GOVERNMENT OF TAMILNADU
DIRECTORATE OF TECHNICAL EDUCATION
CHENNAI – 600 025

STATE PROJECT COORDINATION UNIT

Diploma in Electronics and Communication Engineering

Course Code: 1040

M – Scheme

e-TEXTBOOK
on
MICRO CONTROLLER
for
V Semester DECE

Convener for ECE Discipline:

Dr.M.Jeganmohan,
Principal,
138, Government Polytechnic College,
Uthappanayakanur,Usilampatti- 625537.

Team Members for Industrial Electronics:

Mr.A.Mohamed Ali,
Lecturer (Sr.Gr.)/ ECE,
220,Pattukkottai Polytechnic College,
Pattukkottai –614 601 .

Mrs.M.S.Sumathi,
Lecturer (Sr.Gr.) / ECE,
537, Govt.Poly.College,
Gandharvakottai,
Pattukkottai –613301 .

Mr.P.Rajkumar,
Lecturer (Sr.Gr.)/ ECE,
340,K.L.N Polytechnic College,
Madurai – 625009.

Validated By

Dr.Mrs.Selvathi,
Professor/ ECE,
4690,Mepco Schlenk Engineering College,
Virudhunagar –626 005 .

GOVERNMENT OF TAMILNADU
DIRECTORATE OF TECHNICAL EDUCATION
CHENNAI – 600 025

STATE PROJECT COORDINATION UNIT

Diploma in Electronics and Communication Engineering

Course Code: 1040

M – Scheme

e-TEXTBOOK
on
MICRO CONTROLLER
for
V Semester DECE

Convener for ECE Discipline:

Dr.M.Jeganmohan,
Principal,
138, Government Polytechnic College,
Uthappanayakanur,Usilampatti- 625537.

Team Members for Industrial Electronics:

Mr.A.Mohamed Ali,
Lecturer (Sr.Gr.)/ ECE,
220,Pattukkottai Polytechnic College,
Pattukkottai –614 601 .

Mrs.M.S.Sumathi,
Lecturer (Sr.Gr.) / ECE,
537, Govt.Poly.College,
Gandharvakottai,
Pattukkottai –613301 .

Mr.P.Rajkumar,
Lecturer (Sr.Gr.)/ ECE,
340,K.L.N Polytechnic College,
Madurai – 625009.

Validated By

Dr.Mrs.Selvathi,
Professor/ ECE,
4690,Mepco Schlenk Engineering College,
Virudhunagar –626 005 .

CONTENTS

UNIT_-1ARCHITECTURE & INSTRUCTION SET of 8051

UNIT_-2
PROGRAMMING EXAMPLES

Unit – IV
INTERRUPT AND SERIAL COMMUNICATION

Unit – V
INTERFACING TECHNIQUES

UNIT I

1.ARCHITECTURE & INSTRUCTION SET

1.1ARCHITECTURE OF 8051

1.1.1**COMPARISON OF MICROPROCESSOR AND MICROCONTROLLER**

	Microprocessor	Microcontroller
1	It is suited for general purpose system	It is suited for special purpose system
2	Memory, Timer/Counter, I/O ports are not inbuilt in this chip	Memory, Timer/Counter, I/O ports are inbuilt in this chip
3	To make the system using microprocessor, More peripherals are required	To make the system using microcontroller, lesser no. of peripherals are enough
4	Examples: 8085,8086,pentium	Examples:8051,pic

1.1.1 Block Diagram of Microcontroller

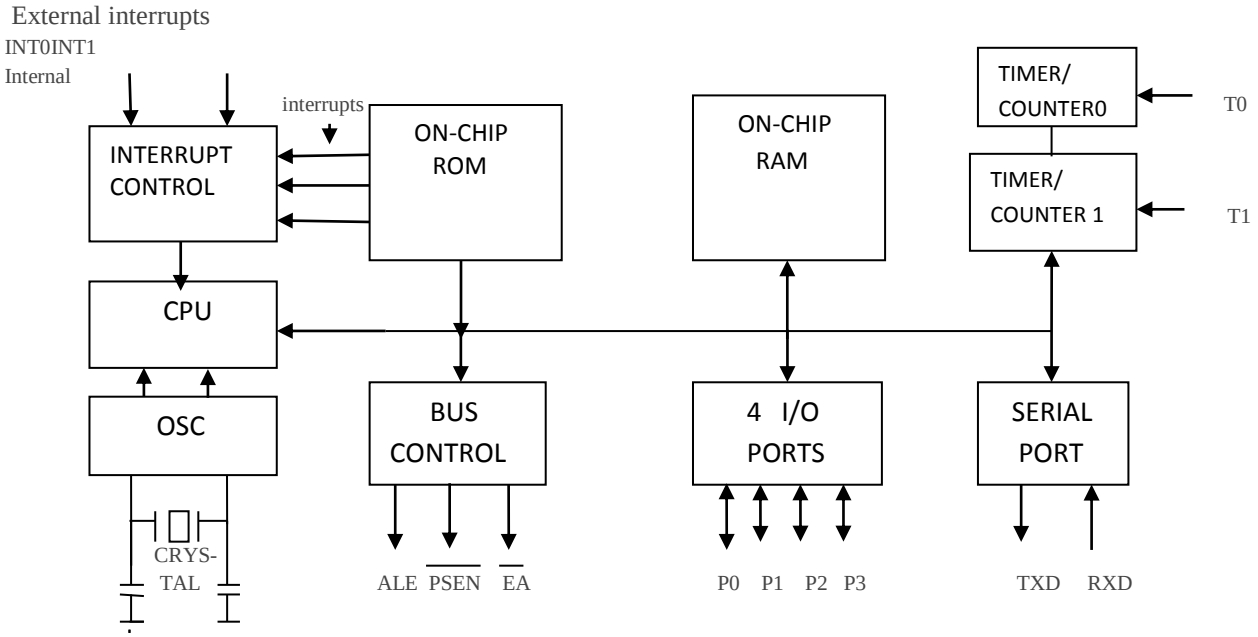


Fig 1.1 Block Diagram of Microcontroller

1.1.2 FUNCTIONS OF EACH BLOCK

Shown in the Fig 1.1 Block Diagram of Microcontroller

CPU- Central Processing Unit comprising of ALU and Control units

ALU-Arithmetic and Logic Unit performing the arithmetic and logical operations. These operations are addition, subtraction , multiplication, logical AND,OR etc. To do this operations one operand should be in Accumulator, another may in B register or in general purpose register. Mostly the result of the ALU operations are in A register. Some results are in B register also.

OSC

Oscillator provides clock for controller operation .Crystal oscillator provides stability and perfect clock. So crystal oscillator is used in this microcontroller .The crystal connected to the pins are intended for this purpose.

INTERRUPT CONTROLLER

Some interrupts are needed for microcontroller operation. Five interrupts are used.The controller controls the operation interrupts. ie some interrupts may be allowed, some others are disabled and priority assigned and changed.

BUS CONTROL

In 8051 ,Data Bus has a width of 8 bits and Address Bus has a width of 16 bits. Lower byte address bus are used for both Address and data. The bus usage is controlled by BUS control. There are 3 control signals, EA, PSEN and ALE. These signals known as External Access (EA), Program Store Enable (PSEN), and Address Latch Enable (ALE) are used for external memory interfacing.

ON CHIP RAM

The 8051 has 4 kilobyte of inbuilt ROM. It is otherwise called program memory. Usually program-code is stored in ROM. To store program into ROM , programmer is needed. If more area is required in ROM, an external ROM may be connected. Maximum of 64kb ROM memory can be used.

ON CHIP ROM

The 8051 has 128 byte of RAM as inbuilt . Some versions have 256 byte also. It is used as data memory. If the system needs more memory, external RAM may be connected up to 64kb.In 128 byte RAM chip, 00h to 71Fh are the address range .In this range ,00H to 1FH are the general purpose registers . 20H to 2F are the bit Addressable area and rest of this are byte addressable .This is used as general purpose scratch pad. In 256 byte RAM chip, another 128 byte are used for Special Function Registers.

I/O PORTS

There are four IO ports in 8051.These are named as P0,P1 ,P2,P3.All are bidirectional. Each port have its address , latch, output driver and input buffer. Each port are output by default. To make the port as input, it should be initialized by sending '1' level in each pin.

SERIAL PORT

TXD and RXD are used for serial port. These pins are available in Port 3. To transmit data serially TXD pin is used. To receive data serially RXD pin is used .Each pin have separate buffer registers named SBUF.

TIMER/COUNTER

There are two types counter/timer in 8051.These are named as Timer/Counter0 and Timer/Counter1. Each one may be used either as Timer or Counter. 16 bit timer register are used for counting in timer operation or counter operation. Clock pulses are counted in Timer operation and external events are counted in counter operation.

1.1.4 PIN DETAILS OF 8051

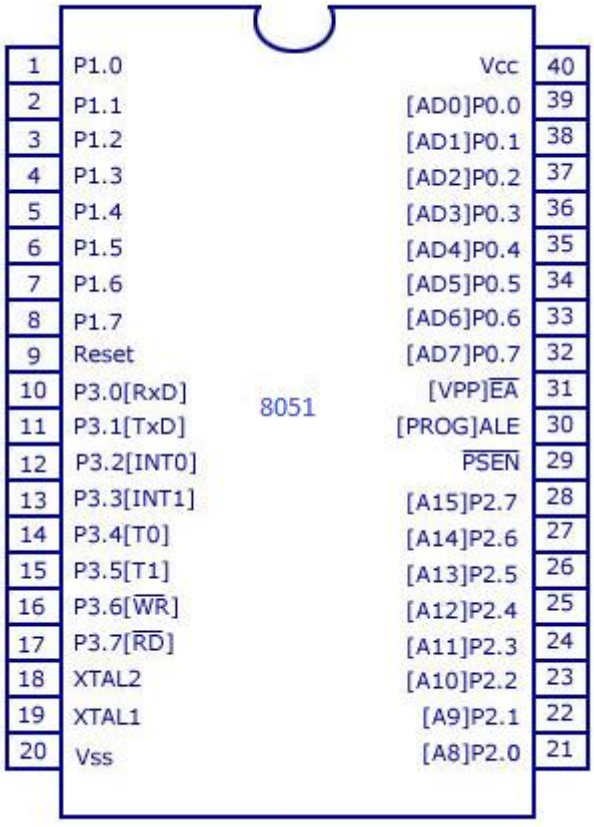


Fig 1.2 Pin Details Of 8051

Shown in the fig 1.2 Pin Details Of 8051

Pin-40 : This pin is named as VCC. Usually +5V DC is given to this pin.

Pins 32-39: Known as Port 0 (P0.0 to P0.7) – In addition to serving as I/O port, lower order address and data are multiplexed with this port (16 bit address is used for the purpose of external memory interfacing). This is a bi directional I/O port and external pull up resistors are required to use this port as I/O.

Pin-30:- ALE Address Latch Enable is used to de multiplex the address and data signal of port 0 (for external memory interfacing.) When address moves on port 0,ALE will be high. It is the indication to hold the Address in latch .

Pin-31:- $\overline{EA}$ External Access input is used to enable or disable external memory interfacing. It is low enable pin. If there is no external memory requirement, this pin is pulled high by connecting it to Vcc.

Pin- 29:- $\overline{PSEN}$ or Program Store Enable is used to read data from external program memory.

Pins- 21-28:- Known as Port 2 (P 2.0 to P 2.7) – in addition to serving as I/O port, higher order address bus signals are multiplexed with this port.

Pin 20:- Named as Vss – it represents ground (0 V) connection.

Pins 18 and 19:- Used for connecting an crystal externally to provide system clock.

Pins 10 – 17:- Known as Port 3. This port also serves some other functions like interrupts, timer input, control signals for external memory interfacing $\overline{RD}$ and $\overline{WR}$, serial communication signals RxD and TxD etc. This is a quasi bi directional port with internal pull up resister. It is also called multifunctional port

Pin 9:- As explained before RESET pin is used to set the 8051 microcontroller to its initial values, while the microcontroller is working or at the initial start of application. To reset the microcontroller ,the reset pin must be set high for 2 machine cycles.

Pins 1 – 8:- Known as Port 1. Unlike other ports, this port does not serve any other functions. Port 1 is an internally pulled up, quasi bi directional I/O port.

1.1.5 ALU

ALU- Arithmetic and Logic Unit. Arithmetic and logical operations are carried out. The arithmetic operations are Addition, Subtraction, multiplication and Division etc. The logical operations are AND,OR,XOR etc. To do the operations, one operand must be in A register another may be general purpose register. In multiplication and division ,the other operand must be in B register. The result of the arithmetic and logical operations are available in A register. In multiplication, the low byte answer is in A register and high byte answer is in B register. In division, quotient is in A register and remainder is in B register. Clear, Complement and Rotate are also done in ALU.

1.1.6 ROM

The 8051 has 4 kilobytes of inbuilt ROM. Additional ROM can be used as external memory at the maximum size of 64 kilo bytes.

To use inbuilt ROM, the pin $\overline{EA}$ must be high. To use external ROM, the pin $\overline{EA}$ must be low. The ROM is used for code storage. So it is called program (code) memory. The controller 8031 is ROM less version. The external ROM must be used when the ROM less version is used.

1.1.7 RAM

The 8051 has 128 byte of inbuilt RAM. Some versions have 256 bytes also.128 bytes are used for Banked register and bit addressable area and byte addressable area and rest for scratch pad. The remaining 128 bytes in 256byte chip are used for SFR's. It can be extended as external RAM up to 64 kilobytes. The RAM is also called data memory. MOV instruction is used to access internal memory. MOVX instruction is used to access external memory. $\overline{RD}$ and $\overline{WR}$ signals are used in external memory access. While accessing external RAM , data pointer DPTR is used to hold memory address.

1.1.8 MEMORY ORGANIZATION

The 8051 has two types of memory . They are Program Memory and Data Memory. Program Memory (ROM) is used to save the program permanently .This will be executed .The Data Memory (RAM) is used for storing data and intermediate results temporarily .This results are created and used during the operation of the microcontroller. In the 8051 microcontroller family , at most a 4 Kb of ROM and 128 or 256 bytes of RAM is used. All 8051 microcontrollers have a 16-bit addressing bus and are capable of addressing 64 kb memory. Arranging the available memory for programmer use, is called memory organization.

Program Memory(ROM)

The first models of the 8051 microcontroller family did not have internal program memory. It was added as an external separate chip. These models are recognizable by their label beginning with 803X (for example 8031 or 8032). All later models have a few Kbyte ROM inbuilt. Some times it is necessary to use additional memory if the amount of inbuilt memory is not sufficient for writing most of the programs. To access external ROM, the pin $\overline{EA}$ must be connected to Vss(GND). If $\overline{EA}$ connected to Vss, the addresses 0000h to 0FFF h are directed to external ROM. If $\overline{EA}$ connected to Vcc, the addresses 0000h to 0FFF h are directed to internal ROM , The addresses 1000 h to FFFF h are always directed to external ROM irrespective of EA.

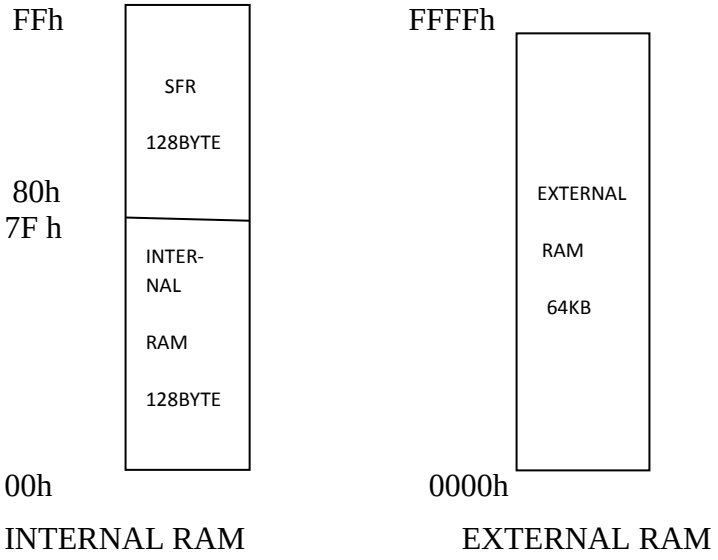


Fig 1.3 Memory Map of data memory

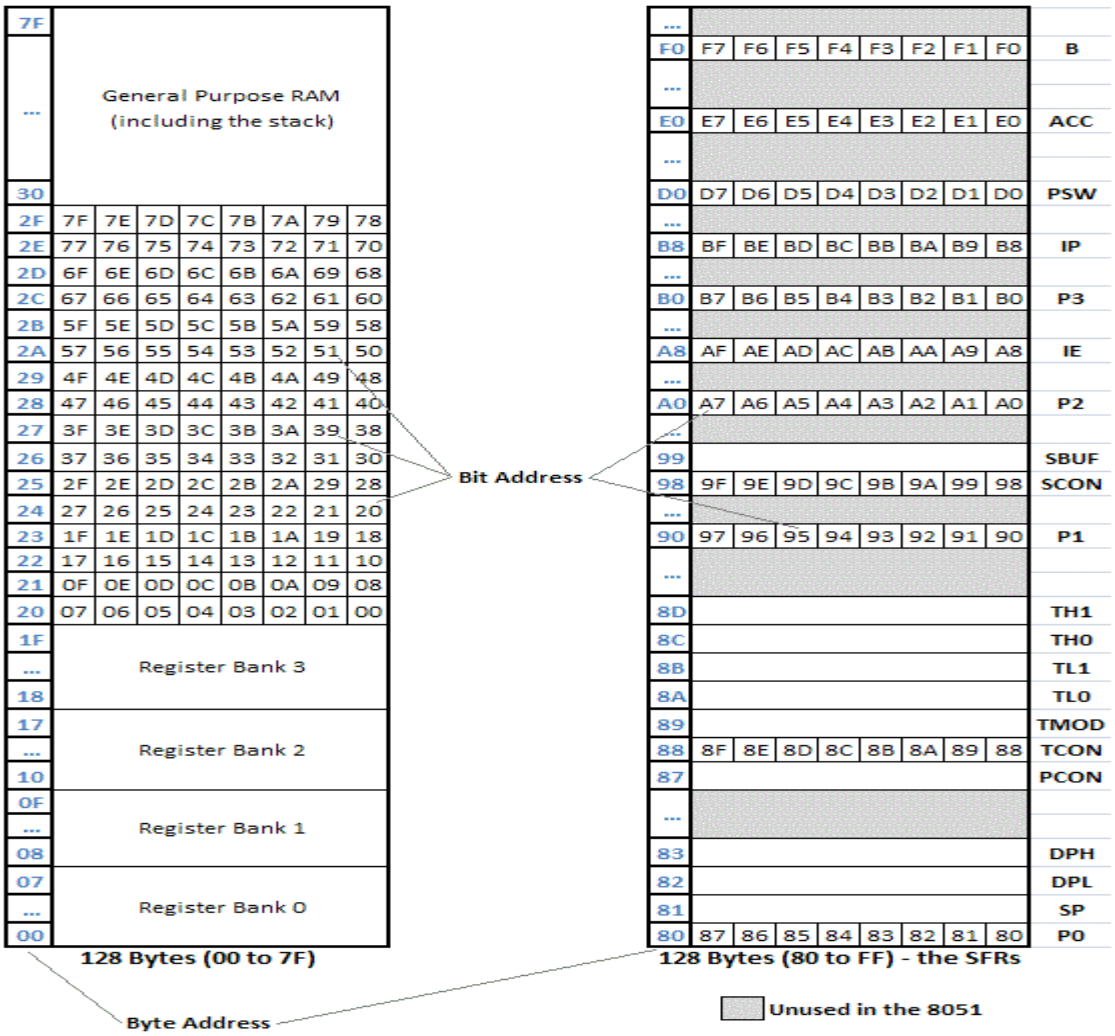


Fig 1.6 Register Banks

There are four register banks from 00H to 1FH. On power-up, registers R0 to R7 are located at 00H to 07H. However, this can be changed so that the register set points to any of the other three banks (if you change to Bank 2, for example, R0 to R7 is now located at 10H to 17H).

Bit-addressable Locations

The 8051 contains 210 bit-addressable locations. In this 210 locations, 128 are at the address 20H to 2FH in internal RAM. The rest are in the SFRs. Each of the 128 bits from 20H to 2FH have a unique number (address) attached to them, as shown in the table above. The 8051 instruction set allows you to set or reset any single bit in this section of RAM. In the general purpose RAM from 30H to 7FH and the register banks from 00H to 1FH are only byte addressable locations. In this locations as byte can be read and/ or write as byte . However, with bit-addressable RAM (20H to 2FH) read or write any single bit in this region by using the unique address for that bit.

1.1.9 SPECIAL FUNCTION REGISTERS(SFR)

Locations 80H to FFH contain the special function registers. As we can see from the diagram above, not all locations are used by the 8051 These extra locations are used by other family members (8052, etc.) for the extra features those microcontrollers possess. Also note that not all SFRs are bit-addressable. We progress through the course, but for the moment you should take note of the accumulator (ACC) at address 0E0H and the four port registers at addresses 80H for P0, 90h for P1, 0A0 for P2 and 0B0 for P3.

Special Function Registers (SFRs) are registers used for special functions and the operation of the microcontroller. Each one has its own name and address. Only 21 SFR's in 8051 are available to user. Among the 21 registers, 11 registers are bit and byte addressable. The remaining 10 are bit addressable . Each one are 8 bit registers .

TMOD,TCON,TH1,TL0,TH1and TH0 are used for Timer operation . SCON and SBUF are the SFR's used for serial communication. P0,P1,P2 andP3 are used for I/O port. PCON is used for power control. IE and IP are used as SFR used for interrupt control.DPH and DPL are the 8bit SFR,s in DPTR . A and B registers are also the SFR's used in instruction

.Symbol	Name	Byte Address	Bit Address MSB LSB
ACC*	Accumulator	0EOH	E7H - EOH
B*	B register	0FOH	F7H - FOH
PSW*	Program Status Word	0D0H	D7H - D0H
SP	Stack pointer	81H	-
DPL	Low Byte DPTR	82H	-
DPH	High Byte DPTR	83H	-
P0*	Port 0	80H	87H - 80H
P1*	Port 1	90H	97H - 90H
P2*	Port 2	0A0H	A7H - A0H
P3*	Port 2	0B0H	B7H - B0H
IP*	Interrupt Priority control	0B8H	BFH - B8H
IE*	Interrupt Enable control	0A8H	AFH - A8H
TMOD	Timer/Counter mode control	89H	
TCON*	Timer / Counter control	88H	8FH - 88H
TH0	Timer / Counter 0 High Byte	8CH	-
TL0	Timer / Counter 0 High Byte	8AH	-
TH1	Timer / Counter 1 High Byte	8DH	-
TL1	Timer / Counter 1 Low Byte	88H	-
SCON*	Serial Control	98H	9FH - 98H
SBUF	Serial Data Buffer	99H	-
PCON	Power Control	87H	-

Fig 1.7 Table of the SFR Bit & Byte Address

1.1.10 PROGRAM COUNTER

The Program Counter (PC) is a 2-byte register. It is memory pointer. It holds the address of instruction. This instruction will be executed next. When the 8051 is initialized, PC always starts at 0000h. The first instruction to be executed must be stored in the address 0000h. The address in the program counter may be in the range of 0000h to FFFFh. It is incremented each time an instruction is executed. It is important to note that PC is not always incremented by one. Since some instructions require 2 or 3 bytes the PC will be incremented by 2 or 3 in these cases. The Program Counter is special .There is no way to directly modify its value.

1.1.11 STACK

It is a part of RAM in which data will be stored temporary during execution of program. STACK works on last in first out principle. To store and retrieve data during program execution in stack, push and pop instruction are used .PUSH is used to store data into stack.POP is used to retrieve data from stack.

During the execution of CALL instructions, the microcontroller stores the content of program counter in the stack memory. During the RET instructions , the content of stack is moved in to program counter. Stack Pointer is the register(SFR) used to point the address of stack. The address of stack memory is placed in stack pointer. The stack pointer in the 8051 is only 8 bits wide, which means that it can take value 00 to FFH.

During PUSH operation ,the Stack Pointer has the next address of stack The stack pointer is incremented during PUSH.

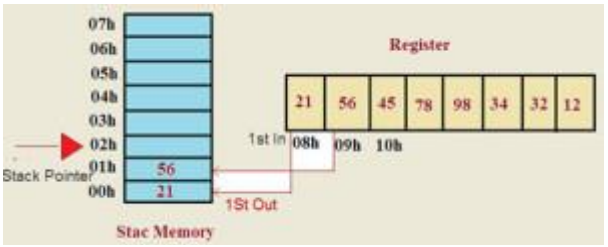


Fig 1.8 Function of Stack Memory

During POP operation ,the Stack Pointer has the address of stack from which data is accessed next. The Stack Pointer is used to indicate where the next value to be removed from the stack. The stack pointer is decremented during POP.

1.1.12 Program Status Word (PSW)

CY	AC	F0	RS1	RS0	OV	-	P

It is an eight bit register. This register is used to know the status ALU operation and used to select the register bank. It is a bit addressable register.
The table below describes the function of each bit.

Bit	Symbol	Address	Description
PSW.7	CY	D7H	Carry flag
PSW.6	AC	D6H	Auxiliary carry flag
PSW.5	F0	D5H	Flag 0
PSW.4	RS1	D4H	Register bank select 1
PSW.3	RS0	D3H	Register bank select 0
PSW.2	OV	D2H	Overflow flag
PSW.1	--	D1H	Reserved
PSW.0	P	D0H	Even parity flag

Carry Flag

The carry flag has two functions.

- It is used as the carry-out in 8-bit addition/subtraction. For example, if the accumulator contains FDH and we add 3 to the contents of the accumulator (ADD A, #3), the accumulator will then contain zero and the carry flag will be set. It is also set if a subtraction causes a borrow into bit 7. In other words, if a number is subtracted from another number smaller than it, the carry flag will be set. For example, if A contains 3DH and R3 contains 4BH, the instruction SUBB A, R3 will result in the carry bit being set (4BH is greater than 3DH).

- The carry flag is also used during Boolean operations. For example AND the contents of bit 3DH with the carry flag, the result being placed in the carry flag - ANL C, 3DH

Register Bank Select Bits

Bits 3 and 4 of the PSW are used for selecting the register bank. Since there are four register banks, two bits are required for selecting a bank, as detailed below.

PSW.4 RS 1	PSW.3 RS 0	Register Bank	Address of Register Bank
0	0	0	00H to 07H
0	1	1	08H to 0FH
1	0	2	10H to 17H
1	1	3	18H to 1FH

For example, if we wished to activate register bank 3 we would use the following instructions -
SETB RS1

SETB RS0

If we then moved the contents of R4 to the accumulator (MOV A, R4) we would be moving the data from location 1CH(location of Bank 3) to A.

Flag 0

Flag 0 is a general-purpose flag available to the programmer.

Parity Bit

The parity bit is automatically set or cleared in every machine cycle to ensure even parity with the accumulator. The number of 1-bits in the accumulator plus the parity bit is always even. In other words, if the number of 1s in the accumulator is odd then the parity bit is set to make the overall number of bits even. If the number of 1s in the accumulator is even then the parity bit is cleared to make the overall number of bits even.

For example, if the accumulator holds the number 05H, this is 0000 0101 in binary => the accumulator has an even number of 1s, therefore the parity bit is cleared. If the accumulator holds the number F2H, this is 1111 0010 => the accumulator has an odd number of 1s, therefore the parity bit is set to make the overall number of 1s even.

As we shall see later in the course, the parity bit is most often used for detecting errors in transmitted data.

1.1.13 I/O PORTS

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

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

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

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

Port 0

The P0 (zero) port is characterized by two functions −When the external memory is used then the lower address byte (addressesA0…A7) is applied on it, else all bits of this port are configured as input/output.The P0 port pins have no pull- up resisters. The other ports (P1,P2,P3) pins have built-in pull-up resistors. The pull-up resisters are connected between 5V power supply and the pins of this port.

Port 1

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

Port 2

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

Port 3

This port is called multifunctional port. Each pins are assigned alternate functions

PORT 3 ALTERNATE FUNCTIONS :

P3 BIT	FUNCTION	PIN
P3.0	RXD	10
P3.1	TXD	11
P3.2	$\overline{\text{INT0}}$	12
P3.3	$\overline{\text{INT1}}$	13
P3.4	TO	14
P3.5	TI	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17

Functions of port 3

P3.0 and P3.1 are used for the RxD and TxD serial communications signals. Bits P3.2 and P3.3 are set aside for external interrupts. Bits P3.4 and P3.5 are used for timers 0 and 1.

Finally P3.6 and P3.7 are used to provide the WR and RD signals of external memories connected in 8031 based systems.

Initialising 8051 Port Pins

Because of the way the 8051 port pin circuitry is designed, to use a port pin as an output requires no initialization. You simply write to the port. For example, SETB P1.5 will send a logic 1 to P1 bit 5. MOV P0, A will send the data in the accumulator to P0.

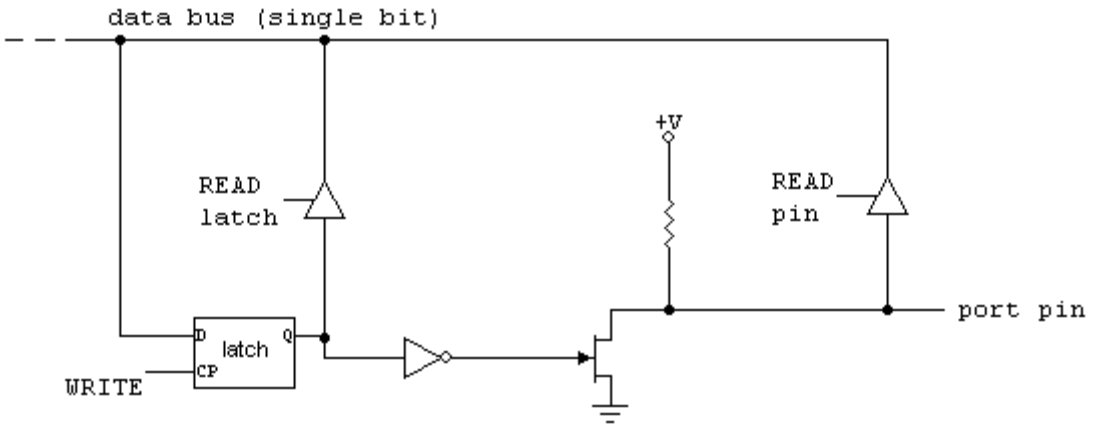
To initialise a port pin as an input ,we must first write a logic 1 to that pin. For example, to set P3.2 pin as an input we could write SETB P3.2.

To make the port P3 as input , the instruction MOV P3, #0FFH must be used. This instruction is exactly the same as to make all P3 pins are high.This is looks like a output of 1's at all pins of PORT3 . But if we are using this port as INPUT, we are using only input devices like keys. So the keys positions are sensed as input.

It is very important when initialising a port pin on the 8051, a comment is used. otherwise it is difficult to know whether data is being sent out to the port pin, or the port pin is being initialised for input. Taking the above example, the instruction should be written something like:
MOV P3, #0FFH ; initialising P3, all pins, as inputs

Why must a logic 1 be written to the port pin to make it an input pin?

To understand why writing a 1 to a port pin is necessary if the pin is to be used as an input, take a look at the schematic for an 8051 port pin.



8051 single port pin

However, if this pin is to be used as an input and there is a logic 0 on the latch, then the pin will always be at 0 because it is connected directly to ground through the switched on FET. No matter what voltage is applied to the port pin it will not rise above 0V.The pin value is always 0 level.

1.1.14 TIMER

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

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

However, this is difficult and imperfect. And there is another disadvantage is that CPU is occupied for the maintaining the delay. If we use the on-chip timers, the CPU could be doing something more useful work. The timers take care on timer work.

The Timers' SFRs

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

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

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

1.1.15 SERIAL PORT

The 8051 includes an on-chip serial port that can be programmed to operate in one of four different modes and at a range of frequencies. In serial communication the data is rate is known as the baud rate. The baud rate simply means the number of bits transmitted per second. In the serial port modes that allow variable baud rates, this baud rate is set by timer 1.

