8051 has four I/O ports of 8 bits, for a total of 32 I/O lines. Whether a given I/O line is high or low and the value read from the line are controlled by the SFRs in green.

The SFRs with yellow backgrounds are SFRs which in some way control the operation or the configuration of some aspect of the 8051. For example, **TCON** controls the timers, SCON controls the serial port.

The remaining SFRs, with green backgrounds, are "other SFRs." These SFRs can be thought of as auxillary SFRs in the sense that they don't directly configure the 8051 but obviously the 8051 cannot operate without them. For example, once the serial port has been configured using **SCON**, the program may read or write to the serial port using the **SBUF** register.

SFR Descriptions

This section will endeavor to quickly overview each of the standard SFRs found in the above SFR chart map. It is not the intention of this section to fully explain the functionality of each SFR--this information will be covered in separate chapters of the tutorial. This section is to just give you a general idea of what each SFR does.

P0 (Port 0, Address 80h, Bit-Addressable): This is input/output port 0. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 0 is pin P0.0, bit 7 is pin P0.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

SP (Stack Pointer, Address 81h): This is the stack pointer of the microcontroller. This SFR indicates where the next value to be taken from the stack will be read from in Internal RAM. If you push a value onto the stack, the value will be written to the address of SP + 1. That is to say, if SP holds the value 07h, a PUSH instruction will push the value onto the stack at address 08h. This SFR is modified by all instructions which modify the stack, such as PUSH, POP, LCALL, RET, RETI, and whenever interrupts are provoked by the microcontroller.

DPL/DPH (Data Pointer Low/High, Addresses 82h/83h): The SFRs DPL and DPH work together to represent a 16-bit value called the *Data Pointer*. The data pointer is used in operations regarding external RAM and some instructions involving code memory. Since it is an unsigned two-byte integer value, it can represent values from 0000h to FFFFh (0 through 65,535 decimal).

PCON (Power Control, Addresses 87h): The Power Control SFR is used to control the 8051's power control modes. Certain operation modes of the 8051 allow the 8051 to go into a type of "sleep" mode which requires much less power. These modes of operation are controlled through PCON. Additionally, one of the bits in PCON is used to double the effective baud rate of the 8051's serial port.

TCON (Timer Control, Addresses 88h, Bit-Addressable): The Timer Control SFR is used to configure and modify the way in which the 8051's two timers operate. This SFR controls whether each of the two timers is running or stopped and contains a flag to indicate that each timer has overflowed. Additionally, some non-timer related bits are located in the TCON SFR. These bits are used to configure the way in which the external interrupts are activated and also contain the external interrupt flags which are set when an external interrupt has occured.

TMOD (Timer Mode, Addresses 89h): The Timer Mode SFR is used to configure the mode of operation of each of the two timers. Using this SFR your program may configure each timer to be a 16-bit timer, an 8-bit auto reload timer, a 13-bit timer, or two separate timers. Additionally, you may configure the timers to only count when an external pin is activated or to count "events" that are indicated on an external pin.

TL0/TH0 (Timer 0 Low/High, Addresses 8Ah/8Ch): These two SFRs, taken together, represent timer 0. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

TL1/TH1 (Timer 1 Low/High, Addresses 8Bh/8Dh): These two SFRs, taken together, represent timer 1. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

P1 (Port 1, Address 90h, Bit-Addressable): This is input/output port 1. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 1 is pin P1.0, bit 7 is pin P1.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

SCON (Serial Control, Addresses 98h, Bit-Addressable): The Serial Control SFR is used to configure the behavior of the 8051's on-board serial port. This SFR controls the baud rate of the serial port, whether the serial port is activated to receive data, and also contains flags that are set when a byte is successfully sent or received.

SBUF (Serial Control, Addresses 99h): The Serial Buffer SFR is used to send and receive data via the on-board serial port. Any value written to SBUF will be sent out the serial port's TXD pin. Likewise, any value which the 8051 receives via the serial port's RXD pin will be delivered to the user program via SBUF. In other words, SBUF serves as the output port when written to and as an input port when read from.

P2 (Port 2, Address A0h, Bit-Addressable): This is input/output port 2. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 2 is pin P2.0, bit 7 is pin P2.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

IE (Interrupt Enable, Addresses A8h): The Interrupt Enable SFR is used to enable and disable specific interrupts. The low 7 bits of the SFR are used to enable/disable the specific interrupts, where as the highest bit is used to enable or disable ALL interrupts. Thus, if the high bit of IE is 0 all interrupts are disabled regardless of whether an individual interrupt is enabled by setting a lower bit.

P3 (Port 3, Address B0h, Bit-Addressable): This is input/output port 3. Each bit of this SFR corresponds to one of the pins on the microcontroller. For example, bit 0 of port 3 is pin P3.0, bit 7 is pin P3.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to a low level.

IP (Interrupt Priority, Addresses B8h, Bit-Addressable): The Interrupt Priority SFR is used to specify the relative priority of each interrupt. On the 8051, an interrupt may either be of low (0) priority or high (1) priority. An interrupt may only interrupt interrupts of lower priority. For example, if we configure the 8051 so that all interrupts are of low priority except the serial interrupt, the serial interrupt will always be able to interrupt the system, even if another interrupt is currently executing. However, if a serial interrupt is executing no other interrupt will be able to interrupt the serial interrupt routine since the serial interrupt routine has the highest priority.

PSW (Program Status Word, Addresses D0h, Bit-Addressable): The Program Status Word is used to store a number of important bits that are set and cleared by 8051 instructions. The PSW SFR contains the carry flag, the auxiliary carry flag, the overflow flag, and the parity flag. Additionally, the PSW register contains the register bank select flags which are used to select which of the "R" register banks are currently selected.

ACC (Accumulator, Addresses E0h, Bit-Addressable): The Accumulator is one of the most-used SFRs on the 8051 since it is involved in so many instructions. The Accumulator resides as an SFR at E0h, which means the instruction **MOV A,#20h** is really the same as **MOV E0h,#20h**. However, it is a good idea to use the first method since it only requires two bytes whereas the second option requires three bytes.

B (B Register, Addresses F0h, Bit-Addressable): The "B" register is used in two instructions: the multiply and divide operations. The B register is also commonly used by programmers as an auxiliary register to temporarily store values.

ADDRESSING MODES

8051 addressing modes are classified as follows.

1. Immediate addressing.
2. Register addressing.
3. Direct addressing.
4. Indirect addressing.
5. Relative addressing.
6. Absolute addressing.
7. Long addressing.
8. Indexed addressing.
9. Bit inherent addressing.
10. Bit direct addressing.

1. Immediate addressing.

In this addressing mode the data is provided as a part of instruction itself. In other words data immediately follows the instruction.

Eg. MOV A,#30H
ADD A, #83

2. Register addressing.

In this addressing mode the register will hold the data. One of the eight general registers (R0 to R7) can be used and specified as the operand.

```
Eg.     MOV A,R0
        ADD A,R6
```

R0 – R7 will be selected from the current selection of register bank. The default register bank will be bank 0.

3. Direct addressing

There are two ways to access the internal memory. Using direct address and indirect address. Using direct addressing mode we can not only address the internal memory but SFRs also. In direct addressing, an 8 bit internal data memory address is specified as part of the instruction and hence, it can specify the address only in the range of 00H to FFH. In this addressing mode, data is obtained directly from the memory.

```
Eg.     MOV A,60h
        ADD A,30h
```

4. Indirect addressing

The indirect addressing mode uses a register to hold the actual address that will be used in data movement. Registers R0 and R1 and DPTR are the only registers that can be used as data pointers. Indirect addressing cannot be used to refer to SFR registers. Both R0 and R1 can hold 8 bit address and DPTR can hold 16 bit address.

```
Eg.     MOV A,@R0
        ADD A,@R1
        MOVX A,@DPTR
```

5. Indexed addressing.

In indexed addressing, either the program counter (PC), or the data pointer (DTPR)—is used to hold the base address, and the A is used to hold the offset address. Adding the value of the base address to the value of the offset address forms the effective address. Indexed addressing is used with JMP or MOVC instructions. Look up tables are easily implemented with the help of index addressing.

Eg. MOVC A, @A+DPTR // *copies the contents of memory location pointed by the sum of the accumulator A and the DPTR into accumulator A.*

MOVC A, @A+PC // *copies the contents of memory location pointed by the sum of the accumulator A and the program counter into accumulator A.*

6. Relative Addressing.

Relative addressing is used only with conditional jump instructions. The relative address, (offset), is an 8 bit signed number, which is automatically added to the PC to make the address of the next instruction. The 8 bit signed offset value gives an address range of +127 to —128 locations. The jump destination is usually specified using a label and the assembler calculates the jump offset accordingly. The advantage of relative addressing is that the program code is easy to relocate and the address is relative to position in the memory.

```
Eg.     SJMP LOOP1
        JC BACK
```

7. Absolute addressing

Absolute addressing is used only by the AJMP (Absolute Jump) and ACALL (Absolute Call) instructions. These are 2 bytes instructions. The absolute addressing mode specifies the lowest 11 bit of the memory address as part of the instruction. The upper 5 bit of the destination address are

the upper 5 bit of the current program counter. Hence, absolute addressing allows branching only within the current 2 Kbyte page of the program memory.

```
Eg.     AJMP LOOP1
        ACALL LOOP2
```

8. Long Addressing

The long addressing mode is used with the instructions LJMP and LCALL. These are 3 byte instructions. The address specifies a full 16 bit destination address so that a jump or a call can be made to a location within a 64 Kbyte code memory space.

```
Eg.     LJMP FINISH
        LCALL DELAY
```

9. Bit Inherent Addressing

In this addressing, the address of the flag which contains the operand, is implied in the opcode of the instruction.
Eg. CLR C *; Clears the carry flag to 0*

10. Bit Direct Addressing

In this addressing mode the direct address of the bit is specified in the instruction. The RAM space 20H to 2FH and most of the special function registers are bit addressable. Bit address values are between 00H to 7FH.
Eg. CLR 07h *; Clears the bit 7 of 20h RAM space*
SETB 07H *; Sets the bit 7 of 20H RAM space.*

8051 Instructions Set

The instructions of 8051 can be broadly classified under the following headings.

1. Data transfer instructions
2. Arithmetic instructions
3. Logical instructions
4. Branch instructions
5. Subroutine instructions
6. Bit manipulation instructions

Data transfer instructions.

In this group, the instructions perform data transfer operations of the following types.

a. Move the contents of a register Rn to A
i. MOV A,R2
ii. MOV A,R7

b. Move the contents of a register A to Rn
i. MOV R4,A
ii. MOV R1,A

c. Move an immediate 8 bit data to register A or to Rn or to a memory location(direct or indirect)
i. MOV A, #45H
ii. MOV R6, #51H
iii. MOV 30H, #44H
iv. MOV @R0, #0E8H
v. MOV DPTR, #0F5A2H
vi. MOV DPTR, #5467H

d. Move the contents of a memory location to A or A to a memory location using direct and indirect addressing
i. MOV A, 65H
ii. MOV A, @R0
iii. MOV 45H, A
iv. MOV @R1, A

e. Move the contents of a memory location to Rn or Rn to a memory location using direct addressing
i. MOV R3, 65H
ii. MOV 45H, R2

f. Move the contents of memory location to another memory location using direct and indirect addressing
i. MOV 47H, 65H
ii. MOV 45H, @R0

g. Move the contents of an external memory to A or A to an external memory *i.* MOVX A,@R1

ii. MOVX @R0,A

iii. MOVX A,@DPTR

iv. MOVX@DPTR,A

h. Move the contents of program memory to A

i. MOVC A, @A+PC

ii. MOVC A, @A+DPTR

i. Push and Pop instructions

```
[SP]=07 //CONTENT OF SP IS 07 (DEFAULT VALUE)
MOV R6, #25H [R6]=25H //CONTENT OF R6 IS 25H
MOV R1, #12H [R1]=12H //CONTENT OF R1 IS 12H
MOV R4, #0F3H [R4]=F3H //CONTENT OF R4 IS F3H
PUSH 6 [SP]=08 [08]=[06]=25H //CONTENT OF 08 IS 25H
PUSH 1 [SP]=09 [09]=[01]=12H //CONTENT OF 09 IS 12H
PUSH 4 [SP]=0A [0A]=[04]=F3H //CONTENT OF 0A IS F3H
POP 6 [06]=[0A]=F3H [SP]=09 //CONTENT OF 06 IS F3H
POP 1 [01]=[09]=12H [SP]=08 //CONTENT OF 01 IS 12H
POP 4 [04]=[08]=25H [SP]=07 //CONTENT OF 04 IS 25H
```

j. Exchange instructions
The content of source ie., register, direct memory or indirect memory will be exchanged with the contents of destination ie., accumulator.

i. XCH A,R3

ii. XCH A,@R1

iii. XCH A,54h

k. Exchange digit. Exchange the lower order nibble of Accumulator (A0-A3) with lower order nibble of the internal RAM location which is indirectly addressed by the register.

i. XCHD A,@R1

ii. XCHD A,@R0

Arithmetic instructions.

The 8051 can perform addition, subtraction. Multiplication and division operations on 8 bit numbers.

Addition

In this group, we have instructions to

i. Add the contents of A with immediate data with or without carry.

i. ADD A, #45H

ii. ADDC A, #OB4H

ii. Add the contents of A with register Rn with or without carry.

i. ADD A, R5

ii. ADDC A, R2

iii. Add the contents of A with contents of memory with or without carry using direct and indirect addressing

i. ADD A, 51H

ii. ADDC A, 75H

iii. ADD A, @R1

iv. ADDC A, @R0

CY AC and OV flags will be affected by this operation.

Subtraction

In this group, we have instructions to

i. Subtract the contents of A with immediate data with or without carry.

i. SUBB A, #45H

ii. SUBB A, #OB4H

ii. Subtract the contents of A with register Rn with or without carry.

i. SUBB A, R5

ii. SUBB A, R2

iii. Subtract the contents of A with contents of memory with or without carry using direct and indirect addressing

i. SUBB A, 51H

ii. SUBB A, 75H

iii. SUBB A, @R1

iv. SUBB A, @R0

CY AC and OV flags will be affected by this operation.

Multiplication

MUL AB. This instruction multiplies two 8 bit unsigned numbers which are stored in A and B register. After multiplication the lower byte of the result will be stored in accumulator and higher byte of result will be stored in B register.

```
Eg.     MOV A,#45H ;[A]=45H
        MOV B,#0F5H ;[B]=F5H
        MUL AB ;[A] x [B] = 45 x F5 = 4209
               ;[A]=09H, [B]=42H
```

Division

DIV AB. This instruction divides the 8 bit unsigned number which is stored in A by the 8 bit unsigned number which is stored in B register. After division the result will be stored in accumulator and remainder will be stored in B register.

```
Eg. MOV A,#45H ;[A]=0E8H
MOV B,#0F5H ;[B]=1BH
DIV AB ;[A] / [B] = E8 /1B = 08 H with remainder 10H
;[A] = 08H, [B]=10H
```

DA A (Decimal Adjust After Addition).

When two BCD numbers are added, the answer is a non-BCD number. To get the result in BCD, we use DA A instruction after the addition. DA A works as follows.

- If lower nibble is greater than 9 or auxiliary carry is 1, 6 is added to lower nibble.
- If upper nibble is greater than 9 or carry is 1, 6 is added to upper nibble.

```
Eg 1:   MOV A,#23H
        MOV R1,#55H
        ADD A,R1 // [A]=78
        DA A // [A]=78 no changes in the accumulator after da a
Eg 2:   MOV A,#53H
        MOV R1,#58H
        ADD A,R1 // [A]=ABh
        DA A // [A]=11, C=1 . ANSWER IS 111. Accumulator data is changed after DA A
```

Increment: *increments the operand by one.*

INC A INC Rn INC DIRECT INC @Ri INC DPTR

INC increments the value of source by 1. If the initial value of register is FFh, incrementing the value will cause it to reset to 0. The Carry Flag is not set when the value "rolls over" from 255 to 0.

In the case of "INC DPTR", the value two-byte unsigned integer value of DPTR is incremented. If the initial value of DPTR is FFFFh, incrementing the value will cause it to reset to 0.

Decrement: *decrements the operand by one.*

DEC A DEC Rn DEC DIRECT DEC @Ri

DEC decrements the value of *source* by 1. If the initial value of is 0, decrementing the value will cause it to reset to FFh. The Carry Flag is not set when the value "rolls over" from 0 to FFh.

Logical Instructions

Logical AND

ANL destination, source: ANL does a bitwise "AND" operation between *source* and *destination*, leaving the resulting value in *destination*. The value in source is not affected. "AND" instruction logically AND the bits of source and destination.

ANL A,#DATA ANL A, Rn
ANL A,DIRECT ANL A,@Ri
ANL DIRECT,A ANL DIRECT, #DATA

Logical OR

ORL destination, source: ORL does a bitwise "OR" operation between *source* and *destination*,leaving the resulting value in *destination*. The value in source is not affected. " OR " instruction logically OR the bits of source and destination.

ORL A,#DATA ORL A, Rn
ORL A,DIRECT ORL A,@Ri
ORL DIRECT,A ORL DIRECT, #DATA

Logical Ex-OR

XRL destination, source: XRL does a bitwise "EX-OR" operation between *source* and *destination*, leaving the resulting value in *destination*. The value in source is not affected. " XRL " instruction logically EX-OR the bits of source and destination.

XRL A,#DATA XRL A,Rn
XRL A,DIRECT XRL A,@Ri
XRL DIRECT,A XRL DIRECT, #DATA

Logical NOT

CPL complements *operand*, leaving the result in *operand*. If *operand* is a single bit then the state of the bit will be reversed. If *operand* is the Accumulator then all the bits in the Accumulator will be reversed.

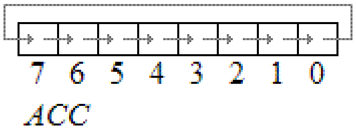
CPL A, CPL C, CPL bit address

SWAP A – Swap the upper nibble and lower nibble of A.

Rotate Instructions

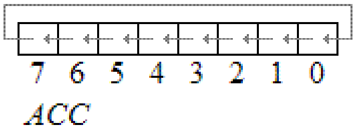
RR A

This instruction is rotate right the accumulator. Its operation is illustrated below. Each bit is shifted one location to the right, with bit 0 going to bit 7.



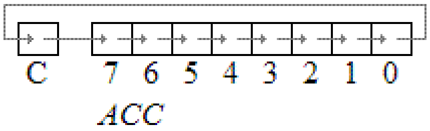
RL A

Rotate left the accumulator. Each bit is shifted one location to the left, with bit 7 going to bit 0



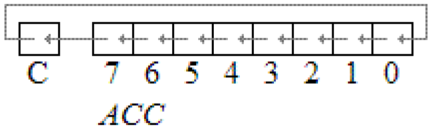
RRC A

Rotate right through the carry. Each bit is shifted one location to the right, with bit 0 going into the carry bit in the PSW, while the carry was at goes into bit 7



RLC A

Rotate left through the carry. Each bit is shifted one location to the left, with bit 7 going into the carry bit in the PSW, while the carry goes into bit 0.



Branch (JUMP) Instructions

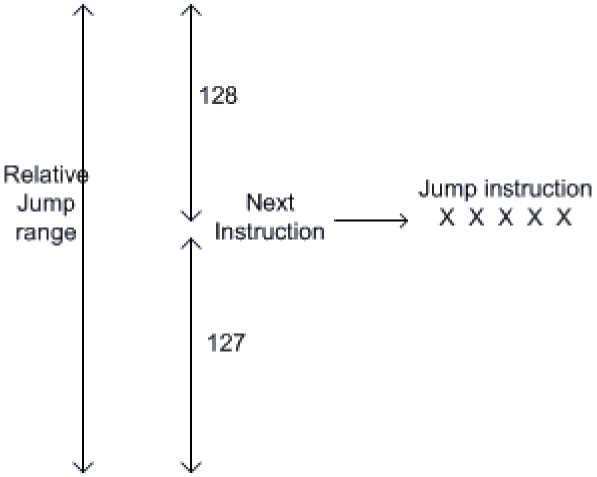
Jump and Call Program Range

There are 3 types of jump instructions. They are:-

1. Relative Jump
2. Short Absolute Jump
3. Long Absolute Jump

Relative Jump

Jump that replaces the PC (program counter) content with a new address that is greater than (the address following the jump instruction by 127 or less) or less than (the address following the jump by 128 or less) is called a relative jump.



The advantages of the relative jump are as follows:-

1. Only 1 byte of jump address needs to be specified in the 2's complement form, ie. For jumping ahead, the range is 0 to 127 and for jumping back, the range is -1 to -128.
2. Specifying only one byte reduces the size of the instruction and speeds up program execution.
3. The program with relative jumps can be relocated without reassembling to generate absolute jump addresses.

Disadvantages of the absolute jump: -

1. Short jump range (-128 to 127 from the instruction following the jump instruction)

Instructions that use Relative Jump

SJMP <relative address>; *this is unconditional jump*

The remaining relative jumps are conditional jumps

JC <relative address>

JNC <relative address>

JB bit, <relative address>
JNB bit, <relative address>
JBC bit, <relative address>
CJNE <destination byte>, <source byte>, <relative address>
DJNZ <byte>, <relative address>
JZ <relative address>
JNZ <relative address>

Short Absolute Jump

In this case only 11bits of the absolute jump address are needed. The absolute jump address is calculated in the following manner.

