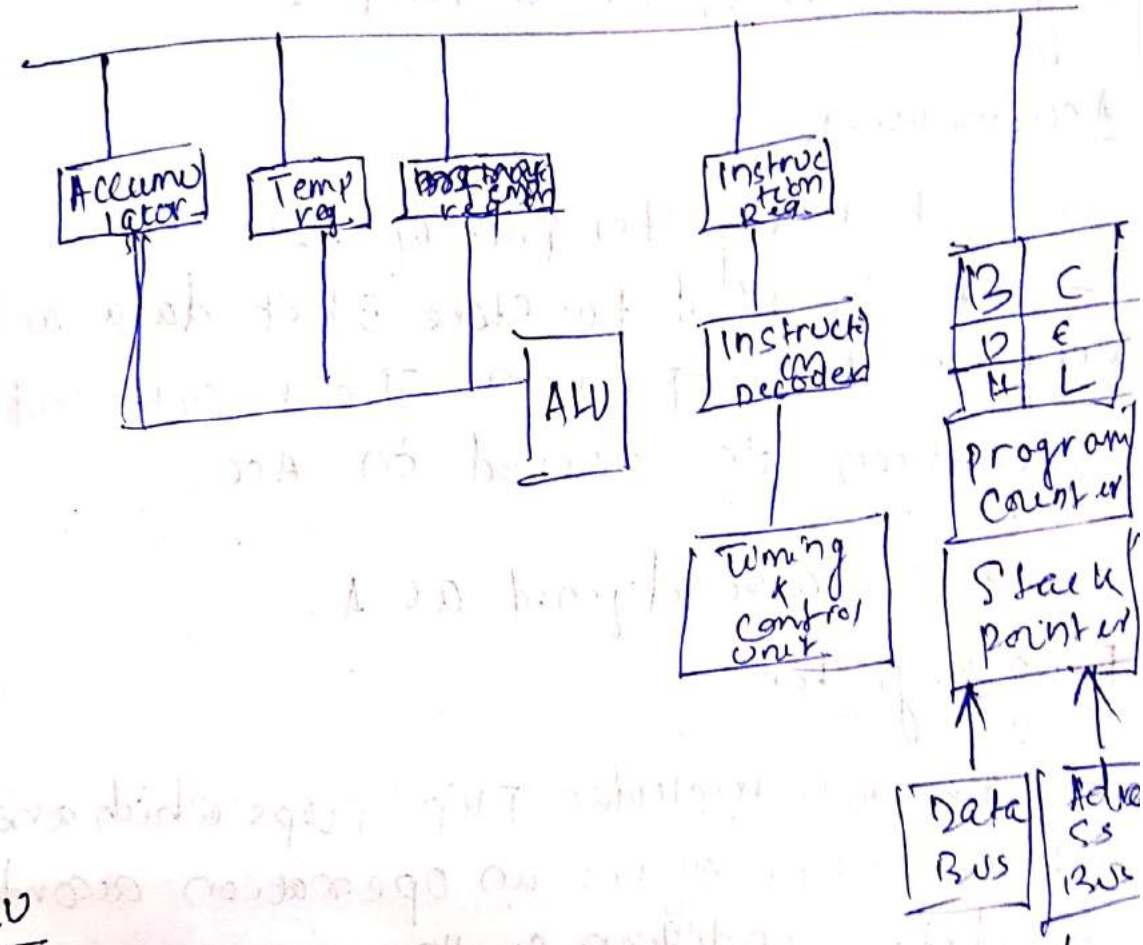
# Microprocessor 8085

## Architecture

- The 8085 mp is a 8 bit processor available as a 40 pin IC package and uses +5 V For power.

- It can run maximum Frequency of 3 MHz.




## ALU

It performs the actual numerical and Logical operation such as Addition, Subtraction, AND, OR etc. It uses data from memory and from Accumulator to perform operation.

## Register

- It includes six register. one Accumulator, one Flag register. In addition it has 2 16 bit reg, Stack pointer and Program counter.

- It has also six 8 bit general Purpose register. B, C, D, E, H, and L.

## Accumulator

- 8 bit register part of ALU.
- It is used to store 8 bit data and to perform ALU operation. The result and operation is stored in Acc.
- It is also defined as A.

## Flag register

→ The ALU includes Flip-Flops which are set or reset after an operation according to data condition of the result in Accumulator and register. They are Zero carry, Sign, Parity, and Auxiliary Carry.

## Program counter

- This 16 bit register deals with sequencing the execution of instruction.
- This reg is a memory pointer,
⇒ The mp uses this reg to sequence the execution of instruction.
- Function of Program counter is to point the memory address from which next byte is to be Fetched.

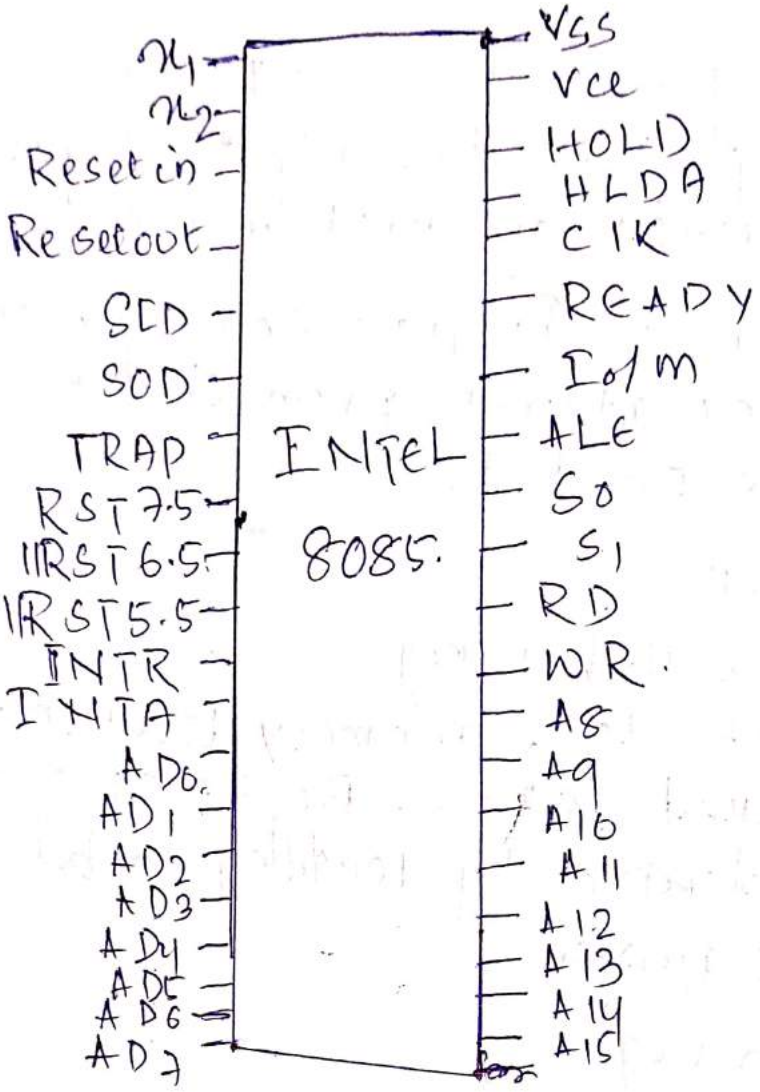
## Stack Pointer

- It is a 16 bit reg
- It points to a memory location in R/W memory called Stack. Beginning of the Stack is defined by loading 16 bit address in Stack pointer.

## Instruction reg

- It is a 8 bit register.
- It stores opcode of an instruction.

# 8085 pin Description




$A_8 - A_{15}$ : Eight MSB of memory address and 8 bits of I/O Address.

$AD_0 - AD_7$ : Lower significant 8 bit of memory address and 8 bit of the I/O address during first clock cycle.

## control & status signal

ALE : Address Latch Enable signal.

RD : Read control signal

WR : Write control signal

IO/M, $S_0$, $S_1$ - Status signal.

$V_{CC}$ = +5 v dc power supply

$V_{SS}$ = Ground state.

$X_1$, $X_2$ : A crystal having 6 MHZ Freq is connected through these pins.

CLK : Clock output.

Reset in : When the signal on this pin is Low, pc is set to 0 and buses are tricstated.

Reset out : This indicates the processor is reset.

Ready : When signal is Low, the Processor waits for Integral clock cycle.

HOLD : It indicates Peripheral like DMA controller is requesting the use of busses.

HLDA : Signal acknowledges HOLD request.

INTR : General purpose Interrupt.

RST 7.5, RST 6.5, RST 5.5 : Restart Interrupts

SID : Serial Input signal

SOD : Serial output signal.

## Instruction Set

Based on the design of the ALU and decoding unit, mp provides instruction. It is devided in 5 groups.

- Data Transfer Group.
- Arithmetic Group
- Logical Group
- Branch control Group
- Machine control Group.

## Addressing Mode

The process of specifying the data to be operated by the instruction is called addressing.

- classified into 5 types of addressing.
  1) Immediate Addressing
  2) Memory Direct Addressing.
  3) Indirect Addressing
  4) Register Addressing
  5) Immediate Addressing.

## Immediate Addressing

In this mode operand is given in the instruction. a byte or coord transfers to the destination reg or memory Location.

Ex. MVI A 9A H.

- Operand is a part of instruction

## Direct Addressing

It moves a byte or word between memory Location and register.

eg: LDA 850FH

This instruction is used to load the content of memory address 850FH in Accumulator.

## Register ~~Indirect~~ direct

It transfer the copy of a byte or word from source reg to destination reg.

ex. MOV B, C

It copies content of C to B.

## Indirect ~~Register~~ Addressing

It transfers a byte between register and memory location

Ex. MOV A M.

Here data in the memory location pointed by HL pair. The data is moved to Accumulator.

# Instruction set

## Data Transfer Group

| Opcode | Operand | Description |
|---|---|---|
| MOV | Rd Rs<br>M Rs<br>Rd M | This instruction copies the contents of the source register into the destination register |
| MVI | Rd data<br>M data | The 8 bit data is stored in the destination reg or memory. If operand is in ML its location is specified by the content of HL register. |
| LDA | 16 bit address | The data of the 16 bit memory is loaded to Accumulator |
| LDAX | B/D reg pair | The content of designed register pair point to a memory location. |
| LXI | Reg. pair, 16 bit data | It loads 16 bit data in the register pair designed in operand. |
| LHLD | 16 bit address | The instruction copies the content of ML pointed out by 16 bit address into reg L and L copies the content of next ML into H. |

| Opcode | Operand | Description |
|---|---|---|
| STA | 16 bit address | The content of the Accumulator copied into the mL specified by the Operand. |
| STAX | Reg pair | The contents of the Accu. are copied into the memory location specified by contents of the Operand. |
| SHLD | 16 bit Address | The contents of reg L are Stored into mL Specified by 16 bit Address in the operand and contents of H are Stored into next mL by incrementing operand. |
| XCHG | None | content of D-E and H-L are exchanged. |
| XTHL | None | content of L reg are exchanged with the Stack location pointed out by the contents of the Stack pointer register. |

## Arithmetic Group

| Opcode | Operand | Description |
|---|---|---|
| ADD | R, M | The content of reg and mL are added to Accumulator. |
| ADC | R, M | The content of reg and mL are added to Accumulator if there is a carry. |
| ADI | 8 bit data | The immediate data is added to Accumulator |
| ACI | 8 bit data | The immediate data is added to Acc if there a carry. |
| DAD | Reg pair | 16 bit content of the Specified reg pair are added to content of HL register and sum is Stored in Acc. |
| SUB | R, M. | content of register and mL are substracted from Acc. |
| SBB | R, M. | content of register and mL are substracted from Accumulator if there is a bm |

| | | |
|---|---|---|
| SUI | 8 bit data | The 8 bit data is subtracted from Accu. |
| SBI | 8 bit data | The 8 bit data is subtracted from Accumu when there is a borrow. |
| INR R, M | | The content of designed register are incremented by 1 and result is stored in same place |
| INX R | | content of the designed register pair are incremented by 1 and result is placed in same place |
| DCR R, M | | content of designed reg are decremented by 1 and result is stored in same place. |
| DCX R | | content of designated reg pair are decremented by 1 and result is stored in same place. |
| DAA | None | It converts the Hexadecimal data to decimal. |

# Branch Instruction

| Opcode | Operand | Description |
|---|---|---|
| JMP | 16 bit address | The program sequence is transferred to memory location specified by 16 bit address |
| Jump conditionally | | Program sequence is transferred to ML specified by 16 bit address given in the operand based on flag. |
| JC | Jump on carry | |
| JNC | Jump not carry | |
| JP | Jump on positive | |
| JM | Jump on Negative | |
| JZ | Jump on Zero | |
| JNZ | Jump on no Zero | |
| JPE | Jump on even Parity | |
| JPO | Jump on odd Parity | |
| CALL (unconditional) | 16 bit address | Program sequence is transferred to ML specified by 16 bit address given in the operand unconditionally |
| CALL (conditional) | 16 bit address. | |

| | | |
|---|---|---|
| CC | Call on carry | Cy = 1 |
| CNC | Call on no carry | cy = 0 |
| CP | Call on positive | S = 0 |
| CM | Call on minus | S = 1 |
| CZ | Call on Zero | Z = 1 |
| CNZ | Call on no Zero | Z = 0 |
| CPE | Call on even parity | P = 1 |
| CPO | Call on odd parity | P = 0 |

Return From subroutine:

| | | |
|---|---|---|
| RET (unconditional) | None | Program sequence is transffed from subroutine to the calling Program |

RET (conditional) None

| | |
|---|---|
| RC | Return on carry |
| RNC | Return on No carry |
| RP | Return on positive |
| RM | Return on Negative |
| RZ | Return on zero |
| RNZ | Return on No Zero |
| RPE | Return on even parity |
| RPO | Return on odd parity. |

| | | |
|---|---|---|
| PCHL | None | content of register H and L are copied in the program counter. |
| Restart (RST) 0-7 | | It is equivalent to a 1 byte call instruction to one of the 8 ml depending upon the nm. |

## Logical Instruction

| Opcode | operand | Description |
|---|---|---|
| CMP | R, M | content of the operand are compared with the contents of Accumulator |
| CPI | 8 bit data | 2nd byte is compared with the contents of the Accumulator. The values being compared remain unchanged. |
| ANA | R, M | content of the accumulator are logically ANDed with the contents of operand and result is placed in the Accumulator. |
| ANI | 8 bit data | contents of Accumulator are logically ANDed with the 8 bit data and result is placed in Accumulator. |

| Opcode | Operand | Description |
|---|---|---|
| XRA | R, M | The contents of Accumulator are exclusive ORed with the content of operand and result is placed in Accumulator. |
| XRI | 8 bit data | content of Accumulator are exclusive ORed with 8 bit data and the result is placed in the Acc. |
| ORA | R, M | content of accumulator are logically ored with the content of operand and the result is placed in the accumulator. |
| ORI | 8 bit data | content of Accumulator are logically ored with 8 bit data and result is placed in Accumulator. |
| RLC | None | Each binary bit of Acc. is rotated left by one position. |
| RRC | None | Each binary bit of Accumulator is rotated to its right. |
| RAL | None | Each binary bit of accumulator is rotated to the left through carry bit. |

| Opcode | Operand | Description |
|---|---|---|
| CMA | None | Content of accumulator is complimented. |
| CMC | None | Carry Flag is complimented. |
| STC | None | Carry Flag is Set to 1. |

## Control Instruction

| | | |
|---|---|---|
| NOP | None | No operation is performed. |
| HLT | None | CPU finishes executing the current instruction and halts further instruction. |
| DI | None | It enable Flip-Flop is reset and all interrupts are disabled. |
| EI | None | Interrupt Flip-Flop is Set and all interrupts are enabled. |
| RIM | None | It is a multipurpose instruction used to read the Status of Interrupt and read Serial data input. |

# Microprocessor Programming

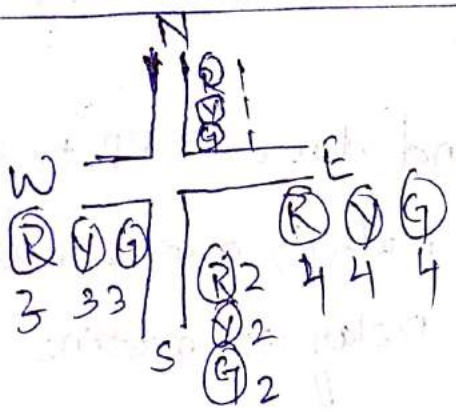


## Traffic Light programming

* Design a Microprocessor system to control traffic lights. The traffic light arrangements is as shown in Figure. Traffic should be controlled in the following manner.

- Allow traffic From W to E and E to W transition for 20 sec.
- Give Transition Period 5 sec (yellow ON)
- Allow Traffic From N to S and S to N for 20 sec.
- Give transition period 5 sec (Green)

## Hardware For Traffic Light.




(Digram to show 12 electric bulb)

| Pins | Light | Pins | Light |
|---|---|---|---|
| $PA_0$ | $R_1$ | $PB_0$ | $R_3$ |
| $PA_1$ | $Y_1$ | $PB_1$ | $Y_3$ |
| $PA_2$ | $G_1$ | $PB_2$ | $G_3$ |
| $PA_3$ | $R_2$ | $PB_3$ | $R_4$ |
| $PA_4$ | $Y_2$ | $PB_4$ | $Y_4$ |
| $PA_5$ | $G_2$ | $PB_5$ | $G_4$ |

(Electric bulb are controlled by $R_{2,4}$ 8255 pins are used to control relay on-off action with driver ckt.)

## I/O map

| Ports | Address Line $A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0$ | Address |
|---|---|---|
| Port A | 10000000 | 80H |
| Port B | 10000001 | 81H |
| Port C | 10000010 | 82H |
| control Reg | 10000011 | 83H |

## Source Program

```
MVI A 80H      Initialize 8255 Port A and Port
OUT 83H        in output mode          B
START: MVI A 09H
   OUT 80H (PA)   Send data on PB to glow R1 & R2
   MVI A 24H
   OUT 81H (PB)   Send data on PB to glow G3 & G4
   MVI C 28H      Load multiply count for delay
   CALL Delay     Call Delay subroutine
   MVI A 12H
   OUT (81H) PB   Send data on Port A to glow Y1 and Y2
   OUT (81H) PB   send data on Port B to glow Y3 & Y4
   MVI C, 0A H    Load multiplier count.
   CALL Delay
   MVI A 24H
   OUT (80H) PA   Send data on Port A to glow G1 & G2
   MVI A 09H
```

```
OUT (81H) PB        ; Send data on port B to glow R3 & R4
MVI C 28H           ; Load multiplier count for Delay
CALL Delay
MVI A 12H
OUT PA              ; Send data on port A to glow Y1 & Y2
OUT PB              ; Send data on port B to glow Y3 & Y4
MVI C, 04H          ; Load multiplier count.
CALL Delay
JMP START.
```

Delay subroutine

```
Delay: LXI D count  ; Load count For 0.5sec Delay
BACK : DCX D        ; Decrement counter
MOV A D
ORA E
JNZ Back            ; If Not 0 repeat.
DCR C
JNZ Delay
RET.
```

* Addition of Two 8 bit Nm: (Programming)

Program

| | | |
|---|---|---|
| MVI | C 00 | Initialize C reg to 00 |
| LDA | 4150 | Load the value to Accumulator |
| MOV | B A | Move the content of Accumulator to register B |
| LDA | 4151 | Load the value in A. |
| ADD | B | Add register B with A. |
| JNC | LOOP | Jump on no carry |
| INR | C | increment C |
| STA | 4152 | Store the value in Acc. |
| STA | 4153 | Store the value in Acc (Carry) |
| HLT | | Program ends. |

*

# Microprocessor
## Interfacing Programming

* Write a program to initiate ADC and to store the digital data in memory

Program:

```
        MVI   A 10
        OUT   C8
        MVI   A 18
        OUT   C8
        MVI   A 10
        OUT   D0
        XRA   A
        XRA   A
        XRA   A
        MVI   A 00
        OUT   D0
Loop    IN    D8
        ANI   01
        CPI   01
        JNZ   LOOP
        IN    C0
        STA   4150
        HLT.
```

# 8051 Microcontroller Program

* W.A.P to add two 8 bit number

Programming

```
ORG   4100
CLR   C
MOV   A ≠ data1
ADD   A ≠ data2
MOV   DPTR ≠ 4500
MOV X @ DPTR A
SJMP  HERE.
```

* W.A.P to perform subtraction of two 8 bit n using 8051 instruction set.

```
ORG   4100
CLR   C
MOV   A ≠ data1
SUBB  A ≠ data2
MOV   DPTR ≠ 4500
MOV X @ DPTR A
SJMP  HERE.
```

* W.A.P to perform multiplication of two 8 nu

```
ORG   4100
CLR   C
MOV   A ≠ data
```

```
MOV     B ≠ data2
MUL     AB
MOV     DPTR ≠ 4500
MOVX    @ DPTR A
INC     DPTR.
MOV     A, B
MOV X   @ DPTR, A
SJMP    HERE.
```

* W.A.P to exhibit RAM direct addressing and bit addressing scheme of 8051 microcontroller.