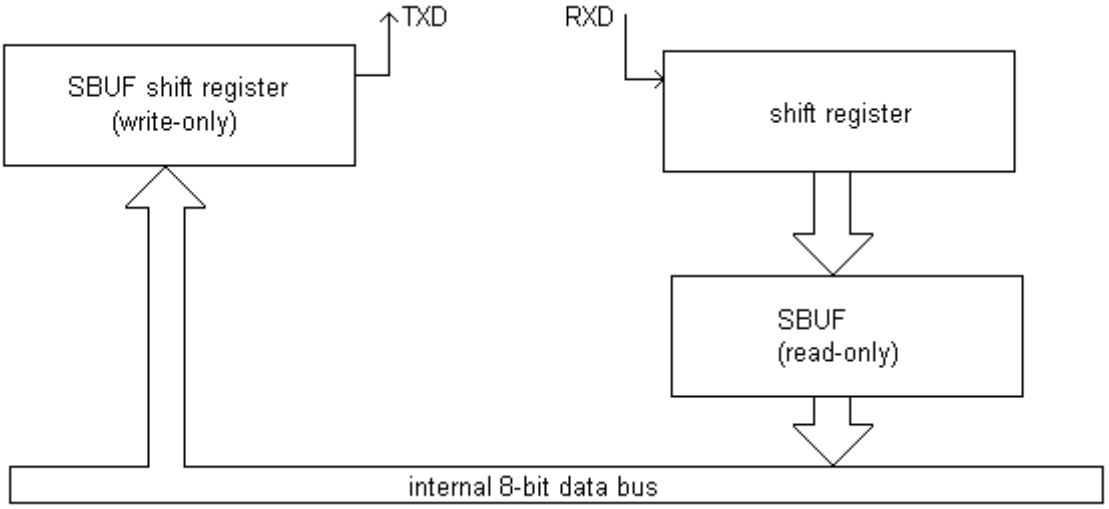


Fig 1.9 Diagram of Serial port

The 8051 serial port is full duplex. In other words, it can transmit and receive data at the same time. Shown in the Fig 1.9 Diagram of Serial port.

The block diagram above shows how this is achieved. If you look at the memory map you will notice at location 99H the serial buffer special function register (SBUF). Unlike any other register in the 8051, SBUF is in fact two distinct registers - the write-only register and the read-only register. Transmitted data is sent out from the write-only register while received data is stored in the read-only register. There are two separate data lines, one for transmission (TXD) and one for reception (RXD). Therefore, the serial port can be transmitting data down the TXD line while it is at the same time receiving data on the RXD line.

The TXD line is pin 11 of the microcontroller (P3.1) while the RXD line is on pin 10 (P3.0). Therefore, external access to the serial port is achieved by connecting to these pins. For example, if you wanted to connect a keyboard to the serial port you would connect the transmit line of the keyboard to pin 10 of the 8051. If you wanted to connect a display to the serial port you would connect the receive line of the display to pin 11 of the 8051. This is detailed in the Fig 1.10 Diagram of Serial Communication diagram below.

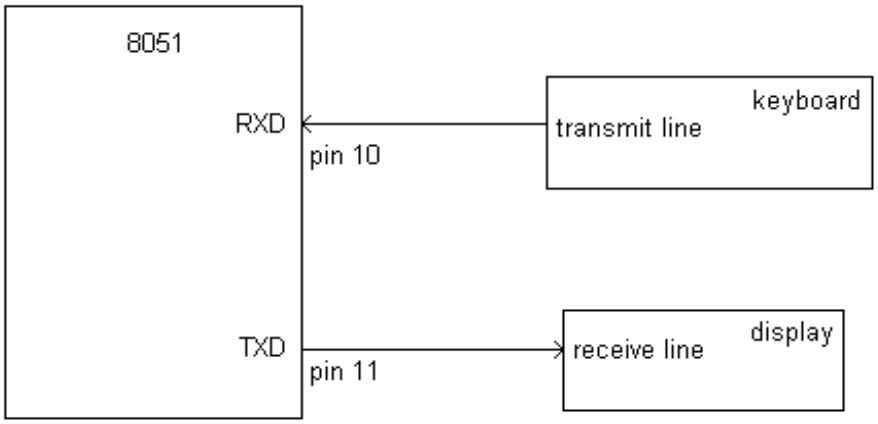


Fig 1.10 Diagram of Serial Communication

Transmitting and Receiving Data

Essentially, the job of the serial port is to change parallel data into serial data for transmission and to change received serial data into parallel data for use within the microcontroller.

- Serial transmission is changing parallel data to serial data.
- Serial reception is changing serial data into parallel data.
- Both are achieved through the use of shift registers.

1.1.16 INTERRUPTS

What is Interrupt

The interrupts refer to a notification, communicated to the controller, by a hardware device or software, on receipt of which controller momentarily stops and responds to the interrupt. Whenever an interrupt occurs the controller completes the execution of the current instruction and starts the execution of an **Interrupt Service Routine** (ISR) or Interrupt Handler. **ISR** is a piece of code that tells the processor or controller what to do when the interrupt occurs. After the execution of ISR, controller returns back to the instruction it has jumped from (before the interrupt was received).

The 8051 has five interrupt sources.

Interrupt Vectors

When an interrupt occurs the address of the interrupt service routine is loaded into the PC. This address is known as the interrupt vector.

The table below details the interrupt vectors for the 8051.

Interrupt	Flag	Vector Address
External interrupt 0	IE0	0003H
Timer 0	TF0	000BH
External interrupt 1	IE1	0013H
Timer 1	TF1	001BH
Serial port	RI or TI	0023H

A system reset is a special type of interrupt. It interrupts the running program and loads the PC with the vector address 0000H. This is the address the microcontroller begins with on power-up. Therefore, reset is similar to powering down and the powering up the system.

When an interrupt in the 8051 occurs, the vector address, as shown above, is loaded into the PC. For example, if timer 1 overflows (and interrupts are enabled; we'll talk more about enabling and disabling interrupts shortly) the PC is loaded with the value 001BH. The programmer must ensure the ISR for timer 1 is placed at this address.

1.1.17OSCILLATOR AND CLOCK

XTAL1 and XTAL2 pin: Although 8051 have on chip crystal oscillator. But still it requires an external clock oscillator. External crystal oscillator is connected to XTAL1 and XTAL2 pins. It also requires two capacitors of 30pF as shown in figure below. Capacitors one terminal is connected with crystal oscillator and other terminal with ground. Processing speed of 8051 microcontroller depends on crystal oscillator frequency. But each microcontroller have maximum limit of operating frequency. We cannot connect crystal oscillator more than maximum operating limit frequency. The Oscillator provides the clock to microcontroller.

If crystal frequency of 11.0592 Mhz is used, the clock pulse of width 1.09µs will be produced. The exact baud rate(without fraction) is attained when using this type of 11.0592Mhz (instead of 12Mhz) .

1.1.18 CLOCK CYCLE

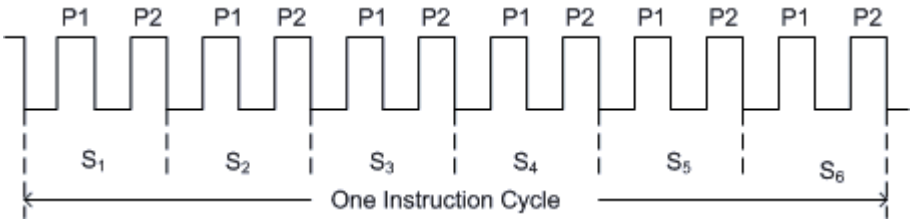


Fig 1.9 Clock Cycle Of 8051

In 8051, the crystal oscillator generates clock cycles. Depending upon the frequency of crystal the clock cycle time varies. The diagram shows 12 clock cycles. Machine cycle consists of twelve (12) clock cycles. Combination of one on- time and one off- time pulse is referred as cycle. One instruction cycle consists of one or more machine cycles .The instruction cycle is also called as Machine cycle. Shown in the Fig 1.9 Clock Cycle Of 8051**.**

1.1.19 STATE

T state-It is defined as the one subdivision of operation performed in one clock period.

In 8051, each instruction cycle has six states (S_1 - S_6). Each state has two pulses (P1 and P2).The instruction cycle is also referred as machine cycle. The state is defined as combination of two pulses . Shown in the Fig 1.10 Instruction cycle of 8051**.**

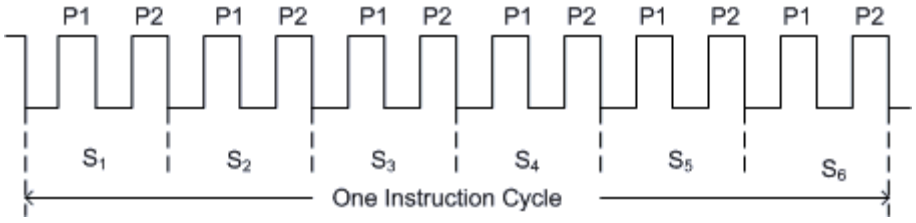


Fig 1.10 : Instruction cycle of 8051

1.1.20 Machine cycle for the 8051

Machine cycle is the unit of time elapsed during the execution. It is the unit used to describe the operation of 8051operation.The CPU takes a certain number of clock cycles to execute an instruction. In the 8051 family, these time elapsed are referred as machine cycles. The length of the machine cycle depends on the frequency of the crystal oscillator connected to the 8051 system. Usually 12 clock cycles are referred as machine cycles.

1.1.21 INSTRUCTION CYCLE

Instruction cycle is the time required to complete the execution of an instruction. The Instruction may take different machine cycles to complete execution. So the instruction cycle may be of one or more machine cycles.

1.1.22 RESET

Pin number 9 is a reset pin. It is active high pin. It is used to reset. If we apply active high signal to this pin, 8051 microcontroller will reset and turn off all its functions. It will erase all values of registers and it will make all program counter values to zero. It will not affect memory content.

1.1.23 POWER ON RESET

To reset the microcontroller, the reset pin must be high (+5v) for two machine cycle time. To ensure that , the power-on reset circuit (shown below) using capacitor and resistor is used. When power on, the 8051 must be reset. During power-on , the capacitor is charging. This maintains a high voltage across Resistor R and then gradually the RC circuit discharges to bring the reset pin to 0. This time period is necessary for getting reset.

To reset the microcontroller, the switch **s** is pressed. Shown in Figure 1.11 Power on reset circuit below shows the power on reset circuitry.

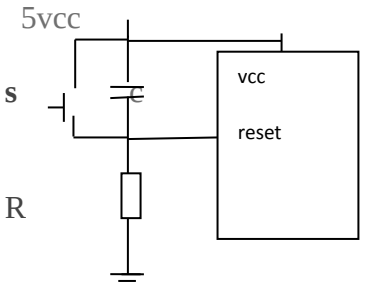


Fig 1.11 Power on reset circuit

1.1.24 COMPARISON OF 8051 FAMILY

In 1981, Intel Corporation introduced an 8-bit microcontroller called the 8051. This microcontroller had 128 bytes of RAM, 4K bytes of on-chip ROM, two timers, one serial port, and four ports (each 8-bits wide) all on a single chip. At the time it was also referred to as a "system on a chip". The 8051 is an 8-bit processor, meaning that the CPU can work on only 8 bits of data at a time. Data larger than 8 bits has to be broken into 8-bit pieces to be processed by the CPU. The 8051 has a total of four I/O ports, each 8 bits wide. The 8051 became widely popular after Intel allowed other manufacturers to make and market any flavors of the 8051 they please with the condition that they remain code-compatible with the 8051. This has led to many versions of the 8051 with different speeds and amounts of on-chip ROM marketed by more than half a dozen manufacturers. This means that if you write your program for one, it will run on any of them regardless of the manufacturer.

BYTES

MICROCONTROLLER	**RAM** IN BYTES	**ROM**	**TIMER**	**INTERRUPT**	**IO PORTS**	**SERIAL PORTS**
8051	128	4K	2	5	4	1
8052	256	8K	3	7	4	1
8031	128	NIL	2	5	4	1
89C51	128	4K FLASH	3	5	4	1

1.2 INSTRUCTION SET OF 8051

Data Transfer Instructions
Arithmetic instructions
Logical Instructions
Branch Instructions
Bit manipulation instructions

1.2.1 DATA TRANSFER INSTRUCTIONS

Data transfer instructions move the content of one register to another. The register the content of which is moved remains unchanged. If they have the suffix "X" (MOVX), the data is exchanged with external memory.

Mnemonic	**Description**	**Byte**	**Cycle**
MOV A,Rn	Moves the content of register to the accumulator	1	1

MOV A,direct	Moves the content of direct address to the accumulator	2	2
MOV A,@Ri	Moves the content of address in Ri to the accumulator	1	2
MOV A,#data	Moves the immediate data to the accumulator	2	2
MOV Rn,A	Moves the content of accumulator to the register	1	2
MOV Rn,direct	Moves the content of direct address to the register	2	4
MOV Rn,#data	Moves the immediate data to the register	2	2
MOV direct,A	Moves the content of accumulator to the direct adderss	2	3
MOV @Ri,A	Moves the accumulator to the indirect RAM	1	3
MOV @Ri,direct	Moves the direct byte to the indirect RAM	2	5
MOV @Ri,#data	Moves the immediate data to the indirect RAM	2	3
MOV DPTR,#data	Moves a 16-bit data to the data pointer	3	3
MOVC A,@A+DPTR	Moves the code byte relative to the DPTR to the accumulator (address=A+DPTR)	1	3
MOVX A,@Ri	Moves the external RAM (8-bit address) to the accumulator	1	3-10
MOVX A,@DPTR	Moves the content of external RAM (16-bit address) to the accumulator	1	3-10
MOVX @Ri,A	Moves the content of accumulator to the external RAM (8-bit address)	1	4-11

MOVX @DPTR,A	Moves the content of accumulator to the external RAM (16-bit address)	1	4-11
PUSH direct	Pushes the direct byte onto the stack	2	4
POP direct	Pops the direct byte from the stack/td>	2	3
XCH A,Rn	Exchanges the register with the accumulator	1	2
XCH A,direct	Exchanges the direct byte with the accumulator	2	3
XCH A,@Ri	Exchanges the indirect RAM with the accumulator	1	3
XCHD A,@Ri	Exchanges the low-order nibble indirect RAM with the accumulator	1	12

1.2.2 Arithmetic instructions

Arithmetic instructions perform several basic operations such as addition, subtraction, division, multiplication etc. After execution, the result is stored in the first operand. For example:

`ADD A,R1` – The result of addition (A+R1) will be stored in the accumulator.

ARITHMETIC INSTRUCTIONS

Mnemonic	Description	Byte	Cycle
ADD A,Rn	Adds the register to the accumulator	1	1
ADD A,direct	Adds the direct byte to the accumulator	2	2
ADD A,@Ri	Adds the indirect RAM to the accumulator	1	2
ADD A,#data	Adds the immediate data to the accumulator	2	2
ADDC A,Rn	Adds the register to the accumulator with a carry flag	1	1

ADDC A,direct	Adds the direct byte to the accumulator with a carry flag	2	2
ADDC A,@Ri	Adds the indirect RAM to the accumulator with a carry flag	1	2
ADDC A,#data	Adds the immediate data to the accumulator with a carry flag	2	2
SUBB A,Rn	Subtracts the register from the accumulator with a borrow	1	1
SUBB A,direct	Subtracts the direct byte from the accumulator with a borrow	2	2
SUBB A,@Ri	Subtracts the indirect RAM from the accumulator with a borrow	1	2
SUBB A,#data	Subtracts the immediate data from the accumulator with a borrow	2	2
INC A	Increments the accumulator by 1	1	1
INC Rn	Increments the register by 1	1	2
INC Rx	Increments the direct byte by 1	2	3
INC @Ri	Increments the indirect RAM by 1	1	3
DEC A	Decrements the accumulator by 1	1	1
DEC Rn	Decrements the register by 1	1	1
DEC Rx	Decrements the direct byte by 1	1	2
DEC @Ri	Decrements the indirect RAM by 1	2	3
INC DPTR	Increments the Data Pointer by 1	1	3
MUL AB	Multiplies A and B	1	5
DIV AB	Divides A by B	1	5

DA A	Decimal adjustment of the accumulator according to BCD code	1	1

1.2.3 Logical Instructions

Logical instructions perform logic operations upon corresponding bits of two registers. After execution, the result is stored in the first operand.

LOGICAL INSTRUCTIONS

Mnemonic	Description	Byte	Cycle
ANL A,Rn	AND register to accumulator	1	1
ANL A,direct	AND direct byte to accumulator	2	2
ANL A,@Ri	AND indirect RAM to accumulator	1	2
ANL A,#data	AND immediate data to accumulator	2	2
ANL direct,A	AND accumulator to direct byte	2	3
ANL direct,#data	AND immediate data to direct register	3	4
ORL A,Rn	OR register to accumulator	1	1
ORL A,direct	OR direct byte to accumulator	2	2
ORL A,@Ri	OR indirect RAM to accumulator	1	2
ORL direct,A	OR accumulator to direct byte	2	3
ORL direct,#data	OR immediate data to direct byte	3	4
XRL A,Rn	Exclusive OR register to accumulator	1	1
XRL A,direct	Exclusive OR direct byte to accumulator	2	2

XRL A,@Ri	Exclusive OR indirect RAM to accumulator	1	2
XRL A,#data	Exclusive OR immediate data to accumulator	2	2
XRL direct,A	Exclusive OR accumulator to direct byte	2	3
XORL direct,#data	Exclusive OR immediate data to direct byte	3	4
CLR A	Clears the accumulator	1	1
CPL A	Complements the accumulator (1=0, 0=1)	1	1
SWAP A	Swaps nibbles within the accumulator	1	1
RL A	Rotates bits in the accumulator left	1	1
RLC A	Rotates bits in the accumulator left through carry	1	1
RR A	Rotates bits in the accumulator right	1	1
RRC A	Rotates bits in the accumulator right through carry	1	1

1.2.4 Branch Instructions

There are two kinds of branch instructions:

Unconditional jump instructions: upon their execution a jump to a new location from where the program continues execution is executed. Conditional jump instructions: a jump to a new program location is executed only if a specified condition is met. Otherwise, the program normally proceeds with the next instruction.

BRANCH INSTRUCTIONS

Mnemonic	Description	Byte	Cycle
ACALL addr11	Absolute subroutine call	2	6

LCALL addr16	Long subroutine call	3	6
RET	Returns from subroutine	1	4
RETI	Returns from interrupt subroutine	1	4
AJMP addr11	Absolute jump	2	3
LJMP addr16	Long jump	3	4
SJMP rel	Short jump (from –128 to +127 locations relative to the following instruction)	2	3
JC rel	Jump if carry flag is set. Short jump.	2	3
JNC rel	Jump if carry flag is not set. Short jump.	2	3
JB bit,rel	Jump if direct bit is set. Short jump.	3	4
JBC bit,rel	Jump if direct bit is set and clears bit. Short jump.	3	4
JMP @A+DPTR	Jump indirect relative to the DPTR	1	2
JZ rel	Jump if the accumulator is zero. Short jump.	2	3
JNZ rel	Jump if the accumulator is not zero. Short jump.	2	3
CJNE A,direct,rel	Compares direct byte to the accumulator and jumps if not equal. Short jump.	3	4
CJNE A,#data,rel	Compares immediate data to the accumulator and jumps if not equal. Short jump.	3	4
CJNE Rn,#data,rel	Compares immediate data to the register and jumps if not equal. Short jump.	3	4
CJNE @Ri,#data,rel	Compares immediate data to indirect register and jumps if not equal. Short jump.	3	4

DJNZ Rn,rel	Decrements register and jumps if not 0. Short jump.	2	3
DJNZ Rx,rel	Decrements direct byte and jump if not 0. Short jump.	3	4
NOP	No operation		

1.2.5 Bit Manipulation Instructions

Similar to logic instructions, bit manipulation instructions perform logic operations.

The difference is that these are performed upon single bits.

BIT MANIPULATION INSTRUCTIONS

Mnemonic	Description	Byte	Cycle
CLR C	Clears the carry flag	1	1
CLR bit	Clears the direct bit	2	3
SETB C	Sets the carry flag	1	1
SETB bit	Sets the direct bit	2	3
CPL C	Complements the carry flag	1	1
CPL bit	Complements the direct bit	2	3
ANL C,bit	AND direct bit to the carry flag	2	2
ANL C,/bit	AND complements of direct bit to the carry flag	2	2
ORL C,bit	OR direct bit to the carry flag	2	2
ORL C,/bit	OR complements of direct bit to the carry flag	2	2
MOV C,bit	Moves the direct bit to the carry flag	2	2
MOV bit,C	Moves the carry flag to the direct bit	2	3

Review questions

Part A

1.How many IO ports are available in 8051? Mention the functions of port3.
2.Explain DPTR.
3.What is the purpose of PSW register?
4.What is Micro Controller?
5.What is use of PSEN signal?
6.Explain JMP@A+DPTR instruction.
7.How many register banks are placed in internal RAM?
8.What is the purpose of SERs?
9.Mention the three difference of internal RAM, external RAM.
10.Mention the three difference address space of 8051.

PART B

1. Compare micro processor and microcontroller
2.Explain PSW Register with diagram
3.What is program counter?
4.Define Instruction cycle.
5.Define Machine Cycle.
6.What is the use of carry flags in 8051?
7.How many instructions used in 8051 & explain any one.
8.What is ALU?.

PART C

1. Draw the block diagram of 8051 .Explain the functions of each block?
2.Explain with diagram how a port is changed as input port?
3.Draw & explain the pin diagram of 8051.
4.Explain with diagram of memory organization of 8051.
5.Explain the arithmetic instructions used in 8051.
6.Explain the external data memory & external program memory.
7.Explain the bit manipulation instructions used in 8051.

UNIT - II
ASSEMBLER AND ADDRESSING MODES

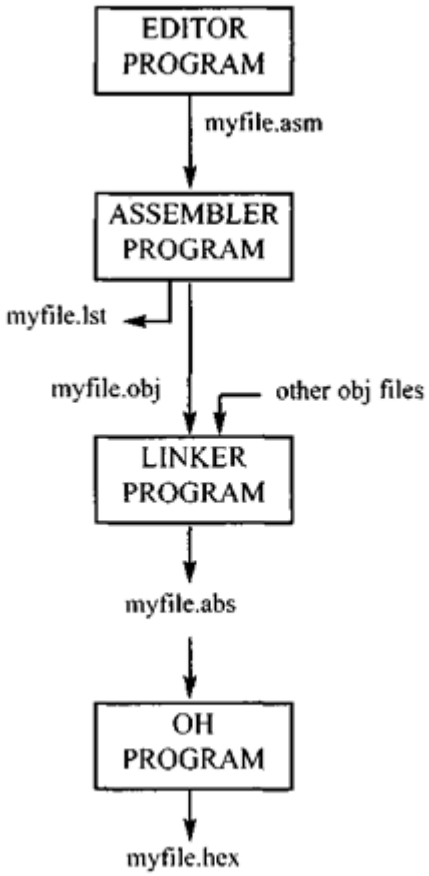
2.1 ASSEMBLING AND RUNNING AN 8051 PROGRAM

Now that the basic form of an Assembly language program has been given, the next question is: How it is created, assembled, and made ready to run?

The steps to create an executable Assembly language program are outlined as follows.

1. First we use an editor to type in a program. Many excellent editors or word processors are available that can be used to create and/or edit the program.
2. A widely used editor is the MS-DOS EDIT program (or Notepad in Windows), which comes with all Microsoft operating systems. Notice that the editor must be able to produce an ASCII file. For many assemblers, the file names follow the usual DOS conventions, but the source file has the extension "asm". The "asm" extension for the source file is used by an assembler in the next step. The "asm" source file containing the program code created in step 1 is fed to an 8051 assembler. The assembler converts the instructions into machine code. The assembler extension for list file is'lst'
3. Assemblers require a third step called linking. The link program takes one or more object files and produces an absolute object file with the extension "abs" This abs file is used by 8051 trainers that have a monitor program.

4. Next, the "abs" file is fed into a program called "OH" (object to hex convert er), which creates a file with extension "hex" that is ready to burn into ROM. This program comes with all 8051 assemblers. Recent Windows-based assemblers
   combine steps 2 through 4 into one step



2.2 STRUCTURE OF ASSEMBLY LANGUAGE PROGRAM

An instruction may be represented on a line of maximum 128 characters, the general form being:

[<label>:] [<opcod>[<operatives>][;<comments>]]

where:

<label> is a name, maximum 31 characters (letters, numbers or special characters _,?,@,..), the first character being a letter or one of the special characters. Each label has a value attached and also a relative address in the segment where it belongs to.

<opcode>the mnemonic of the instruction.

<operatives> the operative (or operatives) associated with the instruction concordant to the syntax required for the instruction. It may be a constant, a symbol or expressions containing these.

<comments> a certain text forego of the character ";" .comment is optional. It is for readability

2.3 ASSEMBLER DIRECTIVES

It is a pseudo instruction.
It is not an executable .
It gives directions to Assembler

Some examples of Assembler Directives

ORG,END, EQU,DB,DW, DATA

ORG: It gives direction to Assembler that the program should be started at the Address following ORG.

Ex. ORG 8000h

END : It give direction to Assembler that the program ends at that point

Ex. END

EQU : It is used to give direction to Assembler to assign some value to some variable.

Ex. EQU PI 3.14

DB : Define Byte

It directs the Assembler that the number following this DB is byte

Ex. DB 39h
DB 00110101b

DATA: It gives direction to Assembler that the numbers following DATA are the data

Ex. DATA 32,43,65,23,01

DIFFERENT ADDRESSING MODES

The method of specifying the data in instruction is called Addressing

Types of Addressing Modes

1.Register Addressing
2.Indirect Addressing
3.Direct Addressing
4.Immediate Addressing
5.Index Addressing

1.Register Addressing

In this Addressing mode, the registers(R0…R7,A,B,DPTR,CARRY) are used as operands .R0…R7 can be selected in any one of four modes. The modes can be selected in PSW register.

Ex.

```
MOV A,R1
```

In this mode, the content of R1 is moved to A

```
ADD A,R3
```

In this mode, the content of R3 is added with the content of A

ANL A,R1

In this mode, the content of R1 is AND immediate with the content of A

2. Indirect Addressing

In this Addressing mode, the operand's address is specified in register R0,R1or DPTR To access the address register, the symbol '@' is preceded with above register.

Both internal and external RAM can be accessed in this mode. 8 bit address is specified in R0 an R1 registers. 16 bit address is specified in DPTR

Ex.

MOV A,@R1

In this mode, the content of address specified in R1 is moved to A

ADD A,@R0

In this mode, the content of address specified in R0 is added with the content of A

MOVX A,@DPTR

In this mode, the content of external memory address specified in DPTR is moved to A. 'X' in the MOVX represent External memory.

3.Direct Addressing

In this mode of addressing, 8 bit address of operand is specified in instruction. Address of internal RAM is specified.

Ex.

MOV A,34h

In this mode, the content of address 34h is moved to A

ADD A,23h

In this mode, the content of address 23h is added with the content of A

ANL A,45h

In this mode, the content of address 45h is AND immediate with the content of A

4.Immediate Addressing

In this addressing mode , the operand is data. The actual data is specified in the instruction itself. The symbol '#' is proceeded with data. The data is accessed immediately.

Ex.

MOV A , #67h

In this mode, the hexadecimal data 67 is moved to A

ADD A,#89h

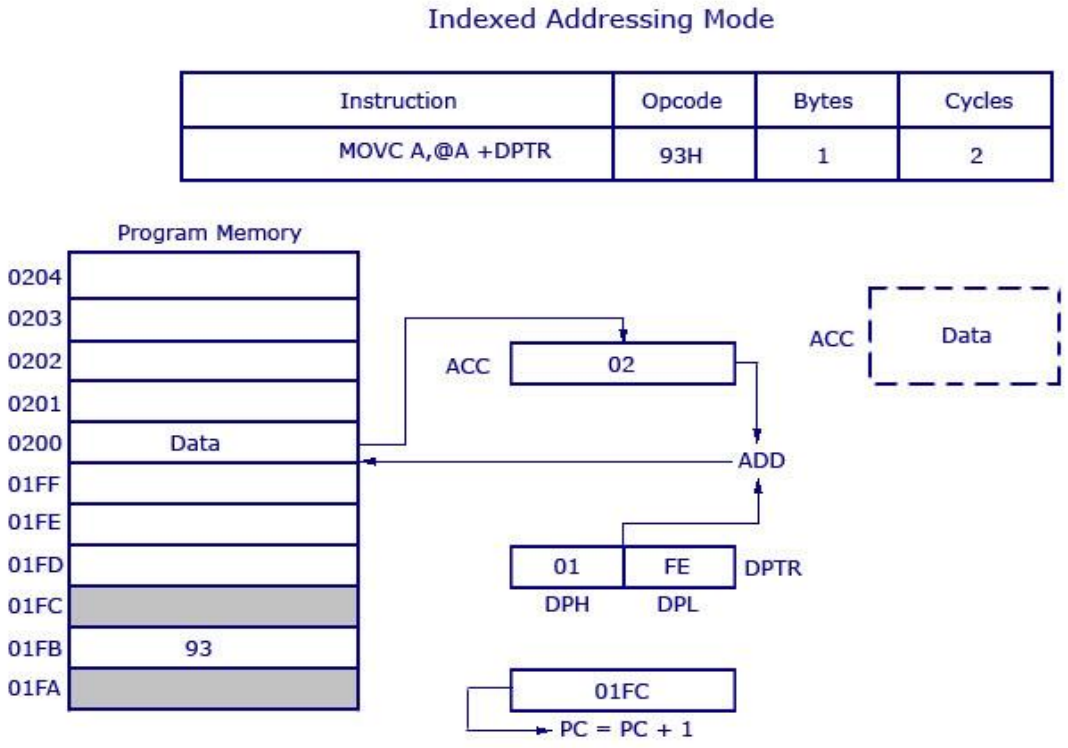
In this mode, the hexadecimal data 89 is added with the content of A

ANL A,#91

In this mode the hexadecimal data 91 is AND immediate with the content of A

5. Index Addressing

This addressing mode is used access lookup table in program memory.
Ex.
MOVC A, @A+DPTR



2.2 PROGRAMS

2.2.1 MULTIBYTE ADDITION

Multi byte numbers like 8254h and 65f3h may be added.

i) Add f3 and 54 using ADD and store the result in memory location
ii) Add 82 and 65 using ADDC and store the result in another location

```
MOV DPTR ,#8200H        ;memory address is loaded to store result
MOV  R1,#00H            ;R1 is initialized .It  is used to store carry
MOV A,#F3H              :LSB of second data is moved to A
ADD A,#54H              ;LSB of first data is added with A
MOVX @DPTR,A            ;added value in A is moved to memory
```

```
        INC DPTR            ;DPTR is incremented
        MOV A,#65H          ;MSB of second data is moved to A
        ADDC A,#82H         MSB of first data is added with carry in A
        MOVX @DPTR,A        ;added value is moved to memory
        INC DPTR            ;DPTR is incremented  to store 1 if carry
        JNC NEXT            ;if carry, jump to labeled NEXT
        INC R1              ;R1 is incremented if carry available
NEXT:   MOV A,R1            ;R1 value is moved to A
        MOVX @DPTR,A        ;value in A is moved to memory
HLT:    SJMP  HLT
```

2.2.2 8 BIT MULTIPLICATION

Assume that 8 bit data are available in memory address 8400h and 8401h
And the result is to be stored in 8402h,8403h

```
        MOV DPTR,#8400H     ;DPTR is initialized
        MOVX A,@DPTR        ;data in memory address 8400 is moved to A
        MOV B,A             ;value in A is moved to B
        INC DPTR            ;DPTR is incremented
        MOVX A,@DPTR        ;next data in  address 8401 is moved to A
        MUL AB              ;both data are multiplied
        INC DPTR            ;DPTR is incremented
        MOVX @DPTR,A        ;low byte answer is moved to memory
        MOV A,B             ;high byte answer in B is moved to A
        INC DPTR            ;DPTR incremented
        MOVX @DPTR,A        ;high answer in A is moved to memory
HLT:    SJMP HLT
```

2.2.3 8 bit DIVISION

Assume that 8 bit data(denominator) is available in memory address 8400h and data(Numerator) in 8401h
And the quotient is to be stored in 8402h and remainder is to be stored in 8403h

```
MOV DPTR,#8400H  ;DPTR is initialized
MOVX A,@DPTR     ;data(divisor) in memory address 8400 is moved toA
MOV B,A          ;value in A is moved to B
INC DPTR         ;DPTR is incremented
MOVX A,@DPTR     ;next data(dividend) in  address 8401 is moved to A
DIV AB           ; data in A is divided by data in B
INC DPTR         ;DPTR is incremented
MOVX @DPTR,A     ;answer quotient is moved to memory
```

```
                MOV A,B          ; answer  remainder in B is moved to A
                INC DPTR         ;DPTR incremented
                MOVX @DPTR,A     ; answer remainder in A is moved to memory
HLT:            SJMP HLT
```

2.2.4 BIGGEST NUMBER

Assume that the data to be arranged are available in array which starts from 8401h and the array length is available in 8400h.