In 8051, 64 kbyte of program memory space is divided into 32 pages of 2 kbyte each. The hexadecimal addresses of the pages are given as follows:-

Page (Hex) Address (Hex)
00 0000 - 07FF
01 0800 - 0FFF
02 1000 - 17FF
03 1800 - 1FFF

.
1E F000 - F7FF
1F F800 - FFFF

It can be seen that the upper 5bits of the program counter (PC) hold the page number and the lower 11bits of the PC hold the address within that page. Thus, an absolute address is formed by taking page numbers of the instruction (from the program counter) following the jump and attaching the specified 11bits to it to form the 16-bit address.
Advantage: The instruction length becomes 2 bytes.
Example of short absolute jump: -
ACALL <address 11>
AJMP <address 11>

Long Absolute Jump/Call

Applications that need to access the entire program memory from 0000H to FFFFH use long absolute jump. Since the absolute address has to be specified in the op-code, the instruction length is 3 bytes (except for JMP @ A+DPTR). This jump is not re-locatable.
Example: -
LCALL <address 16>
LJMP <address 16>
JMP @A+DPTR
Another classification of jump instructions is
1. Unconditional Jump
2. Conditional Jump

1. **The unconditional jump** is a jump in which control is transferred unconditionally to the target location.
a. **LJMP** (long jump). This is a 3-byte instruction. First byte is the op-code and second and third bytes represent the 16-bit target address which is any memory location from 0000 to FFFFH

eg: LJMP 3000H

b. **AJMP:** this causes unconditional branch to the indicated address, by loading the 11 bit address to 0 -10 bits of the program counter. The destination must be therefore within the same 2K blocks.

c. **SJMP** (short jump). This is a 2-byte instruction. First byte is the op-code and second byte is the relative target address, 00 to FFH (forward +127 and backward -128 bytes from the current PC value). To calculate the target address of a short jump, the second byte is added to the PC value which is address of the instruction immediately below the jump.

Subroutine CALL And RETURN Instructions

Subroutines are handled by CALL and RET instructions
There are two types of CALL instructions

1. LCALL address(16 bit)

This is long call instruction which unconditionally calls the subroutine located at the indicated 16 bit address. This is a 3 byte instruction. The LCALL instruction works as follows.
a. During execution of LCALL, [PC] = [PC]+3; (if address where LCALL resides is say, 0x3254; during execution of this instruction [PC] = 3254h + 3h = 3257h
b. [SP]=[SP]+1; (if SP contains default value 07, then SP increments and [SP]=08
c. [[SP]] = [PC7-0]; (lower byte of PC content ie., 57 will be stored in memory location 08.
d. [SP]=[SP]+1; (SP increments again and [SP]=09)
e. [[SP]] = [PC15-8]; (higher byte of PC content ie., 32 will be stored in memory location 09.

With these the address (0x3254) which was in PC is stored in stack.
f. [PC]= address (16 bit); the new address of subroutine is loaded to PC. No flags are affected.

2. ACALL address(11 bit)

This is absolute call instruction which unconditionally calls the subroutine located at the indicated 11 bit address. This is a 2 byte instruction. The SCALL instruction works as follows.
a. During execution of SCALL, [PC] = [PC]+2; (if address where LCALL resides is say, 0x8549; during execution of this instruction [PC] = 8549h + 2h = 854Bh
b. [SP]=[SP]+1; (if SP contains default value 07, then SP increments and [SP]=08
c. [[SP]] = [PC7-0]; (lower byte of PC content ie., 4B will be stored in memory location 08.
d. [SP]=[SP]+1; (SP increments again and [SP]=09)
e. [[SP]] = [PC15-8]; (higher byte of PC content ie., 85 will be stored in memory location 09.

With these the address (0x854B) which was in PC is stored in stack.

f. [PC10-0]= address (11 bit); the new address of subroutine is loaded to PC. No flags are affected.

RET instruction

RET instruction pops top two contents from the stack and load it to PC.
g. [PC15-8] = [[SP]] ;content of current top of the stack will be moved to higher byte of PC.
h. [SP]=[SP]-1; (SP decrements)
i. [PC7-0] = [[SP]] ;content of bottom of the stack will be moved to lower byte of PC.
j. [SP]=[SP]-1; (SP decrements again)

Bit manipulation instructions.

8051 has 128 bit addressable memory. Bit addressable SFRs and bit addressable PORT pins. It is possible to perform following bit wise operations for these bit addressable locations.
1. LOGICAL AND
a. ANL C,BIT(BIT ADDRESS) ; ‘LOGICALLY AND’ CARRY AND CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY

b. ANL C, /BIT; ; ‘LOGICALLY AND’ CARRY AND COMPLEMENT OF CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY

2. LOGICAL OR
a. ORL C,BIT(BIT ADDRESS) ; ‘LOGICALLY OR’ CARRY AND CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY
b. ORL C, /BIT; ; ‘LOGICALLY OR’ CARRY AND COMPLEMENT OF CONTENT OF BIT ADDRESS, STORE RESULT IN CARRY
3. CLR bit
a. CLR bit ; CONTENT OF BIT ADDRESS SPECIFIED WILL BE CLEARED.
b. CLR C ; CONTENT OF CARRY WILL BE CLEARED.
4. CPL bit
a. CPL bit ; CONTENT OF BIT ADDRESS SPECIFIED WILL BE COMPLEMENTED.
b. CPL C ; CONTENT OF CARRY WILL BE COMPLEMENTED.

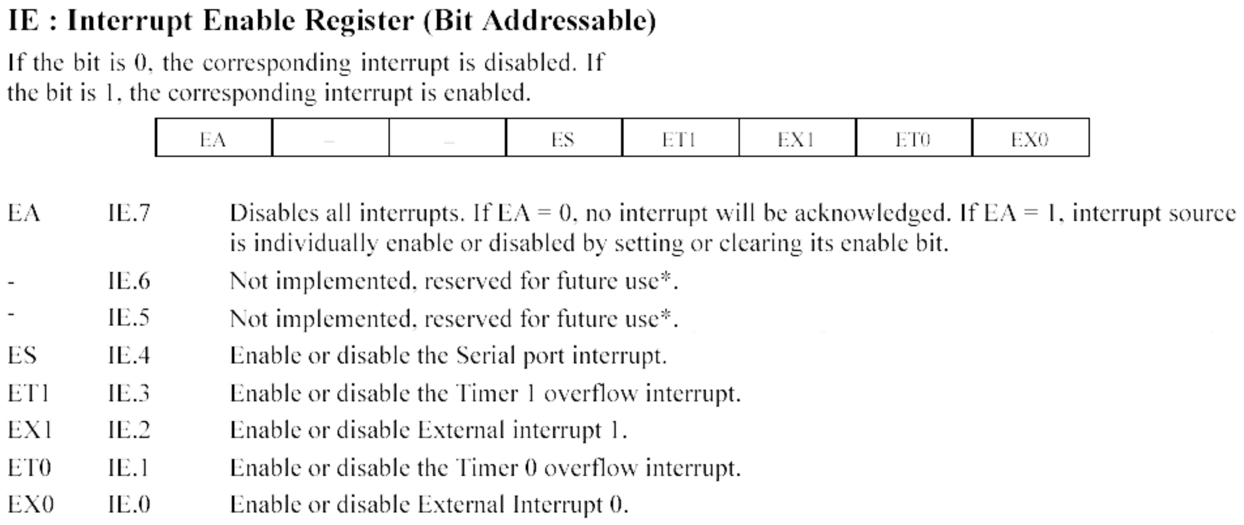
8051 INTERRUPT STRUCTURE.

8051 has five interrupts. They are maskable and vectored interrupts. Out of these five, two are external interrupt and three are internal interrupts.

Interrupt source	*Type*	*Vector address*	*Priority*
External interrupt 0	External	0003	Highest
Timer 0 interrupt	Internal	000B	
External interrupt 1	External	0013	
Timer 1 interrupt	Internal	001B	
Serial interrupt	Internal	0023	Lowest

8051 makes use of two registers to deal with interrupts.
1. IE Register

This is an 8 bit register used for enabling or disabling the interrupts. The structure of IE register is shown below.



IE : Interrupt Enable Register (Bit Addressable)

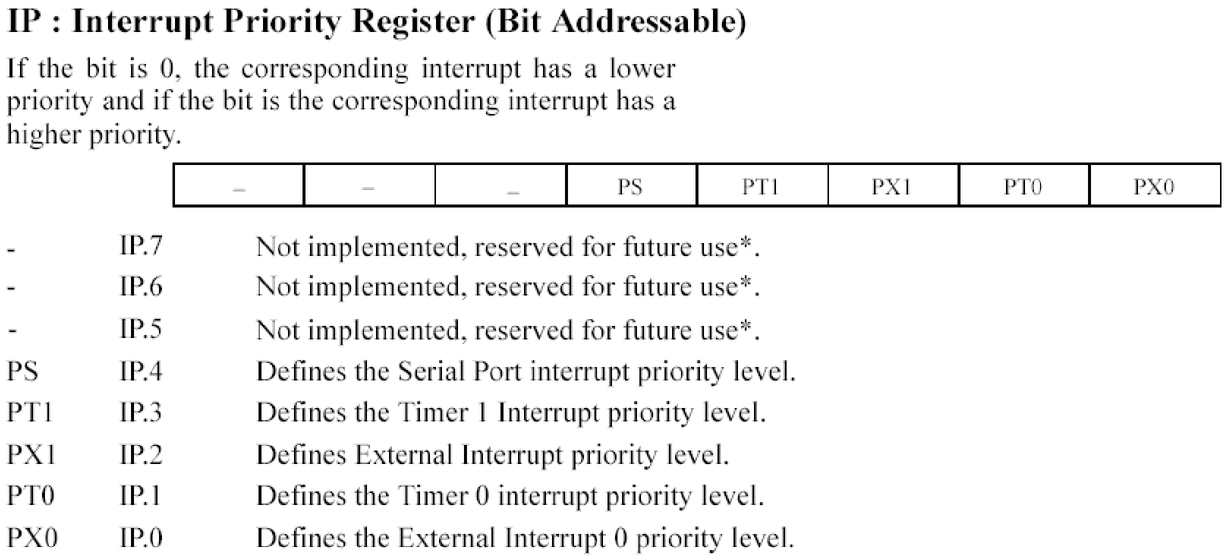
If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.

EA	–	–	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
-	IE.6	Not implemented, reserved for future use*.
-	IE.5	Not implemented, reserved for future use*.
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External Interrupt 0.

2. IP Register.

This is an 8 bit register used for setting the priority of the interrupts.



IP : Interrupt Priority Register (Bit Addressable)

If the bit is 0, the corresponding interrupt has a lower priority and if the bit is the corresponding interrupt has a higher priority.

–	–	–	PS	PT1	PX1	PT0	PX0

-	IP.7	Not implemented, reserved for future use*.
-	IP.6	Not implemented, reserved for future use*.
-	IP.5	Not implemented, reserved for future use*.
PS	IP.4	Defines the Serial Port interrupt priority level.
PT1	IP.3	Defines the Timer 1 Interrupt priority level.
PX1	IP.2	Defines External Interrupt priority level.
PT0	IP.1	Defines the Timer 0 interrupt priority level.
PX0	IP.0	Defines the External Interrupt 0 priority level.

The 8051 Timers

The basic 8051 has two on-chip timers that can be used for timing durations or for counting external events.

Interval timing allows the programmer to perform operations at specific instants in time. For example, in our LED flashing program the LED was turned on for a specific length of time and then turned off for a specific length of time. We achieved this through the use of time delays. Since the microcontroller operates at a specific frequency, we could work out exactly how many iterations of the time delay was needed to give us the desired delay.

However, this is cumbersome and prone to error. And there is another disadvantage; the CPU is occupied, stepping through the loops. If we use the on-chip timers, the CPU could be off doing something more useful while the timers take on the menial task of keeping track of time.

The Timers' SFRs

The 8051 has two 16-bit timers. The high byte for timer 1 (TH1) is at address 8DH while the low byte (TL1) is at 8BH.

The high byte for timer 0 (TH0) is at 8CH while the low byte (TL0) is at 8AH.

Both timers can be used in a number of different modes. The programmer sets the timers to a specific mode by loading the appropriate 8-bit number into the Timer Mode Register (TMOD) which is at address 89H.

Timer Mode Register

TMOD			
Bit	**Name**	**Timer**	**Description**
7	Gate	Timer1	Gate bit; when set, timer only runs while INT-bar is high. This bit is used in conjunction with interrupts and will be dealt with later.
6	C/T-bar	1	Counter/timer select bit; when set timer is an event **c**ounter, when cleared timer is an interval **t**imer.
5	M1	1	Mode bit 1
4	M0	1	Mode bit 0
3	Gate	0	Gate bit; when set, timer only runs while INT-bar is high.
2	C/T-bar	0	Counter/timer select bit; when set timer is an event **c**ounter, when cleared timer is an interval **t**imer.
1	M1	0	Mode bit 1
0	M0	0	Mode bit 0

The functions of the 8-bits of TMOD are described in the above table. The top four bits are for timer 1 and the bottom four bits have the exact same function but for timer 0.

The *Gate* bits are used in conjunction with interrupts and will be dealt with at a later stage. For the moment we can take it that bits 7 and 3 are always cleared.

As mentioned above, the timers can be used for counting external events or for timing intervals. If you wish the timer to be an event counter you set the corresponding C/T-bar bit. Similarly, if you wish it to be an interval timer you reset the corresponding C/T-bar bit.

There are two mode bits (M1 and M0) for each timer. The table below describes their function.

M1	M0	Mode	Description
0	0	0	13-bit timer mode (this mode exists simply to keep the 8051 backwards compatible with its predecessor, the 8048, which had a 13-bit timer) - we will not be using mode 0.
0	1	1	16-bit timer mode
1	0	2	8-bit auto-reload mode
1	1	3	Split timer mode - this mode will be dealt with at a later stage

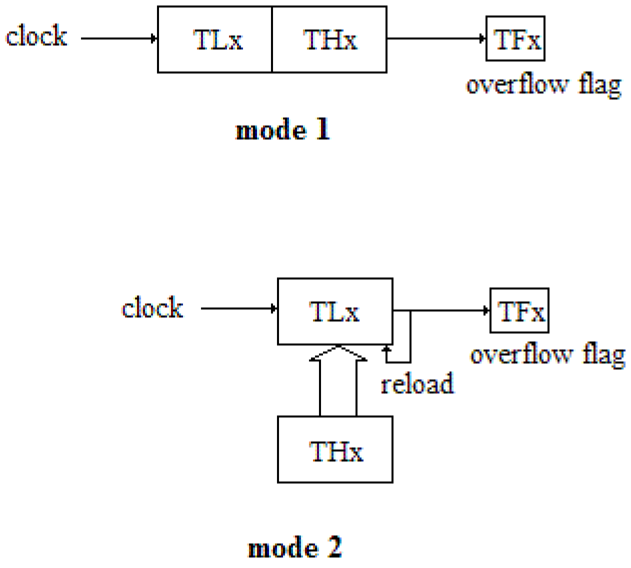
There are four timer modes, set by the bits M1 and M0. Mode 0 is not commonly used. Mode 3 we will deal with later.

Mode 1 - 16-bit mode

The high byte (THx) is cascaded with the low byte (TLx) to produce a 16-bit timer. This timer counts from 0000H to FFFFH - it has 2^{16} (65,536) states. An overflow occurs during the FFFFH to 0000H transition, setting the overflow flag (to be dealt with shortly).

Mode 2- 8-bit auto-reload mode

The timer low byte (TLx) operates as an 8-bit timer (counting to FFH) while the high byte holds a reload value. When the timer overflows from FFH, rather than starting again from 00H, the value in THx is loaded into TLx and the count continues from there.



mode 1

mode 2

A diagrammatic representation of mode 1 and mode 2.

Timer Control Register

TCON			
Bit	**Symbol**	**Bit Address**	**Description**
7	TF1	8FH	Timer 1 overflow flag; set by hardware upon overflow, cleared by software.
6	TR1	8EH	Timer 1 run-control bit; manipulated by software - setting starts timer 1, resetting stops timer 1.
5	TF0	8DH	Timer 0 overflow flag; set by hardware upon overflow, cleared by software.
4	TR0	8CH	Timer 0 run-control bit; manipulated by software - setting starts timer 0, resetting stops timer 0.
3	IE1	8BH	The bottom four bits of TCON are used in conjunction with interrupts - they will be dealt with at a later stage.
2	IT1	8AH	
1	IE0	89H	
0	IT0	88H	

The top four bits of TCON are the only ones that interest us at the moment since the bottom four are used with interrupts.

Bit 7 and bit 5 are the timer overflow flags (TFx). The overflow flag is set by the hardware once an overflow occurs. For example, if timer 0 is in mode 1 (16-bit mode) then, during the state transition from FFFFH to 0000H the overflow flag TF0 is set by the hardware.

Bit 6 and bit 4 are the timer run-control bits (TRx). A timer is started by setting TRx and stopped by clearing TRx.

The TCON register is bit addressable because the programmer needs ease of access to the individual bits. You may wish to test the contents of TF1, to see when timer 1 overflows. Or you may wish to stop timer 0 without effecting timer 1, as shown below. Clearing TR0 does not effect the other seven bits of TCON.

CLR TR0

Initialising the Timers

The timers are initialised (ie; put into a particular mode of operation) by moving the appropriate value into the TMOD register. For example, if you wished to use timer 1 as a 16-bit interval timer and timer 0 as an 8-bit auto-reload event counter you would load the following value into TMOD.

	bit	**value**	**note**
Timer 1	*7*	0	gate (set when using interrupts, reset otherwise)
	6	0	C/T-bar (0 because timer 1 is to be an interval **t**imer)
	5	0	mode bits (01 for mode 1 - 16-bit)
	4	1	
Timer 0	*3*	0	gate (set when using interrupts, reset otherwise)

	2	1	C/T-bar (1 because timer 0 is to be an event **c**ounter)
	1	1	mode bits (10 for mode 2 - 8-bit auto-reload)
	0	0	

The instruction for initialising the timers as above would therefore be:

```
MOV TMOD, #16H
```

(0001 0110$_{\text{bin}}$ = 16H)

Starting and Stopping the Timers

The run/control bits in the TCON register are used for starting and stopping the timers. As shown in the TCON table above, bit 6 is the run control bit for timer 1 while bit 4 is the run control bit for timer 0. The following two instructions start timer 0 and timer 1 respectively.

```
SETB TR0
SETB TR1
```

Setting the run control bit starts the timer. To stop the timer, you clear the corresponding run control bit

```
CLR TR0; stop timer 0
CLR TR1; stop timer 1
```

Reading the Timers on the Fly

Reading the current value of a timer when it is not running is quite easy. You simply move the high byte and the low byte to the desired memory location (usually a register).

MOV R7, TH0; move the high byte of timer 0 into R7 MOV R6, TL0; move the low byte of timer 0 into R6

However, to read the contents of a timer while it is running (ie; on the fly) poses a problem. Let's say we put timer 0's current value into R7 and R6, as shown above. The problem arises when the value in the low byte goes from FFH to 00H immediately after we read the high byte.

For example, let's say the high byte is 08H and the low byte is FFH. The controller moves the high byte into R7. Then, before it gets a chance to read the low byte, it changes from FFH to 00H. The controller then moves this value into R6, getting an overall 16-bit reading of 0800H when the value should have been 08FFH.

The solution is to read the high byte, then read the low byte, then read the high byte again. If the two readings of the high byte are not the same repeat the procedure. The code for this method is detailed below.

```
 tryAgain:
```

```
MOV A, TH0
MOV R6, TL0
CJNE A, TH0, tryAgain; if the first reading of the high byte (in A) is not equal

to current reading in the high byte (TH0) try again
MOV R7, A; if both readings of the high byte are the same move the first reading into
R7

- the overall reading is now in R7R6
```

Having said all that, I tend to not bother reading the timer on the fly. Instead, I wait for it to overflow, as we shall see,

Clock Sources

As discussed earlier, both timers can be used either as interval timers or event counters. The only difference between the timer as an interval timer and an event counter is the clock source. The timers are triggered by a negative edge to their clock inputs.

When the timer is operating as an interval timer it is being clocked by the internal clock (this internal clock, usually 12MHz, is actually divided by twelve to yield a reasonable clock frequency for most applications - therefore, the internal clocking signal is usually 1MHz).

When the timer is operating as an event counter it is triggered by an external source connected to pin T0 (port 3, pin 4) for timer 0 and pin T1 (port 3, pin 5) for timer 1. In this way the timer can count events rather than keep track of time.

For example, if a sensor on a conveyor belt system produces a negative edge everytime an item on the belt passes by, this sensor could be connected to pin T0. If timer 0 were then initialised as an event counter and started, everytime an item passed in front of the sensor, a negative edge would be generated on pin T0, thus triggering timer 0 which would proceed to the next stage in its count sequence. In this way, timer 0 is counting the number of items passing by on the conveyor belt.

Generating Pulse Trains

We can very effectively use the timers where exact timing conditions are necessary. For example, if we need to generate a 4KHz pulse train on pin 5 of port 1, we could use either timer 0 or timer 1 in 8-bit auto-reload interval timing mode.

Let's say we decide to use timer 0. We must first work out the value that must be placed in the TMOD register to put timer 0 into the desired mode.

	bit	**value**	**note**
Timer 1	*7*	0	since we are not using timer 1 we can put all four upper bits to 0
	6	0	
	5	0	
	4	0	
Timer 0	*3*	0	gate (set when using interrupts, reset otherwise)
	2	0	C/T-bar (0 because timer 0 is to be an interval **t**imer)

	1	1	mode bits (10 for mode 2 - 8-bit auto-reload)
	0	0	

Therefore, the value to be placed in TMOD is 02H.

Mode 2 is the 8-bit auto-reload mode. The low byte of the counter behaves like an 8-bit counter. The high byte is used for storing the reload value. When an overflow occurs (ie; state transition from FFH) the timer does not revert to zero. Rather, the value in the high byte is placed in the low-byte and the sequence begins again from there.

If the high byte contains 00H, then the timer behaves exactly like an 8-bit timer, counting continuously from 00H to FFH and back to 00H again.

As an interval timer, the clocking signal arrives every 1us (microsecond). The 12MHz system clock is divided by 12 and then fed to the timer. Therefore, the timer's clocking signal is 1MHz, resulting in a negative edge every 1us (time = 1/frequency).

If the high byte contains 00H, the counter goes through the full 256 states (00H to FFH). Therefore, an overflow occurs every 256us.

We require a 4KHz pulse train on pin 5 of port 1; in other words, P1.5 must toggle every 125us (4KHz signal - one cycle every 250us - one half cycle every 125us).

We need to load the high byte with a suitable value to result in an overflow every 125us. To find the value, we subtract the required time from the maximum time (256us - 125us = 131). Changing this value to HEX gives 83H.

