Unit-I

Introduction to 8086 microprocessor

- **RISC and SISC processors:**

RISC Processor

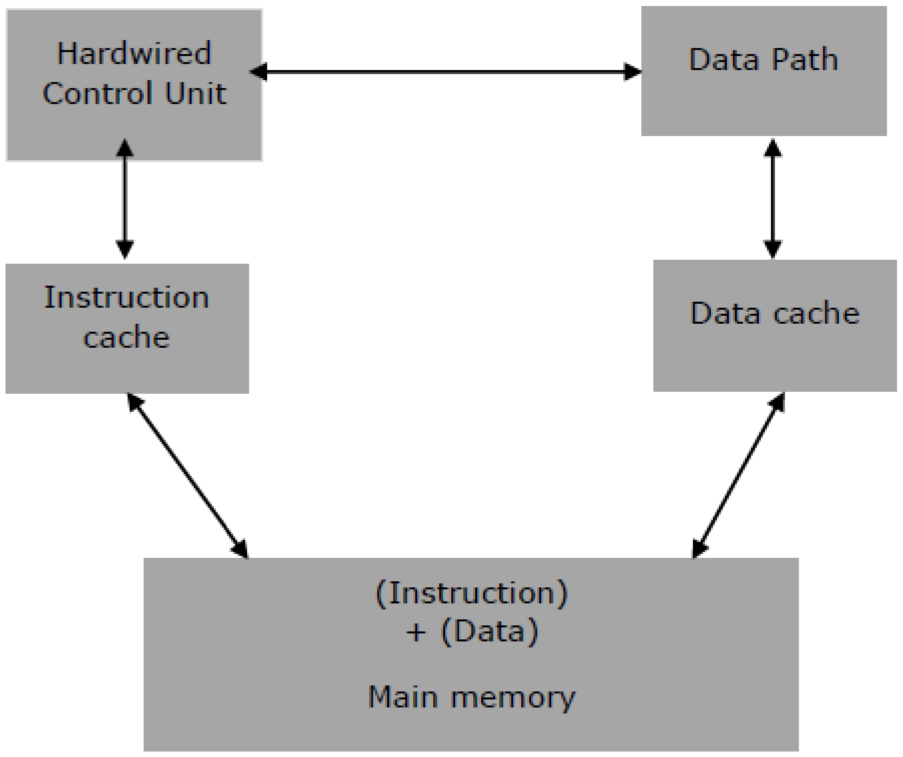
RISC stands for **Reduced Instruction Set Computer**. It is designed to reduce the execution time by simplifying the instruction set of the computer. Using RISC processors, each instruction requires only one clock cycle to execute results in uniform execution time. This reduces the efficiency as there are more lines of code, hence more RAM is needed to store the instructions. The compiler also has to work more to convert high-level language instructions into machine code.

Some of the RISC processors are −

- Power PC: 601, 604, 615, 620
- DEC Alpha: 210642, 211066, 21068, 21164
- MIPS: TS (R10000) RISC Processor
- PA-RISC: HP 7100LC

Architecture of RISC

RISC microprocessor architecture uses highly-optimized set of instructions. It is used in portable devices like Apple iPod due to its power efficiency.



Characteristics of RISC

The major characteristics of a RISC processor are as follows −

- It consists of simple instructions.
- It supports various data-type formats.
- It utilizes simple addressing modes and fixed length instructions for pipelining.
- It supports register to use in any context.
- One cycle execution time.
- "LOAD" and "STORE" instructions are used to access the memory location.
- It consists of larger number of registers.
- It consists of less number of transistors.

CISC Processor

CISC stands for **Complex Instruction Set Computer**. It is designed to minimize the number of instructions per program, ignoring the number of cycles per instruction. The emphasis is on building complex instructions directly into the hardware.

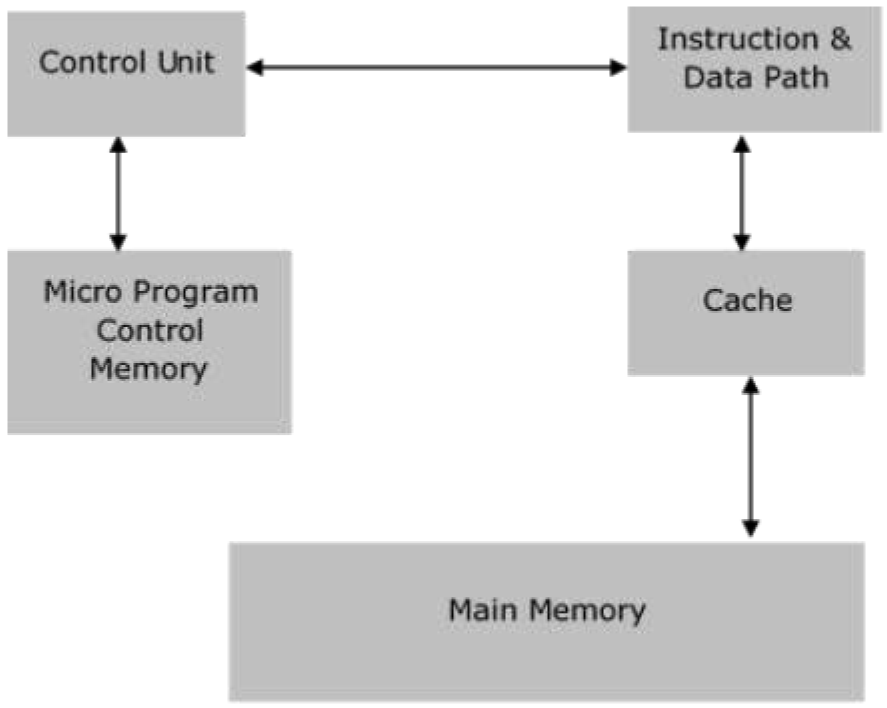
The compiler has to do very little work to translate a high-level language into assembly level language/machine code because the length of the code is relatively short, so very little RAM is required to store the instructions.

Some of the CISC Processors are −

- IBM 370/168
- VAX 11/780
- Intel 80486

Architecture of CISC

Its architecture is designed to decrease the memory cost because more storage is needed in larger programs resulting in higher memory cost. To resolve this, the number of instructions per program can be reduced by embedding the number of operations in a single instruction.



Characteristics of CISC

- Variety of addressing modes.
- Larger number of instructions.
- Variable length of instruction formats.

- *Several cycles may be required to execute one instruction.*
- *Instruction-decoding logic is complex.*
- *One instruction is required to support multiple addressing modes.*

• Architecture and pin diagram of 8086 and description of various signals:

A **Microprocessor** is an Integrated Circuit with all the functions of a CPU however, it cannot be used stand alone since unlike a microcontroller it *has no memory or peripherals.*
8086 does not have a RAM or ROM inside it. However, it has *internal registers* for storing intermediate and final results and interfaces with memory located outside it through the System Bus.
In case of 8086, it is a 16-bit **Integer processor** in a 40 pin, Dual Inline Packaged IC.

The size of the internal registers(present within the chip) indicate how much information the processor can operate on at a time (*in this case 16-bit registers*) and how it moves data around internally within the chip, sometimes also referred to as the internal data bus.
8086 provides the programmer with 14 internal registers, each 16 bits or 2 Bytes wide.

Memory segmentation:

- To increase execution speed and fetching speed, 8086 segments the memory.
- It's 20 bit address bus can address 1MB of memory, it segments it into 4 64kB segments.
- 8086 works only with four 64KB segments within the whole 1MB memory.

The internal architecture of Intel 8086 is divided into 2 units: The Bus Interface Unit (BIU), and The Execution Unit (EU). These are explained as following below.

1. The Bus Interface Unit (BIU):

It provides the interface of 8086 to external memory and I/O devices via the System Bus. It performs various machine cycles such as memory read, I/O read etc. to transfer data between memory and I/O devices.

BIU performs the following functions-

- It generates the 20 bit physical address for memory access.
- It fetches instructions from the memory.
- It transfers data to and from the memory and I/O.
- Maintains the 6 byte prefetch instruction queue(**supports pipelining**).

BIU mainly contains the **4 Segment registers**, the **Instruction Pointer**, a prefetch queue and an **Address Generation Circuit**.

Instruction Pointer (IP):

- It is a 16 bit register. It holds offset of the next instructions in the Code Segment.
- IP is incremented after every instruction byte is fetched.
- IP gets a new value whenever a branch instruction occurs.
- CS is multiplied by 10H to give the 20 bit physical address of the Code Segment.
- Address of the next instruction is calculated as CS x 10H + IP.

Example:

CS = 4321H IP = 1000H

then CS x 10H = 43210H + offset = 44210H

This is the address of the instruction.

Code Segment register:

CS holds the base address for the Code Segment. All programs are stored in the Code Segment and accessed via the IP.

Data Segment register:

DS holds the base address for the Data Segment.

Stack Segment register:

SS holds the base address for the Stack Segment.

Extra Segment register:

ES holds the base address for the Extra Segment.

Address Generation Circuit:

- The BIU has a Physical Address Generation Circuit.
- It generates the 20 bit physical address using Segment and Offset addresses using the formula:
- Physical Address
  = Segment Address x 10H + Offset Address

6 Byte Pre-fetch Queue:

- It is a 6 byte queue (FIFO).
- Fetching the next instruction (by BIU from CS) while executing the current instruction is called pipelining.
- Gets flushed whenever a branch instruction occurs.

2. The Execution Unit (EU):

The main components of the EU are General purpose registers, the ALU, Special purpose registers, Instruction Register and Instruction Decoder and the Flag/Status Register.

1. Fetches instructions from the Queue in BIU, decodes and executes arithmetic and logic operations using the ALU.
2. Sends control signals for internal data transfer operations within the

microprocessor.

3. Sends request signals to the BIU to access the external module.
4. It operates with respect to T-states (clock cycles) and not machine cycles.

8086 has four 16 bit general purpose registers AX, BX, CX and DX. Store intermediate values during execution. Each of these have two 8 bit parts (higher and lower).

- **AX register:**
  It holds operands and results during multiplication and division operations. Also an accumulator during String operations.
- **BX register:**
  It holds the memory address (offset address) in indirect addressing modes.
- **CX register:**
  It holds count for instructions like loop, rotate, shift and string operations.
- **DX register:**
  It is used with AX to hold 32 bit values during multiplication and division.

Arithmetic Logic Unit (16 bit):
Performs **8 and 16 bit** arithmetic and logic operations.

Special purpose registers (16-bit):

- **Stack Pointer:**
  Points to Stack top. Stack is in Stack Segment, used during instructions like PUSH, POP, CALL, RET etc.
- **Base Pointer:**
  BP can hold offset address of any location in the stack segment. It is used to access random locations of the stack.
- **Source Index:**
  It holds offset address in Data Segment during string operations.
- **Destination Index:**
  It holds offset address in Extra Segment during string operations.

Instruction Register and Instruction Decoder:
The EU fetches an opcode from the queue into the instruction register. The instruction decoder decodes it and sends the information to the control circuit for execution.

Flag/Status register (16 bits):
It has 9 flags that help change or recognize the state of the microprocessor.

6 Status flags:

1. carry flag(CF)
2. parity flag(PF)
3. auxiliary carry flag(AF)
4. zero flag(Z)
5. sign flag(S)
6. overflow flag (O)

Status flags are updated after every arithmetic and logic operation.

3 Control flags:

1. trap flag(TF)
2. interrupt flag(IF)
3. direction flag(DF)

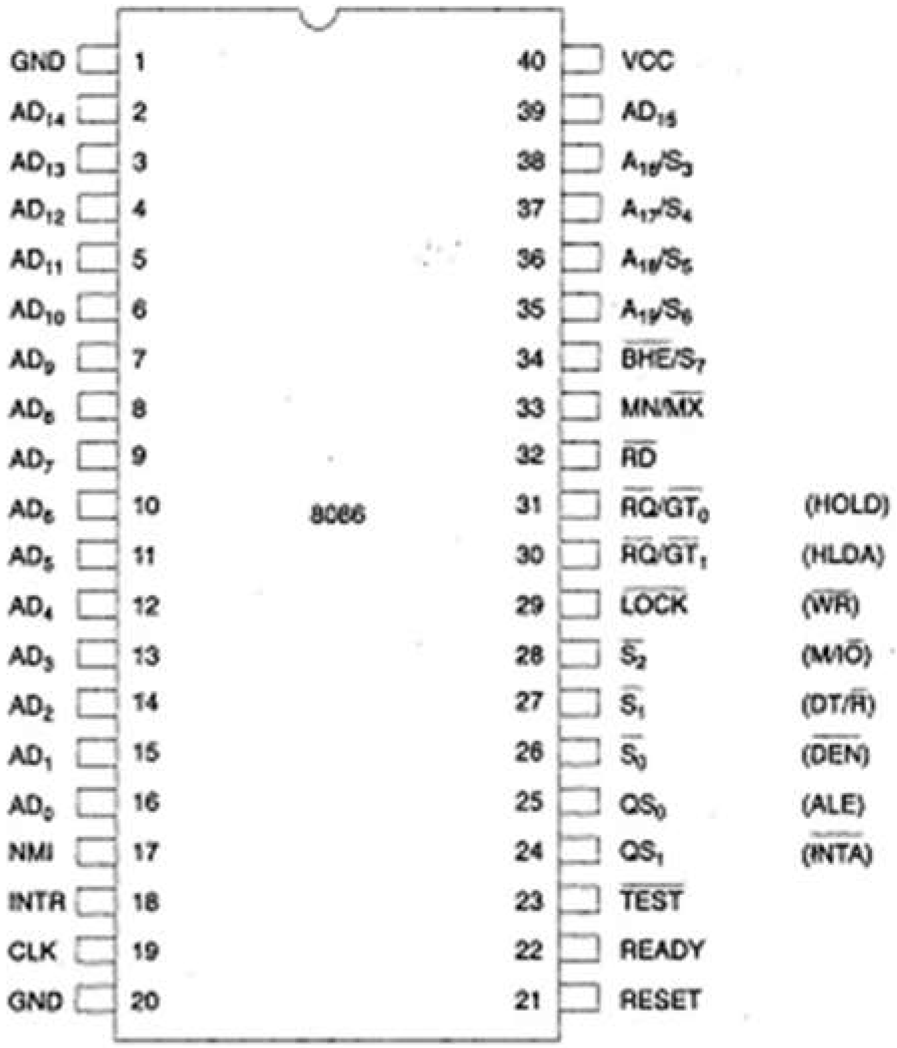
These flags can be set or reset using control instructions like CLC, STC, CLD, STD, CLI, STI, etc.

The Control flags are used to control certain operations.

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 and 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 AD8AD15 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 22. 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-aversa.

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 and 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 −

QS_0	QS_1	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 −

S_2	S_1	S_0	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 and 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.

• Register organization of 8086: Register organization of 8086

General 16-bit registers
The registers AX, BX, CX, and DX are the general 16-bit registers.

AX Register: Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16- bit register AX. AL in this case contains the
low-order byte of the word, and AH contains the high-order byte. Accumulator can be
used for I/O operations, rotate and string manipulation.
BX Register: This register is mainly used as a base register. It holds the starting base
location of a memory region within a data segment. It is used as offset storage for forming physical address in case of certain addressing mode.
CX Register: It is used as default counter or count register in case of string and loop
instructions.
DX Register: Data register can be used as a port number in I/O operations and implicit
operand or destination in case of few instructions. In integer 32-bit multiply and divide
instruction the DX register contains high-order word of the initial or resulting number.
Segment registers:
To complete 1Mbyte memory is divided into 16 logical segments. The complete 1Mbyte memory segmentation is as shown in fig 1.5. Each segment contains 64Kbyte of
memory. There are four segment registers.
Code segment (CS) is a 16-bit register containing address of 64 KB segment with processor instructions. The processor uses CS segment for all accesses to instructions
referenced by instruction pointer (IP) register. CS register cannot be changed directly.
The CS register is automatically updated during far jump, far call and far return instructions. It is used for addressing a memory location in the code segment of the
memory, where the executable program is stored.
Stack segment (SS) is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the

stack
pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register
can be changed directly using POP instruction. It is used for addressing stack segment of memory. The stack segment is that segment of memory, which is used to store stack data.
Data segment (DS) is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general
registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment. DS register can be changed directly using POP and LDS instructions. It points to the data segment memory where the data is resided.
Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It also refers to segment which essentially is another data segment of the memory. It also contains data.
Pointers and index registers.
The pointers contain within the particular segments. The pointers IP, BP, SP usually contain offsets within the code, data and stack segments respectively
Stack Pointer (SP) is a 16-bit register pointing to program stack in stack segment.
Base Pointer (BP) is a 16-bit register pointing to data in stack segment. BP register is
usually used for based, based indexed or register indirect addressing.
Source Index (SI) is a 16-bit register. SI is used for indexed, based indexed and register
indirect addressing, as well as a source data addresses in string manipulation instructions.
Destination Index (DI) is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation
instructions.