The result is to be stored in 8500h

```
                MOV B, #00H              ;to hold the biggest number, B is initialised
                MOV  DPTR ,#8400H        ;DPTR initialized with8400
                MOVX A, @DPTR            ;array length in 8400 in moved to A
                MOV R0,A                 ;value (array length) in A is copied into R0
AGAIN:          INC DPTR                 ;DPTR incremented
                MOVX A, @DPTR            ;data in memory is moved to A
                CJNE A,B , NEXT          ;A and B compared. Carry will be generated if
                                          B is bigger
 NEXT:          JC  L1                   ; if carry ,jump to label L1
                MOV B,A                  ;if no carry, move the value in A to B
   L1:          DJNZ  R0,  AGAIN         ;array length is decremented and jump to
                                          label AGAIN till array length is 0
                MOV DPTR,#8500H          ;DPTR is initialized with 8500 to store bigger
                                          value in B
                MOV A,B                  ;value in B (bigger value) is moved to A
                MOVX @DPTR,A             ;this value (bigger value) is moved to
                                          memory addressed by DPTR(8500)
HLT:            SJMP   HLT
```

2.2.5 SMALLEST NUMBER

Assume that the data are available in array which starts from 8401h and the array length is available in 8400h.The result is to be stored in 8500h

```
                MOV B, #00H              ;to hold the biggest number, B is initialised
                MOV  DPTR ,#8400H        ;DPTR initialized with8400
                MOVX A, @DPTR            ;array length in 8400 in moved to A
                MOV R0,A                 ;value(array length) in A is copied into R0
AGAIN:          INC DPTR                 ;DPTR incremented
                MOVX A, @DPTR            ;data in memory is moved to A
                CJNE A,B , NEXT          ;A and B compared. Carry will be generated if
                                          B is bigger
 NEXT:          JC  L1                   ; if carry ,jump to label L1
                MOV B,A                  ;if no carry, move the value in A to B

   L1:          DJNZ  R0,  AGAIN         ;array length is decremented and jump to
                                          label AGAIN till array length is 0
                MOV DPTR,#8500H          ;DPTR is initialized with 8500 to store bigger
                                          value in B
```

```
        MOV A,B          ;value in B (bigger value) is moved to A
        MOVX @DPTR,A     ;this value (bigger value) is moved to
                          memory addressed by DPTR(8500)
HLT:    SJMP  HLT
```

2.2.6 ASCENDING ORDER

Assume that the data to be arranged are available in array which starts from 8401h and the array length is assumed as 09 .

```
        MOV R0, #08H        ; array length 08h(09-01) is stored
AGAIN:  MOV  A, R0          ;08h is moved to R1
        MOV R1, A           ;Data moved to R1
        MOV DPTR, #8401H    ;DPTR is initialized with 8401h
BACK:   PUSH DPH            ;84 is saved in stack
        PUSH DPL            ;01 is saved in stack
        MOVX A, @DPTR       ;first data is moved to A
        MOV B, A            ;this data is copied into B
        INC DPTR            ;DPTR incremented
        MOVX A, @DPTR       ;second data is moved to A
        CJNE A,B, LOOP      ;first data in B and next data in A are  compared. Carry will
                             generate if B value is bigger
LOOP:   JNC NEXT            ; if no carry(second data is bigger)instruction labeled NEXT
                             will be executed
        POP DPL             ;01 from stack is moved to DPL
        POP DPH             ;84 from stack is moved to DPH
        MOVX @DPTR, A       ;second data is moved is moved to first location
        INC DPTR            ;DPTR incremented
        MOV A ,B            ;first data in B is moved to A
        MOVX @DPTR,A        ;this first data is moved to second location
NEXT:   DJNZ R1, BACK       ;jump for next two data comparison
        DJNZ R0,  AGAIN     ;jump for next scan
HLT:    SJMP HLT            ;stay at here
```

2.2.7 DESCENDING ORDER

Assume that the data to be arranged are available in array which starts from 8401h and the array length is assumed as 09 .

```
        MOV R0, #08H        ; array length 08h(09-01) is stored
AGAIN:  MOV  A, R0          ;08h is moved to R1
        MOV R1, A           ;Data moved to R1
        MOV DPTR, #8401H    ;DPTR is initialized with 8401h
BACK:   PUSH DPH            ;84 is saved in stack
        PUSH DPL            ;01 is saved in stack
        MOVX A, @DPTR       ;first data is moved to A
        MOV B, A            ;this data is copied into B
        INC DPTR            ;DPTR incremented
        MOVX A, @DPTR       ;second data is moved to A
        CJNE A,B, LOOP      ;first data in B and next data in A are  compared. Carry will
                             generate if B value is bigger
```

```
LOOP:   JC NEXT          ; if carry(second data is smaller)instruction labeled NEXT
                           will be executed
        POP DPL          ;01 from stack is moved to DPL
        POP DPH          ;84 from stack is moved to DPH
        MOVX @DPTR, A    ;second data is moved is moved to first location
        INC DPTR         ;DPTR incremented
        MOV A,B          ;first data in B is moved to A
        MOVX @DPTR,A     ;this first data is moved to second location
NEXT:   DJNZ R1, BACK    ;jump for next two data comparison
        DJNZ R0, AGAIN   ;jump for next scan
HLT:    SJMP HLT         ;stay at here
```

2.2.8.BCD TO ASCII CONVERSION

To covert BCD number, each digit is separately considered and equivalent ASCII value is generated. Ex. To convert 65 ,5 is converted into ASCII value 35 and 6 is converted into ASCII 36.The

Assume that the BCD value 65 is stored in memory location 8400h and ASCII values are stored in 8401 ,8402.

```
MOV  DPTR,#8400H ; DPTR is initialized with 8400h
        MOV XA, @DPTR    ; First Data Is Moved To A
        MOV R1,A         ; Data moved to R1
        ANL  A,#0FH      ; Get the Lower data
        ORL A,#30H       ; OR logic for 30h
        INC DPTR         ; DPTR incremented
        MOVX @DPTR,A     ; Second Data Is Moved To A
        INC DPTR         ; DPTR incremented
        MOV A,R1         ;Data moved to R1
        ANL A,#F0H       ; Get the Upper data
        SWAP A           ; Move the lower value
        ORL A,#30H       ;OR logic for 30h
        MOVX @DPTR,A     ; Store the Result
HLT:    SJMP  HLT        ; stay at here
```

2.2.9 ASCII TO BINARY CONVERSION

Assume that ASCII value is stored in 8400h and answer to be stored in 8401h

```
        MOV DPTR, #8400H ; DPTR is initialized with 8400h
        MOV A, @DPTR     ; First Data Is Moved To A
        MOV R1, A        ; Data moved to R1
        CJNE A,#40H, NEXT; first data in 40h and next data in A are  compared. Carry
                           will  generate if B value is bigger
NEXT:    JC  LOOP        ; if carry(second data is smaller)instruction labeled NEXT
                           will be executed
        CLR  C           ; Clear carry flag
        SUBB A, #07H     ; Subtract the value Data from  07 h
LOOP:   CLR C            ; Clear carry flag
        SUBB A, #30H     ; Subtract the value Data from  30 h
        INC DPTR         ;DPTR incremented
        MOVX  @DPTR ,A   ;Store the Result
HLT:    SJMP  HLT        ;stay at here
```

2.2.10 ODD PARITY GENERATOR

Odd parity means that the total number of 1's in data as well as in parity bit is ODD.

Assume that the data 42H for which the parity is to be generated and the result is to be Stored in 8400h.

The data 42h has 2 nos. of '1' s . So one bit is generated to make ODD parity.

```
        MOV DPTR, #8400H      ; memory address is loaded in DPTR
        MOV R1,#08            ;to rotate the data 8times,R1 is initialised
        MOV R2,#00            ;counter initialized to count no.of '1's
        MOV A,#42h            ;the data 42h is stored in A
BACK:   RRC A                 ;the data is rotated
        JNC   XXX             ;if not carry, jump to label XXX
        INC R2                ;if carry, R2 is incremented
XXX:    DJNZ R1, BACK         ;8 rotation is checked
        MOV A,R2              ;no. of 1 s in R2 is moved to A
        MOV B,#02             ;to test the even ,that should be
        DIV AB                ; divide by 02
        MOV A,B               ; the remainder(it is 0 for even no of 1's in data) is
                               Move to A
        CPL  A                ; A value is complemented
        MOVX  @DPTR,A,        ; Complemented value is stored in memory
HLT:    SJMP   HLT
```

2.2.**11 EVEN PARITY GENERATOR**

Even parity means that the total number of 1's in data as well as in parity bit is even. Assume that the data 42H for which the parity is to be generated and the result is to be Stored in 8400h.The data 42h has 2 nos. of '1' s. So NO need of one bit is generated to make even parity.

```
        MOV DPTR, #8400H      ; memory address is loaded in DPTR
        MOV R1,#08            ;to rotate the data 8times,R1 is initialised
        MOV R2,#00            ;counter initialized to count no of '1's
        MOV A,#42h            ;the data 42h is stored in A
BACK:   RRC A                 ;the data is rotated
        JNC   XXX             ;if not carry, jump to label XXX
        INC R2                ;if carry, R2 is incremented
XXX:    DJNZ R1, BACK         ;8 rotation is checked
        MOV A,R2              ;no. of 1 s in R2 is moved to A
        MOV B,#02             ;to test the even ,that should be
        DIV AB                ; divide by 02
        MOV A,B               ; the remainder(it is 0 for even no.of 1's in data) is
                               moved  to A
        MOVX  @DPTR,A,        ;A value is stored in memory
HLT:    SJMP   HLT            ;stay at here
```

2.2.**12 TIME DELAY ROUTINE**

To hold the output or operation of controller, the controller must be forced into delay Routine. This delay may be increased by increasing number of loops.

Delay Routine using single loop

```
        MOV  R1,#FFH      ;value FF is loaded in to R1
WAIT:   DJNZ  R1, WAIT    ;R1 is decremented  till reach to zero
 HLT:   SJMP HLT          ;stay at here
```

Delay Routine using double loop

```
        MOV  R1,#FFH      ;R1 is initialized  with FF
LOOP:   MOV R2, #FEH      ;R2 is initialized with FE
WAIT:   DJNZ  R2, WAIT    ;R2 is decremented till reach to zero
        DJNZ  R1, LOOP    ;once R2 reach  zero, R1 is decremented till to zero
                            and again R2   initialised  with FE
 HLT:   SJMP HLT          ;stay at here
```

Review Questions

PART A

1.What is assembler?
2.What is operand field?
3.What are the addressing modes in 8051.
4.Mention the two assembler directives.
5.What are the instructions used to access external RAM.
6.Mention any two examples of direct addressing instructions.

PART B

1.What is Assembler?Mention the types.
2.Explain indirect addressing with example?
3.Explain assembler directives?
4.Explain the structure of assembly language.
5.Explain the operation of MOVC A,@A+DPTR

PART C

1.Explain different Addressing modes in 8051?
2.Write an Assembly language program to find the biggest of numbers in an array.
3.Write an ALP for finding maximum value in array.
4.Write an ALP for Multi byte Addition.
5.Write an ALP for Ascending order.

UNIT - III

I/O AND TIMER

3.1 I/O

Bit addresses for I/O

In the 8051 microcontroller there are a total of four ports for I/O Operations . Totally 32 pins are aside for the four ports P0,P1,P2and P3,where each port takes 8Pins.All ports are bidirectional.

Port 0 contains a total of 8 pins (32 to 39) can be used for input or output or both input and output ports, as a bidirectional lower order address and data bus for external memory.

Port 1 contains a total of 8 pins (1 to 8) can be used as input or outputs. Its not done the dual functions. so its out latch is connected to the external pins through only the output driver.

Port 2 contains a total of 8 pins (21 to 28) can be used as input or output port. Higher order port2 must be used along with Po to provide the 16bit address for external memory.Port2 also designated as A8-A15,indicating its dual function.

Port 3 contains a total of 8 pins (pin 10 to 17) can be used as input or output. Port 3 has the additional function of providing some extremely important signals such as interrupts.

When the Port Pin may be used as an input, a 1 must be sent to the Port, it must be Programmed, which makes the pin to float in a high impedance state, and the pin is essentially connected to the input buffer.

When Port is used as an output, a 0 is written to a Port, the pin latches that are Programmed are transferred to its output pin.

Table 3.1 Provides the alternate functions of P3.

P3 Bit	Function	Pin
P3.0	RXD	10
P3.1	TXD	11
P3.2	$\overline{\text{INT0}}$	12
P3.3	$\overline{\text{INT1}}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17

All four Ports of microcontroller 8051, are bit and byte addressable. Table 3.2 provides the bit and byte addresses of all ports.

Name	Function	Byte address	Bit addresses MSB - LSB
P0	Input/ Output port 0 latch	80H	87H- 80H
P1	Input/ Output port 1 latch	90H	97H -90H
P2	Input/ Output port 2 latch	A0H	A7H – A0H
P3	Input/ Output port 3 latch	0B0H	B7H – B0H

Table 3.2 8051 PORTS ADDRESS (Byte and Bit)

Bit addresses for RAM

The 128-byte internal RAM of the 8051 is accessed in to three distinct areas.

i)32 bytes from 00H to 1FH can be accessed in to 32 registers contains as four banks (0-3).Each bank have 8 registers named R0 to R7.

ii)16 bytes of RAM locations from 20H to 2FH form a total of 128 addressable bit theory are addressed as 00H to 7FH. Since 16x8=128(00-7FH).

iii)Only byte addressable area totally 80 bytes of RAM from 30H to 7FH. These general purpose RAM area is called Scratch Pad. From fig 3.1 ,we see the structure of internal RAM for Both bit and Byte accessible.

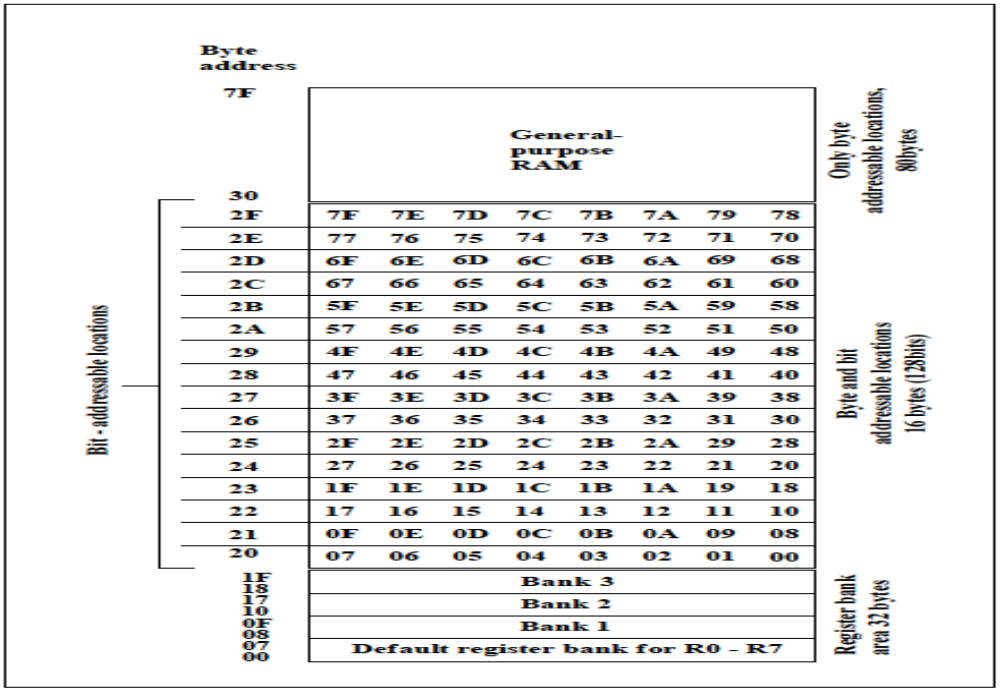


Fig. 3.1 Structure of Internal RAM (Both bit and byte accessible)

Single bit instructions use only one addressing mode and that is direct addressing mode. There is no indirect addressing mode for single –bit instruction.

In order to access these 128 bits of RAM locations and other bit-addressable space of 8051 individually, we can use only single bit instructions. Table 3.3 Provides a list of single bit instruction

Instruction	Function
SETB bit	Set the bit (bit=1)
CLR bit	Clear the bit (bit=0)
CPL bit	Complement the bit (bit=NOT bit)
JB bit, target	Jump to target if bit=1(jump if bit)
JNB bit, target	Jump to target if bit=0(jump if no bit)
JBC bit, target	Jump to target if bit=1,clear bit(jump if bit, then clear)

Table 3.3 Single Bit Instructions

I/O Programming

In the 8051,there are a total of four Ports for I/O Operations. Shown in the Fig 3.2 8051 Pin diagram contains totally 40 pins, a total of 32 pins are set aside for the four ports P0,P1,P2and P3where each port taken 8 pins.

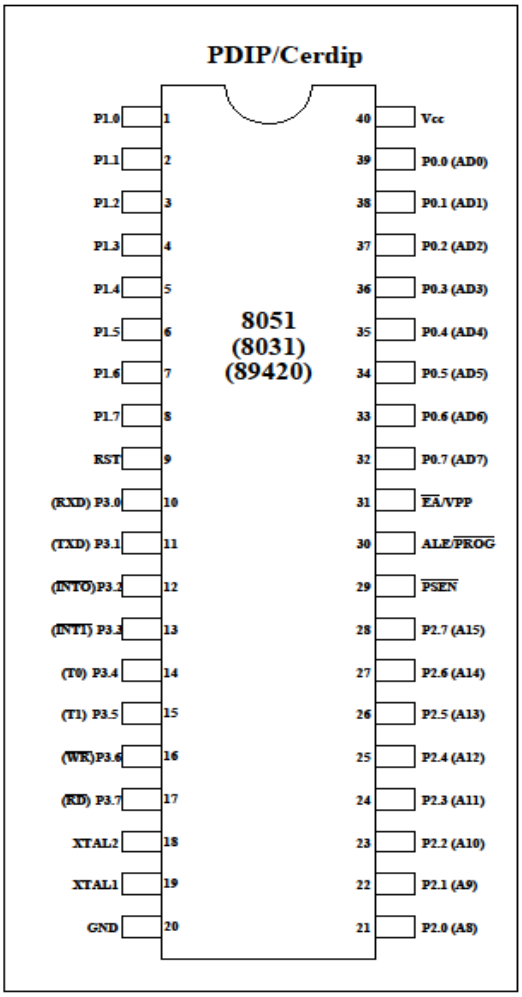


Fig 3.2 Pin Diagram

I/O Port Pins and their functions

The four ports P0, P1,P2 and P3each use 8 pins , making them 8-bit ports. All the ports upon RESET are configured as inputs, ready to be used as input ports. When the first 0 is written to a port, it becomes an output. To reconfigure it as an input, a 1 must be sent to the port, it must be Programmed.

Port 0

Port 0 contains a total of 8 pins (32 to 39) can be used for input or output or both input and output ports, as a bidirectional lower order address and data bus for external memory.

Each Port 0 Pin must be connected externally to 10K ohm Pull- up resistor . This is due to the fact that P0 is an open drain, unlike P1,P2and P3.Open drain is a term used for MOS chips in the same way that open collector is used for TTL chips. In any system using 8051/52 chip , we take normally connect P0 to pull up resistors. See fig 3.3, In this way we take advantage of port0 for both Input and Output.

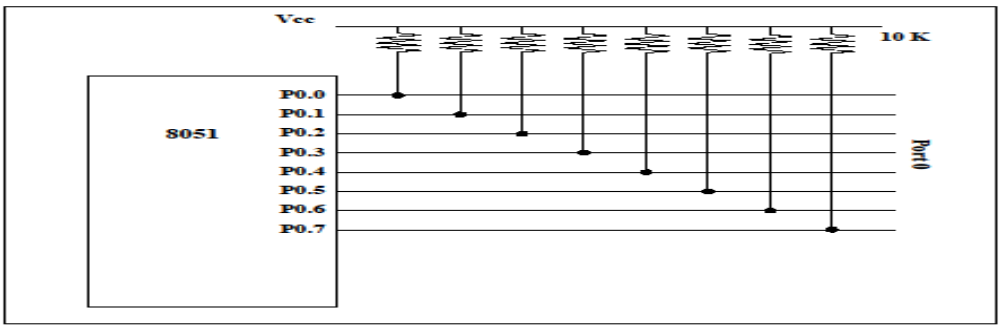


Fig 3.3 Port 0 with Pull-up resistors

Example 3.1

The following code will continuously send out to port 0 the alternating values of 55H and AAH.

; Toggle all bits of P_0

```
Back:   MOV     A,  #55H
        MOV     P0, A
        ACALL   DELAY
        MOV     A,  #0AAH
        MOV     P0, A
        ACALL   DELAY
        SJMP    BACK
```

NOTE: 55H (01010101) turns it into AAH (10101010).By sending 55H and AAH to a given Port continuously , we toggle all the bits of the Port.

Port 1

Port 1 contains 8 pins (1 to 8).It can be used as Input or Output. In contrast to Port 0 ,this Port does not need any Pull-up resistors since it already has Pull-up resistors internally. Upon reset, Port 1 is configured as an input port. The following code will continuously send out to Port 1 the alternating values 55H and AAH.

; Toggle all bits of P_1 Continuously.

```
        MOV     A, #55H
Back:   MOV     P1, A
        ACALL   DELAY
        CPL A           ;Complement (Invert ) reg. A
        SJMP    BACK
```

Port 2

Port 2 contains 8 pins (21 to 28).It can be used as Input or Output. Just like P1,Port 2 does not need any pull-up resistors since it already has pull-up resistors internally. Upon reset, Port 2 is configured as an input port .The following code will send out continuously to port 2 the alternating values 55H and AAH .That is all the bits of P_2 toggle continuously.

```
        MOV     A, #55H
Back:   MOV     P1, A
        ACALL   DELAY
        CPL A           ;Complement reg. A
        SJMP    BACK.
```

Port 3

Port 3 contains 8 pins (10 to 17).It can be used as Input or Output. P3 does not need any pull-up resistors, just as P1 and P2 did not .Table 3.1 Provides these alternate functions of P3.This function common to both 8051 and 8031 chips.

Table 3.4 Provides the alternate functions of P3.

P3 Bit	Function	Pin
P3.0	RXD	10
P3.1	TXD	11
P3.2	$\overline{INT0}$	12
P3.3	$\overline{INT1}$	13
P3.4	T0	14
P3.5	T1	15
P3.6	$\overline{WR}$	16
P3.7	$\overline{RD}$	17

I/O BIT MANIPULATION PROGRAMMING

I/O bit manipulation instructions is a powerful and widely used feature of the 8051 family. By using this instructions ,we can perform various functions like clear, set, complement , AND logic ,OR logic ,move and control jump in bit level. The single bit data may be placed either in the internal ″bit and byte addressable RAM″ or in the ″ bit and byte addressable SFR″. The carry flag acts as the accumulator for bit manipulation operations. For example , the following code toggles bit P1.2continuously.

Example:3.2

```
Back :      CPL      P1.2          ;Complement P1.2 only
            ACALL    DELAY
            SJMP     BACK

;another variation of the above program follows

AGAIN:      SETB     P1.2          ;change only P1.2 = high
            ACALL    DELAY
            CLR      P1.2          ;change only P1.2 = low
            ACALL    DELAY
            SJMP     AGAIN
```

Table 3.3 lists the Single bit Instructions for the 8051 .

Example:3. 3

Write the following Program to create a square wave of 50% duty cycle on bit 0 of Port 1.

Solution:

The 50% duty cycle means that the on and off states (or the high and low portion of the pulses) have the same length. we toggle P1.0 with a time delay in between each state.

```
HERE:       SETB     P1.0          ;Set to high bit 0 of Port 1
            LCALL    DELAY         ;Call the delay subroutine
            CLR      P1.0          ; P1.0=0
            LCALL    DELAY
            SJMP     HERE          ;Keep doing it.
```

Another way to write the above program is:

```
            CPL      P1.0          ;Complement bit 0 of Port 1
            LCALL    DELAY         ;call the delay subroutine
            SJMP     HERE          ;Keep doing it.
```

Checking an input bit

The JNB (Jump if no bit) and JB (Jump if bit =1) instructions are widely used single bit operations.JNB and JB can be used for any bits of I/O ports 0,1,2 and 3.Since all ports are bit addressable.

Example 3.4

A Switch is connected to Pin P1.7 .Write a program to check the status of Sw and Perform the following.

(a) If Sw =0, send letter 'N' to P2

(b) If Sw =1,send letter 'Y' to P2

Solution:

```
        SETB P1.7          ;Make P1.7 an input
AGAIN : JB P1.7 ,OVER      ; Jump if P1.7 =1
        MOV P2 ,# 'N'      ;Sw=0,issue 'Y' to P2.
        SJMP AGAIN         ;Keep monitoring
OVER :  MovP2,# 'Y'        ;Sw =1,issue 'Y' to P2.
        SJMP AGAIN         ;Keep monitoring.
```

Reading a Single bit in to the carry flag

We can also use the carry flag to save or examine the status of a single bit of the port.

Example: A switch is connected to pin P1.0and an LED to pin P2.7.Write a program to get the status of the switch and send it to the LED.

Solution

```
        SET B P1.7          ;Make P1.7 an input

AGAIN   MOV C, P1           ;read the Sw status into CF
        MOV P2.7,C          ;Send the Sw status to LED
        SJMP AGAIN          ;Keep repeating.
```

Note: The instruction ″MOV P2.7,P1.0″ is wrong since such an instruction does not exist. However ,″ MOV P2.7,P1.0″ is a valid instruction.

Reading input Pins Vs Port latch

When reading Ports there are two possibilities

1)Read the status of the input Pin.

2)Read the internal latch of the output Port.

Read – Modify –Write Technique

The Instructions that read the Port latch normally read a value, Perform an Operation (and possibly change it),than rewrite it back to the port latch. This is often called ″Read – Modify – Write″.

This feature save many lines of code by combining in a single instruction all three actions of

i)reading the Port

ii)Modifying its value

iii)Writing to the Port

Mnemonic Example	
ANL PX	ANL P1, A
ORL PX	ORL P2, A
XRL PX	XRL P0, A
JBC PX .Y, TARGET	JBC P1.1, TARGET
CPL PX .Y	CPL P1.2
INC PX	INC P1
DEC PX	DEC P2
DJNZ PX .Y, TARGET	DJNC P1, TARGET
MOV PX .Y, C	MOV P1.2 , C
CLR PX .Y	CLR P2.3
SETB PX .Y	SETB P2.3

Note: x is 0,1,2 or 3 for PO-P3.

Table 3.5 Instructions Reading a Latch (Read- Modify -Write)

Example 3.5

The following code first places 01010101(binary) into part 1.Next ,the instructions "XLR P1 ,#0FFH" perform an XOP logic operation on P1with 11111111(binary) and then writes the result back into P1 .

```
                MOV P1,   #55H          ; P1=01010101

AGAIN:          XLR  P1 ,  #0FFH        ; EX-OR P1with 11111111
                ACALL    DELAY
                SJUMP    AGAIN
```

Note: The XOR of 55H and FFH gives AAH, likewise the XOR of AAH and FFH gives 55H.

3.2 TIMER

The 8051 has two timers / Counters , namely timer 0 and timer 1. They can be used either as timers to generate a time delay or as counters to count events happening outside the

Microcontroller. Timers/Counters may be Controlled , Set, read and Configured individually. The timers have three general functions :

1.Keeping time and /or calculating the amount of time between events.

2.Counting the events themselves.

3.Generating band rates for the serial port.

Two Timers: Timer 0 and Timer 1 can be used either as timers or as event Counters. Both Timer 0 and Timer 1are 16 -bits wide. Since the 8051 has an 8-bit architecture, each 16-bit timer is accessed as two separate registers of low byte and high byte.

3.2.1 Timer 0 Registers

The 16-bit register of timer 0 is accessed as low byte and high byte. The low byte register is called TL0 (Timer 0 low byte) and high byte register is called TH0 (Timer 0 High byte).These registers can be accessed like any other register such as A,B,R0,R1,R2, etc. For example :MOV TL0,#2FH moves the value 2FH into TL0, the low byte of timer 0.These registers can be also read like any other registers. For example: MOV R5, TH0 saves TH0 (high byte of Timer 0) in R5.

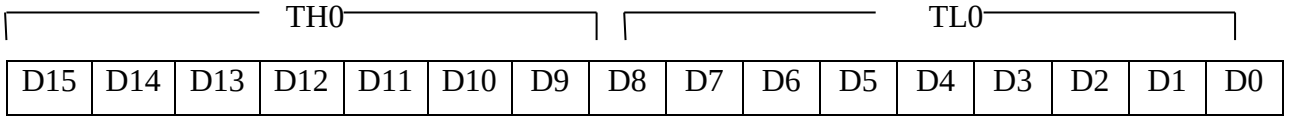


Fig 3.4 Timer 0 Register

3.2.2 Timer 1 Registers

Timer 1 is also 16-bits, and its 16-bit register is split into two bytes, referred to as TL1 (Timer 1 low byte) and TH1(Timer 1 high byte).These registers are accessible in the same way as the registers of Timer 0.

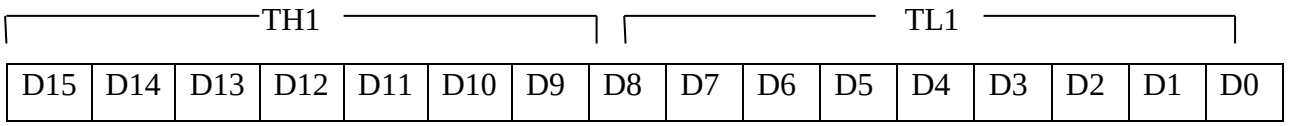


Fig 3.5 Timer 1 Register

3.2.3 Timer Mode Control Register (TMOD)

TMOD is an 8bit register. Both timers 0 and 1 use the same register called TMOD, to set the various timer operation modes. The lower 4 bits are set aside for Timer 0 and the upper 4 bits for Timer 1.TMOD is used to set the mode function of counter.

(MSB) **(LSB)**

7	6	5	4	3	2	1	0
Gate	C/T	M1	M0	Gate	C/T	M1	M0
TIMER 1				TIMER 0			

Fig 3.6 TMOD Register

Bit	Symbol	Function
7/3	GATE	OR Gate. Function used to run or stop 1/0. When set to 1 by program will enable to run if bit TR1/0 IN TCON register is set, and external interrupt Pin $\bar{I}NT$ 0/1 pin is high. when set to 0 by program Will enable timer to run if bit TR1/0 in TCON is set. This is independent of the state of the $\bar{I}NT$ 0/1 pin.
6/2	C/T	When set to 1 the counter/timer functions as a Counter ,counting pulses from external inputs. Timer 1 from pin 5 on port 3 and timer 0 from pin 4 on port 3. Set to 0 the counter /timer functions as a timer counting internal frequency. The count frequency =(Oscillator frequency/12)
5/1	M1	Mode select bit for timer/counter. Set or cleared by program.
4/0	M0	Mode select bit for timer/counter. Set or cleared by program.