Program

Bit Addressing:

```
SET B   PSW 3
MOV     IRO ≠ data1
MOV     A ≠ data2
ADD     A R0
MOV     DPTR # 4500
MOV X   @ DPTR A
SJMP    HERE
```

Direct Addressing

```
MOV     30 ≠ data1
MOV     A ≠ data2
ADD     A, 30
MOV     DPTR ≠ 4500
```

* W.A.P to interface stepper motor with 8051 parellel port and to vary speed of motor direction of motor.

Program:

```
        ORG   4100
START.  MOV   DPTR #4500H
        MOV   R0 #04
AGAIN   MOVX  A @DPTR.
        PUSH  DPH
        PUSH  PDL
        MOV   DPTR # FFC0H
        MOV   R2 04H
        MOV   R1 # FFH
DLY1:   MOV   R3 #FFH
        DJNZ  R3 DLY
        DJNZ  R1 DLY1
        DJNZ  R2 DLY1
        MOV X @ DPTR A
        POP   DPL
        INC   DPTR.
        DJNZ  R0 AGAIN
        SJMP  START.
```

* Write a program to perform multiplication of two 8 bit using 8085.

Program :-

```
MVI  D 00    Initialize reg D to 00
MVI  A 00    Initialize Accumulator content
LXI  H 4150  Load the content in H-L
MOV  B M     Get the First nm in reg B
MOV  C M     Get second nm in reg C
ADD  B       Add content of A to B.
JNC  Next    Jump on no carry to next
INR  D       Increment register D
DCR  C       Decrement register C
JNZ  LOOP
STA  4152    Store the result.
MOV  A D
STA  4153
HLT.         Program Ends.
```

* Write a program to Find the largest nm in an array of data using 8085.

Program

```
LXI  H 4200  Set pointer for array
MOV  B M     Load the count
INX  H
MOV  A M
```

```
        DCR  B        Decrement the count.
LOOP:   INX  H
        CMP  M        IF A>M goto Ahead
        JNC  AHEAD
        MOV  A M

AHEAD   DCR  B
        JNZ  LOOP
        STA  4300     Store the result
        HLT.
```

* W.A.P to Arrange an array of data in Ascending order.

Program:

```
        LXI  H 4200
        MOV  C M
        DCR  C
REPEAT: MOV  D C
        LXI  H 42014
LOOP:   MOV  A M
        INX  H
        CMP  M
        JC   SKIP
        MOV  B M
        MOV  M A
        DCX  H
        MOV  M B
        INX  H
SKIP    DCR  D
```

```
JNZ  LOOP
DCR  C
JNZ  REPEAT
HLT
```

W.A.P to convert two BCD numbers in memory to the equivalent HEX number using 8085.

Program:

```
LXI  H 4150
MOV  A M      Initialize memory
ADD  A        MSD X 2
MOV  B A      Store MSD X 2
ADD  A        MSD X4
ADD  A        MSD X8
ADD  B        MSD X10
INX  H        point to LSD
ADD  M        ADD to Form Hex
INX  H
MOV  M A      Store result
HLT.
```

* W.A.P to Find square of a number using Look up table

```
LXI  H 4125.  Initialize Lookup table address
LDA  4150     Get Data
CPI  0A
JC   AFTER.   If yes error
MVI  A FF
STA  4151
HLT
```

```
AFTER   MOV C A
        MVI B 00
        DAD B
        MOV A M
        STA 4151
        HLT.
```

* W.A.P to convert given ASCII character into its equivalent Hexa Decimal number using 8085.

Program:

```
        LDA 4500
        SUI 30
        CPI 0A
        JC SKIP
        SUI 07
SKIP    STA 4511
        HLT
```

* W.A.P to design a microcontroller based system for simple application like security system combination lock.

Program :-

```
        MOV   51H #
        MOV   52H #
        MOV   53H #
        MOV   54H #
        MOV   R1 #51
        MOV   R0 # 50
        MOV   R3 # 04
        MOV   R2 # 08
        MOV   DPTR #FFC2
        MOV   A #00
        MOV X @ DPTR A
        MOV   A #CC
        MOV X @ DPTR A
        MOV   A#90
        MOV X @ DPTR A
        MOV   A #FF
        MOV   DPTR #FFC0
LOOP    MOV X @ DPTR A
        DJNZ  R2 LOOP
AGAIN   MOV   DPTR # FFC2
WAIT    MOVX  A @ DPTR
        ANL   A#07
        JZ    WAIT
```

```
        Mov     A#40
        MovX    @DPTR A
        Mov     DPTR #FFCO
        MovX    A@ DPTR.
        Mov     @R0, A
        Mov     A @R1
        CJNE    A, 50H NEQ.
        INC     R1
        DJNZ    R3 AGAIN
        Mov     DPTR #FFCO
        Mov     A # 0C
        Mov X   @ DPTR, A
        SJMP    XX
NEQ     Mov     DPTR # FFCO
        Mov     A # 68
        Mov X   @ DPTR. A
        SJMP    YY.
```

Small type Question bank of mpmc carries 5 mks each.

1) What are the technical features of 8085?

2) Explain the Function of ALU Section of 8085?

3) Describe the Function of Following blocks of 8085.

1) ALU (ii) Timing & control unit

(iii) Instruction Decoder.

* Explain the Function of various register of 8085.

* Draw the block of 8085 & explain IR, stack pointer and program counter.

* What are the various Flags of 8085?

* What are the pointers of 8085. Explain the Function of pointers of 8085.

* Explain the Function of Interrupts section of 8085.

* List maskable and Non maskable Interrupts of 8085.

* Explain the Function of SID and SOD of 8085.

* Describe microprocessor evolution with suitable examples

* Differentiate any six between 8085 and 8086.

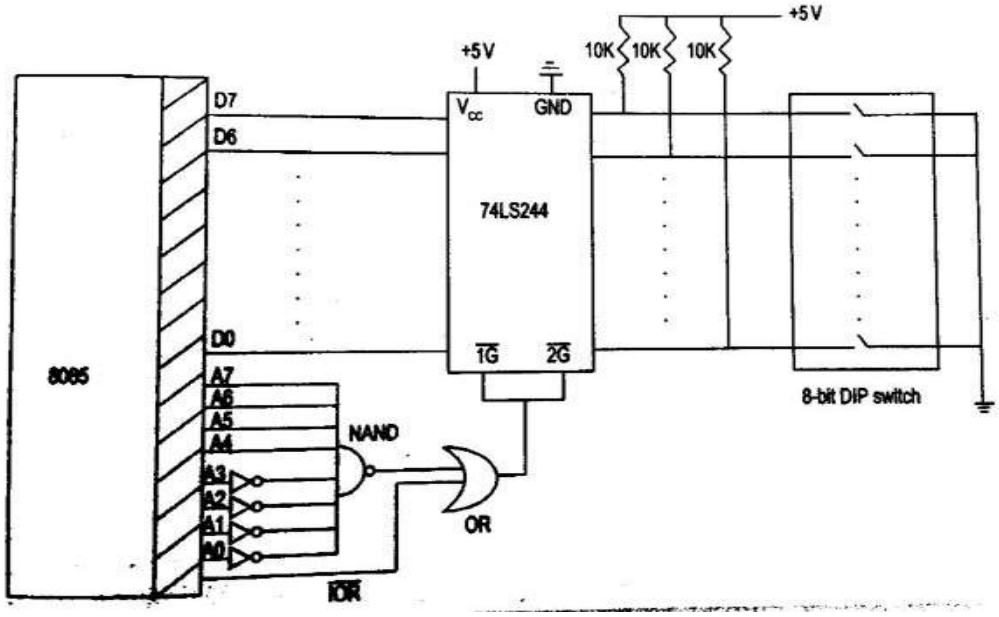



Fig. 21 interfacing of 8-bit DIP switch with 8085

## 4. **MEMORY MAPPED I/O INTERFACING**

In memory-mapped I/O, each input or output device is treated as if it is a memory location. The$\overline{\text{MEMR}}$ and $\overline{\text{MEMW}}$ control signals are used to activate the devices. Each input or output device is identified by unique 16-bit address, similar to 16-bit address assigned to memory location. All memory related instruction like LDA 2000H, LDAX B, MOV A, M can be used. Since the I/O devices use some of the memory address space of 8085, the maximum memory capacity is lesser than 64 KB in this method.

Ex: Interface an 8-bit DIP switch with the 8085 using logic gates such that the address assigned to it is F0F0H.

Since a 16-bit address has to be assigned to a DIP switch, the memory-mapped I/O technique must be used. Using LDA F0F0H instruction, the data from the 8-bit DIP switch can be transferred to the accumulator. The steps involved are:

i. The address F0F0H is placed in the address bus A0 – A15.

ii. The$\overline{\text{MEMR}}$ signal is made low for some time. iii. The data in the data bus is read and stored in the accumulator.

Fig. 22 shows the interfacing diagram.

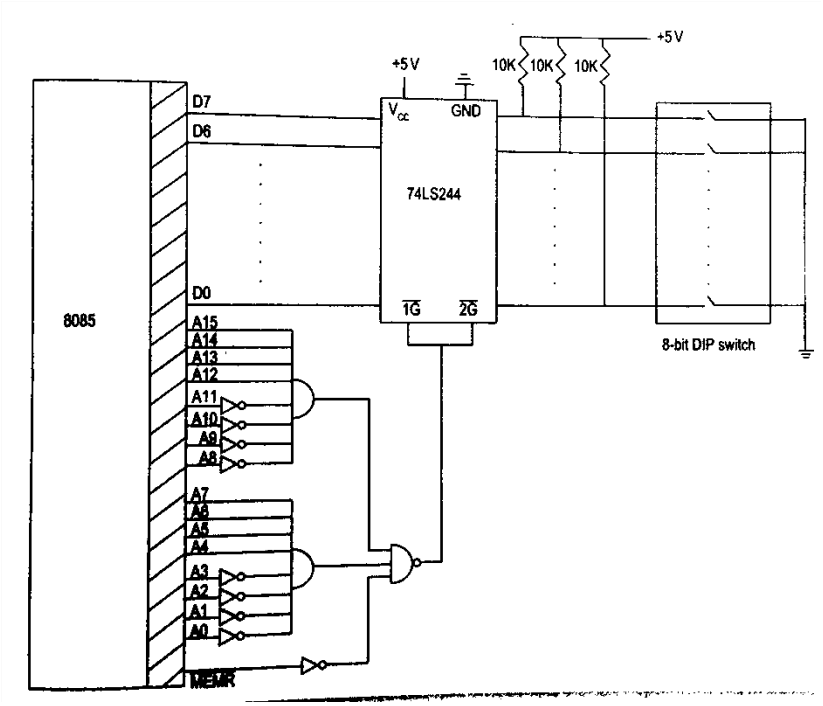



Fig. 22 Interfacing 8-bit DIP switch with 8085

When 8085 executes the instruction LDA F0F0H, it places the address F0F0H in the address lines A0 – A15 as:

| A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 | A0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | = F0F0H |

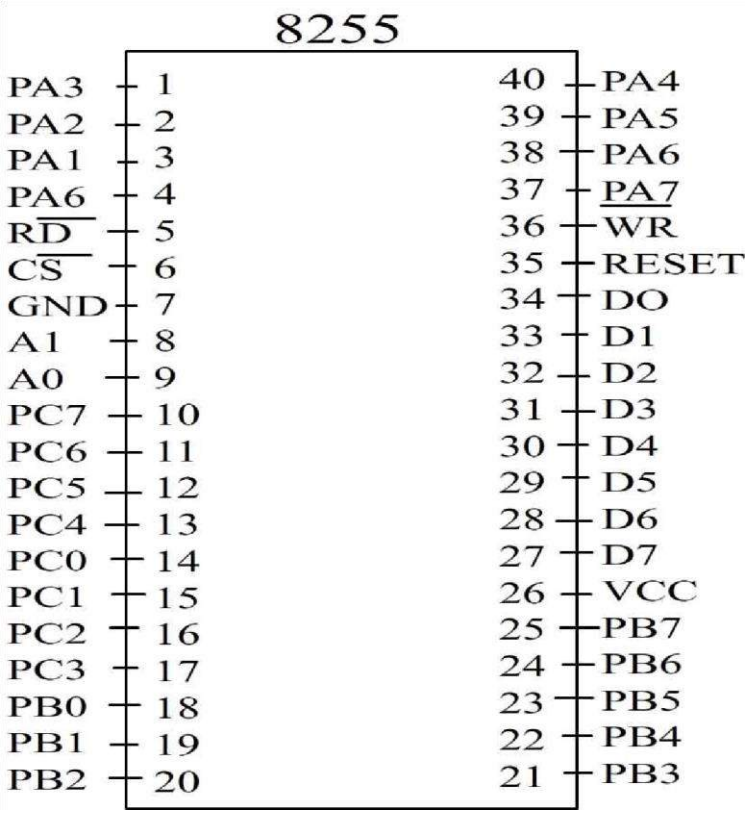
The address lines are connected to AND gates. The output of these gates along with $\overline{MEMR}$ signal are connected to a NAND gate, so that when the address F0F0H is placed in the address bus and $\overline{MEMR}$ = 0 its output becomes 0, thereby enabling the buffer 74LS244. The data from the DIP switch is placed in the 8085 data bus. The 8085 reads the data from the data bus and stores it in the accumulator.

## INTEL 8255: (Programmable Peripheral Interface)

The 8255A is a general purpose programmable I/O device designed for use with Intel microprocessors. It consists of three 8-bit bidirectional I/O ports (24I/O lines) that can be configured to meet different system I/O needs. The three ports are PORT A, PORT B & PORT C. Port A contains one 8-bit output latch/buffer and one 8-bit input buffer. Port B is same as PORT A or PORT B. However, PORT C can be split into two parts PORT C lower ($PC_0$-$PC_3$) and PORT C upper ($PC_7$-$PC_4$) by the control word. The three ports are divided in two groups Group A

(PORT A and upper PORT C) Group B (PORT B and lower PORT C). The two groups can be programmed in three different modes. In the first mode (mode 0), each group may be programmed in either input mode or output mode (PORT A, PORT B, PORT C lower, PORT C upper). In mode 1, the second's mode, each group may be programmed to have 8-lines of input or output (PORT A or PORT B) of the remaining 4-lines (PORT C lower or PORT C upper) 3-lines are used for hand shaking and interrupt control signals. The third mode of operation (mode 2) is a bidirectional bus mode which uses 8-line (PORT A only for a bidirectional bus and five lines (PORT C upper 4 lines and borrowing one from other group) for handshaking.

The 8255 is contained in a 40-pin package, whose pin out is shown below:




PIN Names

RESET – Reset input

- Chip selected

- Read input

- Write input

A $A_1$ – Port Address PA –

$PA_0$ – PORT A

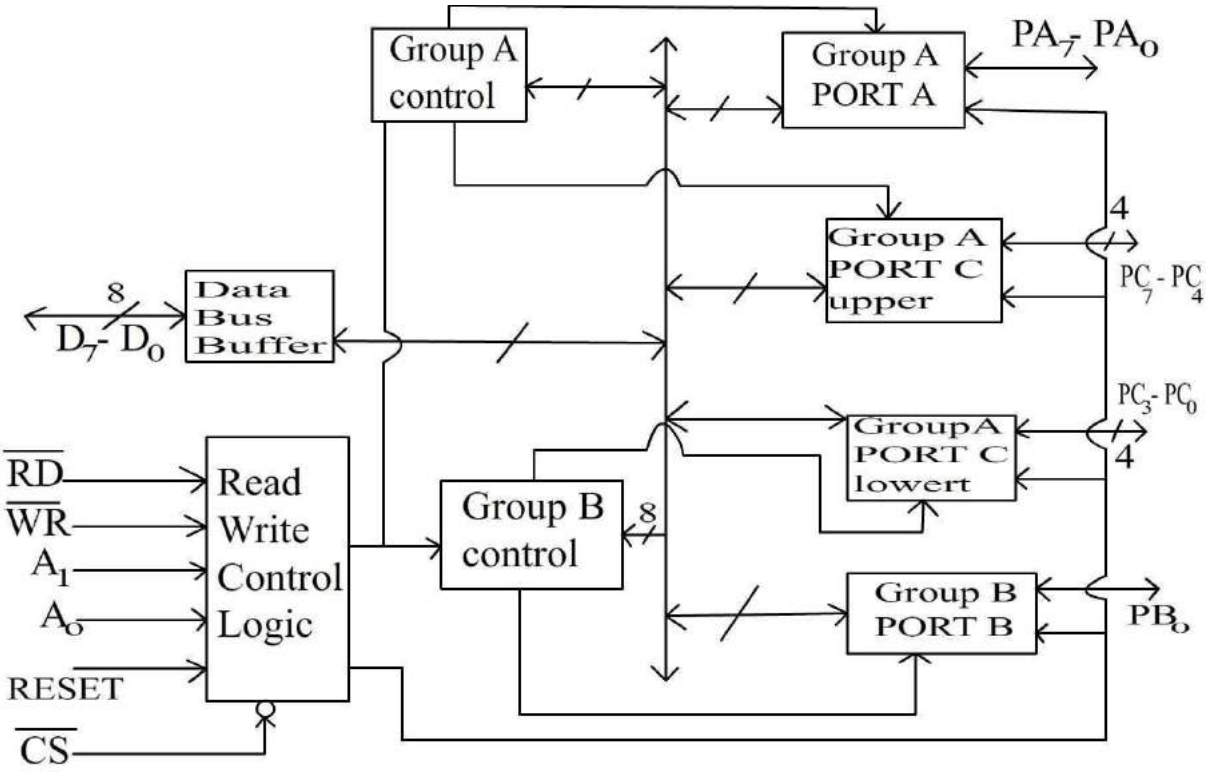
$PB_7$ – $PB_0$ – PORT B

PC – $PC_0$ – PORT C VCC

- +5v

GND - Ground

The block diagram is shown below:




Functional Description:

This support chip is a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. It is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer:

It is a tri-state 8-bit buffer used to interface the chip to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer. The data lines are connected to BDB of 8p.

Read/Write and logic control:

The function of this block is to control the internal operation of the device and to control the transfer of data and control or status words. It accepts inputs from the CPU address and control buses and in turn issues command to both the control groups.

$\overline{CS}$ Chip Select:

A low on this input selects the chip and enables the communication between the 8255 A & the CPU. It is connected to the output of address decode circuitry to select the device when it $\overline{RD}$ (Read). A low on this input enables the 8255 to send the data or status information to the CPU on the data bus.

$\overline{WR}$ (Write):

A low on this input pin enables the CPU to write data or control words into the 8255 A.

$A_1$, $A_0$ port select:

These input signals, in conjunction with the $\overline{RD}$ and $\overline{WR}$ inputs, control the selection of one of the three ports or the control word registers. They are normally connected to the least significant bits of the address bus ($A_0$ and $A_1$).

Following Table gives the basic operation,

| $A_1$ | $A_0$ | $\overline{RD}$ | $\overline{WR}$ | $\overline{CS}$ | Input operation |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | PORT A ⟶ Data bus |
| 0 | 1 | 0 | 1 | 0 | PORT B ⟶ Data bus |
| 1 | 0 | 0 | 1 | 0 | PORT C ⟶ Data bus |
| 0 | 0 | 1 | 0 | 0 | Output operation<br>Data bus ⟶ PORT A |
| 0 | 1 | 1 | 0 | 0 | Data bus ⟶ PORT B |
| 1 | 0 | 1 | 0 | 0 | Data bus ⟶ PORT C |
| 1 | 1 | 1 | 0 | 0 | Data bus ⟶ control |

All other states put data bus into tri-state/illegal condition.

**RESET:**

A high on this input pin clears the control register and all ports (A, B & C) are initialized to input mode. This is connected to RESET OUT of 8255. This is done to prevent destruction of circuitry connected to port lines. If port lines are initialized as output after a power up or reset, the port might try to output into the output of a device connected to same inputs might destroy one or both of them.

**PORTs A, B and C:**

The 8255A contains three 8-bit ports (A, B and C). All can be configured in a variety of functional characteristic by the system software.

**PORTA:**

One 8-bit data output latch/buffer and one 8-bit data input latch.

**PORT B:**

One 8-bit data output latch/buffer and one 8-bit data input buffer.

**PORT C:**

One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signals inputs in conjunction with ports A and B.

**Group A & Group B control:**

The functional configuration of each port is programmed by the system software. The control words outputted by the CPU configure the associated ports of the each of the two groups. Each control block accepts command from Read/Write content logic receives control words from the internal data bus and issues proper commands to its associated ports.

Control Group A – Port A & Port C upper

Control Group B – Port B & Port C lower

The control word register can only be written into No read operation if the control word register is allowed.

