

This is the Reference copy for the **LECTURERS** to conduct **ONLINE Classes.** - **Maanyas MGB Publications**

Kindly don't send this soft copy to the STUDENTS.

ECET -2021 STUDY MATERIALS & 20+ YEARS PREVIOUS PAPERS WITH SOLUTIONS

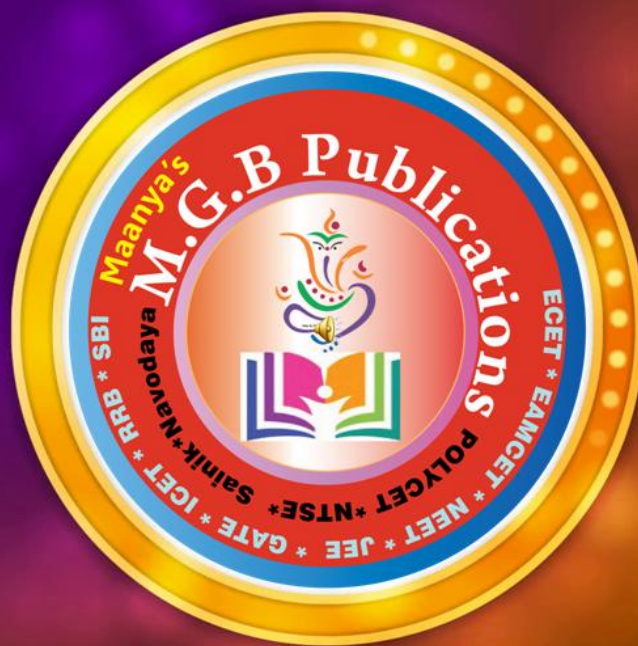
MPC	Objective Study Material ECET Objective Study Material MATHEMATICS VOL-1 & VOL-2 4900+ MCQ's All Examinations ₹ 430 ₹ 720	Previous Question Papers ECET Objective Study Material Physics & Chemistry ₹ 320 ₹ 540	Previous Question Papers ECET Previous Question Papers Mathematics & Chemistry ₹ 270 ₹ 380	Previous Question Papers ECET Previous Question Papers Mechanical Engineering 3500 MCQ's ₹ 110 ₹ 180	MECHANICAL
	Objective Study Material POLY CET Objective Study Material Mathematics ₹ 430 ₹ 720	Objective Study Material POLY CET Objective Study Material Physics & Chemistry ₹ 430 ₹ 720	Previous Question Papers POLY CET Previous Question Papers Mathematics, Physics & Chemistry ₹ 430 ₹ 720	Objective Study Material ECET Objective Study Material Mechanical ENGINEERING ₹ 430 ₹ 720	
	Previous Question Papers ECET Objective Question Papers Electronics & Communication Engg. 2000 MCQ's ₹ 130 ₹ 225	Objective Study Material ECET Objective Study Material Electronics & Communications Engg. ₹ 430 ₹ 720	Previous Question Papers ECET Objective Question Papers CIVIL ENGINEERING 1700 MCQ's ₹ 105 ₹ 175	Objective Study Material ECET Objective Study Material CIVIL ENGINEERING ₹ 390 ₹ 680	
	Previous Question Papers ECET Objective Question Papers Electrical & Electronics Engg. 2000 MCQ's ₹ 135 ₹ 230	Objective Study Material ECET Objective Study Material Electrical & Electronics Engg. ₹ 430 ₹ 720	Previous Question Papers ECET Objective Question Papers COMPUTER SCIENCE ENGG. ₹ 130 ₹ 225	Objective Study Material ECET Objective Study Material Computer Science ENGINEERING ₹ 390 ₹ 680	
FREE	FREE	FREE	FREE		

COMBO OFFER	MPC + Engg (PQP)	45% OFF	₹ 370	₹ 675
	MPC + Engg (OSM)	50% OFF	₹ 990	₹ 1980
	MPC (PQP + OSM)	45% OFF	₹ 940	₹ 1710
	ENGINEERING (PQP + OSM)	45% OFF	₹ 520	₹ 945

MAANYA'S MGB Publications

All SEMESTER **TEXTBOOKS** are available for the **STUDENTS** at **FREE** of **COST** in our **MOBILE APP**

Hyderabad: 9290429549 & Tirupati: 9000305079



For **FREE**

Study
Materials

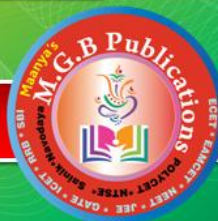
Practice
Papers

Recorded Video
Lectures

Online
Exams

Download

Maanyas MGB



Publications

Mobile app



1st *Edition*

AP EC 502 MC



Microcontrollers



For D.E.C.E III Year (V sem)

N. Dhananjaya * P. Srinivas
D. Kiran Kumar * D. V Ramana



Price Rs: 175

Maanyas M.G.B Publications
Hyderabad & Tirupati. Cell: 9000 3050 79

Microcontrollers

First Edition: June - 2018

© All Rights Reserved

Printing of books passes through many stages—writing, composing, proof reading, printing etc. We try our level best to make the book error-free. If any mistake has inadvertently crept in, we regret it and would be deeply indebted to those who point it out. We do not take any legal responsibility.

No part of this book may be reproduced, stored in any retrieval system or transmitted in any form by any means - electronic, mechanical photocopying, recording or otherwise without the prior written, permission of the author and publishers.

For Copies Please Contact


M.G.B Publications

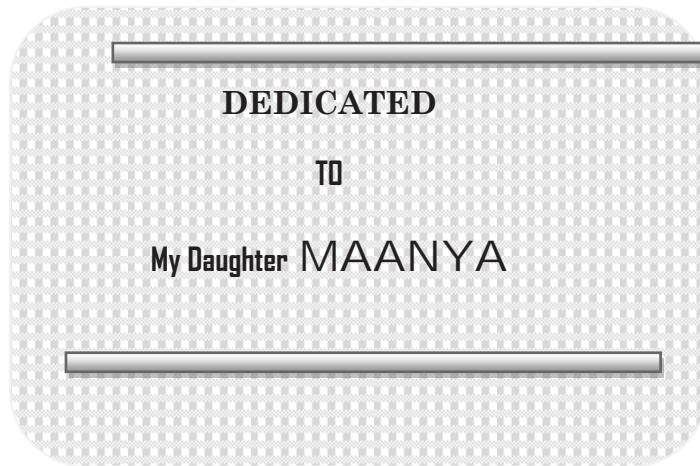
Cell: 9000305079

Also Available at All Leading Book Shops



TABLE OF CONTENTS

		
Chapter Name	Page. No
1. Architecture of Microcontroller 8051	1.1 - 1.42
2. Instruction set of 8051 microcontroller	2.1 - 2.62
3. 8051 Programming Concepts	3.1 - 3.37
4. Interfacing Simple I/O devices	4.1 - 4.22
5. Programming 8051 Timers, Serial port & Simple Applications	5.1 - 5.37



Author
N. Dhananjaya

CHAPTER 1

Architecture of Microcontroller 8051

OBJECTIVES

- 1.1 Draw the block diagram of a microcomputer and explain the function of each block.
- 1.2 List the features of micro controllers.
- 1.3 Compare Microprocessors and Microcontrollers
- 1.4 State the details of INTEL microcontroller family chips.
- 1.5 State the features of Intel 8051 Micro Controller.
- 1.6 Draw the functional block diagram of 8051 microcontroller
- 1.7 Draw the register structure of 8051 and explain.
- 1.8 Explain the function of various special function registers.
- 1.9 Draw the pin diagram of 8051 micro controller and specify the purpose of each pin.
- 1.10 Explain internal memory Organization in 8051
- 1.11 Explain external memory access in 8051
- 1.12 Explain various ports of 8051.
- 1.13 Explain counters & timers in 8051
- 1.14 Explain serial input/output of 8051
- 1.15 Explain interrupts in 8051

1.1. Draw the block diagram of a microcomputer and explain the function of each block.

- A digital computer in which one Microprocessor has been provided to act as a CPU is called micro computer.
- It is the smallest and least expensive general purpose computer.
- The fig. 1.1 shows the block diagram of Microcomputer.
- It mainly consists of four sections namely:
 1. Microprocessor
 2. Input device
 3. Output device
 4. Memory

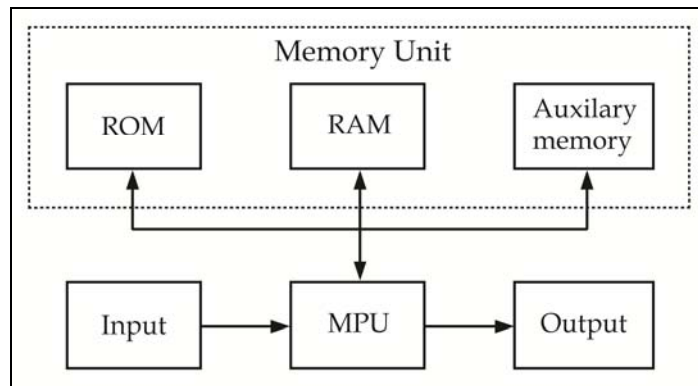


Fig. 1.1(a) Block diagram of Microcomputer

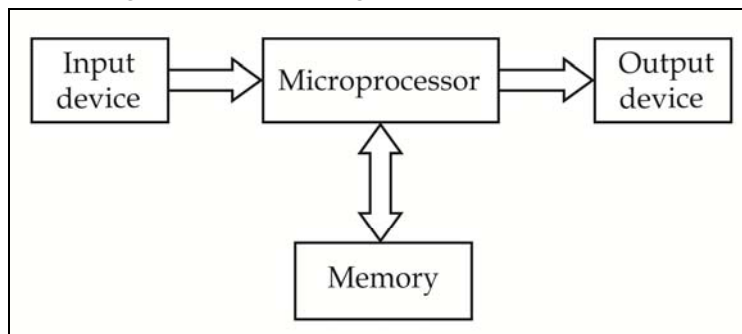


Fig. 1.1(b) Block diagram of Microcomputer (additional fig)

1. MPU (Microprocessor Unit)

- The CPU of a microcomputer is a microprocessor. The major components of a microprocessor are: ALU, register array and timing and control unit.
- The heart of a microcomputer is microprocessor unit (MPU).

- The MPU is a VLSI device.
- It is a general purpose processing unit, built into a single integrated circuit.
- It is a decision making unit, which decides the task to be performed by the system.
- It may be 8-bit or 16-bit or 32-bit processor.
- The MPU executes instructions of the program and process data.
- It is responsible for performing all arithmetic operations and making the logical decision initiated by the computer's program.

2. Memory Unit

- The memory unit is used to store data, address, or instructions in hexadecimal form.
- The memory unit is subdivided into two parts as Primary memory and Secondary memory or auxiliary memory.
- The primary memory is also called as semiconductor memory.
- The primary memory is again divided into number of sub units in which RAM and ROM are important.
- The RAM is used to store the temporary data whereas ROM is used to store permanent data.
- The secondary or auxiliary memory is a magnetic memory. It is used to store large amount of data. Example for secondary memory is hard disk.

3. Input Device

- The computer receives data and instructions through input devices.
- An input devices converts instructions, input data and signals into proper binary form suitable for a digital computer.
- A keyboard and simple switches are used as input devices.

4. Output Device

- The computer sends the processed data (Results) to the output devices.
- An output device may store, print, display or send electrical signal to control certain equipment.
- The examples of simple output devices are LEDs, CRT, D/A converter, Video Monitor and Printers etc.

1.2. List the features of micro controllers.

Definition: The special digital system contains a microprocessor, memory (RAM/ROM), I/O ports and timer on a single chip is known as **Microcontroller**.

The main features of Microcontroller are:

- A **microcontroller** is a single chip computer.

- It is used to control electronic or electro-mechanical devices. It is also called as dedicated computer or embedded microcontroller.
- It has the specified computational capabilities. In compare with general purpose microprocessor, it is less cost and more effective.
- It is designed for a specific task.
- It is designed with VLSI technique.
- It is highly integrated chip that consists all the components comprising a controller. Basically it contains a CPU, RAM, ROM, I/O ports, and timers.
- Specifically it is designed for a single task to control one device. It is small in size. Microcontrollers are selected based on bus width,
- Internal logic technique (VLSI or VHDL or Verilog).
- Present 8, 16 and 32bit microcontrollers are in use.