- ***Description of address computations and memory segmentations:***

Segmentation *is the process in which the main memory of the computer is divided into different segments and each segment has its own base address. It is basically used to enhance the speed of execution of the computer system, so that processor is able to fetch and execute the data from the memory easily and fast.*
Need for Segmentation –
The Bus Interface Unit (BIU) contains four 16 bit special purpose registers

(mentioned below) called as Segment Registers.

- **Code segment register (CS):** is used for addressing memory location in the code segment of the memory, where the executable program is stored.
- **Data segment register (DS):** points to the data segment of the memory where the data is stored.
- **Extra Segment Register (ES):** also refers to a segment in the memory which is another data segment in the memory.
- **Stack Segment Register (SS):** is used for addressing stack segment of the memory. The stack segment is that segment of memory which is used to store stack data.

The number of address lines in 8086 is 20, 8086 BIU will send 20bit address, so as to access one of the 1MB memory locations. The four segment registers actually contain the upper 16 bits of the starting addresses of the four memory segments of 64 KB each with which the 8086 is working at that instant of time. A segment is a logical unit of memory that may be up to 64 kilobytes long. Each segment is made up of contiguous memory locations. It is independent, separately addressable unit. Starting address will always be changing. It will not be fixed.

Note that the 8086 does not work the whole 1MB memory at any given time. However it works only with four 64KB segments within the whole 1MB memory.

Types Of Segmentation –

1. **Overlapping Segment –** A segment starts at a particular address and its maximum size can go up to 64kilobytes. But if another segment starts along this 64kilobytes location of the first segment, then the two are said to be *Overlapping Segment*.
2. **Non-Overlapped Segment –** A segment starts at a particular address and its maximum size can go up to 64kilobytes. But if another segment starts before this 64kilobytes location of the first segment, then the two segments are said to be *Non-Overlapped Segment*.

Advantages of the Segmentation The main advantages of segmentation are as follows:

- It provides a powerful memory management mechanism.
- Data related or stack related operations can be performed in different segments.
- Code related operation can be done in separate code segments.
- It allows to processes to easily share data.
- It allows to extend the address ability of the processor, i.e. segmentation allows the use of 16 bit registers to give an addressing capability of 1 Megabytes. Without segmentation, it would require 20 bit registers.
- It is possible to enhance the memory size of code data or stack segments beyond 64 KB by allotting more than one segment for each area.

- Segment override:

As we already know that the effective address is calculated by appending the **segment registers** value and adding up the value of the respective offset. But what if we want to choose some other offset than the assigned one.

This freedom is provided to us in 8086 microprocessor through the concept of **Segment override prefix**.

The **Segment Override Prefix** says that if we want to use some other segment register than the default segment for a particular code, then it is possible. It can simply be one by mentioning the segment that is to be used before the address location or the offset register containing that address. By doing so, the machine, i.e. the 8086 microprocessor, while calculating the effective address will consider the mentioned segment for calculating the effective address rather than opting for the default one.

The syntax of doing so, as mentioned earlier is by mentioning the segment just before the address location and proceeded by a colon. The following abbreviations for each segment register are used for this purpose:

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

Let us take the following examples to further understand this concept:

```
MOV AX , [BX]
```

This is a normal instruction without any segment overriding. Hence the effective address will be calculated by using the default segment itself. Therefore, the effective address for the above-mentioned instruction:

Effective address = DS X 10H + content of BX register

```
MOV AX, SS : [BX]
```

Here, in this case, the Stack segment register is used as a prefix for the offset **BX**. So, instead of **DS**, which is the default segment register for **BX**, the **SS** will be used for finding the effective address location. Therefore, the effective address in the above-mentioned equation will be:

Effective address = SS X 10H + content of BX register

```
MOV BX, SS:[1002H]
```

Here, in this case, the address from where the data is to be fetched is directly mentioned in the instruction. By default, such cases take the Data Segment (**DS**) as its offset. But as the **SS** is the mentioned register in the given instruction, then it will be used here. Therefore, the effective address for the given instruction is:

Effective Address = SS X 10H + 1002H

- Instruction pipelining:

Most of the digital computers with complex instructions require instruction pipeline to carry out operations like fetch, decode and execute instructions.

In general, the computer needs to process each instruction with the following sequence of steps.

1. Fetch instruction from memory.
2. Decode the instruction.
3. Calculate the effective address.
4. Fetch the operands from memory.
5. Execute the instruction.
6. Store the result in the proper place.

Each step is executed in a particular segment, and there are times when different segments may take different times to operate on the incoming information. Moreover, there are times when two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory.

The organization of an instruction pipeline will be more efficient if the instruction cycle is divided into segments of equal duration. One of the most common examples of this type of organization is a **Four-segment instruction pipeline.**

A **four-segment instruction** pipeline combines two or more different segments and makes it as a single one. For instance, the decoding of the instruction can be combined with the calculation of the effective address into one segment.

Segment 1:

The instruction fetch segment can be implemented using first in, first out (FIFO) buffer.

Segment 2:

The instruction fetched from memory is decoded in the second segment, and eventually, the effective address is calculated in a separate arithmetic circuit.

Segment 3:

An operand from memory is fetched in the third segment.

Segment 4:

The instructions are finally executed in the last segment of the pipeline organization

- *The HOLD pin is checked at the end of each bus cycle.*
- *If it is received active by the processor before T_4 of the previous cycle of during T_1 state of the current cycles, the CPU activates HLDA in the next clock cycle and for the succeeding bus cycles; the bus will be given to another requesting master.*
-
-

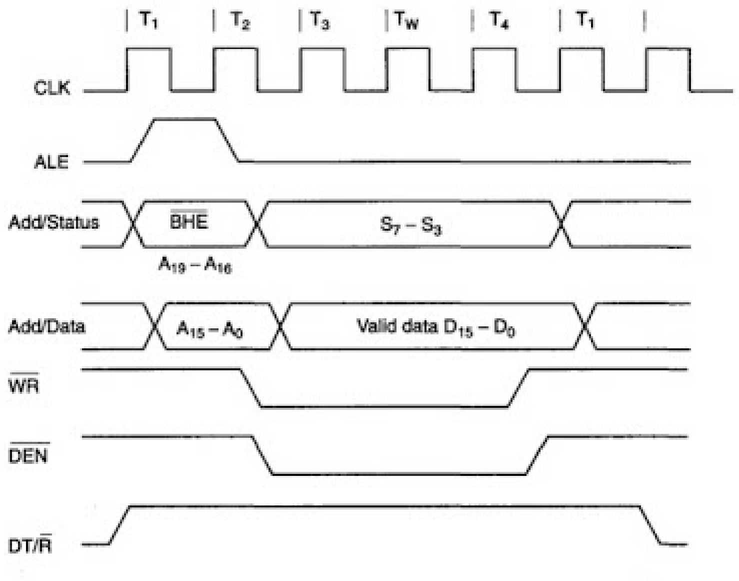


Fig.1.9(b) **Write Cycle Timing Diagram for Minimum Operation**

-
-
-
- *The control of the bus is not regained by the processor until the requesting master does not drop the HLDA pin low.*
-
-

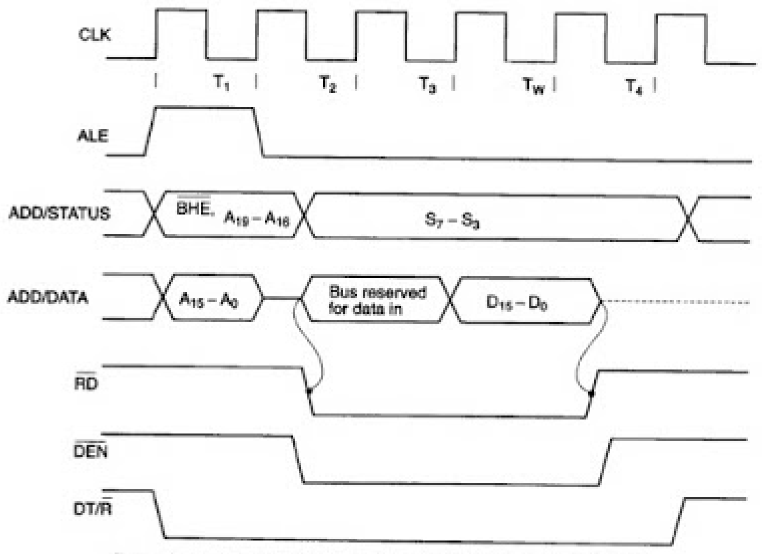
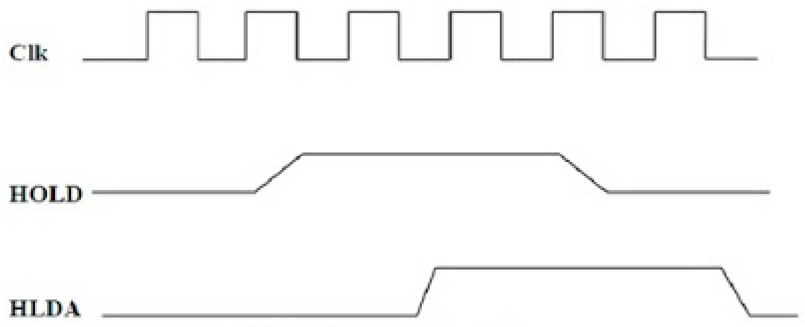


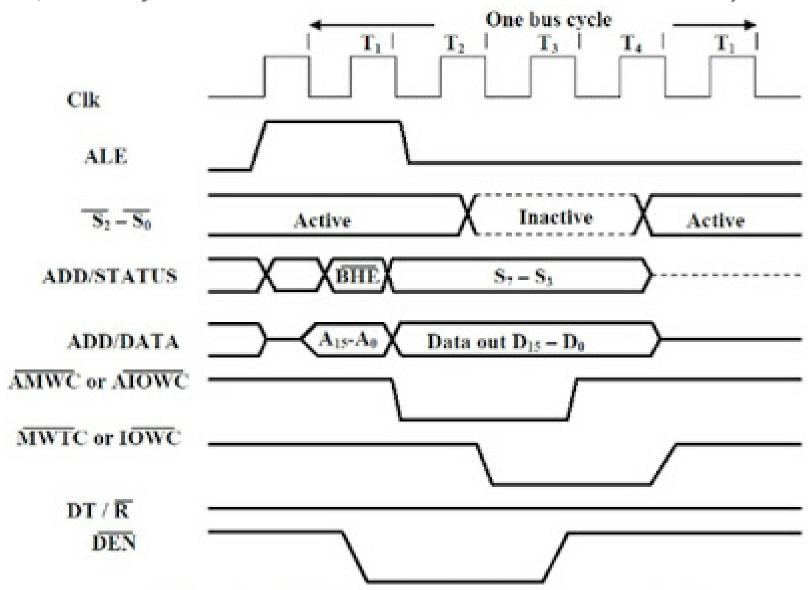
Fig.1.9(a) *Read Cycle Timing Diagram for Minimum Mode*

- *When the request is dropped by the requesting master, the HLDA is dropped by the processor at the trailing edge of the next clock, as shown in Fig*



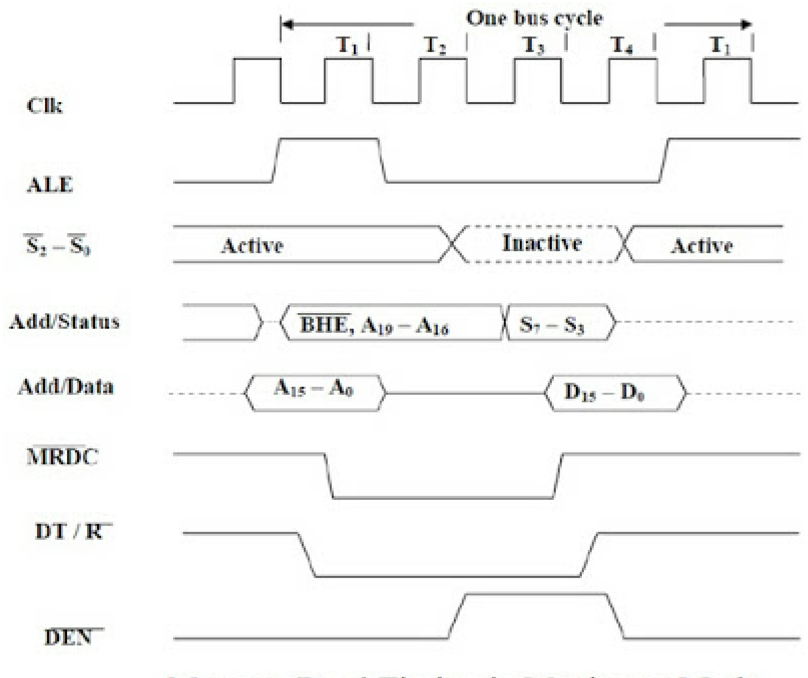
Bus Request and
Bus Grant Timings in Minimum Mode System

- *The other conditions have already been discussed in the signal description section for the HOLD and HLDA signals.*
- ***TIMING DIAGRAM of MAXIMUM MODE 8086***



- Memory Write Timing in Maximum mode.

-
-
-



- Memory Read Timing in Maximum Mode

- ***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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

- 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.

Unit-2

Instruction set of 8086:

Instruction Set of 8086

Instructions are classified on the basis of functions they perform. They are categorized into the following main types:

Data Transfer instruction

All the instructions which perform data movement come under this category. The source data may be a register, memory location, port etc. the destination may be a register, memory location or port. The following instructions come under this category:

Instruction	Description
MOV	Moves data from register to register,

	register to memory, memory to register, memory to accumulator, accumulator to memory, etc.
LDS	Loads a word from the specified memory locations into specified register. It also loads a word from the next two memory locations into DS register.
LES	Loads a word from the specified memory locations into the specified register. It also loads a word from next two memory locations into ES register.
LEA	Loads offset address into the specified register.
LAHF	Loads low order 8-bits of the flag register into AH register.
SAHF	Stores the content of AH register into low order bits of the flags register.
XLAT/XLATB	Reads a byte from the lookup table.
XCHG	Exchanges the contents of the 16-bit or 8-bit specified register with the contents of AX register, specified register or memory locations.
PUSH	Pushes (sends, writes or moves) the content of a specified register or memory location(s) onto the top of the stack.
POP	Pops (reads) two bytes from the top of the stack and keeps them in a specified register, or memory location(s).
POPF	Pops (reads) two bytes from the top of the stack and keeps them in the flag register.
IN	Transfers data from a port to the accumulator or AX, DX or AL register.
OUT	Transfers data from accumulator or AL or AX register to an I/O port identified by the second byte of the instruction.

Arithmetic Instructions

Instructions of this group perform addition, subtraction, multiplication, division, increment, decrement, comparison, ASCII and decimal adjustment etc.

The following instructions come under this category:

Instruction	Description
ADD	Adds data to the accumulator i.e. AL or AX register or memory locations.
ADC	Adds specified operands and the carry status (i.e. carry of the previous stage).
SUB	Subtract immediate data from accumulator, memory or register.
SBB	Subtract immediate data with borrow from accumulator, memory or register.
MUL	Unsigned 8-bit or 16-bit multiplication.
IMUL	Signed 8-bit or 16-bit multiplication.
DIV	Unsigned 8-bit or 16-bit division.
IDIV	Signed 8-bit or 16-bit division.
INC	Increment Register or memory by 1.
DEC	Decrement register or memory by 1.
DAA	**Decimal Adjust after BCD Addition:** When two BCD numbers are added, the DAA is used after ADD or ADC instruction to get correct answer in BCD.
DAS	**Decimal Adjust after BCD Subtraction:** When two BCD numbers are added, the DAS is used after SUB or SBB instruction to get correct answer in BCD.
AAA	**ASCII Adjust for Addition:** When ASCII codes of two decimal digits are added, the AAA is used after addition to get correct answer in unpacked BCD.
AAD	**Adjust AX Register for Division:** It converts two unpacked BCD digits in AX to the equivalent binary number. This adjustment is done before dividing two unpacked BCD digits in AX by an unpacked BCD byte.