The code for generating the pulse train is detailed below:

```
MOV TMOD, #02H; set up timer 0 as 8-bit auto-reload interval timer
MOV TH0, #83H; put reload value into timer 0 high byte
SETB TR0; start timer 0

waitForOverflow:
JNB TF0, $; repeat this line while timer 0 overflow flag is not set
CLR TF0; timer 0 overflow flag is set by hardware on transition from FFH - the flag must be reset by software

CPL P1.5; complement (invert) pin 5 on port 1 - this instruction toggles the specified bit
JMP waitForOverflow; go back and wait for overflow again
```

Note: with the above program, P1.5 is at 1 for the same length of time that it's at 0 (125us each), therefore the waveform has 50% duty cycle.

A One Second Delay

One second is a very long time for a microcontroller. If the microcontroller has a system clock frequency of 12 MHz then the longest delay we can get from either of the timers is 65,536 usec. To generate delays longer than this, we need to write a subroutine to generate a delay of (for example) 50 ms and then call that subroutine a specific number of times. To generate a 1 second delay we would call our 50 ms delay 20 times.

In the example below timer 1 is used. There is no particular reason for this; timer 0 could have been used.

The value placed in the timer before it is started is 65,536 - 50,000 = 15,536. To get a delay of 50 ms (or 50,000 us) we start the timer counting from 15,536. Then, 50,000 steps later it will overflow. Since each step is 1 us (the timer's clock is 1/12 the system frequency) the delay is 50,000 us.

```
using 0
...
MOV TMOD, #10H; set up timer 1 as 16-bit interval timer
...

fiftyMsDelay:
CLR TR1 ; stop timer 1 (in case it was started in some other subroutine)
MOV TH1, #3CH
MOV TL1, #0B0H ; load 15,536 (3CB0H) into timer 1
SETB TR1 ; start timer 1
JNB TF1, $; repeat this line while timer 1 overflow flag is not set
CLR TF1; timer 1 overflow flag is set by hardware on transition from FFFFH - the flag must be reset by software
CLR TR1 ; stop timer 1
RET

oneSecDelay:
PUSH PSW
PUSH AR0 ; save processor status
MOV R0, #20 ; move 20 (in decimal) into R0
loop:
CALL fiftyMsDelay ; call the 50 ms delay
DJNZ R0, loop ; 20 times - resulting in a 1 second delay
POP AR0
POP PSW ; retrieve processor status
RET
```

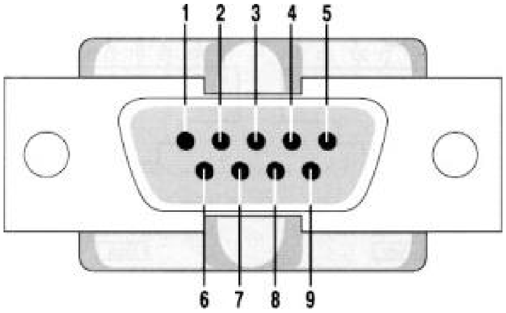
CONNECTIONS TO RS-232

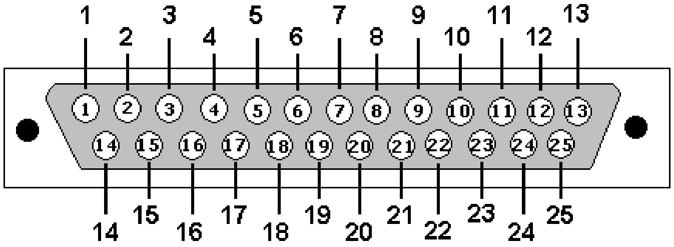
RS-232 standards:

To allow compatibility among data communication equipment made by various manufactures, an interfacing standard called RS232 was set by the Electronics Industries Association (EIA) in 1960. Since the standard was set long before the advent of logic family, its input and output voltage levels are not TTL compatible.

In RS232, a logic one (1) is represented by -3 to -25V and referred as MARK while logic zero (0) is represented by +3 to +25V and referred as SPACE. For this reason to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to convert the TTL logic level to RS232 voltage levels and vice-versa. MAX232 IC chips are commonly referred as line drivers.

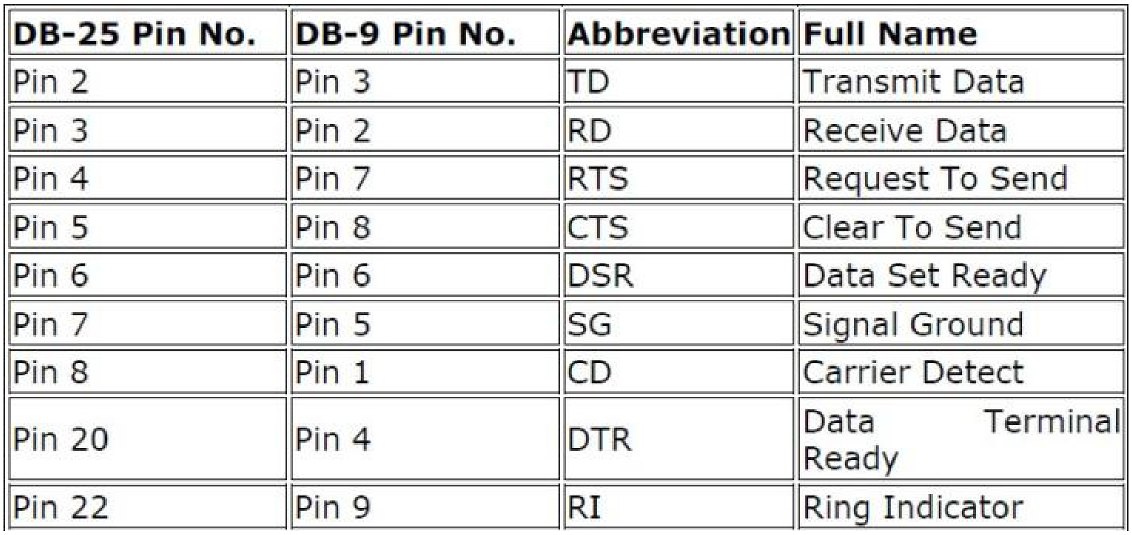
In RS232 standard we use two types of connectors. DB9 connector or DB25 connector.





DB9 Male Connector *DB25 Male Connector*

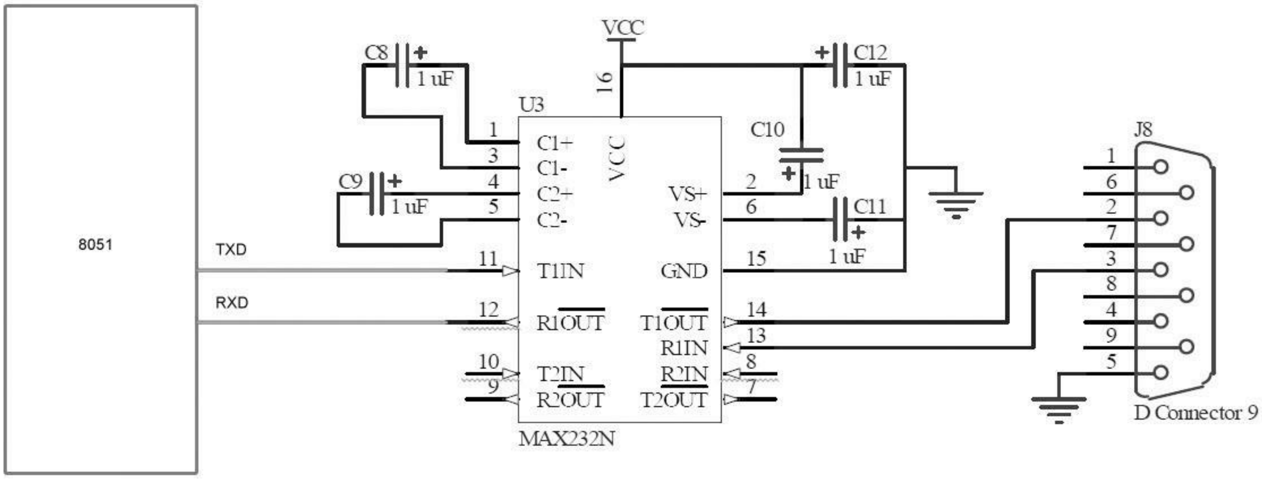
The pin description of DB9 and DB25 Connectors are as follows



DB-25 Pin No.	DB-9 Pin No.	Abbreviation	Full Name
Pin 2	Pin 3	TD	Transmit Data
Pin 3	Pin 2	RD	Receive Data
Pin 4	Pin 7	RTS	Request To Send
Pin 5	Pin 8	CTS	Clear To Send
Pin 6	Pin 6	DSR	Data Set Ready
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

The 8051 connection to MAX232 is as follows.

The 8051 has two pins that are used specifically for transferring and receiving data serially. These two pins are called TXD, RXD. Pin 11 of the 8051 (P3.1) assigned to TXD and pin 10 (P3.0) is designated as RXD. These pins TTL compatible; therefore they require line driver (MAX 232) to make them RS232 compatible. MAX 232 converts RS232 voltage levels to TTL voltage levels and vice versa. One advantage of the MAX232 is that it uses a +5V power source which is the same as the source voltage for the 8051. The typical connection diagram between MAX 232 and 8051 is shown below.



SERIAL COMMUNICATION PROGRAMMING IN ASSEMBLY AND C.

Steps to programming the 8051 to transfer data serially

1. The TMOD register is loaded with the value 20H, indicating the use of the Timer 1 in mode 2 (8-bit auto reload) to set the baud rate.
2. The TH1 is loaded with one of the values in table 5.1 to set the baud rate for serial data transfer.
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits.
4. TR1 is set to 1 start timer 1.
5. TI is cleared by the "CLR TI" instruction.
6. The character byte to be transferred serially is written into the SBUF register.
7. The TI flag bit is monitored with the use of the instruction JNB TI, target to see if the character has been transferred completely.
8. To transfer the next character, go to step 5.

Example 1. Write a program for the 8051 to transfer letter 'A' serially at 4800- baud rate, 8 bit data, 1 stop bit continuously.

```
ORG 0000H
LJMP START
ORG 0030H
START: MOV TMOD, #20H ; select timer 1 mode 2
MOV TH1, #0FAH ; load count to get baud rate of 4800
MOV SCON, #50H ; initialize UART in mode 2
; 8 bit data and 1 stop bit
SETB TR1 ; start timer
AGAIN: MOV SBUF, #'A' ; load char 'A' in SBUF
BACK: JNB TI, BACK ; Check for transmit interrupt flag
CLR TI ; Clear transmit interrupt flag
SJMP AGAIN
END
```

Interfacing to External Memory

8051 External Data Memory Interfacing 8051 has 128K bytes of address space of which 64K bytes are set aside for program code and the other 64K bytes are set aside for data. Program space is accessed using the program counter (PC) to locate and fetch instructions, but the data memory space is accessed using the DPTR register and an instruction called MOVX, where X stands for external.

External Data ROM:

To connect the 8031/51 to external ROM containing data, we use RD. For the ROM containing the program code, PSEN is used to fetch the code. For the ROM containing data, the RD signal is used to fetch the data.

MOVX is a widely used instruction allowing access to external data memory space. To bring externally stored data into the CPU, we use the instruction "MOVX A, @DPTR". This instruction will read the byte of data pointed to by register DPTR and store it in the accumulator. In writing data to external data RAM, we use the instruction "MOVX @DPTR, A" where the contents of register A are written to external RAM whose address is pointed to by the DPTR register.

Read from External Data ROM:

```
ORG 0H
 MYDATA EQU 1000H
 COUNT EQU 30
```

```
           MOV DPTR,
           #MYDATA ; point to data in external data ROM
           MOV R2, #COUNT
    AGAIN : MOVX A, @DPTR
           MOV P1, A
           INC DPTR
            DJNZ R2, AGAIN
            END
```

Memory more than 64KB:

In some applications we need a large amount (256K. bytes, for example) of memory to store data. However, the 8051 can support only 64K bytes of external data memory since DPTR is 16-bit. To solve this problem, we connect A0 – A15 of the 8051 directly to the external memory's A0 – A15 pins, and use some of the PI pins to access the 64K-byte blocks inside the single 256Kx8 memory chip.

Unit-III - 80x86 Processors - 8086 Architecture- Pin Configuration- 8086 Minimum and Maximum mode configurations- Addressing modes- Basic Instructions- 8086 Interrupts- Assembly levels programming-Introduction to 80186- 80286- 80386- 80486 and Pentium processors.

8086 Microprocessor is an enhanced version of 8085Microprocessor that was designed by Intel in 1976.It is a 16-bit Microprocessor having 20 address lines and16 data lines that provides up to 1MB storage. It consists of powerful instruction set, which provides operations like multiplication and division easily.

It supports two modes of operation, i.e. Maximum mode and Minimum mode. Maximum mode is suitable for system having multiple processors and Minimum mode is suitable for system having a single processor.

Features of 8086

The most prominent features of a 8086 microprocessor are as follows:

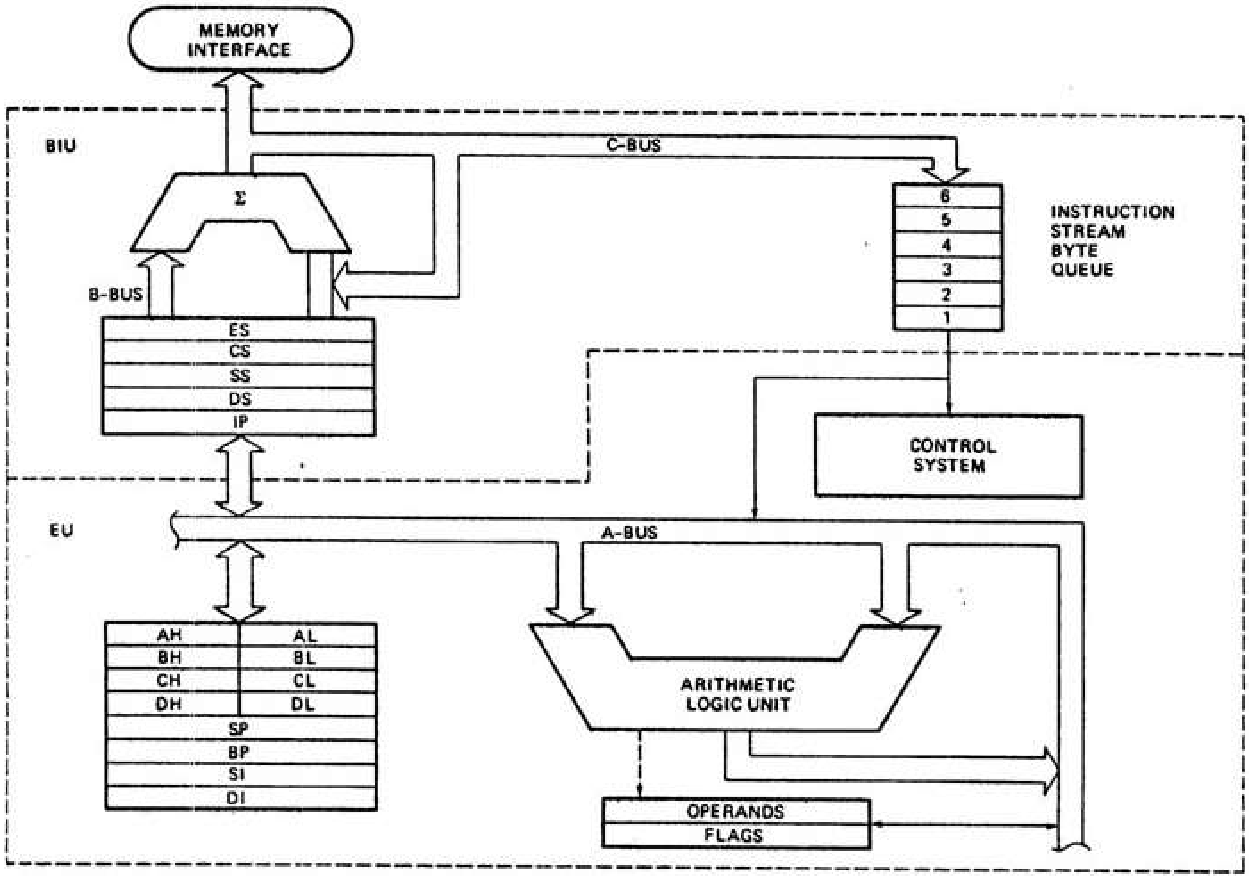
- It has an instruction queue, which is capable of storing six instruction bytes from the memory resulting in faster processing.
- It was the first 16-bit processor having 16-bit ALU, 16-bit registers, internal data bus, and 16-bit external data bus resulting in faster processing.
- It is available in 3 versions based on the frequency of operation:
    - 8086 -> 5MHz
    - 8086-2 ->8MHz
    - (c)8086-1 ->10 MHz
- It uses two stages of pipelining, i.e. Fetch Stage and Execute Stage, which improves performance.
- Fetch stage can prefetch up to 6 bytes of instructions and stores them in the queue.
- Execute stage executes these instructions.
- It has 256 vectored interrupts.
- It consists of 29,000 transistors.

Comparison between 8085 &8086 Microprocessor

- **Size**: 8085 is 8-bit microprocessor, whereas 8086 is 16-bit microprocessor.
- **Address Bus**: 8085 has 16-bit address bus while 8086 has 20-bit address bus.
- **Memory**: 8085 can access up to 64Kb, whereas 8086 can access up to 1 Mb of memory.
- **Instruction**: 8085 doesn't have an instruction queue, whereas 8086 has an instruction queue.
- **Pipelining**: 8085 doesn't support a pipelined architecture while 8086 supports a pipelined architecture.
- **I/O**: 8085 can address 2^8 = 256 I/O's, whereas 8086 can access 2^16 = 65,536 I/O's.
- **Cost**: The cost of 8085 is low whereas that of 8086 is high.

Architecture of 8086

The following diagram depicts the architecture of a 8086 Microprocessor:



8086 Microprocessor is divided into two functional units, i.e., **EU** (Execution Unit) and **BIU** (Bus Interface Unit).

EU (Execution Unit)

Execution unit gives instructions to BIU stating from where to fetch the data and then decode and execute those instructions. Its function is to control operations on data using the instruction decoder & ALU. EU has no direct connection with system buses as shown in the above figure, it performs operations over data through BIU.

Let us now discuss the functional parts of 8086 microprocessors.

ALU

It handles all arithmetic and logical operations, like +,-,×,/, OR, AND, NOT operations.

Flag Register

It is a 16-bit register that behaves like a flip-flop, i.e. it changes its status according to the result stored in the accumulator. It has 9 flags and they are divided into 2 groups: Conditional Flags and Control Flags.

Conditional Flags

It represents the result of the last arithmetic or logical instruction executed. Following is the list of conditional flags:

- **Carry flag**: This flag indicates an overflow condition for arithmetic operations.

- **Auxiliary flag**: When an operation is performed at ALU, it results in a carry/barrow from lower nibble (i.e. D0 – D3) to upper nibble (i.e. D4 – D7), then this flag is set, i.e. carry given by D3 bit to D4 is AF flag. The processor uses this flag to perform binary to BCD conversion.

- **Parity flag**: This flag is used to indicate the parity of the result, i.e. when the lower order 8-bits of the result contains even number of 1’s, then the Parity Flag is set.
For odd number of 1’s, the Parity Flag is reset.

- **Zero flag**: This flag is set to 1 when the result of arithmetic or logical operation is zero else it is set to 0.

- **Sign flag**: This flag holds the sign of the result, i.e. when the result of the operation is negative, then the sign flag is set to 1 else set to 0.

- **Overflow flag**: This flag represents the result when the system capacity is exceeded.

Control Flags

Control flags controls the operations of the execution unit. Following is the list of control flags:

- **Trap flag**: It is used for single step control and allows the user to execute one instruction at a time for debugging. If it is set, then the program can be run in a single step mode.

- **Interrupt flag**: It is an interrupt enable/disable flag, i.e. used to allow/prohibit the interruption of a program. It is set to 1 for interrupt enabled condition and set to 0 for interrupt disabled condition.

- **Direction flag**: It is used in string operation. As the name suggests when it is set then string bytes are accessed from the higher memory address to the lower memory address and vice-a-versa.

General purpose register

There are 8 general purpose registers, i.e., AH, AL, BH, BL, CH, CL, DH, and DL. These registers can be used individually to store 8-bit data and can be used in pairs to store 16- bit data. The valid register pairs are AH and AL, BH and BL, CH and CL, and DH and DL. It is referred to the AX, BX, CX, and DX respectively.

- **AX register**: It is also known as accumulator register. It is used to store operands for arithmetic operations.

- **BX register**: It is used as a base register. It is used to store the starting base address of the memory area within the data segment.

- **CX register**: It is referred to as counter. It is used in loop instruction to store the loop counter.
- **DX register**: This register is used to hold I/O port address for I/O instruction.

Stack pointer register

It is a 16-bit register, which holds the address from the start of the segment to the memory location, where a word was most recently stored on the stack.

BIU (Bus Interface Unit)

BIU takes care of all data and addresses transfers on the buses for the EU like sending addresses, fetching instructions from the memory, reading data from the ports and the memory as well as writing data to the ports and the memory. EU has no direction connection with System Buses so this is possible with the BIU. EU and BIU are connected
with the Internal Bus.

It has the following functional parts:

- **Instruction queue**: BIU contains the instruction queue. BIU gets upto 6 bytes of next instructions and stores them in the instruction queue. When EU executes instructions and is ready for its next instruction, then it simply reads the instruction from this instruction queue resulting in increased execution speed.

- Fetching the next instruction while the current instruction executes is called **pipelining**.

- **Segment register**: BIU has 4 segment buses, i.e. CS, DS, SS& ES. It holds the addresses of instructions and data in memory, which are used by the processor to access memory locations. It also contains 1 pointer register IP, which holds the address of the next instruction to executed by the EU.