1.3. Compare Microprocessors and Microcontrollers

S.No.	Microprocessor (MP)	Microcontroller (MC)
1	Microprocessor contains ALU, control unit (clock and timing), different registers and interrupt circuit.	Microcontroller contains microprocessor, memory (ROM and RAM), I/O ports and peripheral devices such as A/D converter, timer, counter etc.
2	Microprocessor must have many additional parts to function as a computer.	Microcontroller can function as a computer with the addition of no external parts.
3	Microprocessor based system requires more hardware.	Microcontroller based system requires less hardware reducing PCB size and increasing the reliability.
4	Microprocessor based system is more flexible in design point of view.	Less flexible in design point of view.
5	It has single memory map for data and code.	It has separate memory map for data and code.
6	Very few pins are programmable.	Most of the pins are programmable that is, capable of having several functions.
7	Time taken to complete a process is more.	Time taken to complete a process is less.
8	It has many instructions to move data between memory and CPU.	It has one or two instruction to move data between memory and CPU.
9	It may have one or two bit handling instructions.	It may have many bit handling instructions.

1.4. State the details of INTEL microcontroller family chips.

1. In 1976, Intel introduced 8-bit microcontroller 8048. It is also known as MCS-48. The same company continued to drive the evolution of single chip microcontrollers.
2. In the year 1980, Intel introduced MCS-51 (8051 family) microcontroller, with higher performance than 8048.
3. The below table shows the main features of Intel 8051 family.

Features	8051	8052	8031
1. ROM (on-chip)	4K bytes	8K bytes	0
2. RAM (on-chip)	128 bytes	256 bytes	128 bytes
3. Timers	2	3	2
4. I/O Pins	32	32	32
5. Serial ports	1	1	1
6. Interrupt sources	6	8	6

4. In **1983**, Intel introduced a 16-bit single chip microcomputer series called MCS-96(8096 **family**). It contains 16-bit CPU, 8 K ROM, 232 bytes RAM, timer/counter, parallel I/O, 10-bit A/D, high control such as missile guidance and control, complex instrumentation system, control of sophisticated machines etc.

Note: The 8051 is the original member of the 8051 family. It is an 8-bit microcontroller designed by Intel in the year 1981.

1.5. State the features of Intel 8051 Micro Controller.

The features of Intel 8051 are listed below.

- The 8051 is an 8-bit microcontroller
- It was designed by Intel in the year 1981.
- It is a 40 pin IC chip.
- It operates with 12 MHz clock and a single +5 V supply.
- Its internal ROM (on-chip program memory) capacity is 4K bytes.
- It's internal RAM (on - chip data memory) capacity is 128 bytes.
- It has external data memory of 64Kbytes and external program memory of 64Kbytes.
- It has two 16-bit Timers/Counters.
- It has 32 bidirectional I/O lines organized as four 8-bit ports (P0-P3).

- It has one serial port and four 8-bit I/O ports.
- It has four bank registers.
- It has five interrupts, 2 from external devices, 2 from timer/counters and 1 from serial port.
- It has 111 instructions: 49 single byte, 45 two byte and 17 three byte.

1.6. Draw the functional block diagram of 8051 microcontroller

Fig. 1.6(a) shows the **block diagram of a microcontroller**. A detailed functional block diagram of 8051(or) the **internal architecture of 8051** is shown in fig. fig. 1.6(b). The functional description of each block is presented briefly below.

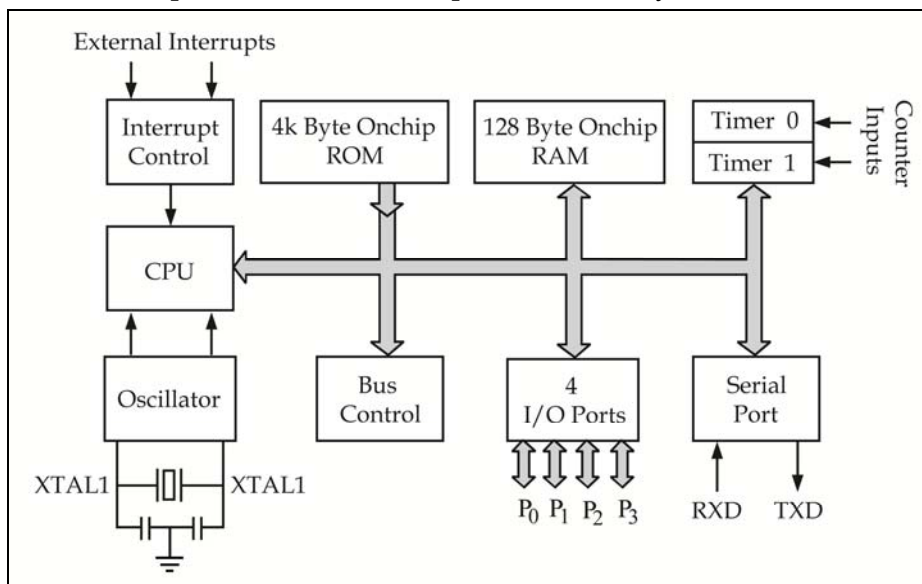


Fig. 1.6(a)

1. CPU (Microprocessor)

CPU is a general purpose microprocessors such as Intel's X86 family or Motorola's 680x0 family.

The CPU of 8051 is capable in performing arithmetic and logical operations like additions, subtraction, multiplication, division, logical AND, OR, EX-OR, rotate, clear and complement. This CPU can do bit wise as well as byte wise manipulations. It works based on the code written in the ROM.

2. ROM/RAM Memory

It is a fixed amount of on chip or internal memory. It stores programs and data temporarily during program execution. In other words the program codes are stored in the ROM and the data could be stored in RAM.

5. Timers and Counters

8051 has two 16-bit timers: timer-0 and timer-1. They can be used either as timers or event counters. Counters are used to count the events where as timers are used to maintain time delays between the actions.

6. Interrupt control

Interrupt is a signal send by an external device or from internal unit to the microcontroller so as to request the controller to perform a particular task or function. The interrupt control unit transfers the microcontroller attention from one task to other.

7. Oscillator

This circuit generates the basic timing clock signal for the operation of the circuit using crystal oscillator. The oscillator circuit generates a train of pulses at a frequency of the crystal.

Note: The machine cycle is group of six states (or 12 crystal pulses). A state is the basic time period for the microcontroller to fetching an op-code, decoding an op-code, executing an op-code or writing data byte.

8. Bus control

A control bus is used by microcontroller for communicating with other devices within the system. Basically Bus is a collection of wires which work as a communication channel or medium for transfer of Data. Buses are of three types: i) Address ii) Bus Data Bus and iii) Control bus.

9. Special Function Registers (SFRs)

The 8051 has 21 Special Function Registers. These are: A, B, DPTR, PSW, IP, IE, TCON, SCON, PCON, P₀, P₁, P₂, P₃, SBUF etc. All these are byte addressable and some of them are bit addressable also. The SFRs can be accessed by their names or by their addresses. These are useful in accessing I/O ports, timers/counters, UART, power Controlling etc.

10. Timing and Control Unit

This unit synchronizes all the operations with the clock and generates control signals for proper functioning.

11. Instruction Register

It is an 8-bit register. When instruction is fetched from memory it is loaded in the instruction register. It holds the instruction which is to be decoded. It determines the operation to be followed in executing the entire instruction.

12. Temporary Registers

There are two 8-bit temporary registers TMP1, TMP2. They are also integral part of ALU. Comparisons and certain other operations use the contents of these registers.

1.7. Draw the register structure of 8051 and explain.

Registers are used to store the data temporarily. The data may be of op-codes or operands or address of a memory location or address of a peripheral. The 8051 has wide range of registers.

These are classified as:

1. General purpose registers
2. Special purpose registers (MCU related registers)
3. Special function registers

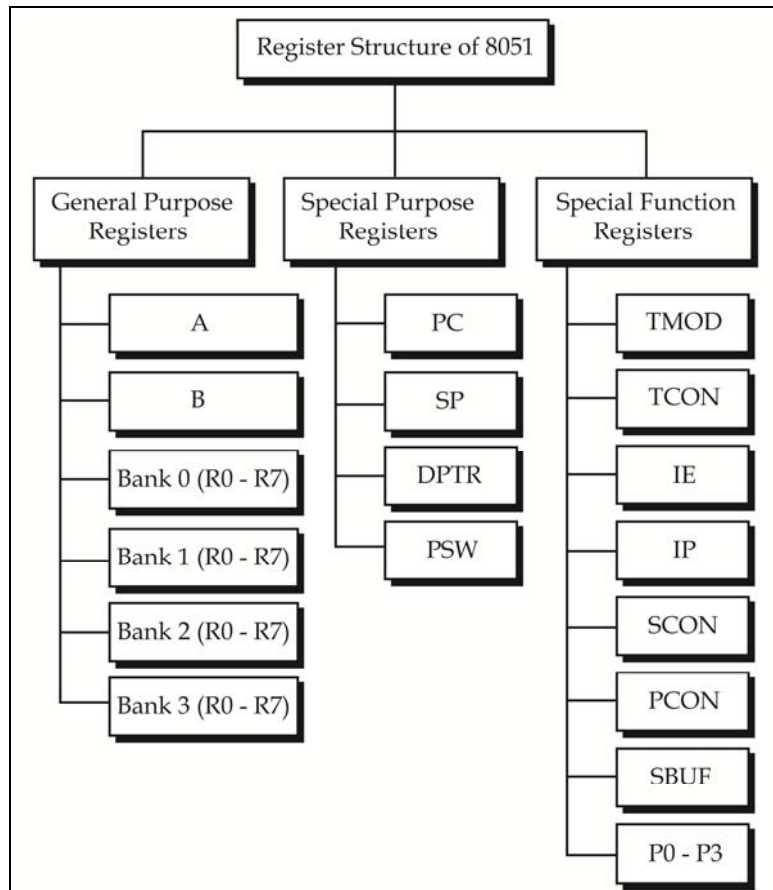


Fig.1.7

1.7.1. General purpose registers

The 8051 has 34 general purpose registers. They are Accumulator, Register B and remaining 32 registers are arranged a part of the internal 128 byte RAM. All these registers are programmable registers.

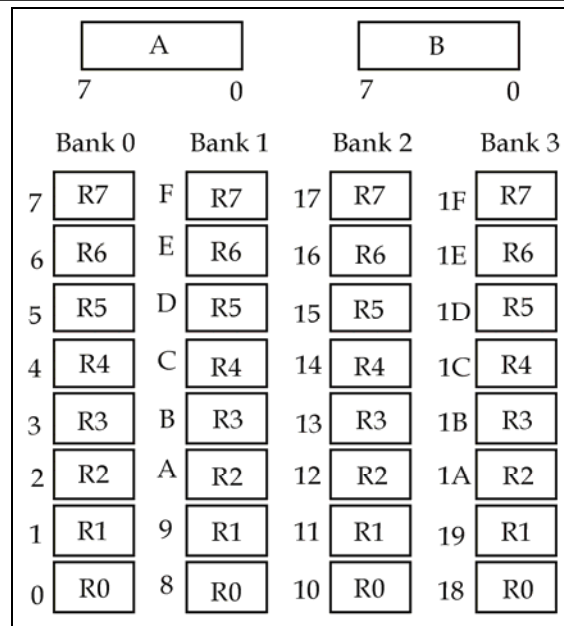


Fig. 1.7.1

GPRs	Function
1. R0-R7	<ul style="list-style-type: none"> • The lowest 32 bytes (address 00H to 1FH) of the on-chip RAM form 4 banks. • There are 128 bytes of RAM in the 8051. • The lowest 32 bytes (address 00H to 1FH) of the on-chip RAM form 4 banks. • The 4 register banks are numbered 0 to 3. Each bank is made up of 8 registers named R0 to R7. These are used to store data temporarily. • RAM locations from 00H to 07H are set aside for bank 0 of R0-R7 where R0 is RAM location 00H, R1 is RAM location 01H, R2 is location 02H, and so on, until memory location 07H, which belongs to R7 of bank 0. • The second bank of registers R0-R7 starts at RAM location 08H and goes to location 0FH. • The third bank of R0-R7 starts at memory location 10H and goes to location 17H. • Finally, RAM locations 18H to 1FH are set aside for the fourth bank of R0-R7. • The fig. a shows how the 32 bytes are allocated into 4 banks. • Note that only one register bank can be accessed by 8051 at a time. • The bits RS1 (PSW.4) and RS0 (PWS.3) of program status word (PSW)

