UNIT-1 ~~&2~~

Pin Configuration of 8086 Microprocessor:-

→ It is a 16-bit microprocessor.

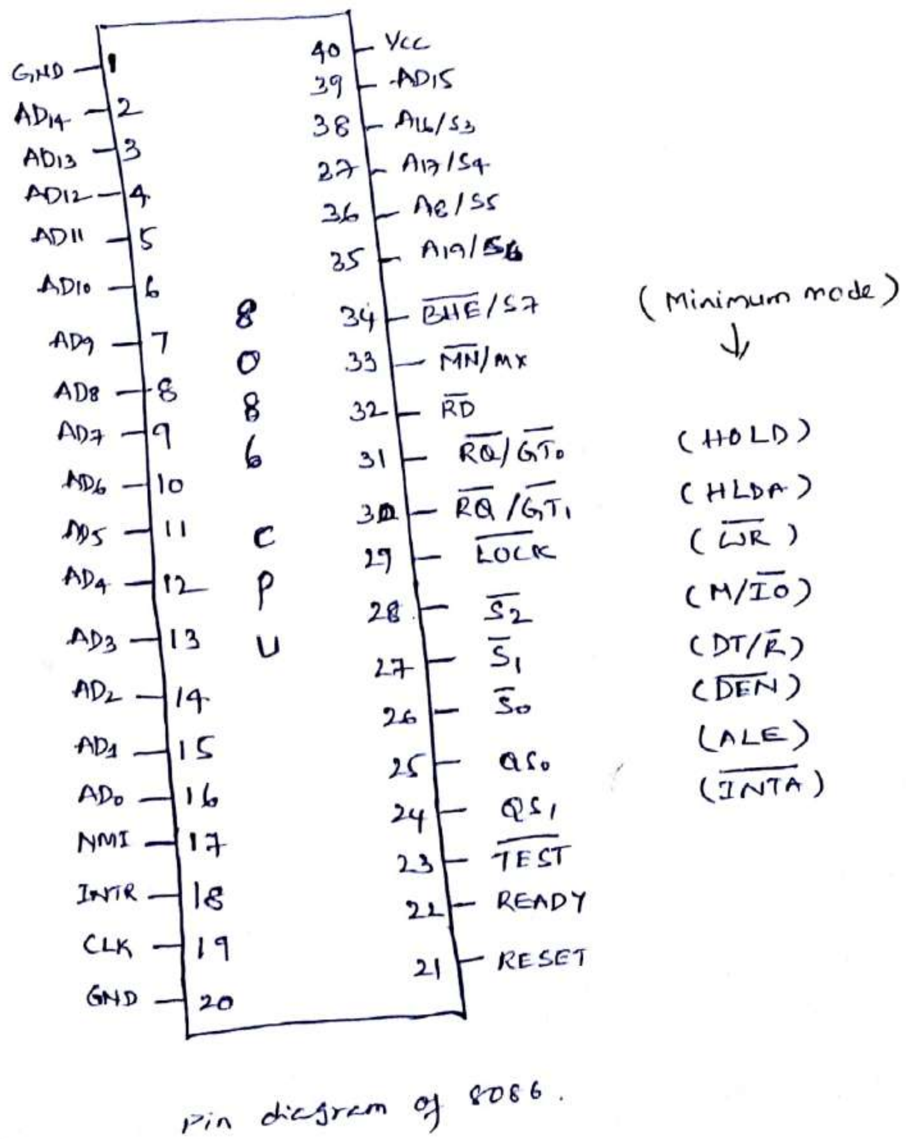
→ 8086 has a 20-bit address bus can access upto 2^{20} memory locations (1MB)

→ It can Support up to 64K I/O ports.

→ It has multiplexed Address and databus AD0-AD15 and A16-A19.

→ It requires single phase Clock with 33% duty Cycle to provide Internal timing.

→ It required +5V power Supply.



Pin diagram of 8086.

Signal Description of 8086 :-

→ The 8086 operates in single processor (or) multiprocessor configuration to achieve high performance.

→ The 8086 Signals can be categorized in three groups. The first are the signal having common functions in minimum as well as maximum mode.

→ The second are the signals which have special functions for minimum mode and third are the signals having special functions for maximum mode.

→ The following signal descriptions are common for both modes.

AD15 - AD0 : These are the time multiplexed memory I/O Address and data lines.

- Address remains on the lines during T1 state, while the data is available on the data bus during T2, T3, Tw and T4.
- These lines are active high and float to a tri-state during Interrupt acknowledge and local bus hold acknowledge cycles.

A19/S6, A18/S5, A17/S4, A16/S3 : These are the time multiplexed address and status lines.

- During T1 these are the most significant address lines for memory operations.
- During I/o operations, these lines are low. During memory (or) I/o operations, status Information is available on those lines for T2, T3, ~~T4~~ Tw and T4.
- The status of the Interrupt enable flag bit is updated at the beginning of each clock cycle.
- The S4 and S3 combinedly indicate which segment register is presently being used for memory accesses as in below fig.

S_4	S_3	Indication
0	0	Alternate Data
0	1	Stack
1	0	Code
1	1	Data.

→ These lines float to tri-state of during the Local bus hold Acknowledge. The status Line S6 is always low.

→ The address bit are Separated from the status bit using Latches controlled by the ALE Signal.

$\overline{BHE}$/S7:

→ The bus high enable is used to indicate the transfer of data over the higher order (D15-D8) Data bus.

→ It goes low for the data transfer over D15-D8 and is used to derive chip selects of odd address memory bank (or) peripherals.

→ $\overline{BHE}$ is low during T1 for read, write and interrupt Acknowledge cycles, whenever a byte is to be transferred on higher byte of data bus.

→ The status Information is available during T2, T3 and T4.

$\overline{BHE}$	A_0	
0	0	Whole word
0	1	upper byte from (or) to even address
1	0	Lower byte from (or) to even address
1	1	None.

$\overline{RD}$ Read: This Signal on low indicates the peripheral that the processor is performing memory (or) I/O read operation.

READY :- This is the Acknowledgement from the slow device (or) memory that they have completed the data transfer.

INTR :- This is a triggered input. This is sampled during the last clock cycles of each instruction to determine the availability of the request. If any interrupt request is pending, the processor enters the Interrupt acknowledge cycle.

$\overline{TEST}$: This Input is Examined by 'WAIT' Instruction. If the TEST pin goes low, execution will continue, else the ~~the Input~~ processor remains in an idle state.

CLK : The clock Input provides the basic timing for processor operation and bus control activity.

MN/$\overline{MX}$: The logic level at This pin decides whether the processor is to operate in either minimum (or) maximum mode.

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

M/$\overline{IO}$: This is a status Line logically equivalent to S2 in maximum mode. When It is low, it indicates the CPU is having an I/O operation, and When It is high, It indicates the CPU is having a memory operation.

$\overline{INTA}$: This signal is used as a read strobe for Interrupt acknowledge cycles. i.e. When It goes low, the processor has accepted the Interrupt.

ALE : This output signal indicates the availability of the valid address on the address/data lines, and is connected to latch enable input of latches.

DT/$\overline{R}$: this output is used to decide the direction of data flow through the transreceivers (Bidirectional Buffers) When the processor sends out data, This signal is high and when the processor is receiving data, This signal is low.

$\overline{DEN}$: This signal indicates the availability of valid data over the Address/data lines.

It is used to enable the transreceivers to separate the data from the multiplexed Address/data signal.

HOLD, HLDA : When the HOLD line goes high, it indicates to the processor that another master is requesting the bus access.

- the processor, after receiving the HOLD request, issues the hold acknowledge signal on HLDA pin, ~~is the~~
- At the same time, the processor floats the local bus and control lines.
- If the DMA request is made while the CPU is performing a memory (or) I/O cycle, it will release the local bus during T4 provided.

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

$\overline{S2}, \overline{S1}, \overline{S0}$ - These are status lines which reflect the type of operation, being carried out by the processor.

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Indication
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O port
0	1	0	Write I/O port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read memory
1	1	0	Write memory
1	1	1	Passive.

$\overline{LOCK}$: This output pin indicates that other system bus master will be prevented from gaining the system bus, while the Lock signal is low.

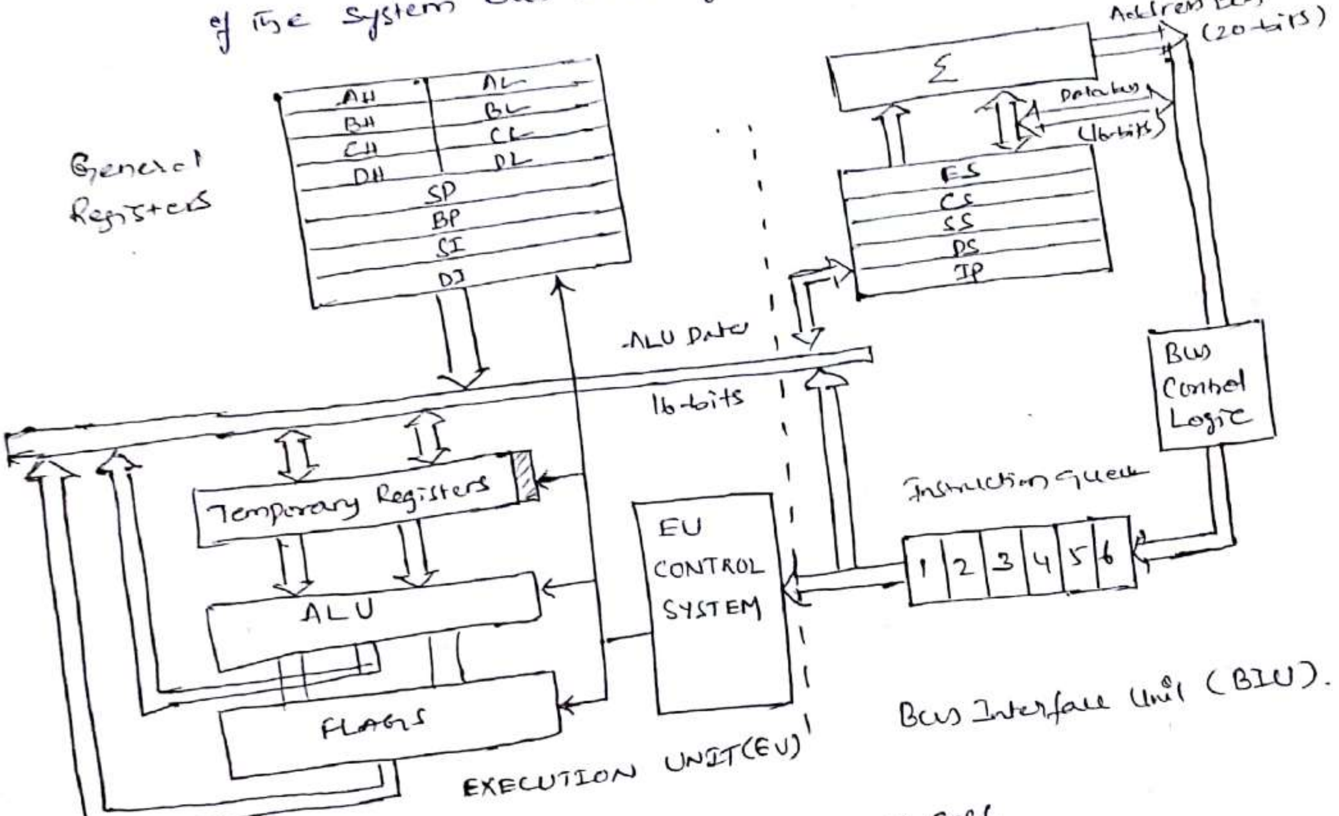
QS1, QS0 : These lines give information about the status of the Code prefetch queue.

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

$\overline{RQ}/\overline{GT0}$, $\overline{RQ}/\overline{GT1}$: These pins are used by the other local bus master in maximum mode, to force the processor to realease the Local bus at the end of the processor current bus cycle.
- each of the pin is bidirectional with $\overline{RQ}/\overline{GT0}$ having higher priority than $\overline{RQ}/\overline{GT1}$.

Internal Architecture of 8086 :-

- ~~8086~~ 8086 has two blocks BIU and EU.
- The BIU performs all bus operations such as Instruction fetching, reading and writing operands for memory and calculating the addresses of the memory operands. The Instruction bytes are transferred to the Instruction queue.
- EU executes Instructions from the Instruction system byte queue.
- Both units operate Asynchronously to give the 8086 an overlapping Instruction fetch and Execution mechanism which is called as pipelining. This results in efficient use of the system bus and system performance.



Block Diagram of 8086.

1. Bus Interface unit :-

- It provides a full 16-bit bidirectional data bus and 20-bit address bus.
- Bus Interface unit is responsible for performing all External bus operations.

Specifically it has functions like Instruction fetch, Instruction ~~[illegible]~~ ~~[illegible]~~ queuing, operand fetch and storage, Address relocation and Bus Control.

- The BIU uses a mechanism known as an Instruction stream queue. This queue permits pre-fetch of up to six bytes of Instruction code.
- These pre fetching Instructions are held in It's FIFO queue. With its 16-bit data bus, The BIU fetches two Instruction bytes in a single memory cycle.
- The EU accesses The queue from The output end.
- The BIU also contains a dedicated adder which is used to generate The 20-bit physical address, That is output on The address bus. This address is formed by adding an appended 16-bit Segment-address and a 16-bit offset address.
- The BIU is also responsible for generating bus control Signals such as Those for memory read (or) write and I/O read (or) write

2. Execution unit :-

The Execution unit is responsible for decoding and executing all Instructions.

- The EU Extracts Instructions from The top of The queue in The BIU, decodes Them, generates operands If necessary, passes Them to The BIU and requests It to perform read (or) write bus cycles to memory (or) I/O and perform The operation Specified by The Instruction on the operands
- During The execution of The Instruction, The EU Tests The Status and control flags and updates Them based on The results of executing The Instruction.

✓ When the EU executes a branch (or) jump Instruction, It transfers control to a location corresponding to another set of sequential Instructions, whenever this happens, the BIU automatically resets the queue and then begins to fetch Instructions from this new location to refill the queue.

General Bus operation of 8086 microprocessor:-

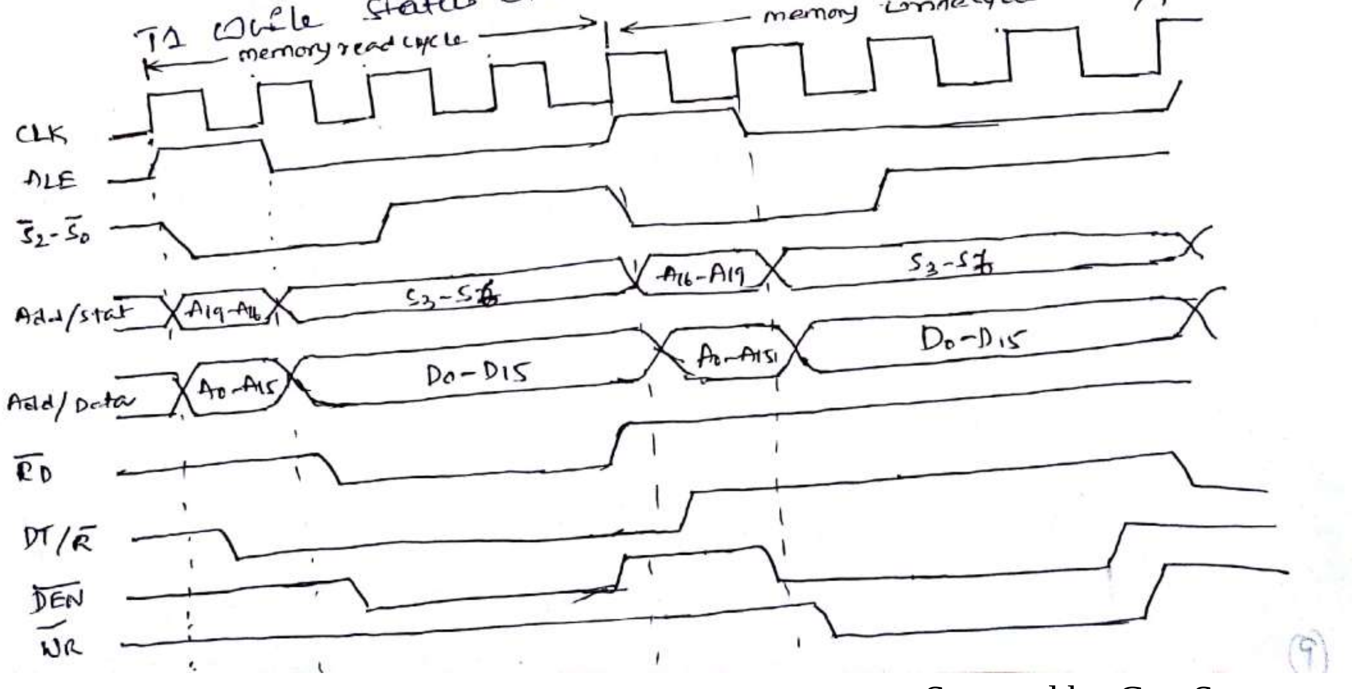
✓ The 8086 has a combined address and data bus commonly referred as a time multiplexed address and data bus.

✓ The bus can be demultiplexed using a few Latches and Transreceivers, whenever required.

✓ Basically, all the processor bus cycles consists of at least four clock cycles, these are referred to as T1, T2, T3, T4. The address is transmitted by the processor during T1. It is present on the bus only for one cycle.