Table 3.6 Functions of TMOD REGISTER

M1	M0	MODE	Timer/Counter Operating Modes
0	0	Mode 0	13 bit Timer Mode 8-bit Timer/Counter THX with TLX as 5-bit pre scalar
0	1	Mode 1	16 bit Timer Mode 16-bit Timer/Counters THX and TLX are cascaded; there is no pre scalar
1	0	Mode 2	8-bit auto – reload 8-bit auto – reload Timer/Counter ; THX holds a value that is to be reloaded into TLX each time it overlaps
1	1	Mode 3	Split Timer (Two 8-bit Timer) Mode

Table 3.7 Modes of operations of Timer/Counter

3.2.4 Timer Control Register (TCON)

TCON is an 8-bit register. The upper four bits are used to store the TF and TR bits of both Timer 0 and Timer 1.The lower four bits are set aside for controlling the interrupt bits. TR0 and TR1 flags are used to turn on or turn off the timers.

(MSB) **(LSB)**

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Fig 3.7 TCON REGISTER

Bit	Symbol	Function
7	TF1	Timer 1 Overflow Flag: Set and cleared by hardware. When it is set to 1 indicates Timer Overflow has occurred.
6	TR1	Timer 1 Run Control Bit: Set to 1 by software to make the timer 1 count .Set to 0 to make the timer hold its count.
5	TF0	Timer 0 Overflow Flag: Set and cleared by hardware. When it is set to 1 indicates Timer Overflow had occurred.
4	TR0	Timer 0 Run Control bit: Set to 1 by software to make the timer 1 count .Set to 0 to make the timer hold its count.
3	IE1	Interrupt 1 Edge Flag: Set to 1 when external interrupt is detected. Cleared by hardware when interrupt is serviced.
2	IT1	Interrupt 1 Type Control Bit: Set or cleared by software. Set to specify falling edge interrupt . Clear for low level interrupt.
1	IE0	Interrupt 0 Edge Flag: Set to 1 when external interrupt is detected. Cleared by hardware when interrupt is serviced.
0	IT0	Interrupt 0 Type Control Bit :Set or cleared by software. Set to specify falling edge interrupt. Clear for low level interrupt.

Table 3.8 Functions of TCON Register

3.2.5 Different Modes of Timers

The Timer registers of 8051 Microcontroller operates in four different modes namely,

1)Mode 0 - 13 – bit Timer mode.

2)Mode 1 - 16 – bit Timer mode.

3)Mode 2 - 8 – bit Auto – reload mode.

4)Mode 0 - Two 8 bit (Split) timer mode.

Mode 0 (13 bit timer mode)

Timer mode "0"is a 13-bit timer. The 13-bit register consists of all 8-bits of TH1 and the lower 5 bits of TL1.Mode 0 operation is same for timer 0 and timer 1.

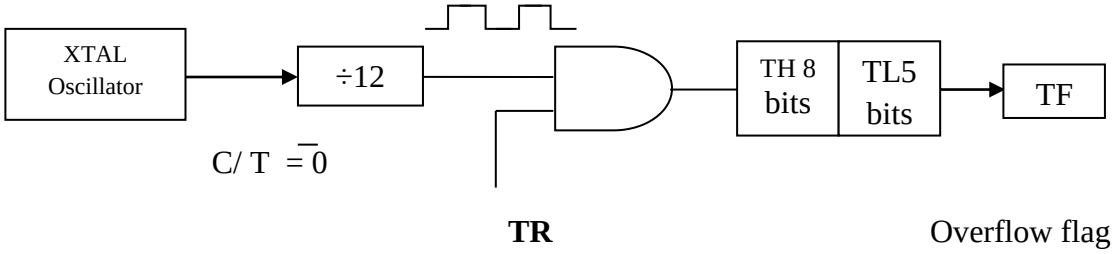


Fig 3.8 Mode 0 Operation: 13-BIT TIMER MODE

Mode 1 (16 bit Timer mode)

Timer mode 1 is a 16 bit Timer. Mode 1 is similar to mode 0 except that the timer register is being run with all 16 bits. Maximum count is being 65,536.

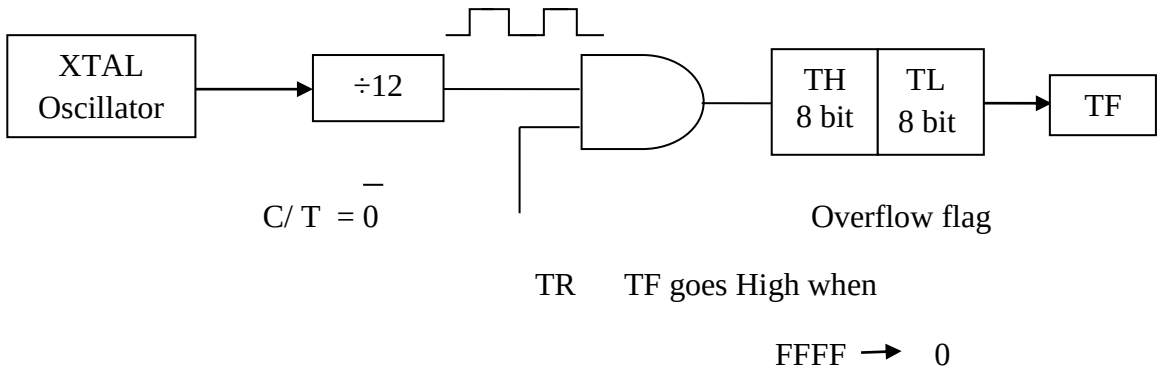


Fig 3.9 Mode 1 Operation:16-bit Timer Mode

Mode 2 (8 bit auto reload timer mode)

Timer mode 2 is an 8 bit auto reload mode. TL is reloaded automatically with the original value kept by the TH register.

i)Only TL0 and TL1 are used i.e 8 bit counting.
ii)Initial preset value is loaded to TH0 or TH1 by software.

iii)The value is loaded to TL0 or TL1 by hardware automatically before starts of counting.

iv)When count rolls from all once is (i.e FFH) to all 0's (i.e 00H). TF0 or TF1 flag is set.

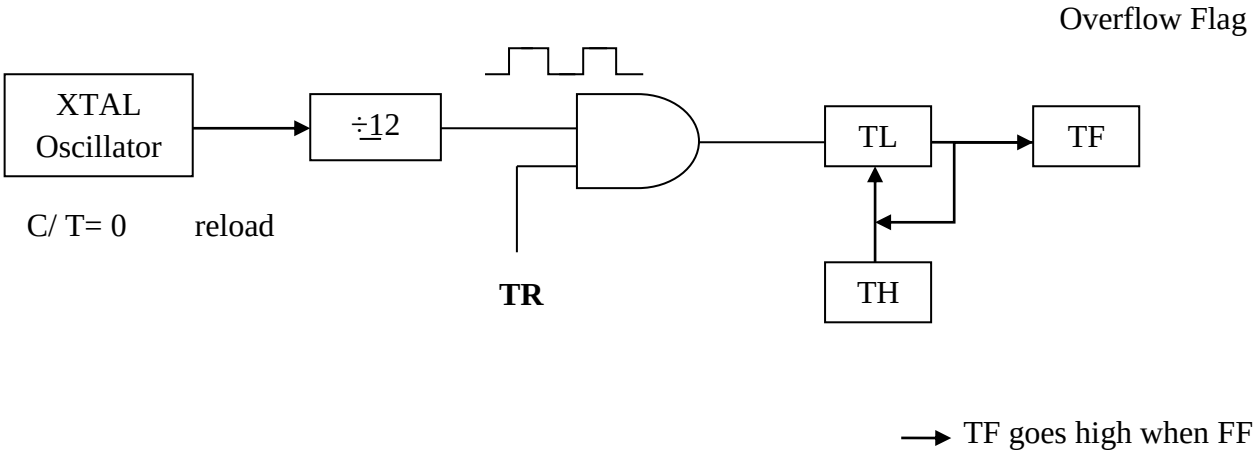


Fig 3.10 Mode 2 Operation:8-bit Auto Reload Timer Mode

Mode 3 (Two 8 bit Timer 0 mode)

Timer mode 3 is a Split-timer mode. When timer 0 is placed in mode 3,it becomes two separate 8 bit timers. Timer 0 is TLO and the Timer 1 is THO. Timer 1 can operates in mode 0,1 or 2.When Timer 0 is in mode 3, Timer 1 holds its count. It is not updated.

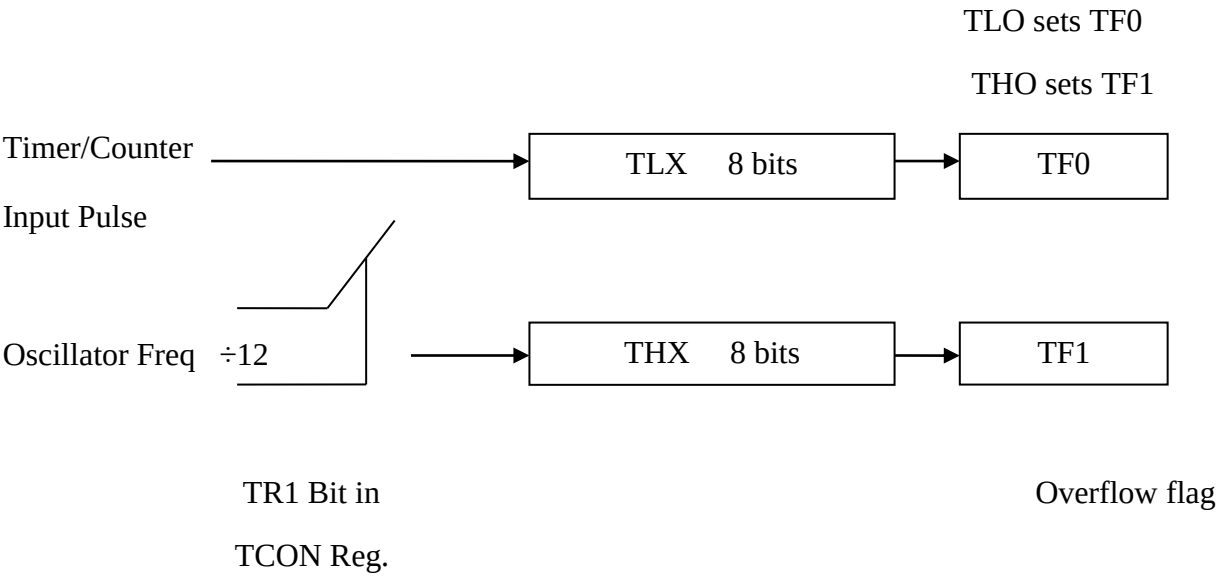


Fig 3.11 Mode 3 Operation: (Two 8 bit timer mode)

3.2.6 Mode 0 Programming

Mode 0 is exactly like mode 1 except that it is a 13-bit timer instead of 16-bit.The 13-bit counter can hold values between 0000 to 1FFFH in TH-TL. Therefore, when the timer reaches its maximum of 1FFH, it rolls over to 0000, and TF is raised.

EXAMPLE 1

To write an ALP for creating a square wave signal with 50% duty cycle at pin P1.5, by using timer 0 in mode 0

```
        MOV TMOD ,#00H          ;Set timer 0 in mode 0
        CLR EA                  ;Disable all interrupts
REPEAT   MOV TLO,#1FH           ; TL0=0001 1111 B
        MOV THO ,#00H           ;TH0=0000 0000 B
        CLR TF0                 ;Clear timer 0 overflow flag bit
        SETB R0                 ;Start Timer 0 function
WAIT   JNB TF0,WAIT             ;Wait for timer 0 overflow
        CPL P1.5                ;Complement the previous output
        SJMP REPEAT             ;Repeat the process
```

In this program the timer 0 is incremented from 001F(TH0 = 0000 0000 ; TL0 =XXX 1 1111) (=0 0000 0001 1111 B =001FH) at every machine cycle. When the timer value reaches 1FFFH , then it will rollover at the next machine cycle.

3.2.7 Mode 1 Programming

The following are the characteristic and operation of mode 1.

1.It is a 16-bit timer therefore, it allows value of 0000 to FFFFH to be loaded into the timer register TL and TH.

2.After TH and TL are loaded with a 16-bit initial value , the timer must be started. This is done by " SETB TR0 " for timer
and " SETB TR1 " for Timer 1.

i)Only TL0 and TL1 are used i.e 8 bit counting.

ii)Initial preset value is loaded to TH0 or TH1 by software.

iii)The value is loaded to TL0 or TL1 by hardware automatically before starts of counting.

iv)When count rolls from all once is (i.e FFH) to all 0's (i.e 00H)
TF0 or TF1 flag is set.

3.After the timer is started ,it starts to counted .It counts up until it reaches it limit of FFFFH. When it rolls over from

FFFFH to 0000, it lets high monitored. When this their flag is raised, one option would be to stop the timer with the instruction "CLR TR0" or "CLR TR1" ,for Timer 0 and Timer 1.

4.After the Timer reaches its limit and rolls over ,in order to repeat the process the register TH and TL must be reloaded with the original value , and TF must be reset to 0.

Steps to Program in Mode 1:

To generate a time delay, using the timers mode 1, the following steps are taken.

1)Load the TMOD value register indicating which timer is to be used and which timer mode (0 or 1) is selected.

2)Load register TL and TH with initial count values.

3)Start the timer.

4)Keep monitoring the timer flag (TF) with the " JNB TFX , target " instruction to one of it is raised. Get out if the loop when TF becomes high.

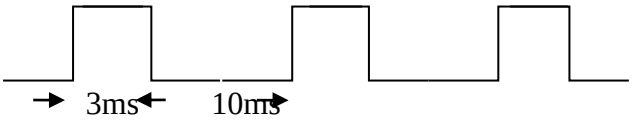
5)Stop the timer.

6)Clear the TF flag for the next round.

7)Go back to step 2 to load TH and TL again.

Example :1

Generate a square wave with an ON time of 3ms and an OFF time of 10ms on all pins of port 0. Assume an XTAL of 22 MHz.



Solution:

Tested for an AT89C51 with a crystal frequency of 22 MHz.

Let us use Timer 0 in Mode 1.

```
        MOV   TMOD , #01H       ; Timer 0 in mode 1
BACK:   MOV     TL0 , # 075H    ; to generate the OFF time , load TL0
        MOV   TH0 , # 0B8H      ; load OFF time value in TH0
        MOV   P0 , # 00H        ; make port bits low
        ACALL       DELAY       ; call delay routine
        MOV   TL0 , # 8AH       ; to generate the ON time , load TL0
```

```
        MOV   TH0 , # 0EAH      ; load ON time value in TH0
        MOV   P0 , # 0FFH       ; make port bits high
        ACALL       DELAY       ; call delay
        SJMP  BACK              ; repeat for reloading counters to get a
                                ; continuous square wave
        ORG   300H
DELAY:  SET TR0                 ; start the counter
AGAIN:  JNB TF0,AGAIN           ; check timer overflow
        CLR   TR0               ; when TF0 is set , stop the timer
        CLR   TF0               ; clear timer flag
        RET
        END                     ; end of file
```

For OFF Time calculation:

10ms/0.546μs = 18,315 cycle

65536 – 18,315 = 47,221 = B875H

3.2.8 Mode 2 Programming

The following are the characteristics and operations of mode 2 .

1.It is the 8 bit Timer. It allows only values if 00 to FFH to be loaded into the timer's register TH.

2.After TH is loaded with the 8 bit value ,the 8051 gives a copy of it to TL. Then the timer must be started this is done by the instruction." SETB TR0 "for Timer 0 and "SETB TR1" for Timer 1 (Just like mode 1).

3.After the timer is started ,it starts to count up by incrementing the TL register. It counts up until it reaches the limit of FFH. when it rolls over from FFH to 00,it sets high the TF (using Timer 0 ,TF0 goes high and using TF1 goes high.)

4.When the TL register rolls from FFH to 0 and TF is set to 1,TL is reloaded automatically with the original value kept by the TH register. This makes mode 2 an auto-reload, in contrast with mode 1in which the programmer has to reload TH and TL.

Steps to program in mode 2 :

To generate a time delay using the timer's mode 2 by take the following steps.

1.Load the time value register indicating which timer (0 or 1) is to be used and select the timer mode.(mode 2).

2.Load the TH register with the initial count value.

3.Start the timer.

4.Keep monitoring the timer flag (TF) with the " JNB TFX , target " instruction to see whether it is raised. Get out of the loop when TF goes high.

5.Clear the TF flag.

6. Go back the step 4,Since mode 2 is auto-reload.

Example 1

Assuming that XTAL =11.0592 MHz, find (a) the frequency of the square wave generated on pin $P_{1.0}$ in the following program and(b)the smallest frequency achievable in this program ,and the TH value to do that.

Solution:

(a) First notice the target address of SJMP

In the mode 2,we do not need to reload TH since it is auto-reload. Now,(256-05) X 1.085 μ S=251 X 1.085 μ S = 272.33 μ S is the high portion of the pulse.

Note: Clock frequency of 1/12 is 11.0592 MHz/12 =921.6KHz.

T=1/F =1/921.6KHz =1.085 μ S

Since, It is a 50% duty cycle square wave ,the period T is twice.

T =2x272.33 μ S =544.67μ S

Frequency = 1.83597 KHz.

(b)To get the smallest frequency ,

we need largest T achieved, when TH=00

T=2x 256x1.085μ S = 555.52 μ S

Frequency =1/T =1/555.52 μ S = 1.8 KHz

Program

```
        MOV TMOD, #20H          ; T1/Mode 2/8 bit /auto -reload.
        MOV  TH1,#5             ; TH1=5
        SETB TR1                ;Start Timer 1
BACK:   JNB TF1,BACK            ; Stay until timer rolls over
        CPL P1.0                ; Complement P1.0 TO GET li,l0
        CLR TF1                 ; Clear Timer 1 flag
        SJMP BACK               ; mode 2 is auto-reload
```

Example 2

Assuming that we are Programming the timers for mode 2,find the value loaded into TH for each of the following order.

(a)MOV TH1,# 200

(b)MOV TH0,# -60

(C)MOV TH1,# -3

(d) MOV TH1,# -12

(e) MOV TH0,# -48

Solution

You can use the windows Scientific calculator to verify the results provided by the assembler. In windows calculator, select decimal and enter 200.Then select here, the +/- to get the TH value. Remember that we only use the right two digits and ignore the rest since our data is an 8-bit data.

Decimal	2's Complement(TH value)
-200	38H
-60	C4H
-3	FDH
-12	F4H
-48	D0H

3.2.9 Counter Programming

A Counter , given the count inputs at regular intervals (called clock input),also functions as a timer. Timers can also be used as counter counting events happening outside the 8051.When it is used as a counter it is a pulse outside of the 8051 that increments the TH,TL registers.

The only difference between counting and timing is the source of the clock pulses to the counters. when used as a counter ,Pin T0 ($P_{3.4}$) supplies. Pulses to counter 0 ,and pin T1 ($P_{3.5}$) to counter 1.The $C/\overline{T}$ bit in TMOD must be set 0,to enable pulses from the TX pin to reach the control circuit.

C/T Bit in TMOD Register

The C/T bit in the TMOD register decides the source of the clock for the timer.

i)When C/T =1 ,the timer is used as a counter and gets its pulses from outside the 8051.

ii)The counter counts up as pulses are fed from pins 14 and 15,these pins are called T0 (Timer 0 input) and T1 (Timer 1 input)

Table 3.9 Port 3 Pins used for Timer 0 and Timer 1

Pin	Port Pin	Function	Description
14	$P_{3.4}$	T0	Timer/counter 0 external input
15	$P_{3.5}$	T1	Timer/counter 1 external input

3.2.10 Different Mode of Counter

TMOD Register is an 8 bit register in which to lower 4 bits are set aside for Timer 0 and the upper 4 bits for Timer 1. C/T bits for both Timers used to decides Timers or counter selected

Cleared for timer operations. Set for counter operations (Input from TX input pin) when $C/\overline{T}=1$,the counter counts up as pulses are fed from T0 ($P_{3.4}$,Pin 14) and ($P_{3.5}$,Pin 15) for counter 0 and counter 1 respectively.

The counter is operating in 4 different modes. The M1 and M0 bits of TMOD register is used to select the counter mode. Refer table 3.7.

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				Timer 0			

Fig 3.12 TMOD Register

Table 3.10 Different Modes of Counter

M1	M0	Mode	Operating mode
0	0	0	13 bit counter 8 bit THX with TLX as 5 bit Pre scalar
0	1	1	16 bit counter ,THX and TLX are cascaded.
1	0	2	8 bit auto -reload counter THX holds a value that is to be loaded intoT0,T1 each time it overflows.
1	1	3	Split (2-8 bit) counter mode

3.2.11 Mode 0 Programming

Example 1

To write an ALP for switching connected at port pin P1.0 ON/OFF alternatively when the counter 0 operates in mode 0 overflows.

```
        ORG 4100            ; Origin at 4100H
        MOV TMOD,#04H       ; Set counter 0 in mode 0
        CLR EA              ; Disable all interrupts
        SETB P3.4           ; Make port pin P3.4 as input port.
REPEAT  CLR TF0             ; Clear overflow flag bit.
        MOV TL0,#05H        ; Load TL0=05H
        MOV TH0,#F2H        ; Load TH0=F2H
        SETB TR0            ; Start the counter 0 function
WAIT    JNB TF0,WAIT        ; Wait for counter overflows
        CLR TR0             ; Stop the counter function
        CPL P1.0            ; Complement the previous output
        SJMP REPEAT         ; Repeat the process
        END
```

3.2.12 Mode 1 Programming

Example 1

Write an ALP using counter 0 in Mode 1 to ON specific load connected at port pin P1.3,after the counter counting 25,000D number of clock pulses.

Initial value to be loaded in counter register

= 65536-25000 = 40536 D = 9E58H

```
        ORG 4500H       ; Origin at 4500H
        CLR EA          ; Disable all interrupts
        SETB P3.4       ; Make port pin P3,4 as input port.
        CLRP1.3         ; OFF the output load
        MOV TMOD,#05H   ; Set counter 0 in mode 1
        MOV TH0,#9EH    ; Load initial value in TH0 register
        MOV TL0,#58H    ; Load initial value in TL0 register
        CLR TF0         ; Clear overflow flag bit
        SETB TR0        ; Start counter function
WAIT    JNB TF0,WAIT    ;Wait for counter 0 overflows
        SETB P1.3       ;ON the output load
        CLR TR0         ; Stop the counter function
HLT     SJMP HLT        ;Halt here
```

Example 2

Write an ALP using counter 0 in Mode 1 to calculate the frequency of the input signal occurred at counter 0 (T0) terminal. Assume the frequency of the clock signal is 12 Mhz.

Frequency of the clock signal = 12 MHz

Time period of one clock signal

$= 1/12x10^{6} = 8.33x10^{-8}$ sec

Time period of one machine cycle

$= 8.33x10^{-8}$ x12 = 1µsec

No. Of machine cycles needed to make one second time period

= 1 sec/ 1µs = 1000000D = F4240 H

The count value of F4240H is not directly placed in any timer.

So, we can split the value as = 62500D x16 D

=F424H x10H

The initial value to be loaded in timer register

= 65536D – 62500D = 3036D = 0BDCH

For executing this program load the value of 0BDCH in timer 1 registers and operate timer 1 as timer in mode 1 and 16D (10H) as timer loop value (Place this value in R0 register).

```
        ORG 4100H          ;Origin at 4100H
        MOV TMOD,#15H      ; Set timer 1 in mode 1 and counter 0 in mode 1
        CLR EA             ; Disable all interrupts
        SETB P3,4          ; Make port pin P3,4 as input port.
        MOV R0,#10H        ; Load 10H in register R0 for Looping
        MOV TH0,#00H       ; Make TH0=0000H
        MOV TL0,#00H       ; Make TL0=0000H
        SETB TR0           ; Start counter 0 function
REPEAT  MOV TH1,#0BH       ; Load initial value in TH1 register
        MOV TL1,#DCH       ; Load initial value in TL1 register
        CLR TF1            ; Clear timer overflow flag bit
        SETB TR1           ; Start timer 1 function
WAIT    JNB TF1 WAIT       ; Wait for timer 1 overflows
        CLR TR1            ; Stop timer 1 function
        DJNZ R0, REPEAT    ; Repeat the above process
        CLR TR1            ; Stop timer one function
        DJNZ R0, REPEAT    ; Repeat the above process
        CLR TR0            ; Stop the counter function
        END                ; End of program
```

After the execution of this program the frequency of the input signal is placed in counter 0 registers of TH0 and TL0.

3.2.13 Mode 2 Programming

Example 1

To Write an ALP for alternatively switch ON/OFF the load connected at port pin $P_{1.0}$ when the counter overflows , by using counter 1 in mode 2.

```
        ORG 4100H          ; Origin at 4100H
        MOV TMOD,#60H      ; Set  in counter 1 in mode 2
        SETB  T1           ; Make port pin P3.5 as input port.
        CLR EA             ; Disable all interrupts
        MOV TH1,#05H       ; Load  TH1=05H
        MOV TL1,#05H       ; Load TL1=05H
REPEAT  CLR TF1            ; Clear overflow flag bit
        SETB TR1           ; Start counter 1 function
WAIT    JNB TF1 WAIT       ; Wait for Counter overflows
        CLR TR1            ; Stop Counter function
        SJMP REPEAT        ; Repeat the process
        END                ; End of program
```

Timer Vs Counter (Differences)

Timer - Counts Machine Cycles

Counters - Counts events as a result of falling slope of external input signal put on a pin.

Timer mode and Counter mode are relative to machine cycle.

Timer - Input from internal system clock.

Counter - Show the number of events on register.

- External input from T0 input pin ($P_{3.4}$) for Counter 0.

- External input from T1 input pin ($P_{3.5}$) for Counter 1.

- External input from TX input pin ($P_{3.4}$).

-We use TX to denote T0 or T1.

REVIEW QUESTIONS

PART-A (2 MARKS)

1. Write the bit address of port1 and port 3.
2. What is the capacity of bit addressable area of Internal RAM of 8051?
3. In 8051, which port need a pull up resistor in order to be used as I/O?
4. In the 8051 , how many pins are designated as I/O port pins?
5. Is the instructing "CPL P1" a valid instruction ?
6. Mention the operating modes of 8051 timers.
7. What is the function of gate signal in timer?
8. State the function of timer flag TF in TCON register.
9. Mention the SFR register used in the Timer operation .
10. What is the use of counter?

PART-B (3 MARKS)

1. Write the alternative function of port 3 in 8051.
2. What is the advantage of bit –addressability for 8051 port?
3. Explain Read –modify-write Instruction.
4. What is the difference between the operation of a timer and a counter?
5. Explain TMOD register in 8051.
6. How many timers available in the 8051? Draw the timer.
7. Draw TCON register in detail.
8. Find the timer 's clock frequency for the crystal frequency of 11 .0592 MHz?
9. What is the function of gate signal in timer?
10. Explain the different operating modes of counter in 8051.

PART-C (10 MARKS)

1. Explain the bit addresses for I/O of 8051.
2. Explain the bit addresses for RAM.
3. Explain the Programming of I/O ports in 8051
4. Explain TMOD and TCON registers.
5. Explain the different operating modes of timer in 8051.
6. Explain in details about the programming of 8051 timer.
7. Explain the steps to program the timer in mode1 and mode2.
8. Write the Program to generate square wave of 50hz frequency on Pin $P_{1.2}$ using timer0 interrupt .Assume crystal frequency in 11.0592MHZ.
9. Explain counter programming.
10. Write an ALP for alternatively switch ON/OFF the load connected at Port Pin $P_{1.0}$ when the counter overflows , by using counter1 in mode 2.

Unit – IV

INTERRUPT AND SERIAL COMMUNICATION

4.1 SERIAL COMMUNICATION

Introduction

Computers transfer data in two ways: Parallel and serial. In Parallel data transfers, often 8 or more lines (wire conductors) are used to transfer data to a device that is only a few feet away. Examples of Parallel transfers are Printers and hard disks; each uses cables with many wire strips. Although in such cases a lot of data can be transferred in a short of time by using many wires in parallel, the distance cannot be great. To transfer to a device located many meters away, the serial method is used. In Serial Communication, the data is sent one bit at a time, in contrast to parallel Communication, in which the data is sent a byte (or) more at a time.

4.1.1. Basics of Serial Communication

When a microprocessor Communication with the outside world, it provides the data in byte-sized chunks. In some cases, such as Printers, the information is simply grabbed from the 8-bit data bus and presented to the 8-bit data bus of the Printer. This can work only if the cable is not too long, since long cable diminish and even distort signals. Furthermore an 8-bit data path is expensive. For these reasons, serial communication is used for transferring data between two system located at distances of hundreds of feet to millions of miles apart. Show in the fig 4.1 serial versus parallel data transfers.

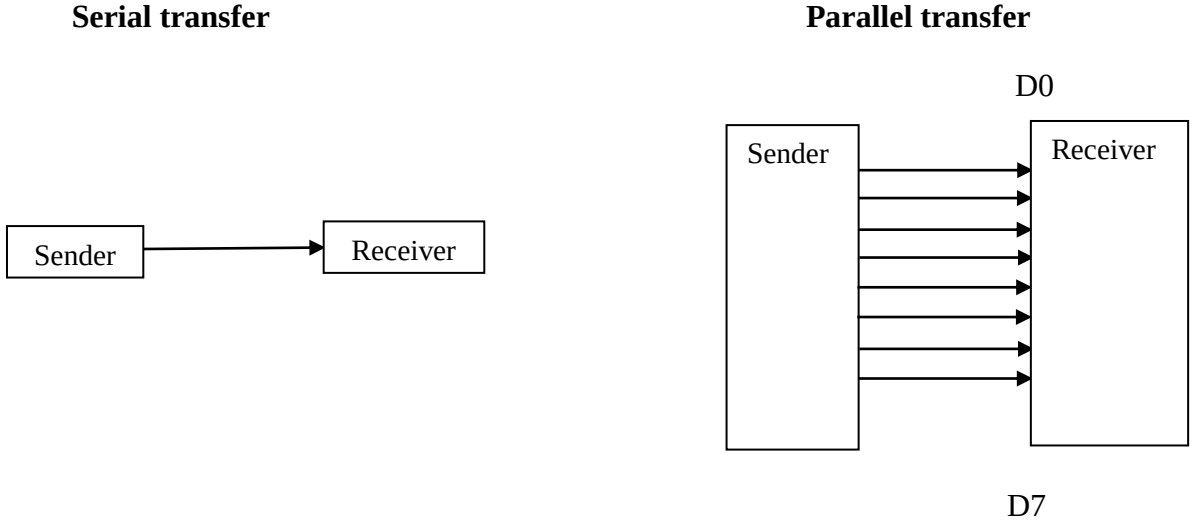


Fig: 4.1 Serial Versus Parallel Data Transfer

In fact that serial communication uses a single data line instead of the 8-bit data line of parallel communication not only makes it much cheaper but also enables two computer located in two different cities to communicate over telephone. For serial data communication, the byte of data must be connected to serial bits using a parallel –in-serial-out shift register, then it can be transmitted over a single data line.

At the receiving end using serial-in-parallel-out shift register to receive the serial data and pack them into a byte. This conversion is performed by a peripheral device called a modern, which stands for "modulator/demodulator".

Serial communication uses two methods, asynchronous and synchronous. The synchronous method transfers a block of data (characters) at a time, while the special IC chips (VART and USART) made by many manufacturers for serial data communication.

Half and Full duplex transmission

In data transmission if the data can be transmitted and received it is a duplex transmission. Simplex transmission such as Printers. Duplex transmissions can be half or full duplex, depending on whether or not the data transfer can be simultaneous. If data is transmitted one way at a time is called half duplex. If the data can go both ways at the same time is called full duplex. Full duplex requires two wire conductors for the data lines, one for transmission and one for reception, in order to transfer and receive data simultaneously. See figure 4.2.