	<p>are used to select any one of the four register banks as follows.</p> <table border="1"> <thead> <tr> <th>RS1</th> <th>RS0</th> <th>Bank selection</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Bank 0: 00- 07H</td> </tr> <tr> <td>0</td> <td>1</td> <td>Bank 1: 08- 0FH</td> </tr> <tr> <td>1</td> <td>0</td> <td>Bank 2: 10- 17H</td> </tr> <tr> <td>1</td> <td>1</td> <td>Bank 3: 18- 1FH</td> </tr> </tbody> </table> <ul style="list-style-type: none"> • Default bank (On reset) is bank 0 (locations 00 – 07H). 	RS1	RS0	Bank selection	0	0	Bank 0: 00- 07H	0	1	Bank 1: 08- 0FH	1	0	Bank 2: 10- 17H	1	1	Bank 3: 18- 1FH
RS1	RS0	Bank selection														
0	0	Bank 0: 00- 07H														
0	1	Bank 1: 08- 0FH														
1	0	Bank 2: 10- 17H														
1	1	Bank 3: 18- 1FH														
2. A	<p>A (Accumulator)</p> <ul style="list-style-type: none"> • It is an 8-bit register. • It is used for all arithmetic and logical operations. • The A-register must be used as destination for all addition and subtraction operations. • For logical operations this can be used as a source or a destination. • There are some specific operations like clear, complement, rotate and swapping the data must be reside in A-register only. <p>Do You Know?</p> <ul style="list-style-type: none"> • While multiplying, it holds one of the 8-bit operands and after the execution of the multiplication instruction; it stores the lower byte of the result. • While dividing, it holds an 8-bit dividend and after the execution of division instruction, the quotient is stored in A-register. 															
3. B	<ul style="list-style-type: none"> • It is an 8-bit register. • It is used for multiplication and division along with the accumulator. <p>Do You Know?</p> <ul style="list-style-type: none"> • While multiplying, it holds one of the 8-bit operands and after the execution of the multiplication instruction; it stores the higher byte of the result. • While dividing, it holds an 8-bit divisor and after the execution of division instruction, the remainder is stored in B-register. 															

1.7.2. Special purpose registers

The microcontroller contains four special purpose registers. They are

1. Program counter
2. Stack Pointer
3. Data Pointer
4. Program Status Word

SPRs	Function
1. PC	<ul style="list-style-type: none"> The program counter (PC) is a 16-bit register. It is used to hold the address of next instruction to be fetched. The PC is automatically incremented to point the next instruction in the program sequence after execution of the current instruction. The PC is the only register that does not have an internal (on-chip) RAM address.
2. SP	<ul style="list-style-type: none"> A group of memory locations in RAM is referred to as stack. The stack can store data as well as address during execution of a program. The 8-bit stack pointer (SP) register is used by the 8051 to access the stack. It holds the address of top of the stack. After the RESET operation, the stack pointer is initialized to 07H, causing the stack to begin at 08H. When data is stored on to the stack, the SP value is incremented by 1. Similarly while retrieving the data from the stack; the SP value is decremented by 1. The data is pushed onto the stack using instruction PUSH and the data is retrieved using instruction POP.
3. DPTR	<ul style="list-style-type: none"> The DPTR register is made up of two 8-bit registers named DPH and DPL as shown in fig. 1.7.2. <div data-bbox="582 1147 1173 1363" style="text-align: center;"> <pre> graph LR subgraph DPTR [16-bit DPTR] DPH[DPH (83H)] DPL[DPL (82H)] end DPH --- DPL DPL -- 16 --> MA[Memory Address] style DPH fill:none,stroke:none style DPL fill:none,stroke:none style MA fill:none,stroke:none </pre> </div> <p style="text-align: center;">Fig. 1.7.2</p> <ul style="list-style-type: none"> Its function is to hold a 16-bit address. The data pointer is used for addressing the off-chip data and code with the MOVX and MOVC commands, respectively.
4. PSW	<ul style="list-style-type: none"> PSW (Program Status Word) is an 8-bit register. It consists of 4 flags and 2-bits for bank selection of internal RAM.

Additional Information

Program Status Word (Flag Register)

- Program Status Word or simply PSW register is one of the most important SFRs.
- It is an 8-bit register. It consists of:
 - 4 conditional flags that set /reset automatically to the outcomes of math operations (CY, AC, OV & P).
 - 2 Register bank select bits (RS1 & RS0).
 - A general purpose flag (F0).
- Fig. 1.7.3 shows the bit pattern of the program status word.

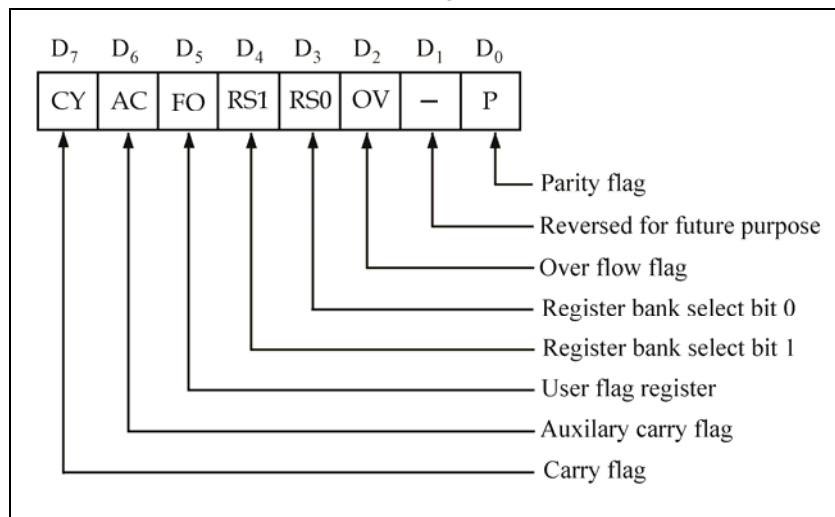


Fig.1.7.3

a) Conditional Flags**1. Carry Flag (CY)**

- Carry flag is set if there is a carry from bit 7.
- It is used in arithmetic, jump, rotates, and Boolean instructions.

2. Auxiliary Carry Flag (AC)

- Auxiliary Carry flag (AC) is set when there is a carry out of 3rd bit, during addition or subtraction operation and otherwise cleared.
- Auxiliary Carry Flag is used for BCD operations only.

3. Overflow Flag (OV)

- The overflow flag is set if there is a carry from bit 7, but not from bit 6, or there is a carry from bit 6, but not from bit 7 otherwise it is cleared.

4. Parity Flag (P)

- The parity flag is set if the accumulator contains an odd number of 1s.
- The parity flag is reset if the accumulator contains an even number of 1s.
- It is mainly used during data transmit and receive via serial communication.

b) Register Bank Select bits (RS1 and RS0)

- These two bits are used to select one of four register banks of RAM.
- Below table shows the address ranges of four register banks along with RS1 & RS0 bits.

RS1	RS0	Register bank selected	Address range in the on chip RAM
0	0	Bank 0	00-07 H
0	1	Bank 1	08-0F H
1	0	Bank 2	10-17 H
1	1	Bank 3	18-1FH

c) F0

- F0 is available to user as a general purpose flag. This flag can be set/cleared by software, or its status can be observed by software.
- The user can define its role.

*Note:

1. PSW is a bit addressable register.
2. Each of the PSW bits is referred as PSW.X.
3. Thus, PSW.0 is the least significant bit (LSB), which is a parity flag, and the most significant bit (MSB) PSW.7 is the carry flag.

1.7.3. Special function registers

SFRs	Function
1. TMOD	<ul style="list-style-type: none"> • TMOD (Timer Mode) register is used to specify the mode of operation of timers/counters of 8051. • It is dedicated particularly to the two timers T0 and T1.
2. TCON	<ul style="list-style-type: none"> • TCON (Timer Control) register is used to control the entire operation of the timer/counter of 8051.
3. IE	<ul style="list-style-type: none"> • IE (Interrupt Enable) register is used to enable/disable all or any of the interrupts of 8051.
4. IP	<ul style="list-style-type: none"> • IP (Interrupt Priority) register is used to alter the priority (high

	or low) among the interrupts of 8051.
5. SCON	<ul style="list-style-type: none"> • SCON (Serial Control) register is used to specify the mode in which the serial port is to work.
6. PCON	<ul style="list-style-type: none"> • PCON (Power Control) register is used to control the baud clock generated from the timer.
7. SBUF	<ul style="list-style-type: none"> • SBUF is physically two registers. • One is write only and is used to hold data to be transmitted out of the 8051 via TXD. • The other is read only and holds received data from external sources via RXD.
8. P0, P1, P2, & P3	<ul style="list-style-type: none"> • These registers specify the value to be output on an output port or the value read from an input port. • They are also bit addressable. Each port is connected to an 8-bit register in the SFR.

1.8. Explain the function of various special function registers.

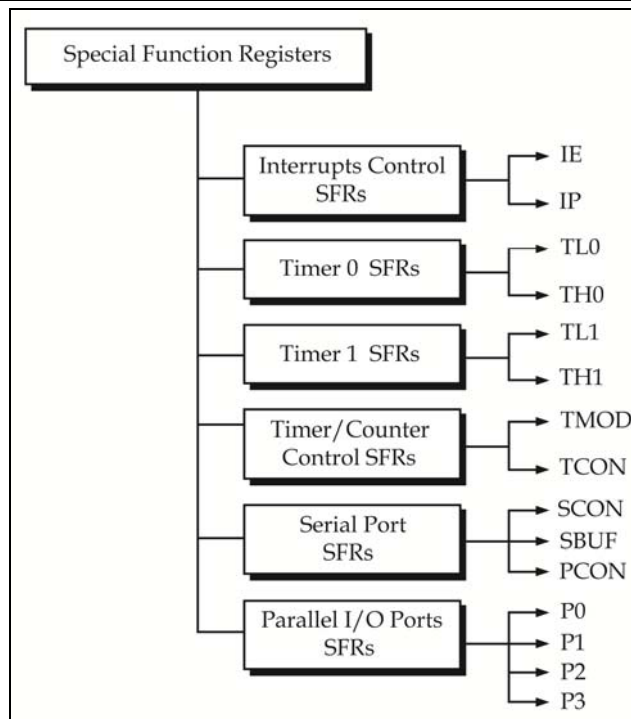


Fig. 1.8

Some of the special function registers are discussed in the previous section. Let us consider the remaining SFRs.

1.8.1. Interrupts control SFRs

The 8051 have five interrupts. There are two registers to handle interrupts. These are:

- a) IP (Interrupt Priority) SFR
- b) IE (Interrupt Enable) SFR

a) IP (Interrupt Priority) SFR

- It is an 8-bit SFR.
- It is used to alter the priority (high or low) among the interrupts of 8051.
- If the bit is 0, the corresponding interrupt has a lower priority and if the bit is 1, the corresponding interrupt has a higher priority.
- Bit addressing is allowed with this register also.

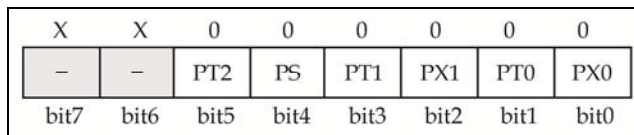


Fig. 1.8.1(a)

Bit	Symbol	Function
7	-	Not used (reserved for future).
6	-	Not used (reserved for future).
5	PT2	Not used (reserved for future).
4	PS	Serial port interrupt priority bit (Set to '1' for having highest priority)
3	PT1	Timer 1 interrupt priority bit (Set to '1' for having highest priority)
2	PX1	External interrupt 1 priority bit (Set to '1' for having highest priority)
1	PT0	Timer 0 interrupt priority bit (Set to '1' for having highest priority)
0	PX0	External interrupt 0 priority bit (Set to '1' for having highest priority)

*Note:

1. The priority bit may be set to 1 (highest) or 0 (lowest).
 - 0 → Low priority
 - 1 → High priority
2. Bit addresses: B8H to BFH

3. Upon the reset the 8051 the priorities among the five interrupts are listed in below table.

Interrupt	Priority
External interrupt 0	(Highest) ↓
Timer 0 interrupt	
External interrupt 1	
Timer 1 interrupt	
Serial port interrupt	

b) IE (Interrupt Enable) SFR

- It is an 8-bit bit addressable SFR.
- It is used to enable/disable all or any of the interrupts of 8051.
- If the bit is 0, the corresponding interrupt is disabled.
- If the bit is 1, the corresponding interrupt is enabled.
- Upon reset, all interrupts are masked (disabled).
- Fig. 1.8.1(b) shows the bit arrangement of IE register.