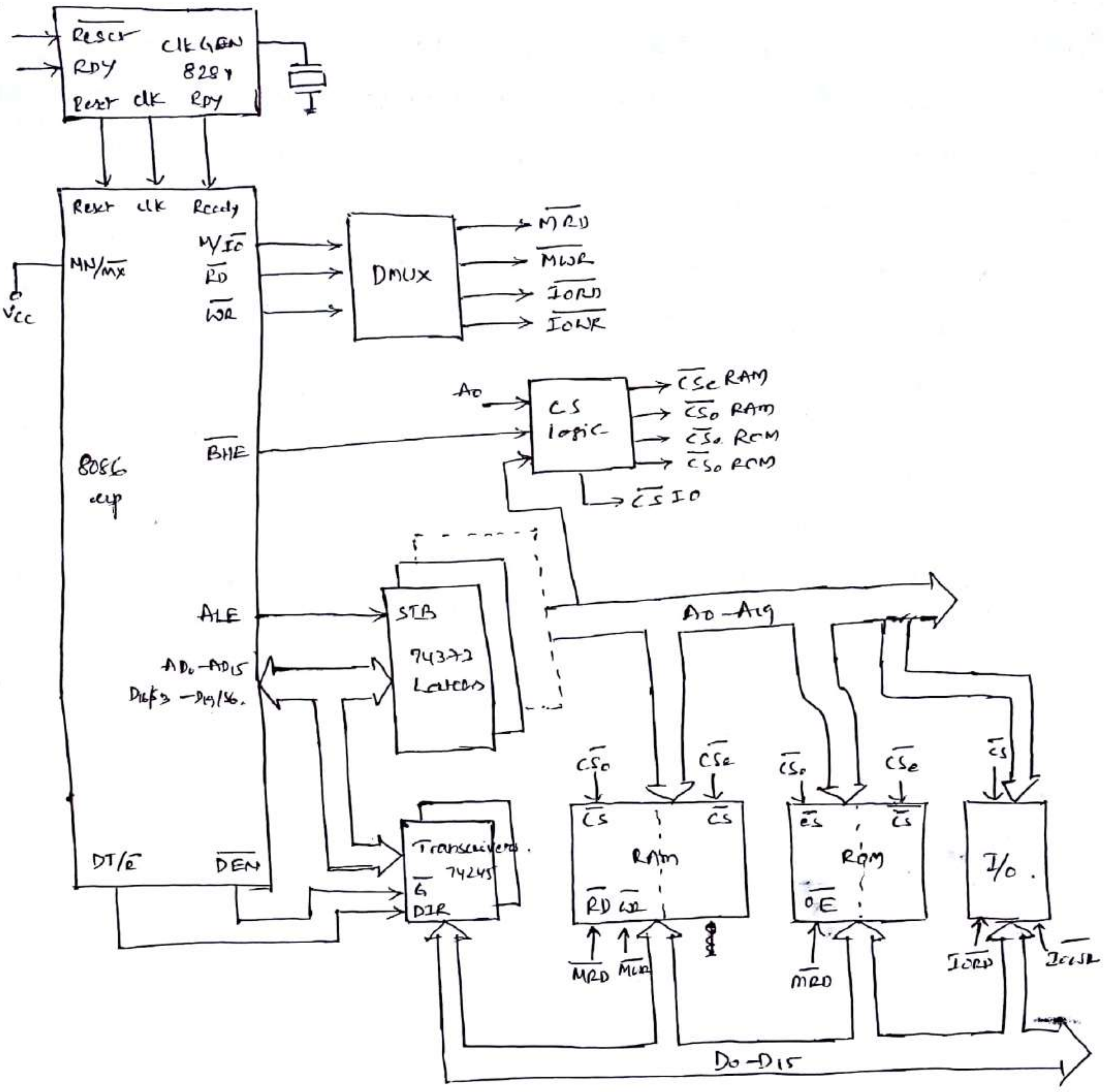
✓ The negative edge of the ALE pulse is used to separate the address and the data (or) status Information. In maximum mode, the status lines S0, S1 and S2 are used to indicate the type of operation.

✓ The status bits S3 to S7 are multiplexed with higher order address bits and the BHE signal. Address is valid during T1 while status bits S3 to S7 are valid during T2 through T4



Minimum mode operation of 8086 microprocessor:-

- ✓ The microprocessor 8086 is operated in minimum mode by strapping It's MN/MX pin to logic 1.
- ✓ In This mode, all the control signals are given out by the microprocessor chip Itself. There is a single microprocessor in the minimum mode system.
- ✓ The remaining components in the system are Latches, trans-receivers, clock generator, memory, and I/o devices. Some type of chip selection logic may be required for selecting memory (or) I/o devices, depending upon the address map of the system.
- ✓ Latches are generally buffered output D-type flip-flops like 74LS373 (or) 8282. They are used for separating valid address from the multiplexed address/data signals and are controlled by the ALE signal generated by 8086.
- ✓ Transreceivers are the bidirectional Buffers and some times they are called as data amplifiers, They are required to separate the valid data from the time multiplexed address/data signals. They are controlled by two signals, DEN / DT/R.
- ✓ usually, EPROM are used for monitor storage, while RAm for users program storage. A system may contain I/o devices.
- ✓ The working of the minimum mode configuration system can be described in terms of the timing diagrams, the first is the timing diagram for read cycle and the second is the timing diagram for write cycle.



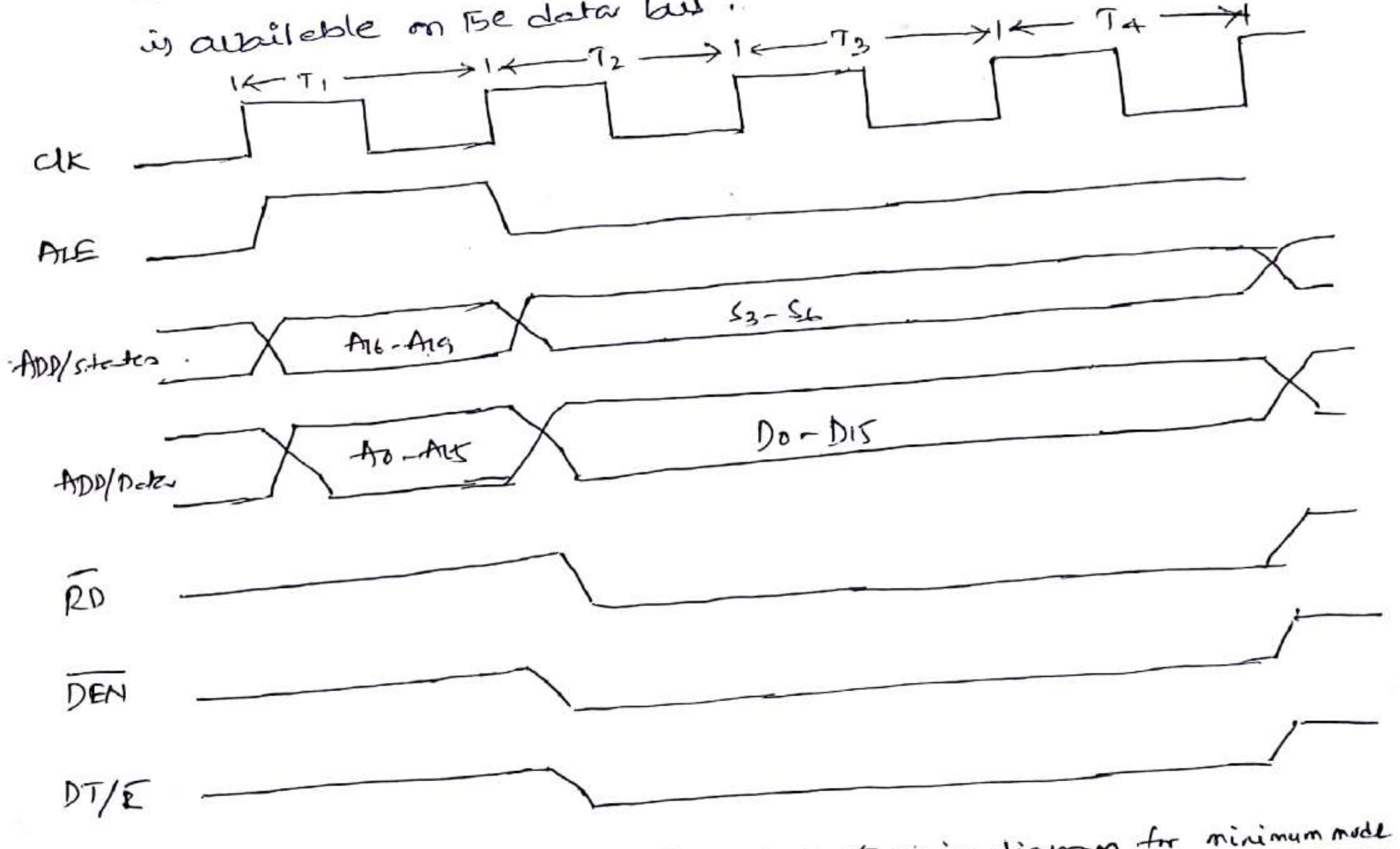
Minimum mode 8086 System.

✓ The read cycle begins in T1 with the assertion of address Latch enable (ALE) Signal and also M/IO Signal. During the negative going edge of this signal, the valid Address is latched on the local bus.

✓ The BHE and A0 signals address Low, high (or) both bytes. From T1 to T4, the M/IO Signal indicates a memory (or) I/o operation.

✓ At T2, the address is removed from the local bus and is sent to the output. The bus is then tristated. The read (RD) Control signal is also activated in T2.

✓ The read (RD) Signal causes the address device to enable It's data bus drivers. After RD goes low, the valid data is available on the data bus.



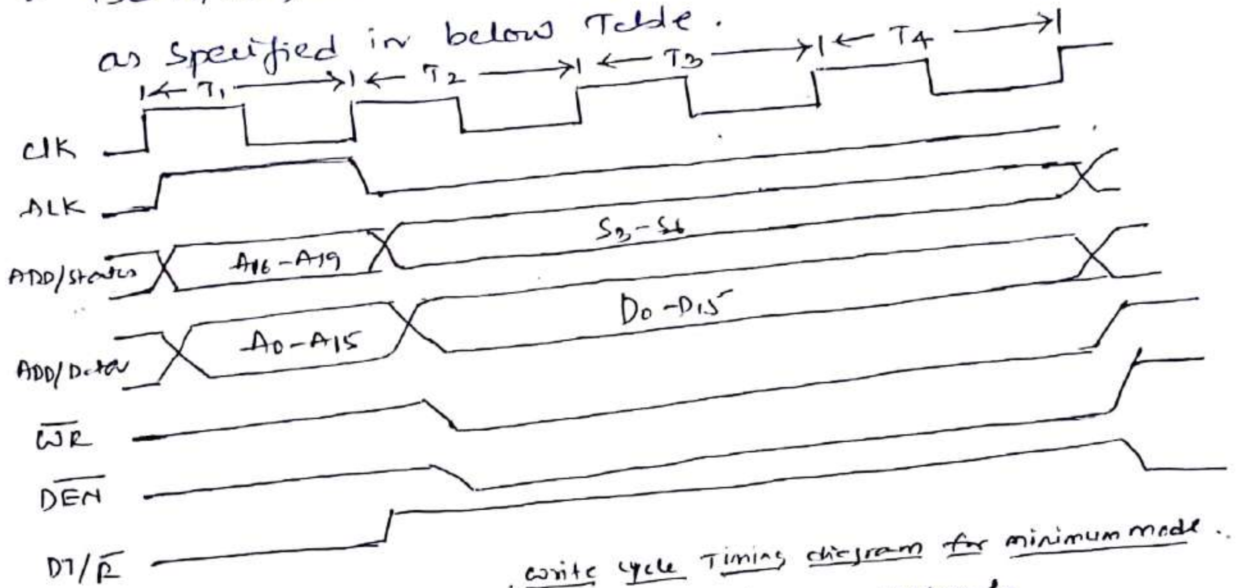
Read cycle ~~for~~ Timing diagram for minimum mode

✓ The write cycle also begins with the assertion of ALE and the emission of the address. The M/IO Signal is again asserted to indicate a memory (or) I/o operation. In T2, After sending the address in T1, the processor sends the data to be written to the addressed location.

✓ The data remains on the bus until middle of T4 State. ~~The~~ The WR becomes active at the beginning of T2.

✓ BHE and A0 Signals are used to select the proper byte (or) bytes of memory (or) I/O word to be read (or) write.

✓ The M/IO, RD and WR signals indicate the type of data Transfer as specified in below Table.



Write cycle Timing diagram for minimum mode.

Maximum mode operation of 8086 microprocessor:-

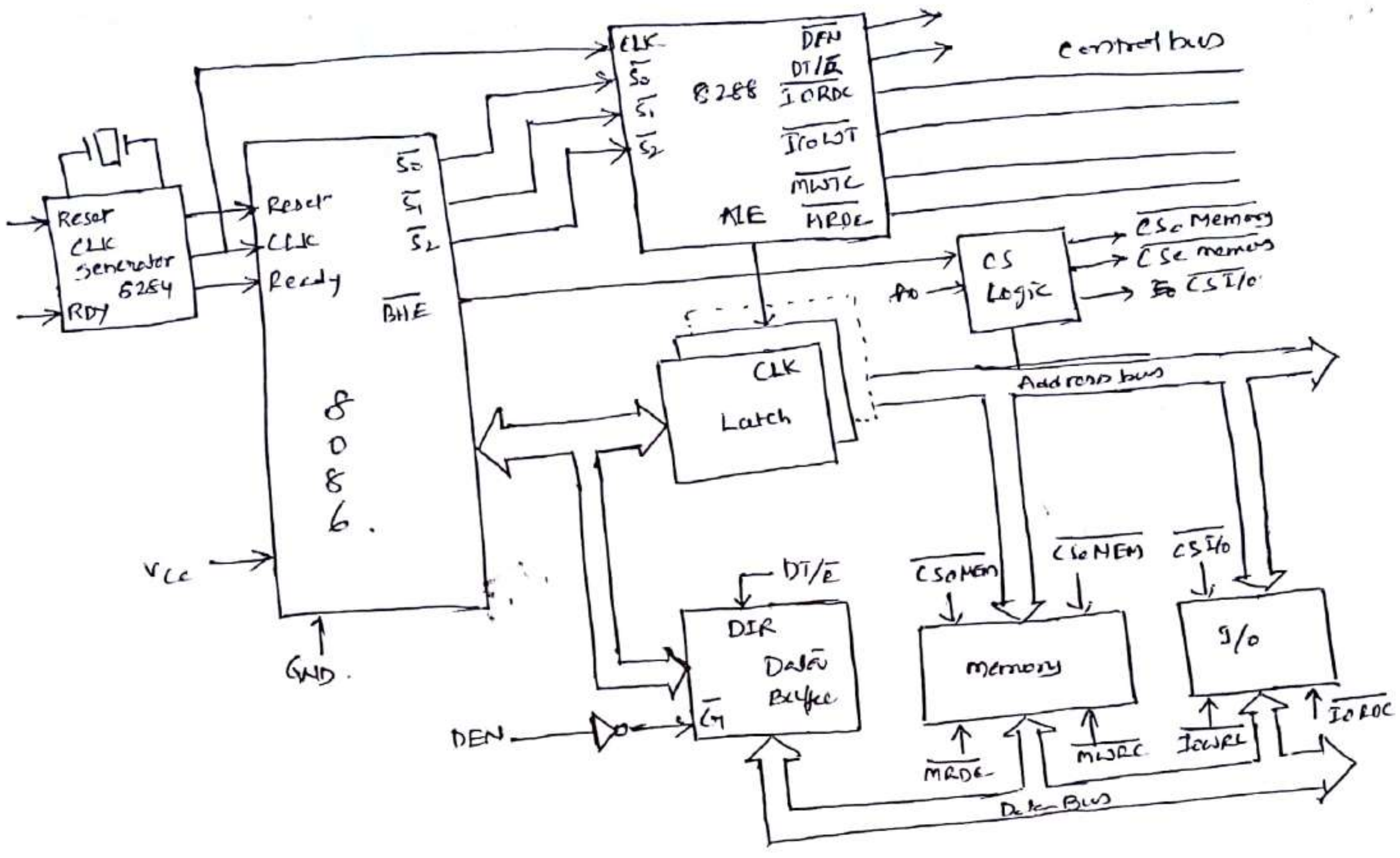
✓ In the maximum mode, the 8086 is operated by strapping the MN/MX pin to ground.

✓ In this mode, processor drives the status signals S2, S1, S0. another chip called bus controller drives the control signal using this status information.

✓ In the maximum mode, there may be more than one micro-processor in the system configuration.

✓ The Basic function of the bus controller chip IC 8288, is to derive control signals like RD and WR, DEN, DT/R, ALE etc. using the Information by the processor on the status lines.

✓ The Bus controller chip has Input lines S2, S1, S0 and CLK. These Inputs to 8288 are driven by CPU.



Maximum mode 8086 System.

- ✓ It drives the outputs ALE, DEN, DT/R, MRDC, MWTC, AMWC, IORC, IOWC and AIOWC. The AEN, IOB and CEN pins are specially useful for multiprocessor systems.
- ✓ If IOB is grounded, It acts as master cascade enable to control cascade 8259A, else It acts as peripheral data enable used in the multiple bus configurations.
- ✓ INTA pin used to issue two Interrupt acknowledge pulses to the Interrupt controller (or) to an Interrupting device.
- ✓ IORC, IOWC are I/o read command and I/o write command signals respectively. These signals enable an I/o Interface to read (or) write the data from (or) to the address port.
- ✓ The MRDC, MWTC are memory read command and memory write command signals respectively and may be used as memory read (or) write signals.

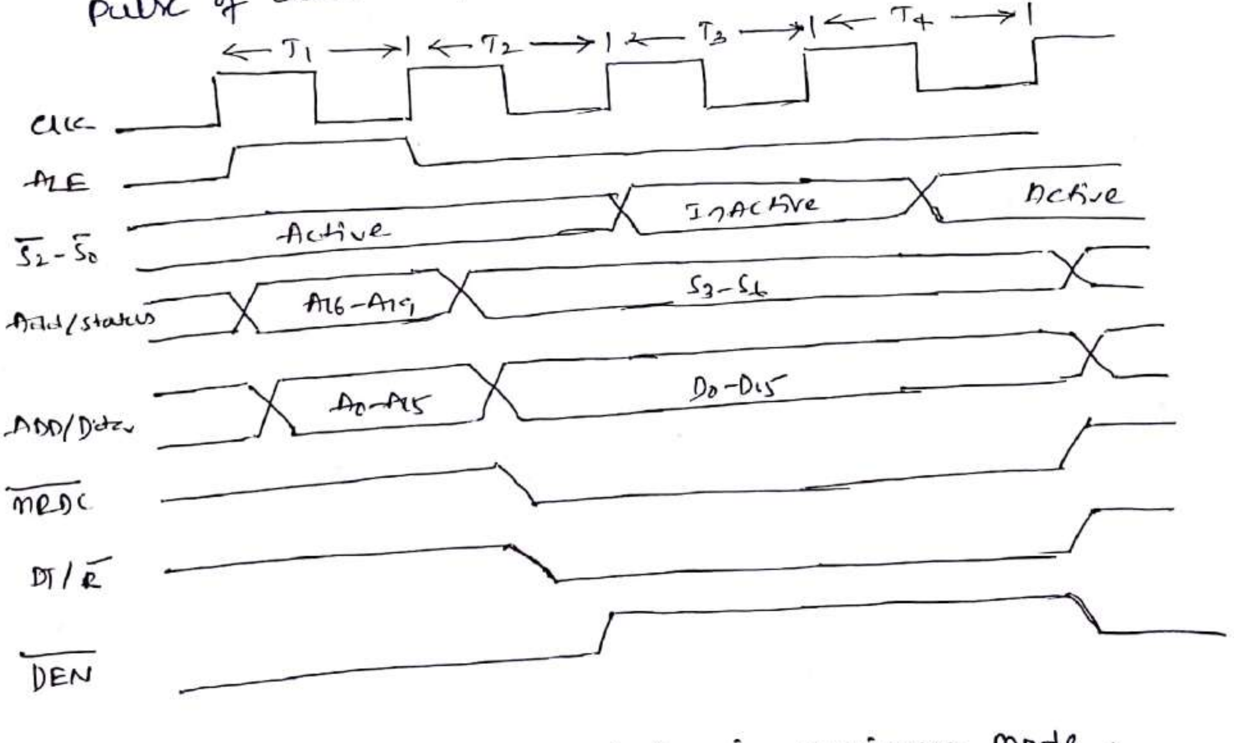
✓ ~~the~~ All these command signals instructs the memory to acept (or) send data from (or) to the bus.