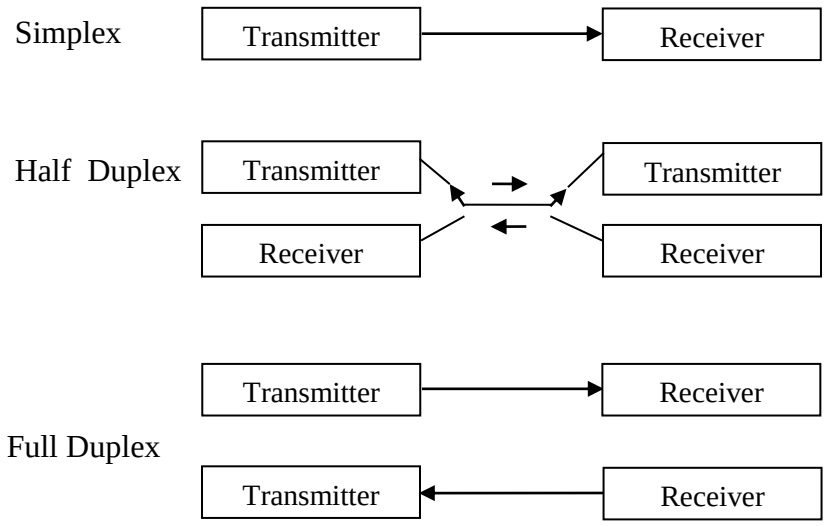


Fig 4.2 Simplex, Half and Full Duplex Transfers

Asynchronous Serial Communication and data framing

The data coming in at the receiving and of the data line in a serial data transfer is all O's and I's. It is difficult to make sense of the data unless the sender and receiver agree on a set of rules, a protocol, on how the data is packed, how many bits with a character, and when the data begins and ends.

Start and Stop bits

Asynchronous Serial data Communication is widely used for character-oriented transmissions, while block- oriented data transfers use the synchronous method.

In the asynchronous method, each character is placed between start and stop bits. This is called framing. In data framing for asynchronous communications, the data, such as ASCII character are packed between a start bit and a stop bit.

The Start bit is always one bit, but the stop bit can be one (or) two bits. The Start bit is always a 0 (low) and the stop bit is 1 (high). In fig 4.3 in which the ASCII character " A " (8- bit binary 0100 0001) is framed between the start bit and a single stop bit. Note: The LSB is sent out first.

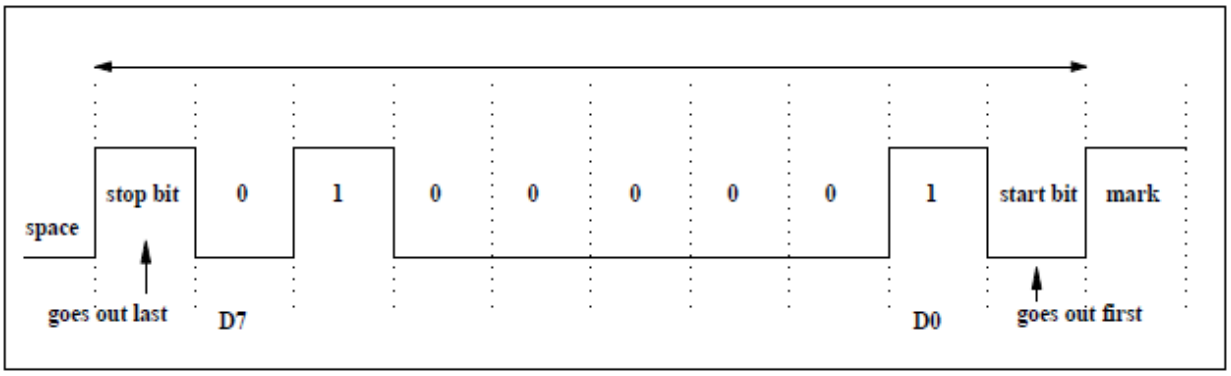


Fig 4.3 Framing ASCII " A " (41H)

Notice in fig 4.3 that when there is no transfer, the signal is 1 (high) which is referred to as mark. The 0 (low) is referred to as space. The transmission begins with a start bit followed by D0, which is the LSB, then the rest of the bits until the MSB (D7) and finally, the one stop bit indicating the end of the character " A ".

4.1.2 RS 232 STANDARDS

RS 232 is widely used I/O interfacing standard. RS 232 was set by the Electronics Industries Association (EIA) in 1960. This standard is used in PC's and numerous types of equipment. However, since the standard was set long before the advent of the TTL Logic family, its input and output voltage levels are not TTL compatible. In RS 232, a 1 is represented by -3 to -25 V, while a 0 bit is +3 to +25 V, making -3 to +3 undefined. For this reason, to connect any RS 232 to a microcontroller system, we must use voltage converters such as MAX 232 to convert the TTL logic levels to the RS 232 voltage levels and vice versa. MAX 232 IC chips are commonly referred to as line drivers.

RS 232 Pins

Table 4.1 provides the pins and their labels for the RS 232 cables, commonly referred to as DB-25 connector. In labeling, DB-25P refers to the plug connector (male) and DB-25S is for the socket connector (female).Shown in the fig 4.4 RS 232 Connector DB – 25.

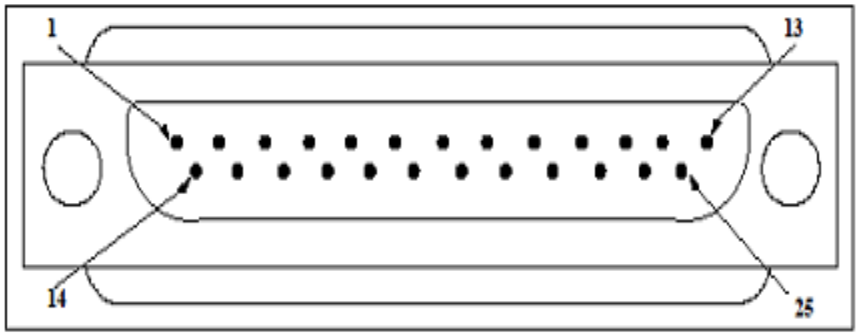


Fig 4.4 RS 232 Connector DB - 25

Since not all the pins are used in PC cables IBM introduced the DB-9 version of the serial I/O standard, which uses 9 pins only, as table 4.2. The DB-9 pins are shown in fig 4.4

Table 4.1 RS 232 Pins (DB-25)

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

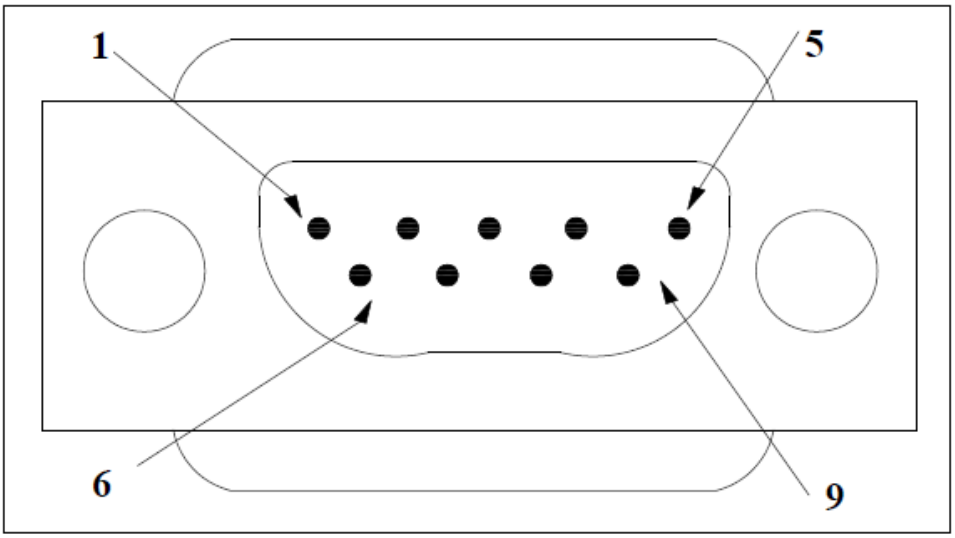


Fig 4.5 DB – 9 9-Pin Connector

Table 4.2 IBM PC DB-9 Signals

Pin	**Description**
1	Data Carrier detect ($\overline{DCD}$)
2	Received data (RXD)
3	Transmitted data
4	Data terminal ready ($\overline{DTR}$)
5	Signal ground (GND)
6	Data set ready ($\overline{DSR}$)
7	Request to send ($\overline{RTS}$)
8	Clear to send ($\overline{CTS}$)
9	Ring indicator (RI)

Data Communication Classification

Current terminology classifies data communication equipment as DTE (Data Terminal Equipment) or DCE (Data Communication Equipment). DTE refers to terminology and computers that send and receive data, while DCE refers to communication equipments such as modems, that are responsible for transferring the data.

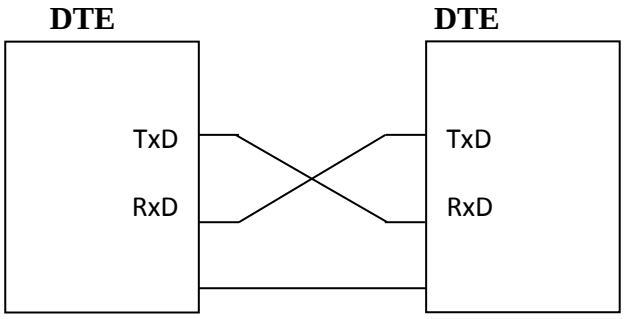


Fig 4.6 Null Modem Connection

The simplest connection between a PC and a microcontroller requires a minimum of three pins, TXD, RXD and ground, as fig 4.5. notice that the RXD and TXD pins are interchanged.

RS 232 Hand Shaking Signals

Many of the pins of the RS 232 connector are used for hand shaking signals. See fig 4.7. The descriptions of some hand shaking signals are referred in table 4.3.

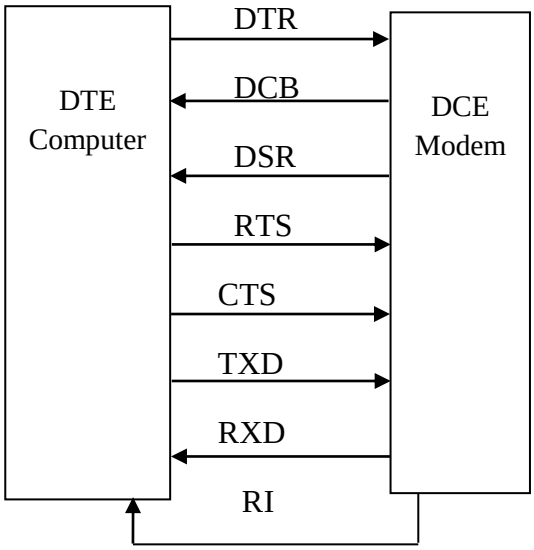


Fig 4.7 Hand Shaking Signals

Table 4.3 Hand shaking Signals

Circuit				Direction		DB-25
Sl.no	**Name**	**Typical Purpose**	**Abbreviation**	**DTE**	**DCE**	
1	Data Terminal Ready	DTE is read to receive, initiate, or continue a call	DTR	Out	In	20
2	Data Carrier Detect	DCE is receiving a carrier from a remote DCE	DCD	In	Out	8
3	Data Set Ready	DCE is ready to receive and send data	DSR	In	Out	6
4	Ring Indicator	DCE has detected an incoming ring signal on the telephone line	RI	In	Out	22
5	Request To Send	DTE requests the DCE prepare to transmit Data	RTS	Out	In	4
6	Ready To Receive	DTE is ready to receive data from DCE. If in use RTS is assumed to be always asserted	RTR	Out	In	4
7	Clear To Send	DCE is ready to accept data from the DTE	CTS	In	Out	5
8	Transmitted Data	Carries data from DTE to DCE	TXD	Out	In	2
9	Received Data	Carries data from DCE to DTE	RXD	In	Out	3
10	Common Ground	Zero voltage reference for all of the above	GND	Common		7
11	Protective Ground	Connected to chassis ground	PG	Common		1

Features

1) It is used for Serial Communication.
2) It is a Protocol Standard as well as electrical Standard.
3) It is used for short distances, up to 50 foot.
4) It's maximum data rate is 20,000 bd.
5) It is not TTL, Compatible.

Limitations of RS 232

1) Total load Capacitors on a signal line should not exceed 2500 PF.
2) To Connect with UART or TTL Circuit, additional current and voltage level Converters are needed.
3) Maximum data rate is 20,000 bd.

4.1.3 8051 CONNECTIONS TO RS 232

The 8051 has two pins that are used specifically for transferring and receiving data serially. These two pins are called RXD and TXD and are part of the port 3 group ($P_{3.0}$ and $P_{3.1}$). Pin 11 of the 8051 ($P_{3.1}$) is assigned to TXD and pin 10 ($P_{3.0}$) is designated as RXD. This pins are TTL compatible. The RS 232 Standards are not TTL compatible, therefore they require to make them RS 232 compatible. One such line driver is a MAX 232 chip.

MAX 232

The RS 232 is not compatible with today's microprocessors and microcontrollers, we need a line driver (Voltage convertor) to convert the RS 232's signals to TTL RXD pins. One example of such a convertor is MAX 232 from Maxim corporation. The MAX 232 converts from RS 232 Voltage levels to TTL Voltage level and vice versa.

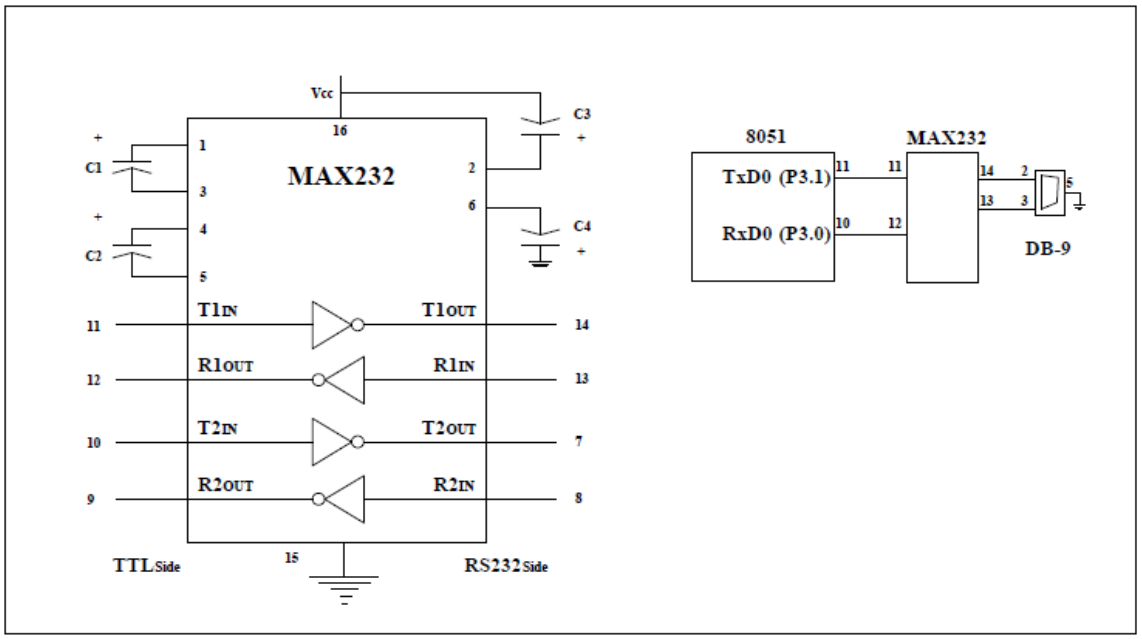


Fig 4.8 (a) Inside MAX 232 (b) Connections to the 8051(Null Modem)

One advantage of the MAX 232 Chip is that it uses a +5V Power's Supply. So a single +5V Power Supply can be used to Power both the 8051 and MAX 232. The MAX 232 has two data of line drivers for transferring and receiving data as shown in fig.4.7 (a). The line drivers used for TXD are called T1 and T2, while the line drivers for RXD are called R1 and R2.

The T1 in Pin is the TTL side and is connected to TXD of the micro controller, while T1 out is the RS 232 side that is connected to the RXD Pin of the RS 232 DB Connector. The R1 line driver has a designator of R1 in and R1 out on Pin numbers 13 and 12 respectively. The R1 in (Pin 13) is the RS 232 side that is connected to the TXD Pin of the RS 232 DB Connector and R1 Out (Pin 12) is the TTL side that is connected to the RXD Pin of the microcontroller. See fig.4.7 (b).In null modem Connection where RXD for one is TXD for the other. MAX 232 requires four capacitors ranging from 1 to 22µF.The most widely used value for these capacitors is 22µF.

4.1.4 8051 Serial Communication Programming

In this Section, we discuss the Serial Communication registers of the 8051 and show how to program them to transfer and receive data serially. Since ISM PC/Compatible Computers are so widely used to Communicate with 8051 based systems, we will emphasize Serial Communication of the 8051 with the COM Port of the Pc. To allow data transfer between the Pc and an 8051 system without any error, we must make sure that the band rate of the 8051 system matches the band rate of the Pc's COM Port. Examine the band rates by going to the windows Hyper Terminal Program and clicking on the Communication Setting Option. The Hyper Terminal Program comes with Windows.

Baud rate in 8051

The 8051 transfers and receives data serially at many different baud rates. The baud rate in the 8051 is Programmable. This is done with the help of Timer 1. First we will lose at the relations between the crystal frequency and the baud rate in the 8051. The 8051 divides the crystal

frequency by 12 to get the machine cycle frequency. In Case, XTAL = 11.0592 MHZ, the machine cycle frequency is 921.6 KHZ (11.0592 MHZ /12 = 921.6 KHZ).

The baud rate in Mode 0 is fixed.

$$\text{Mode 0 baud rate} = \frac{\text{Oscillator frequency}}{12}$$

The baud rate in Mode 2 depends on the value of SMOD in Special function register PCON. If the baud rate is fosc/64. If SMOD = 0, the baud rate is fosc / 64. If SMOD = 1 the baud rate is fosc / 32.

$$\text{Mode 2 baud rate} = 2^{SMOD} \times \frac{(\text{Oscillator frequency})}{64}$$

The baud rates in modes 1and 3 are determined by the Timer 1 overflow rate.

$$\text{Mode 1 and 3baud rate} = \frac{2^{SMOD}}{32} \times (\text{Timer 1 Overflows})$$

If the Timer Operation is in auto related mode.

$$\text{Mode 1 and 3 baud rate} = \frac{2^{SMOD}}{32} \times \frac{(\text{Oscillator frequency})}{12 \times [256 - (TH1)]}$$

Table 4.4 Timer 1 TH1 Register values for various Baud Rates

Baud Rate	THI (Decimal)	TH1 (Hex)
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8

SBUF Register

SBUF is an single 8- bit register used for Serial Communication in the 8051. For a byte of data to be transferred via the TXD line, it must be placed in the SBUF register. Similarly SBUF holds the byte of data when it is received by the 8051 ‘s RXD line.The Serial Port receive and transmit register are both accessed at Serial; function register SBUF. "Writing to SBUF "loads the transmit register and "reading from SBUF "accesses a Physically Separate receive register.

Example

```
MOV    SBUF, # ‘H’          ; load ASCII for ‘H’
MOV   SBUF, A               ; Copy accumulator in to SBUF
MOV   A, SBUF               ; Copy SBUF in to accumulator
```

The Moment a byte is written in to SBUF, it is framed with the start and stop bits and transferred serially via the TXD Pin. Similarly, when the bits are received serially via RXD , the 8051 de frames it by eliminating the stop and start bits, making a byte out of the data received and then placing it in the SBUF.

SCON (Serial Control) register

SCON register is an 8 bit register used to Program the start bit, stop bit and the data bit of data framing among other things. This register contains not only the mode selection bits , but also the 9^{th} data bit for transmit and receive (TB8 and RB8) , and also the serial Port interrupt bits. (TI and RI).

(MSB) D7 D6 D5 D4 D3 D2 D1 D0 (LSB)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Fig 4.9 SCON Register

Table 4.5 Functions of SCON Register (Bit Addressable)

Bit	**Symbol**	**Functions**
SCON.7	SM0	Serial Port mode specifier.
SCON.6	SM1	Serial Port mode specifier.
SCON.5	SM2	It enables the multiprocessor communication feature in mode 2 and 3. In mode 2 and 3 , if SM2 is set to 1, the RI will not be activated if the received 9-th bit (RB8) is '0'. In mode 1, if SM2=1, the RI will not be activated , if a valid stop bit was not received . In mode 0 , SM2 should be '0'.
SCON.4	REN	Enable serial reception , it is set/cleared by software to enable /disable reception.
SCON. 3	TB8	It is the 9-th data bit that will be transmitted in modes 2 and 3 . Set/cleared by software.
SCON. 2	RB8	In modes 2 and 3,this is the 9-th bit that was received. In mode 1,if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.
SCON. 1	TI	Transmit interrupt flag. Set by hardware at the end of transmission. This must be cleared by software.
SCON. 0	RI	Receive interrupt flag. Set by hardware at the end of reception. This must be cleared by software.

Table 4.6 Modes of Serial Communication

SM0	SM1	Mode	Description	Band rate
0	0	0	Shift register	$F_{osc}/12$
0	1	1	8 – bit UART	Variable
1	0	2	9 – bit UART	$F_{osc}/64$ or $F_{osc}/32$
1	1	3	9 – bit UART	Variable

PCON (Power Control)Register

The PCON is an 8-bit register. It's only a byte addressable register of the 8 bits, Some are unused, and some are used for the Power Control Capacitors of the 8051. Serial mode bit is used to determine Serial Communication Port based rate with timer 1. If Timer 1 is used to generate baud rate and SMOD = 1, the baud rate is doubled when the serial Port is used in modes 1,2 or 3.

(MSB) D7 D6 D5 D4 D3 D2 D1 D0 (LSB)

SMOD	---	--- ---	GF1	GF0	PD	IDL

Fig 4.10 PCON Register

Table 4.7 Functions of PCON Register

Bit	Symbol	Functions
PCON. 7	SMOD	Double band rate bit .If it is set to "1", the band rate is doubled, when the serial port is being used in either modes 1,2 or 3.
PCON .6	———	(Reserved)
PCON .5	———	(Reserved)
PCON .4	———	(Reserved)
PCON .3	GF1	General purpose flag bit.
PCON .2	GF0	General purpose flag bit.
PCON .1	PD	Power Down bit. Set this bit activates power down operation.
PCON .0	IDL	Idle Mode bit. Set this bit activates idle mode operation.

4.1.5 PROGRAMMING THE 8051 TO TRANSFER DATA SERIALLY

To Program 8051, to transfer data Serially we have to perform following sequence of actions.

1. Load the TMOD register with the value 20H to use timer 1 in mode 2 (8- bit auto – reload) to set the band rate.
2. Load TH1 to set the desire band rate for serial data transfer.
3. Load SCON register with the value 50H, to use serial mode 1, where an 8-bit data is framed with start and stop bits.
4. Set TR1 to 1 to Start Timer 1.
5. Clear TI with " CLR TI " instructions.
6. Write a character to be sent in to the SBUF register.
7. Check the TI flag bit with instruction "JNB TI , XXXX " to see if an entire character has been transferred completely.
8. Go to Step 5 to transfer the next character.

Importance of the TI flag

1. The byte character to be transmitted is written in to SBUF register.
2. The start bit is transferred.
3. The 8 bit character is transferred one bit at a time.
4. When the stop bit is transferred , the TI flag goes to set condition.
5. Monitoring the TI flag, when it goes to high , load the next byte in to the SBUF register

6. Clear the TI flag by using the instruction "CLR TI"

Example 1

Write an 8051 assembly language program to transfer letter " c " Serially at 9600 band rate Continuously.

```
        MOV   TMOD, # 20 H      ; Timer 1 , mode 2 ( auto – reload )
        MOV    TH1,    # FDH     ; 9600   band rate
        MOV   SCON , # 50 H     ; 8- bit , 1 stop , REW enabled
        SETB  TR1               ; Start Timer 1
START:  MOV   SBUF ,  # "C"     ; Letter  " C " to be transferred
HERE:   JNB   TI,HERE           ; Wait for the last bit to transfer
        CLR  TI                 ; Clear TI for next character
        SJMP START              ; Go to send the character again
```

4.1.6 PROGRAMMING THE 8051 TO RECEIVE DATA SERIALLY

To Program 8051 , to receive data Serially we have to perform following actions.

1.Load the TMOD register with the value 20H to use timer 1 in mode 2 (8- bit auto – reload) to set the band rate.

2. Load TH1 to set the desire band rate for serial data transfer.

3. Load SCON register with the value 50H, to use serial mode 1, where an 8-bit data is framed with start and stop bits.

4.Set TR1 to 1 to start Timer 1.

5.Clear RI with " CLR RI " instructions.

6. Check the RI flag bit with instruction " JNB RI , XXXX " to see if an entire character has been received yet.

7. If RI is set, SBUF has the byte. Save this byte.

8.Go to Step 5 to receive the next character.

Importance of the RI flag

1. It receives the start bit indicating that the next bit is the first bit of the character byte it is about to receive.

2. The 8-bit character is received one bit at a time. When the last bit received, a byte is formed and placed in SBUF.

3. The Stop bit is received. When the Stop bit is received, RI = 1, that an entire character byte has been received.

4. By checking the RI flag bit when it is raised , we know that a character has been received and is sitting in the SBUF register.

5. The received byte must be picked up before it gets over written by an another incoming character.

6. After that SBUF contents are copied in to a memory locations or register.

7. Then clear the RI flag to 0, by using "CLR RI " instruction.

Example 2

Write the 8051 assembly language program to receive bytes of data serially with band rate 9600, 8-bit data and 1 stop bit .Simultaneously send received bytes to Port 2.

```
        MOV   TMOD, # 20 H        ; timer 1 , mode 2 ( auto – reload )
        MOV    TH1,    # FDH      ; 9600   band rate
        MOV    SCON, # 50 H       ; 8- bit , 1 stop , REW enabled
        SETB     TRI              ; Start Timer 1
HERE:   JNB   RI,HERE             ; Wait for character receive completely
        MOV  A ,SBUF              ; Save the received character
        MOV  P2, A                ; Send character to Port 2
        CLR  RI                   ; Get ready to receive next byte
        SJMP HERE                 ; Go to receive next character
```

Doubling the baud rate in the 8051

There are two ways to increase the baud rate of data transfer in the 8051.

1. Use a higher – frequency crystal.
2. Change a bit in the PCON register.

D7	D6	D5	D4	D3	D2	D1	D0
SMOD	---	---	---	GF1	GF0	PD	IDL

Option 1 is not feasible in many situation since the system crystal is fixed. Option 2 is a software way to double the baud rate of the 8051 while the crystal frequency is fixed. This is done with the register called PCON. The PCON register is an 8 bit, some are unused and some are used for the power control capability of the 8051. The bit that is used for the serial communication is D7, the SMOD (Serial mode) bit.

Baud rates for SMOD =0

When SMOD =0, the 8051 divides 1/12 of the crystal frequency by 32 and uses that frequency for Timer 1 to set the baud rate. Note XTAL = 11.0592 MHZ.

Machine cycle frequency =11.0593 MHZ/12 = 921.6 KHZ and 921.6 KHZ /32 = 28,800HZ.Since SMOD =0.

Baud rates for SMOD =1

With the fixed crystal frequency, we can double the baud rate by making SMOD = 1.When the SMOD bit is set to 1, 1/12 of XTAL is divided by 16 (instead of 32) and that is the frequency used by Timer 1 to set the baud rate. Note: XTAL =11.0592 MHZ.

Machine cycle frequency =1.0592 MHZ/12 = 921.6 KHZ and 921.6 KHZ /16 = 57,600HZ.Since SMOD =1.

Table 4.8 Baud Rate Comparison for SMOD = 0 and SMOD = 1

TH1 (Decimal)	(HEX)	SMOD = 0	SMOD = 1
-3	FD	9,600	19,200
-6	FA	4,800	9,600
-12	F4	2,400	4,800
-24	E8	1,200	2,400

Example 3

Write an ALP for transmitting an array of 8 bit data placed in memory locations from 4301H in serial manner in mode 2. The length of array is placed in memory location 4300H. Assume the 9th data bit is always'0'.

```
        ORG    2000              ; Origin at 2000H
        CLR    EA                ; Disable all interrupts
        ANL PCON, #7FH           ; Set SMOD bit to '0'
        MOV SCON, #80H           ; Set Serial Communication in mode 2
        MOV DPTR, 4300H          ; Load DPTR with address 4300H
        MOVX A, @ DPTR           ; Get the length of array in ACC
        MOV R0, A                ; Place the length of array in R0
        CLR TB8                  ; Load the 9th bit data as '0'
NEXT    INC DPTR                 ; To get the next address in DPTR

        CLR TI                   ; Clear Transmit interrupt flag
        MOVX A, @DPTR            ; Get the next data in ACC
        MOV SBUF, A              ; Place the data in SBUF for transmission
WAIT    JNB TI, WAIT             ; Wait for the completion of transmission
                                   of one byte data
        DJNZ R0, NEXT            ; Repeat the above process for next data
HALT    SJMP HALT                ; Hall here
        END                      ; End of program
```

Example 4

Write an ALP for sending a character placed in register A in mode 1, by using timer 1 in mode 2 for band rate.

```
ORG 2000H                ; Set origin at 0000H
CLR EA                   ; Disable all interrupts
ANL PCON, #7FH           ; Set SMOD bit to 0 for ordinary band rate
MOV TMOD, #20H           ; Set timer 1 as an auto – reload mode
MOV TH1, #0F3H           ; TH1 set for divide clock by 13D
MOV TL1, #0F3H           ; TL1 set for divide clock by 13D
```

```
        MOV SCON, #40H          ; Set serial communication in mode 1
        CLR TI                  ; Clear transmit interrupt flag
        SETB TR1                ; Start timer T1
        MOV SBUF, A             ; Place the character in SBUF register ( start operation)
WAIT    JNB TI, WAIT            ; Wait for the end of communication
HALT    SJMP HALT               ; Halt here
        END                     ; End of program
```

Example 5

To write an ALP for taking the data through ports 0, 1 and 2, one after the other and transfer this data continuously.

```
        ORG 2000H               ; Set origin at 0000H
        MOV TMOD,#20            ; Set timer 1 in mode 2

        CLR EA                  ; Clear all interrupts
        MOV TH1, #XXH; Set band rate
        MOV TL1, #XXH           ; Set band rate
        ANL PCON, #7FH          ; Set SMOD bit to '0'
        MOV SCON, #80H          ; Mode 2, Stop bit
        MOV P0, #0FF H          ; Make P0 as input port
        MOV P1, #0FF H          ; Make P1 as input port
        MOV P2, #0FF H          ; Make P2 as input port
        SETB TR1                ; Start timer 1  function
REPT    MOV A, P0               ; Move byte in P0 to A register
        ACALL SEND              ; Call subroutine
        MOV A, P1; Move byte in P1 to A register
        ACALL SEND              ; Call subroutine
        MOV A, P2               ; Move byte in P2 to A register
        ACALL SEND              ; Call subroutine
SEND    MOV SBUF, A             ; Move the data in A register to SBUF
WAIT    JNB TI, WAIT            ; Wait for transmission
        CLR TI                  ; Clear TI flag
        RET                     ; Return to main program
        END                     ; End of program
```

Example 6

Write an ALP for receiving 10D number of characters in serial manner and store the characters from memory location 2100H. Set the timer 1 in auto reload mode and serial reception in mode 1.

```
        ORG 2000H               ; Set origin at 2000H
        CLR EA                  ; Disable all interrupts
        ANL PCON, #7FH          ; Set SMOD bit to 0
```

```
        MOV SCON, #40H          ; Set serial communication in mode 1
        MOV TMOD, #20H          ; Set timer T1 as an 8-bit auto reload
        MOV TH1, #0F3H          ; TH1 set for divide clock by 13D
        MOV TL1, #0F3H          ; TL1 set for divide clock by 13D
        MOV R0, 0AH             ; Load 10 D (0AH) in R0 register
        MOV DPTR, #2100H        ; Load DPTR with starting address
REPEAT  CLR RI                  ; Clear receive interrupt flag
        SETB REN                ; Enable reception
        SETB TR1                ; Start timer T1
WAIT    JNB RI, WAIT            ; Wait for the completion of reception
                                  of one character
        CLR TR1                 ; Stop timer T1
        CLR REN                 ; Stop reception of data
        MOV A,SBUF              ; Move the character to ACC
        MOVX @DPTR, A           ; Store the character
        INC DPTR                ; Increment DPTR
        DJNZ R0, REPEAT         ; Repeat the above process for 10 times
        END                     ; End of program
```

4.2INTERRUPT

4.2.1 8051 INTERRUPTS

An interrupts is an external or internal event that interrupts the microcontroller to inform it that a device needs its service.