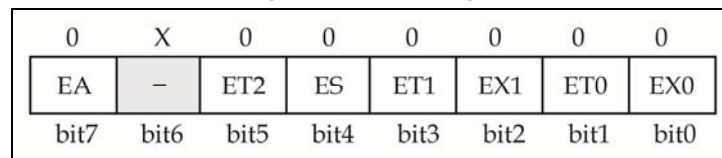


Fig. 1.8.1(b)

Bit	Symbol	Function
7	EA	Set to '0' to disable all interrupts. Set to '1' to control all the interrupts individually.
6	-	Not used (reserved for future).
5	ET2	Not used (reserved for future).
4	ES	SET to '1' to enable Serial port interrupt.
3	ET1	SET to '1' to enable Timer 1 interrupt.
2	EX1	SET to '1' to enable External interrupt 1.
1	ET0	SET to '1' to enable Timer 0 interrupt.
0	EX0	SET to '1' to enable External interrupt 0.

***Note:**

1. EA - Global interrupt enable/disable.
 - a. 0 - Disables all interrupt requests.
 - b. 1 - Enables all individual interrupt requests.
2. Bit address: A8H to AFH.

1.8.2. TL0, TH0, TL1, TH1 SFRs

The 8051 contain two 16 bit timer/counters named as T_0 and T_1 . Since the 8051 is a 8 bit microcontroller, these 16 bit registers are divided into two 8 bit registers. TL_0 and TH_0 and TL_1 and TH_1 .

TL0	:	Timer0 Low byte
TL1	:	Timer1 Low byte
TH0	:	Timer0 High byte
TH1	:	Timer1 High byte

1.8.3. Timer/Counter control SFRs

- a) TMOD (Timer/Counter Mode Control) SFR
- b) TCON (Timer/Counter Control) SFR

a) TMOD (Timer/Counter Mode Control) SFR

- TMOD is an 8-bit SFR.
- It is dedicated particularly to the two timers T0 and T1.
- This register is used to select the operating mode and the timer/counter operation of the timers.
- Fig. 1.8.3(a) shows the bit assignment for TMOD SFR.
- As shown in fig. 1.8.3(a) the lower four bits are used to control the timer 0 and the upper four bits are used to control the timer 1.

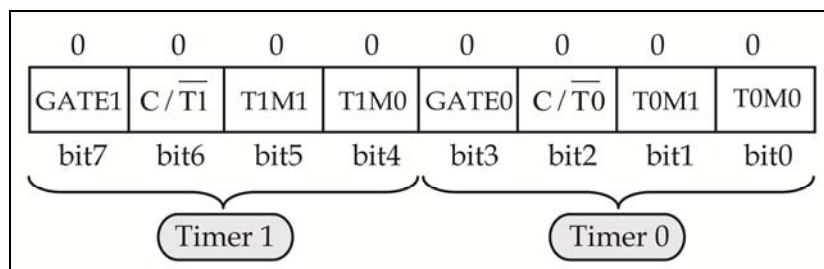


Fig. 1.8.3(a)

Symbol	Function
GATE	Gate control.
C/\bar{T}	Timer or counter selector.
M1 & M0	Timer modes select bits.

- **GATE:**
 - 1 → The timer operation is controlled by external sources through the $\overline{INT0}$ / $\overline{INT1}$ pins available in port 3 of the microcontroller 8051. This is hardware method of controlling the timer.
 - 0 → The timer operation is controlled by software commands such as SETB TR, CLR TR. This is software method of controlling the timer.
- C/\bar{T} :
 - 1 → T1/T0 will work as counters
 - 0 → T1/T0 will work as timers
- **M1, M0:** These two bits will decide the mode of operation of the timer/counter.

M1	M0	Operating Mode
0	0	Mode 0: 13-bit timer (TL-5 bits and TH-8 bits)
0	1	Mode 1: 16-bit timer (TL-8 bits and TH-8 bits).
1	0	Mode 2: 8-bit auto-reload 8-bit counter (TL-8 bit) overflow from TL not only sets TF, but also reloads TL with the contents of TH.
1	1	Mode 3: Split mode In this mode, real timer0 (TL0 & TH0) alone is being used independently as 8-bit timers.

b) TCON (Timer/Counter Control) SFR

- TCON is called timer/counter control register.
- It is also an 8-bit register.
- This register contains timer overflow flags, timer run control bits, external interrupt flags and external interrupt type control bits.
- Fig. 1.8.3(b) show the bits arrangement of TCON register.
- The higher nibble used to turn ON or OFF the specific timer.

- The lower nibble is meant for controlling the interrupts.

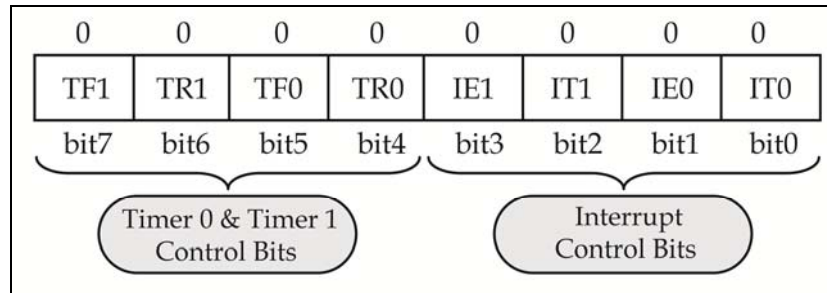


Fig. 1.8.3(b)

Bit	Symbol	Function
7	TF1	Timer 1 Overflow Flag. This bit is automatically set by hardware on the Timer 1 overflow. Cleared when interrupt processed.
6	TR1	Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.
5	TF0	Timer 0 Overflow Flag. This bit is automatically set by hardware on the Timer 0 overflow. Cleared when interrupt processed.
4	TR0	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.
3	IE1	Interrupt 1 Edge Flag. Set by hardware when $\overline{\text{INT1}}$ pin detects high to low transition. Cleared when interrupt processed.
2	IT1	Interrupt 1 type control bit. Set /cleared by software to specify falling edge/low level triggered external interrupts.
1	IE0	Interrupt 0 Edge Flag. Set by hardware when $\overline{\text{INT0}}$ pin detects high to low transition. Cleared when interrupt processed.
0	IT0	Interrupt 0 type control bit. Set /cleared by software to specify falling edge/low level triggered external interrupts.

***Note:**

- All flags can be set by the indicated hardware action.
- The flags are cleared when interrupt is serviced by the processor.

1.8.4. Serial Port SFRs

- SCON (Serial Control) SFR
- SBUF (Serial Data Buffer) SFR
- PCON (Power Control) SFR

a) SCON (Serial Control) SFR

1. It is an 8-bit bit addressable SFR.
2. It is a serial port mode control register and is used to control the operation of the serial port.
3. This register contains not only the mode selection bits, but also the specific bit for transmit and receive; and the serial port interrupt bits.
4. Fig. 1.8.4(a) shows the bits arrangement of SCON SFR.

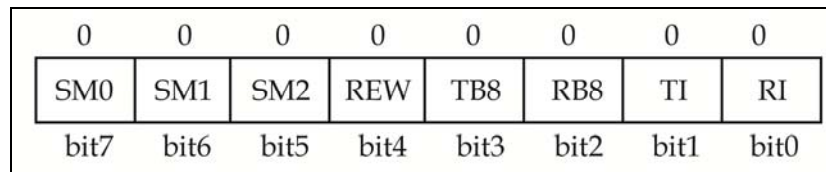


Fig. 1.8.4(a)

Bit	Symbol	Function
7	SM0	Serial port Mode control bit 0. Set/Cleared by software.
6	SM1	Serial port Mode control bit 1. Set/Cleared by software.
5	SM2	Serial port Mode control bit 2. It also known as multiprocessor communication enable bit. For most applications 8051 is not used in multiprocessor environment. Hence SM2=0.
4	REN	Receiver Enable control bit. Set/Cleared by software to enable/disable serial data reception.
3	TB8	Transmit Bit 8. It is the 9 th bit that will be transmitted in modes 2 and 3. It is set or cleared by software as desired.
2	RB8	Receiver Bit 8. In modes 2 and 3 this is the 9 th bit that was received (Cleared by hardware if 9 th bit received is logic 0. Set by hardware if 9 th bit received is logic 1). In mode 1 if SM2 = 0, RB8 is the stop bit that was received. In mode 0 RB8 is not used.
1	TI	Transmit interrupt flag. Set by hardware when byte transmitted. Cleared by software after servicing.
0	RI	Receive interrupt flag. Set by hardware when byte received. Cleared by software after servicing.

***Note:** Bit address 98H to 9FH

b) SBUF (Serial Data Buffer) SFR

- Computers must be able to communicate with other computers in modern multiprocessor distributed systems. One cost-effective way to communicate is to

send and receive data bits serially. The 8051 has a serial data communication circuit that uses register SBUF to hold data.

- SBUF is physically two registers. One is write only and is used to hold data to be transmitted out of the 8051 via TXD.
- The other is read only and holds received data from external sources via RXD. Both mutually exclusive registers use address 99H.

c) PCON (Power Control) SFR

1. PCON is a power control register of 8-bit length.
2. Its vectored address is 87H.
3. It is a byte addressable register.
4. It is used for power control and baud rate selection.
5. The PCON register also contains two general purpose flags and a double baud rate bit.
6. The fig. 1.8.4(b) shows the bit patterns for PCON SFR.

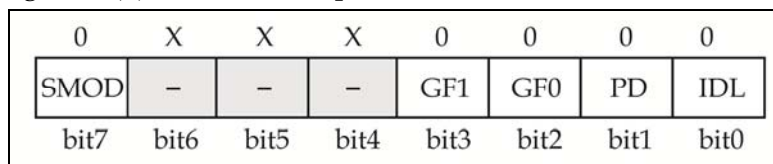


Fig. 1.8.4(b)

Bit	Symbol	Function
7	SMOD	Double baud rate bit.
6,5,4	-	Not implemented.
3	GF1	General purpose user flag bit 1 set/clear by program.
2	GF0	General purpose user flag bit 0 set/clear by program.
1	PD	Power down bit.
0	IDL	Idle mode bit.

***Note:** Bit addresses C8H to CFH

1.8.5. Parallel I/O Ports SFRs

1. In 8051 microcontroller, there are 4 ports for I/O operation. They are Port 0, Port 1, Port 2 and Port 3.
2. Each I/O port takes 8 pins. In the pin diagram of 8051 microcontroller, a total of 32 pins ($4 \times 8 = 32$) are occupied by the 4 I/O ports.
3. All 4 ports are bidirectional, i.e., each pin will be configured as input or output (or both) under software control.
4. The functions of 4 ports are listed in below table.

Port	Functions
Port 0	<ul style="list-style-type: none"> Used as an I/O Port Used as a bi-directional low-order address and data bus for external memory
Port 1	<ul style="list-style-type: none"> Used as an input/output Port
Port 2	<ul style="list-style-type: none"> Used as an input/output port Used as a higher-order address bus for external memory.
Port 3	<ul style="list-style-type: none"> Used as an input/output Port or used for alternate function as shown below. <p>P3.0 - RXD : Serial data input P3.1 - TXD : Serial data output P3.2 - INTO : External interrupt 0 P3.3 - INT1 : External interrupt 1 P3.4 - T0 : External timer 0 input P3.5 - T1 : External timer 1 input P3.6 - WR : External memory write signal P3.7 - RD : External memory read signal</p>

Additional Information

Special Function Registers

SFRs along with their direct addresses are listed in table.