**Operation Description:**

**Mode selection:**

There are three basic modes of operation that can be selected by the system software. Mode 0: Basic Input/output

Mode 1: Strobes Input/output Mode

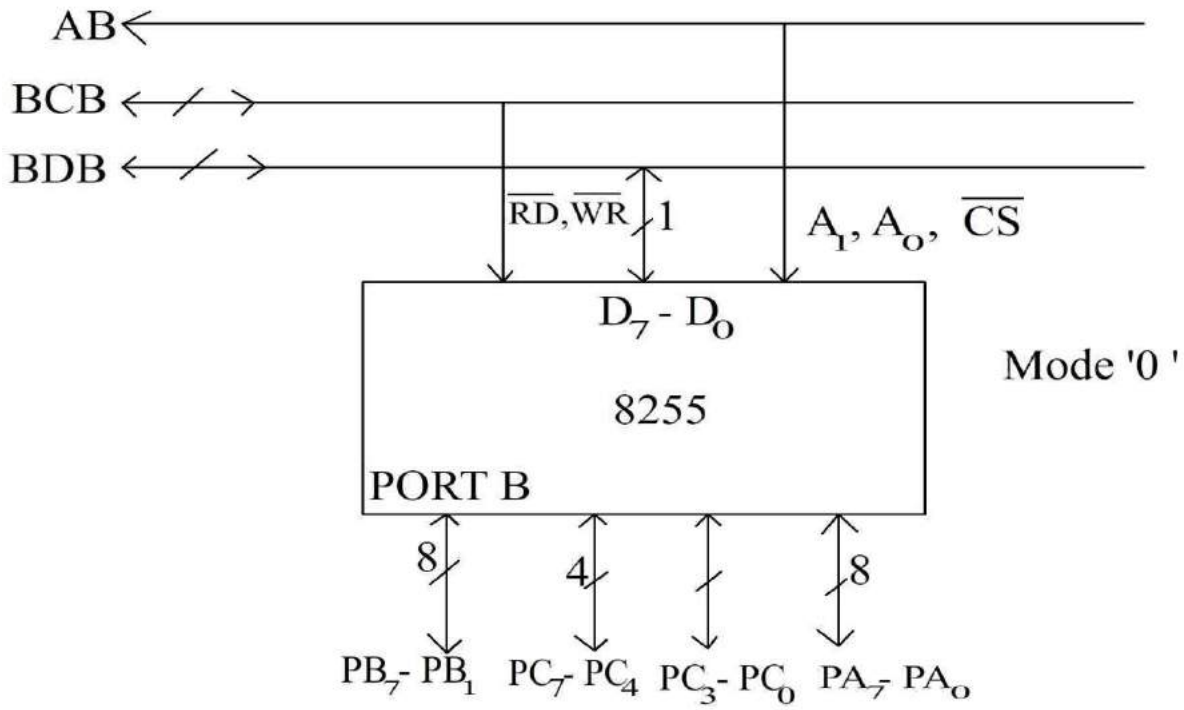
2: Bi-direction bus.

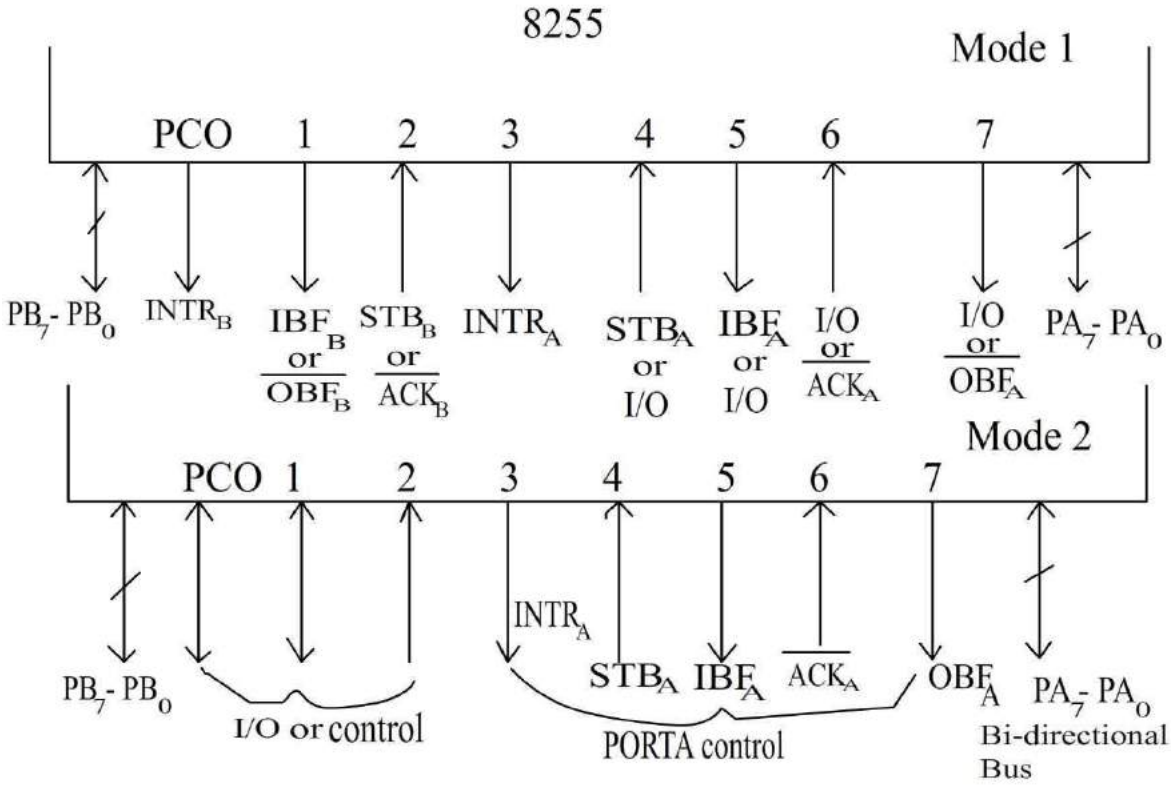
When the reset input goes HIGH all poets are set to mode'0' as input which means all 24 lines are in high impedance state and can be used as normal input. After the reset is removed the 8255A remains in the

input mode with no additional initialization. During the execution of the program any of the other modes may be selected using a single output instruction.

The modes for PORT A & PORT B can be separately defined, while PORT C is divided into two portions as required by the PORT A and PORT B definitions. The ports are thus divided into two groups Group A & Group B. All the output register, including the status flip-flop will be reset whenever the mode is changed. Modes of the two group may be combined for any desired I/O operation e.g. Group A in mode ‘1’ and group B in mode ‘0’.

The basic mode definitions with bus interface and the mode definition format are given in fig (a) & (b),






## INTEL 8259A Programmable Interrupt Controller

The 8259A is a programmable interrupt controller designed to work with Intel microprocessor 8080 A, 8085, 8086, 8088. The 8259 A interrupt controller can

1) Handle eight interrupt inputs. This is equivalent to providing eight interrupt pins on the processor in place of one INTR/INT pin.
2) Vector an interrupt request anywhere in the memory map. However, all the eight interrupt are spaced at the interval of either four or eight location. This eliminates the major drawback, 8085 interrupt, in which all interrupts are vectored to memory location on page $00_H$.
3) Resolve eight levels of interrupt priorities in a variety of modes.
4) Mask each interrupt request individually.

The minimum mode is selected by applying logic 1 to the MN / MX# input pin. This is a single microprocessor configuration. The maximum mode is selected by applying logic 0 to the MN / MX# input pin. This is a multi micro processors configuration.

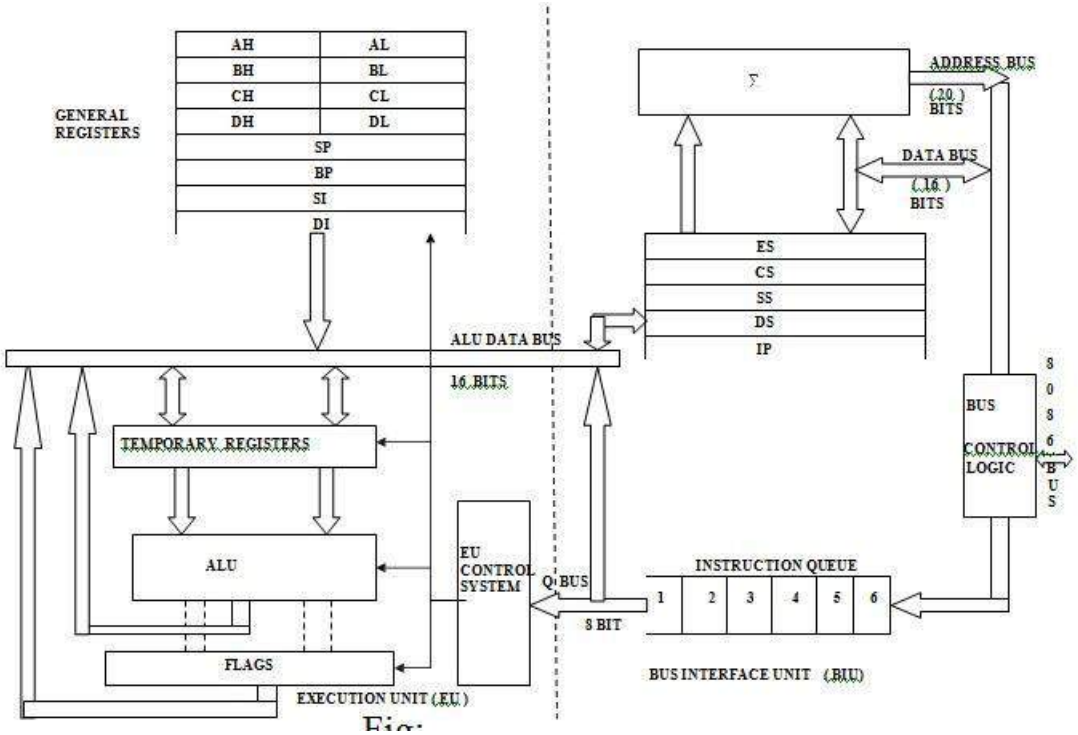
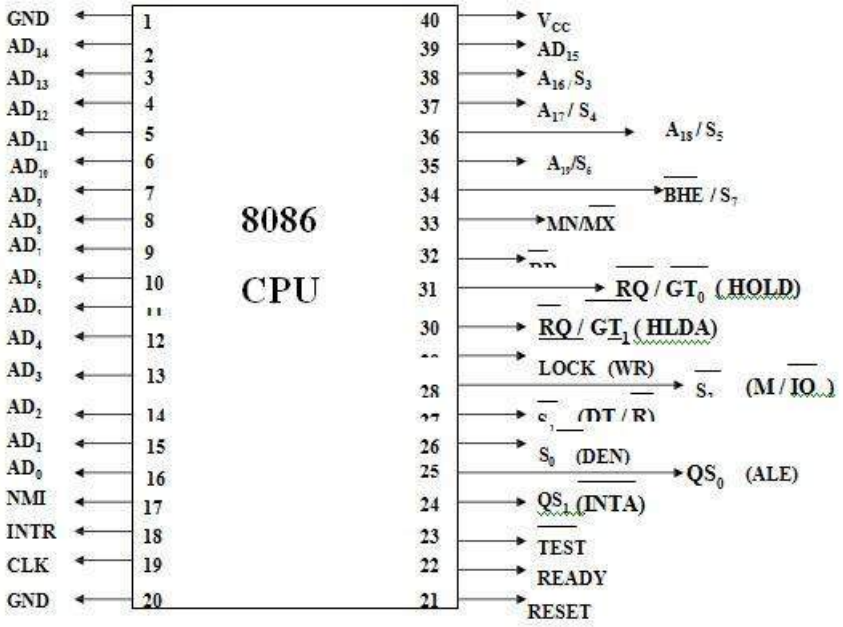



Fig. Architecture of 8086

# Pin Diagram of 8086




Internal Architecture of 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. BIU contains Instruction queue, Segment registers, Instruction pointer, Address adder. EU contains Control circuitry, Instruction decoder, ALU, Pointer and Index register, Flag register.

Bus Interfacr Unit**:**

It provides a full 16 bit bidirectional data bus and 20 bit address bus. The bus interface unit is responsible for performing all external bus operations.

***Specifically it has the following functions***:

Instruction fetch, Instruction queuing, Operand fetch and storage, Address relocation and Bus control. The BIU uses a mechanism known as an instruction stream queue to implement a ***pipeline architecture.***

This queue permits prefetch of up to six bytes of instruction code. When ever the queue of the BIU is not full, it has room for at least two more bytes and at the same time the EU is not requesting it to read or write operands from memory, the BIU is free to look ahead in the program by prefetching the next sequential instruction. These prefetching instructions are held in its FIFO queue. With its 16 bit data bus, the BIU fetches two instruction bytes in a single memory cycle. After a byte is loaded at the input end of the queue, it automatically shifts up through the FIFO to the empty location nearest the output.

The EU accesses the queue from the output end. It reads one instruction byte after the other from the output of the queue. If the queue is full and the EU is not requesting access to operand in memory. These intervals of no bus activity, which may occur between bus cycles are known as ***Idle state***. If the BIU is already in the process of fetching an instruction when the EU request it to read or write operands from memory or I/O, the BIU first completes the instruction fetch bus cycle before initiating the operand read / write cycle. The BIU also contains a dedicated adder which is used to generate the 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. For example, the physical address of the next instruction to be fetched is formed by combining the current contents of the code segment CS register and the current contents of the instruction pointer IP register. The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write.

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

| COMMON SIGNALS | | |
|---|---|---|
| **Name** | **Function** | **Type** |
| $AD_{15}$– $AD_0$ | Address/ Data Bus | Bidirectional 3-state |
| $A_{19}/S_6$– $A_{16}/S_3$ | Address / Status | Output 3-State |
| $\overline{BHE}$ /S 7 | Bus High Enable / Status | Output 3- State |
| MN / $\overline{MX}$ | Minimum / Maximum Mode Control | Input |
| $\overline{RD}$ | Read Control | Output 3- State |
| TEST | Wait On Test Control | Input |
| READY | Wait State Controls | Input |
| RESET | System Reset | Input |
| NMI | Interrupt Request | Input |
| INTR | Interrupt Request | Input |
| CLK | System Clock | Input |
| Vcc | + 5 V | Input |
| GND | Ground | |

| Minimum Mode Signals ($\overline{MN}$/ MX = Vcc) | | |
|---|---|---|
| **Name** | **Function** | **Type** |
| HOLD | Hold Request | Input |
| HLDA | Hold Acknowledge | Output |
| $\overline{WR}$ | Write Control | Output 3- state |
| $\overline{M/IO}$ | Memory or IO Control | Output 3-State |
| $\overline{DT/R}$ | Data Transmit / Receiver | Output 3-State |
| $\overline{DEN}$ | Date Enable | Output 3-State |
| ALE | Address Latch Enable | Output |
| $\overline{INTA}$ | Interrupt Acknowledge | Output |

| Maximum mode signals ( MN / $\overline{MX}$ = GND ) | | |
|---|---|---|
| **Name** | **Function** | **Type** |
| RQ / $\overline{GT1, 0}$ | Request / Grant Bus Access Control | Bidirectional |
| $\overline{LOCK}$ | Bus Priority Lock Control | Output, 3- State |
| $\overline{S_2} - \overline{S_0}$ | Bus Cycle Status | Output, 3- State |
| QS1, QS0 | Instruction Queue Status | Output |

Internal Registers of 8086

The 8086 has four groups of the user accessible internal registers. They are the instruction pointer, four data registers, four pointer and index register, four segment registers.

The 8086 has a total of fourteen 16-bit registers including a 16 bit register called the ***status register***, with 9 of bits implemented for status and control flags. Most of the registers contain data/instruction offsets within 64 KB memory segment. There are four different 64 KB segments for instructions, stack, data and extra data. To specify where in 1 MB of processor memory these 4 segments are located the processor uses four segment registers:

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

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

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

**Extra segment** (ES) is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. ES register can be changed directly using POP and LES instructions. It is possible to change default segments used by general and index registers by prefixing instructions with a CS, SS, DS or ES prefix.

All general registers of the 8086 microprocessor can be used for arithmetic and logic operations. The general registers are:

**Accumulator** register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16-bit register AX. AL in this case contains the low-order byte of the

word, and AH contains the high-order byte. Accumulator can be used for I/O operations and string manipulation.

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

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

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

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 indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation instructions.

Other registers:

**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** (AF) - 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-order 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 calculation.

## Addressing Modes

**Implied** - the data value/data address is implicitly associated with the instruction.

**Register** - references the data in a register or in a register pair.

**Immediate** - the data is provided in the instruction.

**Direct** - the instruction operand specifies the memory address where data is located.

**Register indirect** - instruction specifies a register containing an address, where data is located. This addressing mode works with SI, DI, BX and BP registers.

**Based** :- 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP), the resulting value is a pointer to location where data resides.

**Indexed** :- 8-bit or 16-bit instruction operand is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.

**Based Indexed** :- the contents of a base register (BX or BP) is added to the contents of an index register (SI or DI), the resulting value is a pointer to location where data resides.
**Based Indexed with displacement** :- 8-bit or 16-bit instruction operand is added to the contents of a base register (BX or BP) and index register (SI or DI), the resulting value is a pointer to location where data resides.

## Interrupts

The processor has the following interrupts:

**INTR** is a maskable hardware interrupt. The interrupt can be enabled/disabled using STI/CLI instructions or using more complicated method of updating the FLAGS register with the help of the POPF instruction.

When an interrupt occurs, the processor stores FLAGS register into stack, disables further interrupts, fetches from the bus one byte representing interrupt type, and jumps to interrupt processing routine address of which is stored in location 4 * <interrupt type>. Interrupt processing routine should return with the IRET instruction.

**NMI** is a non-maskable interrupt. Interrupt is processed in the same way as the INTR interrupt. Interrupt type of the NMI is 2, i.e. the address of the NMI processing routine is stored in location 0008h. This interrupt has higher priority then the maskable interrupt.

**Software interrupts** can be caused by:

INT instruction - breakpoint interrupt. This is a type 3 interrupt.

INT <interrupt number> instruction - any one interrupt from available 256 interrupts.

INTO instruction - interrupt on overflow

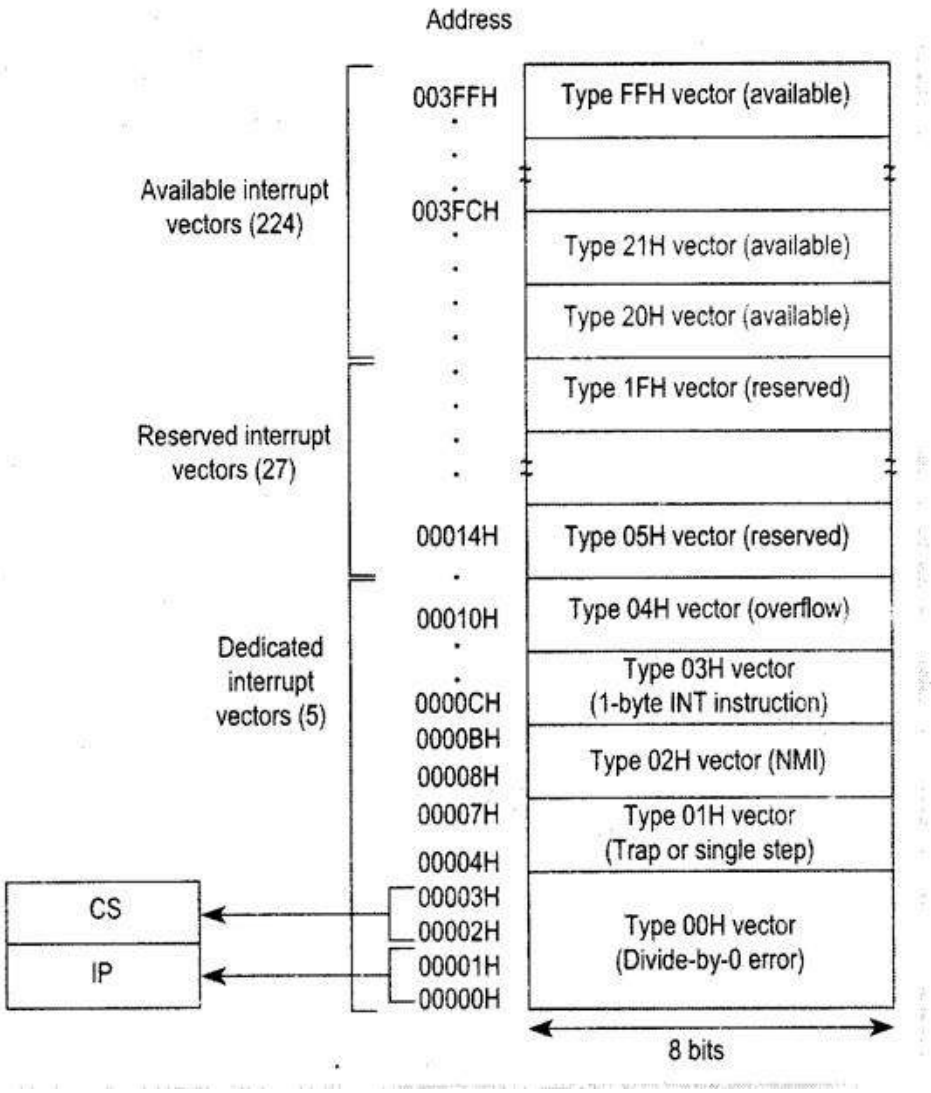
Single-step interrupt - generated if the TF flag is set. This is a type 1 interrupt. When the CPU processes this interrupt it clears TF flag before calling the interrupt processing routine.