  - **CS**: It stands for Code Segment. It is used for addressing a memory location in the code segment of the memory, where the executable program is stored.
  - **DS**: It stands for Data Segment. It consists of data used by the program and is accessed in the data segment by an offset address or the content of other register that holds the offset address.
  - **SS**: It stands for Stack Segment. It handles memory to store data and addresses during execution.
  - **ES**: It stands for Extra Segment. ES is additional data segment, which is used by the string to hold the extra destination data.

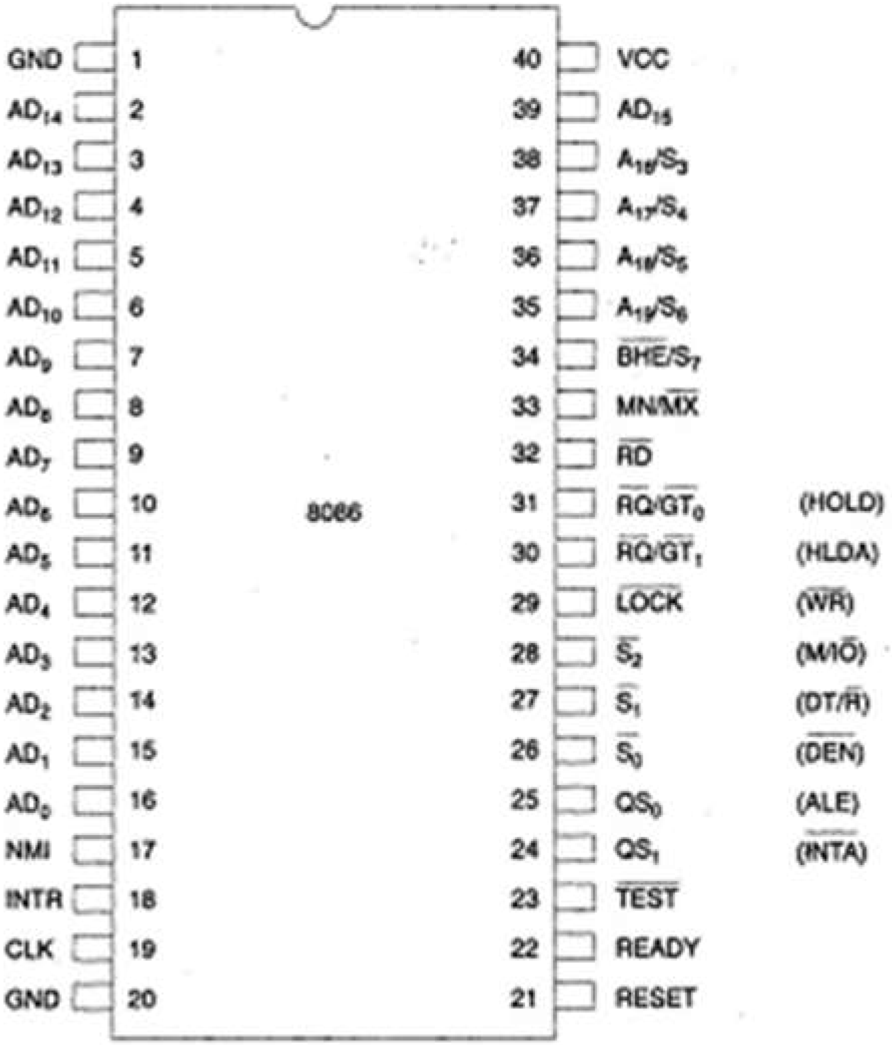
- **Instruction pointer**: It is a 16-bit register used to hold the address of the next instruction to be executed.

8086 – Pin Configuration

8086 was the first 16-bit microprocessor available in 40-pin DIP (Dual Inline Package) chip. Let us now discuss in detail the pin configuration of a 8086 Microprocessor.

8086 Pin Diagram

Here is the pin diagram of 8086 microprocessor:



Let us now discuss the signals in detail:

Power supply & frequency signals

It uses 5V DC supply at V_{CC} pin 40, and uses ground at V_{SS} pin 1 and 20 for its operation.

Clock signal

Clock signal is provided through Pin-19. It provides timing to the processor for operations. Its frequency is different for different versions, i.e. 5MHz, 8MHz and 10MHz.

Address/data bus

AD0-AD15. These are 16 address/data bus. AD0-AD7 carries low order byte data and AD8-AD15 carries higher order byte data. During the first clock cycle, it carries 16-bit address and after that it carries 16-bit data.

Address/status bus

A16-A19/S3-S6. These are the 4 address/status buses. During the first clock cycle, it carries 4-bit address and later it carries status signals.

S7/BHE

BHE stands for Bus High Enable. It is available at pin 34 and used to indicate the transfer of data using data bus D8-D15. This signal is low during the first clock cycle, thereafter it is active.

Read $(\overline{RD})$

It is available at pin 32 and is used to read signal for Read operation.

Ready

It is available at pin 32. It is an acknowledgement signal from I/O devices that data is transferred. It is an active high signal. When it is high, it indicates that the device is ready to transfer data. When it is low, it indicates wait state.

RESET

It is available at pin 21 and is used to restart the execution. It causes the processor to immediately terminate its present activity. This signal is active high for the first 4 clock cycles to RESET the microprocessor.

INTR

It is available at pin 18. It is an interrupt request signal, which is sampled during the last clock cycle of each instruction to determine if the processor considered this as an interrupt or not.

NMI

It stands for non-maskable interrupt and is available at pin 17. It is an edge triggered input, which causes an interrupt request to the microprocessor.

$\overline{TEST}$

This signal is like wait state and is available at pin 23. When this signal is high, then the processor has to wait for IDLE state, else the execution continues.

MN/$\overline{MX}$

It stands for Minimum/Maximum and is available at pin 33. It indicates what mode the processor is to operate in; when it is high, it works in the minimum mode and vice-versa.

INTA

It is an interrupt acknowledgement signal and id available at pin 24. When the microprocessor receives this signal, it acknowledges the interrupt.

ALE

It stands for address enable latch and is available at pin 25. A positive pulse is generated each time the processor begins any operation. This signal indicates the availability of a valid address on the address/data lines.

DEN

It stands for Data Enable and is available at pin 26. It is used to enable Transreceiver 8286. The transreceiver is a device used to separate data from the address/data bus.

DT/R

It stands for Data Transmit/Receive signal and is available at pin 27. It decides the direction of data flow through the transreceiver. When it is high, data is transmitted out and vice-a-versa.

M/IO

This signal is used to distinguish between memory and I/O operations. When it is high, it indicates I/O operation and when it is low indicates the memory operation. It is available at pin 28.

WR

It stands for write signal and is available at pin 29. It is used to write the data into the memory or the output device depending on the status of M/IO signal.

HLDA

It stands for Hold Acknowledgement signal and is available at pin 30. This signal acknowledges the HOLD signal.

HOLD

This signal indicates to the processor that external devices are requesting to access the address/data buses. It is available at pin 31.

QS_1& QS_0

These are queue status signals and are available at pin 24 and 25. These signals provide the status of instruction queue. Their conditions are shown in the following table:

QS0	QS1	Status
0	0	No operation
0	1	First byte of opcode from the queue
1	0	Empty the queue
1	1	Subsequent byte from the queue

S_0, S_1, S_2

These are the status signals that provide the status of operation, which is used by the Bus Controller 8288 to generate memory & I/O control signals. These are available at pin 26, 27, and 28. Following is the table showing their status:

S2	S1	S0	Status
0	0	0	Interrupt acknowledgement
0	0	1	I/O Read
0	1	0	I/O Write
0	1	1	Halt
1	0	0	Opcode fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive

LOCK

When this signal is active, it indicates to the other processors not to ask the CPU to leave the system bus. It is activated using the LOCK prefix on any instruction and is available at pin 29.

RQ/GT_1& RQ/GT_0

These are the Request/Grant signals used by the other processors requesting the CPU to release the system bus. When the signal is received by CPU, then it sends acknowledgment. RQ/GT_0 has a higher priority than RQ/GT_1.

Instruction Sets

The 8086 microprocessor supports 8 types of instructions:

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- String Instructions
- Program Execution Transfer Instructions (Branch & Loop Instructions)
- Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions

Let us now discuss these instruction sets in detail.

Data Transfer Instructions

These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group:

Instruction to transfer a word

- **MOV**: Used to copy the byte or word from the provided source to the provided destination.
- **PPUSH**: Used to put a word at the top of the stack.
- **POP**: Used to get a word from the top of the stack to the provided location.
- **PUSHA**: Used to put all the registers into the stack.
- **POPA**: Used to get words from the stack to all registers.
- **XCHG**: Used to exchange the data from two locations.
- **XLAT**: Used to translate a byte in AL using a table in the memory.

Instructions for input & output port transfer

- **IN**: Used to read a byte or word from the provided port to the accumulator.
- **OUT**: Used to send out a byte or word from the accumulator to the provided port.

Instructions to transfer the address

- **LEA**: Used to load the address of operand into the provided register.
- **LDS**: Used to load DS register and other provided register from the memory
- **LES**: Used to load ES register and other provided register from the memory.

Instructions to transfer flag registers

- **LAHF**: Used to load AH with the low byte of the flag register.
- **SAHF**: Used to store AH register to low byte of the flag register.
- **PUSHF**: Used to copy the flag register at the top of the stack.
- **POPF**: Used to copy a word at the top of the stack to the flag register.

Arithmetic Instructions

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc. Following is the list of instructions under this group:

Instructions to perform addition

- **ADD**: Used to add the provided byte to byte/word to word.
- **ADC**: Used to add with carry.
- **NC**: Used to increment the provided byte/word by 1.
- **AAA**: Used to adjust ASCII after addition.

- **DAA**: Used to adjust the decimal after the addition/subtraction operation.

Instructions to perform subtraction

- **SUB**: Used to subtract the byte from byte/word from word.
- **SBB**: Used to perform subtraction with borrow.
- **DEC**: Used to decrement the provided byte/word by 1.
- **NPG**: Used to negate each bit of the provided byte/word and add 1/2's complement.
- **CMP**: Used to compare 2 provided byte/word.
- **AAS**: Used to adjust ASCII codes after subtraction.
- **DAS**: Used to adjust decimal after subtraction.

Instruction to perform multiplication

- **MUL**: Used to multiply unsigned byte by byte/word by word.
- **IMUL**: Used to multiply signed byte by byte/word by word.
- **AAM**: Used to adjust ASCII codes after multiplication.

Instructions to perform division

- **DIV**: Used to divide the unsigned word by byte or unsigned double word by word.
- **IDIV**: Used to divide the signed word by byte or signed double word by word.
- **AAD**: Used to adjust ASCII codes after division.
- **CBW**: Used to fill the upper byte of the word with the copies of sign bit of the lower byte.
- **CWD**: Used to fill the upper word of the double word with the sign bit of the lower word.

Bit Manipulation Instructions

These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc. Following is the list of instructions under this group:

Instructions to perform logical operation

- **NOT**: Used to invert each bit of a byte or word.
- **AND**: Used for adding each bit in a byte/word with the corresponding bit in another byte/word.
- **OR**: Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.
- **XOR**: Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.
- **TEST**: Used to add operands to update flags, without affecting operands.

Instructions to perform shift operations

- **SHL/SAL**: Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
- **SHR**: Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
- **SAR**: Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

Instructions to perform rotate operations

- **ROL**: Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
- **ROR**: Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
- **RCR**: Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
- **RCL**: Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

String Instructions

String is a group of bytes/words and their memory is always allocated in a sequential order. Following is the list of instructions under this group:

- **REP**: Used to repeat the given instruction till CX≠0.
- **REPE/REPZ**: Used to repeat the given instruction until CX=0 or zero flag ZF=1.
- **REPNE/REPNZ**: Used to repeat the given instruction until CX=0 or zero flag ZF=1.
- **MOVS/MOVSB/MOVSW**: Used to move the byte/word from one string to another.
- **COMS/COMPSB/COMPSW**: Used to compare two string bytes/words.
- **INS/INSB/INSW**: Used as an input string/byte/word from the I/O port to the provided memory location.
- **OUTS/OUTSB/OUTSW**: Used as an output string/byte/word from the provided memory location to the I/O port.
- **SCAS/SCASB/SCASW**: Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
- **LODS/LODSB/LODSW**: Used to store the string byte into AL or string word into AX.

Program Execution Transfer Instructions (Branch & Loop Instructions)

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions:
Instructions to transfer the instruction during an execution without any condition:

- **CALL**: Used to call a procedure and save their return address to the stack.
- **RET**: Used to return from the procedure to the main program.
- **JMP**: Used to jump to the provided address to proceed to the next instruction.
  Instructions to transfer the instruction during an execution with some conditions:
- **JA/JNBE**: Used to jump if above/not below/equal instruction satisfies.
- **JAE/JNB**: Used to jump if above/not below instruction satisfies.
- **JBE/JNA**: Used to jump if below/equal/ not above instruction satisfies.
- **JC**: Used to jump if carry flag CF=1
- **JE/JZ**: Used to jump if equal/zero flag ZF=1
- **JG/JNLE**: Used to jump if greater/not less than/equal instruction satisfies.
- **JGE/JNL**: Used to jump if greater than/equal/not less than instruction satisfies.
- **JL/JNGE**: Used to jump if less than/not greater than/equal instruction satisfies.
- **JLE/JNG**: Used to jump if less than/equal/if not greater than instruction satisfies.
- **JNC**: Used to jump if no carry flag (CF=0)
- **JNE/JNZ**: Used to jump if not equal/zero flag ZF=0
- **JNO**: Used to jump if no overflow flag OF=0
- **JNP/JPO**: Used to jump if not parity/parity odd PF=0
- **NS**: Used to jump if not sign SF=0
- **JO**: Used to jump if overflow flag OF=1
- **JP/JPE**: Used to jump if parity/parity even PF=1
- **JS**: Used to jump if sign flag SF=1

Processor Control Instructions

These instructions are used to control the processor action by setting/resetting the flag values.
Following are the instructions under this group:

- **STC**: Used to set carry flag CF to 1
- **CLC**: Used to clear/reset carry flag CF to 0
- **CMC**: Used to put complement at the state of carry flag CF.
- **STD**: Used to set the direction flag DF to 1
- **CLD**: Used to clear/reset the direction flag DF to 0
- **STI**: Used to set the interrupt enable flag to 1, i.e., enable INTR input.
- **CLI**: Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

Iteration Control Instructions

These instructions are used to execute the given instructions for number of times. Following is the list of instructions under this group:

- **LOOP**: Used to loop a group of instructions until the condition satisfies, i.e., CX=0
- **LOOPE/LOOPZ**: Used to loop a group of instructions till it satisfies ZF=1 & CX=0
- **LOOPNE/LOOPNZ**: Used to loop a group of instructions till it satisfies ZF=0 & CX=0
- **JCXZ**: Used to jump to the provided address if CX=0

Interrupt Instructions

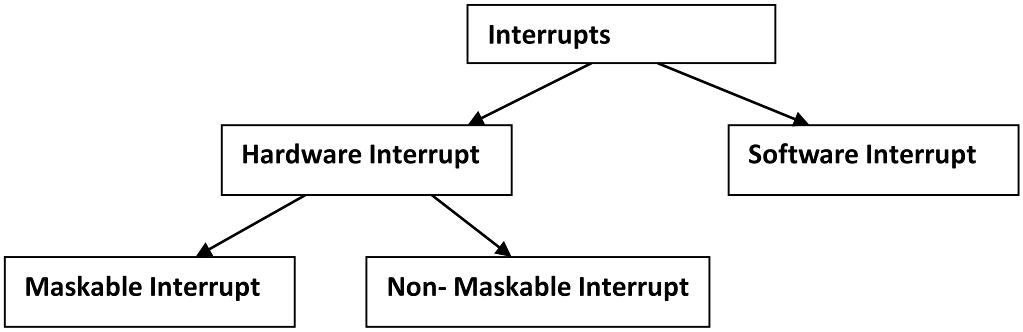
These instructions are used to call the interrupt during program execution.

- **INT**: Used to interrupt the program during execution and calling service specified.
- **INTO**: Used to interrupt the program during execution if OF=1
- **IRET**: Used to return from interrupt service to the main program

8086 – Interrupts

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an **ISR** (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

The following image shows the types of interrupts we have in a 8086 microprocessor:



Hardware Interrupts

Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

The 8086 has two hardware interrupt pins, i.e. NMI and INTR. NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.

NMI

It is a single non-maskable interrupt pin (NMI) having higher priority than the maskable interrupt request pin (INTR)and it is of type 2 interrupt.

When this interrupt is activated, these actions take place:

- Completes the current instruction that is in progress.
- Pushes the Flag register values on to the stack.
- Pushes the CS (code segment) value and IP (instruction pointer) value of the return address on to the stack.
- IP is loaded from the contents of the word location 00008H.
- CS is loaded from the contents of the next word location 0000AH.
- Interrupt flag and trap flag are reset to 0.

INTR

The INTR is a maskable interrupt because the microprocessor will be interrupted only if interrupts are enabled using set interrupt flag instruction. It should not be enabled using clear interrupt Flag instruction.

The INTR interrupt is activated by an I/O port. If the interrupt is enabled and NMI is disabled, then the microprocessor first completes the current execution and sends '0' on INTA pin twice. The first '0' means INTA informs the external device to get ready and during the second '0' the microprocessor receives the 8 bit, say X, from the programmable interrupt controller.

These actions are taken by the microprocessor:

- First completes the current instruction.
- Activates INTA output and receives the interrupt type, say X.
- Flag register value, CS value of the return address and IP value of the return address are pushed on to the stack.
- IP value is loaded from the contents of word location X x 4
- CS is loaded from the contents of the next word location.
- Interrupt flag and trap flag is reset to 0

Software Interrupts

Some instructions are inserted at the desired position into the program to create interrupts. These interrupt instructions can be used to test the working of various interrupt handlers. It includes:

INT- Interrupt instruction with type number

It is 2-byte instruction. First byte provides the op-code and the second byte provides the interrupt type number. There are 256 interrupt types under this group.

Its execution includes the following steps:

- Flag register value is pushed on to the stack.
- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of the word location 'type number' x 4
- CS is loaded from the contents of the next word location.
- Interrupt Flag and Trap Flag are reset to 0

The starting address for type0 interrupt is 000000H, for type1 interrupt is 00004H similarly for type2 is 00008H and ……so on. The first five pointers are dedicated interrupt pointers.
i.e.:

- **TYPE 0** interrupt represents division by zero situation.
- **TYPE 1** interrupt represents single-step execution during the debugging of a program.
- **TYPE 2** interrupt represents non-maskable NMI interrupt.
- **TYPE 3** interrupt represents break-point interrupt.
- **TYPE 4** interrupt represents overflow interrupt.

The interrupts from Type 5 to Type 31 are reserved for other advanced microprocessors, and interrupts from 32 to Type 255 are available for hardware and software interrupts.

INT 3-Break Point Interrupt Instruction

It is a 1-byte instruction having op-code is CCH. These instructions are inserted into the program so that when the processor reaches there, then it stops the normal execution of program and follows the break-point procedure.

Its execution includes the following steps:

- Flag register value is pushed on to the stack.
- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of the word location 3x4 = 0000CH
- CS is loaded from the contents of the next word location.
- Interrupt Flag and Trap Flag are reset to 0

INTO - Interrupt on overflow instruction

It is a 1-byte instruction and their mnemonic **INTO**. The op-code for this instruction is CEH. As the name suggests it is a conditional interrupt instruction, i.e. it is active only when the overflow flag is set to 1 and branches to the interrupt handler whose interrupt type number is 4. If the overflow flag is reset then, the execution continues to the next instruction.