Sl.No.	Symbol	Name	Address
1.	*ACC	Accumulator	0E0H
2.	*B	B Register	0F0H
3.	*PSW	Program Status Word	0D0HS
4.	SP	Stack Pointer	81H
5.	DPL	Data pointer low byte	82H
6.	DPH	Data pointer high byte	83H
7.	*P0	Port 0	80H
8.	*P1	Port 1	90H

9.	*P2	Port 2	0A0H
10.	*P3	Port 3	0B0H
11.	*IP	Interrupt priority Control	0B8H
12.	*IE	Interrupt Enable Control	0A8H
13.	TMOD	Timer/Counter Mode Control	89H
14.	*TCON	Timer/Counter Control	88H
15.	TH0	Timer/Counter 0 High Byte	8CH
16.	TL0	Timer/Counter 0 Low Byte	8AH
17.	TH1	Timer/Counter 1 High Byte	8DH
18.	TL1	Timer/Counter 1 Low Byte	8BH
19.	*SCON	Serial Control	98H
20.	SBUF	Serial Data Buffer	99H
21.	PCON	Power Control	87H

***Note:**

1. * = Bit Addressable.
2. Any address used in the program must start with a number. Thus, the addresses E0, F0, A0 etc., are specified as 0E0, 0F0, 0A0, etc.
3. Failure to use this number convention will result in an assembler error when the program is assembled.
2. The following two points should be noted about the SFR addresses.
 1. The SFRs have addresses between 80H and FFH.
 2. Not all the address space of 80H to FFH is used by the SFRs. The unused locations are reserved and must not be used by the 8051 programmer.

1.9. Draw the pin diagram of 8051 micro controller and specify the purpose of each pin.

1. The 8051 is packaged in a 40-pin DIP (Dual in line package).
2. It is important to note that many pins of 8051 are used for more than one function.
3. The alternative functions of pins are shown in parenthesis.
4. The 16 pins are used for single function. The remaining 24 pins may be used for one of two entirely different functions ($24 \times 2 = 48$ functions) as shown in fig. 2.85.
5. Fig. 1.9 shows the pin diagram of 8051.

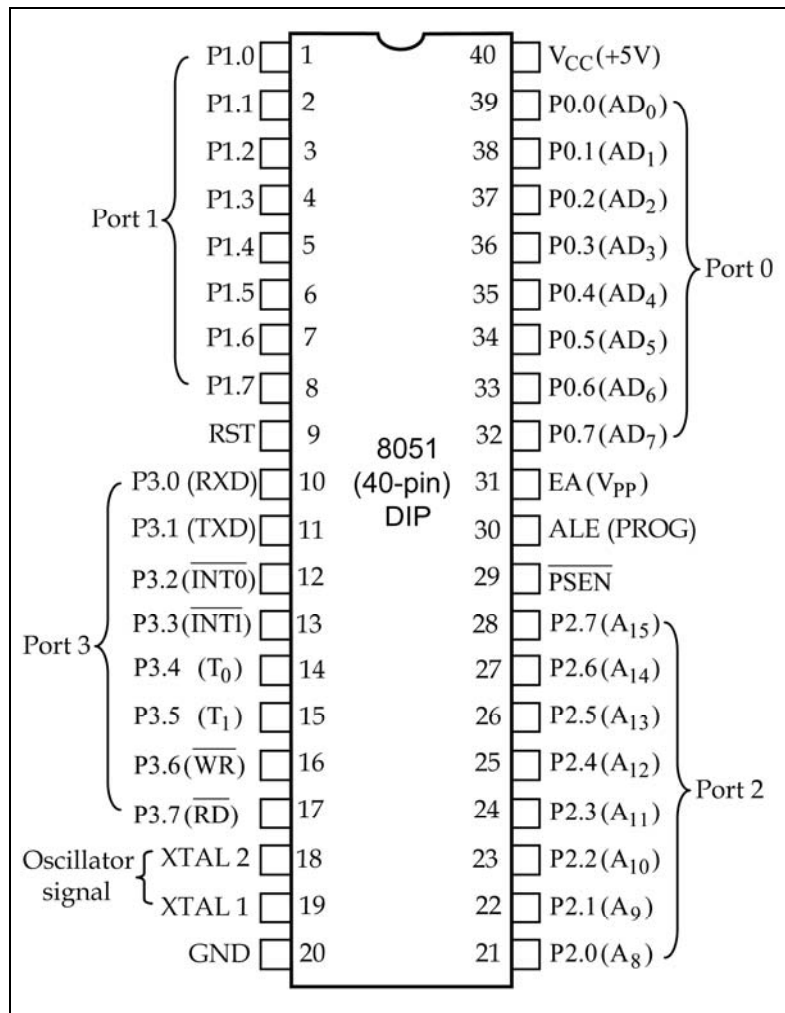


Fig. 1.9(a)

The description of each pin as follows.

Pin 1 - 8: *Port 1*

- A total of 8 pins named as P1.0 to P1.7 are belonging to port1.
- These pins can be used as input or output.

Pin 9: *Reset input (RST)*

- A logic one (high) on this pin resets the microcontroller.
- By applying logic zero to this pin, the program starts execution from the beginning.

Pin 10 - 17: *Port 3*

- Similar to port 1, each of these pins can serve as general input or output.
- In addition these lines provide alternative functions as listed in below table.

Pin No.	Designation	Function
10	P3.0 (RXD)	Serial data input.
11	P3.1 (TXD)	Serial data output.
12	P3.2 ($\overline{\text{INT0}}$)	External interrupt 0 input.
13	P3.3 ($\overline{\text{INT1}}$)	External interrupt 1 input.
14	P3.4 (T0)	External timer 0 input (or) Counter 0 clock input.
15	P3.5 (T1)	External timer 1 input (or) Counter 1 clock input.
16	P3.6 ($\overline{\text{WR}}$)	External memory write signal.
17	P3.7 ($\overline{\text{RD}}$)	External memory read signal

Pin 18-19: XTAL 1 and XTAL2

- The 8051 has on chip oscillator but requires an external clock to run it. Most often a quartz crystal oscillator is connected at these two pins.
- If you decide to use a frequency source other than a crystal oscillator, such as a TTL oscillator, it will be connected to XTAL1, XTAL2 is left unconnected.

Pin 20: V_{SS}

- This is power supply ground.

Pin 21 - 28: Port 2

- If there is no intention to use external memory then these port pins (P2.0 to P2.7) are configured as general inputs/outputs.
- In case external memory is used, the higher address (A8-A15) will appear on this port.

Pin 29: $\overline{\text{PSEN}}$

- Program Store Enable is the active low output pin.
- If external ROM is used for storing program then a logic zero (0) appears on it every time the microcontroller reads a byte from memory.

Pin 30: ALE

- Address Latch Enable is an active high output pin. It is used for de-multiplexing the address and data bus.
- This pin is also the program pulse input (PROG) during EPROM programming.

Pin 31: \overline{EA}

- External Access is an active low input pin.
- $\overline{EA} = 0$: Program code bytes can be fetched exclusively from an external ROM memory addresses 0000H to FFFFH.
- $\overline{EA} = 1$: Program code from address 0000H to 0FFFH is fetched from internal ROM and remaining code where address is 1000H to FFFFH is from external ROM memory.

Pin 32 - 39: Port 0

- Similar to P2, if external memory is not used, these pins can be used as general inputs/outputs.
- Otherwise this port provides low-order address (A7 - A0) and data bus (D7 - D0) for external memory.

Pin 40: V_{CC}

- 8051 operates on d.c power supply of +5V with respect to ground.
- The +5V is to be connected to pin V_{CC} .

Additional Information

Pin diagram of 8051

The fig. (a) shows pin diagram of 8051.

- The description of each pin as follows:

1. Power Supply Pins

- Pin number 40 and 20 are used as power supply pins.
- In which V_{CC} (+5) is given to the 40th pin and ground (V_{SS}) is connected to 20th pin.

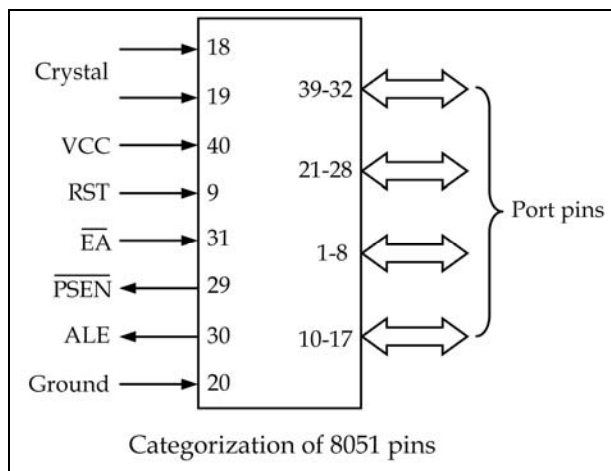


Fig. 1.9(b)

2. Crystal Pins

- The 8051 has an on-chip oscillator but it needs an external clock to activate it. A quartz crystal is connected between XTAL1 (19) and XTAL2 (18). These two pins are input pins.

3. Port Pins

- The 8051 consists four ports and each port needs 8-pins. The ports are Port 0, Port1, Port 2 and Port 3. The function of each port is given below.
- **Port 0:** Pin No. 32 to 39 is assigned to port 0. These pins are bi-directional pins (can be used as input or output). Port 0 is also the multiplexed low-order address and data bus during accesses to external code and data memory. This port needs pull-up resistors.
- **Port 1:** Port 1 requires 8 pins (pins 1 to 8). It can be used for input or output. This port need not require pull-up resistors.
- **Port 2:** Pin No.21 to 28 is assigned to port 2. It can be used for input or output. This port does not require any pull-up resistors. Port 2 is also used for higher order address pins (A8-A15)
- **Port 3:** Pin No.10 to 17 assigned to port 3. These pins can use for input or output. Port 3 has the additional function of providing some external important signals such as P3.0 and P3.1 are used for the RXD and TXD. These two pins are used for serial communication signals. P3.2 and P3.3 are used for external interrupts (INT0 and INT1). Pins P3.4 and P3.5 are used for timers (T 0and T1). Finally, P3.6 and P3.7 are used to provide control signals \overline{RD} and \overline{WR} .

4. RESET

- Pin no.9 is used as reset pin. It is an input pin. When this pin active high, the controller reset and terminate all activities. Hence this is referred as a power-on reset.

5. \overline{EA}

- \overline{EA} means external access.
- The pin no. 31 is assigned for \overline{EA} .
- It is an input pin and it must be connected to either Vcc or GND.
- If this pin is connected to the Vcc, the controller access the internal ROM, if this pin is connected to the GND, the controller access the external ROM.

6. \overline{PSEN}

- \overline{PSEN} means “program strobe enable”.
- This is an output pin.
- Pin no.29 is assigned to \overline{PSEN} .

- This pin is used to control the external memory.
- This pin is connected to OE pin of the ROM.

7. ALE

- Pin no.30 is assigned to ALE.
- ALE means address latch enable.
- This is an output pin.
- This pin is used to de-multiplex the data line from the address lines.

1.10. Explain internal memory Organization in 8051

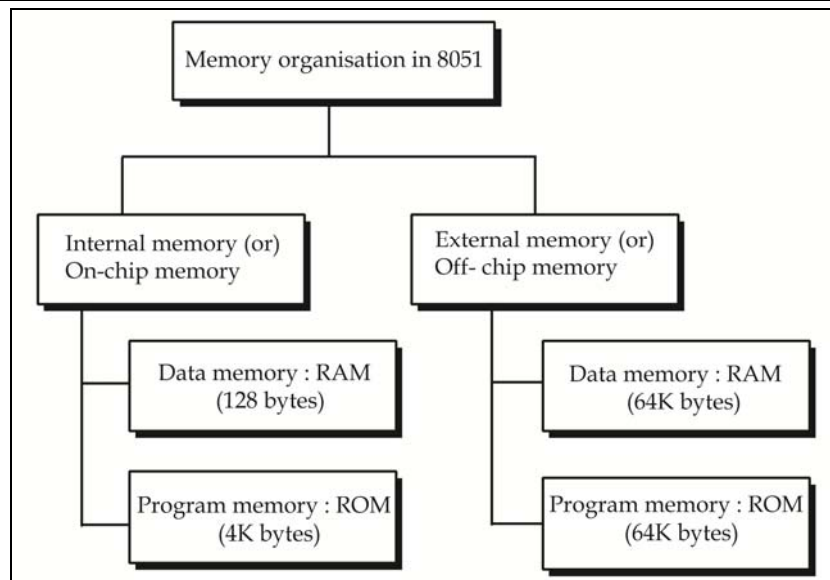


Fig 1.10

- A functioning computer must require memory for program and data. ROM is used for program code and data could be stored in RAM.
- Code memory is the memory that holds the actual program that is to be run. Data memory is the memory space where data are stored.
- The 8051 has internal RAM and ROM memory for these functions. Additional memory can be added externally using suitable circuits.