**Processor exceptions**: Divide Error (Type 0), Unused Opcode

(type 6) and Escape opcode (type 7).

Software interrupt processing is the same as for the hardware interrupts.

The figure below shows the 256 interrupt vectors arranged in the interrupt vector table in the memory.



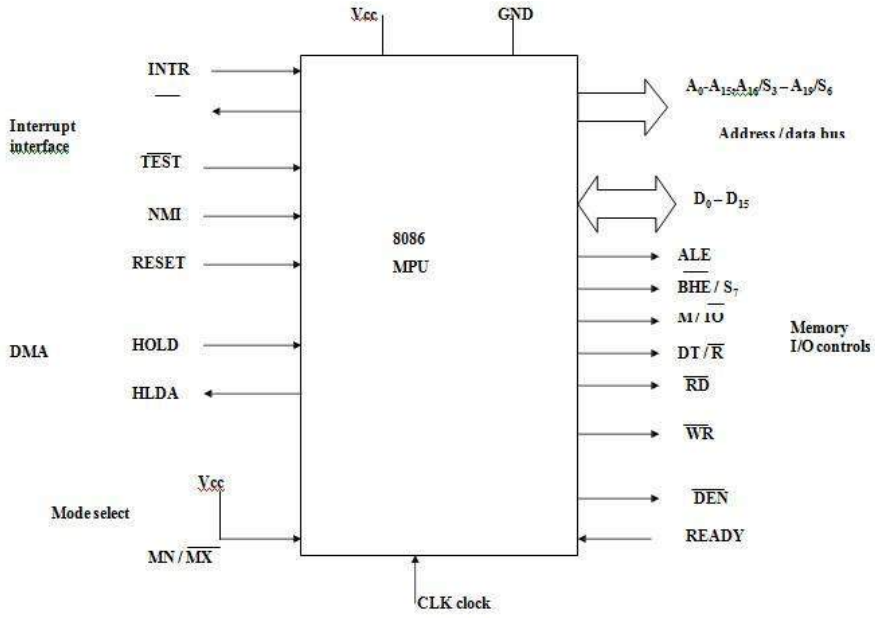

Interrupt Vector Table in the 8086

**Minimum Mode Interface**

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

**Address/Data Bus** : these lines serve two functions. As an address bus is 20 bits long and consists of signal lines A0 through A19. A19 represents the MSB and A0 LSB. A 20bit address gives the 8086 a 1Mbyte memory address space. More over it has an independent I/O address space which is 64K bytes in length.

The 16 data bus lines D0 through D15 are actually multiplexed with address lines A0 through A15 respectively. By multiplexed we mean that the bus work as an address bus during first machine cycle and as a data bus during next machine cycles. D15 is the MSB and D0 LSB. When acting as a data bus, they carry read/write data for memory, input/output data for I/O devices, and interrupt type codes from an interrupt controller.




Block Diagram of the Minimum Mode 8086 MPU

**Status signal** : The four most significant address lines A19 through A16 are also multiplexed but in this case with status signals S6 through S3. These status bits are output on the bus at the same time that data are transferred over the other bus lines. Bit S4 and S3 together from a 2 bit binary code that identifies which of the 8086 internal segment registers are used to generate the physical address that was output on the address bus during the current bus cycle. Code S4S3 = 00 identifies a register known as ***extra segment register*** as the source of the segment address.

| $S_4$ | $S_3$ | Segment Register |
|---|---|---|
| 0 | 0 | Extra |
| 0 | 1 | Stack |
| 1 | 0 | Code / none |
| 1 | 1 | Data |

Memory segment status code

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

**Control Signals** : The control signals are provided to support the 8086 memory I/O interfaces. They control functions such as when the bus is to carry a valid address in which direction data are to be transferred over the bus, when valid write data are on the bus and when to put read data on the system bus.

ALE is a pulse to logic 1 that signals external circuitry when a valid address word is on the bus. This address must be latched in external circuitry on the 1-to-0 edge of the pulse at ALE.

Another control signal that is produced during the bus cycle is BHE bank high enable. Logic 0 on this used as a memory enable signal for the most significant byte half of the data bus D8 through D1. These lines also serves a second function, which is as the S7 status line.

Using the M/IO and DT/R__lines, the 8086 signals which type of bus cycle is in progress and in which direction data are to be transferred over the bus.

The logic level of M/IO tells external circuitry whether a memory or I/O transfer is taking place over the bus. Logic 1 at this output signals a memory operation and logic 0 an

I/O operation.

The direction of data transfer over the bus is 74ignallin by the logic level output at DT/R. When this line is logic 1 during the data transfer part of a bus cycle, the bus is in the transmit mode. Therefore, data are either written into memory or output to an I/O device.

On the other hand, logic 0 at DT/R signals that the bus is in the receive mode. This corresponds to reading data from memory or input of data from an input port.

The signal read RD and write WR indicates that a read bus cycle or a write bus cycle is in progress. The 8086 switches WR to logic 0 to signal external device that valid write or output data are on the bus.

On the other hand, RD indicates that the 8086 is performing a read of data of the bus. During read operations, one other control signal is also supplied. This is DEN (data enable) and it signals external devices when they should put data on the bus.

There is one other control signal that is involved with the memory and I/O interface. This is the READY signal.

READY signal is used to insert wait states into the bus cycle such that it is extended by a number of clock periods. This signal is provided by an external clock generator device and can be supplied by the memory or I/O sub- system to signal the 8086 when they are ready to permit the data transfer to be completed.

**Maximum Mode Interface**

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

**8288 Bus Controller – Bus Command and Control Signals**: 8086 does not directly provide all the signals that are required to control the memory, I/O and interrupt interfaces. Specially the WR, M/IO, DT/R, DEN, ALE and INTA, signals are no longer produced by the 8086. Instead it outputs three status signals S0, S1, S2 prior to the initiation of each bus cycle. This 3- bit bus status code identifies which type of bus cycle is to follow. S2S1S0 are input to the external bus controller device, the bus controller generates the appropriately timed command and control signals.

## Maximum Mode Interface (cont..)

| Status Inputs | | | CPU Cycles | 8288 Command |
|---|---|---|---|---|
| $\overline{S_2}$ | $\overline{S_1}$ | $\overline{S_0}$ | | |
| 0 | 0 | 0 | Interrupt Acknowledge | $\overline{INTA}$ |
| 0 | 0 | 1 | Read I/O Port | $\overline{IORC}$ |
| 0 | 1 | 0 | Write I/O Port | $\overline{IOWC}$, $\overline{AIOWC}$ |
| 0 | 1 | 1 | Halt | None |
| 1 | 0 | 0 | Instruction Fetch | $\overline{MRDC}$ |
| 1 | 0 | 1 | Read Memory | $\overline{MRDC}$ |
| 1 | 1 | 0 | Write Memory | $\overline{MWTC}$, $\overline{AMWC}$ |
| 1 | 1 | 1 | Passive | None |

**Bus Status Codes**

The 8288 produces one or two of these eight command signals for each b_us cycles. For instance, when the 8086 outputs the code S2S1S0 equals 001, it indicates that an ***I/O read cycle*** is to be performed. In the code 111 is output by the 8086, it is 75ignalling that no bus activity is to take place.

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

**8289 Bus Arbiter – Bus Arbitration and Lock Signals:**

This device permits processors to reside on the system bus. It does this by implementing the Multibus arbitration protocol in an 8086-based system. Addition of the 8288 bus controller and 8289 bus arbiter frees a number of the 8086 pins for use to produce control signals that are needed to support multiple processors. Bus priority lock (LOCK) is one of these signals. It is input to the bus arbiter together with status signals S0 through S2.

**Queue Status Signals:** Two new signals that are produced by the 8086 in the maximummode system are queue status outputs QS0 and QS1. Together they form a 2-bit queue status code, QS1QS0. Following table shows the four different queue status.

| $QS_1$ | $QS_0$ | Queue Status |
|---|---|---|
| 0 (low) | 0 | No Operation. During the last clock cycle, nothing was taken from the queue. |
| 0 | 1 | First Byte. The byte taken from the queue was the first byte of the instruction. |
| 1 (high) | 0 | Queue Empty. The queue has been reinitialized as a result of the execution of a transfer instruction. |
| 1 | 1 | Subsequent Byte. The byte taken from the queue was a subsequent byte of the instruction. |

Queue status codes

QUESTIONS:

1. What is the size of address and data bus in the 8086?
2. Draw the register organization of the 8086 and explain typical applications of each register.
3. How is the 20-bit physical memory address calculated in the 8086 processor?
4. Write the different memory segments used in the 8086 and their functions.
5. Write the function of the DF, IF and TF bits in the 8086.
6. The content of the different registers in the 8086 is CS = F000H, DS = 1000H, SS = 2000H and ES = 3000H. Find the base address of the different segments in the memory.
7. What is the difference between the minimum and maximum mode operation of the 8086?
8. What is meant by DMA operation? Which pins of the 8086 are used to perform the DMA operation in the minimum and maximum modes of the 8086?
9. Explain the architecture of the 8086 with a neat functional block diagram.
10. Explain the function of different flags in the 8086.
11. What is the function of segment override prefix? Give two examples.
12. What is the difference between inter-segment and intra-segment jump in the 8086?
13. What is the difference between short and near jump in the 8086?
14. What are the different uses of stack in a microprocessor?
15. What is the difference between the MUL and IMUL instructions in the 8086?

16. What is the difference between the DIV and IDIV instructions in the 8086?
17. What is the function of DAA instruction in the 8086?
18. Write the operations performed when the instruction AAD is executed in the 8086.
19. What is the difference between maskable and non-maskable interrupts?
20. What is the difference between hardware and software interrupts?
21. What is an interrupt vector? What is the maximum number of interrupt vectors that can be stored in the IVT of the 8086?
22. Write a program to move a word string 200 bytes (i.e. 100 words) long from the offset address 1000H to the offset address 3000H in the segment 5000H.
23. Write a program to find the smallest word in an array of 100 words stored sequentially in the memory; starting at the offset address 1000H in the segment address 5000H. Store the result at the offset address 2000H in the same segment.
24. Write a program to add the two BCD data 29H and 98H and store the result in BCD form in the memory locations 2000H: 3000H and 2000H: 3001H.

**MODULE: 4**

## 8051 microcontroller

**What is a Microcontroller?**

A Microcontroller is a programmable digital processor with necessary peripherals. Both microcontrollers and microprocessors are complex sequential digital circuits meant to carry out job according to the program / instructions. Sometimes analog input/output interface makes a part of microcontroller circuit of mixed mode(both analog and digital nature).

**Microcontrollers Vs Microprocessors**

1. A microprocessor requires an external memory for program/data storage. Instruction execution requires movement of data from the external memory to the microprocessor or vice versa. Usually, microprocessors have good computing power and they have higher clock speed to facilitate faster computation.
2. A microcontroller has required on-chip memory with associated peripherals. A microcontroller can be thought of a microprocessor with inbuilt peripherals.
3. A microcontroller does not require much additional interfacing ICs for operation and it functions as a stand alone system. The operation of a microcontroller is multipurpose, just like a Swiss knife.
4. Microcontrollers are also called embedded controllers. A microcontroller clock speed is limited only to a few tens of MHz. Microcontrollers are numerous and many of them are application specific.

**Development/Classification of microcontrollers (Invisible)**

Microcontrollers have gone through a silent evolution (invisible). The evolution can be rightly termed as silent as the impact or application of a microcontroller is not well known to a common user, although microcontroller technology has undergone significant

change since early 1970's. Development of some popular microcontrollers is given as follows.

| | | |
|---|---|---|
| Intel 4004 | 4 bit (2300 PMOS trans, 108 kHz) | 1971 |
| Intel 8048 | 8 bit | 1976 |
| Intel 8031 | 8 bit (ROM-less) | . |
| Intel 8051 | 8 bit (Mask ROM) | 1980 |
| Microchip PIC16C64 | 8 bit | 1985 |
| Motorola 68HC11 | 8 bit (on chip ADC) | . |
| Intel 80C196 | 16 bit | 1982 |
| Atmel AT89C51 | 8 bit (Flash memory) | . |
| Microchip PIC 16F877 | 8 bit (Flash memory + ADC) | . |

**Development of microprocessors (Visible)**

Microprocessors have undergone significant evolution over the past four decades. This development is clearly perceptible to a common user, especially, in terms of phenomenal growth in capabilities of personal computers. Development of some of the microprocessors can be given as follows.

| | | |
|---|---|---|
| Intel 4004 | 4 bit (2300 PMOS transistors) | 1971 |
| Intel 8080<br>8085 | 8 bit (NMOS)<br>8 bit | 1974 |
| Intel 8088<br>8086 | 16 bit<br>16 bit | 1978 |
| Intel 80186<br>80286 | 16 bit<br>16 bit | 1982 |
| Intel 80386 | 32 bit (275000 transistors) | 1985 |
| Intel 80486 SX<br>DX | 32 bit<br>32 bit (built in floating point unit) | 1989 |
| Intel 80586 I<br>MMX<br>Celeron<br>II<br>III<br>IV | 64 bit | <br>1993<br>1997<br>1999<br>2000 |
| Z-80 (Zilog) | 8 bit | 1976 |
| Motorola Power PC<br>601<br>602<br>603 | 32-bit | <br>1993<br>1995 |

We use more number of microcontrollers compared to microprocessors. Microprocessors are primarily used for computational purpose, whereas microcontrollers find wide application in devices needing real time processing / control.
Application of microcontrollers are numerous. Starting from domestic applications such as in washing machines, TVs, airconditioners, microcontrollers are used in automobiles, process control industries , cell phones, electrical drives, robotics and in space applications.

**Microcontroller Chips**

Broad Classification of different microcontroller chips could be as follows:

- Embedded (Self -Contained) 8 - bit Microcontroller
- 16 to 32 Microcontrollers
- Digital Signal Processors

**Features of Modern Microcontrollers**

- Built-in Monitor Program
- Built-in Program Memory
- Interrupts
- Analog I/O
- Serial I/O
- Facility to Interface External Memory
- Timers

**Internal Structure of a Microcontroller**

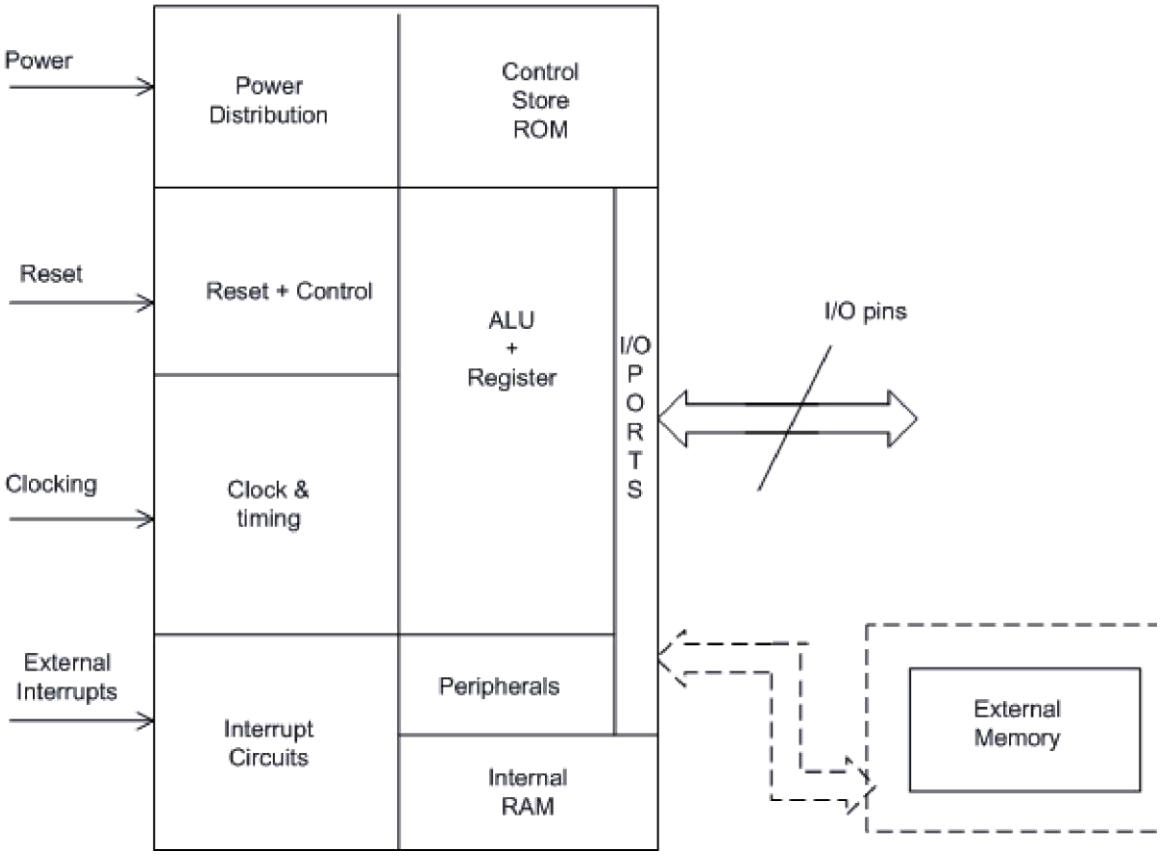



Fig. 4.1 Internal Structure of a Microcontroller

At times, a microcontroller can have external memory also (if there is no internal memory or extra memory interface is required). Early microcontrollers were manufactured using bipolar or NMOS technologies. Most modern microcontrollers are manufactured with CMOS technology, which leads to reduction in size and power loss. Current drawn by the IC is also reduced considerably from 10mA to a few micro Amperes in sleep mode(for a microcontroller running typically at a clock speed of 20MHz).

**Harvard Architecture (Separate Program and Data Memory interfaces)**

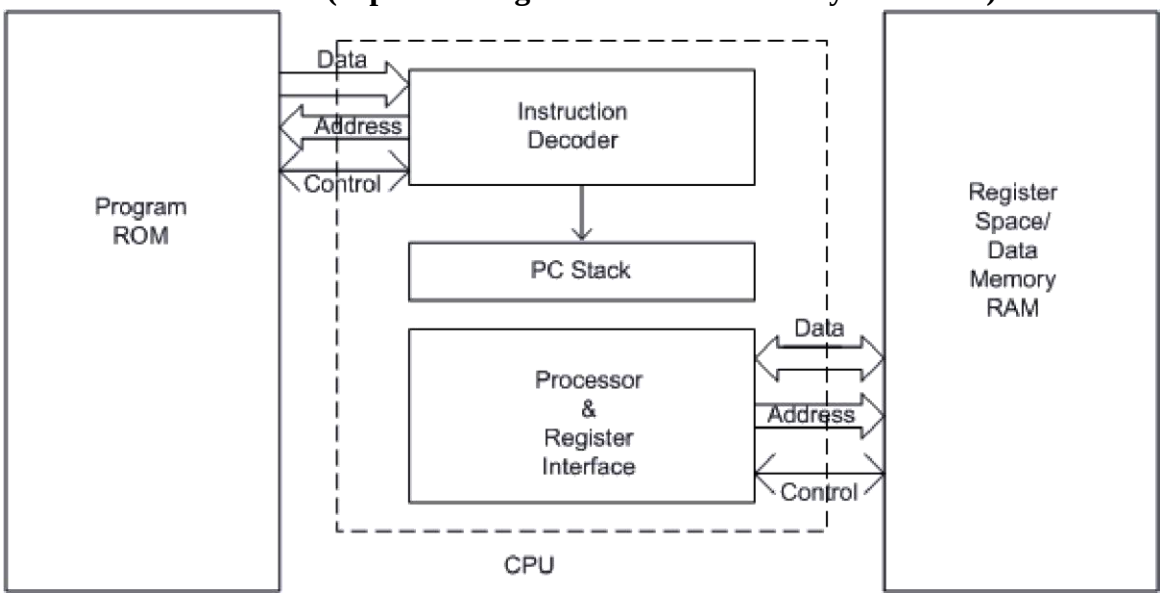



Fig. 4.2 Harvard Arcitecture

The same instruction (as shown under Princeton Architecture) would be executed as follows:

Cycle 1
- Complete previous instruction
- Read the "Move Data to Accumulator" instruction

Cycle 2
- Execute "Move Data to Accumulator" instruction
- Read next instruction

Hence each instruction is effectively executed in one instruction cycle, except for the ones that modify the content of the program counter. For example, the "jump" (or call) instructions takes 2 cycles. Thus, due to parallelism, Harvard architecture executes more instructions in a given time compared to Princeton Architecture.