Interrupts VS Polling

1. A single microcontroller can serve several devices.
2. There are two ways to do that: interrupts and polling.
3. The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handle.

Steps in exceeding an interrupt

1. Finish current interactions and saves the PC on stack.
2. Jumps to a fixed location in memory depend on type of interrupt.
3. Starts to execute the interrupt service routine until RETI (return from interrupt).
4. Upon executing the RETI the microcontroller returns to the place where it was interrupted. Get pop PC from stack.

Interrupt sources

a)Original 8051 has 6 sources of interrupts .

1. Reset
2. Timer Interrupt 0
3. Timer Interrupt 1
4. External Interrupt 0

5. External Interrupt 1
6. Serial port events (buffer full, buffer empty, etc.,)

b) Enhanced version has 22 sources

More timers, programmable counter array, AOC, more external interrupts, another serial port (UART).

Interrupt vectors

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

Table 4.9 Interrupt vector table

Sl.no	Interrupt	Vector address
1	Reset	0000 H
2	External interrupt 0 (INT 0)	0003 H
3	Timer interrupt 0 (TF 0)	000B H
4	External interrupt 1 (INT 1)	0013 H
5	Timer interrupt 1 (TF1)	001B H
6	Serial port interrupt (RI+TI)	0023 H

All these interrupt sources can be individually enabled or disabled by setting or clearing a bit in serial function register IE. The IE register contains also a interrupt disable bit EA, which disables all interrupts when it is '0'.

Interrupt Structure of 8051 Micro Controller

Upon' RESET ' all the interrupts get disabled, and therefore, all these interrupts must be enabled by a software. In all these five interrupts, if any one or all are activated, this sets the corresponding interrupt flags as shown in fig. All these Interrupts can be set or cleared by bit in some special function register that is Interrupt Enabled (IE), and this in turn depends on the priority, which is executed by IP interrupt Priority register.

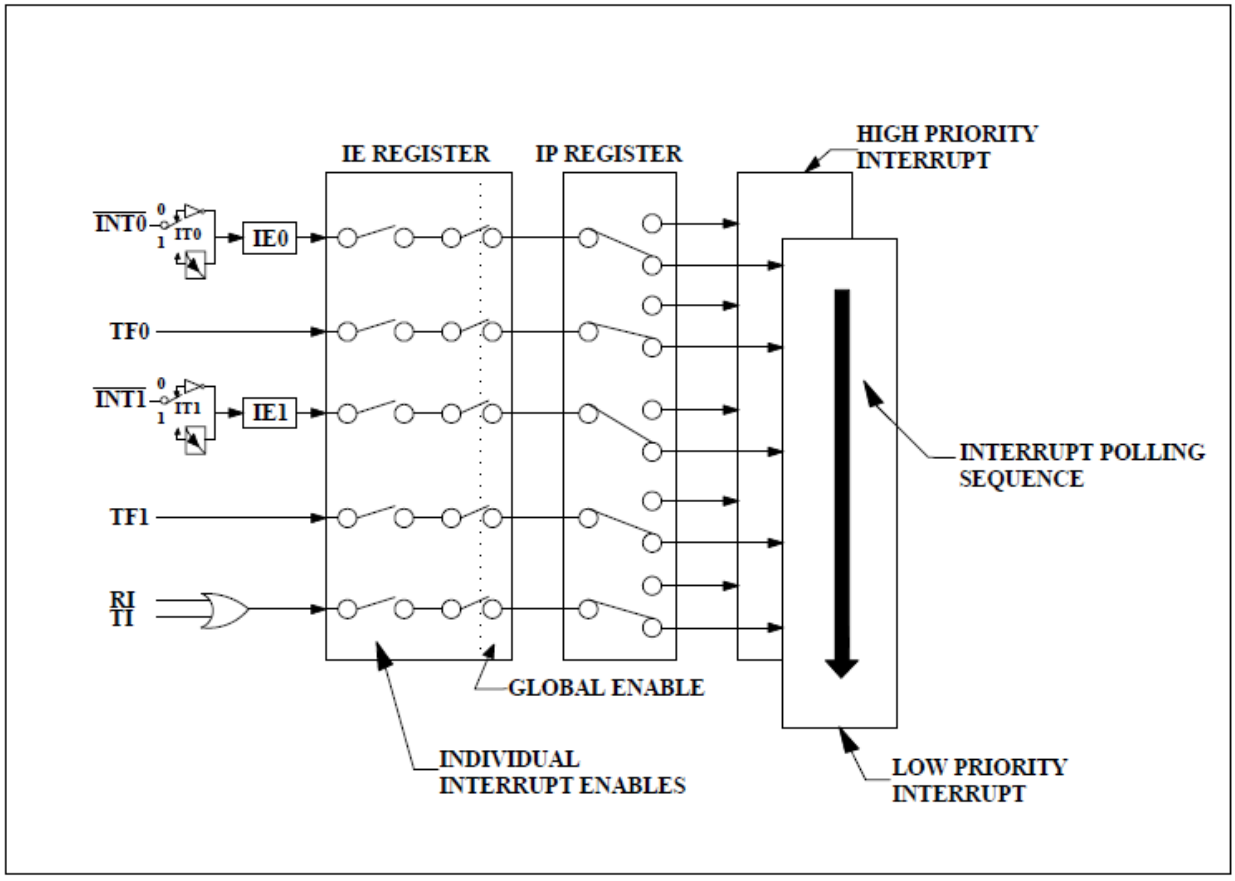


Fig 4.11 Interrupt Structure of 8051 Microcontroller

Interrupt Enable (IE) register

IE register is an 8 bit SFR register .This register is responsible for enabling and disabling the interrupt. It is a bit addressable register in which EA must be set to one for enabling interrupts. This Corresponding bit in this register enables particular interrupt like timer, external and serial inputs. In the below IE register, bit corresponding to 1 activates the interrupt and 0 disables the interrupt.

(MSB) D7 D6 D5 D4 D3 D2 D1 D0 (LSB)

EA	X	X	ES	ET1	EX1	ET0	EX0

Fig 4.12 Interrupt Enable Register

Table 4.10 Functions of Interrupt Enable Register

Bit	Symbol	Functions
IE.7	EA	Disables all interrupts. If EA = 0, no interrupts will be acknowledged. If EA =1, each interrupt source is individually disabled by its corresponding bit.
IE.6	X	Reserved
IE.5	X	Reserved
IE.4	ES	Enables or Disables the serial port interrupt.(1 = Enable or 0 = Disable)
IE.3	ET1	Enables or Disables Timer 1 over flow interrupt.(1 = Enable or 0 = Disable)
IE.2	EX1	Enables or Disables External interrupt 1.(1= Enable or 0= Disable)
IE.1	ET0	Enables or Disables Timer 0 over flow interrupt.(1 = Enable or 0 = Disable)
IE.0	EX0	Enables or Disables External interrupt 0.(1 = Enable or 0 = Disable)

4.2.2 PROGRAMMING TIMER INTERRUPTS

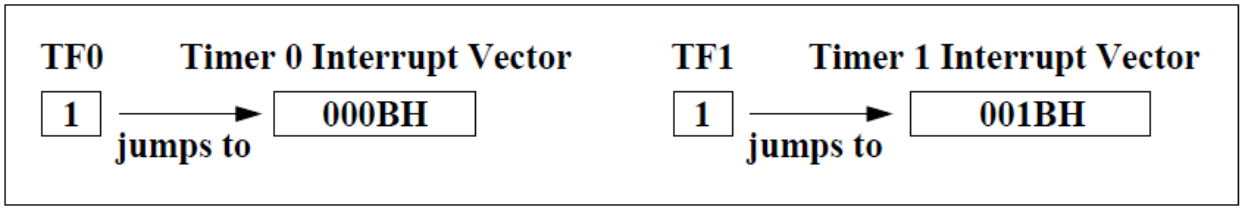


Fig 4.13 TF Interrupt

Timer 0 and Timer 1 interrupts are generated by the timer register bits TF0 and TF1.These timer interrupts Programming involves,

1. Selecting the timer by configuring TMOD register and its mode of operation.
2. Choosing and loading the initial values of TLX and THX for appropriate modes.
3. Enabling the IE registers and corresponding timer bit in it.
4. Setting the timer run bit to start the timer.
5. Writing the subroutine for the timer for time required and clear timer value TRX at the end of subroutine.

Example 7

Write an ALP to switch ON a load connected at port pin 1.3 when timer 0 interrupt occurs. Assume timer 0 operates as timer in mode 1.

```
        ORG 2000H     ; Origin at 2000H
          MOV TMOD, #01H          ; Set timer 0 as timer in mode 1
          MOV IE, #82H            ; Enable timer 0 interrupt only
          CLR P1.3                ; Clear the output load
          CLR TF0                 ; Clear timer 0 overflow flag bit
          MOV TH0, #23H           ; Load TH0 with 23H
          MOV TL0, #CD            ; Load TL0 with CDH
          SETB TR0                ; Start timer 0
WAIT      JNB TF0, WAIT           ; Wait for timer 0 over flow
          END                     ; End of program
000BH :                           ; Timer 0 interrupts subroutine
          SETB P1.3               ; ON the load at P1.3
          CLR TR0                 ; Stop the timer function
          RETI                    ; Return from interrupt
```

Example 8

Write an ALP to produce a square wave signal at port pin P1.1 by using timer T1 interrupt and set the timer 1 in mode 2.

```
        ORG 2100H              ; Origin at 2000H
        MOV TMOD, #20H         ; Set timer T1 as an 8 bit auto reload mode
        MOV TH, #7F            ; Load 7FH in TH1
        MOV TL1, #7FH          ; Load 7FH in TL1
        MOV IE,#88H            ; Enable timer 1 interrupt only
        CLR TF1                ; Clear timer T1 overflow flag bit
REPEAT  SETB TR1               ; Start timer 1 function
WAIT    JNB TF1, WAIT          ; Wait for timer T1 over flow
        SJMP REPEAT            ; Repeat the process
        END                    ; End of program

001BH ;                        ; Timer 1 interrupt subroutine
        CLR TR1                ; Stop the timer function
        CPL P1.1               ; Complement the previous output
        RETI                   ; Return from interrupt
```

Example 9

Write an ALP by using timer 0 interrupt in mode 3 to switch ON a specific load connected at port pin P1.1 after counting 100D number of clock pulses.

The initial values to be loaded for counting 100D number of pulses = 256D -100D =156 D.

It's equivalent to hexadecimal value = 9 CH

```
        ORG 2100H; Origin at 2000H

        MOV TMOD, #20H          ; Set Timer 0 as counter in mode 3
        CLR P1.1    ; OFF the load
        CLR TF0    ; Clear counter 0 overflow flag bit
        MOV TL0, #9CH           ; Load the initial value of count
        MOV IE, #82H            ; Enable timer 0 interrupt only
        SETB TR0                ; Start counter 0 function
WAIT    JNB TF0, WAIT           ; Wait for counter 0 over flow
        END                     ; End of program
001BH ;                         ; Timer 0 interrupt subroutine

        SETB P1.1               ; ON the load
        CLR TR1                 ; Stop the counter function
        CPL P1.1                ; Complement the previous output
        RETI                    ; Return from interrupt
```

4.2.3 PROGRAMMING EXTERNAL HARDWARE INTERRUPTS

8051 Microcontroller consists of two external hardware interrupts: INT0 and INT1 (Pin 12 and Pin 13). These are enabled at port Pin 3.2 and Pin 3.3. These can be edge triggered or level triggered. In level triggering, the low at Pin 3.2 enables the interrupt, while at Pin 3.2 the high to low transition enables the edge triggered interrupt. This edge triggering or level triggering is decided by the TCON register that has been discussed below.

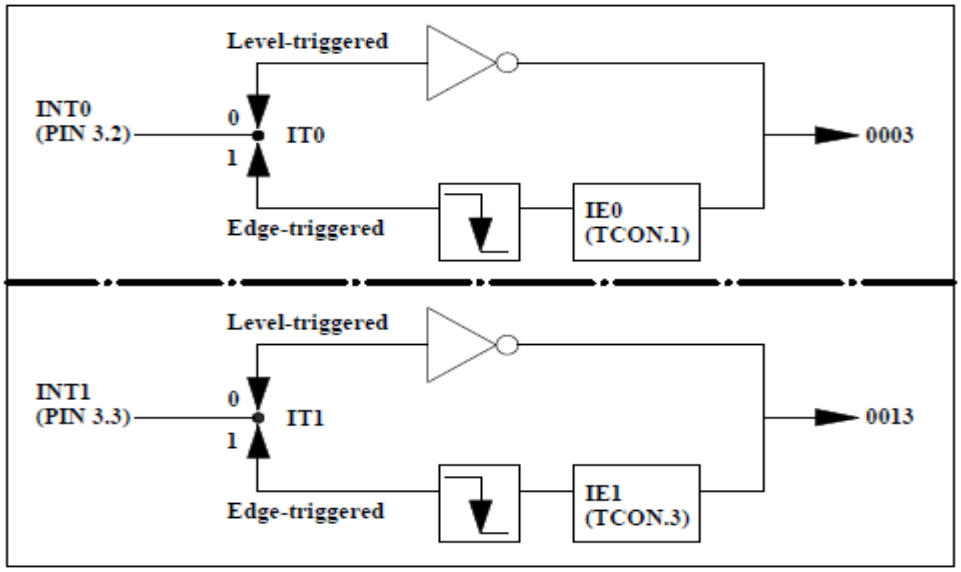


Fig 4.14 Activation of INT0 and INT1

TCON Register

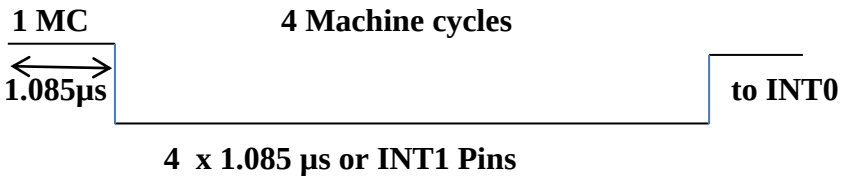
The TCON register specifies the type external interrupt to the 8051 micro controller, as shown in the fig. The two external interrupts, whether edge or level triggered, Specify by this register by a set, or cleared by appropriate bits in it. And, it is also a bit addressable register.

D7	D6	D5	D4	D3	D2	D1	D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Fig 4.15 TCON Register

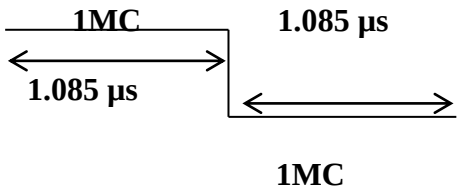
Programming External Hardware Interrupts Procedure in 8051 is as follows

1. Enable the corresponding bit of external interrupt in IE register.
2. If it is level triggering, just write the subroutine appropriate to this interrupt , or else enable the TCON register bit corresponding to the edge triggered interrupt whether it is INT0 and INT1.



Note: On RESET, IT0 (TCON.0) and IT1 (TCON.2) are both low, making external interrupts level-triggered.

Fig 4.16 Minimum Duration of the low level triggered interrupt (XTAL = 11.0592 MHz)



Note: Minimum pulse duration to detect edge- triggered interrupts.

Example 10

Write an ALP to produce 10D cycles of square wave signal at port Pin P1.1 by using external interrupt 0 only after multiplying the content of registers R0 and R1.

10D number of square wave signal contains 20D (14H) number of high level and low level pulses.

```
ORG 2000H          ; Origin at 2000H
MOV R7, #14H       ; Load 14 H at R7 register
CLR IE0            ; Clear external interrupt 0 flag bit
CLR EA             ; Disable all interrupts
SETB IT0           ; Set triggering of external interrupt 0
                     at falling edge
 MOV A, R0         ; Move the data in R0 to ACC
 MOV B, R1         ; Move the data in R1 to B register
 MUL AB            ; Multiply the data
```

```
        MOV IE, #81H          ; Enable external interrupt 0 only
WAIT    JNB IE0, WAIT         ; Wait for external interrupt 0 only
HALT    SJMP HALT             ; Halt here
        END                   ; End of program
        0003H                 ; External interrupt 0 subroutine
        SETB P1.1             ; Make high level output at P1.1
FIRST   MOV R3, #FFH          ; Load the delay loop value
WAIT    DJNZ R3, WAIT         ; Wait for the completion of loop
        CPL P1.1              ; Complement the previous output
        DJNZ R7, FIRST        ; Decrement the number of pulses
        RETI                  ; Return from interrupt
```

Example 11

Write an ALP to divide the data placed in register R0 by an another data placed in register R1 and store the result in memory locations 410H and 4102 H after getting an external interrupt 1 signal.

```
        ORG 2100H             ; Origin at 2100H
        CLR EA     ; Disable all interrupts
        MOV DPTR, #4101H      ; Load the address in DPTR
        SETB IT1              ; Set external interrupt 1
                                as falling edge triggering
        CLR IE1               ; Clear external interrupt 1 flag bit
        MOV A, R0             ; Place first data in ACC
        MOV B, R1             ; Place second data in B register
        DIV AB                ; Divide the values
        MOV IE, #84H          ; Enable external interrupt 1 only
WAIT    JNB IE1, WAIT         ; Wait for external interrupt 1 only
        END                   ; End of program

0013H ;                       ; External interrupt 1 subroutine
        MOVX @ DPTR,A         ; Store the quotient
        INC DPTR              ; Increment DPTR
        MOV A,B               ; Move remainder to ACC
        MOVX @ DPTR,A         ; Store the remainder
        RETI                  ; Return from interrupt
```

4.2.4 PROGRAMMING THE SERIAL COMMUNICATION INTERRUPT

Serial Communication interrupts come in to picture when there is a need to send or receive data. Since one interrupt bit is set for both TI (Transfer Interrupt) and RI (Receiver Interrupt) flags. Interrupt service routine must examine these flags to know the actual interrupt.

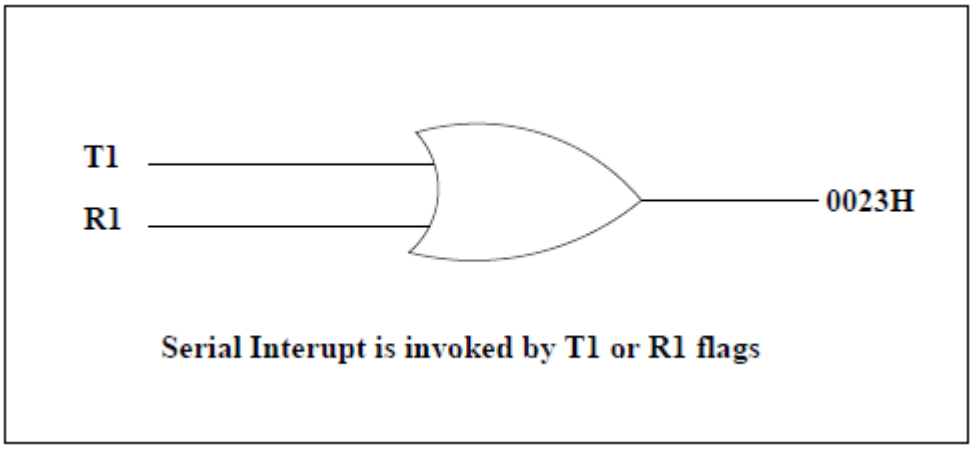


Fig 4.16 Single Interrupt for both TI and RI

The logical or Operation of these two flags (RI and TI) causes this interrupt, and it is cleared by the software alone. Here, a special register SCON is used for controlling communication operation by enabling the corresponding bits in it.

1. Configure the IE register for enabling serial Interrupt.
2. Configure the SCON register for receiving or transferring operation.
3. Write subroutine for this interrupt with appropriate function and clear TI or RI flags with in this routine.

Table 4.11 Interrupt Flag Bits for the 8051

Interrupt	Flag	SFR Register Bit
External 0	IE0	TCON.1
External 1	IE1	TCON.3
Timer 0	TF0	TCON.5
Timer 1	TF1	TCON.7
Serial port	T1	SCON.1

Example 12

Write an ALP by using Serial Communication interrupt to transmit a character 20H times continuously in mode 1 with timer T1 in auto-reload mode.

```
SERIAL:   ORG 0023H          ; Place serial interrupt program here
          CLR TR1            ; Stop timer 1 function
          CLR TI             ; Reset TI flag for next interrupt
          RETI               ; Return from interrupt to wait loop
MAIN :    ORG 2100H          ; Origin at 2100H
          CLR EA             ; Clear all interrupts
          ANL PCON, #7FH     ; Set SMOD bit to 0 for band rate x 32 rate
          MOV TMOD, #20H     ; Set timer T1 as an 8-bit auto reload mode
          MOV TH1, #0F3H     ; TH1 set for divide clock by 13D
          MOV TL1, #0F3H     ; TL1 set for divide clock by 13D
          MOV SCON, #40H     ; Set UART to mode 1
          MOV R0, #20H       ; Load 20H in R0 register
          MOV IE, #90H       ; Enable serial interrupt only
NEXT      SETB TR1           ; Start timer T1
          MOV SBUF,"#U "     ; Load the data character in SBUF
WAIT      JNB TI, WAIT       ; Wait for the completion
                               of one character
          DJNZ R0, NEXT      ; If not completed go to NEXT
          END                ; End of program
```

Example 13

Write an ALP by using serial communication interrupt to receive 10D (0AH) number of characters in mode 1 and store them from memory location 4200H with timer T1 in 8 bit auto reload mode.

```
RECEIVE :ORG 0023H          ; Set receive interrupt here
         CLR REN            ; Stop reception function
         CLR TR1            ; Stop timer T1 function
         CLR RI             ; Reset RI for next interrupt
         MOV A, SBUF        ; Move the data to ACC
         MOVX @DPTR, A      ; Store next character
         INC DPTRB          ; Increment DPTR
         RETI               ; Return from interrupt
```

```
MAIN :      ORG 4100H              ; Main program here
            ANL PCON, #7FH         ; Set SMOD bit to 0 for band rate x 32 rate
            ANL TMOD, #0FH         ; Alter timer T1configuration only
            ORL TMOD, #20H         ; Set Timer T1as an 8 bit auto- reload  mode
            MOV TL1, #0F3H         ; TL1 set for divide clock by 13D
            MOV TH1, #0F3H         ; TH1 set for reloading
            MOV SCON, #40H         ; Set UART to mode 1
            MOV R0, #0AH           ; Place number of characters in R0
            MOV IE, #90H           ; Enable serial interrupt only
NEXT        SETB REN               ; Enable reception
            SETB TR1               ; Start timer function
WAIT        JNB RI, WAIT           ; Wait for reception of one character
            DJNZ R0, NEXT          ; Receive next character
            END                    ; End of program
```

4.2.5 INTERRUPT PRIORITY IN 8051

When the 8051 is powered up, the priorities are assigned to five Interrupts except Reset. All they are vectored interrupts. The interrupt source can also be individually programmed one of two priority levels by setting or clearing a bit in special function register IP. The two priority levels are high level priority and low level priority.

Interrupt Priority (IP) register

Interrupt Priority register is an 8 bit addressable register. It is also possible to change the priority levels of the interrupts by setting or clearing the corresponding bit in the Interrupt Priority (IP) register as shown in fig. This allows the low priority interrupt to interrupt the high priority interrupt, but prohibits the interruption by another low priority. Similarly, the high priority interrupt cannot be interrupted. If these interrupt priorities are not programmed, the microcontroller executes in pre defined manner and it order is INT0, INT1 ,TF1 and SI.

(MSB) D7 D6 D5 D4 D3 D2 D1 D0 (LSB)

X	X	X	PS	PT1	PX1	PT0	PX0

Fig 4.17 Interrupt Priority Register

Table 4.12 Functions Interrupt Priority Register

Bit	Symbol	Functions
IP.7	X	Reserved
IP.6	X	Reserved
IP.5	X	Reserved
IP.4	PS	Define the Serial port interrupt priority level. If PS =1, it becomes higher priority level.
IP.3	PT1	Define the Timer "1" interrupt priority level. If PT1 =1, it becomes higher priority level.
IP.2	PX1	Define the External interrupt"1" priority level. If PX =1, it becomes higher priority level.
IP.1	PT0	Define the Timer "0" interrupt priority level. If PT0 =1, it becomes higher priority level.
IP.0	PX0	Define the External interrupt"0" priority level. If PX =1, it becomes higher priority level.

A low level priority interrupt can itself be interrupted by a high level priority interrupt, but not by another low level priority interrupt. A high level priority interrupt cannot be interrupted cannot be interrupted by any other interrupt source.

If two requests of different priority levels are received simultaneously, the request of the higher priority level is serviced first. If requests of the same priority levels are received simultaneously, an interrupt polling sequence determines which request is service. Thus within each priority level there is a second priority structure determined by the polling sequence as follows.

Table 4.13 8051Interrupt Priority Upon Reset

S. No	**Source**	**Priority within level**
1	External interrupt 0 (INT0)	Highest ↓
2	Timer interrupt 0 (TF0)	
3	External interrupt 1 (INT1)	
4	Timer interrupt 1 (TF1)	
5	Serial port interrupt (RI + TI)	Lowest

Review Questions

PART – A

1. Define Full duplex transmission.
2. What are the register used in Serial Communication ?
3. What is the voltage levels used in RS 232 Serial interface standards?
4. Define baud rate.
5. What is the function of REN bit in SCON register?
6. Specify the different modes of Serial Communication.
7. List the interrupts available in 8051.
8. Write the functions of SMOD bit in PCON register.
9. What are the external hardware interrupts available in 8051?
10. What are the SFR register used in interrupt Operation.

PART – B

1. Write the features and limitations of RS 232.
2. Why are drivers used in between RS 232 and Microcontroller?
3. How will you double the baud rate in 8051?
4. Mention the baud rate of Serial Communication in mode 2 and mode 3.
5. What are the conditions required for initiating a Serial reception?
6. Write the importance of the TI Flag.
7. Write the two activation levels for external hardware interrupt.
8. Draw PCON register.
9. Specify the vector address and Priorities of Interrupts in 8051.
10. What are the functions of RXD and TXD Pins in 8051?

PART – C

1. What is RS 232 ? Explain the interfacing of RS 232 with 8051.
2. Explain SCON and PCON registers.
3. Explain in detail about the Programming 8051 to transfer and receive data Serially.
4. Write an ALP for Sending a character placed in register ″C″ in mode 1,by using timer1 in mode 2 for baud rate.
5. Explain the baud rates of Serial Communication in 8051.
6. Draw and Explain the Interrupt Structure of 8051.
7. Explain the Programming of external hardware interrupts and Serial Communication interrupts.
8. Explain IP and IE registers.
9. Write an ALP for receiving 10D number of characters in Serial manner and store the characters from memory location 2100H. Set the timer 1 in auto- reload mode and Serial reception in mode 1.
10. Explain the Interrupts Priorities in 8051.

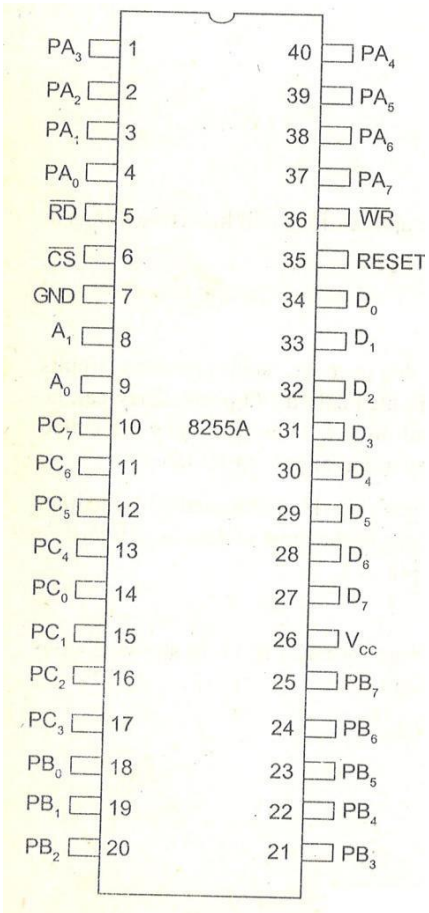
UNIT – 5 INTERFACING TECHNIQUES

5.1IC 8255 (Programmable Peripheral interface)

8255 Pin details and signal diagram

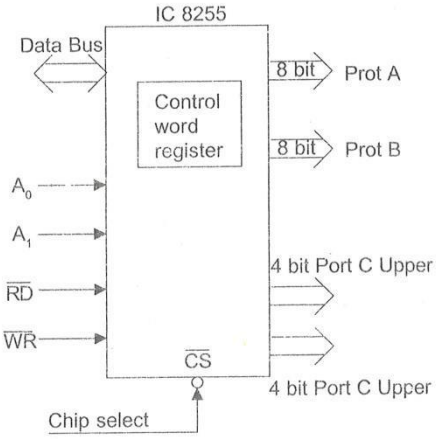
The Intel 8255 is one such peripheral interface chip. It can be used with the Intel microprocessor or microcontroller. The 8255 is one of the most widely used interface devices for expanding number of input/ output pins.

It is a 40 pin DIP IC. It requires +5V DC power Supply for its operations . The pin out diagram and signal diagram of 8255 A are shown in the fig 5.1



(a) Pin diagram

(b) signal diagram



$D_7 - D_0$	Data Bus (Bidirectional)
Reset	Reset Input
CS	Chip select
RD	Read input
WR	Write input
$A_0 - A_1$	Port adddress
$PA_7 - PA_0$	Port A (8 bit)
$PB_7 - PB_0$	Port B (8 bit)
$PC_7 - PC_0$	Port C (8 bit)
V_{cc}	+5 Volts
GND	0 Volts

(c) Signal configuration

Fig 5.1 Programmable Peripheral interface 8255

5.2 FUNCTIONAL BLOCK DIAGRAM of8255

The functional block diagram of 8255 is shown in fig 5.2 It contains Data bus buffer, Read/ Write control Logic Unit and I/O Ports.

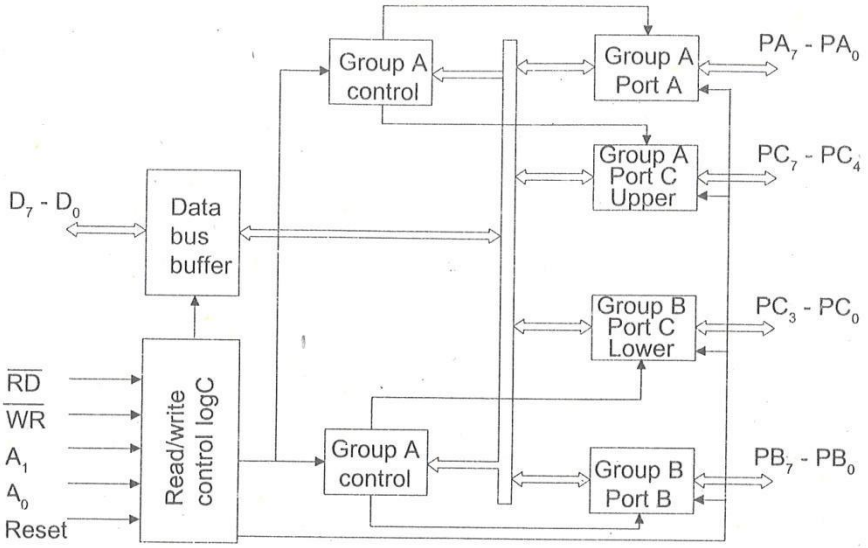


Fig 5.2 Functional block diagram of IC

(i) Input/ Output Ports

The 8255 consists of three numbers of ports namely A,B and C. Each one of them is an 8 bit port. However port C can be divided into two 4 bit ports (port C upper, port C Lower) and used separately.