1.10.1. Internal Memory (or) On-chip memory

- 8051 has 128 bytes of RAM for data memory and 4K bytes of ROM for code memory inside the IC chip.
- These memories are called **internal memory or on chip memory**.

1. Internal RAM (i-RAM)

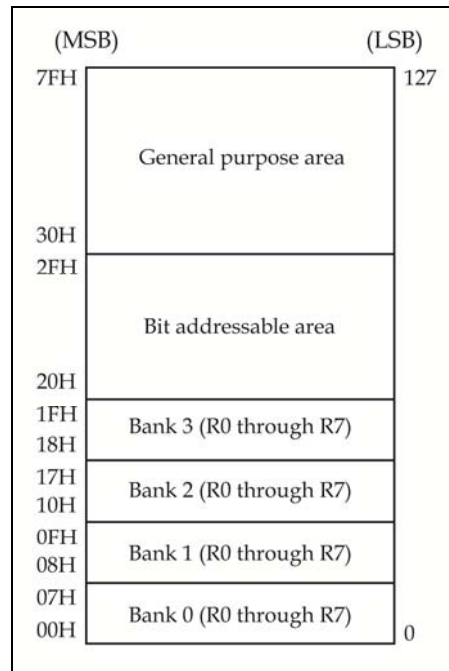


FIG 1.10.1(a)

- The fig. 1.10.1(a) shows the organization of internal RAM.
- The 128-bytes of internal RAM with address space from 00H to 7FH is divided into three groups as follows:

a) Working registers (32)

- The lowest 32 bytes (address 00H to 1FH) of the internal RAM form 4 banks of 8 registers each.
- The 4 register banks are numbered 0 to 3.
- Each bank is made up of 8 registers named R0 to R7.
- Only one register bank can be accessed by 8051 at a time.
- Default bank (On reset) is bank 0 (locations 00 – 07H).
- The bits RS1 (PSW.4) and RS0 (PSW.3) of program status word (PSW) are used to select any one of the four register banks.

b) Bit Addressable area

- The 8051 provides 16 bytes of a bit-addressable area.
- It occupies RAM byte addresses from 20H to 2FH, forming a total of 128 (16 x 8) addressable bits.

c) General purpose

- The last 80 bytes (30H to 7FH) of 8051 internal RAM is called general purpose RAM or scratch pad memory locations.
- The scratch pad area is used for general purpose storage i.e. to store data, results, constants and intermediate results.

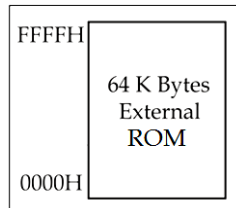


Fig 1.10.1(b)

2. Internal ROM (i-ROM)

- The 8051 has 4 Kbytes of internal ROM with address space from 0000H to 0FFFH.
- This is used to store final version of the program codes hence the name program memory.

Note: It is programmed by manufacturer when the chip is built. This part cannot be erased or altered after fabrication.

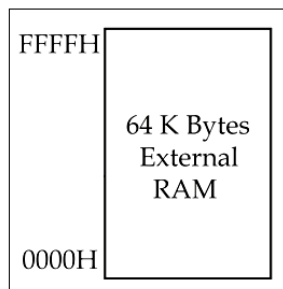


Fig 1.10.1(c)

1.11. Explain external memory access in 8051

- The 8051 can address external memory if there is not enough internal RAM and/or ROM.

1. External RAM or (Off-chip RAM)

- The 8051 has external RAM of 64k bytes.
- The range of external data memory is from 0000H to FFFFH.
- This memory space is used for storing data so frequently called as data memory.

2. External ROM or (Off-chip ROM)

- The 8051 MC has 64k bytes of external program memory.

- Fig. 1.11 shows the map of 8051 program memory.
- The control pin \overline{EA} (pin 31) determines the accessing of either internal or external ROM.

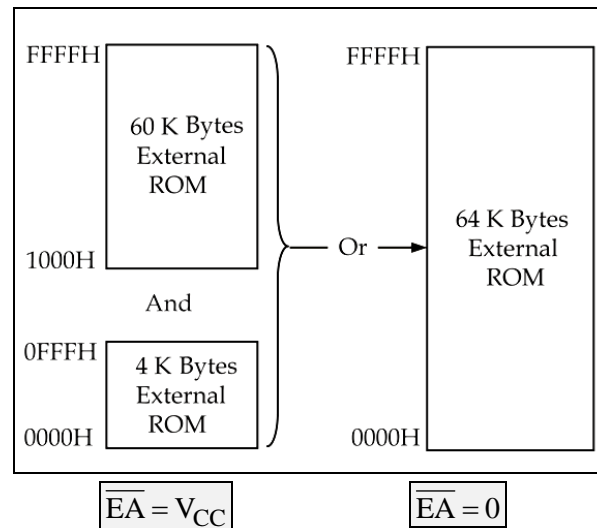


Fig. 1.11

- Let \overline{EA} pin is at V_{CC} means that upon reset the 8051 access the on-chip RAM first (0000 H - 0FFF H) and when reaches (to 0FFF H) end of the on-chip ROM it switches to off-chip ROM (1000H - FFFFH). This type of accessing is widely used in practice.
- If \overline{EA} pin is grounded then only external (off-chip) ROM of 64 KB space can be accessed by the Program Counter.

Do You Know?

External data memory:

1. To access external data memory, microcontroller use the Pins \overline{RD} or \overline{WR} .
2. The external data memory is accessed by using DPTR.
3. The external data memory is accessed using indirect addressing mode only.
4. To access the external data memory MOVX instruction is used.
5. MOVX means Move External. This instruction will transfer data between the accumulator and a byte of external data memory.
6. Access to external data memory can use either one byte address [$@ Ri$ where Ri can be either $R0$ or $R1$ of the selected register bank] or two byte address [$@ DPTR$].
7. Accumulator is always the destination or source during external data RAM accesses.

External program memory:

8. The MOVC instructions can be used to access the internal or external program memory.
9. It is accessed by the DPTR and PC only
10. It is accessed by using indexed addressing mode.
11. The External Code memory can be used not only for storing the program (code), but also for lookup table of different functions required for specific task. Mathematical functions such as Square root, Sine, quadratic equations etc. can be stored in the code memory (internal or External) and these functions can be accessed by using MOVC instruction.
12. To access external code (program) memory $\overline{\text{PSEN}}$ (program store enable) is used as read strobe.
13. External program memory (ROM) is accessed under two conditions:
 - a) Whenever $\overline{\text{EA}} = 0$ (or)
 - b) Whenever PC contains a number that is larger than 0FFFH.

1.12. Explain various ports of 8051.

- The 8051 has four input /output ports.
- The 8051 has a group of 32 I/O pins configured as four 8-bit parallel ports named as P0, P1, P2 and P3 .
- All four ports are bidirectional, i.e, each pin will be configured as input or output (or both) under software control.
- Each port consists of a latch, an output driver and an input buffer.
- All the ports are configured as input upon the RESET of 8051 and ready to be used as input ports.
- In order to change them as an output ports, they must be programmed. In addition to the I/O nature, these lines also have some specific functions.
- The following two instructions are used to make the Port 0 as an output port.

MOV A, #00H	00 H loaded in to A-register.
MOV P0, A	Making Port 0 as an output port while writing 0s (0000 0000).

- The following two instructions are used to make the Port 0 as an input port.

MOV A, #0FFH	FF H loaded in to A-register.
MOV P0, A	Making Port 0 as an input port while writing 1s (1111 1111).

1. Port 0 (P0)

- It is designated as P0; the bits of port 0 are designated as P0.0 to P0.7.
- It is a bit as well as byte addressable register.
- Port 0 pins are also named as AD0-AD7 i.e. the pins of Port 0 allowed to use as address as well as data pins depends upon the status of the ALE signal.
- It has dual function.
- The internal RAM address of Port 0 is 80H.
- **Port 0 is used as:**
 - Input port for transferring data from peripherals to microcontroller.
(or)
 - Output port for transferring data from microcontroller to peripherals.
(or)
 - Lower order address bus (A0 to A7) for external memory.
(or)
 - Data bus (D0 to D7) for external memory.

***Note:** To use the pins of port 0 as both input and output, each pin must be connected externally to a 10 k Ω pull-up resistor.

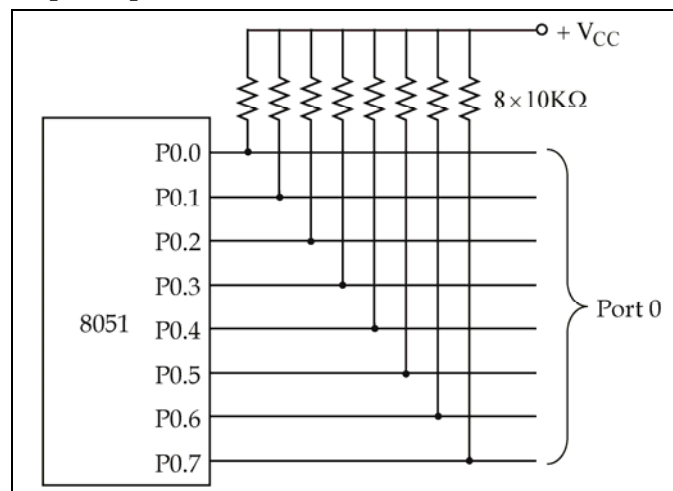


Fig 1.12

Note: The ports P1, P2 and P3 do not require any pull-up resistors since they already have pull-up resistors internally.

2. Port 1 (P1)

- It is designated as P1; the bits of port 1 are designated as P1.0 to P1.7.
- Port 1 has mono function i.e. it is used as input or output port.

- It is a bit as well as byte addressable register.
- The internal RAM address of Port 1 is 90H.
- It can be accessed by direct or indirect address modes.
- The port 1 does not need any pull-up resistors since it already has built in pull up resistors internally.
- When the microcontroller is reset, port 1 is configured as an input port.

3. Port 2 (P2)

- It is designated as P2, the bits of port2 are designated as P2.0 to P2.7.
- Port 2 has dual function, i.e. Port 2 can be used as I/O port as well as higher order address bus (A8 to A15) .
- When external memory is interfaced with 8051, Port2 pins can't be used as I/O port it works as higher order address bus (A8-A15).
- It is a bit as well as byte addressable register.
- The internal RAM address of Port 2 is 0A0H.
- It can be accessed by direct or indirect address modes.
- The port2 does not need any pull-up resistors since it already has built in pull up resistors internally.
- All the pins of port2 are compatible with TTL circuits.
- When the microcontroller is reset, port 2 is configured as an input port.

Note: Port 2 is used as:

○Input port

(or)

○Output port

(or)

○High-order address bus (A8 to A15) for external memory.

4. Port 3 (P3)

- It is designated as P3; the bits of port3 are designated as P3.0 to P3.7.
- Port 3 also have dual functions.
- Similar to the Port 1 and Port 2, the Port -3 also can be used as input or output. However Port-3 lines are commonly used for special functions. The special function of Port-3 was mentioned in below table.
- It is a bit as well as byte addressable register.
- The internal RAM address of Port 3 is 0B0H.
- It can be accessed by direct or indirect address modes.

- The port3 does not need any pull-up resistors since it already has built in pull up resistors internally.

P3.0	RXD	Serial data input
P3.1	TXD	Serial data output
P3.2	$\overline{\text{INT0}}$	External interrupt 0
P3.3	$\overline{\text{INT1}}$	External interrupt 1
P3.4	T0	External timer 0 input
P3.5	T1	External timer 1 input
P3.6	$\overline{\text{WR}}$	External memory write signal
P3.7	$\overline{\text{RD}}$	External memory read signal

Note:

1. RXD and TXD pins are used as input and output pin if 8051 is operated in serial data transmission mode.
2. Through T0 and T1 pins the external clock pulse are given to counter 0 and counter 1 respectively for counter operation.