✓ Here the only difference between in timing diagram between minimum mode and maximum mode is the status signals used and the available control and advanced command signals.

* S0, S1, S2 are set at the beginning of bus cycle. 8288 bus controller will output a pulse as on the ALE and apply a required signal to It's DT/R pin during T1.

* In T2, 8288 will set DEN = 1, thus enabling transceivers, and for an Input it will activate MRDC (or) IORC.

Timing for RQ/GT signals:-
the request/grant response sequence contains a series of three pulses. the request/grant pins are checked at each rising pulse of clock input.



Memory Read Timing in maximum mode.

Internal Registers of 8086 microprocessor:-

- ✓ The 8086 has four groups of the user accessible Internal Registers. They are the Instruction pointer, four data registers, four pointers and index registers, four segment registers.
- ✓ 8086 has a total of fourteen 16-bit registers, including a 16-bit register called the status register. (or) flag register.
- ✓ There are four different 64KB segments for Instructions, stack, data and Extra data.

Code Segment (CS):- is a 16-bit register containing address of 64KB segment with processor Instructions. The processor uses CS segment for all accesses to Instructions referenced by Instruction pointer (IP) register.

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) register is located in the stack segment.

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.

Accumulator:- register consists of two 8-bit registers AL & AH, which can be combined together and used as a 16-bit register AX. Accumulator can be used for I/O operations and string manipulation.

Base register:- Consists of two 8-bit registers BL and BH, which can be combined together and used as a 16-bit register BX. It is usually contains a data pointer used for based, based indexed (or) register indirect Addressing.

Addressing Modes:- Addressing mode refers to the mechanism by which the operands are specified for an operation. Depending on the type of operands used, there are 7 addressing modes available.

1. Immediate mode:- In this mode, the instruction contains 8-bit (or) 16-bit Immediate data along with the mnemonic. Data can be moved to 8-bit (or) 16-bit register, or memory location using the address (or) register.

Ex:- MOV AX, 1234 H
MOV [2000H], 25
~~MOV~~ AND AL, 3FH.

2. Register mode: In this mode, the operands are available in general purpose registers. Instruction specifies the Register Name.

Ex:- MOV CL, DL
ADD BL, AL
XOR CX, DX.

3. Direct mode: Here, the Instruction contains 16-bit offset (Effective Address) EA of the memory Location. the physical address (PA) is calculated by 8086 using EA and segment register, and then the operand will be fetched from memory.

→ DS is Default register.

$$PA = EA + (DS * 10)$$

Ex:- MOV BL, NUM
MOV [3455 H], AL.

4. Indirect modes: 8086 provides many "Indirect addressing modes" (where one operand is in memory) which are extensivly used in programming.

a. Register indirect mode: In this mode, the Instruction contains a 16-bit register name which contains the EA. Using this EA, PA is calculated. Default segment register for memory is DS.

Ex:- MOV AL, BYTE PTR [BX]
AND CX, WORD PTR [BX].

b. Register relative mode (or) indexed mode: In this mode, the Instruction contains a 16-bit name (which contains an address) and a signed displacement. Adding the register contents with displacement gives the EA.

$$PA = EA + (DS * 10H)$$

$$EA = \begin{Bmatrix} (BX) \\ (BP) \\ (SI) \\ (DI) \end{Bmatrix} + \left\{ \begin{matrix} \text{8-bit displacement (sign-Extended)} \\ \text{(or)} \\ \text{16-bit Signed displacement} \end{matrix} \right\}$$

Ex:- MOV AL, BYTE PTR 50 [BX]
DEC WORD PTR [DI + 30].

Count register :- consists of two 8-bit registers CL and CH, which can be combined together and used as a 16-bit register CX. count register can be used in loop, shift/rotate Instructions and as a counter in string manipulation.

Data register :- consists of two 8-bit registers DL and DH, which can be combined together and used as a 16-bit register DX. Data register can be used as a port number in I/o operations In Integer 32-bit multiply and divide Instructions the DX register contains high-order word of the resulting number.

* the following registers are both general and index registers :-

Stack pointer (SP) is a 16-bit register pointing to program stack

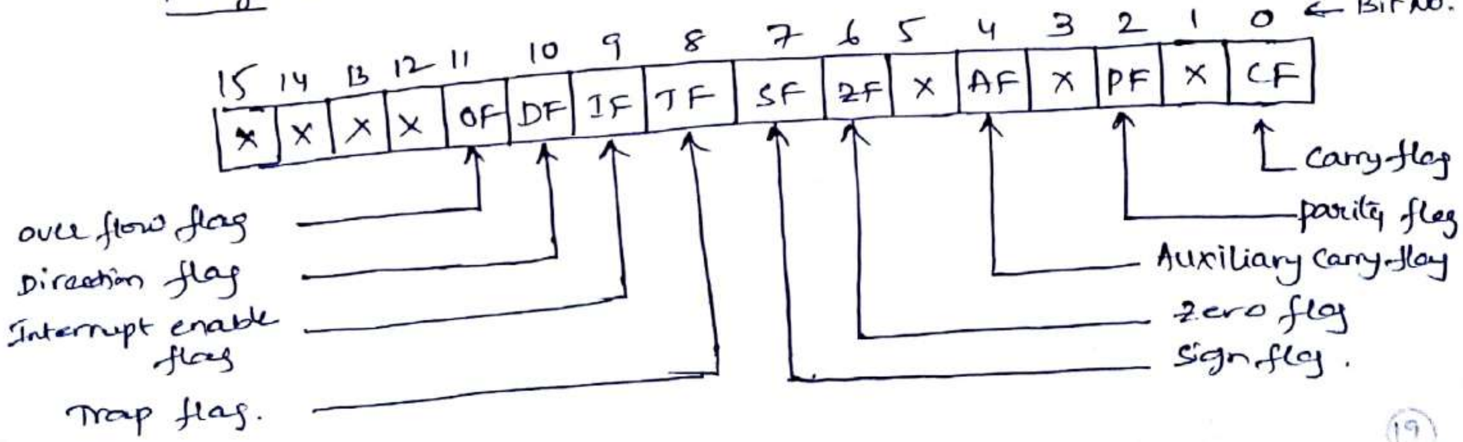
Base pointer (BP): is a 16-bit register pointing to data in stack segment. BP register is usually used for based, based indexed (or) register indirect addressing.

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 address in string manipulation Instructions.

Destination index (DI) :- is a 16-bit register. DI is used for register indirect addressing, indexed, based indexed and register indirect addressing as well as a destination data address in String manipulation Instructions.

Instruction pointer (IP) is a 16-bit register:

Flags is a 16-bit register Containing 9 one bit flags.



Overflow flag (OF) :- Set If the result is too large positive number, or is too small negative number to fit into destination operand.

Direction flag (DF) :- If set then string manipulation instructions will auto-decrement index registers. If cleared then the index registers will be auto-incremented.

Interrupt-enable flag (IF) :- Setting this bit enables maskable Interrupts.

Single-step Flag (TF) :- If set then single-step interrupt will occur after the next Instruction.

Sign Flag (SF) :- set If the most significant bit of the result is set.

Zero flag (ZF) :- Set If the result is zero.

Auxiliary Carry Flag (AC) :- Set If there was a carry from (or) borrow to bits 0-3 in the AL register.

parity Flag (PF) :- set If parity (the number of "1" bits) in the low-byte of the result is even.

Carry Flag (CF) :- Set If there was a carry from (or) borrow to the most significant bit during Last result.

80286 Microprocessor:

There are two operating modes for 80286.

- The real address mode and protected virtual address mode. As explained in the real address mode the processor can address up to 1MB of the physical memory.
- The virtual address mode is for multiuser and multitasking system. In this mode of operation the memory management unit can manage up to 1 GB of the virtual memory though the real memory may be much less, only 16 MB.
- Basically in this mode one user do not interfere with the other. Also users cannot interfere with the operating system. These features are called protection.

The 80286 contains four processing units:

1. Bus unit
2. Instruction unit
3. Execution unit
4. Address unit

- All memory and I/O read /write operations are performed by BU. While the current instruction is being executed, the BU pre fetches instructions and keeps them in a queue of six bytes.
- The function of IU is to decode the perfected instructions and to maintain a queue of 3 decoded instructions for execution. The EU executes instruction.
- The address unit computes address of memory or I/O devices, which is to be sent by BU for read and write operation. All the four units work in parallel within the CPU.
- This type of parallel operation is called pipelining. All modern 16-bit CPU use pipelining. In pipelining several execution units in a processor work simultaneously in parallel.

80386 Microprocessor:

- The Intel 80386 (also called Intel386) is a microprocessor which has been used as the CPU of many personal computers since 1986. The 80386 operated at 5 million instructions per second to 11.4MIPS for the 33MHz model.
- This is a 32 bit microprocessor it has a circuitry of 275000 transistors. It was basically introduced in the year 1985. It is compatible with 8086, 8088, 80186, 80286 microprocessors.
- It also contains a four-level protection mechanism on the chip itself. It has a total of 129 instructions. The 80386 is a 32 bit microprocessor with a non multiplexed 32 bit address bus housed in a 132 pin grid array package.

- Basically this microprocessor has three versions: 80386SX, SL and DX. The DX version has a 32 bit internal architecture and a 32 bit data bus whereas the SX and the SL version have a 32 bit internal architecture but a 16 bit wide data bus.
- These versions operate from 20MHz to 33MHz. It is capable of addressing 4G bytes of physical memory and through its memory management unit it can address 64 terabytes of the virtual memory.
- The processor can operate in two modes: Real and protected. In the real mode physical address space is 1Mbytes (20 address lines), which is extended to 4G bytes in the protected mode (32 address lines). The primary difference between these modes is the availability of the memory space and the addressing scheme.
- The 80386 has 32 bit registers and is upward software compatible with the 8086. The execution of the instructions is highly pipelined and the processor is designed to operate in a multiuser and multitasking environment. It has the protection mechanism for this type of environment.
- It has basically six functional units: bus interface unit, code pre fetch unit, instruction decode unit, execution unit, segmentation unit and the paging unit. It has the provision for both memory segmentation and paging. A page is of fixed size 4KB each. Segment vary in size, 4GB is the maximum size of a segment.
- The 80386 has 11 addressing modes: register, immediate, direct, register indirect, based, indexed, scale indexed, base indexed, base scale indexed, base indexed with displacement and base scale indexed with displacement addressing. In the scale indexed addressing the contents of an indexed register are multiplied with a scaling factor and the result is added to the displacement to obtain the operand's offset.
- As explained earlier it has 32 bit register and has eight general purpose registers, six 16 bit segment registers, also has a 32 bit instruction pointer, six debug registers and a 32 bit status register. The80386 has a segment descriptor register associated with each segment register. The 80386 was widely used in powerful PCs before the 80486 was developed.

80486 Microprocessor:

- Basically this is an upgraded advanced version of 80386 and it was released in the year 1989.
- It contains a 32 bit CPU, a floating-point math coprocessor, unified instruction and data cache memory and memory management unit in a single IC.
- It contains an electronic circuitry of 1.2 million transistors. Its operating frequency for its different versions is 25, 33, 66 and 100MHz. It is 3 to 5 times faster than 80386.
- Basically this is available in two versions: DX and SX. The DX type version is a 32 bit processor housed in a 168 pin grid array package and can operate with the clock frequencies from 25 to 66 MHz as explained earlier. The important additional features of the 486 processor in comparison with the 386 processor are as follows.

The486 processor includes:

- Built in math coprocessor. In the 386 system, a math coprocessor is an external device. Therefore, the math instructions in 486 systems are executed three times faster than in 386 systems.
- 8K byte of code and data cache memory on the chip.
- Highly pipelined execution unit. Therefore the execution time for many instructions is one clock period. Basically we do not use 80486 but instead of that we usei486 because of a court ruling that prohibited trade marking numbers. Intel dropped number-based naming altogether with the successor to the i486-thePentium processor. The 486 contains the following functional units:
  - Execution unit
  - Control unit
  - Bus interface unit
  - Code pre fetch unit
  - Instruction decode unit
  - Segmentation unit
  - Paging unit
  - Cache unit
  - Floating point unit.

- The code pre fetch unit contains a 32 byte queue to store fetched instruction codes. The control unit also contains a control ROM to store micro codes. The segmentation unit calculates linear address (the starting address of the segment plus the offset) from the logical address. The address given in the program is called the logical address.
- It also provides 4-level of protection for isolating and protecting tasks and the operating system from each other. The paging unit provides the paging facility within a segment. It translates the linear address into the physical address. The actual capacity of RAM and ROM existing in a computer is known as physical memory.
- The segmentation and the paging unit constitute memory management unit. In summary, the 486 is a high speed, high performance 32bit microprocessor. It executes many of its instructions in one clock cycle by using highly pipelined execution units.
- It is designed to facilitate the execution of high level languages and suited for multiprocessing and multitasking systems. In the early 1990s, 486 was generally used in high end microcomputers and network environments

UNIT-3

Assembler Directives :-

The word defined in This Section are directions to The assembler, The assembler directives described here are Those for The Intel 8086 macro assembler (ASM86).

ASSUME :- The ASSUME directive is used to tell The assembler The name of The logical Segment it should use for a specified Segment. ASSUME CS : CODE, for Example, tells The assembler That The Instructions for a Program are in a logical segment named CODE.

DB - DEFINE BYTE :- The DB directive is used to declare The byte-type variable, (or) to set aside one (or) more Storage locations of type byte in memory.

The statement CURRENT_TEMPERATURE DB 42H, for Example, tells The assembler to reserve 1 byte of memory for a variable named CURRENT_TEMPERATURE and to put The value 42H in That memory location When The program is loaded into RAM to be run.

DT - DEFINE TEN BYTES :- DT is used to tell The assembler to define a variable Which is 10 bytes in length (or) to reserve 10 bytes of storage in memory.

The statement PACKED-BCD DT 11 22 33 44 55 66 77 88 99 00 will declare an arry named PACKED_BCD which is 10 bytes in length. It will initialize The 10 bytes with The values 11 22 33 44 55 66 77 88 99 00 When The program is loaded into memory to be run.

ENDP - END PROCEDURE :- This directive is used along with The name of The procedure to indicate The end of The procedure to The Assembler.

```
Ex:-  SQUARE _ ROOT PROC  : Start of procedure
                          : procedure instruction
                          : statements
      SQUARE_ROOT ENDP    : End of procedure.
```

ENDS - END SEGMENT :- This directive is used with the name of a segment to indicate the end of that logical segment.
ENDS is used with the SEGMENT directive to "bracket" a logical segment containing instructions (or) data.

Ex :- CODE SEGMENT : Start the logical segment containing code
 : Instruction statements.
CODE ENDS : End of segment named CODE.

EQU - EQUATE :- EQU is used to give a name to some value (or) symbol. Each time the assembler finds the given name in the program, it will replace the name with the value (or) symbol you equate that name.

Ex :- CORRECTION - FACTOR EQU 03H at the start of the your program. and later in the program you write the Instruction statement
ADD AL, CORRECTION - FACTOR. (Here AL + 03H → AL).

EVEN - ALIGN ON EVEN MEMORY ADDRESS :- The EVEN directive tells the assembler to increment the location counter to the next even address if it is not already at an even address.

INCLUDE - INCLUDE SOURCE CODE FROM FILE :- This directive is used to tells the assembler to insert a block of source code from the named file into the current source module.

OFFSET :- OFFSET is an operator which tells the assembler to determine the offset (or) displacement of a named data item (or) procedure from the start of the segment which contains it.
MOV BX, OFFSET PRICES, for Ex, it will determine the offset of the variable PRICES from the start of the segment in which PRICES is defined.

ORG - ORIGINATE :- The ORG directive allows you to set the location counter to a desired value at any point in the program.
ORG 2000H tells the assembler to set the location counter to 2000H.

<u>IF-THEN Program</u>:- The IF-THEN structure has the format

```
IF condition THEN
   action
   action.
```

This structure says that IF the stated condition is found to be true, the series of actions following THEN will be executed. If the condition is false, execution will skip over the actions after the THEN and proced with the next mainline Instruction.

<u>Ex</u>:

```
         CMP AX, BX      ; Compare
         JE  THERE       ; If equal then skip correction
         ADD AX, 0002H   ; add correction factor
THERE;   MOV CL, 07H     ; load count.
```

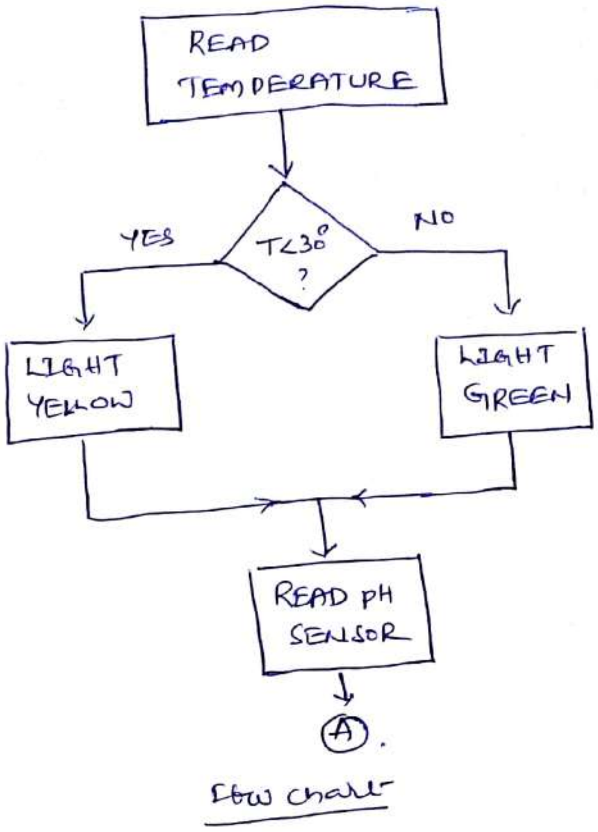
<u>IF-THEN-ELSE program</u>:- The IF-THEN-ELSE structure is used to indicate a choice between two alternative courses of Action.

```
IF condition THEN
   action
ELSE
   action.
```

While making PCB (printed circuit board), part of the Job of This 8086 is to check a ~~&~~ temperature of the solution and turn on a green lamp (or) a yellow lamp depending upon the value of the temperature it reads in.