AAM	**Adjust result of BCD Multiplication:** This instruction is used after the multiplication of two unpacked BCD.
AAS	**ASCII Adjust for Subtraction:** This instruction is used to get the correct result in unpacked BCD after the subtraction of the ASCII code of a number from ASCII code another number.
CBW	Convert signed Byte to signed Word.
CWD	Convert signed Word to signed Doubleword.
NEG	Obtains 2's complement (i.e. negative) of the content of an 8-bit or 16-bit specified register or memory location(s).
CMP	Compare Immediate data, register or memory with accumulator, register or memory location(s).

Logical Instructions

Instruction of this group perform logical AND, OR, XOR, NOT and TEST operations.
The following instructions come under this category:

Instruction	Description
AND	Performs bit by bit logical AND operation of two operands and places the result in the specified destination.
OR	Performs bit by bit logical OR operation of two operands and places the result in the specified destination.
XOR	Performs bit by bit logical XOR operation of two operands and places the result in the specified destination.
NOT	Takes one's complement of the content of a specified register or memory location(s).
TEST	Perform logical AND operation of a specified operand with another specified operand.

Rotate Instructions

The following instructions come under this category:

Instruction	*Description*
RCL	*Rotate all bits of the operand left by specified number of bits through carry flag.*
RCR	*Rotate all bits of the operand right by specified number of bits through carry flag.*
ROL	*Rotate all bits of the operand left by specified number of bits.*
ROR	*Rotate all bits of the operand right by specified number of bits.*

Shift Instructions

The following instructions come under this category:

Instruction	*Description*
SAL or SHL	*Shifts each bit of operand left by specified number of bits and put zero in LSB position.*
SAR	*Shift each bit of any operand right by specified number of bits. Copy old MSB into new MSB.*
SHR	*Shift each bit of operand right by specified number of bits and put zero in MSB position.*

Branch Instructions

It is also called program execution transfer instruction. Instructions of this group transfer program execution from the normal sequence of instructions to the specified destination or target. The following instructions come under this category:

Instruction	*Description*
JA or JNBE	*Jump if above, not below, or equal i.e. when CF and ZF = 0*
JAE/JNB/JNC	*Jump if above, not below, equal or no carry i.e. when CF = 0*
JB/JNAE/JC	*Jump if below, not above, equal or*

	carry i.e. when CF = 0
JBE/JNA	Jump if below, not above, or equal i.e. when CF and ZF = 1
JCXZ	Jump if CX register = 0
JE/JZ	Jump if zero or equal i.e. when ZF = 1
JG/JNLE	Jump if greater, not less or equal i.e. when ZF = 0 and CF = OF
JGE/JNL	Jump if greater, not less or equal i.e. when SF = OF
JL/JNGE	Jump if less, not greater than or equal i.e. when SF ≠ OF
JLE/JNG	Jump if less, equal or not greater i.e. when ZF = 1 and SF ≠ OF
JMP	Causes the program execution to jump unconditionally to the memory address or label given in the instruction.
CALL	Calls a procedure whose address is given in the instruction and saves their return address to the stack.
RET	Returns program execution from a procedure (subroutine) to the next instruction or main program.
IRET	Returns program execution from an interrupt service procedure (subroutine) to the main program.
INT	Used to generate software interrupt at the desired point in a program.
INTO	Software interrupts to indicate overflow after arithmetic operation.
LOOP	Jump to defined label until CX = 0.
LOOPZ/LOOPE	Decrement CX register and jump if CX ≠ 0 and ZF = 1.
LOOPNZ/LOOPNE	Decrement CX register and jump if CX ≠ 0 and ZF = 0.

Here, CF = Carry Flag
ZF = Zero Flag
OF = Overflow Flag
SF = Sign Flag
CX = Register

Flag Manipulation and Processor Control Instructions

Instructions of this instruction set are related to flag manipulation and machine control. The following instructions come under this category:

Instruction	Description
CLC	**Clear Carry Flag**: This instruction resets the carry flag CF to 0.
CLD	**Clear Direction Flag**: This instruction resets the direction flag DF to 0.
CLI	**Clear Interrupt Flag**: This instruction resets the interrupt flag IF to 0.
CMC	This instruction take complement of carry flag CF.
STC	Set carry flag CF to 1.
STD	Set direction flag to 1.
STI	Set interrupt flag IF to 1.
HLT	Halt processing. It stops program execution.
NOP	Performs no operation.
ESC	**Escape**: makes bus free for external master like a coprocessor or peripheral device.
WAIT	When WAIT instruction is executed, the processor enters an idle state in which the processor does no processing.
LOCK	It is a prefix instruction. It makes the LOCK pin low till the execution of the next instruction.

String Instructions

String is series of bytes or series of words stored in sequential memory locations. The 8086 provides some instructions which handle string operations such as string movement, comparison, scan, load and store.

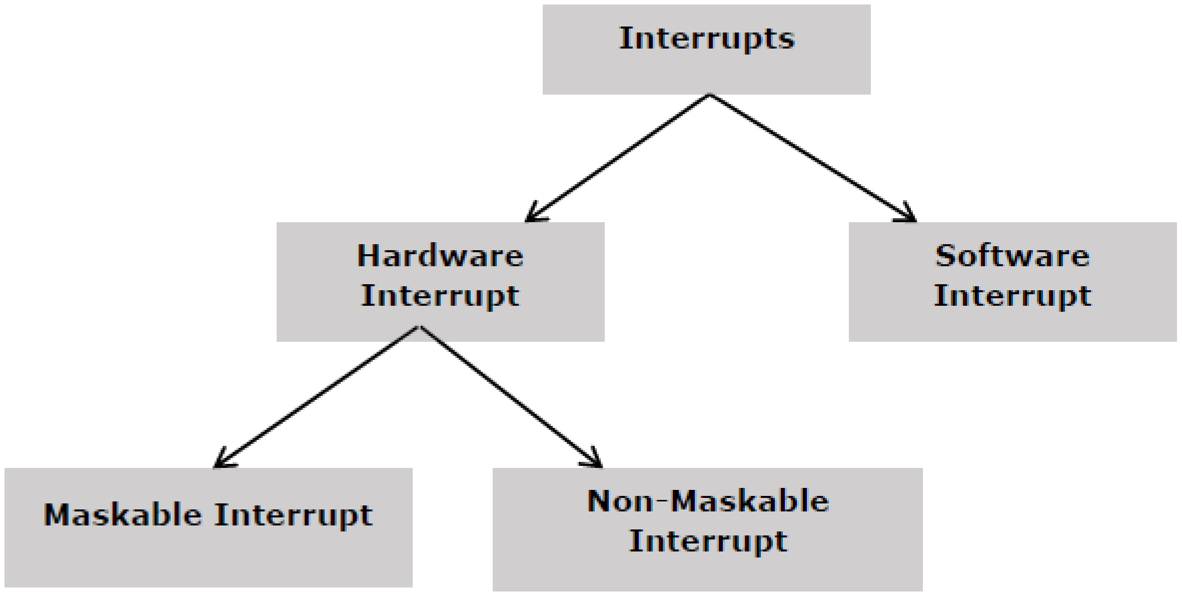
The following instructions come under this category:

Instruction	Description
MOVS/MOVSB/MOVSW	Moves 8-bit or 16-bit data from the memory location(s) addressed by SI register to the memory location addressed by DI register.
CMPS/CMPSB/CMPSW	Compares the content of memory location addressed by DI register with the content of memory location addressed by SI register.
SCAS/SCASB/SCASW	Compares the content of accumulator with the content of memory location addressed by DI register in the extra segment ES.
LODS/LODSB/LODSW	Loads 8-bit or 16-bit data from memory location addressed by SI register into AL or AX register.
STOS/STOSB/STOSW	Stores 8-bit or 16-bit data from AL or AX register in the memory location addressed by DI register.
REP	Repeats the given instruction until CX ≠ 0
REPE/ REPZ	Repeats the given instruction till CX ≠ 0 and ZF = 1
REPNE/REPNZ	Repeats the given instruction till CX ≠ 0 and ZF = 0

Interrupts of 8086 :

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 × 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' × 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 3×4 = 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 = 00010

CS is loaded from the contents of the next word location.

Assembly language program using 8086:

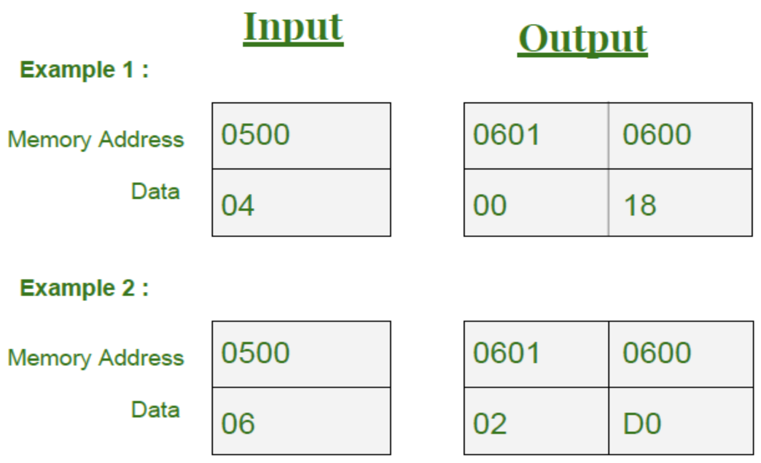
8086 program to find the factorial of a number

Prerequisite – 8085 program to find the factorial of a number
Problem – Write an assembly language program for calculating the factorial of a number using 8086 microprocessor
Examples –

```
Input : 04H
Output : 18H
as In Decimal : 4*3*2*1 = 24
   In Hexadecimal : 24 = 18H

Input : 06H
Output : 02D0H
as In Decimal : 6*5*4*3*2*1 = 720
   In Hexadecimal : 720 = 02D0H
```

Assumptions –
Starting address of program: 0400
Input memory location: 0500
Output memory location: 0600 and 0601

Important –
If the Given Number is a 16-bit number, the AX register is automatically used as the second parameter and the product is stored in the DX:AX register pair. This means that the DX register holds the high part and the AX register holds the low part of a 32-bit number.



Input **Output**

Example 1 :

	Input	Output	
Memory Address	0500	0601	0600
Data	04	00	18

Example 2 :

	Input	Output	
Memory Address	0500	0601	0600
Data	06	02	D0

In 8086 microprocessor, user have direct instruction (MUL) to multiply two numbers, so we don't have to add Multiplicand by Multiplier times like in 8085

Advantage of 8086 over 8085 (In case of Multiply):

Don't have to write a bulky code as 8086 has a small code
Easy to remember
Already have multiplication Instruction

Algorithm –

Input the Number whose factorial is to be find and Store that Number in CX Register (Condition for LOOP Instruction)
Insert 0001 in AX(Condition for MUL Instruction) and 0000 in DX
Multiply CX with AX until CX become Zero(0) using LOOP Instruction
Copy the content of AX to memory location 0600
Copy the content of DX to memory location 0601
Stop Execution

Program –

ADDRESS	MNEMONICS	COMMENTS
0400	MOV CX, [0500]	CX <- [0500]
0404	MOV AX, 0001	AX <- 0001
0407	MOV DX, 0000	DX <- 0000

040A	MUL CX	DX:AX <- AX * CX
040C	LOOP 040A	Go To [040A] till CX->00
0410	MOV [0600], AX	[0600]<- AX
0414	MOV [0601], DX	[0601]<- DX
0418	HLT	Stop Execution

Explanation –
MOV CX, [0500] loads 0500 Memory location content to CX Register
MOV AX, 0001 loads AX register with 0001
MOV DX, 0000 loads DX register with 0000
MUL CX multiply AX with CX and store result in DX:AX pair
LOOP 040A runs loop till CX not equal to Zero
MOV [0600], AX store AX register content to memory location 0600
MOV [0601], DX store DX register content to memory location 0601
HLT stops the execution of program

INTEL 8051 MICRCONTROLLER

Introduction :

A decade back the process and control operations were totally implemented by the Microprocessors only. But now a days the situation is totally changed and it is occupied by the new devices called Microcontroller. The development is so drastic that we can't find any electronic gadget without the use of a microcontroller. This microcontroller changed the embedded system design so simple and advanced that the embedded market has become one of the most sought after for not only entrepreneurs but for design engineers also.

What is a Microcontroller?

A single chip computer or A CPU with all the peripherals like RAM, ROM, I/O Ports, Timers , ADCs etc... on the same chip. For ex: Motorola's 6811, Intel's 8051, Zilog's Z8 and PIC 16X etc...

MICROPROCESSORS & MICROCONTROLLERS:

Microprocessor:

A CPU built into a single VLSI chip is called a microprocessor. It is a general-purpose device and additional external circuitry are added to make it a microcomputer. The microprocessor contains arithmetic and logic unit (ALU), Instruction decoder and control unit, Instruction register, Program counter (PC), clock circuit (internal or external), reset circuit (internal or external) and registers. But the microprocessor has no on chip I/O Ports, Timers , Memory etc.

For example, Intel 8085 is an 8-bit microprocessor and Intel 8086/8088 a 16-bit microprocessor. The block diagram of the Microprocessor is shown in Fig.1

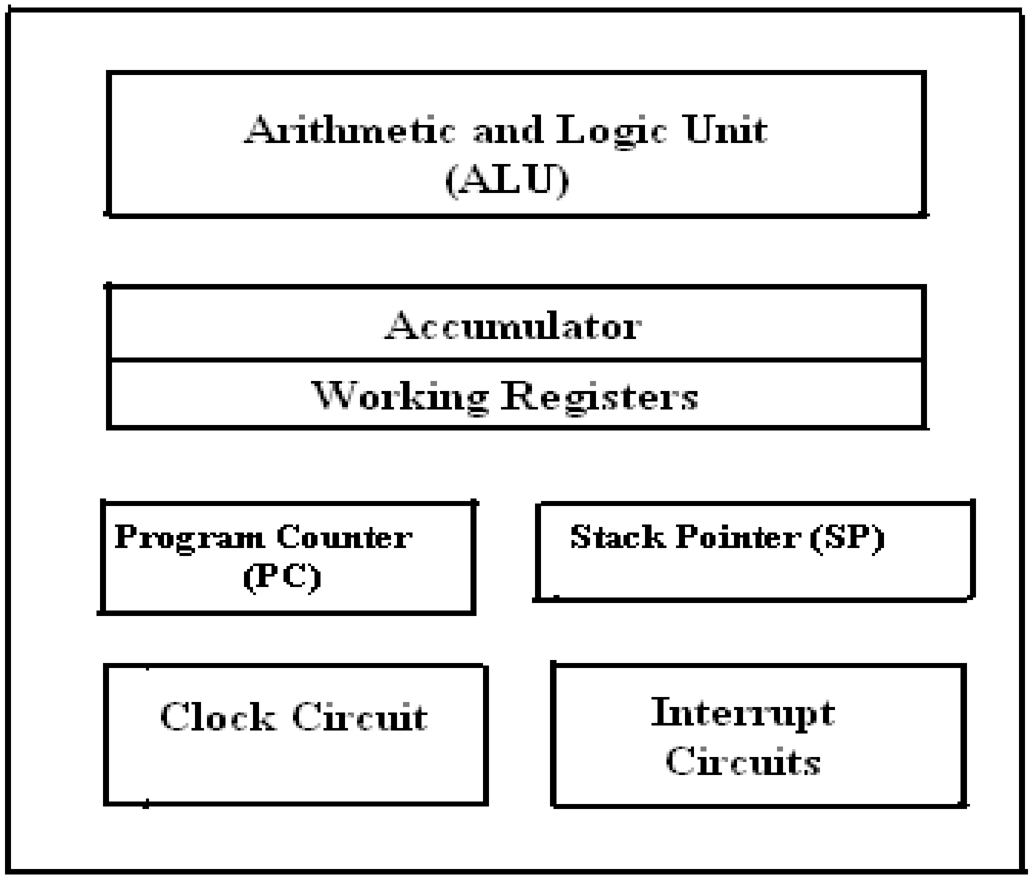


Fig.1 Block diagram of a Microprocessor.