**Memory organization**:

In the 8051, the memory is organized logically into program memory and data memory separately. The program memory is read-only type; the data memory is organized as readwrite memory. Again, both program and data memories can be within the chip or outside.

**Basic 8051 Architecture**

The 8051 is an 8-bit microcontroller i.e. the data bus within and outside the chip is eight bits wide. The address bus of the 8051 is 16-bit wide. So it can address 64 KB of memory. The 8051 is a 40-pin chip as shown is figure below:

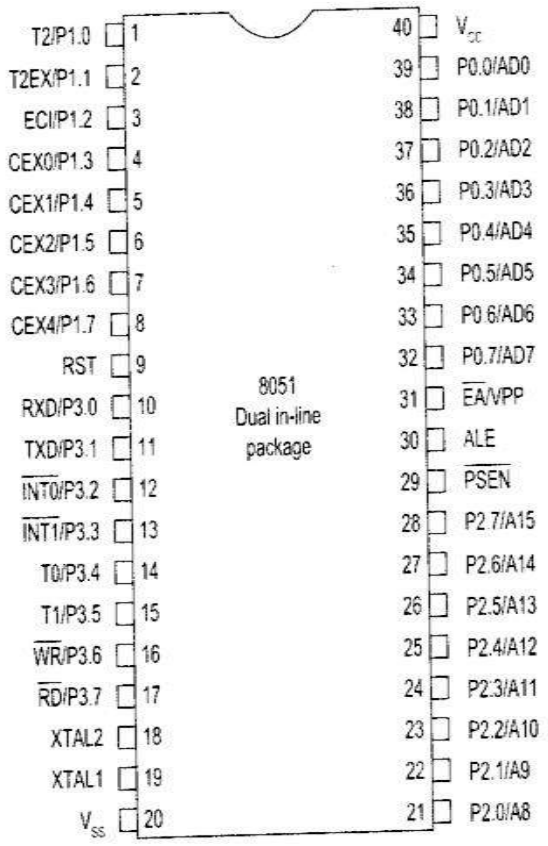



Fig 4.3 Pin details of 8051

8051 employs Harvard architecture. It has some peripherals such as 32 bit digital I/O, Timers and Serial I/O. The basic architecture of 8051 is given in fig 4.4.

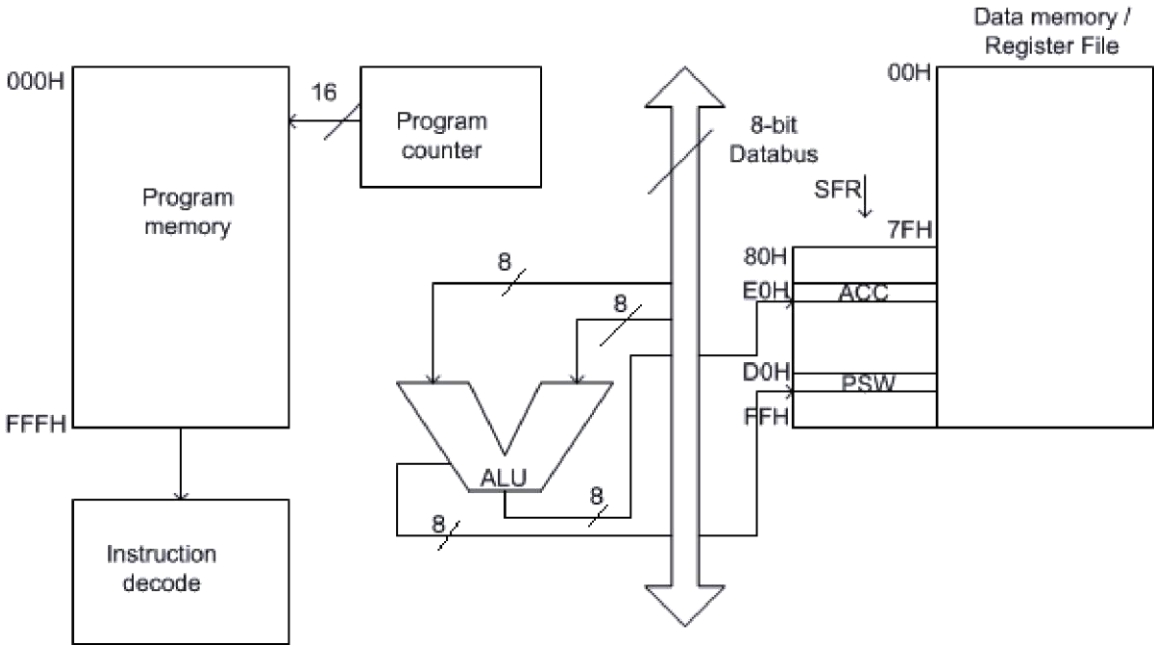



Fig 4.4 : Basic 8051 Architecture

Various features of 8051 microcontroller are given as follows.

- 8-bit CPU
- 16-bit Program Counter
- 8-bit Processor Status Word (PSW)
- 8-bit Stack Pointer
- Internal RAM of 128bytes
- Special Function Registers (SFRs) of 128 bytes
- 32 I/O pins arranged as four 8-bit ports (P0 - P3)
- Two 16-bit timer/counters : T0 and T1
- Two external and three internal vectored interrupts
- One full duplex serial I/O

**8051 Clock and Instruction Cycle**

In 8051, one instruction cycle consists of twelve (12) clock cycles. Instruction cycle is sometimes called as Machine cycle by some authors.

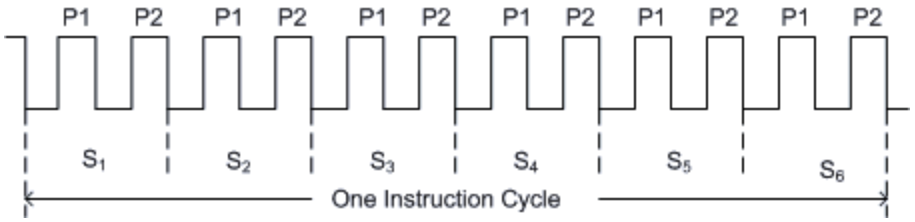



Fig 4.5 : Instruction cycle of 8051

In 8051, each instruction cycle has six states ($S_1$ - $S_6$). Each state has two pulses (P1 and P2)

**128 bytes of Internal RAM Structure (lower address space)**

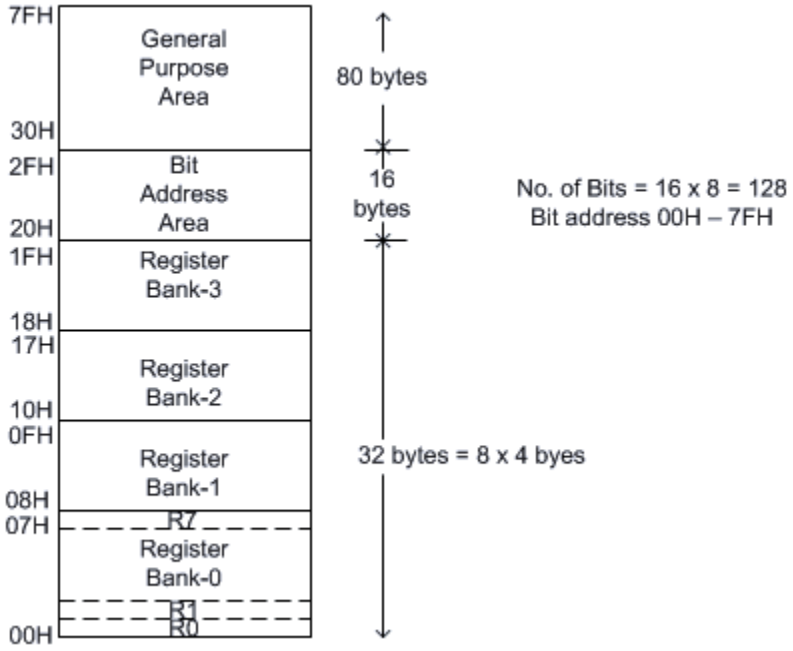



Fig 4.6: Internal RAM Structure

The lower 32 bytes are divided into 4 separate banks. Each register bank has 8 registers of one byte each. A register bank is selected depending upon two bank select bits in the PSW register. Next 16bytes are bit addressable. In total, 128bits (16X8) are available in bitaddressable area. Each bit can be accessed and modified by suitable instructions. The bit addresses are from 00H (LSB of the first byte in 20H) to 7FH (MSB of the last byte in 2FH). Remaining 80bytes of RAM are available for general purpose.

**Internal Data Memory and Special Function Register (SFR) Map**

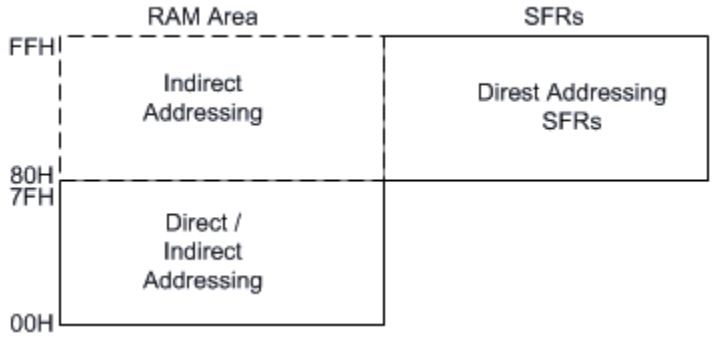



Fig 4.6 : Internal Data Memory Map

The special function registers (SFRs) are mapped in the upper 128 bytes of internal data memory address. Hence there is an address overlap between the upper 128 bytes of data RAM and SFRs. Please note that the upper 128 bytes of data RAM are present only in the 8052 family. The lower128 bytes of RAM (00H - 7FH) can be accessed both by direct or indirect addressing while the upper 128 bytes of RAM (80H - FFH) are accessed by indirect addressing.The SFRs (80H - FFH) are accessed by direct addressing only. This feature distinguishes the upper 128 bytes of memory from the SFRs, as shown in fig 4.6.

**SFR Map**

The set of Special Function Registers (SFRs) contains important registers such as Accumulator, Register B, I/O Port latch registers, Stack pointer, Data Pointer, Processor Status Word (PSW) and various control registers. Some of these registers are bit

addressable (they are marked with a * in the diagram below). The detailed map of various registers is shown in the following figure.

Address

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| F8H | | | | | | | | |
| F0H | B* | | | | | | | |
| E8H | | | | | | | | |
| E0H | ACC* | | | | | | | |
| D8H | | | | | | | | |
| D0H | PSW* | | | | | | | |
| C8H | (T2CON)* | | (RCAP2L) | (RCAP2H) | (TL2) | (TH2) | | |
| C0H | | | | | | | | |
| B8H | IP* | | | | | | | |
| B0H | P3* | | | | | | | |
| A8H | IE* | | | | | | | |
| A0H | P2* | | | | | | | |
| 98H | SCON* | SBUF | | | | | | |
| 90H | P1* | | | | | | | |
| 88H | TCON* | TMOD | TL0 | TL1 | TH0 | TH1 | | |
| 80H | P0* | SP | DPL | DPH | | | | PCON |

Fig 4.7: SFR Map

It should be noted hat all registers appearing in the first column are bit addressable. The bit address of a bit in the register is calculated as follows.

Bit address of 'b' bit of register 'R' is

Address of register 'R' + b

where $0 \leq b \leq 7$

**Processor Status Word (PSW) Address=D0H**

| CY | AC | F0 | RS1 | RS0 | OV | - | P |
|---|---|---|---|---|---|---|---|

Fi g 4.8: Processor Status Word

PSW register stores the important status conditions of the microcontroller. It also stores the bank select bits (RS1 & RS0) for register bank selection.

**Interfacing External Memory**

If external program/data memory are to be interfaced, they are interfaced in the following way.

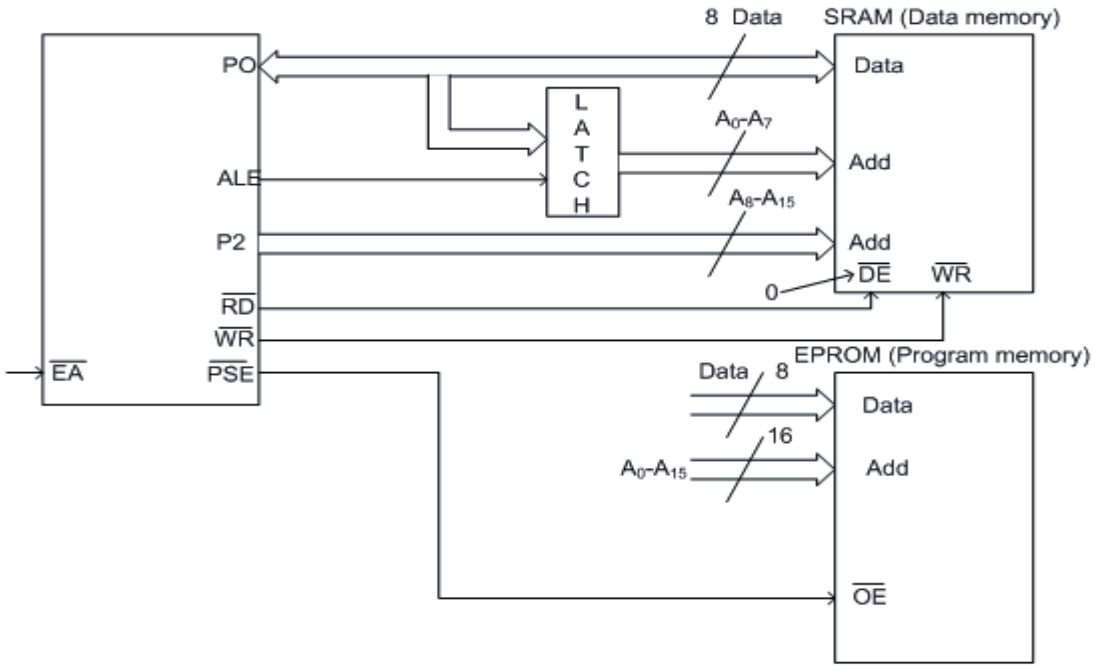



Fig 4.9: Circuit Diagram for Interfacing of External Memory External program memory is fetched if either of the following two conditions are satisfied.

1. $\overline{EA}$ (Enable Address) is low. The microcontroller by default starts searching for program from external program memory.
2. PC is higher than FFFH for 8051 or 1FFFH for 8052.

$\overline{PSEN}$ tells the outside world whether the external memory fetched is program memory or data memory. $\overline{EA}$ is user configurable. $\overline{PSEN}$ is processor controlled. **8051**

**Addressing Modes**

8051 has four addressing modes.

1. Immediate Addressing :

Data is immediately available in the instruction. For example -

ADD A, #77; Adds 77 (decimal) to A and stores in A

ADD A, #4DH; Adds 4D (hexadecimal) to A and stores in A

MOV DPTR, #1000H; Moves 1000 (hexadecimal) to data pointer *2.*

Bank Addressing or Register Addressing :

This way of addressing accesses the bytes in the current register bank. Data is available in the register specified in the instruction. The register bank is decided by 2 bits of Processor Status Word (PSW).

For example-

ADD A, R0; Adds content of R0 to A and stores in A

*3*.. Direct Addressing :

The address of the data is available in the instruction. For example -

MOV A, 088H; Moves content of SFR TCON (address 088H)to A 4.
Register Indirect Addressing :
The address of data is available in the R0 or R1 registers as specified in the instruction. For example -

MOV A, @R0 moves content of address pointed by R0 to A External
Data Addressing :
Pointer used for external data addressing can be either R0/R1 (256 byte access) or DPTR (64kbyte access).
For example -

MOVX A, @R0; Moves content of 8-bit address pointed by R0 to A

MOVX A, @DPTR; Moves content of 16-bit address pointed by DPTR to A External
Code Addressing :
Sometimes we may want to store non-volatile data into the ROM e.g. look-up tables. Such data may require reading the code memory. This may be done as follows - MOVC A, @A+DPTR; Moves content of address pointed by A+DPTR to A

MOVC A, @A+PC; Moves content of address pointed by A+PC to A

**I/O Port Configuration**

Each port of 8051 has bidirectional capability. Port 0 is called 'true bidirectional port' as it floats (tristated) when configured as input. Port-1, 2, 3 are called 'quasi bidirectional port'.
Port-0 Pin Structure
Port -0 has 8 pins (P0.0-P0.7).
The structure of a Port-0 pin is shown in fig 4.10.

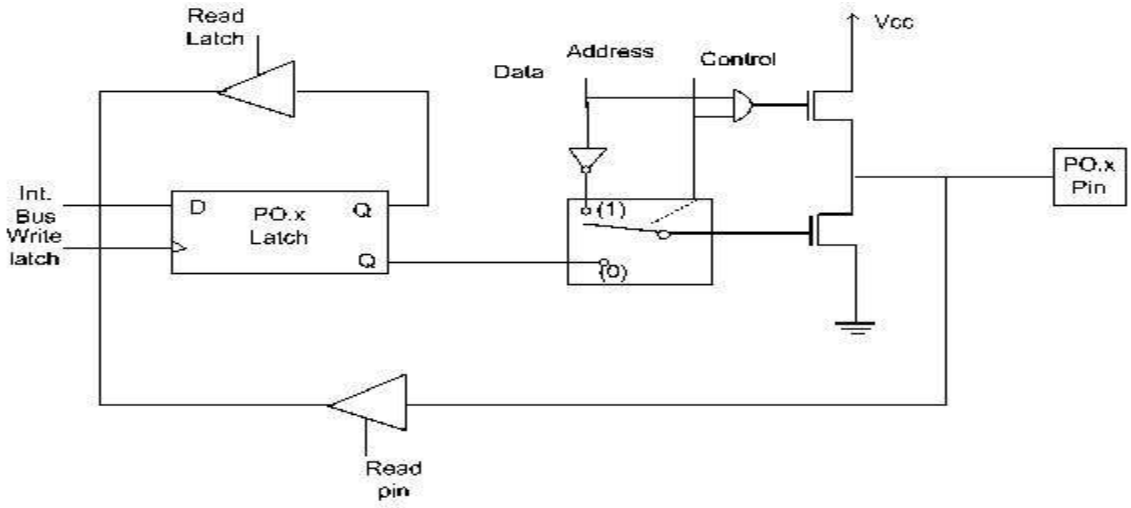



Fig 4.10: Port-0 Structure

Port-0 can be configured as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory. When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a normal bidirectional I/O port.

Let us assume that control is '0'. When the port is used as an input port, '1' is written to the latch. In this situation both the output MOSFETs are 'off'. Hence the output pin floats. This high impedance pin can be pulled up or low by an external source. When the port is used

as an output port, a '1' written to the latch again turns 'off' both the output MOSFETs and causes the output pin to float. An external pull-up is required to output a '1'. But when '0' is written to the latch, the pin is pulled down by the lower MOSFET. Hence the output becomes zero.

When the control is '1', address/data bus controls the output driver MOSFETs. If the address/data bus (internal) is '0', the upper MOSFET is 'off' and the lower MOSFET is 'on'. The output becomes '0'. If the address/data bus is '1', the upper transistor is 'on' and the lower transistor is 'off'. Hence the output is '1'. Hence for normal address/data interfacing (for external memory access) no pull-up resistors are required.

Port-0 latch is written to with 1's when used for external memory access.

Port-1 Pin Structure

Port-1 has 8 pins (P1.1-P1.7) .The structure of a port-1 pin is shown in fig 4.11.

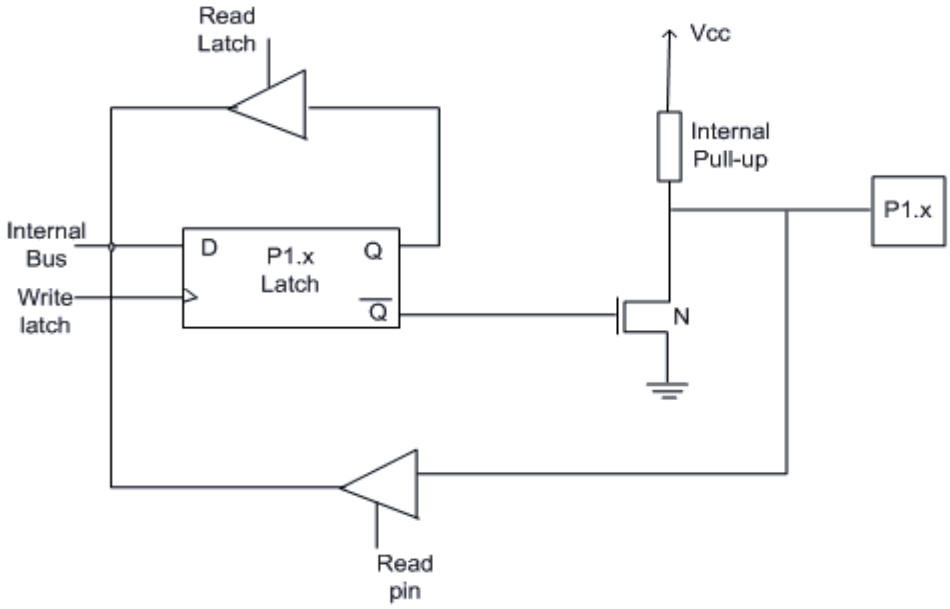



Fig 4.11 Port 1 Structure

Port-1 does not have any alternate function i.e. it is dedicated solely for I/O interfacing. When used as output port, the pin is pulled up or down through internal pull-up. To use port-1 as input port, '1' has to be written to the latch. In this input mode when '1' is written to the pin by the external device then it read fine. But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.