For the convenience of Programming the ports are grouped as Group A and Group B

Group A = Port ($PA_7 - PA_0$)

Port C Upper ($PC_7 - PC_4$)

Group B = Port B ($PB_7 - PB_0$)

Port C Lower ($PC_3 - PC_0$)

ii) Data bus Buffer:

It is a bidirectional 8 bit buffer. It is used to interface the 8255 with the system data bus. The data, control world and status information signal in between the microcontroller and 8255 are communicated only through data bus buffer.

iii) Control Logic:

RD ($\overline{\text{Read}}$) : When this signal in low, the microcontroller reads data from the selected I/O ports of 8255

WR($\overline{\text{Write}}$) : When this signal is low, the microcontroller writes data into the selected I/O port or the control register.

Reset :Reset the all: all ports in the input mode. It is an active high signal.

$\overline{\text{CS}}$ (chip select) : It enables the Communication between the8255 and microcontroller.

A1-A0 : These lines are Used to address the three ports and control word Register (CWR)

Fig 5.3 (a) Table : port selection signals

A1	*A0*	*Selected*
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Control word register

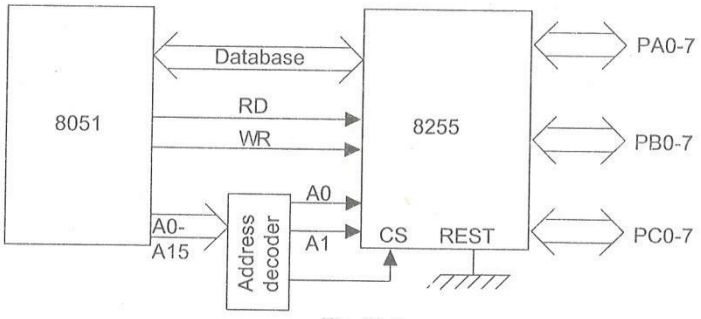


Fig 5.3 (b) Interfacing 8255 with 8051

5.1.2.1 Control Word Register :

It is a 8 bit Register . An 8 bit binary word present in the control register is called control word. The control word Specifies the function for each I/O Port.

Modes of8255

The operation of the ports can be classified into two broad groups.

- I/O Mode
- Bit set/ reset mode (BSR)

I/O Mode

I/o mode offers three specific Modes of Operation. There are

- Mode 0 - Simple I/O Mode
- Mode 1 - Handshake Mode
- Mode 2 - Bidirectional Mode

Control world format for I/O Mode

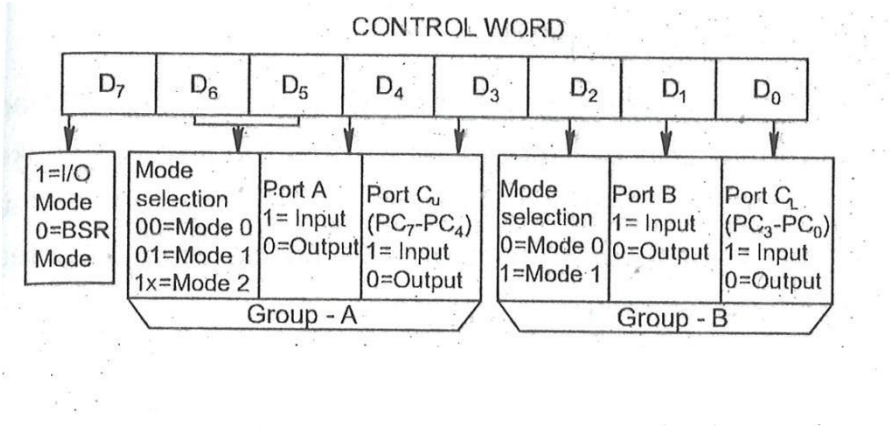


Fig 5.4 Control world format for I/O Mode

Mode O – Simple I/O mode

- In this mode all the ports can be programmed either as input or output port.
- Port C can be divided into two 4 bit ports, the C Lower and C Upper each of them can be set independently for input or output operation.

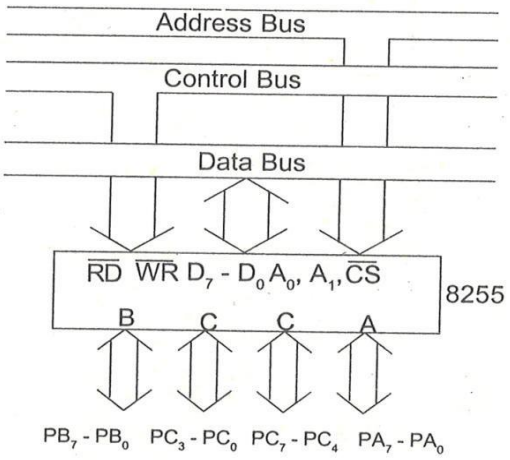


Fig 5.5 Mode O – Simple I/O mode

Mode 1 : Hand shake Mode

Shown in the fig 5.6 Mode 1 : Hand shake Mode

- In this mode 1 the port A and port B can be set for input or output Operation.
- The Port C are used as control Signals. These Control Signals are used for handshaking.

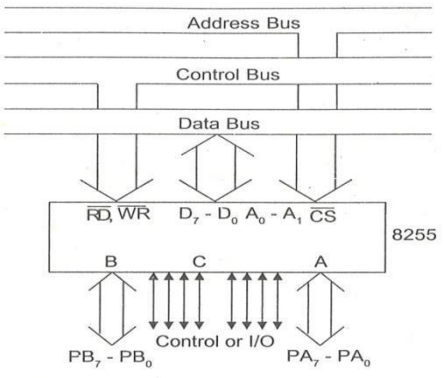


Fig 5.6 Mode 1 : Hand shake Mode

Mode 2 - Bidirectional Mode

- In this mode port A only act as a bidirectional data transfer Port.
- Poet B may be operated as in mode 0 or Mode1
- Port C- 5 bits are used control signals for port A
- Port C – Remaining 3 bits are Used in mode 0 or as handshake signals for port B
- In this mode data transfer is done in both directions between microcontroller and peripheral devices. Shown in the fig 5.7 mode 2 – bidirectional mode

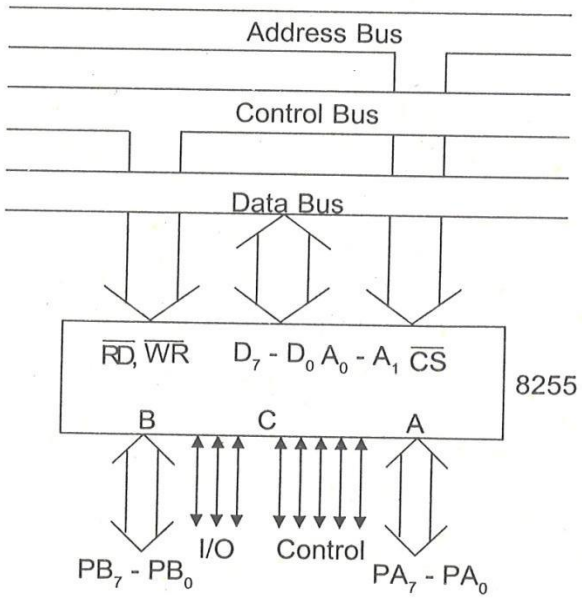


Fig 5.7 Mode 2-Bidirectional Mode

Bit Set/ Reset Mode (BSR Mode)

- This mode is related to only port C. The bits of port C can be controlled directly by the microcontroller. A control world with bit D7 = 0 in called a BSR Control word.
- Control word format for BSR mode as shown in the fig 5.8.

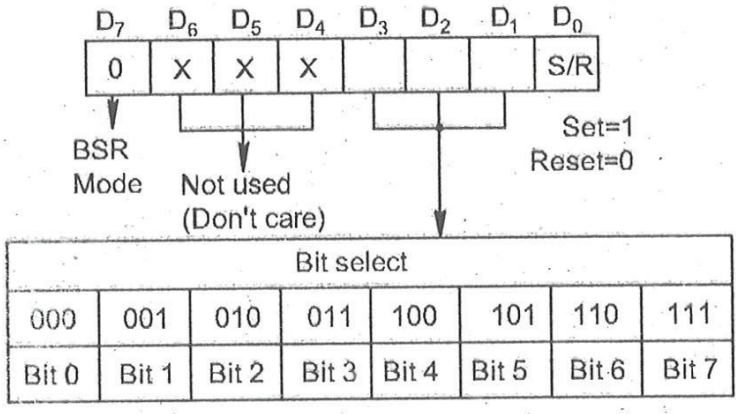


Fig 5.8 Control word format for BSR mode

5.2 INTERFACINGTECHNIQUES

Designing logic circuits and writing instructions to enable the microcontroller to communicate with peripherals (I/O devices- input devices – keyboard, switches, and A/D converters. Output devices – seven segment LED display, D/A converters, Printers and video monitors) is called interfacing.

5.2.1 INTERFACING EXTERNAL MEMORY TO 8051

- The 8051 has 256 bytes of internal data memory (RAM) for data storage. 4Kbytes of internal ROM is also available to store the Program.
- If the internal Memory is not sufficient for storage, it can able to connect the external memory with8051.
- We can able to connect 64 K bytes of data memory (RAM) and 64Kbytes of Program memory (ROM)externally.

The interfacing diagram of 16 K bytes of EPROM and 8Kbytes of RAM with Microcontroller 8051 in shown in fig 5.9 External memory connection with 8051.

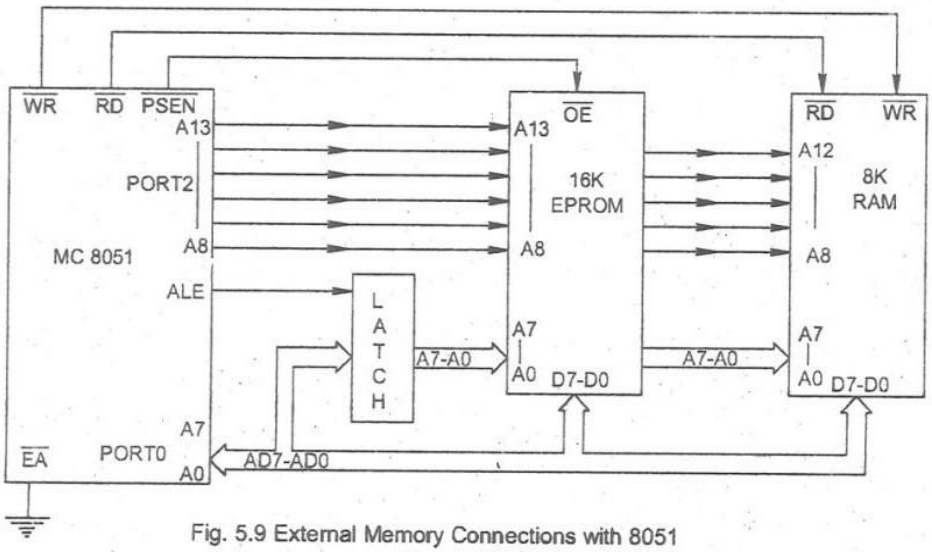


Fig. 5.9 External Memory Connections with 8051

- When ALE in enabled the port 0 Latch the address signal. Otherwise port 0 has a data.
- If the memory access is from external ROM, the $\overline{\text{PSEN}}$ pin will go low
- If the memory access is for a RAM byte
  - $\overline{\text{WR}}$ pin Low - When data flow from data bus to RAM
  - $\overline{\text{RD}}$ pin Low - When data flow from RAM to data bus
- The $\overline{\text{WR}}$ and $\overline{\text{RD}}$ signals are alternate Uses for port 3 pins 16 and17
- The Port 0 is used for the lower address bytes and data
- Port 2 is used for upper address bits

The 8051 accesses external RAM whenever MOVX type Instructions are executed.

External ROM accesses whenever EA $\overline{\text{pin}}$ is connected to ground or when the program counter contains an address in between 1000H and FFFFH.

5.2.2 8051 INTERFACING WITH THE 8255

8051 have four I/O ports (Port 0, Port 1 , Port 2, Port 3) for interfacing the peripherals . If not enough these I/O ports we can expend the I/O ports capability of 8051 interfacing with 8255. The interfacing diagram of 8255 with 8051 in shown in Fig 5.10 8051 interfacing with the 8255.

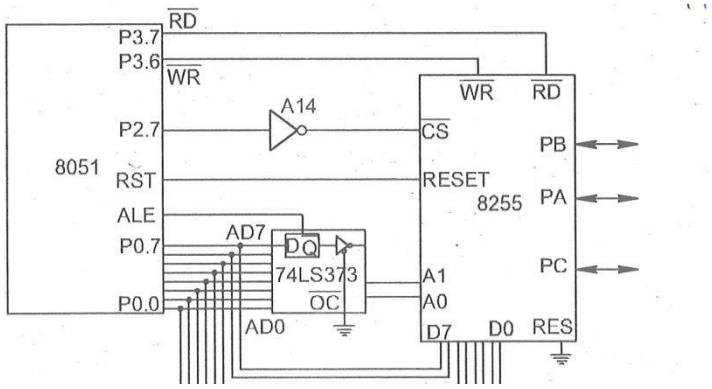


Fig 5.10 : 8051 Connection to the8255

The bidirectional data bus D_0- D_7 of 8255 in connected to port 0 of 8051.The address latch is used / data bus (AD_0- AD_7).

The control lines WR and RD of 8255 in connected to the WR and RD lines of 8051. By Using the MOVX instruction the microcontroller can access the ports and control word Register of8255.

5.2.3. ASM PROGRAMMING

Assembly Language Programming :

Each family of processor has own set of institution for handling various operation. These set of institution are called “Machine language Institution.

Processors Understand only machine language institution which are strings of 1‟s and 0‟s . However machine language using computer for software development. So the low level assembly language is designed for a specific family of processors that represents various instructions in symbolic code and more.

Understandable form :

Advantage of Assemble language

1. It requires less memory and execution time.
2. It allows hardware – specific complex jobs in a easier way.
3. It is suitable for time critical jobs.

5.2.4 RELAYS

The electro mechanical relays have been used for many years in industry to control high dc or ac voltages and currents. Relays also provide isolation between the controller and the circuit under control. Relays are made up of three basic components:

1. Electromagnet
2. Spring
3. Some Contacts

Relays have two states – Open and close. A contact can be either normally open or normally close. The State of the contact can be changed by passing specified amount of current through the coil of the electromagnet.

Relay interfacing with microcontroller 8051

For energize the relay the voltage required for the coil of the electromagnet is normally +5V (or) +12V. On the other hand contact voltage can be 100 or more. Interfacing diagram of relay with 8051 is shown in fig 5.11 interfacing diagram of relay with 8051.

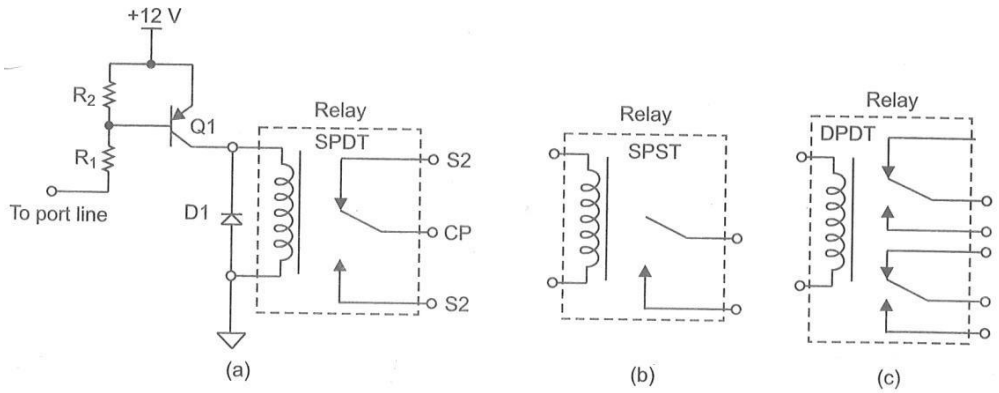


fig 5.11 Interfacing diagram of relay with 8051

Micro controller pins could not produce sufficient voltage (or) current to driver the relay. For this reason, we place a driver or a power transistor in between the microcontroller and the relay. Here SPDT (Single pole Double – throw) type relay is Interface with help of driver circuit. Single – refer common point, double - refer to two contact paints. One between CP and S1 is normally close and the contact between CP and S2 is normally Open, When the Electromagnet is energized by passing the desired amount of current through the coil the conditions reverses.

Program

```
        ORG 4100 H          ; Origin at 4100 h
START   SETBP1.0            ; Set 1 at port pin p1.0
        ACALL Delay         ; Call the delay routine
        CLR P1.0            ; Set the Port Pin P1.0
        A CALL Delay        ; Call the delay routine
        SJMP  START         ; Jump to the start
        END
```

DELAY PROGRAM

```
      MOV R0#FF
L3    MOV R1# F0
L2    MOV R2 # FE
L1    DJNER2      L1
      DJNZR1      L2
      DJNZR0      L3
        RET
```

5.2.5 INTERFACING AND OPTO COUPLER:

An Opto Coupler used to isolate the port line of the microcontroller from the relay circuit as shown in the Fig 5.12 interfacing and Opto coupler.

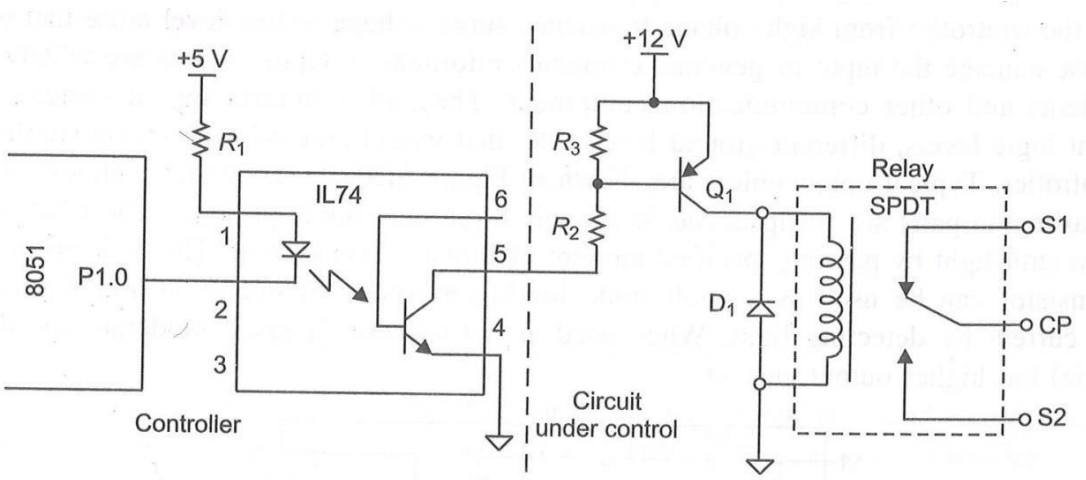


Fig 5.12 Interfacing And Opto Coupler

Opto coupler is a one of the solid state Relay. In this relay there is no coil, spring (or) mechanical contact. The entire Relays is made out of semiconductor materials. The switching time of solid state relay is faster than that of Electromechanical relay. So solid state relays are ideal for microcontrollers. They are widely used in controlling pumps, alarms, and other power applications. It is also used in communication equipment.

Program

```
ORG 2100H              ; Origin at 4100 h
REPT  MOV R1,#ABH      ; Load the value ABh  at R1 Register
 L2     MOV R1,#FFH    ; Load the value FFh  at R1 Register
L1      DJNZ R2,L1     ; Wait the complete in inner loop
        DJNZ R2,L2     ; Wait the complete in inner loop
        CPL P1.0       ;Complement the previous output
        SJMP  REPT     ;Repeat the Loop
         END
```

5.2.6 SENSOR :

Sensor is a energy conversion devices**.** Which receive physical data such as temperature, pressure, light intensity, Speed or flow and generate electrical signals such as voltage, current,

resistance or capacitance depending on the type of sensor used . Sensors also called as Transducers. The function of a Sensors is shown in the fig 5.13 interfacing sensor with 8051.

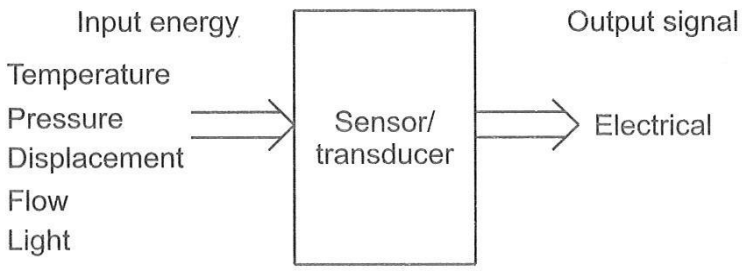


Fig 5.13 Interfacing Sensor to Microcontroller 8051

Temperature is one of the most important parameters that need to be monitored or controlled in a variety of application. Different types of temperature transducers are commercially available.

(i)LM34 - Series of transducers are Precision Fahren height temperature Sensors.

(ii) LM35 - Series of transducers are precision Celsius temperature sensors giving output of 10mv for each degree of Celsius temperature.

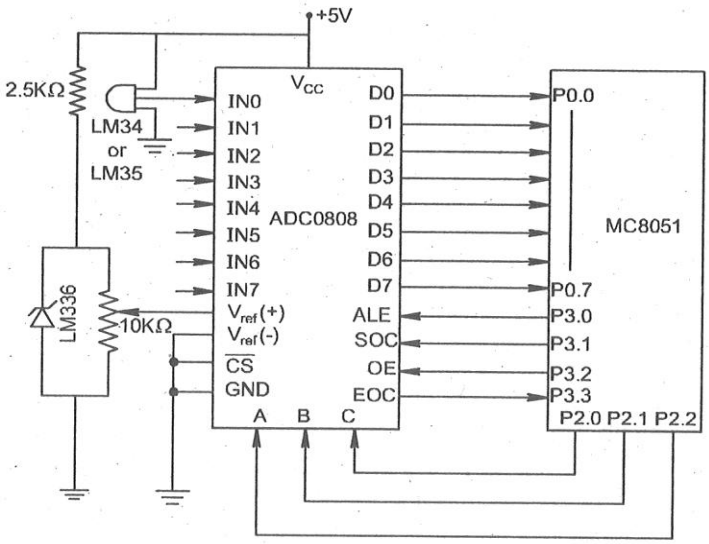


Fig 5.14 Interfacing diagram ADC0808 with 8051

The connection diagram for interfacing the temperature sensor LM35 with microcontroller through an ADC is shown in the Fig 5.14 . The ADC 0808 has 8 bit resolution, with maximum of 256 (2^8) steps. The LM 35 produces 10mV for every degree of temperature change. We can condition V_{in} of the ADC 0808 to produce a V_{out} of 2.550mV (2.55V) for full scale output. Therefore in order to produce the full scale V_{out} of 2.55V for the ADC 0808 correspond directly to the temperature as mentioned by LM35. The zener diode is used to give a steady voltage across the 10 K pot, which overcomes any fluctuations in the power supply.

LM 35 is connected directly to the IN0 input pin of ADC 0808. Port 0 pins of MC 8051 are directly connected to the digital output terminals of ADC 0808.

The port pin P3.0 is connected to ALE, the port pin P3.1 is connected to SOC, the port pin P3.2 is connected to OE and the port pin P3.3 is connected to EOC of 0808 ADC respectively. The port Pins P2.0, P2.1 and P2.2 are connected to the address lines of A,B and C Respectively .

Program :

```
        ORG4100H        ;   Originat4100H
        MOVP0,#FFH      ;   Make port 0 as input
        SETBP3.3        ;   Set EOC as input port pin
        MOVP2#00H       ;   Select channel 0 (INO)
        SETBP3.0        ;   Make ALE as High, enable ALE
        CLR P3.0        ;   Again make ALE as Low
        SETBP3.1        ;   Make SOC as High, initiate start of Conversion
        CLR P3.1        ;   Again make SOC as low
WAITJNBP3.3.WAIT        ;    wait for the completion of conversation Process
        SETBP3.2        ;    Enable the output
        MOVA.P0         ;    Read data throughport0 CLR
        P3.2            ;    Clear OE
        ACALL CONVER    ;    Hexadecimal to ASCII Conversion
        ACALL DISPLAY   ;    Go To Display Subroutine Conversion
        MOV B, #0AH     ;     Load The OAH In B Register
        DIV A,B         ;     Divide By using OAH
        MOV RO,B        ;     Store The I's In Ro Register
        DIV A,B         ;     Divide By Using OAH
        MOV R1,B        ;     Store The 10's In R1 Register
```

```
MOV R2,A        ;       Store The 100's In R2 Register
RET             ;       Return  To Main Program

DISPLAY

MOV P3,RO       ;       Display 1's
ACALL DELAY     ;       Call To Delay Subroutine
MOV P3,R1       ;       Display In 10's
ACALL DELAY     ;       Call To Delay Subroutine
MOV P3,R2       ;       Display In 100's
ACALL DELAY     ;       Call To Delay Subroutine
RET
```

5.2.7 ADC INTERFACING

Digital computers (or) microcontrollers use binary values, but we get electrical signal from sensor (Transducers) are Analog signal. The Transducers convert the physical Quantifies like Temperature, Pressure, speed etc, into electrical signal. So whenever we want binary values the Analog to digital converters widely used.

One of the Analog to digital converters IC is ADC 0808. It is a Parallel analog to digital converter. The pin diagram of ADC 0808 are shown in fig 5.15

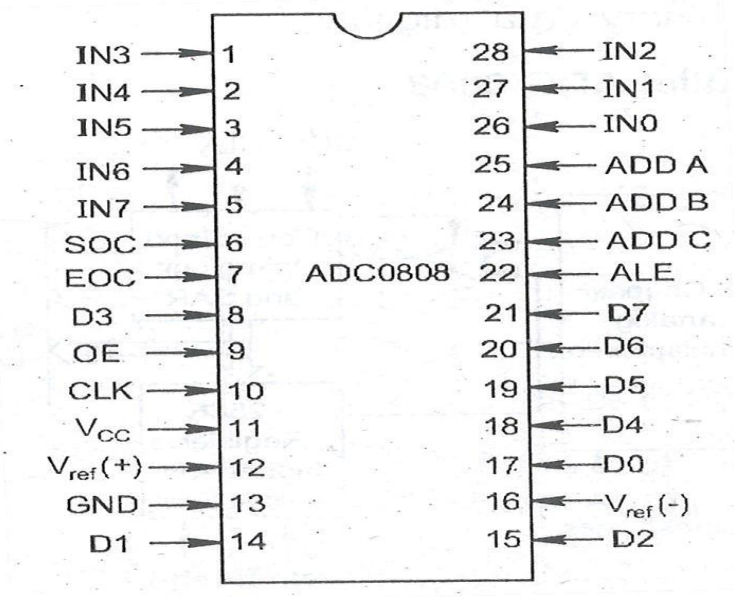


Fig 5.15 pin diagram of ADC 0808

Pin Description are Shown in table

Pin names	Description
IN0 - IN7	Analog inputs
A, B, C	Address lines for selecting analog inputs
D0 – D7	8-bit digital output
SOC	Start of conversion
EOC	End of conversion
OE	Output Latch Enable
CLK	Clock input
Vcc	Supply voltage
GND	Ground
$V_{ref}(+)$	Reference positive voltage (+5V max)
$V_{ref}(-)$	Reference negative voltage (0V min)

5.2.7.1 Interfacing ADC 0808 with micro controller 8051

The interfacing diagram of ADC 0808 with microcontroller 8051 in shown in fig 5.15 (a)

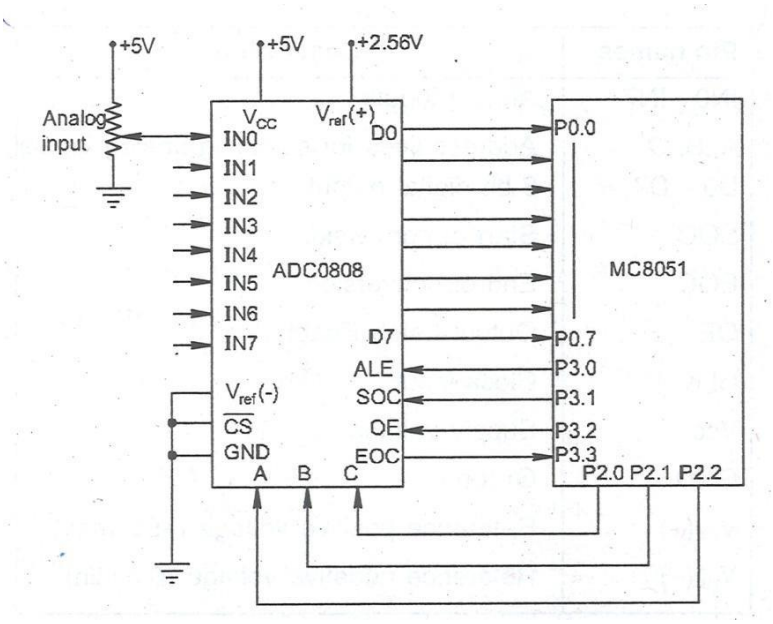


Fig 5.15 (a) Interfacing diagram of ADC 0808

The analog signal is applied to the any input pin INO – IN7. The digital output terminals (D_0- D_7) of ADC 0808 is directly connected to port 0 in 8051. Port 3 in micro controller are connected different control signals in ADC 0808. P3.0 – ALE, P3.1 – SOC P3.2 – OE, P 3.3 – EOC respectively. Port 2 in micro controllers are connected to the address lines of A,B and C in ADC 0808. The Address lines A,B,C are used to select the particular input Lines.

Program:

```
ORG4100H              ;   Origin4100H
MOV P0# FFH           ;   Set port 0asinput
SETBP3.3              ;   Set P3.3 as input
MOV P2#00H            ;   Select channel using Address Line
SETBP3.0              ;   Enable ALE
CLRP3.0               ;   disable ALE
SETBP3.1              ;   Initiate start of conversion
CLR P3.1              ;   Stop the start of Conversion
WAIT JNBP3.3 WAIT     ;   wait for Conversion
SETBP3.2              ;   Enable the output
MOVA,P0               ;   Read data in P0 to ACC
CLR P3.2              ;   Clear OE
ACALL CONVER          ;   Hexadecimal to ASCII Conversion
ACALL DISPLAY         ;   Go To Display Subroutine Conversion
END                   ;   End of Program
```

5.2.8 DAC INTERFACING:

The Microcontroller output is binary values, but application equipments display, motor, speakers etc. work in analog signal. So we need digital to Analog converters. The IC 0808 is an 8 bit DAC. The pin diagram of DAC 0808 in shown in fig 5.16

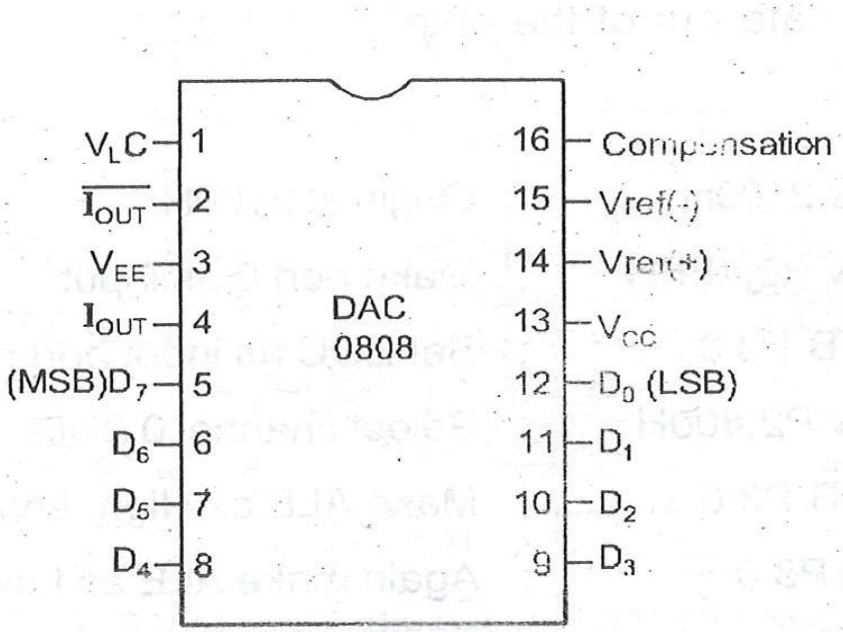


Fig 5.16 pin diagram of DAC 0808

In DAC 0808 the digital inputs are converted to current (I_{out}). The current I_{out} is converted into voltage using Op- amp The Current value in I_{out} pin is depends upon the binary numbers at the Do – D7 inputs of the DAD 0808 and the reference current (I ref)

$$I_{out} = I_{ref}\left\{\frac{D_7}{2^1} + \frac{D_6}{2^2} + \frac{D}{2^3} + \frac{D4}{2^4} + \frac{D3}{2^5} + \frac{D2}{2^6} + \frac{D1}{2^7} + \frac{D0}{2^8}\right.$$

The interfacing diagram of DAC 0808 with microcontroller 8051 in show in fig 5.17 interfacing diagram of DAC with 8051.