If the temperature is below 30°c, we want to turn on a yellow lamp to tell the operator that the solution is not up to temperature If the Temperature is greater than (or) equal to 30°C, we want to light a green lamp.

We can represent the algorithm for this problem ~~in two ways~~ as ~~in~~ flow charts and with pseudo code.



flow chart

```
READ TEMPERATURE
TEMPERATURE<30° THEN
LIGHT YELLOW LAMP
ELSE
LIGHT GREEN LAMP.
READ pH SENSOR
```

Pseudo Code.

Program:-

```
        ASSUME  CS : CODE
        MOV DX, FFFEH          ; point DX to port control Register
        MOV AL, 99H            ; Load Control word to Initialize ports.
        OUT DX, AL             ; Send control word to port control registers.

        MOV DX, FFF8H          ; point DX at input port.
        IN AL, DX              ; Read Temp from sensor on Input port
        CMP AL, 30             ; Compare Temp with 30°C.
        ~~JMP~~ JB  YELLOW     ; If Temp <30° THEN light yellow lamp
        JMP   GREEN            ; ELSE light GREEN lamp.

YELLOW: MOV AL, 01H            ; load Code to light yellow lamp
        MOV DX, FFFAH          ; point Dx at output port
        OUT DX, AL             ; Send code to light yellow lamp
        JMP EXIT               ; Go to next mainline Instruction

~~GREEN:~~ ~~MOV AL, 02H~~
        ~~MOV DX, FFFAH~~
        ~~OUT DX, AL~~
```

P.T.O.

#

```
GREEN : MOV AL, 02H        ; Load code to light green lamp.
        MOV DX, FFFAH      ; Point DX at output port
        OUT DX, AL         ; Send code to light green lamp.

EXIT  : MOV DX, FFFCH      ; next mainline instruction
        IN AL, DX          ; Read ph sensor.

CODE  ENDS
      END
```

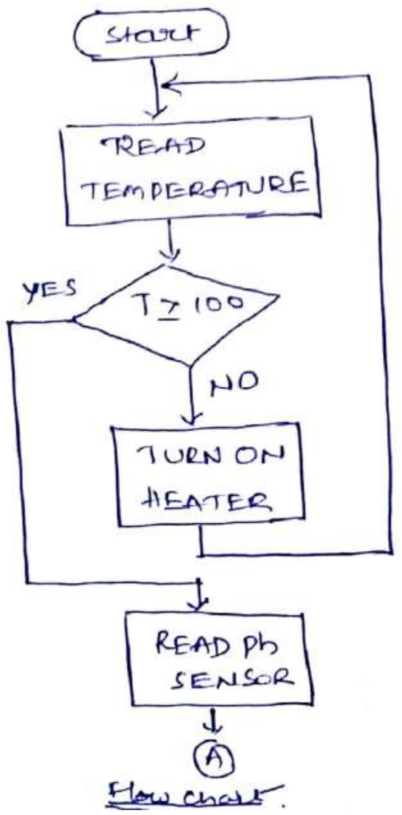
<u>WHILE-DO</u> <u>Program</u>:- WHILE-DO structure has the form.

```
WHILE some condition is present DO
    action
    action.
```

In controlling a chemical process, we want to bring the temperature of a solution up to 100°C before going on to the next step in the process.

If the solution temperature is below 100°C, we want to turn on a heater and wait for the temperature to reach 100°.

The WHILE-DO structure fits this problem because we want to check the condition (temperature) before we turn on the heater.



<u>Flow chart.</u>

```
READ TEMPERATURE
WHILE TEMPERATURE < 100°C DO
    TURN HEATER ON
TURN HEATER OFF
```

<u>Pseudo Code</u>

Program:- ASSUME CS: CODE

```
            MOV DX, FFFEH          ; point DX to port Control register
            MOV AL, 99H            ; Control word to set up output port
            OUT DX, AL             ; Send Control word to port

TEMP_IN:    MOV DX, FFF8H          ; point at input port
            IN AL, DX              ; Input Temperature data
            CMP AL, 100            ; If Temp ≥ 100 then
            JAE HEATER-OFF         ; turn heater off.
            MOV AL, 80H            ; else load Code for heater on
            MOV DX, FFFAH          ; Point DX to output port.
            OUT DX, AL             ; turn heater ON
            JMP TEMP_IN            ; WHILE temp< 100 read temp again
HEATER_OFF: MOV AL, 00             ; Load Code for heater off.
            MOV DX, FFFAH          ; point DX to output port
            OUT DX, AL             ; turn heater OFF

      CODE ENDS
            END.
```

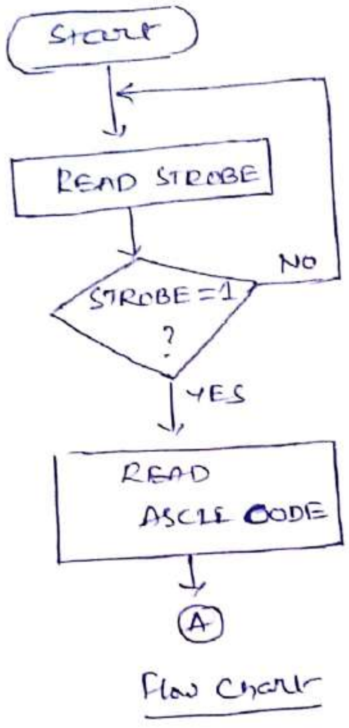
REPEAT - UNTIL PROGRAMS:- REPEAT_UNTIL structure has the form

```
REPEAT
   action
UNTIL some Condition is present.
```

This structure is that, the action (or) series of actions is done once before the Condition is Checked.

If we want to read the data from Keyboard, to ~~read~~ read valid data, we have to look at the Strobe Signal and Test it over and over until It goes high.

Implementation of the algorithm with Assembly language For REPEAT_UNTIL



Flow Chart

```
REPEAT
   READ KEYPRESSED STROBE
UNTIL STROBE = 1
READ ASCII CODE FOR KEY PRESSED
```

Pseudo Code

Program:-

```
              ASSUME  CS:CODE

              MOV DX, FFFAH         ; point DX at strobe port
LOOK-AGAIN;   IN AL, DX             ; Read keyboard strobe
              AND AL, 01            ; mask extra bits & set Flags.
              JZ  LOOK-AGAIN        ; If strobe is low then keep looking
              MOV DX, FFF8H         ; else point DX at data port
              IN AL, DX             ; Read in ASCII code

      CODE    ENDS
              END .
```

Macros

- Macros are just like procedures, but not really. Macros look like procedures, but they exist only until your code is compiled, after compilation all macros are replaced with real instructions.
- If you declared a macro and never used it in your code, compiler will simply ignore it. Emu8086.inc is a good example of how macros can be used; this file contains several macros to make coding easier for you.

—Xk—

INSTRUCTION TYPES

The 8051 instructions are divided among five functional groups:
1. Arithmetic
2. Logical
3. Data transfer
4. Boolean variable
5. Program branching

1. Arithmetic Instructions

With the arithmetic instructions four addressing modes may be used. These modes are direct, indirect, register and immediate. For instance the add-with-carry instruction **ADDC A, operand2** has the following four forms:

Instruction		Operation	Comments
ADDC	A, Rn	Add register to accumulator with carry	Rn = register R0 to R7 from currently selected register bank.
ADDC	A, direct	Add direct byte to accumulator with carry	Direct = 8-bit internal data's location address.
ADDC	A, @Ri	Add indirect RAM to accumulator with carry	@Ri = 8-bit internal data RAM location addressed indirectly through R0 or R1.
ADDC	A, #data	Add immediate data to accumulator with carry	#data = 8-bit constant included in the instruction.

Most of the arithmetic instructions execute in twelve oscillator clock periods with the following three exceptions:

Instruction		Operation	Oscillator periods
INC	DPTR	Increment data pointer	24
MUL	AB	Multiply A and B	48
DIV	AB	Divide A by B	48

2. Logical Instructions

The logical instructions can perform Boolean operations on the data contained either in the accumulator or in an internal RAM location. Those logical instructions that use the accumulator as one of the operands have the same addressing modes as those found in arithmetic instruction. Examples of such instructions are:

Instruction		Operation	Comments
ORL	A, Rn	OR register to accumulator	Rn = register R0 to R7 from currently selected register bank.
ORL	A, direct	OR direct byte to accumulator	Direct = 8-bit internal data's location address.
ORL	A, @Ri	OR indirect RAM to accumulator	@Ri = 8-bit internal data RAM location addressed indirectly through R0 or R1.
ORL	A, #data	OR immediate data to accumulator	#data = 8-bit constant included in the instruction.

Note that in addition to the **ORL A, direct** instruction we also have the equivalent "mirror" instruction **ORL direct, A** (OR the accumulator to the direct byte). All such instructions execute in twelve oscillator clock periods. Apart from the logical instructions that use the accumulator as one of the operands, there are three logical instructions that perform Boolean operations directly on any byte in the internal data memory without going through the accumulator. The table below gives a summary of these instructions.

Instruction		Operation	Comments
ANL	direct, #data	AND immediate data to direct byte	Example: ANL P2,#0FFH
ORL	direct, #data	Or immediate data to direct byte	Example: ORL P2,#0FFH
XRL	direct, #data	Exclusive OR immediate data to direct byte	Example: XRL P2,#0FFH

These three instructions take 24-oscillator clock period to execute. Note that these instructions perform what is known as "*read-modify-write*" operation. In a "read-modify-write" instruction the datum in the direct address location is first read, then the logical operation is performed on the read datum with the immediate byte, and

Finally the result of the logical operation is written back to the direct address location.

The logical group of instructions also contains four rotate instructions, which operate on the contents of the accumulator, and a swap instruction (SWAP A). The swap instruction is useful in BCD arithmetic manipulations.

3. Data Transfer Instructions

This group contains the largest number of instructions that enable us to move data within the internal RAM, move data between the internal RAM and external RAM, and three instructions that allow us to manipulate look-up tables. The following table contains some examples of data transfer instructions. Note that the stack in the 8051 is implemented in the on-chip RAM. Unlike the stack implementations in other microprocessors the stack grows "upwards" in memory, i.e. towards higher memory addresses. The execution of the PUSH instruction first increments the stack pointer, and then copies the indicated byte into the stack.

Instruction	Operation	Comments
MOV A, Rn	Move register to accumulator	Register addressing.
MOV direct, A	Move accumulator to direct byte	Direct addressing.
MOV Rn, #data	Move immediate data to register	Immediate addressing.
MOV @Ri, A	Move accumulator to indirect RAM	Indirect addressing. Move to internal RAM.
MOV direct, @Ri	Move from indirect RAM to direct byte	Indirect addressing. Move from external RAM to internal RAM.
MOV DPTR, #data16	Load data pointer with a 16-bit constant	Immediate address.
MOVX @DPTR, A	Move accumulator to external RAM	Indirect addressing.
MOVC A, @A+PC	Move Code byte relative to PC to ACC	Index addressing.
PUSH direct	Push direct byte into stack	Direct addressing. Stack itself is accessed using indirect addressing through the stack pointer.
XCH A, @Ri	Exchange indirect RAM with accumulator	Indirect addressing.

4. Boolean Instructions

The 8051 has a range of Boolean variable manipulating instructions which enable us to set or reset individual bits within some of the locations in the internal RAM, and some of the special function registers. We give some examples of these instructions in the table below.

Instruction	Operation	Oscillator period
CLR C	Clear Carry bit	12
CLR bit	Clear direct bit	12
SETB C	Set Carry	12
CPL bit	Complement direct bit	12
ANL C, bit	AND complement of direct bit to Carry	24
ORL C, bit	OR direct bit to Carry	24
MOV bit, C	Move Carry to direct bit	24

5. Program branching Instructions

This group contains conditional and unconditional branch instructions. Some of the conditional branch instructions which test individual bits such as JC *rel* (jump if Carry is set) are found in the previous instruction group. The following is a list of some of the program branch instructions.

Instruction	Operation	Comments
ACALL addr11	Absolute subroutine call	Absolute addressing
LCALL addr16	Long subroutine call	Long addressing. Call to subroutine in external memory.
RETI	Return from interrupt	
SJMP rel	Short jump	Relative addressing.
JMP @A+DPTR	Jump indirect relative to the data pointer	Indexed addressing.
CJNE A,#data,rel	Compare the immediate data to the accumulator and jump if not equal	Compare – immediate Jump – relative
CJNE @Ri,#data,rel	Compare immediate data to indirect and jump if not equal	As above
DJNZ Rn, rel	Decrement register and jump if not zero	Register and relative addressing.
DJNZ direct, rel	Decrement the direct byte and jump if not Zero	As above
JNC rel	Jump if Carry is not set	Jump – relative addressing
JBC bit, rel	Jump if direct bit is set and clear bit	As above

ADDRESSING MODES

In short, a way to addressing a operand is nothing but addressing mode. Operand means data on which we are going to operate i.e source data. We can address the operand using direct address or by using register or we can address the data using some numerical value etc.

MOVA,#6BH

In this example 6BH is operand i.e. source data. After execution of this instruction 6BH is added to the accumulator.
We can execute this instruction by using 5 different ways i.e we can use 5 different addressing modes to execute this instruction. Following are the different type of addressing modes.

- Immediate addressing mode
- Direct addressing mode
- Register addressing mode
- Register indirect addressing mode
- Indexed addressing mode.

1. Immediate addressing mode:

As word indicates 'immediate' this addressing mode transfers 8 bit immediate data to destination.

MOVA,#6BH

This instruction moves immediate data hex 6B to accumulator.

2. Direct addressing mode:

This is type of addressing mode in which the address of the data (source data) is given as operand..i.e. it is given directly in the form of numerical data.

MOVB,20H

The instruction above moves value at address 20H memory location to register B .
Here 20H is address of SFR.
The difference between direct addressing and immediate addressing mode is, we don't use '#' in direct addressing mode, unlike immediate mode.

3. Register addressing mode:

This is type of addressing mode in which we use the register name directly as source data.

MOV A, R5

After execution of this instruction content of register R5 is get copied to location accumulator.

4. Register direct addressing mode :

In this mode of addressing address of source data is given by value at register indirectly that's why we call it as a indirect register addressing mode.

e.g. MOV A, @R1

Here in this instruction register R1 holds the address, suppose value at R1 is '20H' then after execution of above instruction value at location 20H is transferred to to accumulator.

Here symbol '@' indicates address. In this type addressing mode we can use only register R0 and R1 to provide indirect address.

5. Indexed addressing mode:

In this type of addressing mode we use following instruction:

1. MOVC A, @A+DPTR and 2. MOVC A, @A+PC

'MOVC A, @A+DPTR' instruction copies value at the location given by result of addition of content of accumulator and 16 bit register DPTR. Suppose Accumulator contain '01H' and DPTR contains value '1000H' then after the execution of instruction 'MOVC A, @A+DPTR the value at address '1001H' will be transferred to **accumulator.**

'MOVC A, @A+PC' instruction copies value at the location given by result of addition of content of accumulator and 16 bit register PC (Program counter). Suppose Accumulator contain '01H' and PC contains value '1000H' then after the execution of instruction 'MOVC A, @A+DPTR the value at address '1001H' will be transferred to accumulator.

ASSEMBLY LANGUAGE PROGRAMMING

Simple addition is done within the 8051 based on 8 bit numbers, but it is often required to add 16 bit numbers, or 24 bit numbers etc. This leads to the use of multiple byte (multi-precision) arithmetic. The least significant bytes are first added, and if a carry results, this carry is carried over in the addition of the next significant byte etc. This addition process is done at 8-bit precision steps to achieve multi precision arithmetic. The ADDC instruction is used to include the carry bit in the addition process. Example instructions using ADDC are:

ADDC A, #55h ; Add contents of A, the number 55h, the carry bit; and put the sum in A

ADDC A, R4 ; Add the contents of A, the register R4, the carry bit; and put the sum in A.

Subtraction Computer subtraction can be achieved using 2's complement arithmetic. Most computers also provide instructions to directly subtract signed or unsigned numbers. The accumulator, register A, will contain the result (difference) of the subtraction operation. The C (carry) flag is treated as a borrow flag, which is always subtracted from the minuend during a subtraction operation. Some examples of subtraction instructions are:

SUBB A, #55d ; Subtract the number 55 (decimal) and the C flag from A; and put the result in A.

SUBB A, R6 ; Subtract R6 the C flag from A; and put the result in A.

SUBB A, 58h ; Subtract the number in RAM location 58h and the C flag From A; and put the result in A.

Increment/Decrement The increment (INC) instruction has the effect of simply adding a binary 1 to a number while a decrement (DEC) instruction has the effect of subtracting a binary 1 from a number. The increment and decrement instructions can use the addressing modes: direct, indirect and register. The flags C, AC, and OV are not affected by the increment or decrement instructions. If a value of FFh is increment it overflows to 00h. If a value of 00h is decrement it underflows to FFh. The DPTR can overflow from FFFFh to 0000h. The DPTR register cannot be decremented using a DEC instruction (unfortunately!). Some example INC and DEC instructions are as follows:

INC R7; Increment register R7

INC A; Increment A INC @R1 ; Increment the number which is the content of the address in R1

DEC A; Decrement register A

DEC 43h; Decrement the number in RAM address 43h

INC DPTR; Increment the DPTR register

Multiply / Divide

The 8051 supports 8-bit multiplication and division. This is low precision (8 bit) arithmetic but is useful for many simple control applications. The arithmetic is relatively fast since multiplication and division are implemented as single instructions. If better precision, or indeed, if floating point arithmetic is required then special software routines need to be written. For the MUL or DIV instructions the A and B registers must be used and only unsigned numbers are supported.

Multiplication

The MUL instruction is used as follows (note absence of a comma between the A and B operands):

MUL AB; Multiply A by B. The resulting product resides in registers A and B, the low-order byte is in A and the high order byte is in B.

Division

The DIV instruction is used as follows:

DIV AB; A is divided by B. The remainder is put in register B and the integer part of the quotient is put in register A.