Its execution includes the following steps:

- Flag register values are pushed on to the stack.
- CS value of the return address and IP value of the return address are pushed on to the stack.
- IP is loaded from the contents of word location 4x4 = 00010H
- CS is loaded from the contents of the next word location.
- Interrupt flag and Trap flag are reset to 0

8086 – Addressing Modes

The different ways in which a source operand is denoted in an instruction is known as **addressing modes.** There are 8 different addressing modes in 8086 programming:

Immediate addressing mode

The addressing mode in which the data operand is a part of the instruction itself is known as immediate addressing mode. **For example**:

```
MOV CX, 4929 H, ADD AX, 2387 H, MOV AL, FFH
```

Register addressing mode

It means that the register is the source of an operand for an instruction. **For example**:

```
MOV CX, AX   ; copies the contents of the 16-bit AX register into
             ; the 16-bit CX register),
ADD BX, AX
```

Direct addressing mode

The addressing mode in which the effective address of the memory location is written directly in the instruction. **For example**:

```
MOV AX, [1592H], MOV AL, [0300H]
```

Register indirect addressing mode

This addressing mode allows data to be addressed at any memory location through an offset address held in any of the following registers: BP, BX, DI& SI. **For example**:

```
MOV AX, [BX] ; Suppose the register BX contains 4895H, then the contents
             ; 4895H are moved to AX
ADD CX, {BX}
```

Based addressing mode

In this addressing mode, the offset address of the operand is given by the sum of contents of the BX/BP registers and 8-bit/16-bit displacement. **For example**:

```
MOV DX, [BX+04], ADD CL, [BX+08]
```

Indexed addressing mode

In this addressing mode, the operands offset address is found by adding the contents of SI or DI register and 8-bit/16-bit displacements. **For example**:

```
MOV BX, [SI+16], ADD AL, [DI+16]
```

Based-index addressing mode

In this addressing mode, the offset address of the operand is computed by summing the base register to the contents of an Index register. **For example**:

```
ADD CX, [AX+SI], MOV AX, [AX+DI]
```

Based indexed with displacement mode

In this addressing mode, the operands offset is computed by adding the base register contents. An Index registers contents and 8 or 16-bit displacement. **For example**:

```
MOV AX, [BX+DI+08], ADD CX, [BX+SI+16]
```

80186 Basic Features

- The 80186 contains 16 – bit data bus
- The internal register structure of 80186 is virtually identical to the 8086
- About the only difference is that the 80186 contain additional reserved interrupt vectors and some very powerful built-in I/O features

Clock Generator: The internal clock generator replaces the external 8284A clock generator used with the 8086 microprocessors. This reduces the component count in a system

Programmable Interrupt Controller:

The PIC arbitrates all internal and external interrupts and controls up to two external 8259A PICs. When an external 8259 is attached, the 80186 microprocessors function as the master and the 8259 functions as the slave

Timers:

- The timer section contains three fully programmable 16-bit timers
- The timers 0 and 1 generate wave-forms for external use and driven by either the master clock of the 80186 or by an external clock
- The third timer, timer 2 is internal and clocked by the master clock

Programmable DMA Unit:

- The programmable DMA unit contains two DMA channels, or four DMA channels in some models
- Each channel can transfer data between memory locations, between memory and IO, or between IO devices

Programmable chip selection unit:

- The chip selection is a built-in programmable memory and I/O decoder
- It has 6 output lines to select memory, 7 lines to select I/O

Power save/Power Down Feature:

- The power save feature allows the system clock to be divided by 4, 8, or 16 to reduce power consumption
- The power saving feature is started by software and exited by a hardware event such as an interrupt

Refresh Control Unit:

- The refresh control unit generates the refresh row address at the interval programmed

80286 Basic Features

- The 80286 microprocessor is an advanced version of the 8086 microprocessor that is designed for multi user and multitasking environments
- The 80286 addresses 16 M Byte of physical memory and 1G Bytes of virtual memory by using its memory-management system
- The 80286 is basically an 8086 that is optimized to execute instructions in fewer clocking periods than the 8086
- Like the 80186, the 80286 doesn’t incorporate internal peripherals; instead it contains a memory management unit (MMU)
- The 80286 operates in both the real and protected modes
- In the real mode, the 80286 addresses a 1MBytememory address space and is virtually identical to 8086
- In the protected mode, the 80286 addresses a 16MByte memory space
- The clock is provided by the 82284 clock generator, and the system control signals are provided by the 82288 system bus controller
- The 80286 contains the same instructions except for a handful of additional instructions that control the memory-management nit

80386 Basic Features

- The 80386 microprocessor is an enhanced version of the 80286 microprocessor and includes a
- memory-management unit is enhanced to provide memory paging
- The 80386 also includes 32-bit extended registers and a 32-bit address and data bus
- The 80386 has a physical memory size of 4GBytes that can be addressed as a virtual memory with up to 64TBytes
- The 80386 is operated in the pipelined mode, it sends the address of the next instruction or memory data to the memory system prior to completing the execution of the current instruction
- This allows the memory system to begin fetching the next instruction or data before the current is completed
- This increases access time, thus reducing the speed of the memory
- The I/O structure of the 80386 is almost identical to the 80286, except that I/O can be inhibited when the 80386 is operated in the protected mode through the I/O bit protection map
- The register set of the 80386 contains extended versions of the registers introduced on the 80286 microprocessor. These extended registers include EAX, EBX, ECX, EDX, EBP, ESP, EDI, ESI, EIP and EFLAGS
- The instruction set of the 80386 is enhanced to include instructions that address the 32-bit extended register set
- Interrupts, in the 80386 microprocessor, have been expanded to include additional predefined interrupts in the interrupt vector table
- The 80386 memory manager is similar to the 80286, except the physical addresses generated by the MMU are 32 bits wide instead of 24-bits
- The 80386 is also capable of paging
- The 80386 is operated in the real mode (i.e. 8086 mode) when it is reset
- The real mode allows the microprocessor to address data in the first 1MByte of memory
- In the protected mode, 80386 addresses any location in its 4G bytes of physical address space

80486 Basic Features

- The 80486 microprocessor is an improved version of the 80386 microprocessor that contains an 8K-byte cache and an 80387 arithmetic co processor; it executes many instructions in one clocking period
- The 80486 microprocessor executes a few new instructions that control the internal cache memory
- A new feature found in the 80486 in the BIST (built-in self-test) that tests the microprocessor, coprocessor, and cache at reset time
- If the 80486 passes the test, EAX contains a zero
- Additional test registers are added to the 80486 to allow the cache memory to be tested
- These new test registers are TR3 (cache data), TR4 (cache status), and TR5 (cache control)

Pentium Processor basic features

- The Pentium microprocessor is almost identical to the earlier 80386 and 80486 microprocessors
- The main difference is that the Pentium has been modified internally to contain a dual cache (instruction and data) and a dual integer unit
- The Pentium also operates at a higher clock speed of 66 MHz
- The data bus on the Pentium is 64 – bits wide and contains eight byte-wide memory banks
- selected with bank enable signals
- Memory access time, without wait states, is only about 18 ns in the 66 MHz Pentium
- The superscalar structure of the Pentium contains three independent processing units: a floating point processor and two integer processing units
- A new mode of operation called the System Memory Management (SMM) mode has been added to the Pentium. It is intended for high-level system functions such as power management and security
- The Built-in Self-test (BIST) allows the Pentium to be tested when power is first applied to the system
- Allows 4MByte memory pages instead of the 4KByte pages

Pentium Pro Processor basic features

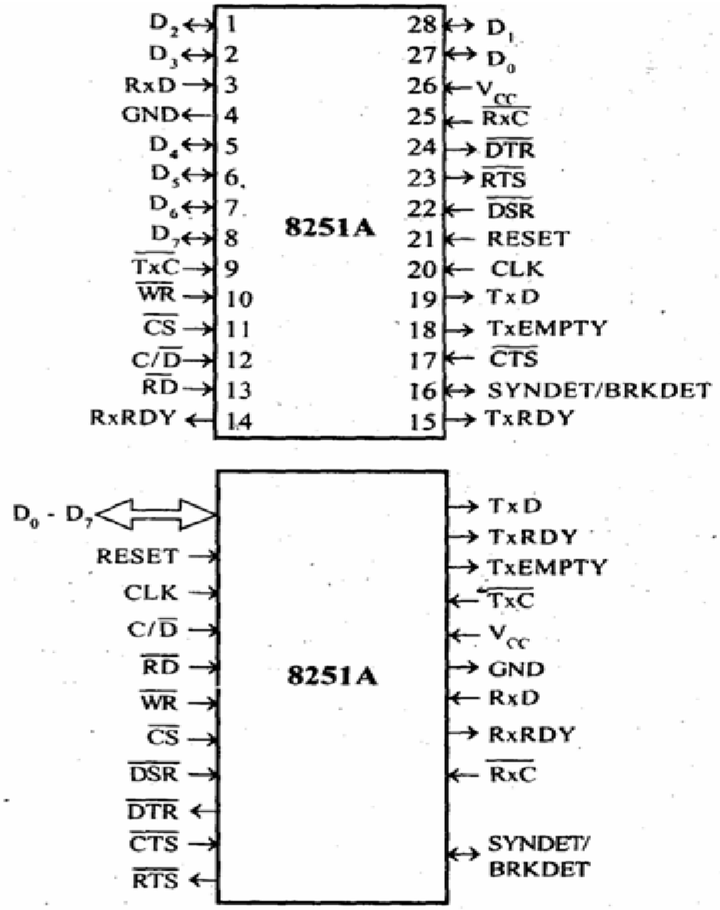
- The Pentium Pro is an enhanced version of the Pentium microprocessor that contains not only the level 1 caches found inside the Pentium, but the level 2 cache of 256 K or 512K found on most main boards
- The Pentium Pro operates using the same 66 MHz bus speed as the Pentium and the 80486
- It uses an internal clock generator to multiply the bus speed by various factors to obtain higher internal execution speeds
- The only significant software difference between the Pentium Pro and earlier microprocessors is the addition of FCMOV and CMOV instructions
- The only hardware difference between the Pentium Pro and earlier microprocessors is the addition of 2M paging and four extra address lines that allow access to a memory address space of 64G Bytes

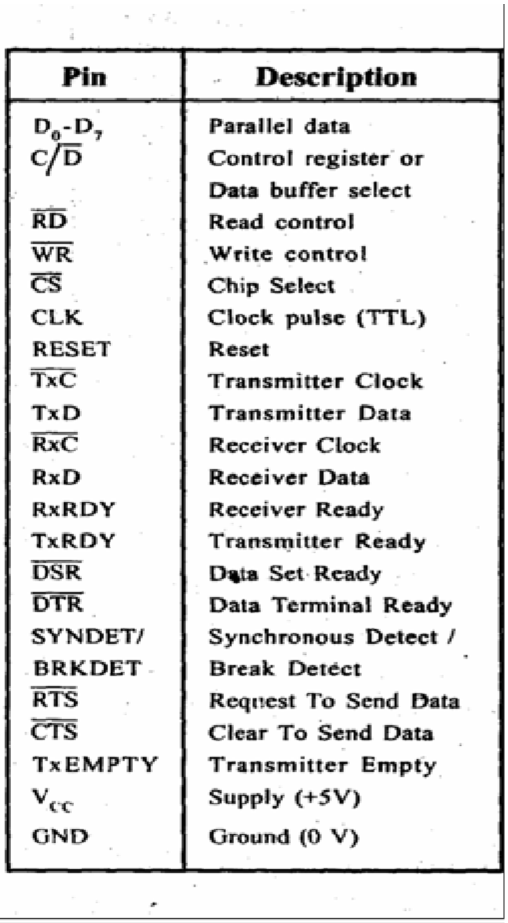
Unit-IV - Peripherals and Interfacing - Serial and parallel I/O (8251 and 8255) – Programmable DMA Controller (8257)-Programmable interrupt controller (8259)- Keyboard display ADC/DAC interfacing-Inter integrated circuits interfacing (I2C standard).

INTERFACING WITH INTEL 8251A (USART)

- The 8251A is a programmable serial communication interface chip designed for synchronous and asynchronous serial data communication.

- It supports the serial transmission of data.
- It is packed in a 28 pin DIP.





Pin	Description
D_0-D_7	Parallel data
$C/\overline{D}$	Control register or Data buffer select
$\overline{RD}$	Read control
$\overline{WR}$	Write control
$\overline{CS}$	Chip Select
CLK	Clock pulse (TTL)
RESET	Reset
$\overline{TxC}$	Transmitter Clock
TxD	Transmitter Data
$\overline{RxC}$	Receiver Clock
RxD	Receiver Data
RxRDY	Receiver Ready
TxRDY	Transmitter Ready
$\overline{DSR}$	Data Set Ready
$\overline{DTR}$	Data Terminal Ready
SYNDET/ BRKDET	Synchronous Detect / Break Detect
$\overline{RTS}$	Request To Send Data
$\overline{CTS}$	Clear To Send Data
TxEMPTY	Transmitter Empty
V_{cc}	Supply (+5V)
GND	Ground (0 V)

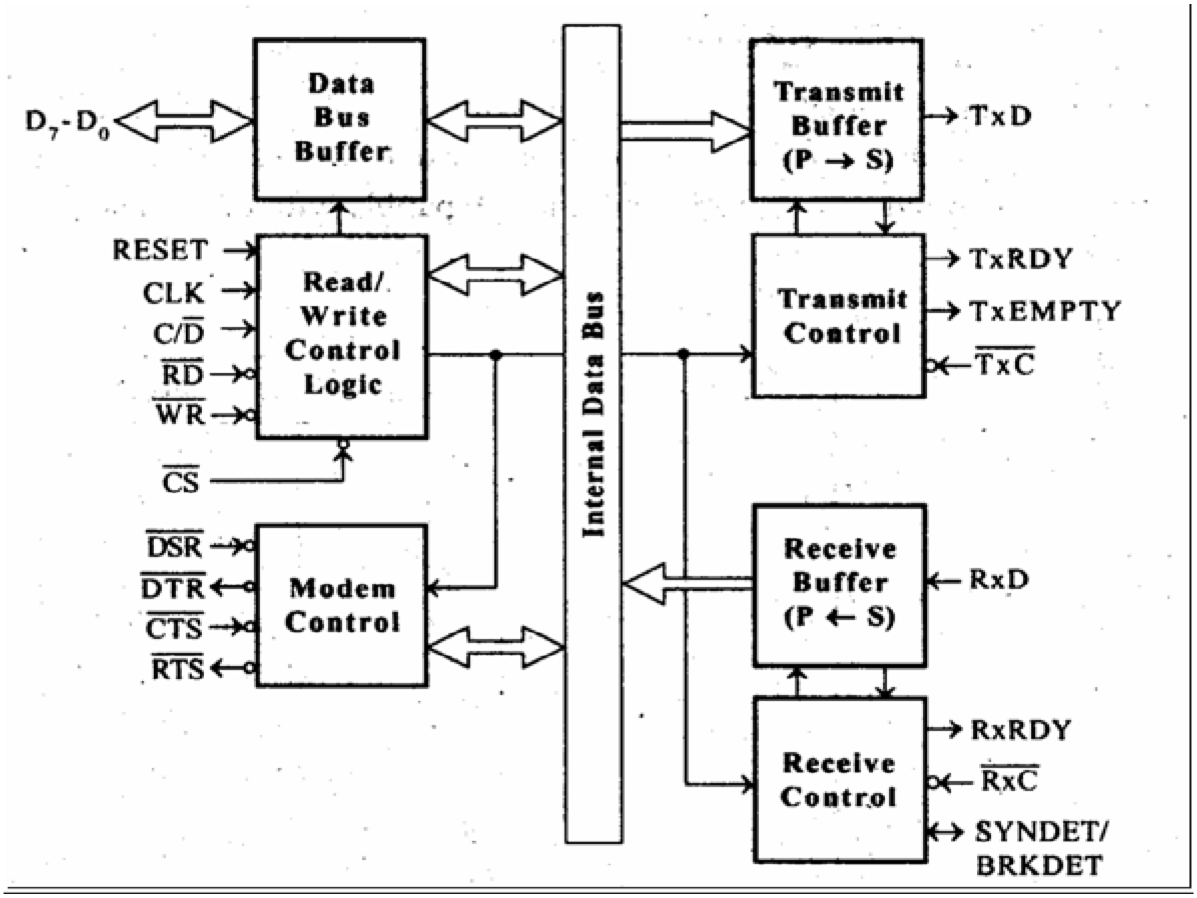
Block Diagram:

The functional block diagram of 825 1A consists five sections. They are:

- Read/Write control logic
- Transmitter
- Receiver
- Data bus buffer

- Modem control.

The functional block diagram is shown in fig:



Read/Write control logic:

- The Read/Write Control logic interfaces the 8251A with CPU, determines the functions of the 8251A according to the control word written into its control register.
- It monitors the data flow.

- This section has three registers and they are control register, status register and data buffer.
- The active low signals RD, WR, CS and C/D(Low) are used for read/write operations with these three registers.

- When C/D(low) is high, the control register is selected for writing control word or reading status word.

- When C/D(low) is low, the data buffer is selected for read/write operation.

- When the reset is high, it forces 8251A into the idle mode.

- The clock input is necessary for 8251A for communication with CPU and this clock does not control either the serial transmission or the reception rate.

Transmitter section:

- The transmitter section accepts parallel data from CPU 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, TxC (low) 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 convert 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 CPU 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 RxC (low) 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.

INTERFACING WITH INTEL 8251A (USART)

- The 825 1A can be either memory mapped or I/O mapped in the system.
- 8251A in I/O mapped in the system is shown in the figure.
- Using a 3-to-8 decoder generates the chip select signals for I/O mapped devices.
- The address lines A4, A5 and A6 are decoded to generate eight chip select signals (IOCS-0 to IOCS-7) and in this, the chip select signal IOCS-2 is used to select 8251A.
- The address line A7 and the control signal IO / M(low) are used as enable for decoder.
- The address line A0 of 8085 is connected to C/D(low) of 8251A to provide the internal addresses.
- The data lines D0 - D7 are connected to D0 - D7 of the processor to achieve parallel data transfer.
- The RESET and clock signals are supplied by the processor. Here the processor clock is directly connected to 8251A. This clock controls the parallel data transfer between the processor and 8251A.
- The output clock signal of 8085 is divided by suitable clock dividers like programmable timer 8254 and then used as clock for serial transmission and reception.
- The TTL logic levels of the serial data lines and the control signals necessary for serial transmission and reception are converted to RS232 logic levels using MAX232 and then terminated on a standard 9-pin D-.type connector.

In 8251A the transmission and reception baud rates can be different or same.

8257 DMA Controller

DMA stands for Direct Memory Access. It is designed by Intel to transfer data at the fastest rate. It allows the device to transfer the data directly to/from memory without any interference of the CPU.

Using a DMA controller, the device requests the CPU to hold its data, address and control bus, so the device is free to transfer data directly to/from the memory. The DMA data transfer is initiated only after receiving HLDA signal from the CPU.

How DMAOperations are Performed?

Following is the sequence of operations performed by a DMA −

- 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 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.
- Now the CPU is in HOLD state and the DMA controller has to manage the operations over buses between the CPU, memory, and I/O devices.

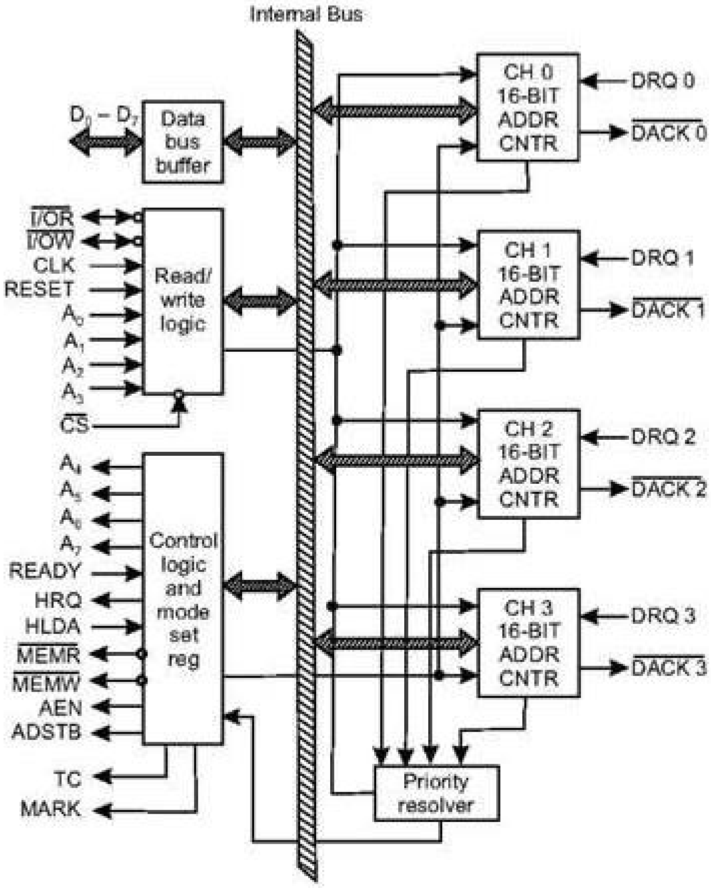
Featuresof 8257

Here is a list of some of the prominent features of 8257 −

- It has four channels which can be used over four I/O devices.
- Each channel has 16-bit address and 14-bit counter.
- 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.
- It requires a single phase clock.
- Its frequency ranges from 250Hz to 3MHz.
- It operates in 2 modes, i.e., **Master mode** and **Slave mode**.

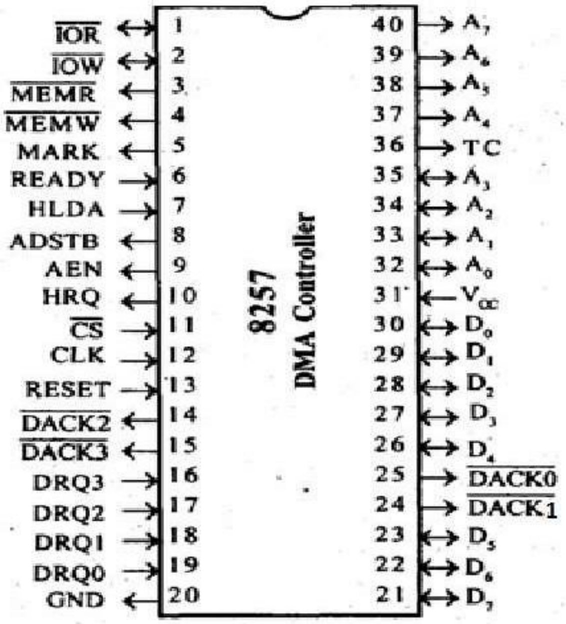
8257Architecture

The following image shows the architecture of 8257 −



8257 Pin Description

The following image shows the pin diagram of a 8257 DMA controller −



DRQ_0–DRQ**3**

These are the four individual channel DMA request inputs, which are used by the peripheral devices for using DMA services. When the fixed priority mode is selected, then DRQ_0 has the highest priority and DRQ_3 has the lowest priority among them.

$DACK_o$ – $DACK_3$

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.

D_o – D_7

These are bidirectional, data lines which are used to interface the system bus with the internal data bus of DMA controller. In the Slave mode, it carries command words to 8257 and status word from 8257. In the master mode, these lines are used to send higher byte of the generated address to the latch. This address is further latched using ADSTB signal.

IOR

It is an active-low bidirectional tri-state input line, which is used by the CPU to read internal registers of 8257 in the Slave mode. In the master mode, it is used to read data from the peripheral devices during a memory write cycle.

IOW

It is an active low bi-direction tri-state line, which is used 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 used to load the data to the peripheral devices during DMA memory read cycle.

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.

A_o - A_3

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

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.

A_4 - A_7

These are the higher nibble of the lower byte address generated by DMA in the master mode.

READY

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

HRQ

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

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

It is the low memory read signal, which is used to read the data from the addressed memory locations during DMA read cycles.

MEMW

It is the active-low three state signal which is used to write the data to the addressed memory location during DMA write operation.