MICROCONTROLLER :

A microcontroller is a highly integrated single chip, which consists of on chip CPU (Central Processing Unit), RAM (Random Access Memory), EPROM/PROM/ROM (Erasable Programmable Read Only Memory), I/O (input/output) – serial and parallel, timers, interrupt controller. For example, Intel

8051 is 8-bit microcontroller and Intel 8096 is 16-bit microcontroller. The block diagram of Microcontroller is shown in Fig.2.

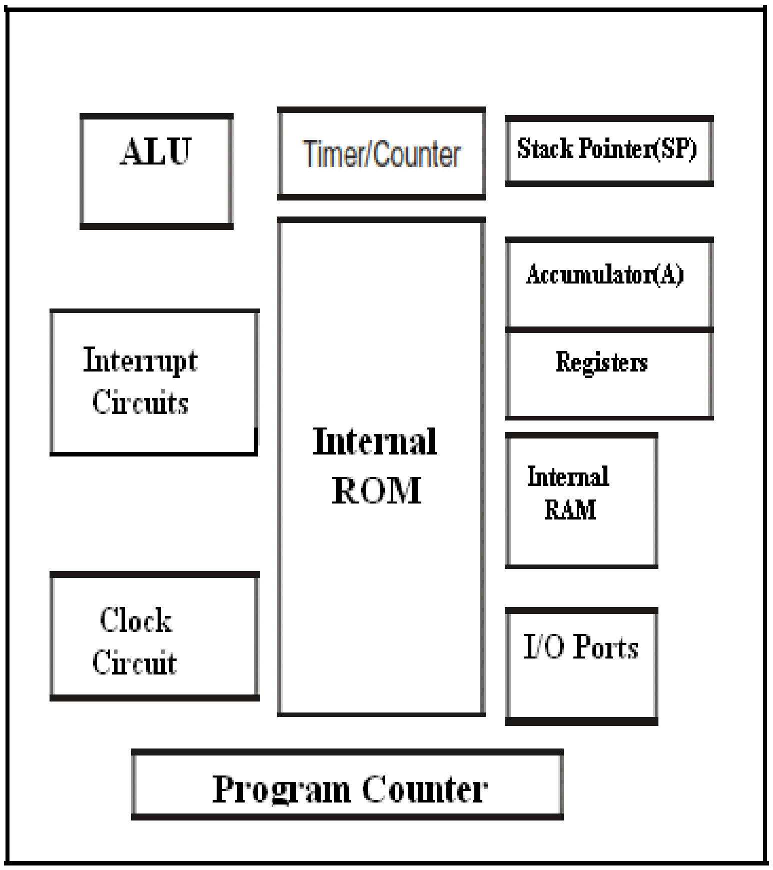


Fig.2.Block Diagram of a Microcontroller

Distinguish between Microprocessor and Microcontroller

S.No	Microprocessor	Microcontroller
1	A microprocessor is a general purpose device which is called a CPU	A microcontroller is a dedicated chip which is also called single chip computer.
2	A microprocessor do not contain onchip I/OPorts, Timers, Memories etc..	A microcontroller includes RAM, ROM, serial and parallel interface, timers, interrupt circuitry (in addition to CPU) in a single chip.
3	Microprocessors are most commonly used as the CPU in microcomputer systems	Microcontrollers are used in small, minimum component designs performing control-oriented applications.
4	Microprocessor instructions are mainly nibble or byte addressable	Microcontroller instructions are both bit addressable as well as byte addressable.
5	Microprocessor instruction sets are mainly intended for catering to large volumes of data.	Microcontrollers have instruction sets catering to the control of inputs and outputs.
6	Microprocessor based system design is complex and expensive	Microcontroller based system design is rather simple and cost effective
7	The Instruction set of microprocessor is complex	The instruction set of a Microcontroller is very simple with

	with large number of instructions.	less number of instructions. For, ex: PIC microcontrollers have only 35 instructions.
8	A microprocessor has zero status flag	A microcontroller has no zero flag.

EVOLUTION OF MICROCONTROLLERS :

The first microcontroller TMS1000 was introduced by Texas Instrumentsin the year 1974. In the year 1976, Motorola designed a Microprocessor chip called 6801 which replaced its earlier chip 6800 with certain add-on chips to make a computer. This paved the way for the new revolution in the history of chip design and gave birth to a new entity called "**Microcontroller".** Later the Intel company produced its first Microcontroller 8048 with a CPU and 1K bytes of EPROM, 64 Bytes of RAM an 8-Bit Timer and 27 I/O pins in 1976. Then followed the most popular controller 8051 in the year 1980 with 4K bytes of ROM,128 Bytes of RAM , a serial port, two 16-bit Timers , and 32 I/O pins. The 8051 family has many additions and improvements over the years and remains a most acclaimed tool for today's circuit designers. INTEL introduced a 16 bit microcontroller 8096 in the year 1982 . Later INTEL introduced 80c196 series of 16-bit Microcontrollers for mainly industrial applications. Microchip, another company has introduced an 8-bit Microcontroller PIC 16C64 in the year 1985.The 32-bit microcontrollers have been developed by IBM and Motorola. MPC 505 is a 32-bit RISC controller of Motorola. The 403 GA is a 32 -bit RISC embedded controller of IBM.

In recent times ARM company (Advanced RISC machines) has developed and introduced 32 bit controllers for high-end application devices like mobiles , Ipods etc...

TYPES OF MICROCONTROLLERS :

Microcontrollers can be classified on the basis of internal bus width, architecture, *memory and instruction set as 4-bit,8-bit,16-bit and 32-bit mircrontrollers.*

4-bit Microcontrollers: *These 4-bit microcontrollers are small size, minimum pin count and low cost controllers which are widely used for low end applications like LED & LCD display drivers ,portable battery chargers etc.. Their power consumption is also low. The popular 4-bit controllers are Renasa M34501 which is a 20 pin DIP chip with 4kB of ROM,256 Bytes of RAM,2-Counters and 14 I/O Pins. Similarly ATAM862 series from ATMEL.*

8-bit Microcontrollers : *These are the most popular and widely used microcontrollers .About 55% of all CPUs sold in the world are 8-bit microcontrollers only.The 8-bit microcontroller has 8-bitinternal bus and the ALU performs all the arithmetic and logical operations on a byte instruction. The well known 8-bit microcontroller is 8051 which was designed by Intel in the year 1980 for the use in embedded systems. Other 8-bit microcontrollers are Intel 8031/8052 and Motorola MC68HC11 and AVR Microcontrollers, Microchip's PIC Microcontrollers 12C5XX ,16C5X and 16C505 etc...*

16-bit Microcontrollers : *When the microcontroller performs 16-bit arithmetic and logical operations at an instruction, the microcontroller is said to be a 16-bit microcontroller. The internal bus width of 16-bit microcontroller is of 16-bit. These microcontrollers are having increased memory size and speed of operation when compared to 8-bit microcontrollers.These are most suitable for programming in Highlevel languages like C or C^{++} .They find applications in disk drivers,modems,printers,scanners and servomotor control. Examples of 16-bit microcontrollers are Intel 8096 family and Motorola MC68HC12 and MC68332 families, The performance and computing capability of 16 bit microcontrollers are enhanced with greater precision as compared to the 8-bit microcontrollers.*

32-Bit Microcontrollers :*These microcontrollers used in highend applications like Automative control, Communication networks,Robotics,Cell phones ,GPRS & PDAs etc..For EX:PIC32,ARM 7,ARM9 ,SHARP LH79520 ,ATMEL 32 (AVR) ,Texas Instrument's –. TMS320F2802x/2803x etc..are some of the popular 32-bit microcontrollers.*

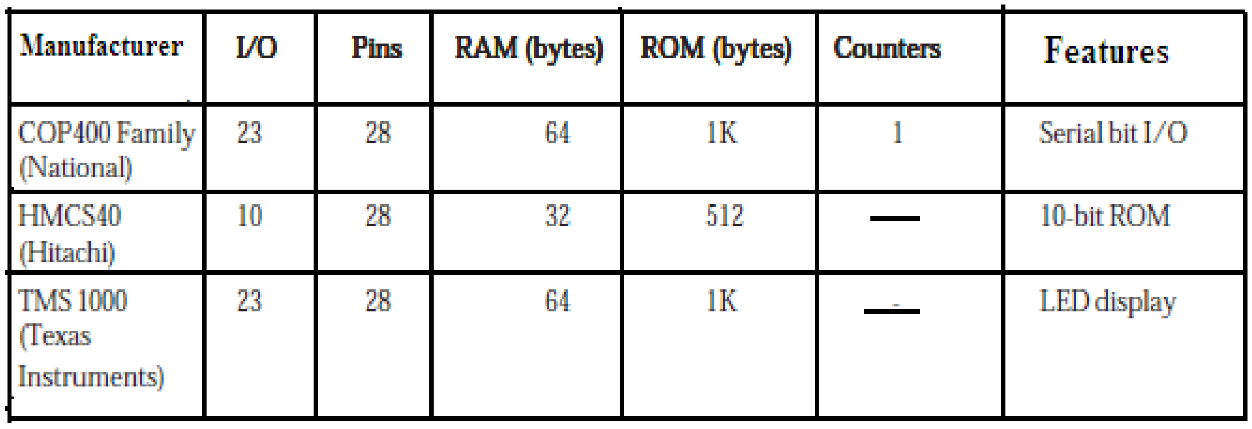
COMMERCIAL MICROCONTROLLERS

There are various manufacturers who are supplying various types of microcontrollers suitable for different applications depending on the power consumption and the available features..They are given below in tables . First the various members of INTEL 51 family are given in below table.

INTEL MCS 51 Family

Microcontroller	On chip RAM (Bytes)	On chip program memory	Timers/Counters	Interrupts	Serial ports
8031	128	None	2	5	1
8032	256	None	3	6	1
8051	128	4K ROM	2	5	1
8052	256	8K ROM	3	6	1
8751	128	4K EPROM	2	5	1
8752	256	8K EPROM	3	6	1

The following table gives the 4-bit microcontrollers from different manufacturers.



Manufacturer	I/O	Pins	RAM (bytes)	ROM (bytes)	Counters	Features
COP400 Family (National)	23	28	64	1K	1	Serial bit I/O
HMCS40 (Hitachi)	10	28	32	512	—	10-bit ROM
TMS 1000 (Texas Instruments)	23	28	64	1K	—	LED display

4-Bit Microcontrollers.

Manufacturer	I/O	Pins	RAM (bytes)	ROM (bytes)	Counters	Features
8048 (Intel)	27	40	64	1K	1	8k External memory
8051 (Intel)	32	40	128	4K	2	128k External memory, Boolean processing, serial port
COP800 Family (National)	24	28	64	1K	1	Serial bit I/O, 8- channel A/D converter
6805 (Motorola)	20	28	64	1K	1	PLL frequency synthe-sizer,
68hc11 (Motorola)	40	52	256	8K	2	A/D, PWM generator, pulse accumulator
TMS370 (Texas)	55	68	256	4K	2	watchdog timer, Instru-ments) Serial ports, A/D (8 bit, 8 channel)
PIC (Micro Chip)	12	18	25	1K	0	small pin count, very low power consumption

8-Bit Microcontrollers.

Manufacturer	I/O	Pins	RAM (bytes)	ROM (bytes)	Counters	Features
80c196(INTEL)	40	68	232 8K	2	—	PWM generator, watchdog timer
HPC Family (National)	52	68	512 16K	4	—	PWM generator, watchdog timer, 8-channel A/ D, serial port

16-Bit Microcontrollers

The following table gives the list of PIC microcontrollers from Micro chip Inc

Microcontroller	Pins	I/O Lines	On chip ADCs	EPROM X 12 words	On chip RAM (Bytes)
16C54	18	12	None	512	25
16C55	28	20	None	512	24
16C56	18	12	None	1k	25
16C57	28	20	None	2k	72
17C42A	40	33	None	2k	232
17C43	40	33	None	4k	454
17C44	40	33	None	8k	454
17C71	18	13	8bit ADCs	1kx14	36
17C752	40	33	10Bit ADC	8kx16	678

MICROCONTROLLER DEVELOPMENT TOOLS:

To develop an assembly language program we need certain program development tools. An assembly language program consists of Mnemonics which are nothing but short abbreviated English instructions given to the controller.The various development tools required for Microcontroller programming are explained below.

1. Editor : An Editor is a program which allows us to create a file containing the assembly language statements for the program. Examples of some editors are PC write Wordstar. As we type the program the editor stores the ACSII codes for the letters and numbers in successive RAM locations. If any typing mistake is done editor will alert us to correct it. If we leave out a program statement an editor will let you move everything down and insert a line. After typing all the program we have to save the program . This we call it as source file. The next step is to process the source file with an assembler.

Ex: Sample. asm

2.Assembler : An Assembler is used to translate the assembly language mnemonics into machine language(i.e binary codes). When you run the assembler it reads the source file of your program from where you have saved it. The assembler generates a filee with the extension **.hex**. This file consists of hexadecimal values encoding a sequence of data and their starting offset or absolute address.

3.Compiler : A compiler is a program which converts the high level language program like "C" into binary or machine code. Using high level languages it is easy to manage complex data structures which are often required for data manipulation. Because of its ease , flexibility and debug options now a days the compilers have become very popular in the market. Compilers like Keil ,Ride and IAR workbench are very popular.

3. Debugger/Simulator : A debugger is a program which allows execute the program, and troubleshoot or debug it. The debugger allows to look into the contents of registers and memory locations after the program runs. We can also change the contents of registers and memory locations and rerun the program. Some debuggers allows to stop the program after each instruction so that you can check or alter memory and register contents. This is called single step debug. A debugger also allows to set a breakpoint at any point in the program. If we insert a break point , the debugger will run the program up to the instruction where the breakpoint is put and then stop the execution.

A simulator is a software program which virtually executes the instructions similar to a microcontroller and shows the results. This will help in evaluating the results without committing any errors. By doing so we can detect the possible logic errors

INTEL 8051 MICRCONTROLLER :

The 8051 microcontroller is a very popular 8-bit microcontroller introduced by Intel in the year 1981 and it has become almost the academic standard now a days. The 8051 is based on an 8-bit CISC core with Harvard architecture. Its 8-bit architecture is optimized for control applications with extensive Boolean processing. It is available as a 40-pin DIP chip and works at +5 Volts DC. The salient features of 8051 controller are given below.

SALIENT FEATURES : The salient features of 8051 Microcontroller are

i. 4 KB on chip program memory (ROM or EPROM)).

ii. 128 bytes on chip data memory(RAM).

iii. 8-bit data bus

iv. 16-bit address bus

v. 32 general purpose registers each of 8 bits

vi. Two -16 bit timers T_0 and T_1

vii. Five Interrupts (3 internal and 2 external).

ix. Four Parallel ports each of 8-bits (PORT0, PORT1,PORT2,PORT3) with a total of 32 I/O

lines.

x. One 16-bit program counter and One 16-bit DPTR (data pointer)

xi. One 8-bit stack pointer

xii. One Microsecond instruction cycle with 12 MHz Crystal.

xiii. One full duplex serial communication port.

ARCHITECTURE & BLOCK DIAGRAM OF 8051 MICROCONTROLLER:

The architecture of the 8051 microcontroller can be understood from the block diagram. It has Harward architecture with RISC (Reduced Instruction Set Computer) concept. The block diagram of 8051 microcontroller is shown in Fig 3. below1.It consists of an 8-bit ALU, one 8-bit PSW(Program Status Register), A and B registers , one 16-bit Program counter , one 16-bit Data pointer register(DPTR),128 bytes of RAM and 4kB of ROM and four parallel I/O ports each of 8-bit width.