PORT 2 Pin Structure

Port-2 has 8-pins (P2.0-P2.7) . The structure of a port-2 pin is shown in fig 4.12.

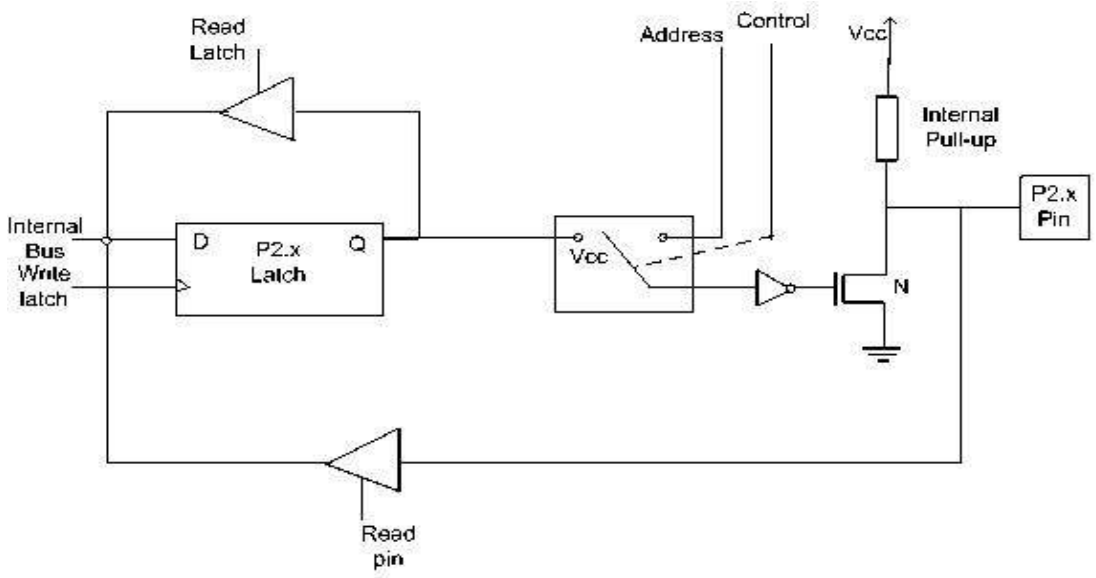



Fig 4.12 Port 2 Structure

Port-2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1. Port-2 latch remains stable when Port-2 pin are used for external memory access. Here again due to internal pull-up there is limited current driving capability.

PORT 3 Pin Structure

Port-3 has 8 pin (P3.0-P3.7) . Port-3 pins have alternate functions. The structure of a port3 pin is shown in fig 4.13.

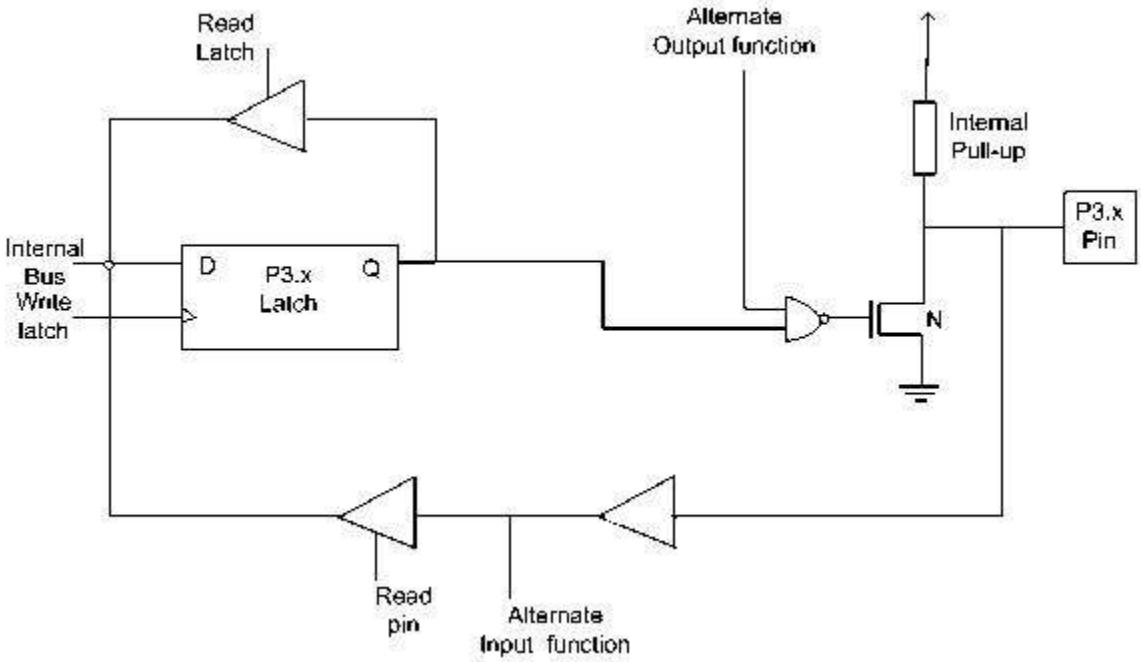



Fig 4.13 Port 3 Structure

Each pin of Port-3 can be individually programmed for I/O operation or for alternate function. The alternate function can be activated only if the corresponding latch has been written to '1'. To use the port as input port, '1' should be written to the latch. This port also has internal pull-up and limited current driving capability.

Alternate functions of Port-3 pins are -

| | |
|---|---|
| P3.0 | RxD |
| P3.1 | TxD |
| P3.2 | $\overline{\text{INT0}}$ |

| | |
|---|---|
| P3.3 | $\overline{INT1}$ |
| P3.4 | T0 |
| P3.5 | T1 |
| P3.6 | $\overline{WR}$ |
| P3.7 | $\overline{RD}$ |

Note:

1. Port 1, 2, 3 each can drive 4 LS TTL inputs.
2. Port-0 can drive 8 LS TTL inputs in address /data mode. For digital output port, it needs external pull-up resistors.
3. Ports-1,2and 3 pins can also be driven by open-collector or open-drain outputs.
4. Each Port 3 bit can be configured either as a normal I/O or as a special function bit.

**Reading a port (port-pins) versus reading a latch**

There is a subtle difference between reading a latch and reading the output port pin.

The status of the output port pin is sometimes dependant on the connected load. For instance if a port is configured as an output port and a '1' is written to the latch, the output pin should also show '1'. If the output is used to drive the base of a transistor, the transistor turns 'on'.

If the port pin is read, the value will be '0' which is corresponding to the base-emitter voltage of the transistor.

*Reading a latch:* Usually the instructions that read the latch, read a value, possibly change it, and then rewrite it to the latch. These are called "read-modify-write" instructions.
Examples of a few instructions are- ORL
P2, A; P2 <-- P2 or A

MOV P2.1, C; Move carry bit to PX.Y bit.
In this the latch value of P2 is read, is modified such that P2.1 is the same as Carry and is then written back to P2 latch.
*Reading a Pin:* Examples of a few instructions that read port pin, are-

MOV A, P0 ; Move port-0 pin values to A

MOV A, P1; Move port-1 pin values to A

Accessing external memory
Access to external program memory uses the $\overline{PSEN}$ signal (Program store enable) as the read strobe. Access to external $\overline{RD}$ or $\overline{WR}$ data memory uses (alternate function of P3.7 and P3.6).
For external program memory, always 16 bit address is used. For example -
MOVC A, @ A+DPTR
MOVC A, @ A+PC
Access to external data memory can be either 8-bit address or 16-bit address - 8-bit address- MOVX A, @Rp where Rp is either R0 or R1

MOVX @Rp, A
16 bit address- MOVX A,@DPTR
MOV X @DPTR, A
The external memory access in 8051 can be shown by a schematic diagram as given in fig 4.14.

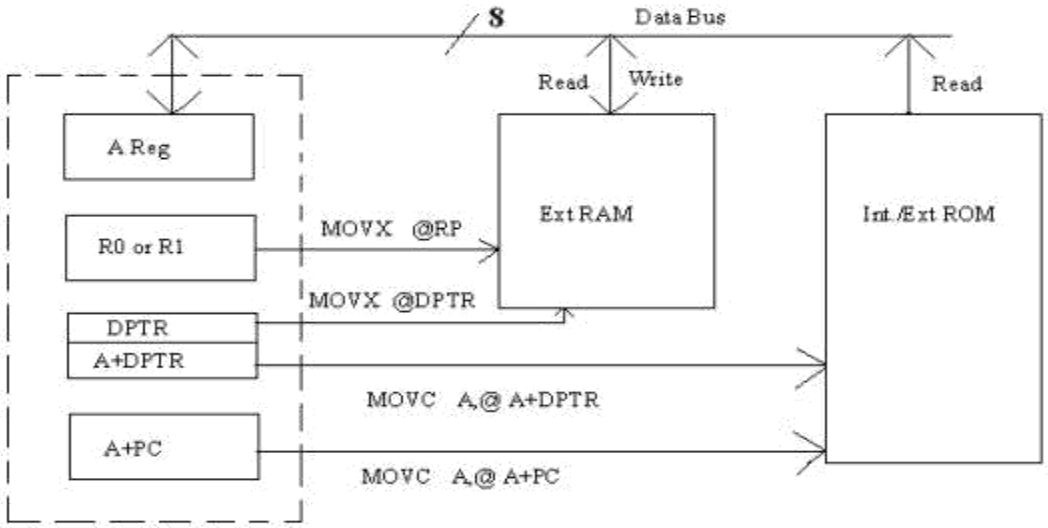



Fig 4.14 Schematic diagram of external memory access

If an 8-bit external address is used for data memory (i.e. MOVX @Rp) then the content of Port-2 SFR remains at Port-2 pins throughout the external memory cycle. This facilitates memory paging as the upper 8 bit address remains fixed.
During any access to external memory, the CPU writes FFH to Port-0 latch (SFR). If the user writes to Port-0 during an external memory fetch, the incoming byte is corrupted.
External program memory is accessed under the following condition.

1. Whenever $\overline{EA}$ is low, or
2. Whenever PC contains a number higher than 0FFFH (for 8051) or 1FFF (for 8052).

Some typical use of code/program memory access:
External program memory can be not only used to store the code, but also for lookup table of various functions required for a particular application. Mathematical functions such as Sine, Square root, Exponential, etc. can be stored in the program memory (Internal or eternal) and these functions can be accessed using MOVC instruction.

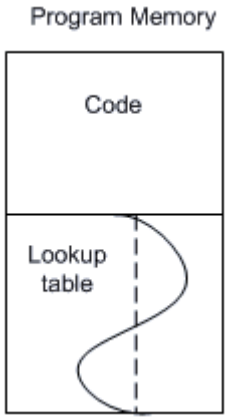


Fig 4.15 Program memory showing the storage of lookup table

Timers / Counters

8051 has two 16-bit programmable UP timers/counters. They can be configured to operate either as timers or as event counters. The names of the two counters are T0 and T1 respectively. The timer content is available in four 8-bit special function registers, viz, TL0,TH0,TL1 and TH1 respectively.

In the "timer" function mode, the counter is incremented in every machine cycle. Thus, one can think of it as counting machine cycles. Hence the clock rate is 1/12 th of the oscillator frequency.

In the "counter" function mode, the register is incremented in response to a 1 to 0 transition at its corresponding external input pin (T0 or T1). It requires 2 machine cycles to detect a high to low transition. Hence maximum count rate is 1/24 th of oscillator frequency.

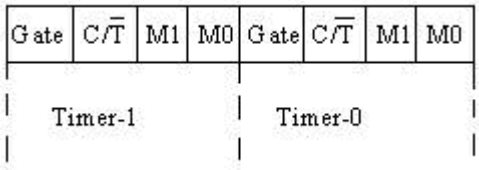
The operation of the timers/counters is controlled by two special function registers, TMOD and TCON respectively.

Timer Mode control (TMOD) Special Function Register:

TMOD register is not bit addressable.

TMOD

Address: 89 H




Various bits of TMOD are described as follows -

Gate: This is an OR Gate enabled bit which controls the effect of $\overline{INT1/0}$ on START/STOP of Timer. It is set to one ('1') by the program to enable the interrupt to start/stop the timer. If TR1/0 in TCON is set and signal on $\overline{INT1/0}$ pin is high then the timer starts counting using either internal clock (timer mode) or external pulses (counter mode).

$\underline{C/\overline{T}}$: It is used for the selection of Counter/Timer mode.

Mode Select Bits:

| M1 | M0 | Mode |
|---|---|---|
| 0 | 0 | Mode 0 |
| 0 | 1 | Mode 1 |
| 1 | 0 | Mode 2 |
| 1 | 1 | Mode 3 |

M1 and M0 are mode select bits.

Timer/ Counter control logic:

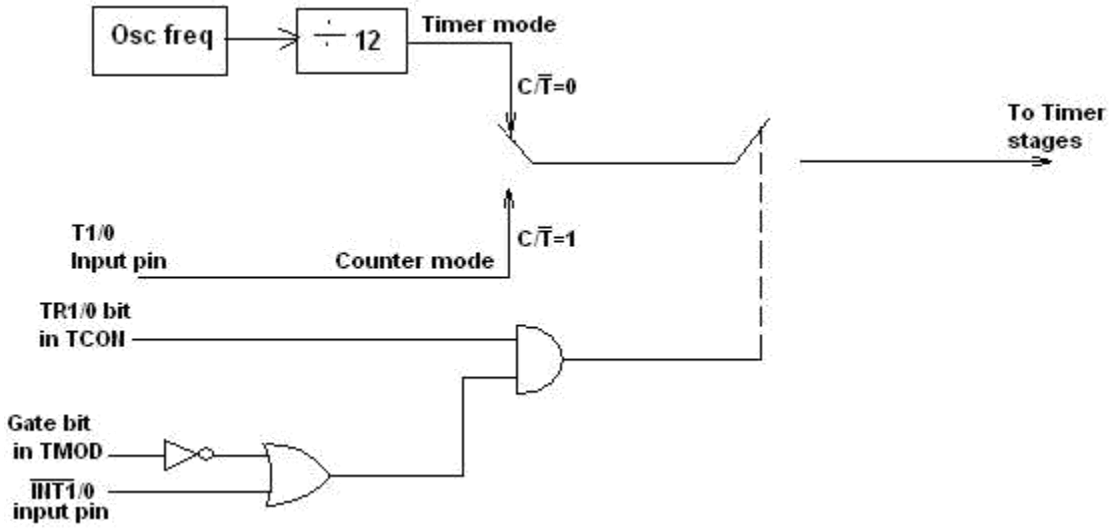



Fig 4.16 Timer/Counter Control Logic Timer

control (TCON) Special function register:

TCON is bit addressable. The address of TCON is 88H. It is partly related to Timer and partly to interrupt.

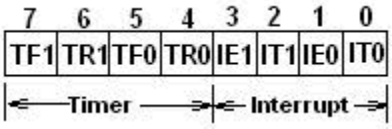



Fig 4.17 TCON Register The

various bits of TCON are as follows.

TF1 : Timer1 overflow flag. It is set when timer rolls from all 1s to 0s. It is cleared when processor vectors to execute ISR located at address 001BH. TR1 : Timer1 run control bit. Set to 1 to start the timer / counter. TF0 : Timer0 overflow flag. (Similar to TF1)

TR0 : Timer0 run control bit.

IE1 : Interrupt1 edge flag. Set by hardware when an external interrupt edge is detected. It is cleared when interrupt is processed. IE0 : Interrupt0 edge flag. (Similar to IE1)

IT1 : Interrupt1 type control bit. Set/ cleared by software to specify falling edge / low level triggered external interrupt.

IT0 : Interrupt0 type control bit. (Similar to IT1)

As mentioned earlier, Timers can operate in four different modes. They are as follows Timer Mode-0:

In this mode, the timer is used as a 13-bit UP counter as follows.

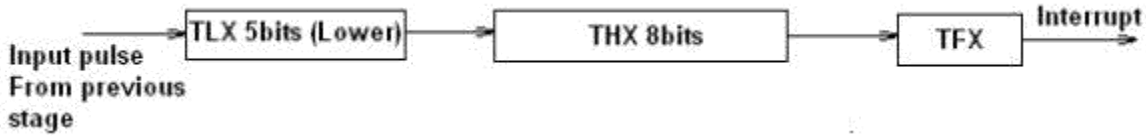



Fig. 4.18 Operation of Timer on Mode-0

The lower 5 bits of TLX and 8 bits of THX are used for the 13 bit count.Upper 3 bits of TLX are ignored. When the counter rolls over from all 0's to all 1's, TFX flag is set and an interrupt is generated.

The input pulse is obtained from the previous stage. If TR1/0 bit is 1 and Gate bit is 0, the counter continues counting up. If TR1/0 bit is 1 and Gate bit is 1, then the operation of the counter is controlled by $\overline{INTX}$input. This mode is useful to measure the width of a given pulse fed to $\overline{INTX}$ input.

Timer Mode-1:

This mode is similar to mode-0 except for the fact that the Timer operates in 16-bit mode.

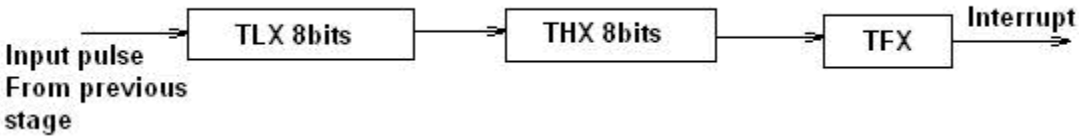



Fig 4.19 Operation of Timer in Mode 1

Timer Mode-2: (Auto-Reload Mode)

This is a 8 bit counter/timer operation. Counting is performed in TLX while THX stores a constant value. In this mode when the timer overflows i.e. TLX becomes FFH, it is fed with the value stored in THX. For example if we load THX with 50H then the timer in mode 2 will count from 50H to FFH. After that 50H is again reloaded. This mode is useful in applications like fixed time sampling.

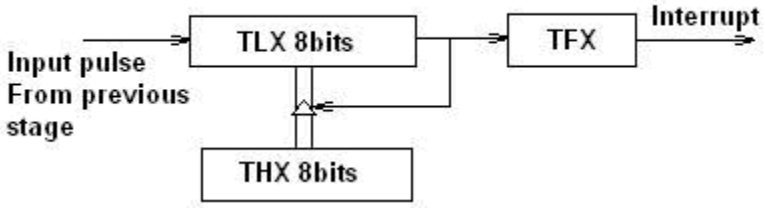



Fig 4.20 Operation of Timer in Mode 2 Timer

Mode-3:

Timer 1 in mode-3 simply holds its count. The effect is same as setting TR1=0. Timer0 in mode-3 establishes TL0 and TH0 as two separate counters.

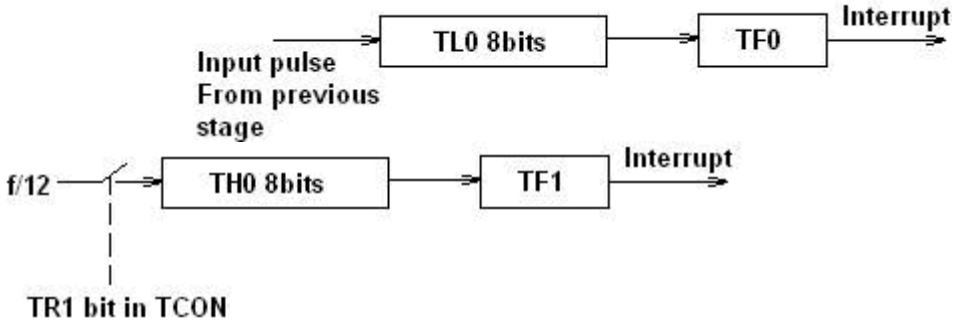



Fig 4.21 Operation of Timer in Mode 3

Control bits TR1 and TF1 are used by Timer-0 (higher 8 bits) (TH0) in Mode-3 while TR0 and TF0 are available to Timer-0 lower 8 bits(TL0).