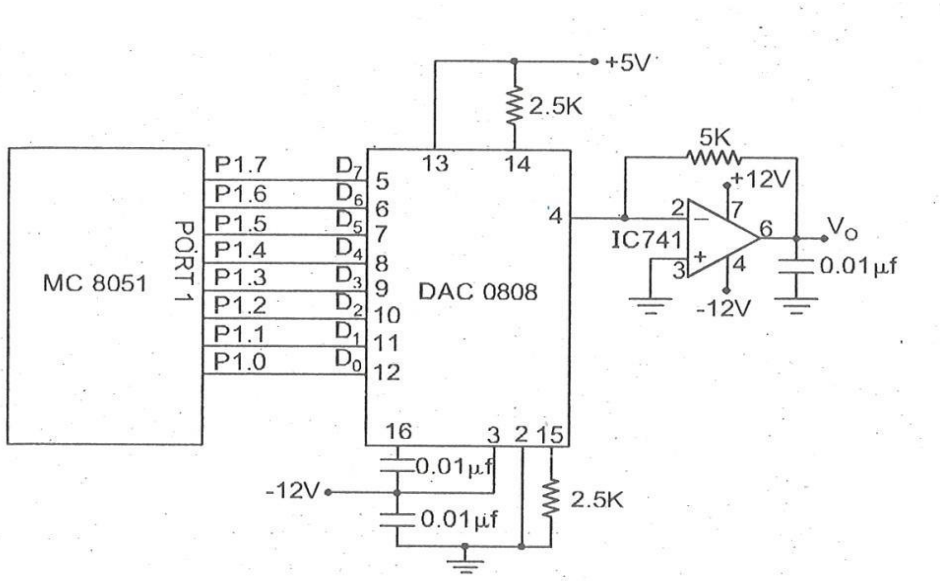


Fig 5.17 Interfacing diagram of DAC with 8051

Program:

```
        ORG4200H            ;   Origin 4200  MOVX
        DPTR, #4300H:       ;   Load 4300 in DPTR
        MOVXA,@DPTR         ;   Get data in
        ACCMOVP1,A          ;   Send it port1
        A Call Delay        ;   Make some delay

Delay   MOV RO,#F2H         ;   Load the value at F2h
   L1   MOV R1,#FFH         ;   Load the value at FFh
   L2   DJNZ R1,L1          ;   Repeat if R2 not = 0
        DJNZ R0,L2          ;   Repeat if R1 not = 0
        End                 ;   End of Program
```

5.2.9 KEY BOARDINTERFACING

A keyboard is a collection of push button type switches. which is commonly used as an input devices to a microcontroller based system. When a key is pressed, the microcontroller identifies the pressed key by using either a software based or hardware based technique and then performs the assigned operation.

The key board is interface with micro controller the keys are arrange in a two dimensional matrix form as shown in fig 5.18

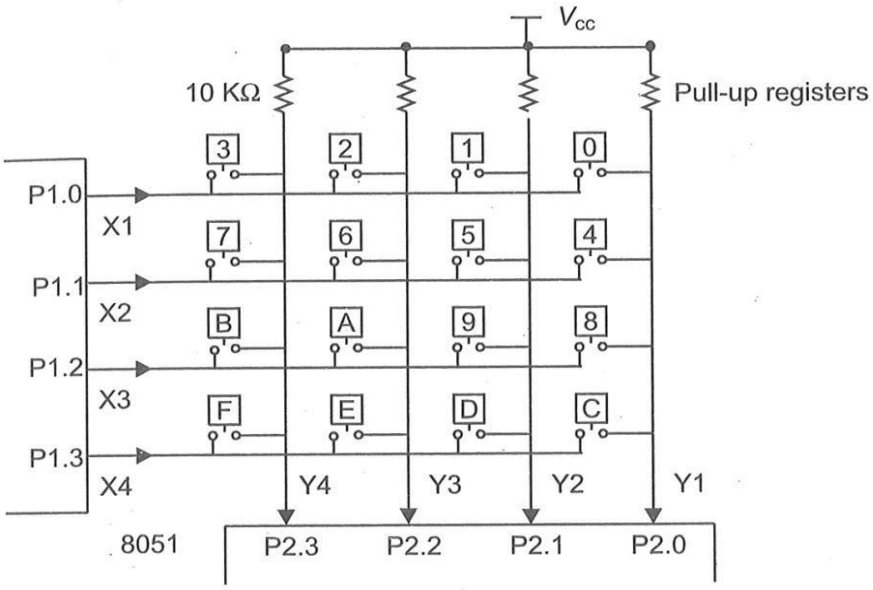


Fig 1.8 4x4 matrix key board interfacing using port 1 & port 3 of 8051

Here a row number and column number together uniquely identify a key. when a key is pressed, it is necessary to identify the row and column numbers of that key. The most commonly used approach for this purpose is known as row scanning technique

While interfacing a keyboard the following issues are taken into consideration

- Key –bounce
- Multiple key press (rollover)
- Key Pressed and held

Key bounce :

The problem of key – bounce arises when a push- button key is pressed or released because of Mechanical Spring action, the key vibrates for a small duration of time say 10 to 20 ms making and breaking the contact several times before finally closing the contact or Opening the contact. So a single key closure may appear to be multiple key closure to the interface circuit. The Microcontroller Should be able to distinguish between a bounce and a genuine key press. Otherwise each time a switch is pressed, the microcontroller will detect several fast key presses.

Key bounce can be Overcome by generating a delay of 10-20ms after sensing a key actuation and then re- sensing the key closure. If the key is still closed, the key closure is accepted as a valid key closure.

Multiple Key Press :

Multiple – key press problem is commonly known as rollover, which arises when two or more key are pressed simultaneously. This problem should be overcome so that the microcontroller does not perform an operation corresponding to a wrong key . There are three approaches to resolve the problem of rollover.

1. **Two – Key rollover**

This provides protection against the simultaneous closure of two keys. To ignore the keyboard until a single key press is detected, and the last key to remain pressed is accepted by the microcontroller. Performing the assigned operation is the best approach to overcome this issues.

2. **N- Key rollover**

This approach gives protected against N- Keys pressed simultaneously. This issues are resolve to store all the key closures in some internal buffer and perform the respective operations in sequence.

3. **K. Key Lockout**

In this method, a single key closures is recognized and additional key closures are ignored until the first one is released.

4. **Key pressed and Held:**

The Pressed key is accepted by the microcontroller after the debounce delay. No additional key press in accepted until all the keys are seen open certain period of time.

Row Scanning Technique:

A 4x4 keyboard matrix interfaced with 8051 microcontroller. There are four rows connected to the four port lines $P_{1.0}$- $P_{1.3}$ of port 1 and four column lines connected to the four port lines $P_{2.0}$-$P_{2.3}$ of port 2.

Closure of any one of the 16 keys in identified by row- scanning technique by searching one row at a time, is a time division multiplexed manner. In row scanning technique the following steps are performed.

Step 1 : In this step the row lines ($P_{1.0}$ - $P_{1.3}$) are configured as output lines. While the column lines ($P_{2.0}$ - $P_{2.3}$) are configured as input lines.

Step 2 : In this step check whether all the keys are open or not . This step in performed by generating 0‟s on all the ($P_{1.0}$ - $P_{1.3}$) row lines and then reading the ($P_{2.0}$ - $P_{2.3}$) column lines. The controller keeps on looping till all the ($P_{2.0}$ – $P_{2.3}$) Column lines are 1, indicating that no key is closed.

Step 3: In this step check whether any new key closure has taken place or not.Thisisdonebyoutputting0‟sonallthe($P_{1.0}$-$P_{1.3}$)rows and reading the $P_{2.0}$- $P_{2.3}$ line values. The controller keeps on looping until one of the column lineis„0;whichindicatesthatakeyclosurehastakenplace.

Step 4 : In this step a delay of 20ms is generated to overcome key bounce problems.

Step 5 : In this step, actual identification of the key pressed is performed by scanning one row at a time. This is performed by outputting „0‟ on the row under scan and „1‟ on the remaining lines and reading $P_{2.0}$ – $P_{2.3}$ Lines A „0‟ on any of the $P_{2.0}$ – $P_{2.3}$ lines indicates that a key on that column in the row under scan has been pressed.

Step 6 : In this step, first the decoded codes corresponding to the row and column numbers are obtained . Then the decoded row and column codes are used. obtain the key code of the key pressed.

The Row Scanning technique is shown in the form of a flowchart as shown in fig 5.19

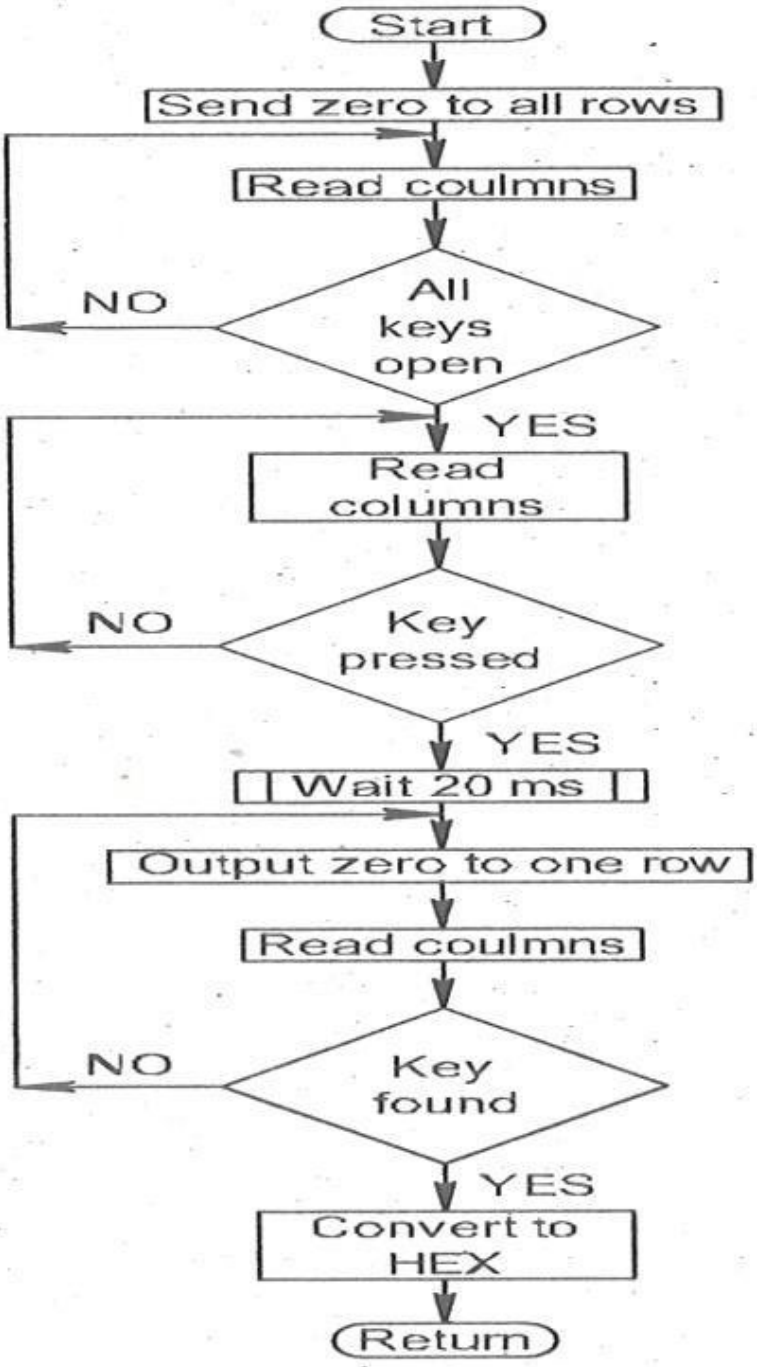


Fig 5.19 flow chart for keyboard scanning subroutine

Program:

```
MOV P3,#OFFh                        ; Make p2 an input port
K1   MOV P1,#00h                    ; Ground all rows at once
     MOV A,P3                       ; Read all Column. ensure all keys open.
 ANL A,# OOOO1111B                  ; Mask unused bits
   CJNE A, #OOOO1111Row _0          ; key row 0,find the col
     MOV P1,#111111O1B              ; ground ROW 1
     MOV A,P3                       ; Read all Column
     ANL A,# OOOO1111B              ; Mask unused bits
     CJNE A, #OOOO1111Row _1        ; key row 1,find the col
  MOV P1,#11111O11B                 ; ground ROW 2
     MOV A,P3                       ; Read all Column
  ANL A,# OOOO1111B                 ; Mask unused bits
     CJNE A, #OOOO1111Row _2        ; key row 2,find the col
     MOV P1,#1111O111B              ; ground ROW 3
     MOV A,P3                       ; Read all Column
  ANL A,# OOOO1111B                 ; Mask unused bits
     CJNE A, #OOOO1111Row _3        ; key row 3,find the col
      LJMP  K2                      ;if  none , false input ,repeat

ROW_0 MOV DPTR,#KCODE0              : Set DPTR = Start of row 0
      SJMP FIND                     : find the col key belong to
ROW_1 MOV DPTR,#KCODE0              : Set DPTR = Start of row 1
      SJMP FIND                     : find the col key belong to
ROW_2 MOV DPTR,#KCODE0              : Set DPTR = Start of row 2
      SJMP FIND                     : find the col key belong to
ROW_3 MOV DPTR,#KCODE0              : Set DPTR = Start of row 3
FIND  RRC A                         : See if any cy bit is low
      JNC          MATCH            : If  Zero ,Get the ASCII CODE
      INC             DPTR          : Point to next col. Address
      SJMP            FIND          : Keep  Searching
      MATCH CLR A                   : Set A=0 (Match is Found )
      MOV A@A+DPTR                  : Get ASCII Code from Table
       MOV PO,A                     : Display pressed key
       LJMP     K1
ASCII   LOOK TABLE FOR EACH ROW
          ORG 0300H
KCODE:  DB    0,  1,  2,  3         : ROW 0
KCODE:  DB    4,  5,  6,  7         : ROW 1
KCODE:  DB    8,  9,  10,11         : ROW 2
KCODE:  DB    C, D, E,  F           : ROW 3

          END
```

5.2.10 SEVEN SEGMENT LED DISPLAYINTERFACING

There are many application where you have to display numbers. The most popular display device used for displaying number is seven segment LED displays . In each module seven (Eight including the decimal point) LED segments are fabricated in a pattern as shown in Fig 5.20 (a). The seven segments are numbered as a,b,c,d e,f,g and the decimal point is dp . To reduce the number of pin counts, either all the cathodes are connected together inside the module providing common cathode type display (1-ON, 0 - OFF) as shown fig 5.20 (b) on all the anodes are connected together providing common anode type display (0 – ON, 1 - OFF) as shown is Figure 5.20 (c). If the display units are multiplexed together, we can display more that one character at a time .

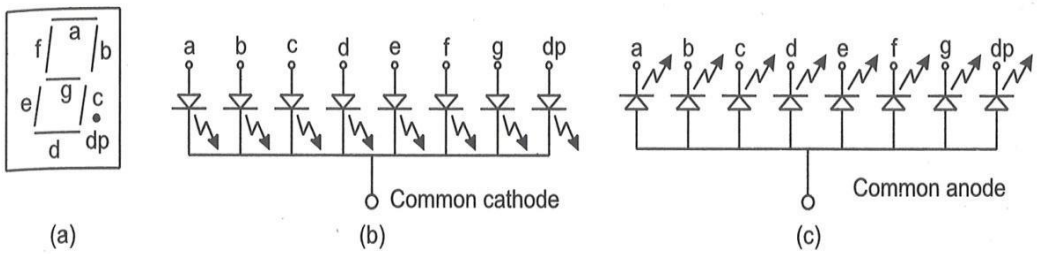


Fig 5.20 (a) Diagram of 7-segment LED
(b) diagram of common cathode 7-segment(c)diagram common anode 7-segment

Interfacing diagram of 4 digit multiplexed LED display with microcontroller 8051 is shown fig 5.21. In this four 7 segment LEDS are multiplexed together. The port 3 pins are used to drive the segments of the LED through driver IC. Similarly the port pins $P_{1.0}$ through $P_{1.3}$ are used to drive the digits through driver transistors.

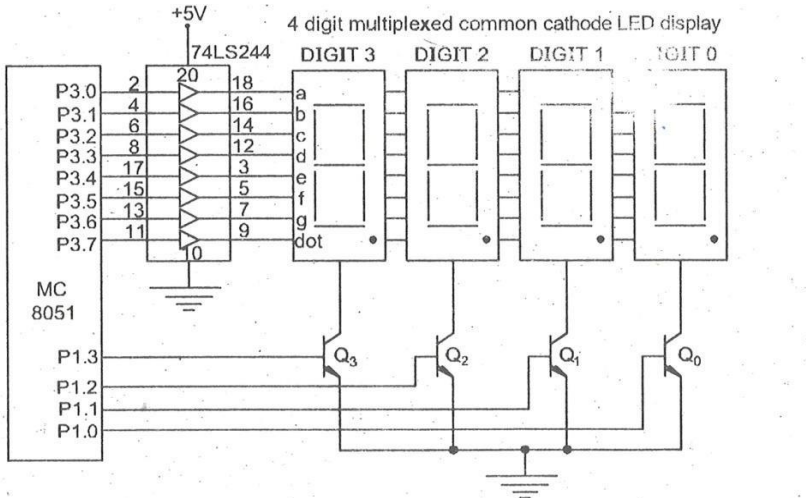


Fig 5.21 7 –Segment LED Display interface with 8051

For displaying a Number 1,2,3,4 in the display unit the hexa code may be formed as follows.

Number to be displayed	.	G	F	E	D	C	B	A	Equivalent Hexa Code
1	0	0	0	0	0	1	1	0	06H
2	0	1	0	1	1	0	1	1	5 B H
3.	0	1	0	0	1	1	1	1	4 F H
4.	0	1	1	0	0	1	1	0	66 H

The Number 1 may be displayed at digit 3 (P $_{1.3}$), the Number 2 Displayed at digit 2 (P$_{1.2}$), the number 3 displayed at digit 1 (P $_{1.1}$) and the 4 may be displayed at digit 0 (P$_{1.0}$)

If more than one display is to be used, the they can be time multiplexed. The human eye can not detect the blinking if each display is relit every 10 ms or so . A segment will be lit only if the segment line is brought high and the common cathode is brought low. Transistor must be used to handle the currents required by the LEDS, typically l0mA for each segment and 7mA for each cathode.

Program:

```
            ORG4100H          ;   Origin at4100H
REPEAT      MOVR0,88H         ;   Load the digit drive codeinR0 MOV
            DPTR, #TABLE      ;   Load the  starting  address of table
                                  in DPTR
            MOVR1, #04H       ;   Load number of charactersinR1 FIRST
           MOVX A,@DPTR       ;   Get the HEXA code in A
            MOVP3,A           ;   Move the HEXA code through Port 3
            MOVP1, R0         ;   Move the digit drive code through
                                  Port1
            ACALLDELAY        ;   Call delay subroutine
            INCDPTR           ;   Get the address of next HEXA code for
                                  displaying next character
            MOVA,R0           ;   Get the previous digit code in A
            RR A              ;   Rotate right to get next digit code
            MOVR0, A          ;   Place the digital codeinR0 DJNZR1,
            FIRST             ;   Go to display then extra character
            SJMPREPEAT        ;   Repeat the Process
            END               ;   End of Program
DELAY:      MOV R3,# 25H      ;   Load the outer loop value
WAIT2       MOV R4,# FFH      ;   Load the inner loop value
 WAIT 1     DJNZ R4,WAIT1;        Wait for delay
            DJNZR3,WAIT 2     ;   Wait for delay
            RET               ;   Return to main program

            TABLE : 03H, 53H, 4FH 66H
```

5.2.11 STEPPER MOTORINTERFACING

Stepper motor is a device used for getting accurate position control of rotating shafts. It converts electrical pulses into mechanical movements. A stepper motor employs rotation of its shaft, in terms of steps rather than continues rotation.

To rotate the shaft of the stepper motor a sequence of pulses is needed for applying to the windings of the stepper motor. The number of pulses required for one complete rotation of the shaft of the stepper motor is equal to its number of teeth on its rotor. The stator teeth and rotor teeth lock with each other to fix a position of the shaft. The schematic diagram of a stepper motor in shown in fig 5.22.

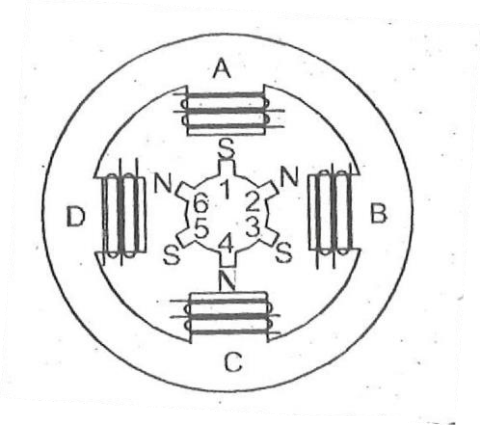


Fig 5.22 Schematic diagram of Stepper motor

Stepper motor is a permanent magnet type stepper motor. It contains a four pole stator and a rotor with six permanent poles. The stator is made up of laminated Soft iron. The stator windings are energized by the application of pulses. Each pole of stator has two coils wound in an opposite sense. So that each pole can be made either a north pole or a south pole as described by applying appropriate pulse to one of the coil. This type of winding in known as filler pole Winding .

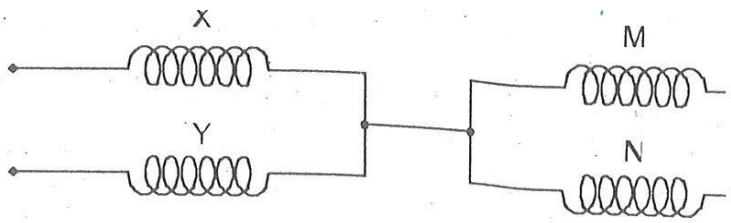


Fig 5.23 bi-filler pole windings

The structure of bi-filler pole windings is shown in the fig 5.23. The X and Y are two coils wound on the same pole. Similarly M and N are the two coils wound on the same pole, which are situated at diametrically opposite position.

The excitation sequence of a stepper motor is shown in the table below.

Step No.	Winding D P2.3	Winding C P2.2	Winding B P2.1	Winding A P2.0	Hexa Code	Direction	
1	1	0	0	1	09H	forward ↓	Reverse ↑
2	0	0	1	1	03H		
3	0	1	1	0	06H		
4	1	1	0	0	0CH		

For doing the above type of excitation, each sequence of data is continuously output through Port 1 for a specific duration of time.

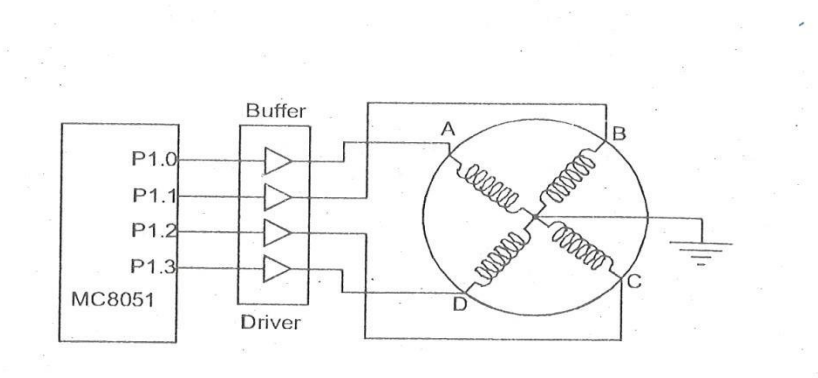


Fig 5.24 Interfacing diagram of stepper motor with 8051

The connection diagram of stepper motor with Microcontroller 8051 shown in the Fig 5.24

The four coils of stepper motor are connected to the port 0 pins through drivers. The port pin P2.0 is connected to the coil A, the Port pin P2.1 is connected to coil B, the Port pin P2.2 its connected to the coil C and the Port pin P23 is connected to the Coil D.

Programe :

```
        ORG4000H        ;   Origin4000H
        MOV A,#09H      ;   Place first data in Acc.
REPEAT  MOVP2,A         ;   Apply the data to Port1
        LCALLDELAY      ;   Call delay subroutine
        RLA             ;   Repeat the next data by using
                            rotation
        SJMPREPEAT      ;   Repeat the above process
        END             ;   End of the program
```

DELAY SUBROUTINE

```
DLEAY   MOV R0, # XXH   ;   Load  R0 register
FIRST   MOV R1, # XXH   ;   Load R1 register
NEXT    DJNZ R1, NEXT   ;   Decrement inner loop up to 00H
        DJNZ R0, FIRST  ;   Decrement outer loop up to 00H
        RET             ;   Return to the main program
```

5.2.12 DC MOTOR INTERFACING USINGPWM

DC Motor rotates continuously at constant speed of DC Motor depends upon the average DC Voltage applied across its armature. If we want to change the speed of the motor to vary the applied DC Voltage, If the polarity of the applied voltage is changed, the motor can run in reverse direction.

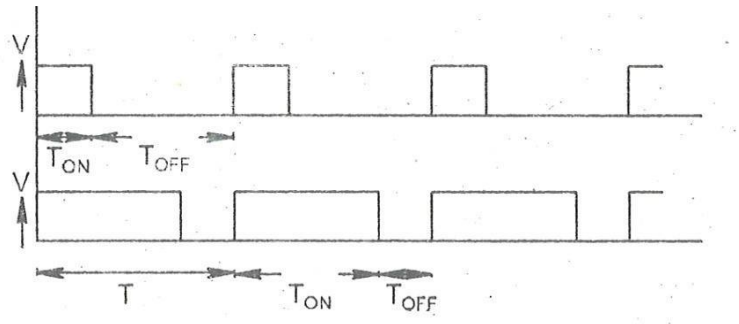


Fig 5.25 PWM signals

In modern days, the speed of the DC Motor is Controlled by using Pulse width modulation(PWM) method. By using the PWM signal, the average voltage of the DC motor can be varied, there by the speed of the DC motor is controlled.

Two different pulse modulated signals are shown in the fig 5.25. In the second waveform, ON period is high and OFF period is low. Therefore its average voltage is high. In the first waveform ON period is low and the OFF period is high. Therefore its average voltage is low. In pulse width modulation, the period of the pulse is kept constant. Pulse ON period and OFF period times may be varied. The interfacing diagram of DC motor with 8051 microcontroller is shown in the fig 5.26

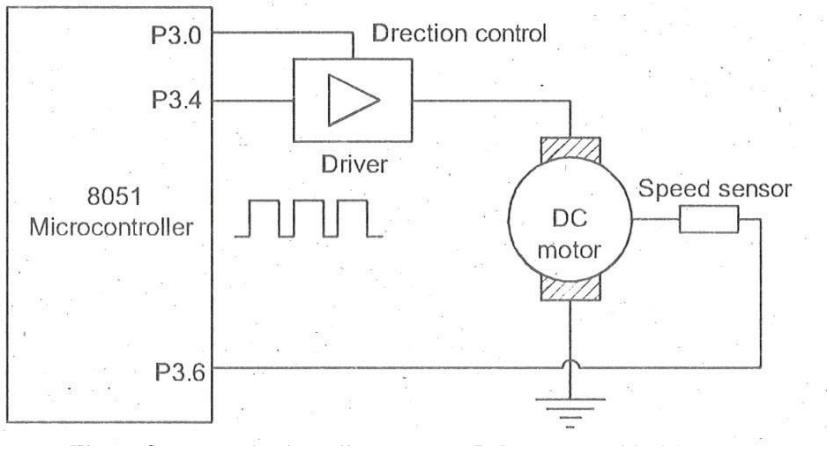


Fig 5.26 Interfacing diagram of DC motor 8051

The speed of the DC motor is increased, when the duty cycle of the signal is high. Similarly, the speed of the DC motor is decreased, when the duty cycle of the signal is low. By using the program, we can be able to change the duty cycle of the PWM signal and also the motor speed may be varied.

Normally the current produced from the microcontroller is not sufficient to drive the DC motor effectively. Hence drivers may be connected in between the motor and microcontroller.

In some applications, we keep the speed of the motor as constant. For doing this, a feedback circuit may be connected to it. The feedback circuit contains a speed sensor for

getting the present speed of the motor. According to the sensor output, the microcontroller adjusts the speed of the motor by changing the duty cycle of the signal applied to it. Hence constant speed can be maintained.

Program 11 :

```
        ORG4000H        ;   Origin at 4000H
        SETBP3.0        ;   Set in forwarded direction
REPEAT  SETBP3.4        ;   Make high level output
        ACALLONDLY      ;   Call ON period delay
        CLRP3.4         ;   Make low level output
        ACALLOFFDLY     ;   OFF period delay
        SJMPREPEAT      ;   Repeat the process
        END             ;   End of Program
ONDLY   MOV R1,# A0     ;   ON time delay
WAIT 2  MOV R2, # FFH   ;   Load  R2 with FFH
WAIT 1  DJNZ R2,WAIT1   ;   Wait for the completion On Time
        DJNZ R1,WAIT2   ;   Wait for the completion On Time
        RET             ;   Return to man program
OFFDLY  MOVR1, #5F      ;   OFF time delay
        MOV R2, # FFH   ;   Load  R2 with FFH
NEXT 2  DJNZ R2,NEXT1   ;   Wait for the completion OFF Time
NEXT 1  DJNZ R1,NEXT 2  ;   Wait for the completion OFF Time
        RET             ;   Return to main program
```

REVIEW QUESTIONS

PART – A

1. Define peripherals.
2. What is interfacing?
3. Mention the ports placed in 8255.
4. State the modes of operation of 8255.
5. What is relay?
6. Mention the 3 important components of electromagnetic relay.
7. Define ADC.
8. Define DAC.
9. Mention the different types of seven segment LEDs.
10. What is the use of Stepper motor?

PART – B

1. What is the use of 8255?
2. Mention the ports placed in group A and group B of 8255.
3. How is the I/O modes of 8255 classified.
4. Define opto isolator.
5. Define transducers.
6. Define ASM.
7. Which technique is used in ADC 0808?
8. State the output current of DAC 0808.
9. Mention the three major tasks of keyboard to get a meaningful data.
10. Which technique is used for varying the speed of DC motors used in microcontrollers?

PART –C

1. With the diagram explain how 8051 is interfaced to external memory?
2. Draw the functional diagram of 8255 and explain each block.
3. Draw the interfacing diagram of LM35 with microcontroller and explain its operation.
4. Explain the various modes of operation of 8255.
5. With the diagram explain how is 8255 interfaced with 8051
6. Draw the interfacing diagram of DAC 0808 with 8051 and explain its operation.
7. Draw the interfacing diagram of matrix keyboard with microcontroller 8051 and explain its operation.
8. Draw the interfacing diagram of stepper motor with 8051 and explain its operation.
9. Draw the interfacing diagram of 4 digit seven segment LED display with microcontroller 8051 and explain its operation.
10. Draw the interfacing diagram of DC motor with microcontroller 8051 and explain its operation.