Decimal Adjust (Special)

The 8051 performs all arithmetic in binary numbers (i.e. it does not support BCD arithmetic). If two BCD numbers are added then the result can be adjusted by using the DA, decimal adjust, instruction:

DA A; Decimal adjust A following the addition of two BCD numbers.

UNIT-5

Architecture of 8051 micro controller:-

→ 8051 micro controller is the 8-bit micro controller.

→ 5v power supply is required to operate, It is a 40 pin IC.

→ It works with a clock frequency of 11.05962 MHz.

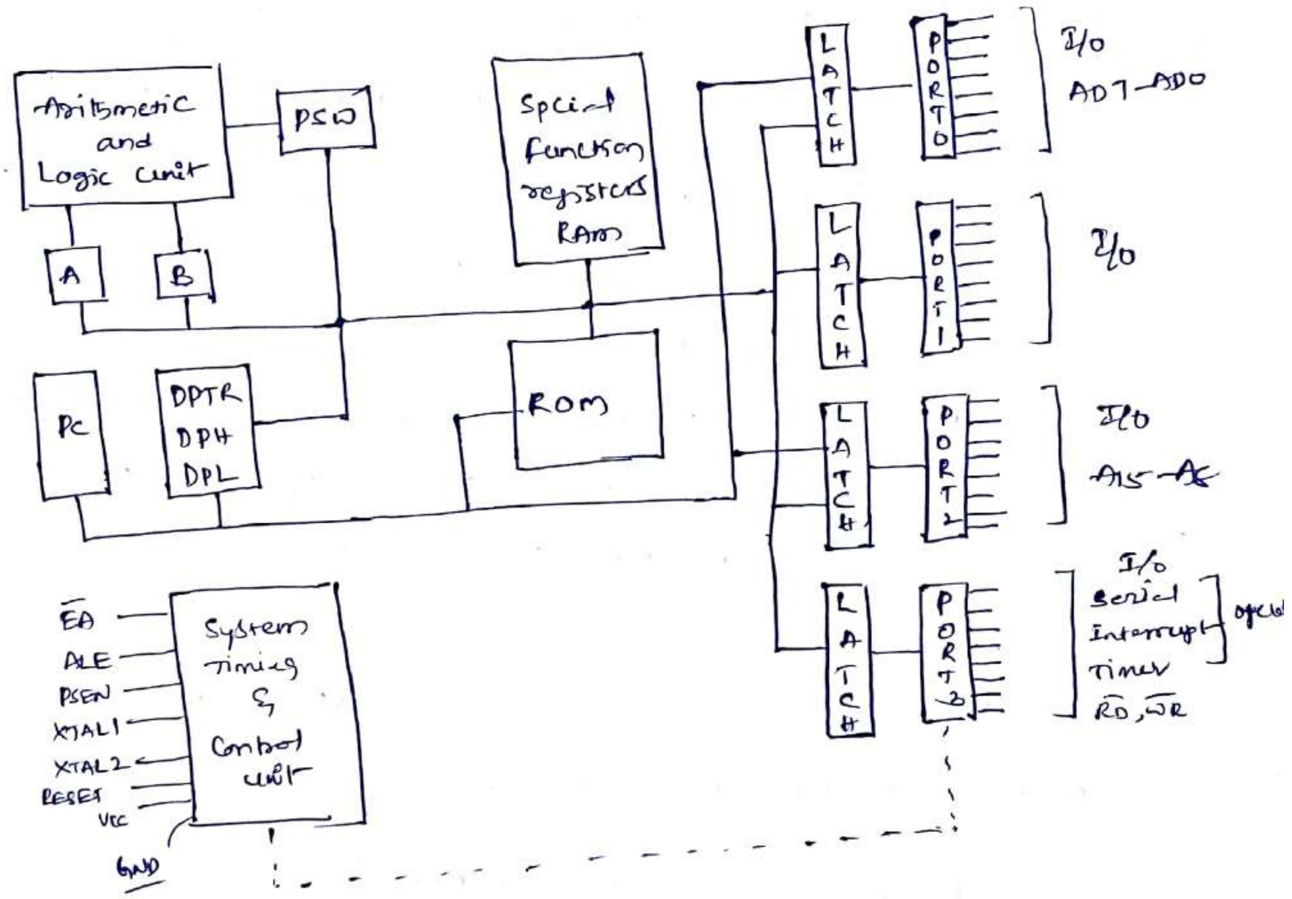


Fig. Architecture of Micro controller 8051.

→ Main functional Blocks in 8051 micro controller are

- CPU
- ROM
- RAM
- I/O ports
- System Timing Control unit
- Timers
- Interrupt Sources.

→ CPU contains

ALU → Arithematic and logical unit is used for the Arithmetic Calculations.

A : Accumulator is a 8-bit register, which is used for Arithematic operations.

B : Register 'B' is a 8-bit register is used in only Two Instructions MUL A;B and DIV A;B.

PC : Program counter is a 16-bit register, points to the address of next Instruction to be executed from ROM.

DPTR : Data pointer is a 16-bit register, it is divided into two parts DPH & DPL. DPH for higher order 8-bits, DPL for lower order 8-bits.

→ ROM memory in 8051 :- We have 4 KB of ROM, in 8051 micro controller. used to store programs.

Address range of memory (ROM) is.

end Address.	0000	1111	1111	1111	(0FFFH)
	- - -	- - -	- - -	- - -	
	. . .	. . .	- - -	- - -	
starting Address	0000	0000	0000	0000	(0000H)

→ 8051 Flag register / PSW :- It is a 8-bit register used to Indicate the Arithmetic condition of Accumulator.

· Flag register in 8051 is called as program status word (PSW). This special function register PSW is also bit addressable.

PSW0.7	PSW0.6	PSW0.5	PSW0.4	PSW0.3	PSW0.2	PSW0.1	PSW0.0
CY	AC	~~F0~~ -	RS1	RS0	OV	-	P

→ In six active flags, four are status flags, two are Register Bank select flags

→ Status flags parity (P), overflow (OV), Auxiliary Carry (AC) and Carry (CY) are reflect the ~~Reg~~ Result of the ALU.

Parity (P) : If result contain even no. of 1's, parity is ~~clr~~ set.

overflow (OV) : In signed operations, If magnitude bit provides carry, overflow gets set.

Auxillary Carry : If Any carry form ~~D~~ Lower nibble to Higher nibble in the result, Auxillary Carry is set.

Carry (CY) : If there is Any carry from most significant bit, Carry flag gets set.

Ex:- . Add 54H to 44H . result will reflect flags as shown below

```
mov A, #54H          ~~0100~~ ~~F~~
mov R1, #44H         0101  0100
ADD A, R1            0100  0100
                     ----------
                     1001  1000
```

result : parity : P = 0 ; (odd no. of ones)

A.C = 0 ; (no carry from D_3 to D_4)

OV = 0 ; (It is not a signed operation)

CY = 0 ; (There is no Carry).

→ Register Bank Select Bits. is used to select the register Bank in RAM.

RS1	RS0	Select BANK
0	0	Register BANK 0
0	1	Register BANK 1
1	0	Register BANK 2
1	1	Register BANK 3.

Initially by default always Bank '0' is selected.

→ Structure of RAM (or) 8051 Register Bank and Stack:

- 128 bytes of RAM is available in RAM, apart from that another 128 bytes are allocated for special function registers.
- Address Range of RAM, 00H to 7FH. is user accessible RAM, Extra 128 Bytes RAM which is 80H to FFH used for storage of. SFR (special function registers).

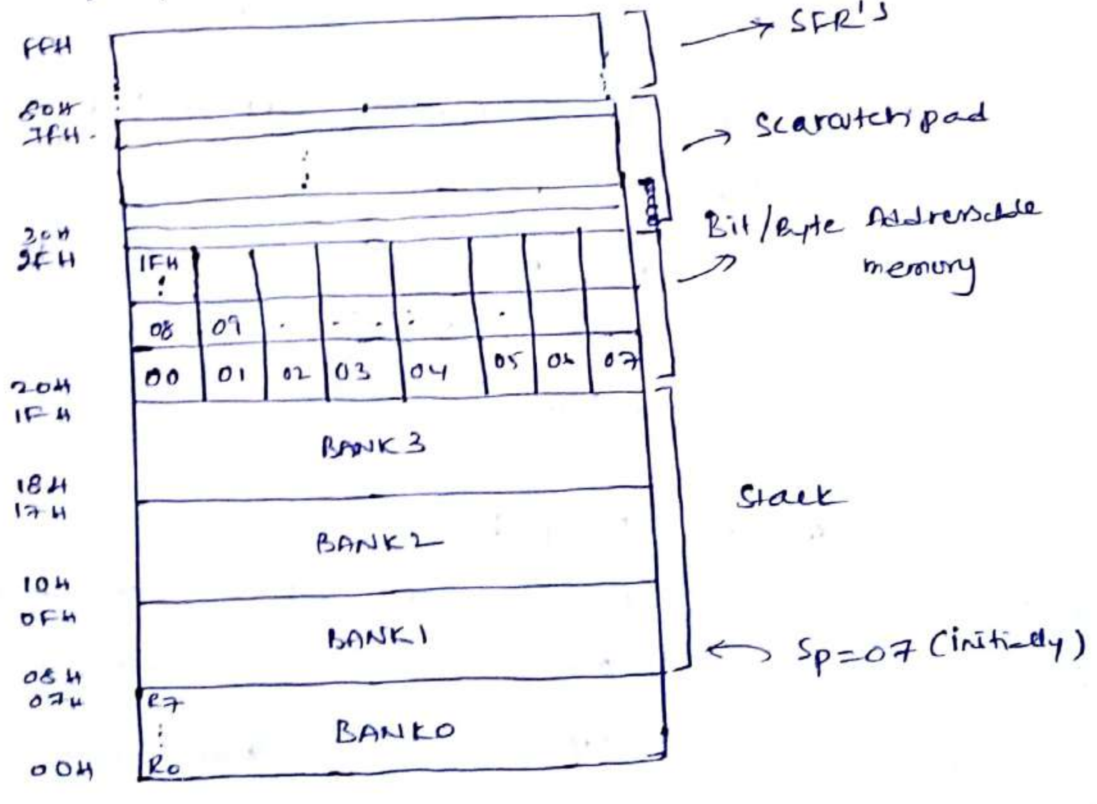


fig. Structure of RAM.

→ There are four register banks, in each register bank There are ~~8~~ eight 8-bit registers available from R0 to R7.

→ By default Bank '0' is selected. For Bank 0, R0 has address 00H, R1 has address 01H . . . . R7 has address 07H.

→ R0 to R7 are byte addressable.

→ Locations 20H to 2FH is bit addressable RAM means each bit from 00H to FFH in This we can set (or) reset.

→ Locations 30H to 7FH is used as Scratch pad means, we can use this space for data reading and writing (or) for data storage.

→ RAM locations from 08H to 1FH Can be used as Stack.
Stack is used to Store the data Temporarily.
Stack is Last in First out (LIFO).

→ Stack pointer is a 8-bit register. It Indicates the current RAM address ~~as~~ available for Stack (or) it points the top of Stack.

→ Initially by default at 07H because first location Stack is 08H.

Special function registers (SFR'S) :-

→ Extra 128 bytes RAM which is used to Store the SFR's, are used to Interrupt Control, timer / counter, UART, Power Control are performed Through registers between 80H to FFH.

For removing this situation we use the stack from location 30H to 7FH by shifting SP to 2FH.

MOV SP,#2FH;

DPTR → Data Pointer in 8051

- 16 bit register, it is divided into two parts DPH and DPL.
- DPH for Higher order 8 bits, DPL for lower order 8 bits.
- DPTR, DPH, DPL these all are SFRs in 8051.

Special Function Register

- (See Fig.) RAM scratch pad, there is extra 128 byte RAM which is used to store the SFRs
- Following figure shows special function bit address, all access to the four I/O ports CPU register, interrupt control register, timer/counter, UART, power control are performed through registers between 80H and FFH.

Byte Addressable SFR with byte address

SP – Stack printer – 81H
DPTR – Data pointer 2 bytes
DPL – Low byte – 82H
DPH – High byte – 83H
TMOD – Timer mode control – 89H
TH0 – Timer 0 Higher order bytes – 8CH
TL0 – Timer 0 Low order bytes – 8AH
TH1 – Timer 1 High bytes = 80H
TL1 – Timer 1 Low order byte = 86H
SBUF – Serial data buffer = 99H
PCON – Power control – 87H.

Instruction set of 8051

The 8051 instructions are divided among five functional groups:

1. Arithmetic
2. Logical
3. Data transfer
4. Boolean variable
5. Program branching

1. Arithmetic Instructions:

ADD A, source ;A = A + source

- ➢ The instruction ADD is used to add two operands
- ➢ Destination operand is always in register A Source operand can be a register, immediate data, or in memory

  Ex: MOV A,#0F5H
  ADD A, #0BH

DA A ; decimal adjust for addition

- ➢ The DA instruction is provided to correct the aforementioned problem associated with BCD addition
- ➢ The DA instruction will add 6 to the lower nibble or higher nibble if need

Ex:

```
MOV A,#47H ;   A=47H first BCD operand
MOV B,#25H ;   B=25H second BCD operand
ADD A,B ;      hex(binary) addition(A=6CH)
DA A ;         adjust for BCD addition (A=72H)
```

After an ADD or ADDC instruction

- ➢ If the lower nibble (4 bits) is greater than 9, or if AC=1, add 0110 to the lower 4 bits
- ➢ If the upper nibble is greater than 9, or if CY=1, add 0110 to the upper 4 bits

SUBB A,source ;A = A – source – CY

- To make SUB out of SUBB, we have to make CY=0 prior to the execution of the instruction
- Notice that we use the CY flag for the borrow

MUL AB ;AxB, 16-bit result in B, A
- The 8051 supports byte by byte multiplication only
- The byte are assumed to be unsigned data

Ex:

```
MOV A,#25H ;                          load 25H to reg. A
MOV B,#65H ;                          load 65H to reg. B
MUL AB ;      25H * 65H = E99         where; B = OEH and A = 99H
```

DIV AB;   divide A by B, A/B
- The 8051 supports byte over byte division only
- The byte are assumed to be unsigned data

Ex:

```
MOV A,#95 ;                    load 95 to reg. A
MOV B,#10 ;                    load 10 to reg. B
MUL AB ;           A = 09(quotient) and;   B = 05(remainder)
```

2. Logical Instructions:

ANL destination, source ;dest = dest AND source

- This instruction will perform a logic AND on the two operands and place the result in the destination
- The destination is normally the accumulator
- The source operand can be a register, in memory, or immediate

Ex:

```
MOV A,#35H ;                A = 35H
ANL A,#0FH ;                A = A AND 0FH
```

ORL destination,source ;dest = dest OR source

- The destination and source operands are ORed and the result is placed in the destination
- The destination is normally the accumulator
- The source operand can be a register, in memory, or immediate

Ex:

```
MOV A,#04H ;            A = 04
ORL A,#68H ;            A = 6C
```

XRL destination,source ;dest = dest XOR source

- This instruction will perform XOR operation on the two operands and place the result in the destination
- The destination is normally the accumulator
- The source operand can be a register, in memory, or immediate

Ex:

```
MOV A,#54H
XRL A,#78H
```

CPL A ; complements the register A
- This is called 1's complement
- To get the 2's complement, all we have to do is to to add 1 to the 1's complement

Ex:

```
MOV A, #55H
CPL A; now A=AAH;  0101 0101(55H);becomes 1010     1010(AAH)
```

CJNE destination,source,rel. addr.

- The actions of comparing and jumping are combined into a single instruction called CJNE (compare and jump if not equal)
- The CJNE instruction compares two operands, and jumps if they are not equal
- The destination operand can be in the accumulator or in one of the Rn registers
- The source operand can be in a register, in memory, or immediate
- The operands themselves remain unchanged
- It changes the CY flag to indicate if the destination operand is larger or smaller

Rotation Instructions:

RR A ;rotate right A

- In rotate right
- The 8 bits of the accumulator are rotated right one bit, and
- Bit D0 exits from the LSB and enters into MSB, D7

RL A ;rotate left A

- In rotate left
- The 8 bits of the accumulator are rotated left one bit, and
- Bit D7 exits from the MSB and enters into LSB, D0

RRC A ;rotate right through carry

- In RRC A
- Bits are rotated from left to right
- They exit the LSB to the carry flag, and the carry flag enters the MSB

RLC A ;rotate left through carry

- In RLC A
- Bits are shifted from right to left
- They exit the MSB and enter the carry flag, and the carry flag enters the LSB

3. **Data Transfer Instructions:**

MOV A, R2 ; moving data from source R2 to destination Accumulator

- This instruction will perform a Move the data in source to the destination.
- The destination is normally the register, memory location.
- The source operand can be a register, in memory, or immediate

4. **Boolean Instructions:**

There are several instructions by which the CY flag can be manipulated directly

Instruction	Function
SETB C	Make CY = 1
CLR C	Clear carry bit (CY = 0)
CPL C	Complement carry bit
MOV b,C	Copy carry status to bit location (CY = b)
MOV C,b	Copy bit location status to carry (b = CY)
JNC	target Jump to target if CY = 0
JC	target Jump to target if CY = 1
ANL C,bit	AND CY with bit and save it on CY
ANL C,/bit	AND CY with inverted bit and save it on CY
ORL C,bit	OR CY with bit and save it on CY
ORL C,/bit	OR CY with inverted bit and save it on CY

5. **Program Branching Instructions**

DJNZ reg, Label

- Repeating a sequence of instructions a certain number of times is called a loop
- The register is decremented
- If it is not zero, it jumps to the target address referred to by the label
- Prior to the start of loop the register is loaded with the counter for the number of repetitions
- Counter can be R0 – R7 or RAM location

Jump only if a certain condition is met

```
JZ label                    ;jump if A=0

JNC label                   ;jump if no carry, CY=0
```

- If CY = 0, the CPU starts to fetch and execute instruction from the address of the label
- If CY = 1, it will not jump but will execute the next instruction below JNC

JBC	Jump if bit = 1 and clear bit
JNB	Jump if bit = 0
JB	Jump if bit = 1
JNC	Jump if CY = 0
JC	Jump if CY = 1
CJNE reg,#data	Jump if byte ≠ #data
CJNE A,byte	Jump if A ≠ byte
DJNZ	Decrement and Jump if A ≠ 0
JNZ	Jump if A ≠ 0
JZ	Jump if A = 0

Addressing Modes of 8051

In short, a way to addressing a operand is nothing but addressing mode. Operand means data on which we are going to operate i.e source data. We can address the operand using direct address or by using register or we can address the data using some numerical value etc.