ADST

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

AEN

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

TC

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

MARK

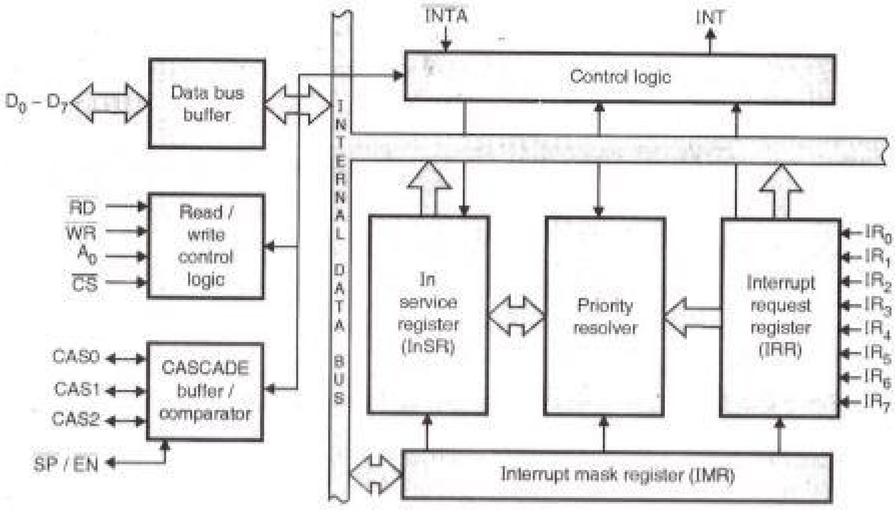
The mark will be activated after each 128 cycles or integral multiples of it from the beginning. It indicates the current DMA cycle is the 128th cycle since the previous MARK output to the selected peripheral device.

V_{cc}

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

8259 Interrupt Controller

The block diagram of 8259 is shown in the figure below:



Functional block diagram of 8259

It contains following blocks-

1. Data bus buffer-

- It is used to transfer data between microprocessor and internal bus.

1. Read/write logic-

- It sets the direction of data bus buffer.
- It controls all internal read/write operations.
- It contains initialization and operation command registers.

1. Cascaded buffer and comparator-

- In master mode, it functions as a cascaded buffer. The cascaded buffers outputs slave identification number on cascade lines.

- In slave mode, it functions as a comparator. The comparator reads slave identification number from cascade lines and compares this number with its internal identification number.
- In buffered mode, it generates an (EN)¯ signal.

1. **Control logic-**

- It generates an INT signal. In response to an (INTA)¯ signal, it releases three byte CALL address or one byte vector number.
- It controls read/write control logic, cascade buffer/comparator, in service register, priority resolver and IRR.

1. **Interrupt request register-**

- It is used to store all pending interrupt requests.
- Each bit of this register is set at the rising edge or at the high level of the corresponding interrupt request line.
- The microprocessor can read contents of this register by issuing appropriate command word.

1. **In service register (InSR)-**

- It is used to store all interrupt levels currently being serviced.
- Each bit of this register is set by priority resolver and reset by end of interrupt command word.
- The microprocessor can read contents of this register by issuing appropriate command word.

1. **Priority resolver-**

- It determines the priorities of the bit set in the IRR. To make decision, the priority resolver looks at the ISR.
- If the higher priority bit in the InSR is set then it ignores the new request.
- If the priority resolvers find that the new interrupt has a higher priority than the highest priority interrupt currently being serviced and the new interrupt is not in service, then it will set appropriate bit in the InSR and send the INT signal to the microprocessor for new interrupt request.

1. **Interrupt mask register (IMR)-**

- It is a programmable register.
- It is used to mask unwanted interrupt request by writing appropriate command word.
- The microprocessor can read contents of this register without issuing any command word.

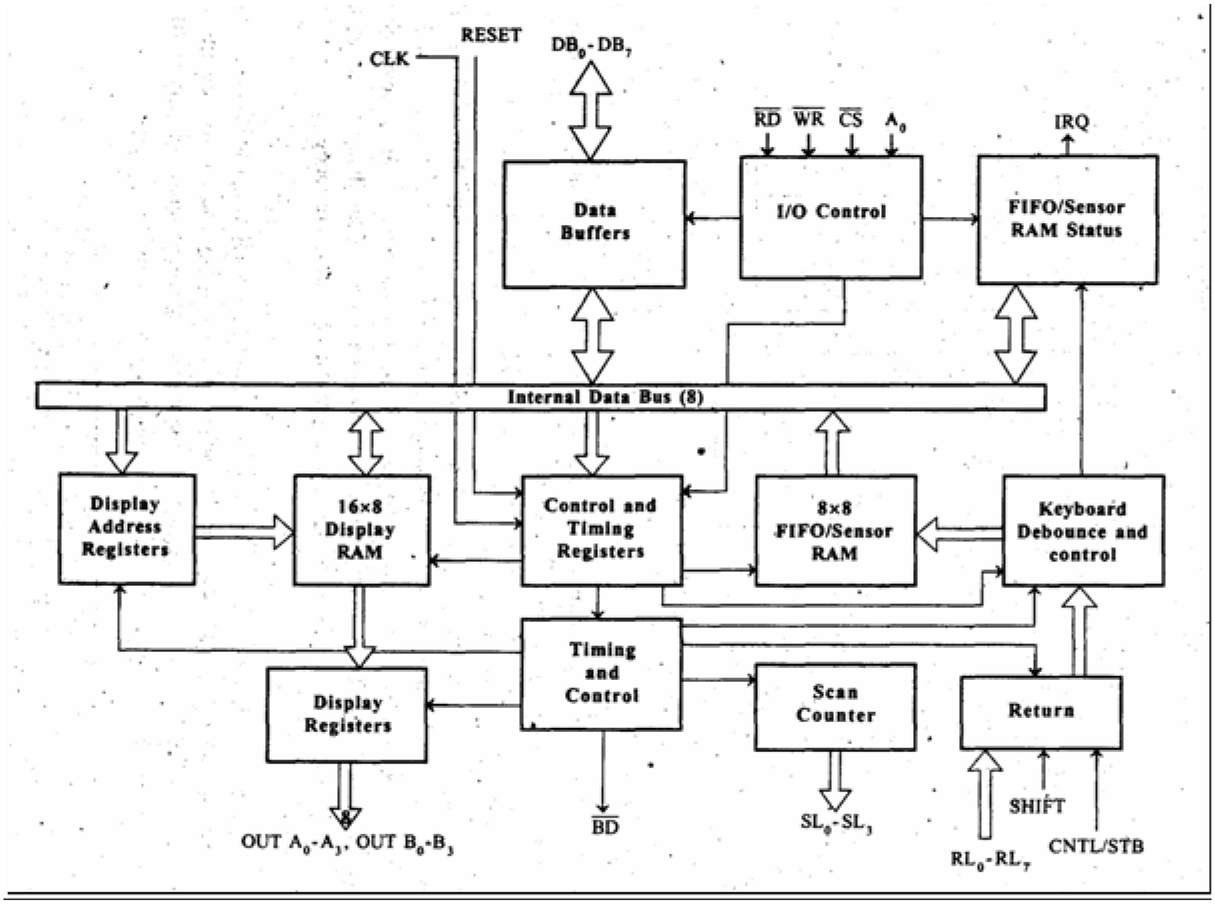
KEYBOARD/DISPLAY CONTROLLER - INTEL 8279

The INTEL 8279 is specially developed for interfacing keyboard and display devices to 8085/8086/8088 microprocessor based system. The important features of 8279 are,

- Simultaneous keyboard and display operations.
- Scanned keyboard mode.
- Scanned sensor mode.
- 8-character keyboard FIFO.
- 1 6-character display.
- Right or left entry 1 6-byte display RAM.
- Programmable scan timing.

Block diagram of 8279:

- The functional block diagram of 8279 is shown.

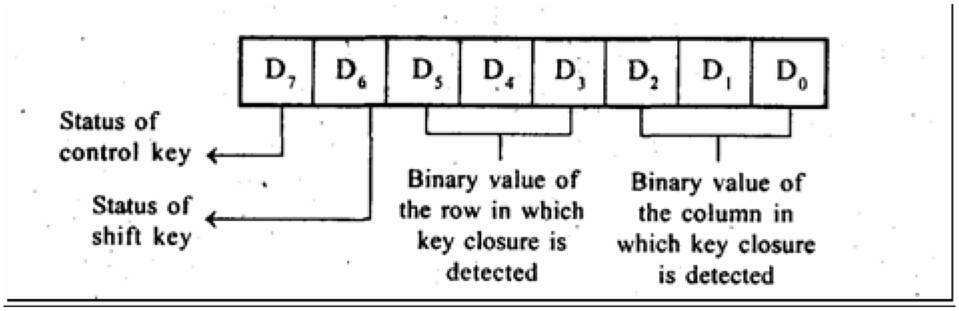


- The four major sections of 8279 are keyboard, scan, display and CPU interface.

Keyboard section:

- The keyboard section consists of eight return lines RL0 - RL7 that can be used to form the columns of a keyboard matrix.

- It has two additional input : shift and control/strobe. The keys are automatically debounced.
- The two operating modes of keyboard section are 2-key lockout and N-key rollover.
- In the 2-key lockout mode, if two keys are pressed simultaneously, only the first key is recognized.
- In the N-key rollover mode simultaneous keys are recognized and their codes are stored in FIFO.
- The keyboard section also have an 8 x 8 FIFO (First In First Out) RAM.
- The FIFO can store eight key codes in the scan keyboard mode. The status of the shift key and control key are also stored along with key code. The 8279 generate an interrupt signal when there is an entry in FIFO. The format of key code entry in FIFO for scan keyboard mode is,


D_7
D_6
D_5
D_4
D_3
D_2
D_1
D_0
Status of control key
Status of shift key
Binary value of the row in which key closure is detected
Binary value of the column in which key closure is detected

- In sensor matrix mode the condition (i.e., open/close status) of 64 switches is stored in FIFO RAM. If the condition of any of the switches changes then the 8279 asserts IRQ as high to interrupt the processor.

Display section:

- The display section has eight output lines divided into two groups A0-A3 and B0-B3.
- The output lines can be used either as a single group of eight lines or as two groups of four lines, in conjunction with the scan lines for a multiplexed display.
- The output lines are connected to the anodes through driver transistor in case of common cathode 7-segment LEDs.
- The cathodes are connected to scan lines through driver transistors.
- The display can be blanked by BD (low) line.
- The display section consists of 16 x 8 display RAM. The CPU can read from or write into any location of the display RAM.

Scan section:

- The scan section has a scan counter and four scan lines, SL0 to SL3.
- In decoded scan mode, the output of scan lines will be similar to a 2-to-4 decoder.
- In encoded scan mode, the output of scan lines will be binary count, and so an external decoder should be used to convert the binary count to decoded output.
- The scan lines are common for keyboard and display.

- The scan lines are used to form the rows of a matrix keyboard and also connected to digit drivers of a multiplexed display, to turn ON/OFF.

CPU interface section:

- The CPU interface section takes care of data transfer between 8279 and the processor.
- This section has eight bidirectional data lines DB0 to DB7 for data transfer between 8279 and CPU.
- It requires two internal address A =0 for selecting data buffer and A = 1 for selecting control register of8279.
- The control signals WR (low), RD (low), CS (low) and A0 are used for read/write to 8279.
- It has an interrupt request line IRQ, for interrupt driven data transfer with processor.
- The 8279 require an internal clock frequency of 100 kHz. This can be obtained by dividing the input clock by an internal prescaler.
- The RESET signal sets the 8279 in 16-character display with two -key lockout keyboard modes.

Programming the 8279:

- The 8279 can be programmed to perform various functions through eight command words.

<u>INTERFACING OF 8279 WITH 8085</u>

In a microprocessor b system, when keyboard and 7-segment LED display is interfaced using ports or latches then the processor has to carry the following task.

- Keyboard scanning
- Key debouncing
- Key code generation
- Sending display code to LED
- Display refreshing

Introducing I2C

• The name I2C is shorthand for Standard Inter-Integrated Circuit bus

• I 2C is a serial data protocol which operates with a master/slave relationship

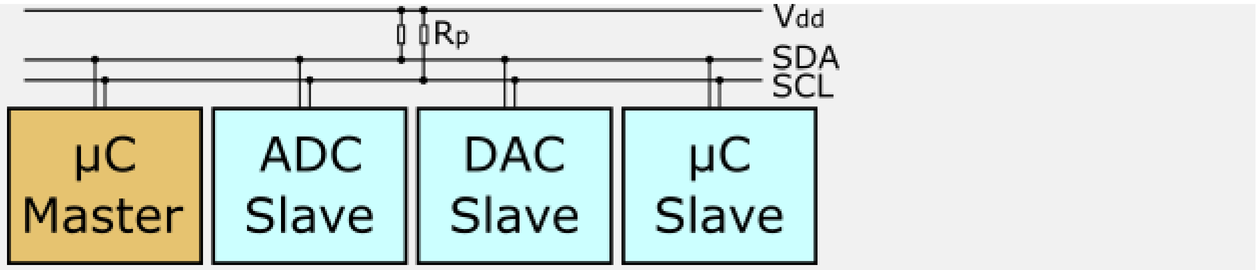
• I 2C only uses two physical wires, this means that data only travels in one direction at a time.

I2C

I2C is ordinarily pronounced “I-two-C”, though it is also sometimes written as IIC (and pronounced “I- I-

C”) or I^{2}C (pronounced “I-squared-C”). The acronym stands for Inter-Integrated-Circuit. It is a type of serial computer bus and communications protocol that was first introduced to the market by Philips Semiconductor in 1982.

I2C is a way of allowing multiple electronic devices (most often low-speed, peripheral integrated circuits) to communicate with each other over a single pair of wires. These wires are also called data lines, or buses. The first of these buses is the data line and is called the SDA (**S**erial **DA**ta) line, and the other bus is the clock, or SCL (**S**erial **CL**ock) line. Since all devices on any I2C circuit are hooked to these two lines to communicate, most I2C-compatible devices have pins labeled SDA and SCL, as well as VIN and GND pins for positive and ground connections.



Typical I2C Schematic / licensed under CC BY-SA 3.0

Both the clock and data buses are open-drain lines. A resistor must be connected between each line and the circuit's positive voltage supply (also referred to as Vcc) in order for the bus to work correctly. The size of these resistors can vary, from 1 kΩ all the way up to 47 kΩ, but they must be present between the bus and the system's HIGH voltage. If they are not present, all lines will be pulled low, and the I2C bus will fail to work. Luckily, only one pair of resistors (one for each line) is necessary for the entire system, not a pair for each device–that could quickly get messy with a lot of devices.

Quite a few devices (also called nodes) can be attached to these buses. In fact, the number of devices that can practically be connected to any I2C circuit is normally limited only by space, the inherent capacitance of the lines, and the addresses of the connected devices. Most experts agree that this limit falls around 1008 devices.

There are two kinds of devices in I2C communications: masters and slaves. In most implementations of the protocol there is one master device connected to many slaves. It is possible to have more than one master communicating with various slave devices as well as other masters, but this form is less common and is a bit beyond the scope of this introduction. Many I2C devices can be configured as either a master or a slave, depending on the desired system results. The master node is the one device that controls the clock (SCL) line, and is the only device that can initiate a data transfer. The slave nodes are limited to listening for and responding to calls from the master node. Each node, including the master, has a unique (normally 7-bit) address that identifies it on the I2C network. In some cases the address can be 10 bits in length, allowing for more than 128 different devices, but this is not a normal setup.

During the operation of an I2C system, the master node sends commands and requests on the data line. These signals are 8 bits in length, are only begun when the clock line is HIGH, and are started with a particular 'start' sequence and finished with a particular 'stop' sequence. The start sequence alerts all connected slave nodes that a data transfer request is imminent. The next sequence the master sends out is the address of the slave with which it wants to communicate. The named slave node then responds to the master, beginning at the next HIGH clock signal, and the other slaves go back to listening for their address to be called.

The eighth bit that is sent with the command, after the 7-bit device address, is a simple read/write bit. It tells the addressed device whether the master device will be reading from or writing to it. This allows the receiving device to either prepare data to send or prepare to receive data along the SDA line at the next clock HIGH signal.

The clock speed (and the associated signal transfers) on most I2C circuits normally falls somewhere between serial communications and SPI speeds, between 100kHz and 400kHz. I2C is commonly used in systems where simplicity, low cost, and low power are more important than speed. Some of these applications include analog-to-digital converters, LCD displays, real-time clocks, and many different sensors such as barometers, compasses, and even GPS receivers.

I2C is a very useful communications protocol, though it may only be applicable to a small number of applications. It is ideal for small, low-power-consumption settings, and as a result has found a following among many hobbyists, who use it to interface sensors and controls with embedded devices and microcontrollers such as the Arduino and the Raspberry Pi. Two of the Pi's GPIO pins are preset to interface with devices using the protocol, and the Arduino wire library allows communications with I2C devices. Learning and using I2C can significantly extend a hobbyist's toolkit when it comes to building

UNIT – V

8051 Microcontroller based Systems Design

INTERFACING TO ALPHANUMERIC DISPLAYS

- Many microprocessor-controlled instruments and machines need to display letters of the alphabet and numbers.

- Light Emitting diodes (LEDs) and Liquid-Crystal Displays (LCDs) are used to display letters and alphabet.

- LCD displays use very low power, so they are often used in portable, battery- powered instruments.

Interfacing LED Displays to Microcomputers

- Alphanumeric LED displays are available in three common formats. A 7-segment displays used to display numbers and hexadecimal letters
- To display numbers and the entire alphabet, 18 segment displays isused
- The 7-segmen t display is most commonly used, and easiest to interface with microprocessor

DIRECTLY DRIVING LED DISPLAYS

- Figure 5.1, shows a circuit to display single digit drivercircuit
- The BCD (Binary Coded Decimal) co de is applied to thiscircuit
- The 7447 decoder IC converts a BCD code applied to its inputs to 7 segment code to display the number represented by the BCDcode.

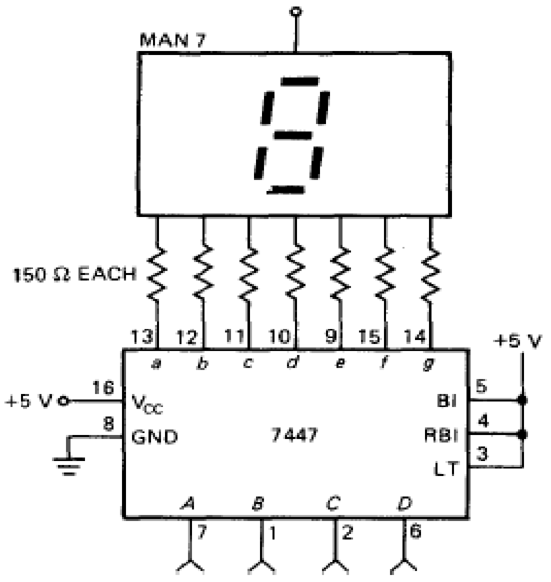


Fig 5.1 Single digit seven segment circuit

- The above circuit is used to display the single digit

- We have to increase the number of 7-segment and 7447 IC to view more number of digits (this arrangement is generally referred to as static display approach)

Limitation

We have the limitation by increasing the number of ICs

1. The first problem is power consumption
2. A second problem of the sta tic approach is that each display digit requires a separate 7447 decoder, The current required by the decoders and the LED displays might be several times the current required by the rest of the circuitry in the instrument.

INTERFACING TO MULTIPLEXED DISPLAYS

- To solve the problems of the static display approach, we use a multiplex method.
- Figure 5.2 shows a circuit for the multiplexed display interface with microcomputer or microprocessor
- In this multiplexed circuit has only one 7447 and that the segment outputs of the 7447 are bused in parallel to the segment inputs of all the digits.
- Multiplexing displays is that only one display digit is turned on at a time.
- The PNP transistor in series with the common anode of each digit acts as an on/off switch for that digit.

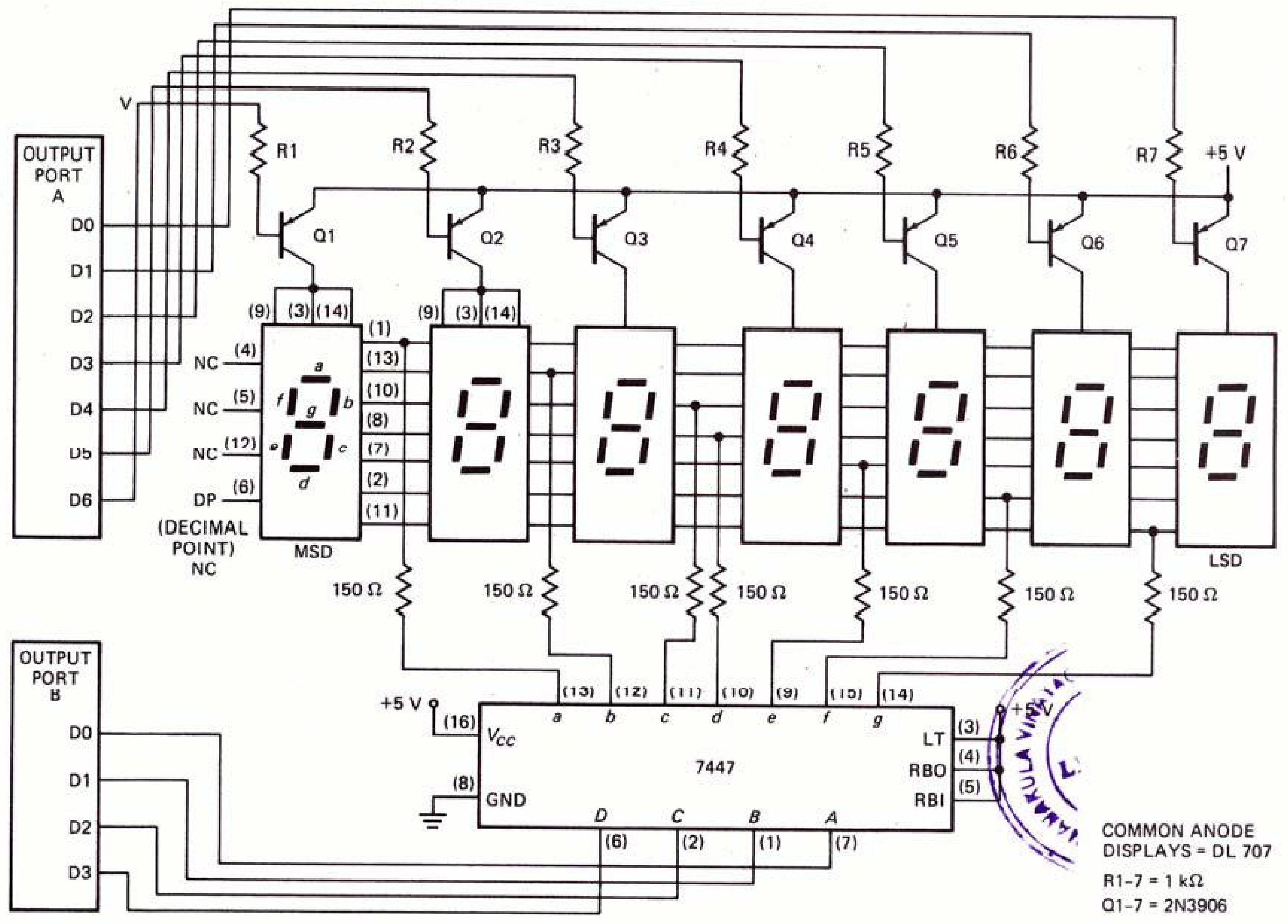


Fig 5.2 Multiplexed LED display with microprocessor

Display for Digit-1

.