1.13. Explain counters & timers in 8051

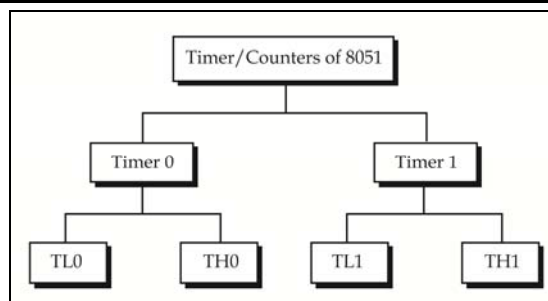


Fig 1.13(a)

1. 8051 has two 16-bit timer/counters, designated as Timer 0(T0) and Timer 1(T1). They can be used as timers or counters.
2. Since the 8051 is an 8-bit controller, the 16-bit timer/counter registers are divided into two 8-bit registers called the timer low (TL0, TL1) and high (TH0, TH1) bytes.

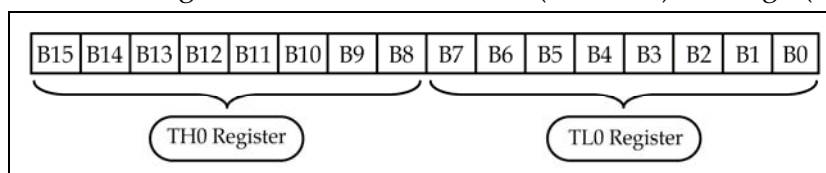


Fig.1.13(b)

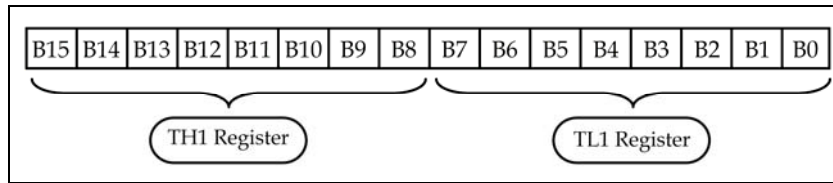
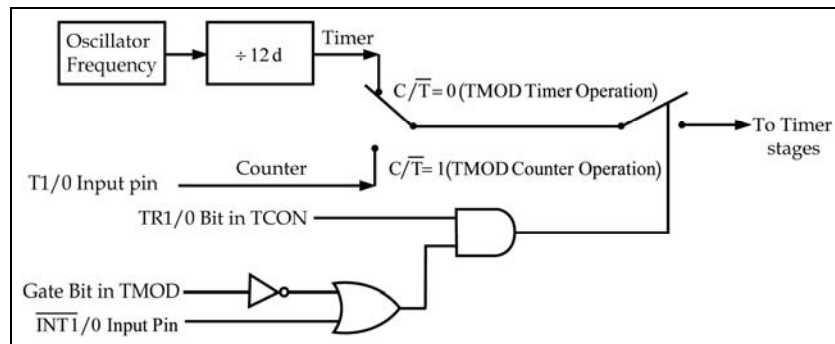


Fig. 1.13(c)

3. A timer always counts up. It doesn't matter whether the timer is being used as a timer, a counter, or a baud rate generator: A timer is always incremented by the microcontroller.
4. Two SFRs TMOD (Timer Mode Control SFR) and TCON (timer control SFR) are used to monitor the nature of working of timer/counters.
5. The timers can be configured to operate either as timers or event counters. The timer/counter mode is selected by the control bit C/\bar{T} in the special function register TMOD.
6. Counters are used to count the events where as timers are used to maintain time delays between the actions.
7. In timer mode, it will count the internal clock frequency of the 8051 divided by 12 d. The register is incremented for every machine cycle.
8. In counter mode, the register is incremented in response to 1 to 0 transition at its corresponding external input pins T_0 and T_1 .



Additional fig:1.13(d) Timer / Counter control logic circuit

1.13.1. Operational Modes of Timers

Timer has four modes of operation. The modes of operation of the timer is depends on the bits of the TMOD register.

a) Mode 0 Operation

- In this mode timer can acts as 13 bit timer.
- Here 8 bits of TH0 and 5 bits of TL0 or 8 bits of TH1 and 5 bits of TL1 are alone used. TL (TL0 or TH0) will count from 0 to 31.
- For the next count, it will "reset" to 0 and increment TH. Thus, effectively, only 13 bits of the 16 bit timer are being used: bits 0-4 of TL and bits 0-7 of TH.

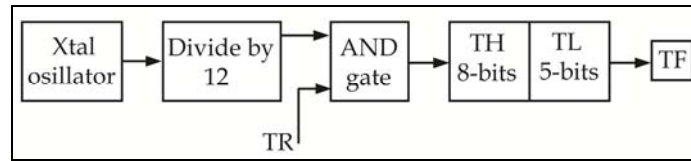


Fig. 1.13.1(a)

- Internal clock pulses are divided by 12 counter and fed to one input of AND gate.
- Whenever TR is set by the program the counting starts. When the count reaches 1FFFH (8192D) the overflow flag sets.

Note: If a 13-bit timer has an initial value of 0000, the overflow flag will set to 1 after 8192 machine cycles.

b) Mode 1 Operation

- In this mode timer can acts as 16 bit timer.
- Here both the registers TL and TH are used to hold the 16-bit value. TL is incremented from 0 to 255. When TL reaches 255 it resets and causes TH to be incremented by 1. Thus both the registers are used effectively to hold 16-bit value.

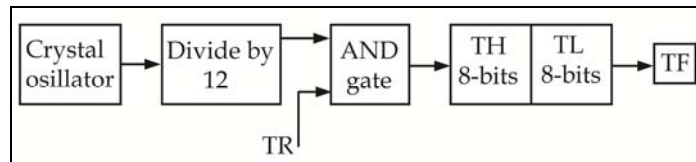


Fig. 1.13.1(b)

TF will set when timer rolls over from FFFF to 0000.

c) Mode 2 Operation

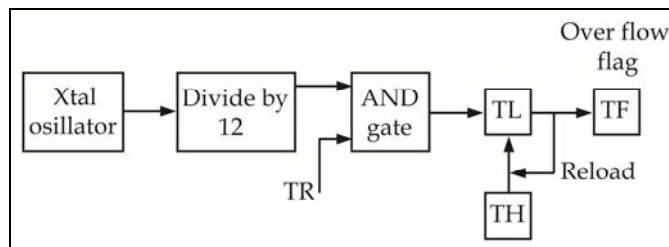


Fig. 1.13.1(c)

- The mode 2 operation is called 8-bit auto reloaded mode. In this mode TH holds the "reloaded value" and TL is the timer itself.
- The timer register TL is loaded with value between 00 and FF. Once TL is loaded with a value, a copy of this value will be stored in TH.
- When the timer operation starts TL starts counting up from the set value. Once TL value reaches FF, the initial value stored in TH will be automatically reloaded into the TL register and the TL starts its operation once again automatically.

- TF goes high when timer rolls over from FFH to 00H.

d) Mode 3 Operation

- In this mode timer 0 is configured as two separate 8 bit timers and timer-1 is stopped.
- The TL₀ is functioned as 8 bit timer and controlled by timer 0 control bits i.e., TR₀ and TF₀ of TCON.
- The TH₀ is functioned as 8 bit timer and controlled by timer 1 control bits TR₁ and TF₁ of TCON.

Do You Know?

1. Since the timer T0 is virtually 16-bit register, the largest value it can store is 65 535. In case of exceeding this value, the timer will be automatically cleared and counting starts from 0. This condition is called an **overflow**.

1.14. Explain serial input/output of 8051

- The 8051 is a 8 bit microcontroller and it transfer 8-bit data simultaneously. This is the parallel I/O mode. It is an expensive method as it requires eight transmission lines between sender and receiver.
- Particularly it is very difficult to adopt this parallel data transfer mode in long distance communication.
- In situations, the Serial Input and Output mode is used, whereby one bit at a time is transferred over a single line.
- The serial input output port provides serial data transfer. Serial data communication is one of the effective ways of transmitting and receiving data bits between computer systems.
- The serial port of 8051 provides full duplex data communication.
- It can transmit and receive data simultaneously.
- The 8051 has a serial data communication circuit (**UART**: Universal Asynchronous Transmitter and Receiver) that uses;
 - Register SBUF to hold data.
 - Register SCON controls data communication.
 - Register PCON controls data rates.
 - Pins RXD (P3.0) and TXD (P3.1) connect to the serial data network.
- The 8051 provides four programmable modes for serial data communication.
- A particular mode can be selected by setting the SM0 and SM1 bits in SCON. The mode selection also decides the baud rate.

SM0	SM1	Mode	Function
0	0	0	Shift register, baud rate = $f/12$
0	1	1	8-bit UART, baud rate= variable
1	0	2	9-bit UART; baud rate = $f/32$ or $f/64$
1	1	3	9-bit, UART : baud rate =variable

1.15. Explain interrupts in 8051

- An interrupt is a signal either from hardware or from a program that interrupts the normal processing of the CPU.
- Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device.
- The program associated with the interrupt is called interrupt service routine (ISR) or interrupt handler.
- When an interrupt is invoked, the microcontroller runs the interrupt service routine.
- The 8051 provides 5 interrupts.
- 2 interrupts are external interrupts and the remaining 3 are internal interrupts. These are :
 - External Interrupts: $\overline{INT0}$ & $\overline{INT1}$.
 - Internal Interrupts: Timer 0, Timer 1 serial communication.

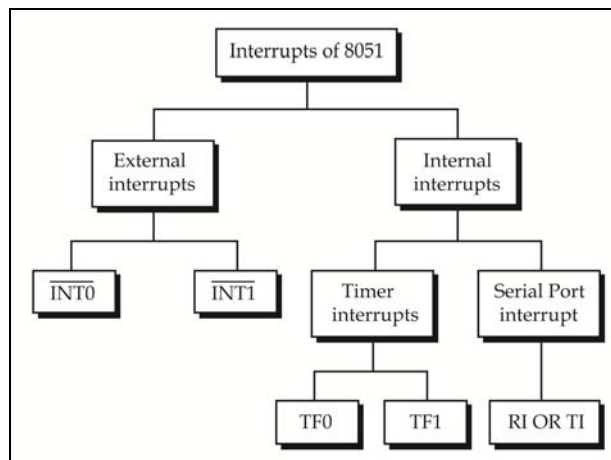


Fig 1.15

***Note:**

1. For example, if $\overline{INT0}$ and TF1 are activated at a time then the 8051 will respond to $\overline{INT0}$ initially as it has higher priority than TF1.
2. The priorities of the interrupts can also be altered by programming the IP register.
3. The external interrupts can be programmed to be either level active or transition activated by setting or clearing the bit IT1 or IT0 in register TCON.
4. All the interrupts of 8051 are maskable interrupts. It means that all the interrupts of 8051 can be individually enabled or disabled by setting or clearing a bit in IE SFR. The IE register contains also an interrupt disable bit EA, which disables all interrupts when it is 0.
5. In order to handle five interrupts the 8051 use SFRs, such as IE, IP and TCON.