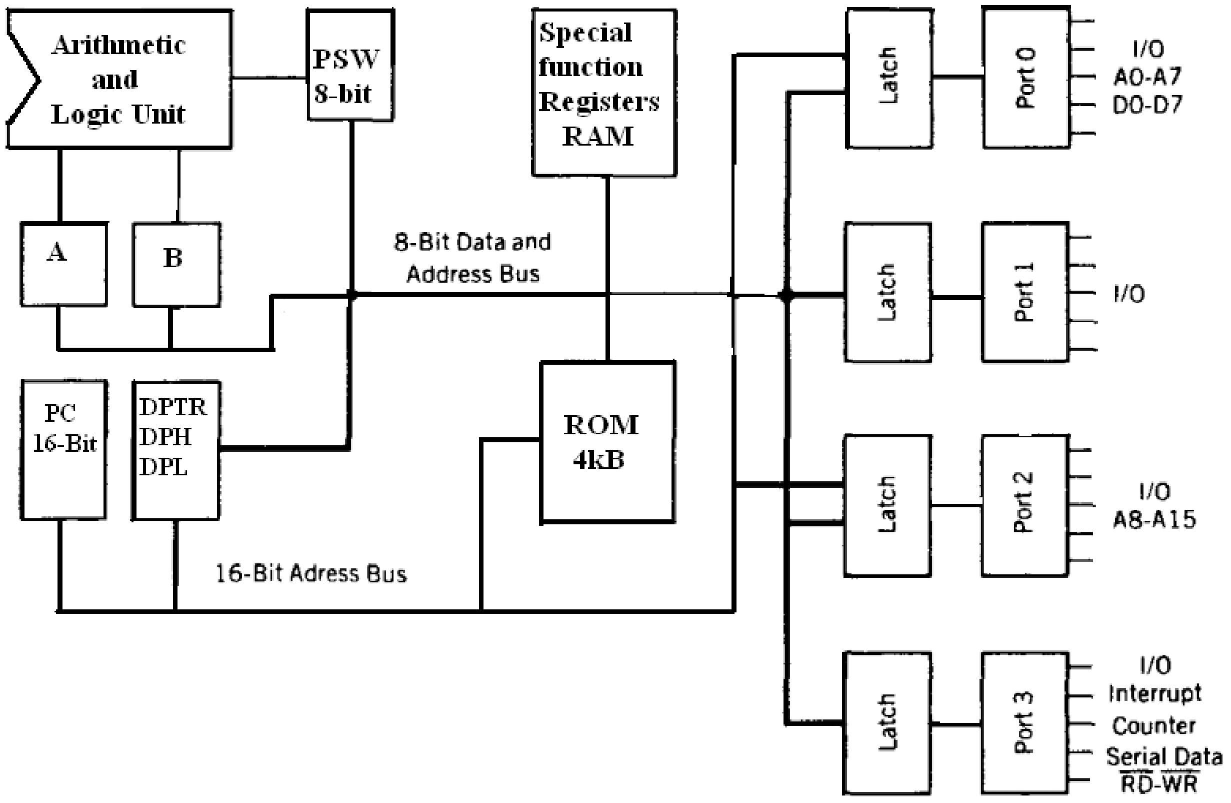

Arithmetic and Logic Unit
PSW 8-bit
Special function Registers RAM
A
B
8-Bit Data and Address Bus
PC 16-Bit
DPTR DPH DPL
ROM 4kB
16-Bit Adress Bus
Latch
Port 0
I/O A0-A7 D0-D7
Port 1
I/O
Port 2
I/O A8-A15
Port 3
I/O Interrupt Counter Serial Data RD-WR

Fig.3. Block Diagram of 8051 Microcontroller

8051 has 8-bit ALU which can perform all the 8-bit arithmetic and logical operations in one machine cycle. The ALU is associated with two registers A & B

A and B Registers *: The A and B registers are special function registers which hold the results of many arithmetic and logical operations of 8051.The A register is also called the* ***Accumulator*** *and as it's name suggests, is used as a general register to accumulate the results of a large number of instructions. By default it is used for all mathematical operations and also data transfer operations between CPU and any external memory.*

The B register is mainly used for multiplication and division operations along with

A register.

MUL AB : DIV AB.

It has no other function other than as a location where data may be stored.

The R registers: The "R" registers are a set of eight registers that are named R0, R1, etc. up to
and including R7. These registers are used as auxillary registers in many operations. The "R" registers are also used to temporarily store values.

Program Counter(PC) : 8051 has a 16-bit program counter .The program counter always points to the address of the next instruction to be executed. After execution of one instruction the program counter is incremented to point to the address of the next instruction to be executed.It is the contents of the PC that are placed on the address bus to find and fetch the desired instruction.Since the PC is 16-bit width ,8051 can access program addresses from 0000H to FFFFH ,a total of 6kB of code.

Stack Pointer Register (SP) : It is an 8-bit register which stores the address of the stack top. i.e the Stack Pointer is used to indicate where the next value to be removed from the stack should be taken from. When a value is pushed onto the stack, the 8051 first increments the value of SP and then stores the value at the resulting memory location. Similarly when a value is popped off the stack, the 8051 returns the value from the memory location indicated by SP, and then decrements the value of SP. Since the SP is only 8-bit wide it is incremented or decremented by two . SP is modified directly by the 8051 by six instructions: PUSH, POP, ACALL, LCALL, RET, and RETI. It is also used intrinsically whenever an interrupt is triggered.

STACK in 8051 Microcontroller : The stack is a part of RAM used by the CPU to store information temporarily. This information may be either data or an address .The CPU needs this storage area as there are only limited number of registers. The register used to access the stack is called the Stack pointer which is an 8-bit register..So,it can take values of 00 to FF H.When the 8051 is powered up ,the SP register contains the value 07.i.e the RAM location value 08 is the first location being used for the stack by the 8051 controller

There are two important instructions to handle this stack.One is the PUSH and the Other is the POP. The loading of data from CPU registers to the stack is done by PUSH and the loading of the contents of the stack back into aCPU register is done by POP.

```
EX :  MOV R6 ,#35 H
      MOV R1 ,#21 H
      PUSH 6
      PUSH 1
```

In the above instructions the contents of the Registers R6 and R1 are moved to stack and they occupy the 08 and 09 locations of the stack.Now the contents of the SP are incremented by two and it is 0A

Similarly POP 3 instruction pops the contents of stack into R3 register.Now the contents of the SP is decremented by 1

In 8051 the RAM locations 08 to 1F (24 bytes) can be used for the Stack.In any program if we need more than 24 bytes of stack ,we can change the SP point to RAM locations 30-7F H.this can be done with the instruction MOV SP,# XX.

Data Pointer Register(DPTR) : It is a 16-bit register which is the only user-accessible. DPTR, as the name suggests, is used to point to data. It is used by a number of commands which allow the 8051 to access external memory. When the 8051 accesses external memory it will access external memory at the address indicated by DPTR. This DPTR can also be used as two 8-registers DPH and DPL.

Program Status Register (PSW) : The 8051 has a 8-bit PSW register which is alsoknown as Flag register.In the 8-bit register only 6-bits are used by 8051.The two unused bits are user definable bits.In the 6-bits four of them are conditional flags .They are Carry –CY,Auxiliary Carry-AC, Parity-P,and Overflow-OV .These flag bits indicate some conditions that resulted after an instruction was executed.

D7							D0
CY	AC	FO	RS1	RS0	OV	-	P

The bits PSW3 and PSW4 are denoted as RS0 and RS1 and these bits are used th select the bank registers of the RAM location. The meaning of various bits of PSW register is shown below.

CY	*PSW.7*	*Carry Flag*
AC	*PSW.6*	*Auxiliary Carry Flag*
FO	*PSW.5*	*Flag 0 available for general purpose .*
RS1	*PSW.4*	*Register Bank select bit 1*
RS0	*PSW.3*	*Register bank select bit 0*
OV	*PSW.2*	*Overflow flag*
---	*PSW.1*	*User difinable flag*
P	*PSW.0*	*Parity flag .set/cleared by hardware.*

The selection of the register Banks and their addresses are given below.

RS1	***RS0***	***Register Bank***	***Address***
0	*0*	*0*	*00H-07H*
0	*1*	*1*	*08H-0FH*
1	*0*	*2*	*10H-17H*
1	*1*	*3*	*18H-1FH*

Memory organization *: The 8051 microcontroller has 128 bytes of Internal RAM and*

4kB of on chip ROM .The RAM is also known as Data memory and the ROM is known as program memory. The program memory is also known as Code memory .This Code memory holds the actual 8051 program that is to be executed. In 8051 this memory is limited to 64K .Code memory may be found on-chip, as ROM or EPROM. It may also be stored completely off-chip in an external ROM or, more commonly, an external EPROM. The 8051 has only 128 bytes of Internal RAM but it supports 64kB of external RAM. As the name suggests, external RAM is any random access memory which is off-chip. Since the memory is off-chip it is not as flexible interms of accessing, and is also slower. For example, to increment an Internal RAM location by 1,it requires only 1 instruction and 1 instruction cycle but to increment a 1-byte value stored in External RAM requires 4 instructions and 7 instruction cycles. So, here the external memory is 7 times slower.

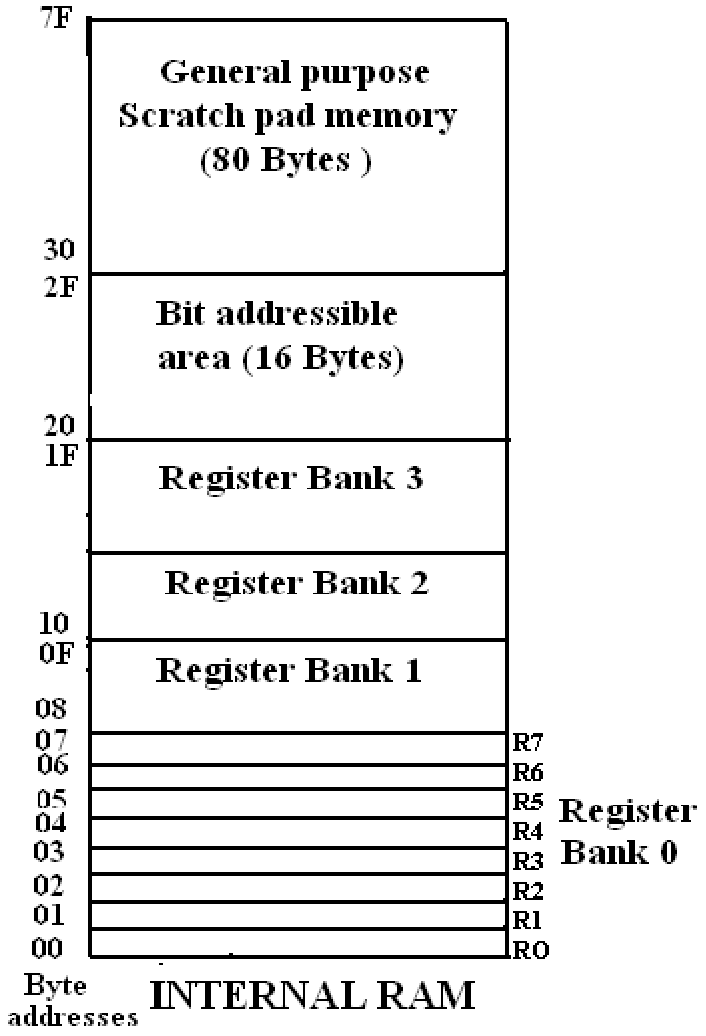
Internal RAM OF 8051 :

This Internal RAM is found on-chip on the 8051 .So it is the fastest RAM available, and it is also the most flexible in terms of reading, writing, and modifying it's contents. Internal RAM is volatile, so when the 8051 is reset this memory is cleared. The 128 bytes of internal RAM is organized as below.

(i) Four register banks (Bank0,Bank1, Bank2 and Bank3) each of 8-bits (total 32 bytes). The default bank register is Bank0. The remaining Banks are selected with the help of RS0 and RS1 bits of PSW Register.

(ii) 16 bytes of bit addressable area and

(iii) 80 bytes of general purpose area (Scratch pad memory) as shown in the diagram below. This area is also utilized by the microcontroller as a storage area for the operating stack.

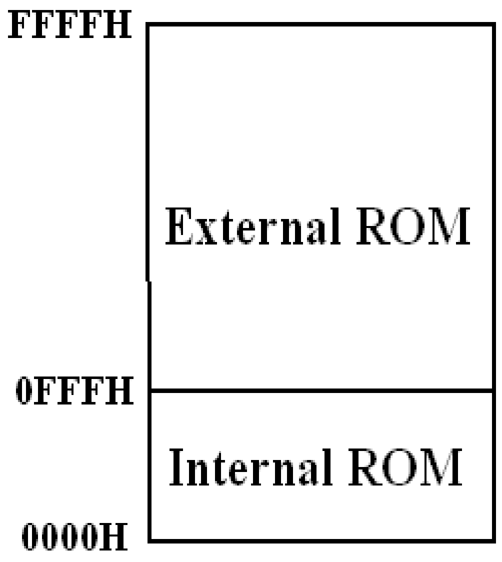


The 32 bytes of RAM from address 00 H to 1FH are used as working registers organized as four banks of eight registers each.The registers are named as R0-R7 .Each register can be addressed by its name or by its RAM address.

For EX : MOV A, R7 or MOV R7,#05H

Internal ROM (On –chip ROM): The 8051 microcontroller has 4kB of on chip ROM but it can be extended up to 64kB.This ROM is also called program memory or code memory. The CODE segment is accessed using the program counter (PC) for opcode fetches and by DPTR for data. The external ROM is accessed when the

EA(active low) pin is connected to ground or the contents of program counter exceeds 0FFFH.When the Internal ROM address is exceeded the 8051 automatically fetches the code bytes from the external program memory.



SPECIAL FUNCTION REGISTERS (SFRs) : *In 8051 microcontroller there certain registers which uses the RAM addresses from 80h to FFh and they are meant for certain specific operations .These registers are called Special function registers (SFRs).Some of these registers are bit addressable also.*

The list of SFRs and their functional names are given below. In these SFRs some of them are related to I/O ports (P0,P1,P2 and P3) and some of them are meant for control operations (TCON,SCON, PCON..) and remaining are the auxillary SFRs, in the sense that they don't directly configure the 8051.

S.No	Symbol		Name of SFR	Address (Hex)
1	ACC*		Accumulator	**0E0**
2	B*		B-Register	**0F0**
3	PSW*		Program Status word register	**0D0**
4	SP		Stack Pointer Register	**81**
5	DPTR	DPL	Data pointer low byte	**82**
		DPH	Data pointer high byte	**83**
6	P0*		Port 0	**80**
	P1*		Port 1	**90**
8	P2*		Port 2	**0A**
9	P3*		Port 3	**0B**
10	IP*		Interrupt Priority control	**0B8**
11	IE*		Interrupt Enable control	**0A8**
12	TMOD		Tmier mode register	**89**
13	TCON*		Timer control register	**88**
14	TH0		Timer 0 Higher byte	**8C**
15	TL0		Timer 0 Lower byte	**8A**
16	TH1		Timer 1Higher byte	**8D**
17	TL1		Timer 1 lower byte	**8B**
18	SCON*		Serial control register	**98**
19	SBUF		Serial buffer register	**99**
20	PCON		Power control register	**87**

The * indicates the bit addressable SFRs

Table:SFRs of 8051 Microcontroller

PARALLEL I /O PORTS :

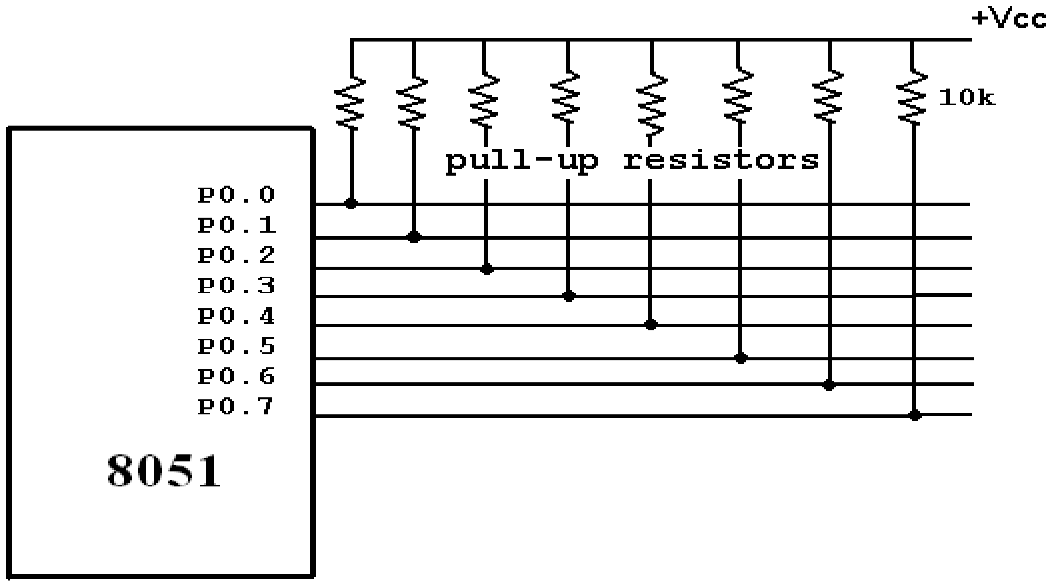
The 8051 microcontroller has four parallel I/O ports , each of 8-bits .So, it provides the user 32 I/O lines for connecting the microcontroller to the peripherals. The four ports are P0 (Port 0), P1(Port1) ,P2(Port 2) and P3 (Port3). Upon reset all the ports are output ports. In order to make them input, all the ports must be set i.e a high bit must be sent to all the port pins. This is normally done by the instruction "SETB".

```
Ex: MOV A,#0FFH        ;  A =  FF
    MOV P0,A           ; make P0 an input port
```

PORT 0:

Port 0 is an 8-bit I/O port with dual purpose. If external memory is used, these port pins are used for the lower address byte address/data (AD_0-AD_7), otherwise all bits of the port are either input or output.. Unlike other ports, Port 0 is not provided with pull-up resistors internally ,so for PORT0 pull-up resistors of nearly 10k are to be connected externally as shown in the fig.2.