- The BCD code for digit 1 is first output from port B to the 7447.
- The 7447 outputs the corresponding 7segment code on the segment bus lines.
- The transistor connected to digit 1 is then turned on by outputting a low to the appropriate bit of port A. (Remember, a low turns on a PNP transistor.)
- All the rest of the bits of port A are made high to make sure no other digits are turned on. After 1 or 2 ms, digit 1 is turned off by outputting all highs to port A.

Display for Digit-2 and other digits

- The BCD code for digit 2 is then output to the 7447 on port B, and a word to turn on digit 2 is output on port A.
- After 1 or 2 ms, digit 2 is turned off and the process is repeated for digit 3.
- The process is continued until all the digits have had a turn.
- Then digit 1 and the following digits are lit again in turn.
- This refresh rate is fast enough that, to your eye, the digits will each appear to be lit all the time. Refresh rates of 40 to 200 times a second are acceptable.

advantages of multiplexing the displays are that only one 7447 is required, and only one digit is lit at a time. We usually increase the current per segment to between 40 and 60 mA for multiplexed displays so that they will appear as bright as they would if they were not multiplexed.

LIQUID-CRYSTAL DISPLAY INTERFACING

- Most LCDs require a voltage of 2 or 3 V between the backplane and a segment to turn on the segment.
- The segment-drive signals for LCDs must be square waves with a frequency of 30 to 150 Hz.
- The 7211M (Programmable 4 to 7 decoder) inputs can be connected to port pins or directly to microcomputer buses as shown.
- To display a character on one of the digits by simply put the 4-bit hex code to the input of Programmable 4 to 7 decoder
- The ICM7211 M converts the 4-bit hex code to the required 7-segment code.
- The rising edge of the CS input signal causes the 7-segment code to be latched in the output latches for the addressed digit.
- An internal oscillator automatically generates the segment and backplane drive waveforms shown in Figure 5.3.

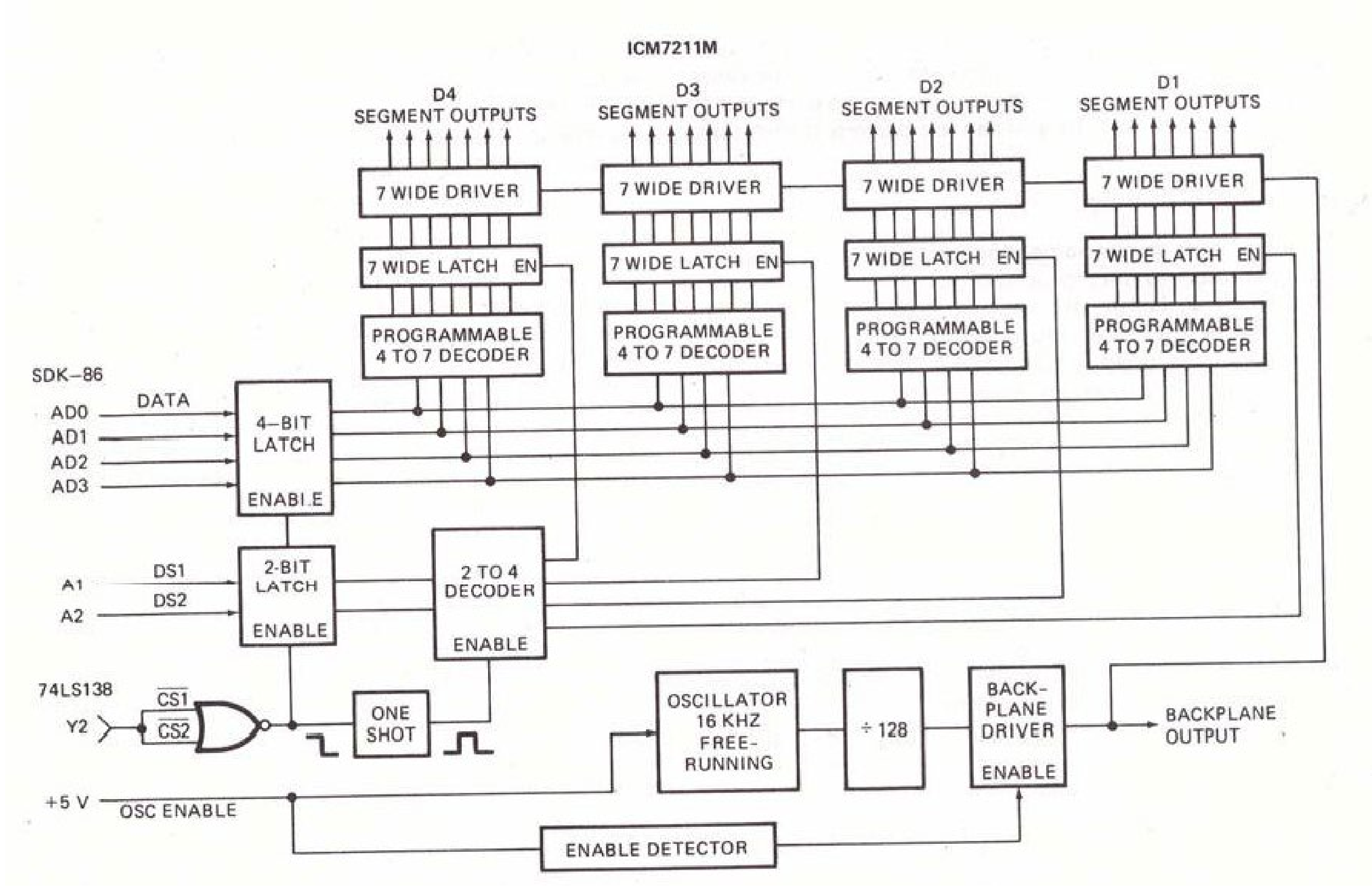


Fig 5.3 non- multiplexed LCD display interface

INTERFACING MICROCOMPUTER PORTS TO HIGH-POWER DEVICES

- The output pins on programmable port devices is a few tenths of a milliampere from the +5-V supply
- This small current (voltage) is not sufficient to drive high-power devices such as lights, heaters, solenoids, and motors with a microcomputer.
- Due to above reason we must use interfacedevices between the port pins and the high-power device.
- This section shows you a few of the commonly used devicesand techniques.

Interfacing to AC Power Devices

- A relay is used to turn 110-V, 220-V, or 440-V ac devices on and offunder microprocessor control

INTERFACING A MICROCOMPUTER TO A STEPPER MOTOR

Features of Stepper motor

- A unique type of motor useful for moving things (shaft) in small increments is a stepper motor.
- Stepper motor rotate or "step," from one fixed position to the next.
- Common step sizes for stepper motors range from 0.9° to 30°.
- A stepper motor is stepped from one position to the next by changing the currents through the. fields in the motor.

Note: Stepper motor operation is given in the end of this lecture notes

- The two common field connections are referred to as two-phase and four- phase.
- Figure 5.4, shows a circuit to the interface a small four-phasestepper
- Since the 7406 buffers are inverting, a high on an output-port pin produces a low on a buffer output. This low turns on the PNP driver transistor and supplies current to a winding.
- Table 5.1 shows the switching sequence to step a motor such as this clockwise or counterclockwise.

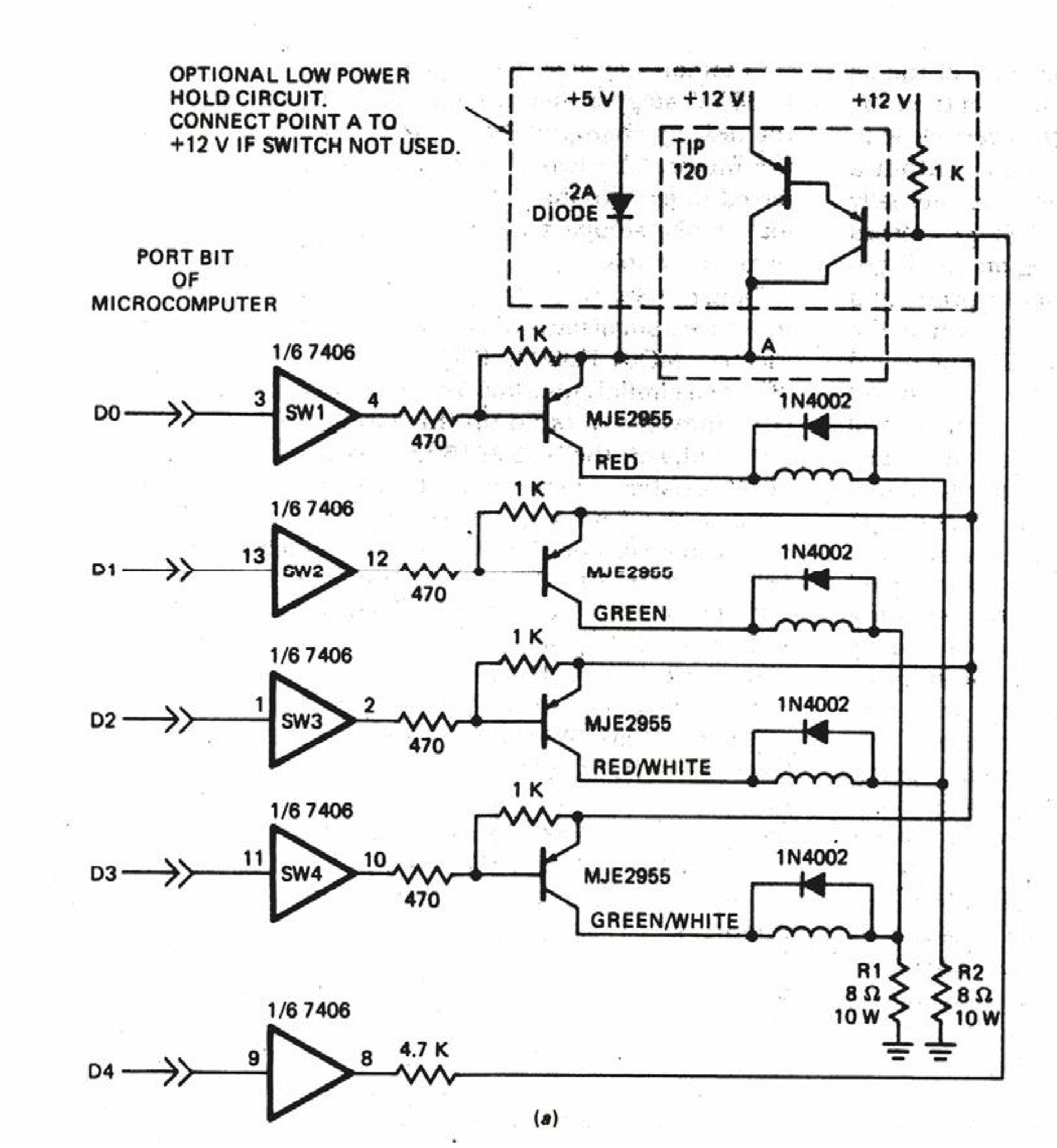
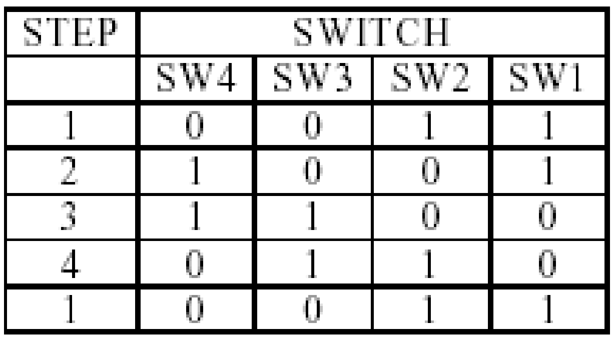


Fig 5.4 four phase stepper motor interface Table 5.1 switching sequence for full step drive signal



STEP	SWITCH			
	SW4	SW3	SW2	SW1
1	0	0	1	1
2	1	0	0	1
3	1	1	0	0
4	0	1	1	0
1	0	0	1	1

- Suppose that SW 1 and SW2 are turned on. Turning off SW2 and turning on SW4 will cause the motor to rotate one step of 1.8° clockwise.
- Changing to SW4 and SW3 on will cause the motor to rotate another 1.8° clockwise.
- Changing to SW3 and SW2 on will cause another step.
- After that, changing to SW2 and SW 1 on again will cause another step clockwise.
- By repeat the sequence until the motor has rotated as many steps clockwise.
- To step the motor counterclockwise, change switch sequence in the reverse direction.
- The motor is held in position between steps by the current through the coils.
- When the step a stepper motor to a new position, it tends to oscillate around the new position before settling down.
- A common software technique to damp out this oscillation is to send the pattern first to step the motor toward the new position.
- When the motor has rotated part of the way to the new position, a word to step the motor backward is output for a short time.
- The step-forward word is then sent again to complete the step to the next position.
- The optional transistor switch and diode connection to the + 5-V supply are used as follows.
  - When the motor is not stepping, the switch to + 12 V is off, so the motor is held in position by the current from the + 5-V supply.
  - Before you send a step command, you turn on the transistor to + 12 V to give the motor more current for stepping.
  - When stepping is done, you turn off the switch to + 12 V, and drop back to the +5-V supply. This cuts the power dissipation.

OPTICAL MOTOR SHAFT ENCODERS

- In order to control the machines in the electronics industry, the microcomputers in these machines often need information about the position, direction of rotation, and speed of rotation of various motor shafts.

- The microcomputer, of course, needs this information in digital form. The circuitry which produces this digital information from each motor for the microcomputer is called a shaft encoder .

1. Absolute shaft encoder
2. Incremental shaft encoder.

Absolute Encoders

- Absolute encoders have a binary-coded disk such as the one shown in Figure 5.4 on the rotating shaft.

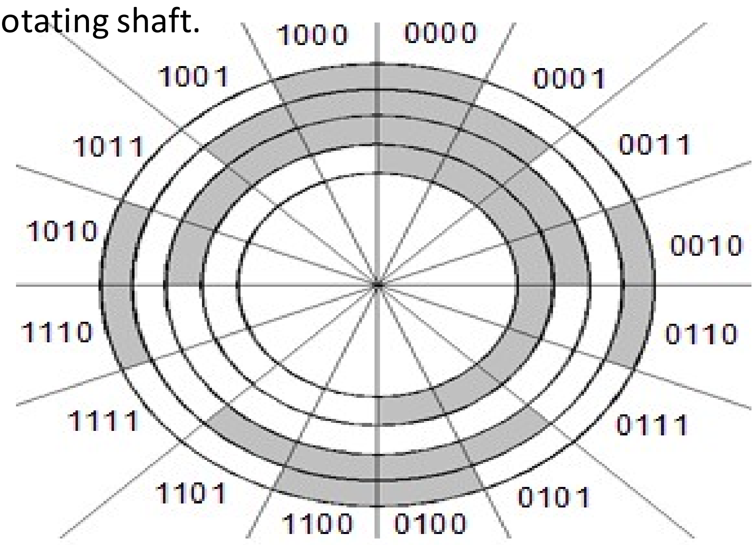


Fig. 1

Types of shaft encoder

Fig 5.4 Gray-code optical-encoder dish used to determine angular position of a rotating shaft.

- Light sections of the disk are transparent, and dark sections areopaque.
- An LED is mounted on one side of each track, and a phototransistor is mounted on the other side of each track, opposite theLED.
- Outputs from the four phototransistors will produce one of the binary codes
- The phototransistor outputs can be conditioned with Schmitt-trigger buffers and connected to input port lines.
- Each code represents an absolute angular position of the shaft in itsrotation.
- With a 4-bit disk, 360° are divided up into 16 parts, so the position of the shaft can be determined to the nearest 22.5°.

Incremental Encoders

- An incremental encoder produces a pulse for each increment of shaftrotation.
- Figure 5.5 shows the incremental encoders to determine the position and direction of rotation for each of its motors.
- For this encoder, a metal disk with two tracks of slotted holes is mounted on each motor shaft.
- An LED is mounted on one side of each track of holes, and a phototransistor is

mounted opposite the LED on the other side of the disk.

- Each phototransistor produces a train of pulses as the disk is rotated. The pulses are passed through Schmitt trigger buffers to sharpen their edges.

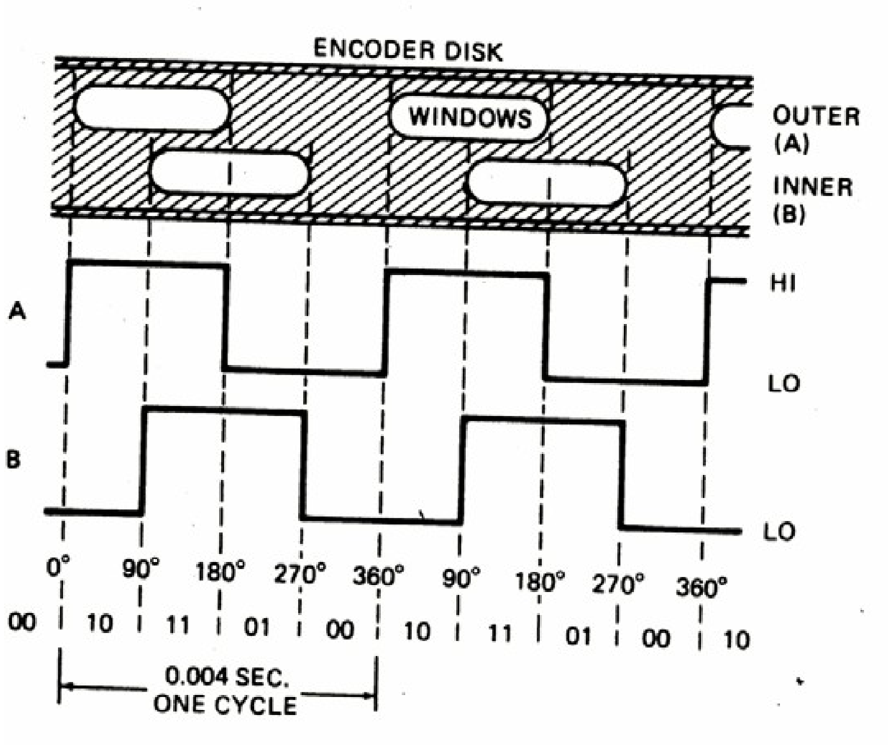


Fig 5.5 incremental shaft encoder

- The top part of Figure 5.5 shows a section of the encoder disk straightened out so it is easier to see the pulses produced as itrotates.
- The two tracks of slotted holes are 90° out of phase with each other, so as the disk is rotated, the waveforms shown at the bottom of Figure 5.5 will be produced by the phototransistors for rotation in one direction.
- Rotation in the other direction will shift the phase of the waveforms 180°, so that the B waveform leads the A waveform by 90° instead of lagging it by 90°.
- To determine the speed of rotation by simply counting the number of pulses from one detector in a fixed time interval, Each track has six holes, so six pulses will be produced for each revolution. Some simple arithmetic will give the speed in revolutions per minute (rpm).
- To determine the direction of rotation with hardware or with software.
- For the hardware approach, connect the A signal to the D input of a D flip-flop and the B signal to the clock input of the flip-flop. The rising edge of the B signal will clock the level of the A signal at that point through the flip-flop to its Q output.
- To determine the direction of rotation with software, you can detect the rising edge of the B signal on a polled or an interrupt basis and then read the logic level on the A signal. As shown in the waveforms, the A signal being high when B goes high represents rotation in one direction, and the A signal being low when B goes high represents rotation in the opposite direction.

Analog Interfacing and Industrial Control

In order to control the machines in our electronics factory, medical instruments, or automobiles with microcomputers, we need to determine the values of vari ables such as pressure, temperature, and flow. There are usually several steps in getting electrical signals which represent the values of these variables and converting the electrical signals to digital forms.

- The first step involves a *sensor,* which converts the physical pressure, temperature, or other variable to a proportional voltage or current.
- The electrical signals from most sensors are quite small, so they must be amplified and perhaps filtered.
- This is usually done with some type of operational-amplifier (op-amp) circuit.
- The final step is to convert the signal to digital form with an analog-to-digital (A/D) converter.

A MICROCOMPUTER-BASED SCALE

Overview of Smart-Scale Operation

- Figure 5.6 , shows a block diagram of our smart scale.
- A load cell converts the applied weight to a proportional electrical signal.
- This small signal is amplified and converted to a digital value which can be read in by the microprocessor and sent to the attached display.
- The user then enters the price using the keyboard, and this price per pound is shown on the display.
- When the user presses the compute key on the keyboard, the microprocessor multiplies the weight times the price per pound and displays the computed price.
- After holding the price display long enough for the user to read it, the scale goes back to reading in the weight and displaying it.