MOVA,#6BH

In this example 6BH is operand i.e. source data. After execution of this instruction 6BH is added to the accumulator.

We can execute this instruction by using 5 different ways i.e we can use 5 different addressing modes to execute this instruction. Following are the different type of addressing modes.

- Immediate addressing mode
- Direct addressing mode
- Register addressing mode
- Register indirect addressing mode
- Indexed addressing mode.

1. **Immediate addressing mode:**

As word indicates 'immediate' this addressing mode transfers 8 bit immediate data to destination.

MOVA,#6BH

This instruction moves immediate data hex 6B to accumulator.

2. **Direct addressing mode**:

This is type of addressing mode in which the address of the data (source data) is given as operand..i.e. it is given directly in the form of numerical data.

MOVB,20H

The instruction above moves value at address 20H memory location to register B .

Here 20H is address of SFR.
The difference between direct addressing and immediate addressing mode is, we don't use '#' in direct addressing mode, unlike immediate mode.

3. **Register addressing mode:**
This is type of addressing mode in which we use the register name directly as source data.
MOV A, R5
After execution of this instruction content of register R5 is get copied to location accumulator.

4. **Register direct addressing mode :**
In this mode of addressing address of source data is given by value at register indirectly that's why we call it as a indirect register addressing mode.
e.g. MOV A, @R1
Here in this instruction register R1 holds the address, suppose value at R1 is '20H' then after execution of above instruction value at location 20H is transferred to to accumulator.
Here symbol '@' indicates address. In this type addressing mode we can use only register R0 and R1 to provide indirect address.

5. **Indexed addressing mode:**
In this type of addressing mode we use following instruction:
1. MOVC A, @A+DPTR and 2. MOVC A, @A+PC
'MOVC A, @A+DPTR' instruction copies value at the location given by result of addition of content of accumulator and 16 bit register DPTR. Suppose Accumulator contain '01H' and DPTR contains value '1000H' then after the execution of instruction 'MOVC A, @A+DPTR the value at address '1001H' will be transferred to **accumulator.**

'MOVC A, @A+PC' instruction copies value at the location given by result of addition of content of accumulator and 16 bit register PC (Program counter). Suppose Accumulator contain '01H' and PC contains value '1000H' then after the execution of instruction 'MOVC A, @A+DPTR the value at address '1001H' will be transferred to accumulator.

Timer Operation in 8051 Micro controller

- The 8051 has two counters/timers which can be used either as timer to generate a time delay or as counter to count events happening outside the microcontroller.
- The 8051 has two timers: timer0 and timer1. They can be used either as timers or as counters. Both timers are 16 bits wide. Since the 8051 has an 8-bit architecture, each 16-bit is accessed as two separate registers of low byte and high byte. First we shall discuss about Timer0 registers.
- Timer0 registers is a 16 bits register and accessed as low byte and high byte. The low byte is referred as a TL0 and the high byte is referred as TH0. These registers can be accessed like any other registers.

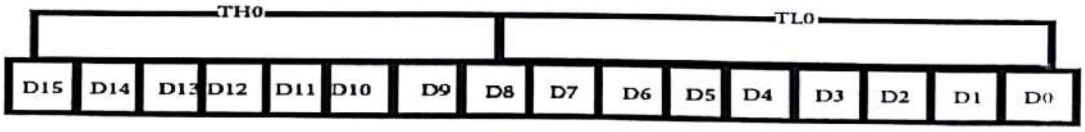


Fig. Timer0

➢ Timer1 registers is also a 16 bits register and is split into two bytes, referred to as TL1 and TH1.

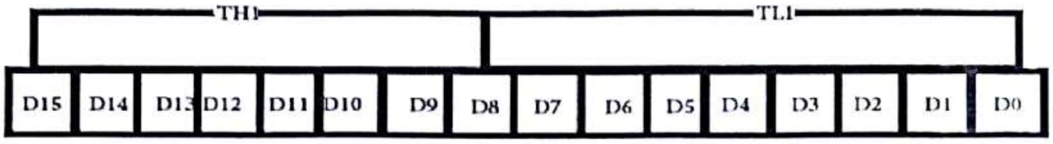


Fig. Timer1

TMOD (timer mode) Register:

➢ This is an 8-bit register which is used by both timers 0 and 1 to set the various timer modes.
➢ In this TMOD register, lower 4 bits are set aside for timer0 and the upper 4 bits are set aside for timer1.
➢ In each case, the lower 2 bits are used to set the timer mode and upper 2 bits to specify the operation.

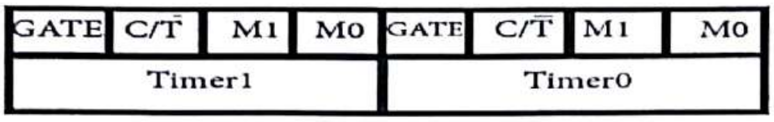


Fig. TMOD Register

➢ The hardware way of starting and stopping the timer by an external source is achieved by making GATE=1 in the TMOD register. And if we change to GATE=0 then we do not need external hardware to start and stop the timers.

➢ The second bit is C/T bit and is used to decide whether a timer is used as a time delay generator or an event counter. If this bit is 0 then it is used as a timer and if it is 1 then it is used as a counter.
➢ In upper or lower 4 bits, the last bits third and fourth are known as M1 and M0 respectively. These are used to select the timer mode.

M0	M1	Mode	Operating Mode
0	0	0	13-bit timer mode, 8-bit timer/counter THx and TLx as 5- bit prescalar.
0	1	1	16-bit timer mode, 16-bit timer/counters THx and TLx are cascaded; There are no prescalar.
1	0	2	8-bit auto reload mode, 8-bit auto reload timer/counter; THx holds a Value which is to be reloaded into TLx each time it overflows.
1	1	3	Spilt timer mode

Mode 1

➢ It is a 16-bit timer; therefore it allows values from 0000 to FFFFH to be loaded into the timer's registers TL and TH.
➢ After TH and TL are loaded with a 16-bit initial value, the timer must be started. We can do it by "SETB TR0" for timer 0 and "SETB TR1" for timer 1.
➢ After the timer is started.
➢ It starts count up until it reaches its limit of FFFFH. When it rolls over from FFFF to 0000H, it sets high a flag bit called TF (timer flag).
➢ This timer flag can be monitored. When this timer flag is raised, one option would be stop the timer with the instructions "CLR TR0" or CLR TR1 for timer 0 and timer 1 respectively.
➢ Again, it must be noted that each timer flag TF0 for timer 0 and TF1 for timer1. After the timer reaches its limit and rolls over, in order to repeat the process the registers TH and TL must be reloaded with the original value and TF must be reset to 0.

Example program

- To create a square wave of 50% duty cycle (with equal portions high and low) on the P1.5 bit. Timer 0 is used to generate the time delay

```
         MOV TMOD,#01              ;Timer 0, mode 1(16-bit mode)
   HERE: MOV TL0,#0F2H             ;TL0=F2H, the low byte
         MOV TH0,#0FFH             ;TH0=FFH, the high byte
         CPL P1.5                  ;toggle P1.5
         ACALL DELAY
         SJMP HERE
```

In the above program notice the following step.
1. TMOD is loaded.
2. FFF2H is loaded into TH0-TL0.
3. P1.5 is toggled for the high and low portions of the pulse.

```
DELAY:
          SETB TR0                 ;start the timer 0
   AGAIN: JNB TF0, AGAIN           ;monitor timer flag 0;until it rolls over
          CLR TR0                  ;stop timer 0
          CLR TF0                  ;clear timer 0 flag
          RET
```

4. The DELAY subroutine using the timer is called.
5. In the DELAY subroutine, timer 0 is started by the SETB TR0 instruction.
6. Timer 0 counts up with the passing of each clock, which is provided by the crystal oscillator. As the timer counts up, it goes through the states of FFF3,FFF4, FFF5, FFF6, FFF7, FFF8, FFF9, FFFA, FFFB, and so on until it reaches FFFFH. One more clock rolls it to 0, raising the timer flag (TF0=1).At that point, the JNB instruction falls through.
7. Timer 0 is stopped by the instruction CLR TR0. The DELAY subroutine ends and the process is repeated.

Mode 2

- It is an 8 bit timer that allows only values of 00 to FFH to be loaded into the timer's register TH.
- After TH is loaded with 8 bit value, the 8051 gives a copy of it to TL. Then the timer must be started. It is done by the instruction "SETB TR0" for timer 0 and "SETB TR1" for timer1. This is like mode 1.
- After timer is started, it starts to count up by incrementing the TL registers. It counts up until it reaches its limit of FFH. When it rolls over from FFH to 00.
- It sets high the TF (timer flag). If we are using timer 0, TF0 goes high; if using TF1 then TF1 is raised.
- When Tl register rolls from FFH to 00 and TF is set to 1, TL is reloaded automatically with the original value kept by the TH register. To repeat the process, we must simply clear TF and let it go without any need by the programmer to reload the original value. This makes mode 2 auto reload, in contrast in mode 1 in which programmer has to reload TH and TL.

Example Program

- To find the frequency of the square wave generated on pin P1.0 in the following program

```
          MOV TMOD,#20H            ;T1/8-bit/auto reload
          MOV TH1,#5               ;TH1 = 5
          SETB TR1                 ; start the timer 1
    BACK: JNB TF1,BACK             ;till timer rolls over
          CPL P1.0                 ;P1.0 to hi, lo
          CLR TF1                  ;clear Timer 1 flag
          SJMP BACK                ;mode 2 is auto-reload
```

Solution:

- First notice the target address of SJMP. In mode 2 we do not need to reload TH since it is auto-reload. Now (256 - 05) × 1.085 us =251 × 1.085 us = 272.33 us is the high portion of the pulse. Since it is a 50% duty cycle square wave, the period T is twice that; as a result T = 2 × 272.33 us = 544.67 us and the frequency =1.83597 kHz

TCON register

TCON register is the timer control special function register. In which lower nibble is used to control interrupt operation and higher nibble is used to control timer operation. Bits and symbol and functions of every bits of TCON are as follows:

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Fig. TCON Register

BIT	Symbol	Functions
7	TF1	Timer1 over flow flag. Set when timer rolls from all 1s to 0. Cleared When the processor vectors to execute interrupt service routine Located at program address 001Bh.
6	TR1	Timer 1 run control bit. Set to 1 by programmer to enable timer to count; Cleared to 0 by program to halt timer.
5	TF0	Timer 0 over flow flag. Same as TF1.
4	TR0	Timer 0 run control bit. Same as TR1.
3	IE1	External interrupt 1 Edge flag. Not related to timer operations.
2	IT1	External interrupt1 signal type control bit. Set to 1 by program to Enable external interrupt 1 to be triggered by a falling edge signal. Set To 0 by program to enable a low level signal on external interrupt1 to generate an interrupt.
1	IE0	External interrupt 0 Edge flag. Not related to timer operations. 0 IT0 External interrupt 0 signal type control bit. Same as IT0.

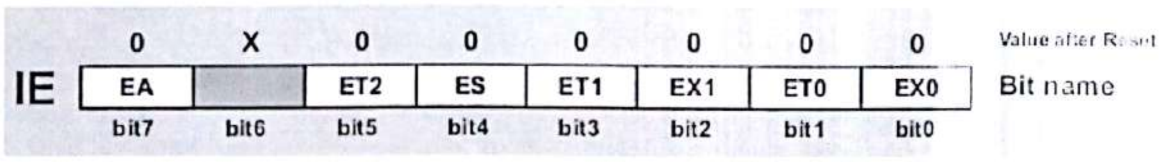
Interrupt Operation in 8051 Micro Controller

- ➢ An interrupt is an external or internal event that interrupts the microcontroller to inform it that a device needs its service.
- ➢ There are six interrupt sources for the 8051, which means that they can recognize 6 different events that can interrupt regular program execution

Six interrupts are allocated as follows

1. Reset – power-up reset
2. Two interrupts are set aside for the timers: one for timer 0 and one for timer 1
3. Two interrupts are set aside for hardware external interrupts P3.2 and P3.3 are for the external hardware interrupts INT0 (or EX1), and INT1 (or EX2)
4. Serial communication has a single interrupt that belongs to both receive and transfer

- ➢ Each interrupt can be enabled or disabled by setting bits of the IE register. Likewise, the whole interrupt system can be disabled by clearing the EA bit of the same register. Refer to figure below.



	0	X	0	0	0	0	0	0	Value after Reset
IE	EA		ET2	ES	ET1	EX1	ET0	EX0	Bit name
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	

EA - global interrupt enable/disable:

- 0 - disables all interrupt requests.
- 1 - Enables all individual interrupt requests.

ES - enables or disables serial interrupt:

- 0 - UART system cannot generate an interrupt.
- 1 - UART system enables an interrupt.

ET1 - bit enables or disables Timer 1 interrupt:

- 0 - Timer 1 cannot generate an interrupt.
- 1 - Timer 1 enables an interrupt.

EX1 - bit enables or disables external 1 interrupt:

- 0 - change of the pin INT0 logic state cannot generate an interrupt.
- 1 - Enables an external interrupt on the pin INT0 state change.

ET0 - bit enables or disables timer 0 interrupt:

- 0 - Timer 0 cannot generate an interrupt.
- 1 - Enables timer 0 interrupt.

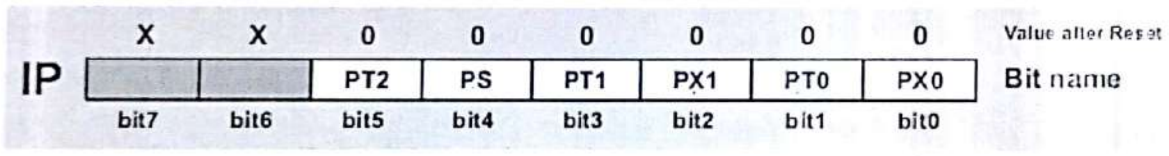
EX0 - bit enables or disables external 0 interrupt:

- 0 - change of the INT1 pin logic state cannot generate an interrupt.
- 1 - Enables an external interrupt on the pin INT1 state change.

Interrupt Priorities

- ➢ If several interrupts are enabled, it may happen that while one of them is in progress, another one is requested.
- ➢ The IP Register (Interrupt Priority Register) specifies which one of existing interrupt sources have higher and which one has lower priority. Interrupt priority is usually specified at the beginning of the program. According to that, there are several possibilities
  - If an interrupt of higher priority arrives while an interrupt is in progress, it will be immediately stopped and the higher priority interrupt will be executed first.
  - If two interrupt requests, at different priority levels, arrive at the same time then the higher priority interrupt is serviced first.
  - If the both interrupt requests, at the same priority level, occur one after another, the one which came later has to wait until routine being in progress ends.
  - If two interrupt requests of equal priority arrive at the same time then the interrupt to be serviced is selected according to the following priority list:
    1. External interrupt INT0
    2. Timer 0 interrupt
    3. External Interrupt INT1
    4. Timer 1 interrupt
    5. Serial Communication Interrupt

IP Register (Interrupt Priority) the IP register bits specify the priority level of each interrupt (high or low priority).



	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	
	X	X	0	0	0	0	0	0	Value after Reset
IP			PT2	PS	PT1	PX1	PT0	PX0	Bit name

PS - Serial Port Interrupt priority bit
PT1 - Timer 1 interrupt priority
PX1 - External Interrupt INT1 priority
PT0 - Timer 0 Interrupt Priority
PX0 - External Interrupt INT0 Priority

Interrupt Handling Process:

➢ Upon activation of an interrupt, the microcontroller goes through the following steps

1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack
2. It also saves the current status of all the interrupts internally (i.e: not on the stack)
3. It jumps to a fixed location in memory, called the interrupt vector table, that holds the address of the ISR

Interrupt Source	Vector (address)
IE0	0003 h
IE1	0013h
TF0	000B h
TF1	001B h
RI, TI	0023 h

The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it

4. It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RETI (return from interrupt)

Serial communication in 8051 micro controller :-

→ The serial port of 8051 is full duplex, i.e. it can Transmit and Receive simulteneously.

→ The Register SBUF is used to hold the data. The special function to hold data to be Transmitted out of the 8051 Via TXD. The other is the holds the Received data from External source vie RXD.

Serial port control Register (SCON) :-

→ Register SCON controls the Serial data Communication.

D7	D6	D5	D4	D3	D2	D1	D0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

REN → Receive ~~en~~ enable bit, If It is 1 Receiver pin is enabled, and Recives the serial data from RXD pin.

TB8 → Transmit bit '8'
RB8 → Receive bit '8'
] Both bits are to sensee the error in the Transmition and Receive opeation.

TI → Transmit Interrupt ~~flag~~ bit : It is going to be "high" When all the Eight bits are Transmitted from "SBUF" registel.

RI → Receive Interrupt bit : It is going to be "high" When all the eight bits are Received to "SBUF" rejiste

SM2 → multi processor communication bit : It is Coming to picture When, ~~8~~ 8051 Controller is in operation.

SM0, SM1 : These two bits determine the framing of data by specifying the no. of bits per character, and the start and stop bits.

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
1	1	Serial Mode 3.

→ Data Transmission:- Transmission of serial data begins at Any time when data is written to SBUF. Pin P3.1 (TXD) is used to transmit data to the serial data network. TI is set to '1' when data has been transmitted. This signifies the SBUF is empty so that another byte can be sent.

→ Data Reception:- Reception of serial data begins if the Receive enable bit is set '1' for all modes. Pin P3.0 (RXD) is used to Receive data from the serial data network. Receive Interrupt flag, RI is set after the data has been Received in all modes. The data gets stored in SBUF register from where it can be used.

→ Serial Data Transmission modes :-

MODE0 :- In This mode, The serial port works like a Shift Register and the data Transmission Works Synchronously With clock frequency.

MODE1 (standard UART mode) :- In mode-1, The serial port functions as a standard universal Asynchronous Receiver Transmitter mode (UART). 10bits are Transmitted through TXD (or) Received through RXD.

The 10 bits consists of one start bit (which is usually '0'), 8 data bits, and a stop bit (which is usually '1'). The baud rate is variable.

The following figure shows the way the bits are Transmitted / Received.