***Dual role of port 0**: Port 0 can also be used as address/data bus(AD0-AD7), allowing it to be used for both address and data. When connecting the 8051 to an external memory, port 0 provides both address and data. The 8051 multiplexes address and data through port 0 to save the pins. ALE indicates whether P0 has address or data. When ALE = 0, it provides data D0-D7, and when ALE =1 it provides address and data with the help of a 74LS373 latch.*



Port 1: Port 1 occupies a total of 8 pins (pins 1 through 8). It has no dual application and acts only as input or output port. In contrast to port 0, this port does not need any pull-up resistors since pull-up resistors connected internally. Upon reset, Port 1 is configured as an output port. To configure it as an input port , port bits must be set i.e a high bit must be sent to all the port pins. This is normally done by the instruction "SETB". For Ex :

```
 MOV  A, #0FFH    ; A=FF HEX
 MOV  P1,A     ;       make P1 an input port by writing  1's to all of its pins
```

Port 2 : Port 2 is also an eight bit parallel port. (pins 21- 28). It can be used as input or output port. As this port is provided with internal pull-up resistors it does not need any external pull-up resistors. Upon reset, Port 2 is configured as an output port. If the port is to be used as input port, all the port bits must be made high by sending FF to the port. For ex,

```
MOV  A, #0FFH          ; A=FF hex
 MOV  P2, A                  ; make P2 an input port by writing all 1's to it
```

Dual role of port 2 : Port2 lines are also associated with the higher order address lines A8-A15. In systems based on the 8751, 8951, and DS5000, Port2 is used as *simple I/O port.. But, in 8031-based systems, port 2 is used along with P0 to provide the 16-bit address for the external memory. Since an 8031 is capable of accessing 64K bytes of external memory, it needs a path for the 16 bits of the address. While P0 provides the lower 8 bits via A0-A7, it is the job of P2 to provide bits A8-A15 of the address. In other words, when 8031 is connected to external memory, Port 2 is used for the upper 8 bits of the 16 bit address, and it cannot be used for I/O operations.*

***PORT 3** : Port3 is also an 8-bit parallel port with dual function.(pins 10 to 17). The port pins can be used for I/O operations as well as for control operations. The details of these additional operations are given below in the table. Port 3 also do not need any external pull-up resistors as they are provided internally similar to the case of Port2 & Port 1. Upon reset port 3 is configured as an output port . If the port is to be used as input port, all the port bits must be made high by sending FF to the port. For ex,*

```
MOV A, #0FFH        ; A= FF hex

 MOV P3, A           ; make P3 an input port by writing all 1's to it
```

***Alternate Functions of Port 3 :** P3.0 and P3.1 are used for the RxD (Receive Data) and TxD (Transmit Data) serial communications signals. Bits P3.2 and P3.3 are meant for external interrupts. Bits P3.4 and P3.5 are used for Timers 0 and 1 and P3.6 and P3.7 are used to provide the write and read signals of external memories connected in 8031 based systems*

S.No	**Port 3 bit**	**Pin No**	**Function**
1	P3.0	10	RxD
2	P3.1	11	TxD
3	P3.2	12	
4	P3.3	13	
5	P3.4	14	T0

6	P3.5	15	T1
7	P3.6	16	
8	P3.7	17	

Table: PORT 3 alternate functions

Interrupt Structure: An interrupt is an external or internal event that disturbs the microcontroller to inform it that a device needs its service. The program which is associated with the interrupt is called the **interrupt service routine** (ISR) or **interrupt handler**. Upon receiving the interrupt signal the Microcontroller , finish current instruction and saves the PC on stack. Jumps to a fixed location in memory depending on type of interrupt Starts to execute the interrupt service routine until RETI (return from interrupt)Upon executing the RETI the microcontroller returns to the place where it was interrupted. Get pop PC from stack

The 8051 microcontroller has **FIVE** interrupts in addition to Reset. They are

Timer 0 overflow Interrupt

Timer 1 overflow Interrupt

External Interrupt 0(INT0)

External Interrupt 1(INT1)

Serial Port events (buffer full, buffer empty, etc) Interrupt

Each interrupt has a specific place in code memory where program execution (interrupt service routine) begins.

External Interrupt 0: 0003 H

Timer 0 overflow: 000B H

External Interrupt 1:0013 H

Timer 1 overflow: 001B H

Serial Interrupt : 0023 H

Upon reset all Interrupts are disabled & do not respond to the Microcontroller.

These interrupts must be enabled by software in order for the Microcontroller to respond to them. This is done by an 8-bit register called Interrupt Enable Register (IE).

Interrupt Enable Register :

EA	—	ET2	ES	ET1	EX1	ET0	EX0

EA : Global enable/disable. To enable the interrupts this bit must be set High.

--- : Undefined-reserved for future use.

ET2 : Enable /disable Timer 2 overflow interrupt.

ES : Enable/disable Serial port interrupt.

ET1 : Enable /disable Timer 1 overflow interrupt.

EX1 : Enable/disable External interrupt1.

ET0 : Enable /disable Timer 0 overflow interrupt.

EX0 : Enable/disable External interrupt0

Upon reset the interrupts have the following priority.(Top to down). The interrupt with the highest PRIORITY gets serviced first.

External interrupt 0 (INT0)

Timer interrupt0 (TF0)

External interrupt 1 (INT1)

Timer interrupt1 (TF1)

Serial communication (RI+TI)

Priority can also be set to "high" or "low" by 8-bit IP register.- Interrupt priority register

——	——	PT2	PS	PT1	PX1	PT0	PX0

IP.7: reserved

IP.6: reserved

IP.5: Timer 2 interrupt priority bit (8052 only)

IP.4: Serial port interrupt priority bit

IP.3: Timer 1 interrupt priority bit

IP.2: External interrupt 1 priority bit

IP.1: Timser 0 interrupt priority bit

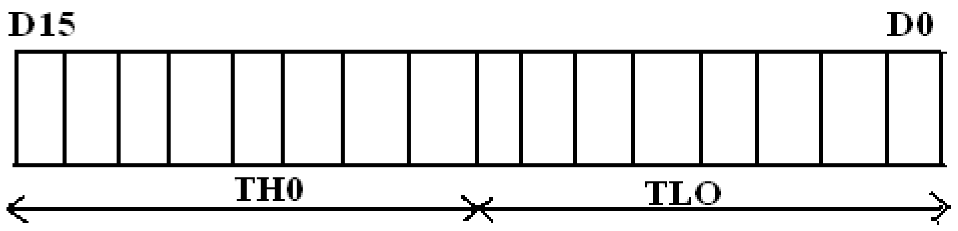
IP.0: External interrupt 0 priority bit

TIMERS in 8051 Microcontrollers : The 8051 microcontroller has two 16-bit timers Timer 0 (T0) and Timer 1(T1) which can be used either to generate accurate time delays or as event counters. These timers are accessed as two 8-bit registers TL0, TH0 & TL1 ,TH1 because the 8051 microcontroller has 8-bit architecture.

TIMER 0 : The Timer 0 is a 16-bit register and can be treated as two 8-bit registers (TL0 & TH0) and these registers can be accessed similar to any other registers like A,B or R1,R2,R3 etc...

Ex : The instruction Mov TL0,#07 moves the value 07 into lower byte of Timer0.

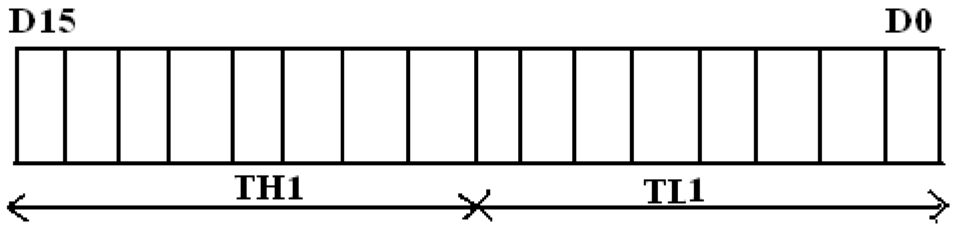
Similarly Mov R5,TH0 saves the contents of TH0 in the R5 register.



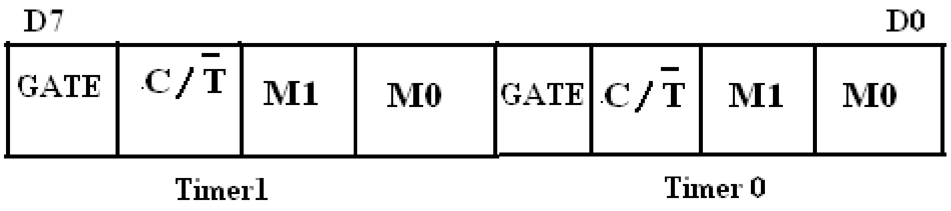
TIMER 1 : The Timer 1 is also a 16-bit register and can be treated as two 8-bit registers (TL1 & TH1) and these registers can be accessed similar to any other registers like A,B or R1,R2,R3 etc...

Ex : The instruction MOV TL1,#05 moves the value 05 into lower byte of Timer1.

Similarly MOV R0,TH1 saves the contents of TH1 in the R0 register



TMOD Register : The various operating modes of both the timers T0 and T1 are set by an 8-bit register called TMOD register. In this TMOD register the lower 4-bits are meant for Timer 0 and the higher 4-bits are meant for Timer1.



GATE: This bit is used to start or stop the timers by hardware .When GATE= 1 ,the timers can be started / stopped by the external sources. When GATE= 0, the timers can be started or stopped by software instructions like SETB TR0 or SETB TR1

C/T (clock/Timer) : This bit decides whether the timer is used as delay generator or event counter. When **C/T = 0** ,the Timer is used as delay generator and if C/T=1 the timer is used as an event counter. The clock source for the time delay is the crystal frequency of 8051.

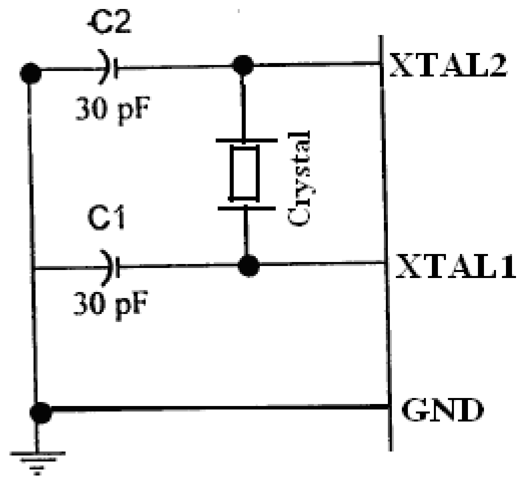
M1,M0 (Mode) : These two bits are the timer mode bits. The timers of the 8051 can be configured in three modes.Mode0, Mode1 and Mode2.The selection and operation of the modes is shown below.

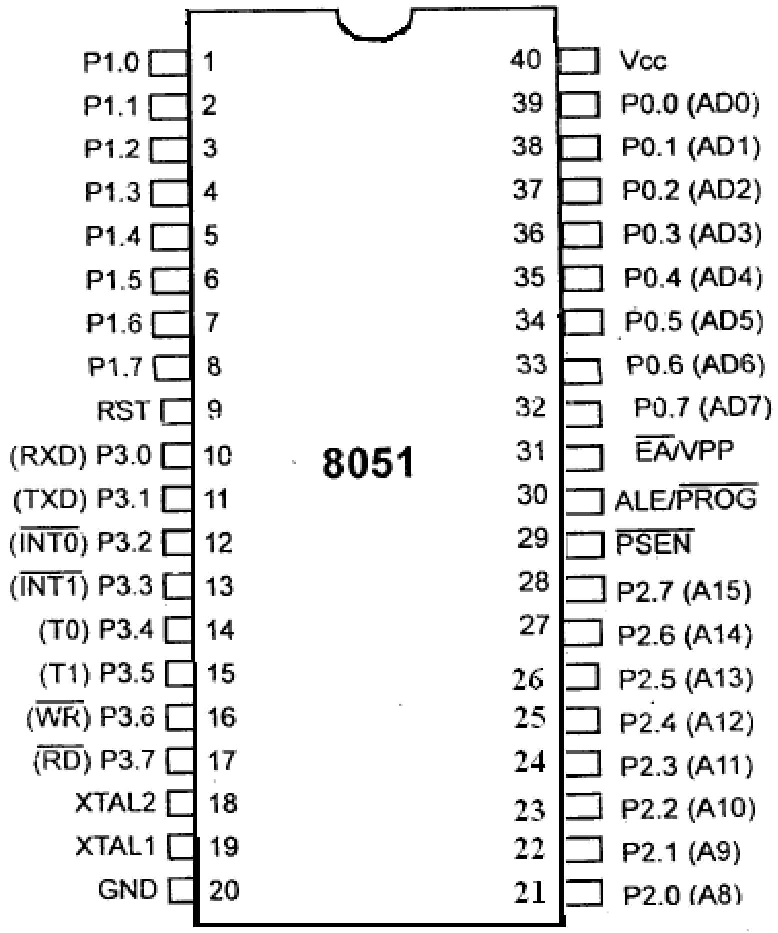
S.No	M0	M1	Mode	Operation

1	0	0	0	13-bit Timer mode 8-bit Timer/counter. THx with TLx as 5-bit prescalar
2	0	1	1	16-bit Timer mode.16-bit timer /counter without pre-scalar
3	1	0	2	8-bit auto reload. THx contains a value that is to be loaded into TLx each time it overflows
4	1	1	3	Split timer mode

PIN Diagram of 8051 Microcontroller : The 8051 microcontroller is available as a 40 pin DIP chip and it works at +5 volts DC. Among the 40 pins , a total of 32 pins are allotted for the four parallel ports P0,P1,P2 and P3 i.e each port occupies 8-pins .The remaining pins are VCC, GND, XTAL1, XTAL2, RST, EA ,PSEN.

XTAL1,XTAL2: These two pins are connected to Quartz crystal oscillator which runs the on-chip oscillator. The quartz crystal oscillator is connected to the two pins along with a capacitor of 30pF as shown in the circuit. If we use a source other than the crystal oscillator, it will be connected to XTAL1 and XTAL2 is left unconnected.