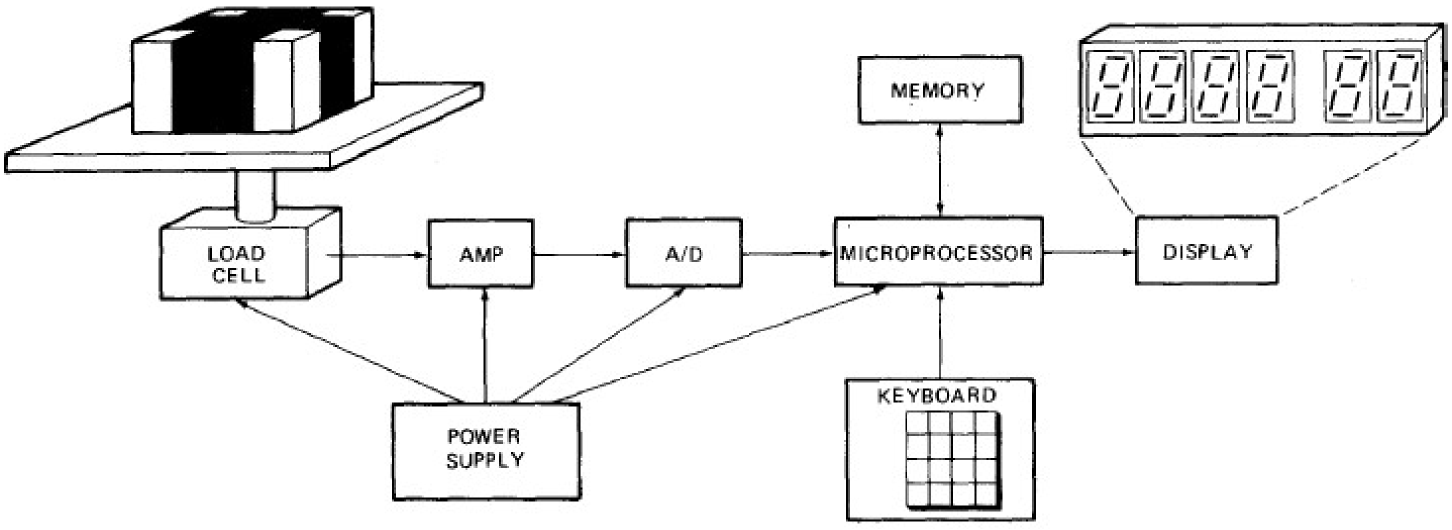


Fig 5.6 Block diagram of microcomputer-based smart scale.

Algorithm for the Smart Scale

- Figure 5.7 shows the flowchart for our smart scale.

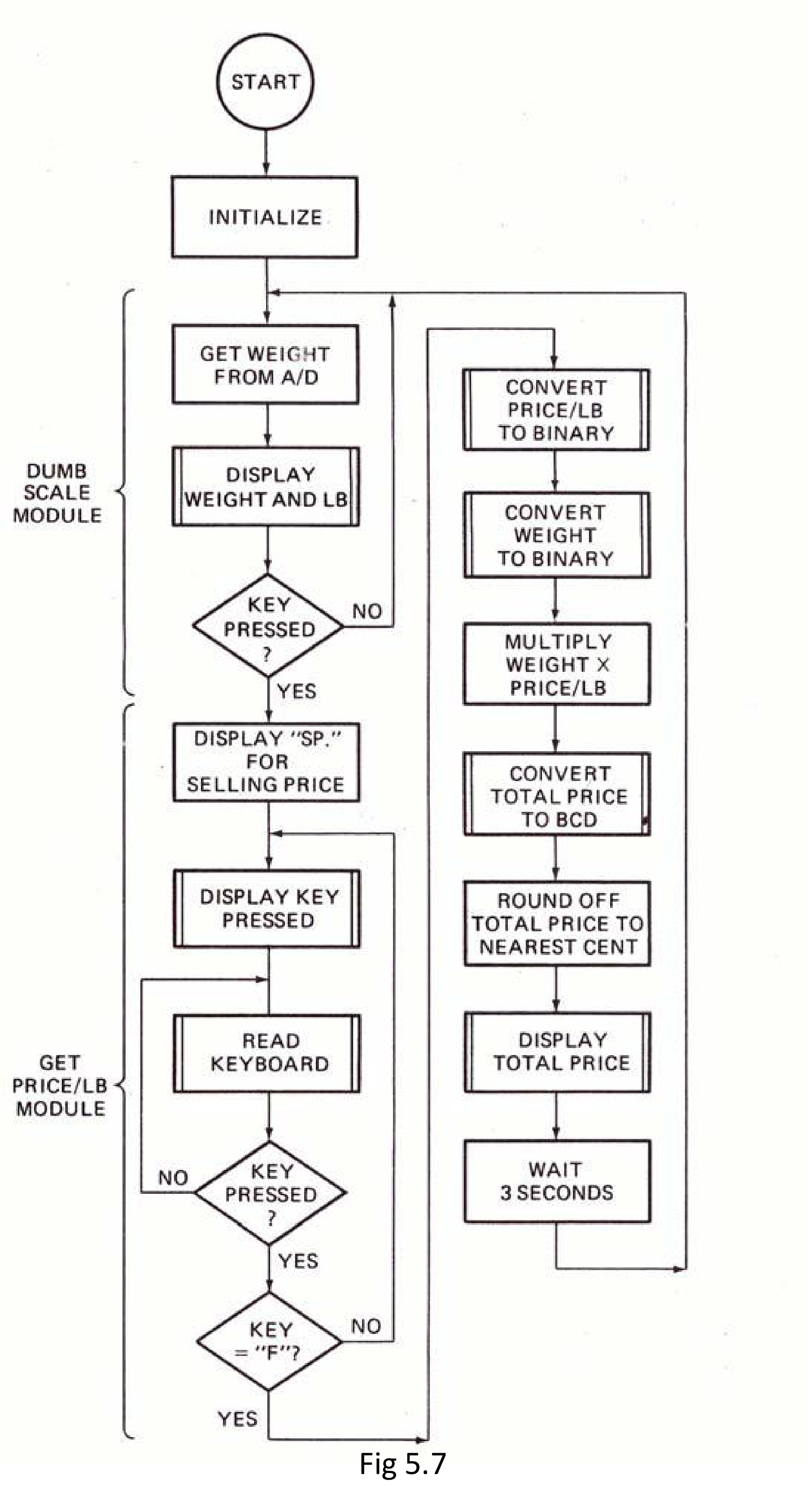


Fig 5.7

A MICROCOMPUTER-BASED INDUSTRIAL PROCESS-CONTROL SYSTEM

Overview of Industrial Process Control

- One area in which microprocessors and microcomputers have a major impact is industrial process control.
- Process control involves first measuring system variables such as motor speed, temperature, the flow of reactants, the level of a liquid in a tank, the thickness of a material, etc.
- The output of the controller then adjusts the value of each variable until it is equal to a predetermined value called a set point.

- The system controller must maintain each variable as close as possible to its set-point value, and it must compensate as quickly and accurately as possible for any change in the variable

Block Diagram

a. microcomputer-based process-control system.

- Figure 5.8 shows a block diagramof
- Data acquisition systems convert the analog signals from various sensors to digital values that can be read in and processed by themicrocomputer.

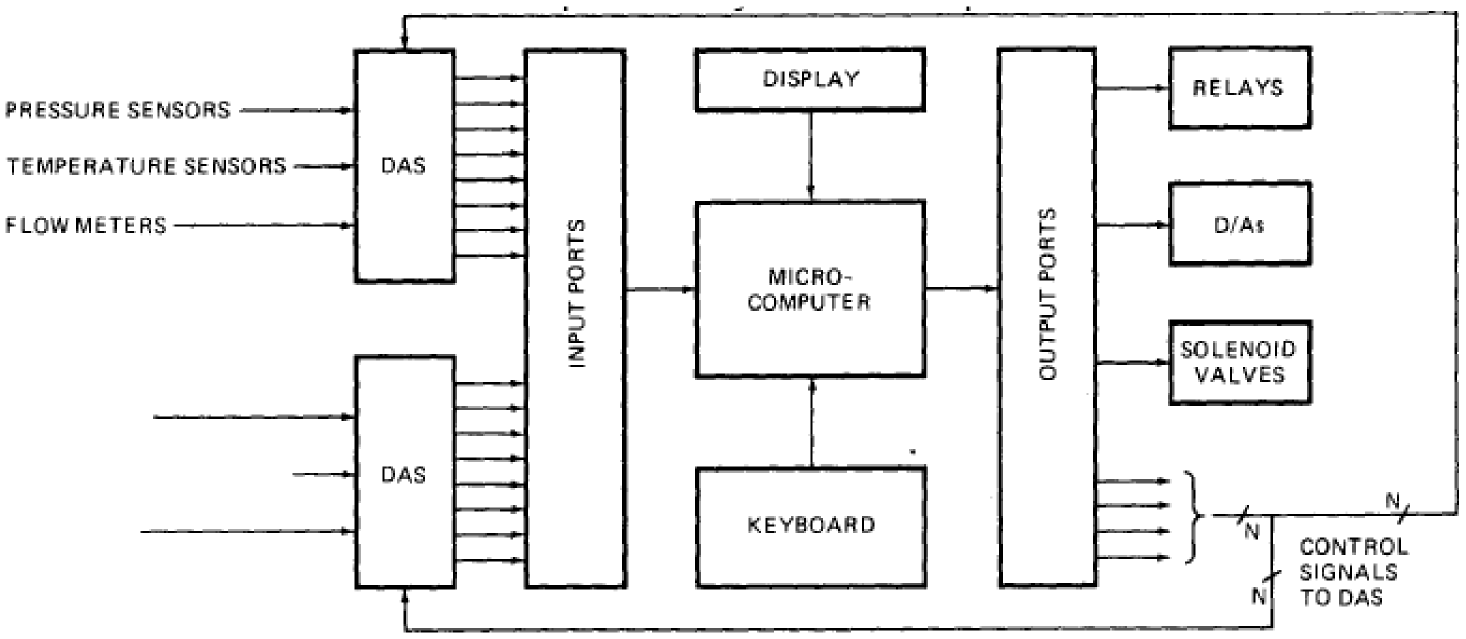


Fig 5.8 Block diagram of microcomputer-based process control system.

- A keyboard and display in the system allow the user to enter set-point values, to read the current values of process variables, and to issuecommands.
- Relays, D/A converters, solenoid valves, and other actuators are used to control process variables under program direction.
- A programmable timer in the system determines the rate at which control loops are serviced.

DIGITAL SIGNAL PROCESSING AND DIGITAL FILTERS

- The term digital signal processing, or DSP, is a very general term used to describe any system which takes samples of a signal with an A/D converter, processes the samples with a microcomputer, and outputs the computed results to a D/A converter or some other device.
- Other applications of DSP include speech recognition and synthesis systems, and contrast enhance ment of images sent back from satellites and planet probes.

Digital Filter Hardware

- The basic parts of a digital filter are an A/D converter, a microcomputer, and a D/A converter. For very low-speed applications the microprocessor used in the microcomputer can be a general-purpose device

- For many real-time applications such as digital speech processing, however, a general purpose microprocessor is not nearly fast enough. There are several reasons for this.
  - The architecture of general-purpose machines is mostly memory-based, so most operands must be fetched from memory. The memory access time the n adds to the processing time.
  - The Von Neuman architecture of general-purpose microprncessors uses the same bus for instructions and data. This means that data cannot be fetched until the code fetch is completed.
  - The multiply and add operations needed in most digital filter applications each require several clock cycles to execute in a general-purpose machine because the internal hardware is not optimized for
- To solve these and other problems, several companies have designed microprocessors which have the specific features needed for digital signal processing applications.
- The leading examples of these types of processors are the TMS320CXX family devices from Texas Instruments
- Since many computations are needed to produce each output value, the time required for these instructions severely limits the sampling rate and the maximum frequency the filter can handle.
- Sizable amounts of on-chip registers, ROM, and RAM, so data and instructions can be accessed very quickly.
- Separate buses for code words and for data words. This approach is commonly referred to as Harvard architecture.

Figure 5.9 shows a block diagram of a complete digital filter system using one of the TNtS320C25 family parts. Note that a simple analog low-pass filter is put in series with the input. Remember the sampling theorem, which stat es that the highest-frequency signal which can be digitized and reconstructed is one which contains two samples per cycle. If higher frequencies are digitized, alias frequencies will be generated when the signal is reconstructed with a D/A converter. This low-pass filter on the input helps prevent aliasing.

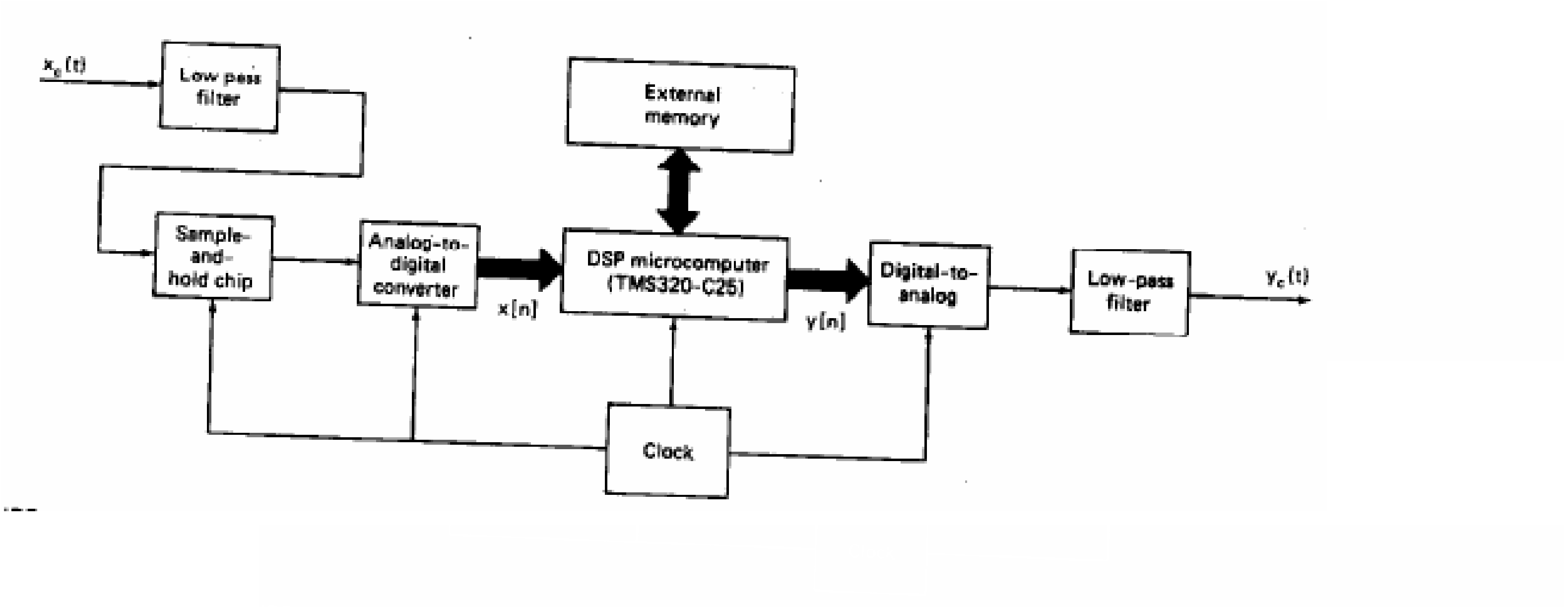


FIGURE 5.9 Block diagram of a TM5320C25 DSP-based filter.

After the anti-alias filter a sample-and-hold is used to keep the value on the input of the A/D constant during conversion. A simple low-pass analog filter is connected to the output of the D/A converter to "smooth" the output signal.

Operation of Stepper motor

Stepper motors consist of a permanent magnet rotating shaft, called the rotor, and electromagnets on the stationary portion that surrounds the motor, called the stator. Figure 1 illustrates one complete rotation of a stepper motor. At position 1, we can see that the rotor is beginning at the upper electromagnet, which is currently active (has voltage applied to it). To

move the rotor clockwise (CW), the upper electromagnet is deactivated and the right electromagnet is activated, causing the rotor to move 90 degrees CW, aligning itself with the active magnet. This process is repeated in the same manner at the south and west electromagnets until we once again reach the starting position.

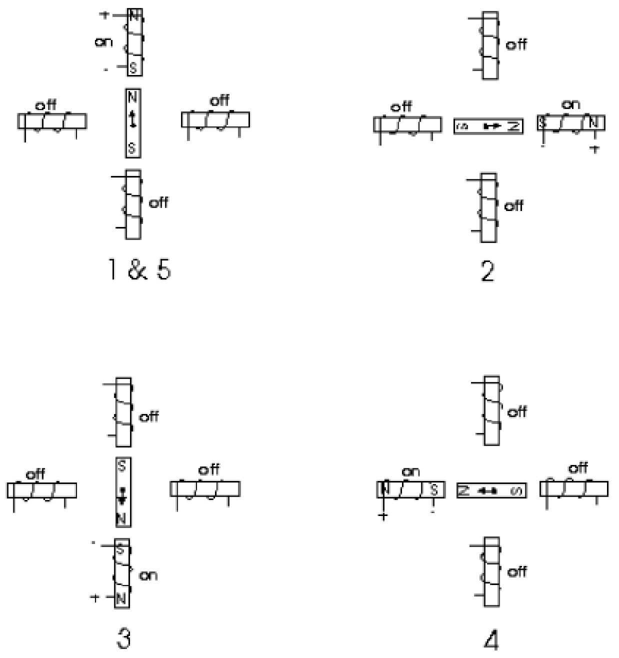


Figure 1

In the above example, we used a motor with a resolution of 90 degrees or demonstration purposes. In reality, this would not be a very practical motor for most applications. The average stepper motor's resolution -- the amount of degrees rotated per pulse -- is much higher than this. For example, a motor with a resolution of 5 degrees would move its rotor 5 degrees per step, thereby requiring 72 pulses (steps) to complete a full 360 degree rotation.

The resolution of some motors by a process known as "half-stepping". Instead of switching the next electromagnet in the rotation on one at a time, with half stepping you turn on both electromagnets, causing an equal attraction between, thereby doubling the resolution. As you can see in Figure 2, in the first position only the upper electromagnet is active, and the rotor is drawn completely to it. In position 2, both the top and right electromagnets are active, causing the rotor to position itself between the two active poles. Finally, in position 3, the top magnet is deactivated and the rotor is drawn all the way right. This process can then be repeated for the entire rotation.

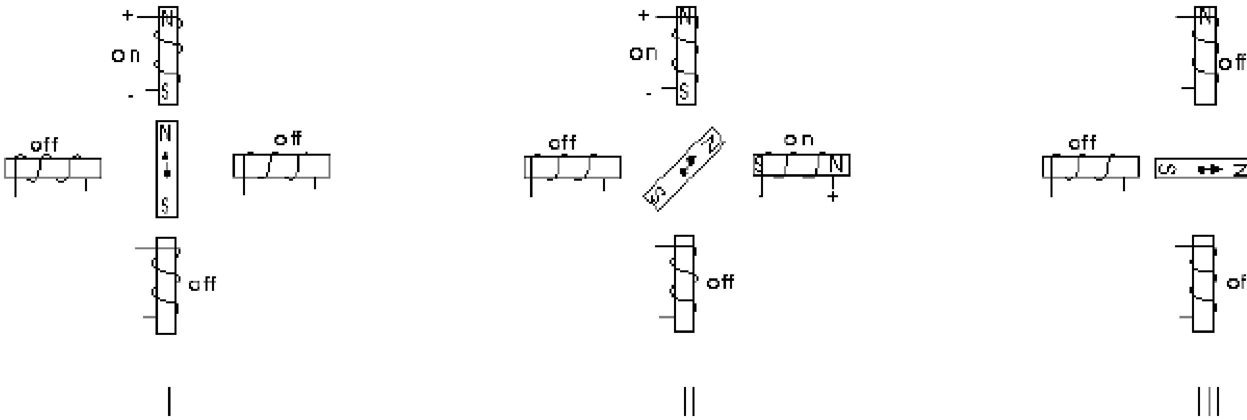


Figure 2

There are several types of stepper motors. 4-wire stepper motors contain only two electromagnets, however the operation is more complicated than those with thre e or four magnets, because the driving circuit must be able to reverse the current after each step. For our purposes, we will be using a 6-wire motor.

Unlike this example motors which rotated 90 degrees per step, real-world motors employ a series of mini-poles on the stator and rotor to increase resolution. Although this may seem to add more complexity to the process of driving the motors, the operation is identical to the simple 90 degree motor we used in our example. An example of a multi pole motor can be seen in Figure 3. In position 1, the north pole of the rotor's permanent magnet is aligned with the south pole of the stator's electromagnet. Note that multiple positions are aligned at once. In position 2, the upper electromagnet is deactivated and the next one to its immediate left is activated, causing the rotor to rotate a precise amount of degrees. In this example, after eight steps the sequence repeats.

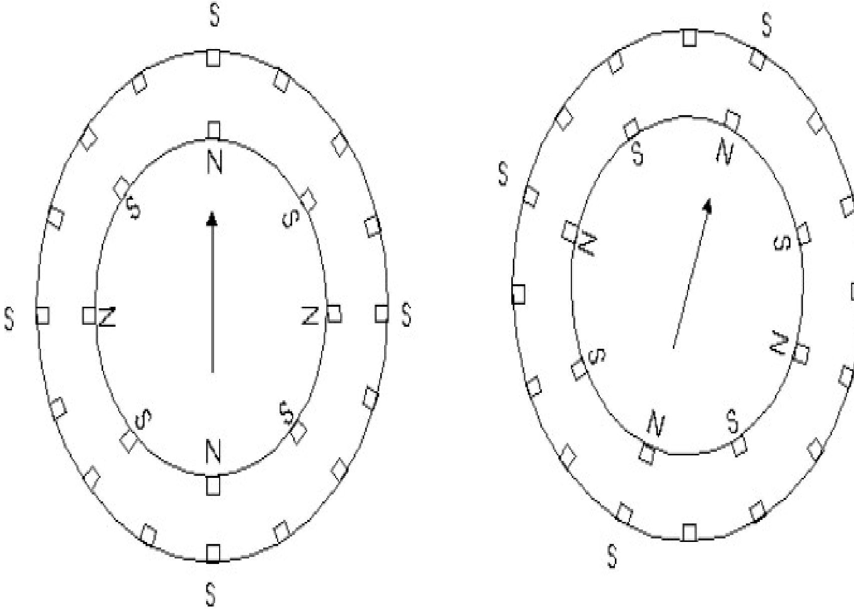


Figure 3