**Interrupts**

8051 provides 5 vectored interrupts. They are -

1. $\overline{INT0}$
2. TF0

3. $\overline{INT1}$
4. TF1
5. RI/TI

Out of these, $\overline{INT0}$ an$\overline{INT1}$ are external interrupts whereas Timer and Serial port interrupts are generated internally. The external interrupts could be negative edge triggered or low level triggered. All these interrupt, when activated, set the corresponding interrupt flags. Except for serial interrupt, the interrupt flags are cleared when the processor branches to the Interrupt Service Routine (ISR). The external interrupt flags are cleared on branching to Interrupt Service Routine (ISR), provided the interrupt is negative edge triggered. For low level triggered external interrupt as well as for serial interrupt, the corresponding flags have to be cleared by software by the programmer.
The schematic representation of the interrupts is as follows -

Interrupt
Vector Location

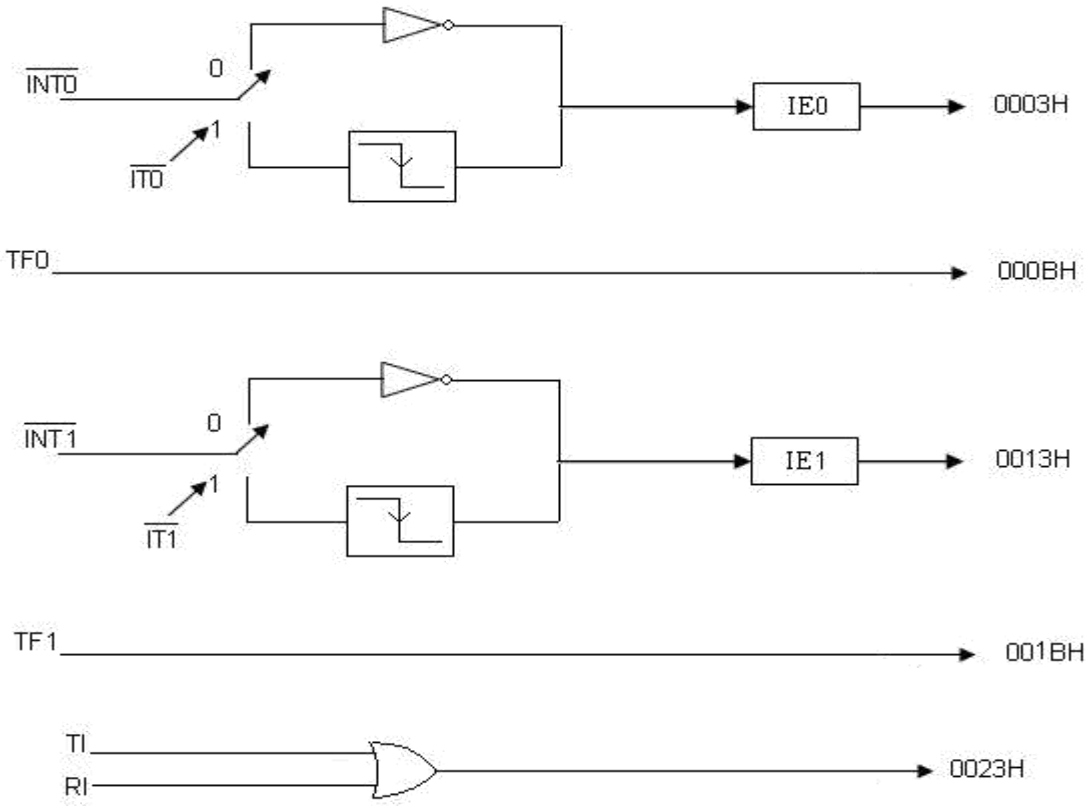



Fig 4.22 8051 Interrupt Details

Each of these interrupts can be individually enabled or disabled by 'setting' or 'clearing' the corresponding bit in the IE (Interrupt Enable Register) SFR. IE contains a global enable bit EA which enables/disables all interrupts at once.

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| EA | — | ET2 | ES | ET1 | EX1 | ET0 | EX0 |

**Interrupt Enable register (IE):** Address: A8H

→ $\overline{INT0}$ interrupt (External) enable bit

EX0

ET0 → Timer-0 interrupt enable bit

EX1 ⟶ $\overline{INT1}$ interrupt (External) enable bit

ET1 ⟶ Timer-1 interrupt enable bit

ES ⟶ Serial port interrupt enable bit

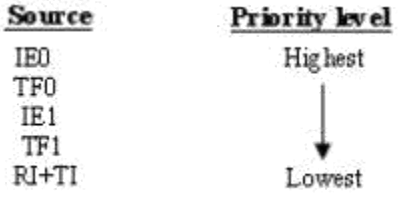
ET2 ⟶ Timer-2 interrupt enable bit

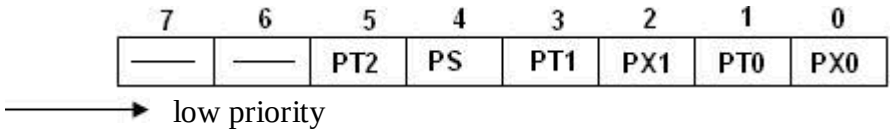
EA ⟶ Enable/Disable all

Setting '1' ⟶ Enable the corresponding interrupt

Setting '0' ⟶ Disable the corresponding interrupt **Priority level structure:**

Each interrupt source can be programmed to have one of the two priority levels by setting (high priority) or clearing (low priority) a bit in the IP (Interrupt Priority) Register . A low priority interrupt can itself be interrupted by a high priority interrupt, but not by another low priority interrupt. If two interrupts of different priority levels are received simultaneously, the request of higher priority level is served. If the requests of the same priority level are received simultaneously, an internal polling sequence determines which request is to be serviced. Thus, within each priority level, there is a second priority level determined by the polling sequence, as follows.




Interrupt Priority register (IP)

'0'

'1' ⟶ high priority **Interrupt handling:**

The interrupt flags are sampled at P2 of S5 of every instruction cycle (Note that every instruction cycle has six states each consisting of P1 and P2 pulses). The samples are polled during the next machine cycle (or instruction cycle). If one of the flags was set at S5P2 of the preceding instruction cycle, the polling detects it and the interrupt process generates a long call (LCALL) to the appropriate vector location of the interrupt. The LCALL is generated provided this hardware generated LCALL is not blocked by any one of the following conditions.

1. An interrupt of equal or higher priority level is already in progress.
2. The current polling cycle is not the final cycle in the execution of the instruction in progress.
3. The instruction in progress is RETI or any write to IE or IP registers.

When an interrupt comes and the program is directed to the interrupt vector address, the Program Counter (PC) value of the interrupted program is stored (pushed) on the stack. The required Interrupt Service Routine (ISR) is executed. At the end of the ISR, the instruction RETI returns the value of the PC from the stack and the originally interrupted program is resumed.

**Reset** is a non-maskable interrupt. A reset is accomplished by holding the RST pin high for at least two machine cycles. On resetting the program starts from 0000H and some flags are modified as follows -

| Register | Value(Hex) on Reset |
|---|---|
| PC | 0000H |
| DPTR | 0000H |
| A | 00H |
| B | 00H |
| SP | 07H |
| PSW | 00H |
| Ports P0-3 Latches | FFH |
| IP | XXX 00000 b |
| IE | 0 XX 00000 b |
| TCON | 00H |
| TMOD | 00H |
| TH0 | 00H |
| TL0 | 00H |
| TH1 | 00H |
| TL1 | 00H |
| SCON | 00H |
| SBUF | XX H |
| PCON | 0 XXXX XXX b |

The schematic diagram of the detection and processing of interrupts is given as follows.

Instruction Cycles ⟶

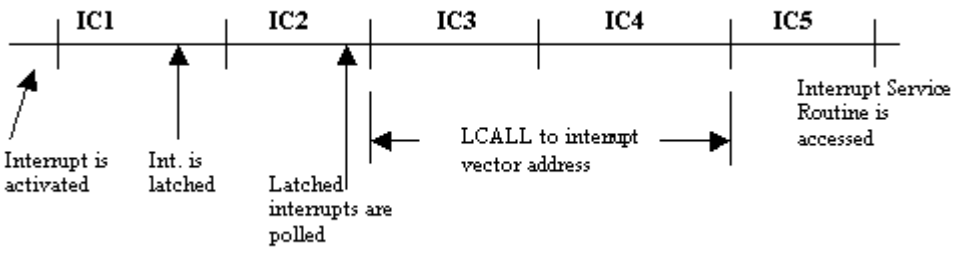


Fig 4.23 Interrupt Handling in 8051

It should be noted that the interrupt which is blocked due to the three conditions mentioned before is not remembered unless the flag that generated interrupt is not still active when the above blocking conditions are removed, i.e. ,every polling cycle is new.

**Jump and Call Instructions**

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

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

**Relative Jump**

Jump that replaces the PC (program counter) content with a new address that is greater than (the address following the jump instruction by 127 or less) or less than (the address following the jump by 128 or less) is called a relative jump. Schematically, the relative jump can be shown as follows: -

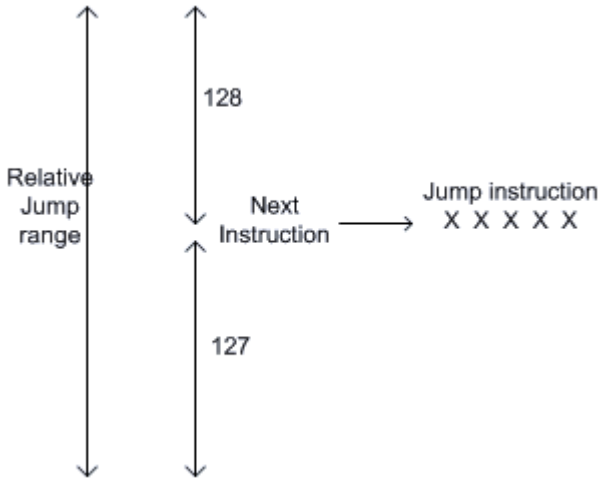



Fig 4.24 Relative Jump

The advantages of the relative jump are as follows:-

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

Disadvantages of the absolute jump: -

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

Instructions that use Relative Jump

SJMP <relative address>

(The remaining relative jumps are conditional jumps)

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

**Short Absolute Jump**

In this case only 11bits of the absolute jump address are needed. The absolute jump address is calculated in the following manner.
In 8051, 64 kbyte of program memory space is divided into 32 pages of 2 kbyte each. The hexadecimal addresses of the pages are given as follows:-

| Page (Hex) | Address (Hex) |
|---|---|
| 00 | 0000 - 07FF |
| 01 | 0800 - 0FFF |
| 02 | 1000 - 17FF |
| 03 | 1800 - 1FFF |
| . | . |
| 1E | F000 - F7FF |
| 1F | F800 - FFFF |

It can be seen that the upper 5bits of the program counter(PC) hold the page number and the lower 11bits of the PC hold the address within that page. Thus, an absolute address is formed by taking page numbers of the instruction (from the program counter) following the jump and attaching the specified 11bits to it to form the 16-bit address.
Advantage: The instruction length becomes 2 bytes.
However, difficulty is encountered when the next instruction following the jump instruction begins from a fresh page (at X000H or at X800H). This does not give any problem for the forward jump, but results in an error for the backward jump. In such a case the assembler prompts the user to relocate the program suitably.
Example of short absolute jump: - ACALL
<address 11>
AJMP <address 11>

**Long Absolute Jump/Call**

Applications that need to access the entire program memory from 0000H to FFFFH use long absolute jump. Since the absolute address has to be specified in the op-code, the instruction length is 3 bytes (except for JMP @ A+DPTR). This jump is not relocatable.
Example: -

LCALL <address 16>

LJMP <address 16> JMP
@A+DPTR

**Serial Interface**

The serial port of 8051 is full duplex, i.e., it can transmit and receive simultaneously.

The register SBUF is used to hold the data. The special function register SBUF is physically two registers. One is, write-only and is used to hold data to be transmitted out of the 8051 via TXD. The other is, read-only and holds the received data from external sources via RXD. Both mutually exclusive registers have the same address 099H.

**Serial Port Control Register (SCON)**

Register SCON controls serial data communication.

Address: 098H (Bit addressable)

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

Mode select bits

| SM0 | SM1 | Mode |
|---|---|---|
| 0 | 0 | Mode 0 |
| 0 | 1 | Mode 1 |
| 1 | 0 | Mode 2 |
| 1 | 1 | Mode 3 |

SM2: multi processor communication bit

REN: Receive enable bit

TB8: Transmitted bit 8 (Normally we have 0-7 bits transmitted/received)

RB8: Received bit 8

TI: Transmit interrupt flag

RI: Receive interrupt flag

**Power Mode control Register**

Register PCON controls processor powerdown, sleep modes and serial data bandrate. Only one bit of PCON is used with respect to serial communication. The seventh bit (b7)(SMOD) is used to generate the baud rate of serial communication.

Address: 87H

| b7 | | | | | | | b0 |
|---|---|---|---|---|---|---|---|
| SMOD | — | — | — | GF1 | GF0 | PD | IDL |

SMOD: Serial baud rate modify bit

GF1: General purpose user flag bit 1

GF0: General purpose user flag bit 0

PD: Power down bit

IDL: Idle mode bit

**Data Transmission**

Transmission of serial data begins at any time when data is written to SBUF. Pin P3.1 (Alternate function bit TXD) is used to transmit data to the serial data network. TI is

set to 1 when data has been transmitted. This signifies that SBUF is empty so that another byte can be sent.

**Data Reception**

Reception of serial data begins if the receive enable bit is set to 1 for all modes. Pin P3.0 (Alternate function bit 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 read.

**Serial Data Transmission Modes:**

Mode-0**:** In this mode, the serial port works like a shift register and the data transmission works synchronously with a clock frequency of $f_{osc}$ /12. Serial data is received and transmitted through RXD. 8 bits are transmitted/ received aty a time. Pin TXD outputs the shift clock pulses of frequency $f_{osc}$ /12, which is connected to the external circuitry for synchronization. The shift frequency or baud rate is always 1/12 of the oscillator frequency.

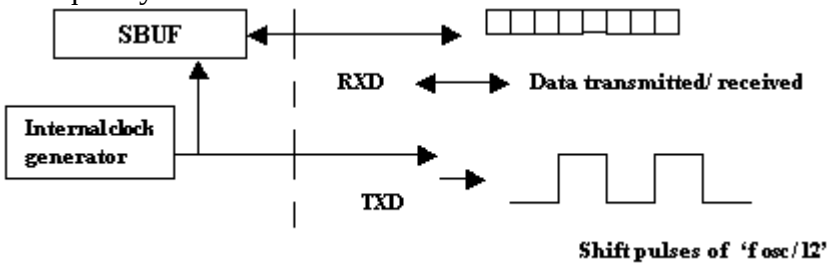



Fig 4.25 Data transmission/reception in Mode-0 **Mode-1** (standard UART mode) **:**

In mode-1, the serial port functions as a standard Universal Asynchronous Receiver Transmitter (UART) mode. 10 bits are transmitted through TXD or received through RXD. The 10 bits consist of one start bit (which is usually '0'), 8 data bits (LSB is sent first/received first), and a stop bit (which is usually '1'). Once received, the stop bit goes into RB8 in the special function register SCON. The baud rate is variable.

The following figure shows the way the bits are transmitted/ received.

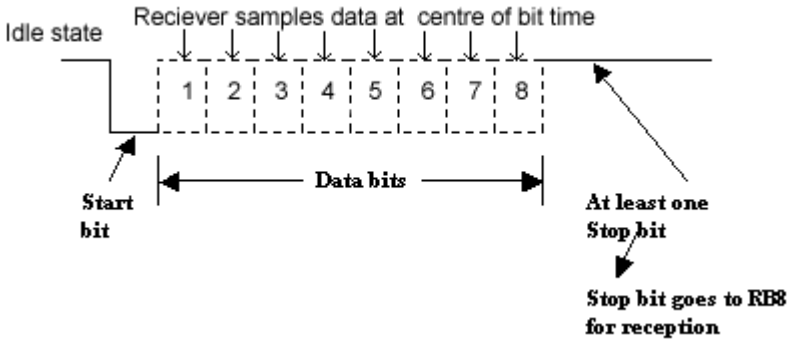



Fig 4.26 Data transmission format in UART mode

Bit time= $1/f_{baud}$

In receiving mode, data bits are shifted into the receiver at the programmed baud rate. The data word (8-bits) will be loaded to SBUF if the following conditions are true.

1. RI must be zero. (i.e., the previously received byte has been cleared from SBUF)
2. Mode bit SM2 = 0 or stop bit = 1.

After the data is received and the data byte has been loaded into SBUF, RI becomes one.

Mode-1 baud rate generation:

Timer-1 is used to generate baud rate for mode-1 serial communication by using overflow flag of the timer to determine the baud frequency. Timer-1 is used in timer mode-2 as an auto-reload 8-bit timer. The data rate is generated by timer-1 using the following formula.

$$f_{baud} = \frac{2^{SMOD}}{32} X \frac{fosc}{12 X [256-(TH1)]}$$

Where,

SMOD is the 7th bit of PCON register

$f_{osc}$ is the crystal oscillator frequency of the microcontroller

It can be noted that $f_{osc}$/ (12 X [256- (TH1)]) is the timer overflow frequency in timer mode-2, which is the auto-reload mode.

If timer-1 is not run in mode-2, then the baud rate is,

$$f_{baud} = \frac{2^{SMOD}}{32} X \text{ (timer-1 overflow frequency)}$$

Timer-1 can be run using the internal clock, fosc/12 (timer mode) or from any external source via pin T1 (P3.5) (Counter mode).

**Example:** If standard baud rate is desired, then 11.0592 MHz crystal could be selected. To get a standard 9600 baud rate, the setting of TH1 is calculated as follows. Assuming SMOD to be '0'

$$9600 = \frac{2^0}{32} X \frac{11.0592 X 10^6}{12 X (256-TH1)}$$

Or,

$$256-TH1 = \frac{1}{32} X \frac{11.0592 X 10^6}{12 X 9600} = 3$$

Or,

$$TH1 = 256 - 3 = 253 = FDH$$

In mode-1, if SM2 is set to 1, no receive interrupt (RI) is generated unless a valid stop bit is received.

**Serial Data Mode-2 - Multiprocessor Mode** :

In this mode 11 bits are transmitted through TXD or received through RXD. The various bits are as follows: a start bit (usually '0'), 8 data bits (LSB first), a programmable 9th (TB8 or RB8)bit and a stop bit (usually '1').

While transmitting, the 9th data bit (TB8 in SCON) can be assigned the value '0' or '1'. For example, if the information of parity is to be transmitted, the parity bit (P) in PSW could be moved into TB8. On reception of the data, the 9th bit goes into RB8 in 'SCON', while

the stop bit is ignored. The baud rate is programmable to either 1/32 or 1/64 of the oscillator frequency.

$$f_{baud} = (2^{SMOD}/64)\, f_{osc}.$$

**Mode-3 - Multi processor mode with variable baud rate :**

In this mode 11 bits are transmitted through TXD or received through RXD. The various bits are: a start bit (usually '0'), 8 data bits (LSB first), a programmable 9 th bit and a stop bit (usually '1').

Mode-3 is same as mode-2, except the fact that the baud rate in mode-3 is variable (i.e., just as in mode-1).

$$f_{baud} = (2^{SMOD}/32) * ( f_{osc} / 12\,(256\text{-}TH1)) .$$

This baudrate holds when Timer-1 is programmed in Mode-2.

**Operation in Multiprocessor mode** :

8051 operates in multiprocessor mode for serial communication Mode-2 and Mode-3. In multiprocessor mode, a Master processor can communicate with more than one slave processors. The connection diagram of processors communicating in Multiprocessor mode is given in fig 4.27.

The Master communicates with one slave at a time. 11 bits are transmitted by the Master, viz, One start bit (usually '0'), 8 data bits (LSB first), TB8 and a stop bit (usually '1'). TB8 is '1' for an address byte and '0' for a data byte.

If the Master wants to communicate with certain slave, it first sends the address of the slave with TB8=1. This address is received by all the slaves. Slaves initially have their SM2 bit set to '1'. All slaves check this address and the slave who is being addressed, responds by clearing its SM2 bit to '0' so that the data bytes can be received.

It should be noted that in Mode 2&3, receive interrupt flag RI is set if REN=1, RI=0 and the following condition is true.