***RST**: The RESET pin is an input pin and it is an active high pin. When a high pulse is applied to this pin the microcontroller will reset and terminate all activities. Upon reset all the registers except PC will reset to 0000 Value and PC register will reset to 0007 value.*

$\overline{\text{EA}}$ ***(External Access)**: This pin is an active low pin. This pin is connected to ground when microcontroller is accessing the program code stored in the external memory and connected to Vcc when it is accessing the program code in the on chip memory. This pin should not be left unconnected.*

$\overline{\text{PSEN}}$ ***(Program Store Enable) :** This is an output pin which is active low. When the* microcontroller is accessing the program code stored in the external ROM ,this pin is connected to the OE (Output Enable) pin of the ROM.

ALE (Address latch enable): This is an output pin, which is active high. When connected to external memory , port 0 provides both address and data i.e address and data are multiplexed through port 0 .This ALE pin will demultiplex the address and data bus .When the pin is High , the AD bus will act as address bus otherwise the AD bus will act as Data bus.

P0.0- P0.7(AD0-AD7) : The port 0 pins multiplexed with Address/data pins .If the microcontroller is accessing external memory these pins will act as address/data pins otherwise they are used for Port 0 pins.

P2.0- P2.7(A8-A15) : The port2 pins are multiplexed with the higher order address pins .When the microcontroller is accessing external memory these pins provide the higher order address byte otherwise they act as Port 2 pins.

P1.0- P1.7 : These 8-pins are dedicated for Port1 to perform input or output port operations.

P3.0- P3.7 : These 8-pins are meant for Port3 operations and also for some control operations like Read,Write,Timer0,Timer1 ,INT0,INT1 ,RxD and TxD

ADDRESSING MODES OF 8051 :

The way in which the data operands are accessed by different instructions is known as the addressing modes. There are various methods of denoting the data operands in the instruction. The 8051 microcontroller supports mainly 5 addressing modes. They are

1.Immediate addressing mode

2.Direct Addressing mode

3.Register addressing mode

4. Register Indirect addressing mode

5.Indexed addressing mode

Immediate addressing mode : The addressing mode in which the data operand is a constant and it is a part of the instruction itself is known as Immediate addressing mode. Normally the data must be preceded by a # sign. This addressing mode can be used to transfer the data into any of the registers including DPTR.

Ex: MOV A , # 27 H : The data (constant) 27 is moved to the accumulator register

ADD R1 ,#45 H : Add the constant 45 to the contents of the accumulator

MOV DPTR ,# 8245H :Move the data 8245 into the data pointer register.

MOV P1,#21 H

Direct addressing mode: The addressing mode in which the data operand is in the RAM location (00 -7FH) and the address of the data operand is given in the instruction is known as Direct addressing mode. The direct addressing mode uses the lower 128 bytes of Internal RAM and the SFRs

MOV R1, 42H : Move the contents of RAM location 42 into R1 register

MOV 49H,A : Move the contents of the accumulator into the RAM location 49.

ADD A, 56H : Add the contents of the RAM location 56 to the accumulator

Register addressing mode :The addressing mode in which the data operand to be manipulated lies in one of the registers is known as register addressing mode.

MOV A,R0 : Move the contents of the register R0 to the accumulator

ADD A,R6 :Add the contents of R6 register to the accumulator

MOV P1, R2 : Move the contents of the R2 register into port 1

MOV R5, R2 : This is invalid .The data transfer between the registers is not allowed.

Register Indirect addressing mode : The addressing mode in which a register is used as a pointer to the data memory block is known as Register indirect addressing mode.

MOV A,@ R0 :Move the contents of RAM location whose address is in R0 into **A** (accumulator)

MOV @ R1 , B : Move the contents of B into RAM location whose address is held by R1

When R0 and R1 are used as pointers, they must be preceded by @ sign

One of the advantages of register indirect addressing mode is that it makes accessing the data more dynamic than static as in the case of direct addressing mode.

Indexed addressing mode : This addressing mode is used in accessing the data elements of lookup table entries located in program ROM space of 8051.

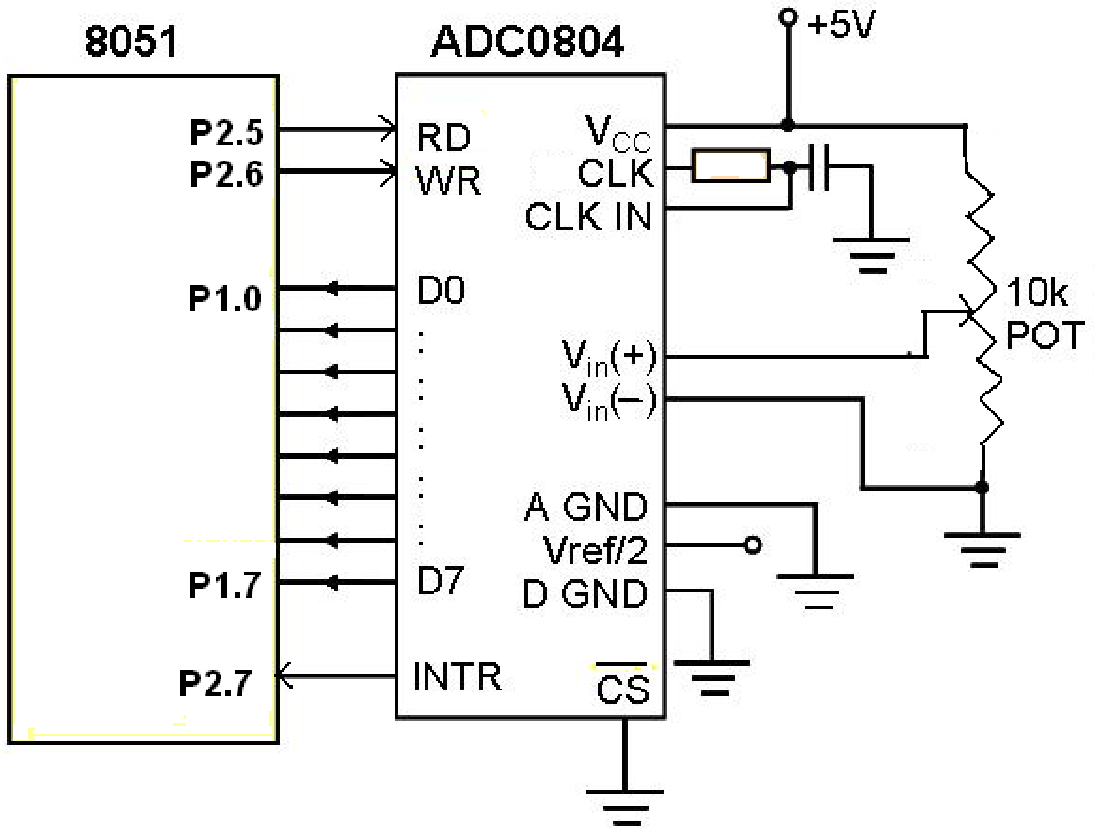
Ex : MOVC A,@ A+DPTR

The 16-bit register DPTR and register A are used to form the address of the data element stored in on-chip ROM. Here C denotes code .In this instruction the contents of A are added to the 16-bit DPTR register to form the 16-bit address of the data operand.

Interfacing of ADC 0804 to 8051 Microcontroller :

ADC 0804 is a single channel analog to digital converter i.e., it can take only one analog signal. ADC 0804 has 8 bit resolution. The higher resolution ADC gives smaller step size. Step size is smallest change that can be measured by an ADC. For an ADC with resolution of 8 bits, the step size is 19.53mV (5V/255). The time taken by the ADC to convert analog data into digital form depends on the frequency of clock source. The conversion time of ADC 0804 is around 110us. To use the internal clock a capacitor and resistor are used as shown in the circuit. The input to the ADC is given from a regulated power supply and a 10K potentiometer

The 8051 Microcontroller is used to provide the control signals to the ADC. CS(chip select) pin of ADC is directly connected to ground. The pin P1.1, P1.0 and P1.2 are connected to the pin WR, RD and INTR of the ADC respectively. When the input voltage from the preset is varied the output of ADC varies also varies.



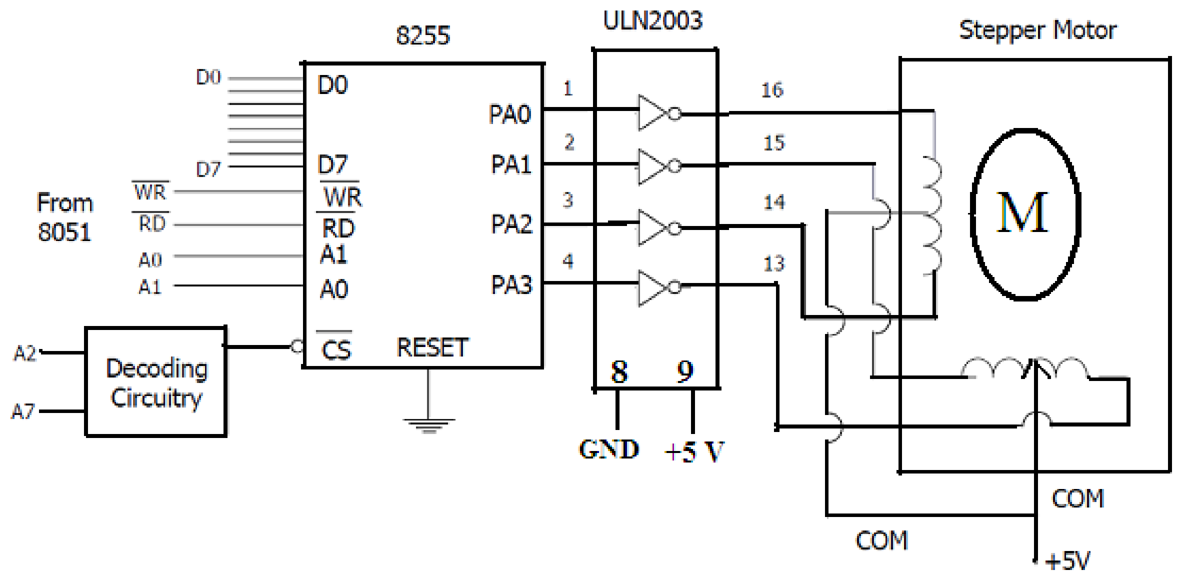
From the circuit it is clear that the ADC interfaced directly to the microcontroller. The Port1 is used as an input port which receives the digital data from the ADC.Port pins P2.5 and P2.6 are used for SOC and EOC operation.When the conversion is over the ADC will send an interrupt signal to the microcontroller through the pin P2.7 .Now the Microcontroller receives digital data through the Port1.This data after conversion to decimal data is displayed on the LCD module .

The assembly language program for ADC is given below .

```
              MOV P1 , 0FF H  ;  Make the port1 high and configure port1 as Input port

BACK:  CLR P2.6               ;  Generation of SOC pulse

            SETB P2.5          ;

LOOP    JB P2.7 ,  LOOP    ;  Wait for conversion, Is conversion over?

            CLR    P2.5          ;    Enable Read the  digital data

              MOV  A ,P1          ;  Read digital data through Port1

              SETB P2.5           ;   Disable read after read operation

              CALL DISPLAY  ;  Display the data on LCD module

              SJMP BACK        ; Continue the conversion process
```

Stepper motor Interfacing

A stepper motor is a device that translates electrical pulses into mechanical movement. The stepper motor rotates in steps in response to the applied signals. It is used in applications such as disk drives, dot matrix printers, plotters and robotics.It is mainly used for position control. Stepper motors have a permanent magnet called rotor (also called the shaft) surrounded by a stator . There are also steppers called variable reluctance stepper motors that do not have a PM rotor. The most common stepper motors have four stator windings that are paired with a center-tapped. This type of stepper motor is commonly referred to as a. four-phase or unipolar stepper motor. The center tap allows a change of current direction in each of two coils when a winding is grounded, thereby resulting in a polarity change of the stator.



ASSEMBLY LANGUAGE PROGRAM

```
Main    mov stepper, #0CH ;  move the  code to phase1 into the port
         acall delay
        mov stepper, #06H  ; phase II code
        acall delay
        mov stepper, #03H       ;Phase III code
        acall delay                ;Call delay subroutine program
         mov stepper, #09H      ;Phase IV code
        acall delay
        sjmp   Main
   CALL DELAY PROGRAM :
        mov r7,#4
       wait2:
       mov r6,#0FFH
      wait1:
      mov r5,#0FFH
```

```
wait:
djnz r5,wait
djnz r6,wait1
djnz r7,wait2
 ret
 end
```

8051-SERIAL COMMUNICATION :

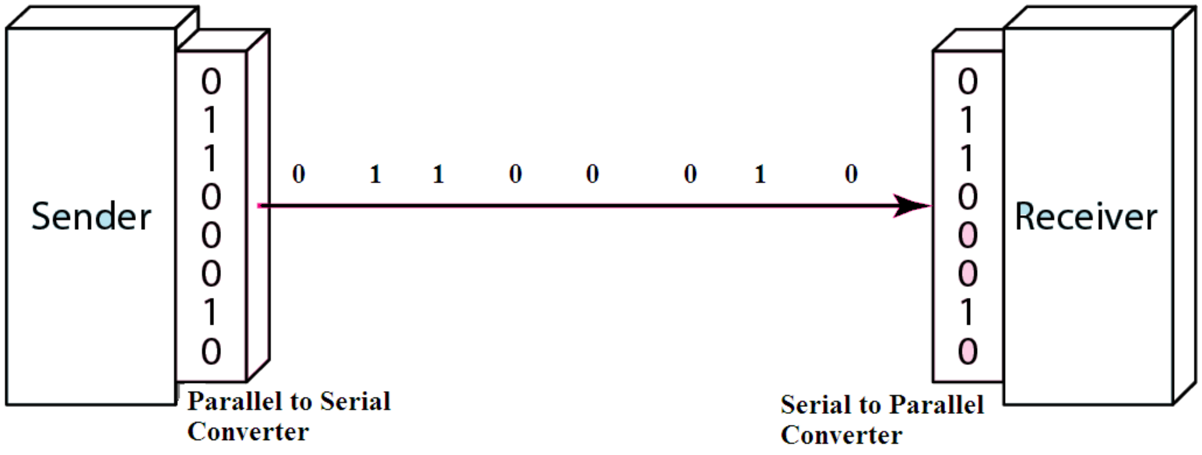
Basics of Serial communication

Data transfer between two electronic devices (Ex Between a computer and microcontroller or a peripheral device) is generally done in two ways

(i).Serial data Transfer and

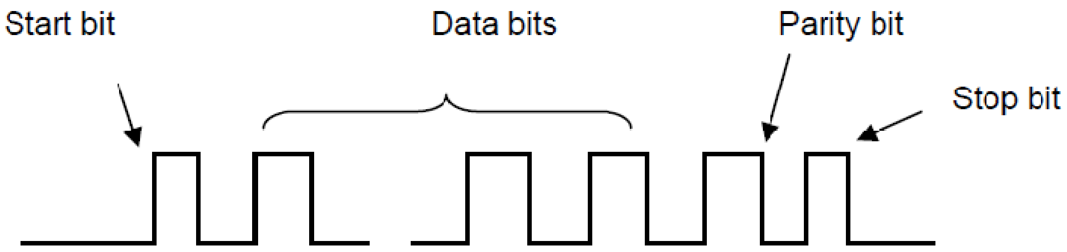
(ii).Parallel data Transfer

Serial communication uses only one or two data lines to transfer data and is generally used for long distance communication. In serial communication the data is sent as one bit at a time in a timed sequence on a single wire. Serial Communication takes place in two methods, Asynchronous data Transfer and Synchronous data Transfer.



Serial Data Transfer

Asynchronous data transfer *allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, special bits will be added to each word in order to synchronize the sending and receiving of the data. When a word is given to the UART for Asynchronous transmissions, a bit called the "Start Bit" is added to the beginning of each word that is to be transmitted. The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter.*



Serial data transmission

After the Start Bit, the individual bits of the word of data are sent .Here each bit in the word is transmitted for exactly the same amount of time as all of the other bits. When the entire data word has been sent, the transmitter may add a Parity Bit that the transmitter generates. The Parity bit may be used by the receiver to perform simple error checking. Then at least one Stop Bit is sent by the transmitter. If the

Stop Bit does not appear when it is supposed to, the UART considers the entire word to be corrupted and will report a Framing Error.