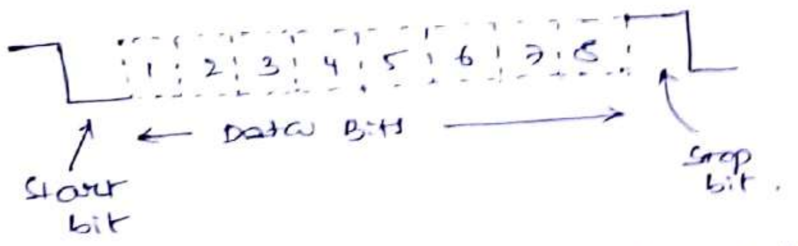


fig. Data transmission format in UART mode.

$$\text{Bit Time} = \frac{1}{f_{baud}}$$

Mode 2 :- In This mode Data Transmission is takes place with Constant baud rate. along with 1 start, 9bits, one stop bit.

mode 3 :- In This mode data Transmission is takes with 1 ~~stop~~ start, 9-bit, 1 stop bit. Variable baud rate along with

* We are Interested in mode 1 Serial data transmission. to Transfer data with Variable Speed. ~~[illegible]~~ along with UART along 1 Start, 8-bit, 1 stop. Transfer.

→ Programming ~~Sequence~~ the 8051 µC to transfer character bytes serially:-

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

Ex:- Write a program for the 8051 to transfer letter 'A' serially at baud rate 4800, continously.

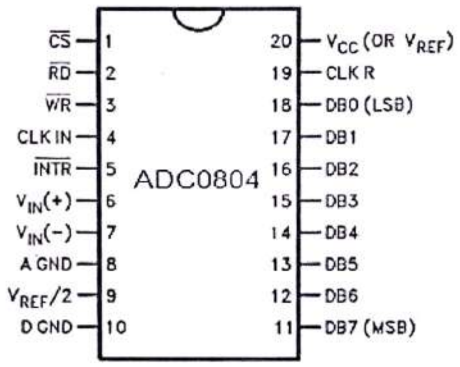
Ans:-

```
        MOV TMOD, #20H      ; Timer 1 mode 2
        MOV TH1, #FAH       ; 4800 Baud rate
        MOV SCON, #50H      ; 8-bit, 1 stop, 1 start bit, REN enabled.
        SETB TR1            ; Start Timer
AGAIN   MOV SBUF, #'A'      ; Letter 'A' To transfer
HERE    JNB TI, HERE        ; wait for the Last bit
        CLR TI              ; clear TI for next char
        SJMP ~~B~~ AGAIN    ; keep sending 'A'.
```

Interfacing of Analog-to-Digital Converter to 8051 microcontroller:

ADC 0804:

- ADC0804 is an 8 bit successive approximation analogue to digital converter from National semiconductors.
- The features of ADC0804 are differential analogue voltage inputs, 0-5V input voltage range, no zero adjustment, built in clock generator, reference voltage can be externally adjusted to convert smaller analogue voltage span to 8 bit resolution etc.
- The pin out diagram of ADC0804 is shown in the figure below.

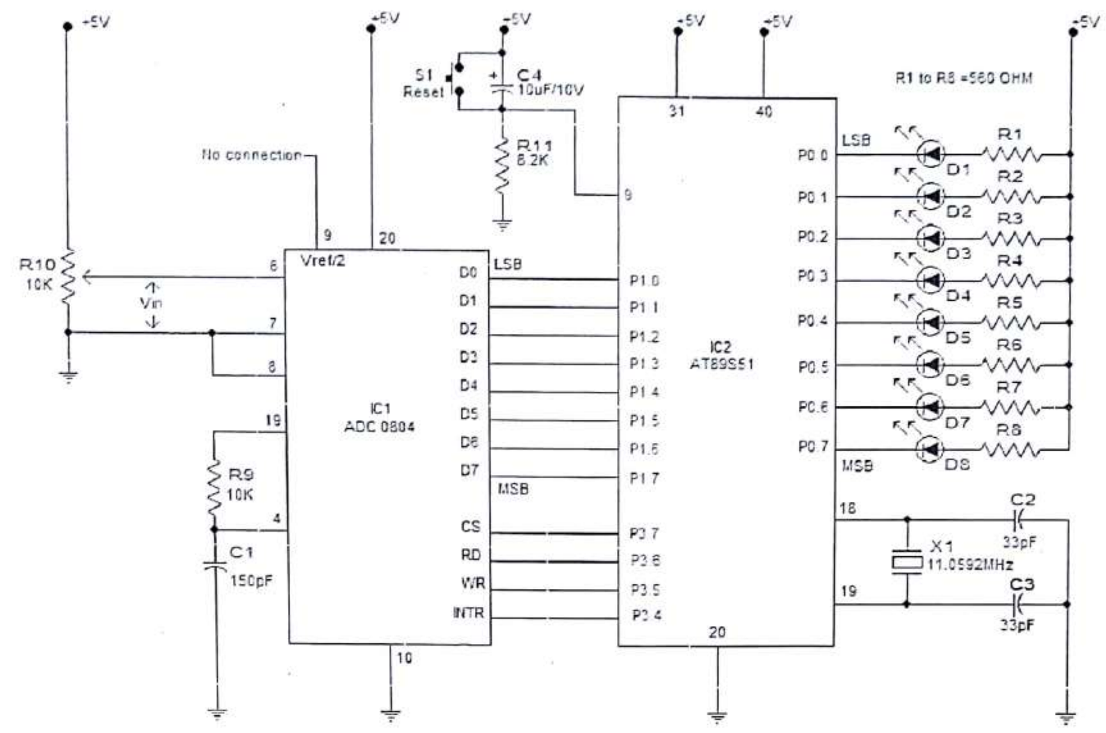


The voltage at Vref/2 (pin9) of ADC0804 can be externally adjusted to convert smaller input voltage spans to full 8 bit resolution.

Steps for converting the analogue input and reading the output from ADC0804:

- Make CS=0 and send a low to high pulse to WR pin to start the conversion.
- Now keep checking the INTR pin. INTR will be 1 if conversion is not finished and INTR will be 0 if conversion is finished.
- If conversion is not finished (INTR=1), poll until it is finished.
- If conversion is finished (INTR=0), go to the next step.
- Make CS=0 and send a high to low pulse to RD pin to read the data from the ADC.

Circuit Diagram:



- The figure above shows the schematic for interfacing ADC0804 to 8051.
- The circuit initiates the ADC to convert a given analogue input, then accepts the corresponding digital data and displays it on the LED array connected at P0.
- For example, if the analogue input voltage Vin is 5V then all LEDs will glow indicating 11111111 in binary which is the equivalent of 255 in decimal.
- AT89s51 is the microcontroller used here. Data out pins (D0 to D7) of the ADC0804 are connected to the port pins P1.0 to P1.7 respectively.
- LEDs D1 to D8 are connected to the port pins P0.0 to P0.7 respectively.
- Resistors R1 to R8 are current limiting resistors. In simple words P1 of the microcontroller is the input port and P0 is the output port.
- Control signals for the ADC (INTR, WR, RD and CS) are available at port pins P3.4 to P3.7 respectively.
- Resistor R9 and capacitor C1 are associated with the internal clock circuitry of the ADC.
- Preset resistor R10 forms a voltage divider which can be used to apply a particular input analogue voltage to the ADC. Push button S1, resistor R11 and capacitor C4 forms a debouncing reset mechanism.

- Crystal X1 and capacitors C2, C3 are associated with the clock circuitry of the microcontroller.

<u>Program:</u>

```
ORG 00H
        MOV P1, #11111111B                              // initiates P1 as the input port
MAIN: CLR P3.7                                          // makes CS=0
        SETB P3.6                                       // makes RD high
        CLR P3.5                                        // makes WR low
        SETB P3.5                  // low to high pulse to WR for starting conversion
WAIT: JB P3.4, WAIT                                     // polls until INTR=0
    CLR P3.7                                            // ensures CS=0
    CLR P3.6               // high to low pulse to RD for reading the data from ADC
    MOV A, P1                                // moves the digital data to accumulator
    CPL A                              // complements the digital data
    MOV P0, A                              // outputs the data to P0 for the LEDs
    SJMP MAIN                              // jumps back to the MAIN program
    END
```

Digital-to-analog (DAC) converter interfacing with 8051 Micro controller:

- ➢ The digital-to-analog converter (DAC) is a device widely used to convert digital pulses to analog signals.

MC1408 DAC (or DAC0808)

- ➢ In the MC1408 (DAC0808), the digital inputs are converted to current (I_{out}), and by connecting a resistor to the I_{out} pin, we get the result to voltage.

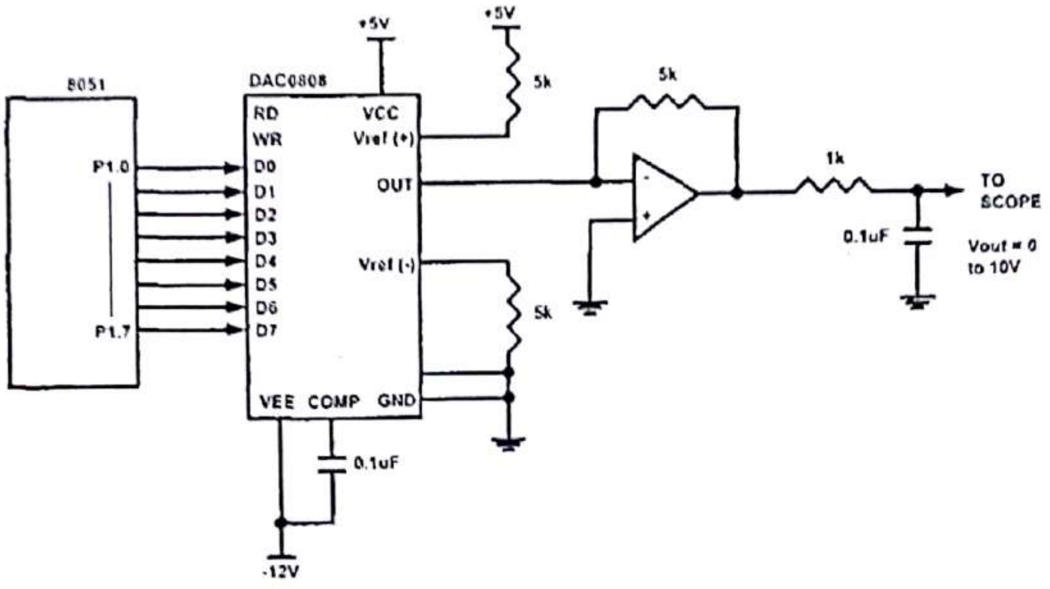


Fig. interfacing circuit

The total current provided by the I_{out} pin is a function of the binary numbers at the DO – D7 inputs of the DAC0808 and the reference current (I_{ref}), and is as follows:

$$I_{out} = I_{ref}\left(\frac{D7}{2}+\frac{D6}{4}+\frac{D5}{8}+\frac{D4}{16}+\frac{D3}{32}+\frac{D2}{64}+\frac{D1}{128}+\frac{D0}{256}\right)$$

- ➢ Where DO is the LSB, D7 is the MSB for the inputs, and I_{ref} is the input current that must be applied to pin 14. The I_{ref} current is generally set to 2.0 mA.

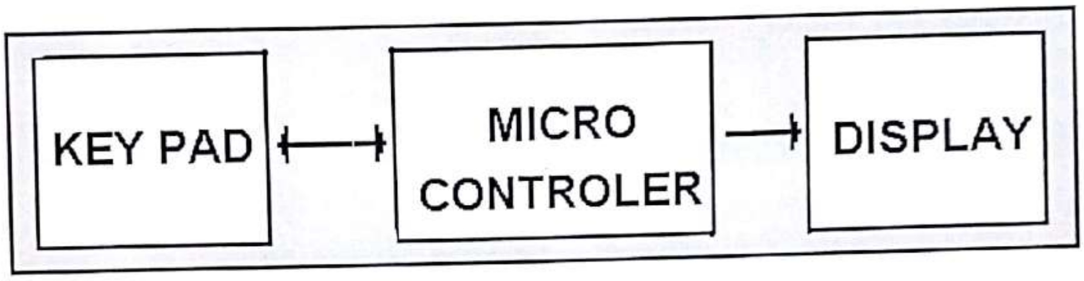
- Figure above shows the generation of current reference (setting $I_{ref} = 2$ mA) by using the standard 5-V power supply and IK and 1.5K-ohm standard resistors.

Program to send data to the DAC to generate a stair-step ramp:

```
              CLR    A
AGAIN:        MOV    P1,A            ;send data to DAC
              INC    A               ;count from 0 to FFH
              ACALL  DELAY           ;let DAC recover
              SJMP   AGAIN
```

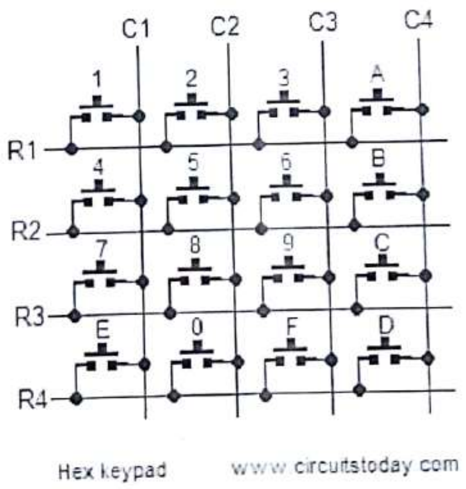
Key board interfacing to 8051 micro controller:

- The key board here we are interfacing is a matrix keyboard. This key board is designed with a particular rows and columns. These rows and columns are connected to the microcontroller through its ports of the micro controller 8051.
- When ever a key is pressed, a row and a column gets shorted through that pressed key and all the other keys are left open. When a key is pressed only a bit in the port goes high. Which indicates microcontroller that the key is pressed. By this high on the bit key in the corresponding column is identified.



Circuit diagram of *INTERFACING KEY BOARD TO 8051.*

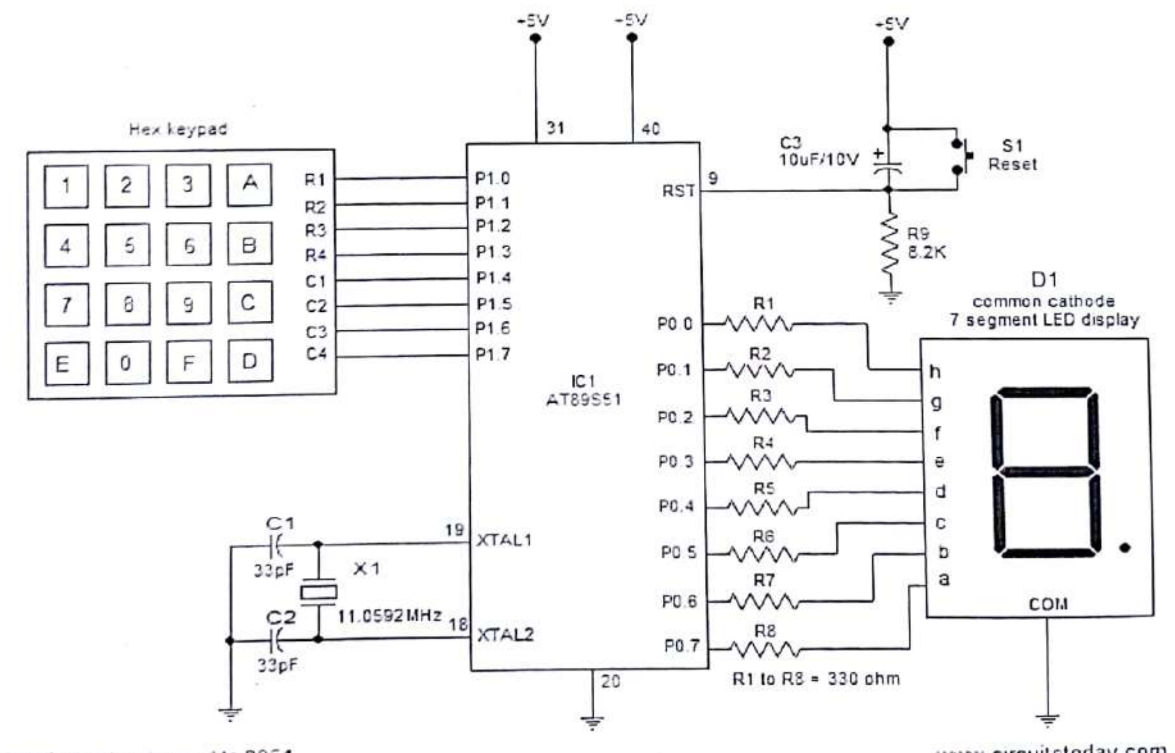
The programming algorithm, program and the circuit diagram is as follows. Here program is explained with comments.



Circuit diagram of *INTERFACING KEY BOARD TO 8051.*

- The hex keypad has 8 communication lines namely R1, R2, R3, R4, C1, C2, C3 and C4.
- R1 to R4 represents the four rows and C1 to C4 represents the four columns.
- When a particular key is pressed the corresponding row and column to which the terminals of the key are connected gets shorted.
- For example if key 1 is pressed row R1 and column C1 gets shorted and so on. The program identifies which key is pressed by a method known as column scanning.
- In this method a particular row is kept low (other rows are kept high) and the columns are checked for low. If a particular column is found low then that means that the key connected between that column and the corresponding row (the row that is kept low) is been pressed.
- For example if row R1 is initially kept low and column C1 is found low during scanning, that means key 1 is pressed.

The circuit is very simple and it uses only two ports of the microcontroller, one for the hex keypad and the other for the seven segment LED display.



Interfacing hex keypad to 8051

www.circuitstoday.com

Program to interface matrix keyboard to microcontroller 8051

Program:

To check that whether any key is pressed

```
ORG 0040H
BACK: MOV P1, #11111111B        // loads P1 with all 1's
      CLR P1.0                  // makes row 1 low
      JB P1.4, NEXT 1           // checks whether column 1 is low and jumps to NEXT1 if not low
      MOV A, #F9                // loads a with 0D if column is low (that means key 1 is pressed)
      MOV P0, A
NEXT1: JB P1.5, NEXT 2          // checks whether column 2 is low and so on...
       MOV A, #A4
       MOV P0, A
NEXT2: JB P1.6, NEXT 3
       MOV A, #B0
       MOV P0, A
NEXT3: JB P1.7, NEXT 4
       MOV A, #XX
       MOV P0, A
NEXT4: SETB P1.0
       CLR P1.1
       JB P1.4, NEXT 5
       MOV A, #99
       MOV P0, A
NEXT5: JB P1.5, NEXT 6
```

```
          MOV A, #92

          MOV P0, A

NEXT6: JB P1.6, NEXT 7

          MOV A, #82

          MOV P0, A

NEXT7: JB P1.7, NEXT 8

          MOV A, #..

          MOV P0, A

NEXT8: SETB P1.1

         CLR P1.2

         JB P1.4, NEXT 9

         MOV A, #F8

         MOV P0, A

NEXT9: JB P1.5, NEXT 10

         MOV A,#80

         MOV P0, A

NEXT10: JB P1.6, NEXT 11

         MOV A,#90

         MOV P0, A

NEXT11: JB P1.7

         MOV A,#..

         MOV P0, A

 SJMP BACK
```

LED (Light Emitting Diodes)

- Light Emitting Diodes (LED) is the most commonly used components, usually for displaying pins digital states. Typical uses of LEDs include alarm devices, timers and confirmation of user input such as a mouse click or keystroke.