1. SM2=1 and RB8=1 and a valid stop bit is received. Or
2. SM2=0 and a valid stop bit is received.

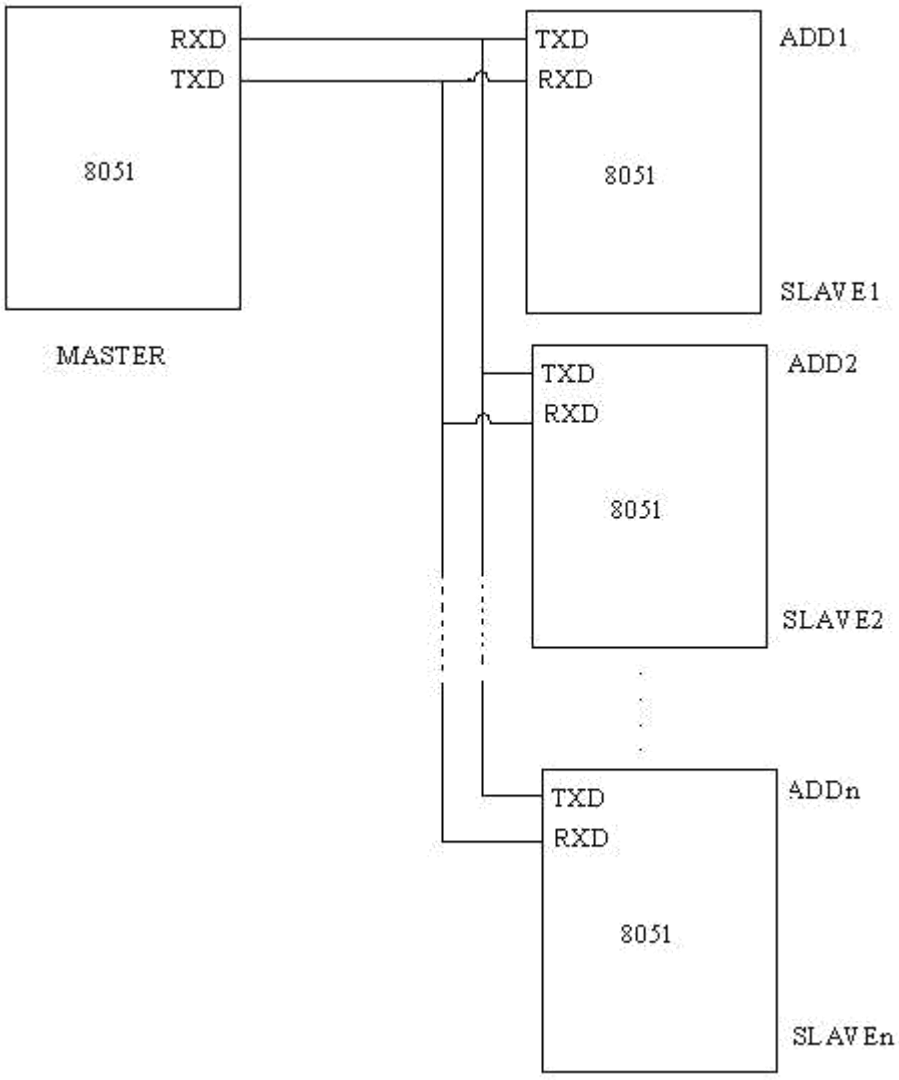



Fig 4.27 8051 in Multiprocessor Communication

After the communication between the Master and a slave has been established, the data bytes are sent by the Master with TB8=0. Hence other slaves do not respond /get interrupted by this data as their SM2 is pulled high (1).

**Power saving modes of operation :**

8051 has two power saving modes. They are -

1. Idle Mode
2. Power Down mode.

The two power saving modes are entered by setting two bits IDL and PD in the special function register (PCON) respectively. The structure of PCON register is as follows.

PCON: Address 87H

| SMOD | | | | GF1 | GFO | PD | IDL |
|---|---|---|---|---|---|---|---|

The schematic diagram for 'Power down' mode and 'Idle' mode is given as follows:

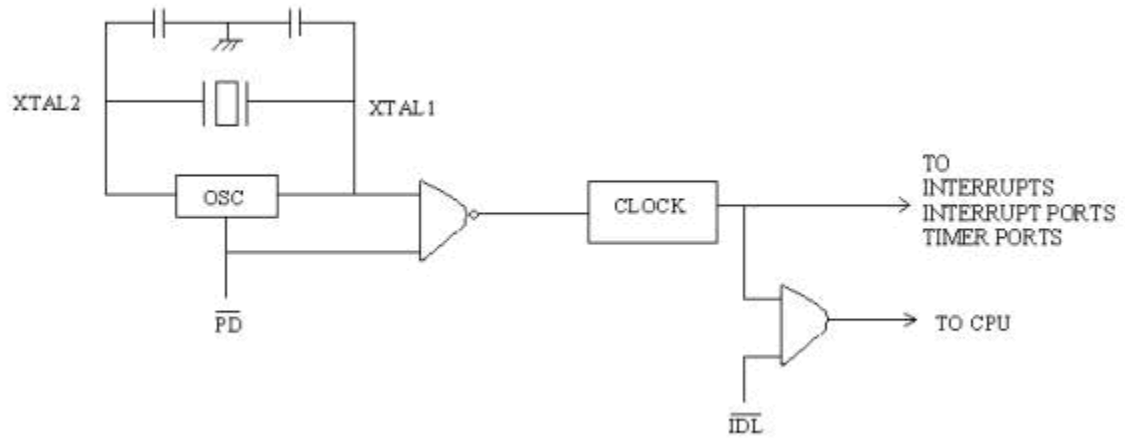



Fig 4.28 Schematic diagram for Power Down and Idle mode implementation

**Idle Mode**

Idle mode is entered by setting IDL bit to 1 (i.e., $\overline{IDL}$ =0). The clock signal is gated off to CPU, but not to the interrupt, timer and serial port functions. The CPU status is preserved entirely. SP, PC, PSW, Accumulator and other registers maintain their data during IDLE mode. The port pins hold their logical states they had at the time Idle was initiated. ALE and $\overline{PSEN}$ are held at logic high levels.

**Ways to exit Idle Mode:**

1. Activation of any enabled interrupt will clear PCON.0 bit and hence the Idle Mode is exited. The program goes to the Interrupt Service Routine (ISR). After RETI is executed at the end of the ISR, the next instruction will start from the one following the instruction that enabled Idle Mode.
2. A hardware reset exits the idle mode. The CPU starts from the instruction following the instruction that invoked the 'Idle' mode.

**Power Down Mode:**

The Power down Mode is entered by setting the PD bit to 1. The internal clock to the entire microcontroller is stopped (frozen). However, the program is not dead. The Power down Mode is exited (PCON.1 is cleared to 0) by Hardware Reset only. The CPU starts from the next instruction where the Power down Mode was invoked. Port values are not changed/ overwritten in power down mode. $V_{cc}$ can be reduced to as low as 2V in PowerDown mode. However, $V_{cc}$ has to be restored to normal value before PowerDown mode is exited.

**8051 Instructions**

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

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

The following nomenclatures for register, data, address and variables are used while write instructions. A: Accumulator

B: "B" register

C: Carry bit

Rn: Register R0 - R7 of the currently selected register bank

Direct: 8-bit internal direct address for data. The data could be in lower 128bytes of RAM (00 - 7FH) or it could be in the special function register (80 - FFH).

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

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

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

Addr11: 11-bit destination address for short absolute jump. Used by instructions AJMP & ACALL. Jump range is 2 kbyte (one page).

Addr16: 16-bit destination address for long call or long jump.

Rel: 2's complement 8-bit offset (one - byte) used for short jump (SJMP) and all conditional jumps.

bit: Directly addressed bit in internal RAM or SFR

**Arithmetic Instructions**

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

| | | | |
|---|---|---|---|
| | | | |
| SUBB A, direct | A ← A - (direct) - C | 2 | 1 |
| SUBB A, @Ri | A ← A - @Ri - C | 1 | 1 |
| SUBB A, #data | A ← A - data - C | 2 | 1 |

**Logical Instructions**

| Mnemonics | Description | Bytes | Instruction Cycles |
|---|---|---|---|
| ANL A, Rn | A ← A AND Rn | 1 | 1 |
| ANL A, direct | A ← A AND (direct) | 2 | 1 |
| ANL A, @Ri | A ← A AND @Ri | 1 | 1 |
| ANL A, #data | A ← A AND data | 2 | 1 |
| ANL direct, A | (direct) ← (direct) AND A | 2 | 1 |
| ANL direct, #data | (direct) ← (direct) AND data | 3 | 2 |
| CLR A | A ← 00H | 1 | 1 |
| CPL A | A ← A | 1 | 1 |
| ORL A, Rn | A ← A OR Rn | 1 | 1 |
| ORL A, direct | A ← A OR (direct) | 1 | 1 |
| ORL A, @Ri | A ← A OR @Ri | 2 | 1 |
| ORL A, #data | A ← A OR data | 1 | 1 |
| ORL direct, A | (direct) ← (direct) OR A | 2 | 1 |
| ORL direct, #data | (direct) ← (direct) OR data | 3 | 2 |
| RL A | Rotate accumulator left | 1 | 1 |
| RLC A | Rotate accumulator left through carry | 1 | 1 |
| RR A | Rotate accumulator right | 1 | 1 |
| RRC A | Rotate accumulator right through carry | 1 | 1 |
| SWAP A | Swap nibbles within Acumulator | 1 | 1 |
| XRL A, Rn | A ← A EXOR Rn | 1 | 1 |
| XRL A, direct | A ← A EXOR (direct) | 1 | 1 |
| XRL A, @Ri | A ← A EXOR @Ri | 2 | 1 |
| XRL A, #data | A ← A EXOR data | 1 | 1 |
| XRL direct, A | (direct) ← (direct) EXOR A | 2 | 1 |
| XRL direct, #data | (direct) ← (direct) EXOR data | 3 | 2 |
| | | | |

**Data Transfer Instructions**

| Mnemonics | Description | Bytes | Instruction Cycles |
|---|---|---|---|
| MOV A, Rn | A ← Rn | 1 | 1 |
| MOV A, direct | A ← (direct) | 2 | 1 |
| MOV A, @Ri | A ← @Ri | 1 | 1 |
| MOV A, #data | A ← data | 2 | 1 |
| MOV Rn, A | Rn ← A | 1 | 1 |
| MOV Rn, direct | Rn ← (direct) | 2 | 2 |
| MOV Rn, #data | Rn ← data | 2 | 1 |
| MOV direct, A | (direct) ← A | 2 | 1 |
| | | | |
| | | | |

MOV direct, (direct) Rn 2 2
Rn
MOV direct1, (direct1) (direct2) 3 2 direct2
MOV direct, (direct) @Ri 2 2
@Ri
MOV direct, (direct) #data 3 2
#data
MOV @Ri, A @Ri A 1 1 MOV @Ri, @Ri (direct)
2 2 direct
MOV @Ri, @Ri data 2 1
#data
MOV DPTR, DPTR data16 3 2
#data16
MOVC A, A Code byte pointed by A + 1 2
@A+DPTR DPTR
MOVC A, A Code byte pointed by A + PC 1 2
@A+PC
MOVC A, A Code byte pointed by Ri 8-bit 1 2
@Ri address)
MOVX A, A External data pointed by 1 2
@DPTR DPTR
MOVX @Ri, @Ri A (External data - 8bit 1 2
A address)
MOVX @DPTR A(External data - 16bit 1 2
@DPTR, A address)
PUSH direct (SP) (direct) 2 2

| | | | |
|---|---|---|---|
| POP direct | (direct) (SP) | 2 | 2 |
| | | | |
| XCH Rn | Exchange A with Rn | 1 | 1 |
| XCH direct | Exchange A with direct byte | 2 | 1 |
| XCH @Ri | Exchange A with indirect RAM | 1 | 1 |
| XCHD A, @Ri | Exchange least significant nibble of A with that of indirect RAM | 1 | 1 |

**Boolean Variable Instructions**

| Mnemonics | Description | Bytes | Instruction Cycles |
|---|---|---|---|
| CLR C | C-bit ⟵0 | 1 | 1 |
| CLR bit | bit ⟵0 | 2 | 1 |
| SET C | C ⟵1 | 1 | 1 |
| SET bit | bit ⟵1 | 2 | 1 |
| CPL C | C⟵ $\overline{\text{C-bit}}$ | 1 | 1 |
| CPL bit | bit ⟵ $\overline{\text{bit}}$ | 2 | 1 |
| ANL C, /bit | C ⟵C . $\overline{\text{bit}}$ | 2 | 1 |
| ANL C, bit | C ⟵C. bit | 2 | 1 |
| ORL C, /bit | C⟵ C + $\overline{\text{bit}}$ | 2 | 1 |
| ORL C, bit | C ⟵C + bit | 2 | 1 |
| MOV C, bit | C⟵ bit | 2 | 1 |
| MOV bit, C | bit ⟵C | 2 | 2 |

**Program Branching Instructions**

| Mnemonics | Description | Bytes | Instruction Cycles |
|---|---|---|---|
| ACALL addr11 | PC + 2 (SP) ; addr 11 PC | 2 | 2 |
| AJMP addr11 | Addr11 PC | 2 | 2 |
| CJNE A, direct, rel | Compare with A, jump (PC + rel) if not equal | 3 | 2 |
| CJNE A, #data, rel | Compare with A, jump (PC + rel) if not equal | 3 | 2 |
| CJNE Rn, #data, rel | Compare with Rn, jump (PC + rel) if not equal | 3 | 2 |
| CJNE @Ri, #data, rel | Compare with @Ri A, jump (PC + rel) if not equal | 3 | 2 |
| DJNZ Rn, rel | Decrement Rn, jump if not zero | 2 | 2 |
| DJNZ direct, rel | Decrement (direct), jump if not zero | 3 | 2 |
| JC rel | Jump (PC + rel) if C bit = 1 | 2 | 2 |
| JNC rel | Jump (PC + rel) if C bit = 0 | 2 | 2 |
| JB bit, rel | Jump (PC + rel) if bit = 1 | 3 | 2 |
| JNB bit, rel | Jump (PC + rel) if bit = 0 | 3 | 2 |
| JBC bit, rel | Jump (PC + rel) if bit = 1 | 3 | 2 |
| JMP @A+DPTR | A+DPTR PC | 1 | 2 |
| JZ rel | If A=0, jump to PC + rel | 2 | 2 |
| JNZ rel | If A ≠ 0 , jump to PC + rel | 2 | 2 |
| LCALL addr16 | PC + 3 (SP), addr16 PC | 3 | 2 |

| | | | |
|---|---|---|---|
| LJMP addr 16 | Addr16 PC | 3 | 2 |
| NOP | No operation | 1 | 1 |
| RET | (SP) PC | 1 | 2 |
| RETI | (SP) PC, Enable Interrupt | 1 | 2 |
| SJMP rel | PC + 2 + rel PC | 2 | 2 |
| JMP @A+DPTR | A+DPTR PC | 1 | 2 |
| JZ rel | If A = 0. jump PC+ rel | 2 | 2 |
| JNZ rel | If A ≠ 0, jump PC + rel | 2 | 2 |
| NOP | No operation | 1 | 1 |

Example programs

Character transmission using a time delay

A program shown below takes the character in 'A' register, transmits it, delays for transmission time, and returns to the calling program. Timer-1 is used to set the baud rate, which is 1200 baud in this program

The delay for one character transmission (in Mode 1 i.e.10 bits) is

10/2400 = 0.00833 seconds

Or, 8.33 milliseconds

Hence software delay of 10ms is used.

Timer-1 generates a baud rate close to 1200. Using a 12MHz crystal, the reload value is

$$256 - \frac{12X10^6}{32X12X2400} = 229.958$$

Or, 230 i.e. E6H

This gives rise to an actual baud rate of 1202.

SMOD is programmed to be 0.

Assembly language Program is as follows

```
RELOAD    EQU 0E6H   ; defining constant for reload value for baudrate
DELAY     EQU 0A6H   ; defining constant for 1 millisecond
DLYLSB    EQU 0AH    ; defining constant for 10 millisecond
DLYMSB    EQU 00H    ; defining constant

ORG 0000H            ; Org directive
ANL PCON, #7FH       ; SMOD = 0
ANL TMOD, #0FH       ; Alter only Timer-1
ORL TMOD, #20H       ; program Timer-1 in mode-2
MOV TH1,  #RELOAD    ; program reload value to TH1
SETB TR1             ; enable Timer-1 run bit (start timer)
MOV SCON, #40H       ; Serial port in mode-1 (receive not enabled)


TRMIT: MOV SBUF, #'A'    ; send the ASCII value of 'A'
       ACALL TRMITTIME   ; wait for DELAY
       SJMP TRMIT        ; again transmit
```

; Code to wait for the transmission to complete

The subroutine TRMITTIME generates a delay of about 10ms. With a clock of 12MHz, one instruction cycle time is

$$\frac{1}{12 \times 10^6} \times 12 = 1 \times 10^{-6}$$

The loop "MILSEC" generates a delay of about 1 x 10-3 sec. This gets executed 10 times for a total delay of 10 x 10-3 sec or 10ms

```
TRITTIME:      MOV A, #DLYLSB
               MOV B, #DLYMSB
               ACALL SOFTIME
               RET
SOFTIME:       PUSH 07H
               PUSH A
               ORL A, B
               CJNE A, #00H, OK
               POP A
               SJMP DONE

OK:            POP A

TIMER:         MOV R7, #DELAY  ; Generate delay for 1 millisecond
MILSEC:        NOP
               NOP
               NOP
               NOP
               DJNZ R7, MILSEC
               NOP
               NOP
               DEC A
               CJNE A, #0FFH, NO ROLL
               DEC B

NO ROLL:       CJNE A, #00H, TIMER
DONE:          POP 07H
               RET
END
```

Interrupt driven character transmission

In 8051, when a character is transmitted, SBUF register becomes empty and this generates a serial port interrupt (TI). TI and RI both point to the vector location 0023H in the program memory. An interrupt service routine can be written at 0023H to send the next character. A program is written here to transmit a character say 'A' continuously based on interrupt. The microcontroller uses a clock of 12MHz with a baud rate of 1202. The program is executed following a hardware reset. Assembly language program is as follows.

```
          RELOAD EQU 0E6H    ; defining reload constant for baudrate generation
          ORG 0000H          ; org directive
          SJMP START         ; jump to main program

SENDCH:   ORG 0023H          ; ISR for RI/TI interrupt
          CLR TI             ; clear transmit flag
          MOV SBUF, #'A'     ; send the ASCII value of 'A'
          RETI               ; return back to main program

START:    ANL PCON, #7FH     ; Set SMOD=0
          ANL TMOD, #0FH     ; Alter only the setting of Timer-1
          ORL TMOD, #20H     ; Timer-1 in mode-2
          MOV TH1, #RELOAD   ; Move the reload value to TH1
          SETB TR1           ; start Timer-1 for baud rate generation
          MOV SCON, #40H     ; set serial port in mode-1
          ORL IE, #90H       ; Enable serial port interrupt

          MOV SBUF, #'A'     ; Transmit a character

WAIT:     SJMP WAIT          ; wait till interrupt occurs

          END
```

Interrupt driven data reception

When a character is received, if receive mode is enabled, RI flag is set. This leads to the interruption of the main program and the processor goes to the interrupt vector location, i.e. 0023H for serial port. The interrupt service routine at 0023H gets executed to read the character so that the next character can be received. The following program receives a character on interrupt basis and outputs the character to port-1, possibly for a display. The crystal frequency is12MHz and baud rate is set at 1202 baud.

Assembly language program is as follows

```
          RELOAD EQU 0E6H    ; defining reload constant for bandrate generation

          ORG 0000H          ; org directive
          SJMP START         ; jump to main program

RCVCH:    ORG 0023H          ; ISR for RI/TI interrupt
          CLR RI             ; clear RI flag
          MOV P1, SBUF       ; write received character to port-1
          RETI               ; return back to main program

START:    ANL PCON, #07FH    ; SMOD=0
          ANL TMOD, #0FH     ; Alter only the setting of Timer-1
          ORL TMOD, #20H     ; Program Timer-1 in mode-2
          MOV TH1, #RELOAD   ; Move the reload value to TH1
          SETB TR1           ; Runs the timer for baud rate generation
          MOV SCON, #40H     ; programs serial port in mode-1
          SETB REN           ; Receive is enabled
          ORL IE, #90H       ; Serial interrupt is enabled

WAIT:     SJMP WAIT          ; wait until receive interrupt occurs

END
```

QUESTIONS:

1. Differentiate between microprocessors and microcontrollers.
2. What is a special function register?
3. Which port of 8051 is used as address/data bus?
4. What is function of RS1 and RS0 bits in the PSW of the 8051?
5. What is the address range of the bit-addressable memory of the 8051?
6. Write note on memory organization in the 8051.
7. Explain the stack operation in the 8051.
8. Where are the registers R0 – R7 located in the 8051?
9. Give one example each for one-byte, two-byte and three-byte instructions of the 8051.
10. When the instruction DJNZ useful?
11. Write a program to multiply two 8-bit numbers in the internal RAM and store the result in the external RAM.
12. Write a program to shift a 4-digit BCD number left by one digit. Assume that the data is stored in 30H and 31H.
13. Write a program to reverse the bits in a byte.
14. Write a program to find the biggest number in a block of data stored in the memory locations 70H – 7FH.
15. Write a program to generate a square wave of 10 KHz on the LSB of port 1 i.e. P1.0, using a timer.

REFERENCES:


1. 0000 to 8085 Introduction to microprocessor for scientist & engineers by Ghosh & Sridhar, PHI.
2. Fundamentals of microprocessor and microcontroller by B. RAM, Dhanpat Rai Publications.
3. Advanced microprocessor and peripherals (architecture, programming and interfacing) by A.K.Roy & K.M.Bhurchandi, TMH Publication.
4. Microprocessor, theory and applications by A.V.Deshmukh, TMH Publication.