Baud rate is a measurement of transmission speed in asynchronous communication , it represents the number of bits/sec that are actually being sent over the serial link. The Baud count includes the overhead bits Start, Stop and Parity that are generated by the sending UART and removed by the receiving UART.

*In the **Synchronous data transfer** method the receiver knows when to "read" the next bit coming from the sender. This is achieved by sharing a clock between sender and receiver. In most forms of serial Synchronous communication, if there is no data available at a given time to transmit, a fill character will be sent instead so that data is always being transmitted. Synchronous communication is usually more efficient because only data bits are transmitted between sender and receiver, however it will be more costly because extra wiring and control circuits are required to share a clock signal between the sender and receiver.*

Devices that use serial cables for their communication are split into two categories.

1. DTE (Data Terminal Equipment). Examples of DTE are computers, printers & terminals.
2. DCE (Data Communication Equipment). Example of DCE is modems.

Parallel Data Transfer :

Parallel communication uses multiple wires (bus) running parallel to each other, and can transmit data on all the wires simultaneously. i.e all the bits of the byte are transmitted at a time. So, speed of the parallel data transfer is extremely high compared to serial data transfer. An 8-bit parallel data transfer is 8-times faster than serial data transfer. Hence with in the computer all data transfer is mainly based on Parallel data transfer. But only limitation is due to the high cost ,this method is limited to only short distance communications.

Differences between Serial data transfer and Parallel data transfer

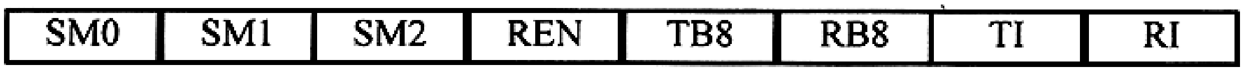
S.No	Serial Communication	Parallel Communication
1	Data is transmitted bit after the bit in a single line	Data is transmitted simultaneously through group of lines(Bus)

2	Data congestion takes place	No, Data congestion
3	Low speed transmission	High speed transmission
4	Implementation of serial links is not an easy task.	Parallel data links are easily implemented in hardware
5.	In terms of transmission channel costs such as data bus cable length, data bus buffers, interface connectors, it is less expensive	It is more expensive
6	No , crosstalk problem	Crosstalk creates interference between the parallel lines.
7	No effect of inter symbol interference and noise	Parallel ports suffer extremely from inter-symbol interference (ISI) and noise, and therefore the data can be corrupted over long distances.
8	The bandwidth of serial wires is much higher.	The bandwidth of parallel wires is much lower.
9	Serial interface is more flexible to upgrade , without changing the hardware	Parallel data transfer mechanism rely on hardware resources and hence not flexible to upgrade.
10	Serial communication work effectively even at high frequencies.	Parallel buses are hard to run at high frequencies.

SERIAL COMMUNICATION IN 8051 MICROCONTROLLER

The 8051 has two pins for transferring and receiving data by serial communication. These two pins are part of the Port3(P3.0 &P3.1) .These pins are TTL compatible and hence they require a line driver to make them RS232 compatible .Max232 chip is one such line driver in use. Serial communication is controlled by an 8-bit register called SCON register,it is a bit addressable register.

SCON (Serial control) register :



SM0	SM1	SM2	REN	TB8	RB8	TI	RI

SM0	SCON.7	Serial port mode selector
SM1	SCON.6	Serial port mode selector
SM2	SCON.5	Used for multiprocessor mode communication (not applicable for 8051)
REN	SCON.4	Receive enable. Set or cleared by making this bit either 1 or 0 foe enable /disable reception.
TB8	SCON.3	9th data bit transmitted in modes 2 and 3
RB8	SCON.2	9th data bit received in modes 2 and 3.it is not used in mode 0 & mode 1.If SM2 = 0 RB8 is the stop bit .
TI	SCON.1	Transmit interrupt flag
RI	SCON.0	Receive interrupt flag.

M0 , SM1 : These two bits of SCON register determine the framing of data by specifying the number of bits per character and start bit and stop bits. There are 4 serial modes.

SM0	SM1	
0	0	: Serial Mode 0
0	1	: Serial Mode 1, 8 bit data, 1 stop bit, 1 start bit
1	0	: Serial Mode 2

REN (Receive Enable) also referred as SCON.4. When it is high,it allows the 8051 to receive data on the RxD pin. So to receive and transfer data REN must be set to 1.When REN=0,the receiver is disabled. This is achieved as below

```
        SETB   SCON.4
&       CLR    SCON.4
```

TI (Transmit interrupt) is the D1 bit of SCON register. When 8051 finishes the transfer of 8-bit character, it raises the TI flag to indicate that it is ready to transfer another byte. The TI bit is raised at the beginning of the stop bit.

RI (Receive interrupt) is the D0 bit of the SCON register. When the 8051 receives data serially ,via RxD, it gets rid of the start and stop bits and places the byte in the SBUF register. Then it raises the RI flag bit to indicate that a byte has been received and should be picked up before it is lost. RI is raised halfway through the stop bit.

Communication through RS232

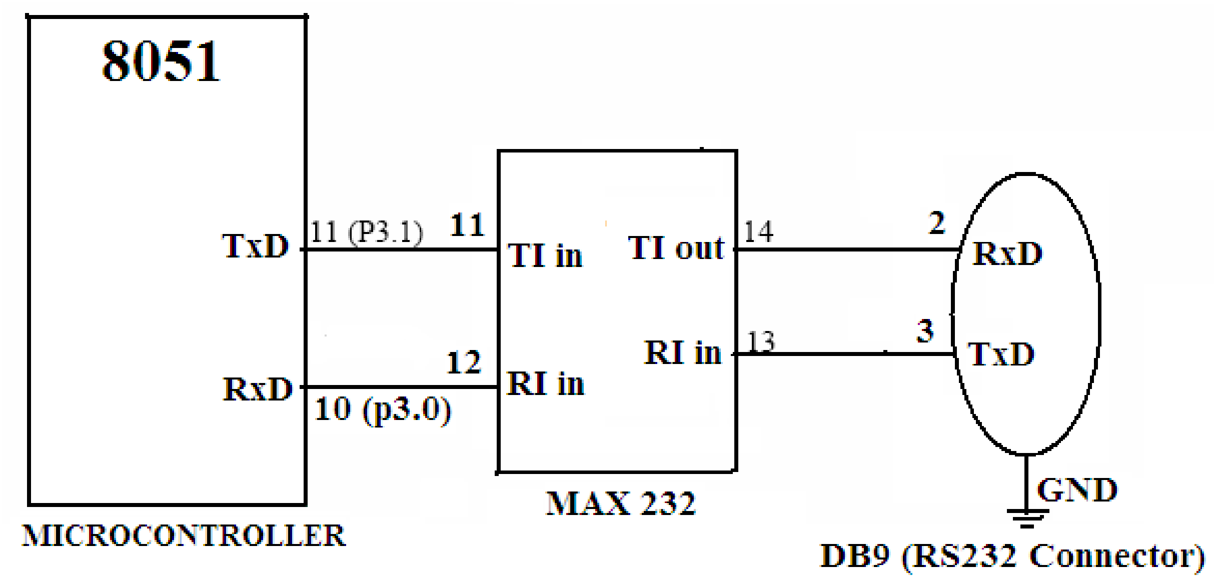
A personal computer has a serial port known as communication port or COM Port used to connect a modem for example or any other device, there could be more then one COM Port in a PC. Serial ports are controlled by a special chip called UART (Universal Asynchronous Receiver Transmitter).

RS 232 standard describes a communication method where information is sent bit by bit on a physical channel. The **RS stands for Recommended Standard**.The information must be broken up in data words. The length of a data word is variable.

It is one of the popularly known interface standard for serial communication between DTE & DCE. This RS-232-C is the commonly used standard when data are transmitted as voltage .This standard was first developed by Electronic industries association(EIA). For the RS-232C, a 25 pin D type connector is used . DB-25P male and DB-25S female. RS-232 standard was first introduced in 1960's by Telecommunications Industry Association(TIA).

Interfacing the 8051 Microcontroller to PC:

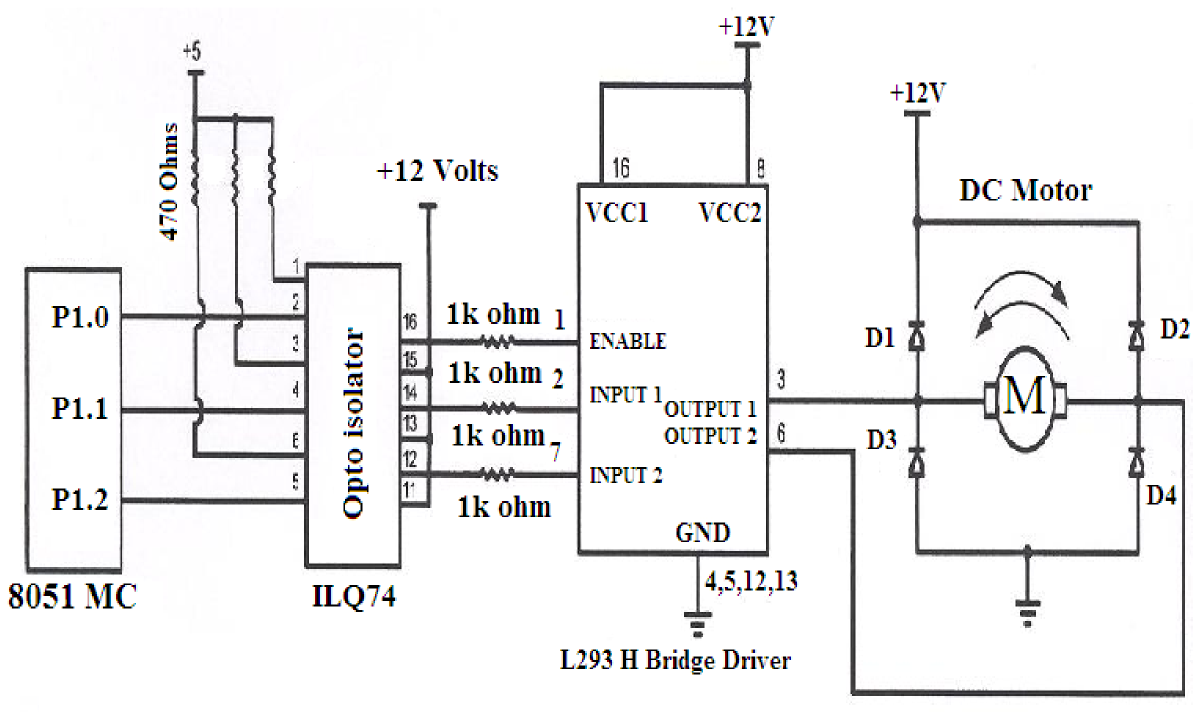
As the RS-232 standard is developed earlier to TTL devices ,a USART such as 8251 is not directly compatible with these signal levels .Because of this ,voltage transistors called line drivers and line receivers are used to interface TTL logic with RS-232 signals . The line driver MC 1488 is used to convert RS-232 to TTL.The microcontroller is connected to the PC using the DB9 connector.



The TxD and Rx D pins are connected to the TI in and RI in pins of the MAX 232 IC and the TI out and RI in pins of the MAX IC are connected to the RxD and TxD pins of the DB9 connector as shown in the interface diagram.

INTERFACING DC MOTOR- 8051

A DC motor runs with the help of Direct Current. It produces torque by using both electricity and magnetic fields. The DC motor has rotor, stator, field magnet, brushes, shaft, commutator. The DC motor requires more current to produce initial torque than in running state.Interfacing the DC motor directly to 8051 microcontroller is not possible. Because the DC motor uses large current (200-300mA in small DC motors) to run. When this current flow into the 8051 microcontroller, the IC will get damaged. Therefore we use a driving circuit with an opto isolator and a L298 Dual H-Bridge driver. The opto-isolator provides additional protection to the microcontroller.



Continuous, sustained operation of the motor will cause the L293 Dual H-Bridge driver to overheat. So,a suitable heat sink must be used.

Assembly Language program

ORG	0000H		Remarks
MAIN	CLR	P1.0	9
	CLR	P1.1	
	CLR	P1.2	
	SETB	P2.7	
MONITOR			
	SETB	P1.0	Enable the H-bridge driver
	JNB	P2.7 CLOCKWISE	
	CLR	P1.1	01 is for Counter clockwise
	SETB	P1.2	
	SJMP	MONITOR	
CLOCKWISE	SETB	P1.1	10 is for clockwise
	CLR P1.2		
	SJMP	MONITOR	

INTERFACING DAC -8051 MICROCONTROLLER

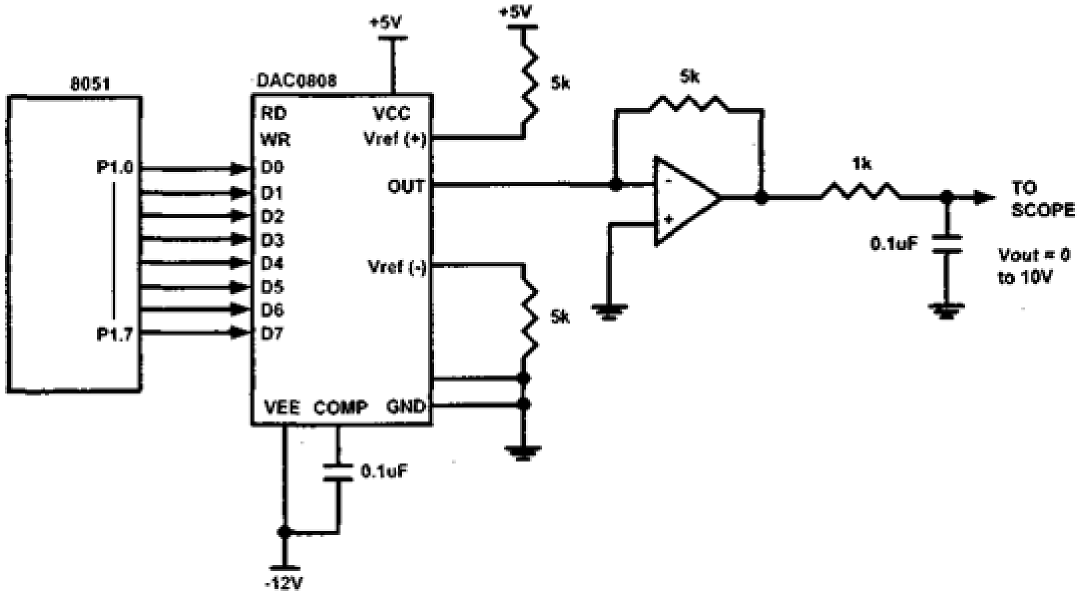
The DAC 0800 is a simple monolithic 8-bit D/A converter. It has fast settling time of 100ns. It can be directly interfaced to TTL, CMOS, PMOS and others. It operates at 4.5V to +18V supply. The number of data bit inputs decides the resolution of the DAC since the number of analog output levels is equal to 2", where n is the number of data bit inputs. Therefore, an 8-input DAC such as the DAC0808 provides 256 discrete voltage (or current) levels of output.

The interfacing circuit is shown below. port 1(8 bits of the microcontroller is connected to the input data lines of DAC-08.The reference current is determined by the resistor R_1 and the reference voltage V $_{ref}$. The resistor R_2 is generally equal to R1 to match the input impedance of reference source. The output (taken from pin number 4 is observed either on a digital multimeter or on a cathode ray oscilloscope.

The output current Io is calculated as follows:

$$I_o = V_{ref}/R_1[A_0/2 + A_1/4 + A_2/8 + \ldots + A7/256]$$

The output voltage Vo is obtained as follows: $V_o = I_o * R_1$



Assembly Language Program

```
MOV    A, #DATA*    ; (A) = #Data
```

```
START :    MOV     90H, A        ; (port -1) = (A)

           INC A

           LJMP    START         ;  Repeat
```

---------------------XXX---------------------

REFERENCE : Muhammed Ali Mazidi, Janice Gillies Pie Mazidi, "The 8051 Microcontroller and Embedded Systems"– Pearson EducationAsia.