Interfacing LED

- Fig. 1 shows how to interface the LED to microcontroller. As you can see the cathode is connected through a resistor to GND & the anode is connected to the Microcontroller pin. So when the Port Pin is HIGH the LED is ON & when the Port Pin is LOW the LED is turned OFF.

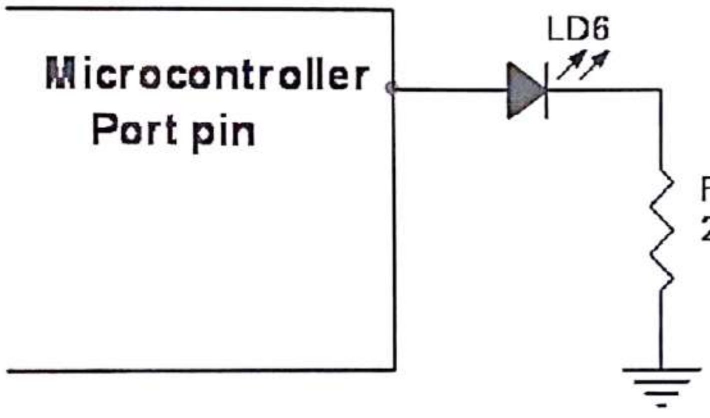
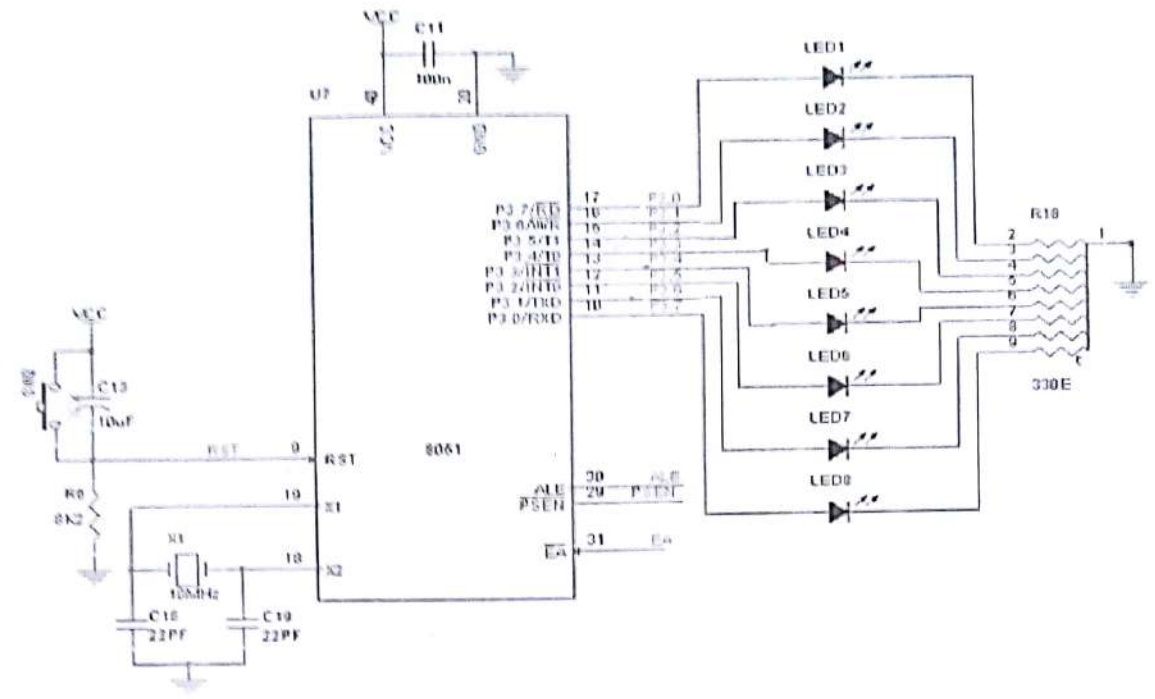


FIG.1.

- Interfacing LED with 8051 we now want to flash a LED in 8051 Primer Board. It works by turning ON a LED & then turning it OFF & then looping back to START.
- However the operating speed of microcontroller is very high so the flashing frequency will also be very fast to be detected by human eye. The 8051 Primer board has eight numbers of point LEDs, connected with I/O Port lines (P3.0 – P3.7) to make port pins high.

Circuit Diagram to Interface LED with 8051



PROGRAM :

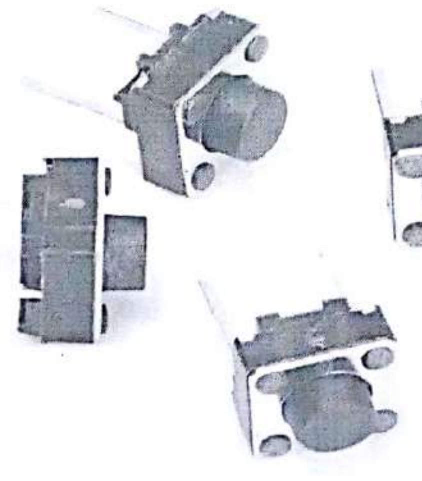
This is the program to flash one by one LED from LED1 to LED8 in above interfacing circuit.

```
ORG 0040H

; MAIN PROGRAM

TOGGLE: MOV A, #01H

         MOV P3, A

         CALL DELAY

         MOV A, #02H

        MOV P3, A

        CALL DELAY

        MOVA, #03H
        MOV P3, A
        CALL DELAY
        MOV A, #04H
        MOV P3, A
        CALL DELAY
        MOV A, #05H
        MOV P3, A
        CALL DELAY
        MOV A, #06H
        MOV P3, A
        CALL DELAY
        MOV A, #07H
        MOV P3, A
        CALL DELAY
        MOV A, #08H
        MOV P3, A
        CALL DELAY
        SJMP TOGGLE

; DELAY SUB-ROUTINE
DELAY:      MOV R5, #10
FIRST:      MOV R6, #200
SECOND:     MOV R7, #200
```

```
THIRD     DJNZ R5, FIRST
          DJNZ R6, SECOND
          DJNZ R7, THIED
          RET
```

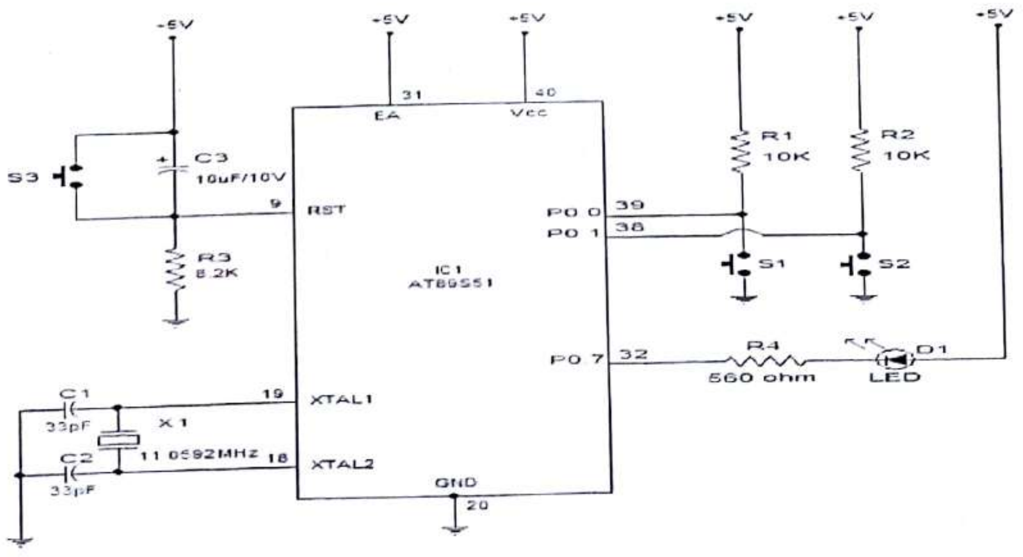
Interfacing push button switches to an 8051 microcontroller:

- Push button switches are widely used in embedded system projects and the knowledge about interfacing them to 8051 is very essential in designing such projects.
- A typical push button switch has two active terminals that are normally open and these two terminals get internally shorted when the push button is depressed.



Pushbutton switch

Circuit diagram:



Interfacing 8051 and pushbutton:

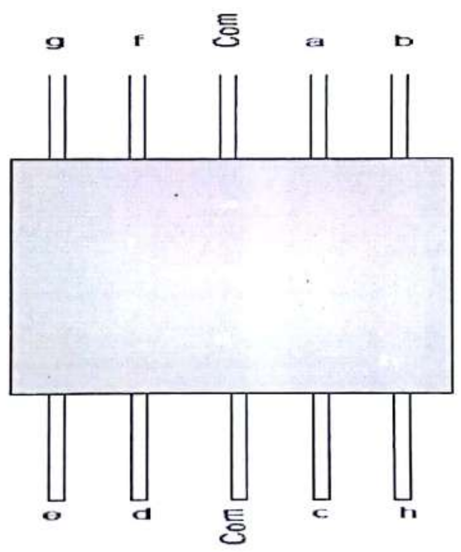
- The circuit diagram for interfacing push button switch to 8051 is shown above. AT89S51 is the microcontroller used here. The circuit is so designed that when push button S1 is depressed the LED D1 goes ON and remains ON until push button switch S2 is depressed and this cycle can be repeated.
- Resistor R3, capacitor C3 and push button S3 forms the reset circuitry for the microcontroller. Capacitor C1, C2 and crystal X1 belongs to the clock circuitry. R1 and R2 are pull up resistors for the push buttons. R4 is the current limiting resistor for LED.

PROGRAM

```
            MOV P0,#83H
READSW: MOV A,P0
            RRC A
            JC NXT
            CLR P0.7
            SJMP    READSW
     NXT: RRC A
            JC        READSW
            SETB P0.7
            SJMP     READSW
END
```

Interfacing seven segment displays with 8086 micro processor:

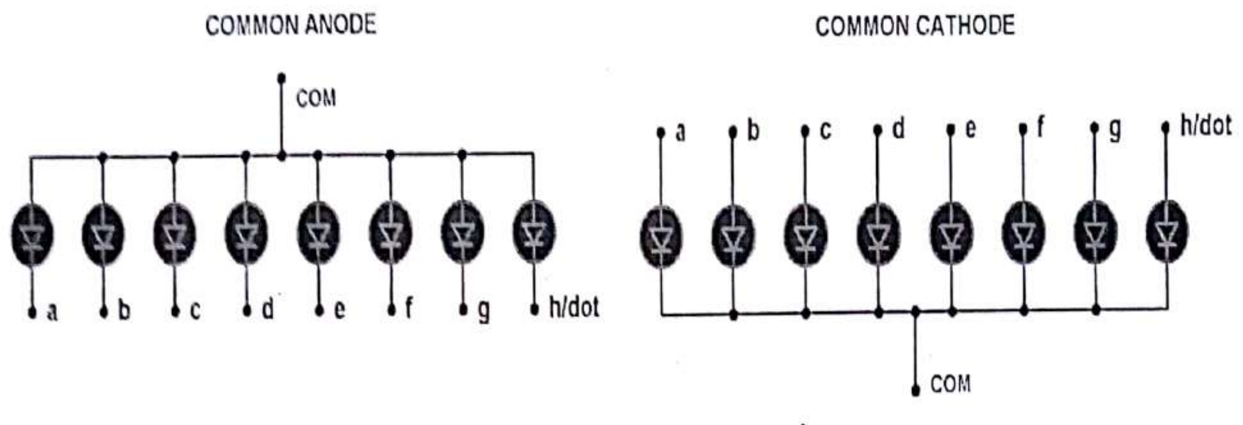
- Seven segment displays are important to display numbers from 0 to 9. It can also display some character alphabets like A,B,C,H,F,E etc.
- 7 segment displays is the simplest unit to display numbers and characters. It just consists of 8 LEDs, each LED used to illuminate one segment of unit and the 8thLED used to illuminate DOT in 7 segment display.
- We can refer each segment as a LINE, as we can see there are 7 lines in the unit, which are used to display a number/character.
- We can refer each line/segment "a,b,c,d,e,f,g" and for dot character we will use "h".
- There are 10 pins, in which 8 pins are used to refer a,b,c,d,e,f,g and h/dp, the two middle pins are common anode/cathode of all he LEDs.
- These common anode/cathode are internally shorted so we need to connect only one COM pin.



- There are two types of 7 segment displays: Common Anode and Common Cathode:

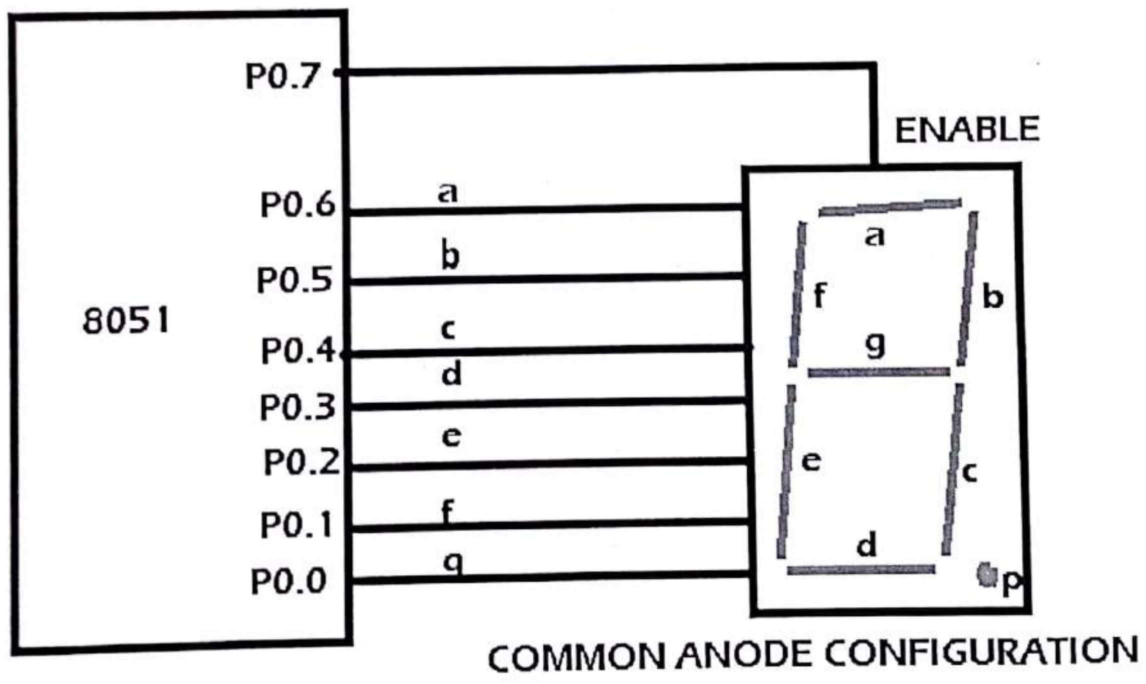
Common Anode: In this all the Negative terminals (cathode) of all the 8 LEDs are connected together (see diagram below), named as COM. And all the positive terminals are left alone.

Common Cathode: In this all the positive terminals (Anodes) of all the 8 LEDs are connected together, named as COM. And all the negative thermals are left alone.



Internal connections of 7 Segment Display

Circuit Diagram and Working Explanation



COMMON ANODE CONFIGURATION

We have connected a,b,c,d,e,f,g,h to pins 0.0 to 0.7 means we are connecting 7 segment to port 0 of microcontroller.

- Now suppose we want to display 0, then we need to glow all the LEDs except LED which belongs to line "g" (see diagram above), so pins 0.0 to 0.6 should be at 0 (should be 0 to TURN ON the LED as per negative logic) and pin 0.7 and 0.8 should be at 1 (should be 1 to TURN OFF the LED as per negative logic).
- So the LEDs connected to pins 0.0 to 0.6 (a,b,c,d,e,f) will be ON and LEDs connected to 0.7 and 0.8 (g and h) will be OFF, that will create a "0" in 7 segment.
- So we need bit pattern 11000000 (Pin 8 is the highest bit so starting from P0.7 to P0.0), and the HEX code for binary 11000000 is "C0". Similarly we can calculate for all the digits.
- Here we should note that we are keeping "dot/h" always OFF, so we need to give LOGIC "1" to it every time. A table has been given below for all the numbers while using Common Anode 7 segment.

Digit to Display	**h g f e d c b a**	**Hex code**
0	11000000	C0
1	11111001	F9
2	10100100	A4
3	10110000	B0
4	10011001	99
5	10010010	92
6	10000010	82
7	11111000	F8

8	10000000	80
9	10010000	90

Code Explanation

- We have created ms delay function to provide the delay in milliseconds, this delay is usually provided in any microcontroller program so that microcontroller can complete its internal operation.
- Then we have created an array of the hex codes for 0 to 9 (see table above), and finally we have sent the hex codes to the port 0, which is connected to common anode 7 segment. So in this way the numbers are shown on the 7 segment display.

Code:

- ORG 0000H

```
REPEAT:
MOV P0,# 11000000B                      ; DISPLAYING 0
ACALL DELAY
MOV P0,# 11111001B                      ; DISPLAYING 1
ACALL DELAY
MOV P0,# 10100100B                      ; DISPLAYING 2
ACALL DELAY
MOV P0,# 10110000B                      ;   DISPLAYING 3
ACALL DELAY
MOV P0,# 10011001B                      ; DISPLAYING 4
ACALL DELAY
MOV P0,# 10010010B                       ; DISPLAYING 5
ACALL DELAY
MOV P0,# 10000010B                       ; DISPLAYING 6
ACALL DELAY
```

```
        MOV P0,# 11111000B                    ; DISPLAYING 7
        ACALL DELAY
        MOV P0,#10000000B                     ; DISPLAYING 8
        ACALL DELAY
        MOV P0,# 10010000B                    ; DISPLAYING 9
        ACALL DELAY
        SJMP REPEAT


; DELAY SUB-ROUTINE

DELAY:      MOV R5, #10

FIRST:      MOV R6, #200

SECOND:     MOV R7, #200

  THIRD    DJNZ R5, FIRST

           DJNZ R6, SECOND

           DJNZ R7, THIED

           RET
```