Review Questions

1. Draw the pin-out diagram of 8051 and label the names. [April -2012]
2. List any six special function registers of 8051 microcontroller. [April/May-2015, April -2008]
3. Explain input/output ports of 8051 [Oct/Nov-2011]
4. List the interrupts in 8051 microcontroller [March/April-2014]
5. Draw the functional block diagram of 8051. Explain the functions of each block. [April/May-2015, 2014, 2013; Oct/Nov-2013]
6. Explain memory organization of 8051 [April-2012; Oct/Nov-2011]
7. Explain the internal RAM-organization in 8051 [April-2012]
8. Explain the pin diagram of 8051 microcontrollers. [April-2008, 2009]
9. Explain the internal memory organization of 8051. [April-2012]
10. Explain the function of various special function registers. [Oct/Nov-2011]
11. Explain the following:
 - a) Timers/ Counters of 8051
 - b) Input/ Output ports
 - c) Interrupts of 8051[March/April-2013]

CHAPTER 2

Instruction set of 8051

OBJECTIVES

- 2.1 State the need for an instruction set.
- 2.2 Write the instruction format of 8051 & illustrate these terms by writing an instruction.
- 2.3 Explain fetch cycle, execution cycle and instruction cycle.
- 2.4 Distinguish between machine cycle and T-state.
- 2.5 Explain the timing diagram for memory write & Memory read operation of 8051
- 2.6 Define the terms machine language, assembly language, and mnemonics.
- 2.7 Write the differences between machine level and assembly level programming.
- 2.8 Classify the instruction set of 8051.
- 2.9 Explain one byte, two byte and three byte instructions of 8051.
- 2.10 List the various addressing modes of 8051 and Explain with examples.
- 2.11 Explain data transfer instructions of 8051.
- 2.12 Explain the arithmetic instructions and recognize the flags that are set or reset for given data conditions.
- 2.13 Explain the logical instructions and recognize the flags that are set or reset for given data conditions.
- 2.14 Explain bit-level logical instructions.
- 2.15 Explain Boolean group of instructions.
- 2.16 Explain unconditional & conditional jump instructions.

2.1. State the need for an instruction set.

- Instruction set is also called a command set. Instruction set means a group of instructions that a microcontroller understand.
 - The instruction set of a microcontroller used to allow efficient control of both its internal devices and the surrounding infrastructures (i.e. those devices are connected to the ports of the microcontroller).
 - Control of the register set of the microcontroller in an easy manner.
 - Ability to access ports and other peripheral control and status registers.
 - Ability to be able to access individual bits of a port or special function registers.
 - Ability to access internal and external memory.
-

2.2. Write the instruction format of 8051 & illustrate these terms by writing an instruction.

- A set of rules defining the way the operands, data and addresses are arranged in an instruction is known as instruction format.
- An instruction is a command given to the microcontroller to perform a given task on specified data. Each instruction has two parts as shown in fig. 2.2.
- Each instruction has two parts consists of two parts, they are opcode and operand .The fig. shows the general format of the instruction.

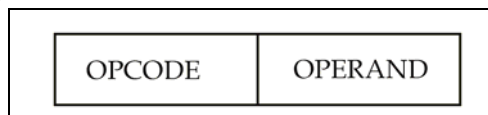


Fig. 2.2

1. Operation code (op-code)

- The part of the instruction that specifies the operation to be performed is called the operation code or op-code.

2. Operand

- The data on which the operation is to be performed is called the operand.
- Operand consists two fields, such as **Destination and Source**.
- The destination field may be a register or an address or port /port address or a memory location. Destination field specifies the destination for the data that is being copied form the source.
- Let us consider an instruction CPL A. The meaning of this instruction is complementing the contents of accumulator. In this instruction CPL is op-code and accumulator (A) is an operand.

Examples:

Instruction		Comment
Op-code	Operand	
ADD	A,Rn	Add register Rn to Accumulator.
MOV	A,#data	Move immediate data to Accumulator.
MOV	direct,Rn	Move register Rn to direct byte.
CLR	bit	Clear direct bit.
SJMP	rel	Jump if Carry is set.

2.3. Explain fetch cycle, execution cycle and instruction cycle.

The total time required to perform the execution of an instruction is known as the **instruction cycle**. Every instruction consists of two parts i.e., opcode and operand. Therefore the instruction cycle is broken into two time intervals known as **Fetch cycle** and **Execute cycle**.

1. Fetch cycle

The time taken by the 8051 for fetching an opcode is known as Fetch cycle. During the fetch cycle, an instruction byte containing the operation code is brought into the microcontroller from memory.

(or)

It is the time required to bring an instruction byte containing the operation code from the memory into the microcontroller. A typical fetch cycle is shown in the fig. 2.3.(a).

The entire operation of fetching an op code takes three states (S) or six clock cycles (P).

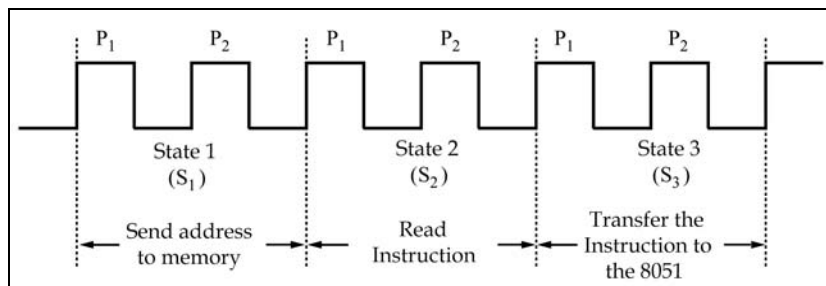


Fig. 2.3.(a)

***Note:**

The fetch operation can be explained by a simple algorithm.

1. Begin
2. Send the address of an instruction to memory.
3. Read the instruction from the memory.

4. Transfer the instruction to the microcontroller.
5. END

2. Execute Cycle (EC)

It is the total time required to decode the instruction fetched and execute it. If the operand is in memory, execution is immediately performed.

Note that if an instruction contains data or operand and address which are still in the memory, the processor has to perform one or two memory read operations to get the desired data. A typical execute cycle is shown in the fig. 2.3(b).

The time required for an execute cycle will depends on the instruction type.

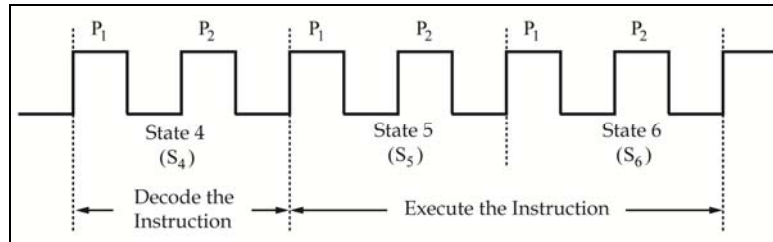


Fig. 2.3(b)

*Note:

It is explained by a simple algorithm.

1. Begin
2. Decode the instruction fetched
3. If operand is in memory, fetch the operand
4. Execute the instruction
5. End.

3. Instruction Cycle

It is the time required for the microcontroller to complete the execution of an instruction. An instruction cycle consists of a fetch cycle and execute cycle i.e. $IC = FC + EC$.

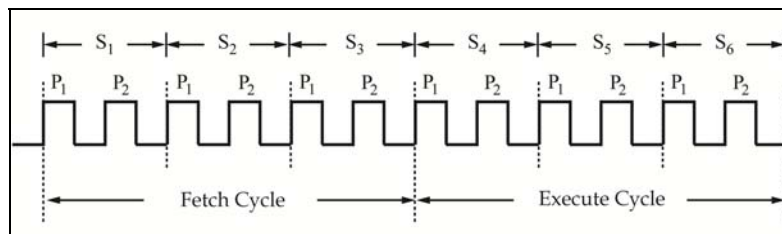


Fig. 2.3(c)

2.4. Distinguish between machine cycle and T-state.

The external basic operations performed by the microcontroller are called machine cycles. The 8051 microcontroller takes one to four machine cycles to execute an instruction.

The basic timing of the 8051 machine cycle is shown in fig. 2.4. The entire timing of a machine cycle of 8051 is divided into 6-states and they are denoted as S1, S2, S3, S4, S5 and S6. The timing of each state is two clock periods and they are denoted as P1 and P2.

A **state** in a machine cycle is a basic time interval for discrete operation of the microcontroller such as fetching an op-code byte, decoding an op-code, excluding an op-code, writing a data, etc.

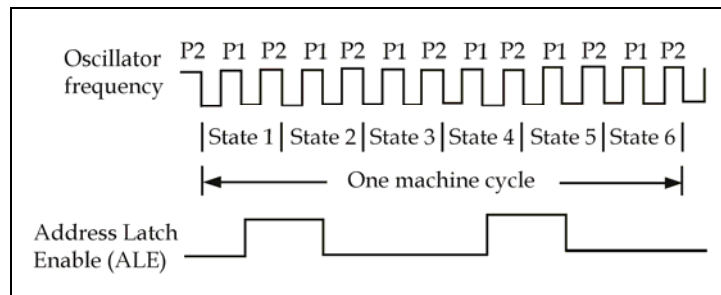


Fig. 2.4

The time taken to execute a machine cycle is 12 clock periods and so time taken to execute an instruction is obtained by multiplying the number of machine cycles of that instruction by 12 clock periods.

$$\text{Instruction execution time} = C \times 12 \times T = C \times 12 \times \frac{1}{f}$$

Where, C = Number of machine cycles of an instruction
 T = Time period of crystal frequency in seconds
 f = Crystal frequency in Hz.

*Note:

1. If the oscillator frequency is 12 MHz, then the time taken to execute one machine cycle is 1 microsecond.
2. An 11.0592 MHz crystal yields a cycle frequency of 921.6 kilohertz, which can be divided evenly by the standard communication baud rates of 19200, 9600, 4800, 2400, 1200 and 300 Hz.
3. The 8051 microcontroller has four machine cycles and they are:
 - a) External program memory fetch cycle
 - b) External data memory read cycle
 - c) External data memory write cycle
 - d) Port operation cycle.

4. In fig. 2.4 there are two ALE pulses per machine cycle. The ALE pulse, which is primarily used as a timing pulse for external memory access, indicates when every instruction byte is fetched.
5. Two bytes of single instruction may thus be fetched, and executed, in one machine cycle.
6. Single byte instructions are not executed in a half cycle, however single byte instructions "throw away" the second byte (which is first of the next instruction). The next instruction is then fetched in the following cycle.

2.5. Explain the timing diagram for memory write & Memory read operation of 8051

1. Interfacing External Program (ROM) Memory

In 8051 when the \overline{EA} pin is connected to V_{CC} , program fetches to addresses 0000H through 0FFFH are directed to the internal ROM and program fetches to addresses 1000H through FFFFH are directed to external ROM/EPROM. On the other hand when \overline{EA} pin is grounded, all addresses (0000H to FFFFH) fetched by program are directed to the external ROM/EPROM. The \overline{PSEN} signal is used to activate output enable signal of the external ROM/EPROM, as shown in fig. 2.5(a).

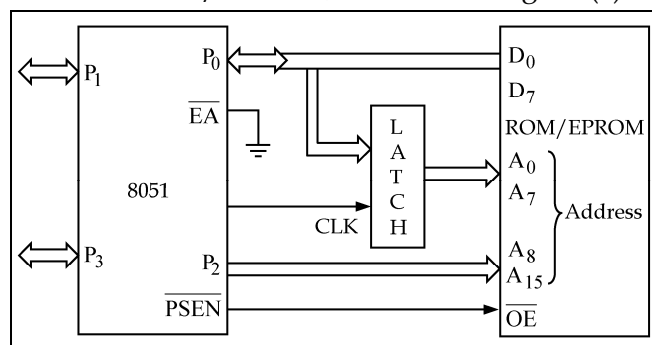


Fig. 2.5(a)

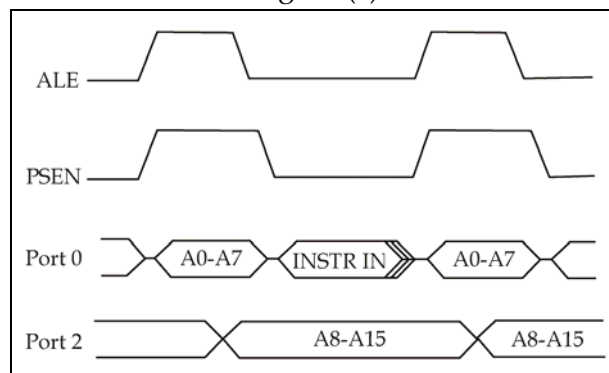


Fig. 2.5(b)

The circuit arrangement for connecting external program memory is shown in fig. 2.5(a).

- The port 0 is used as a multiplexed address/data bus. It gives lower order 8-bit address in the initial T-cycle and later it is used as a data bus.
- The 8-bit address is latched using external latch and ALE signal generated by 8051.
- The port 2 provides the higher order 8-bit address.
- The timing waveforms for external memory read cycle, is as shown in fig. 2.5(b).

2. Interfacing External Data (RAM) Memory

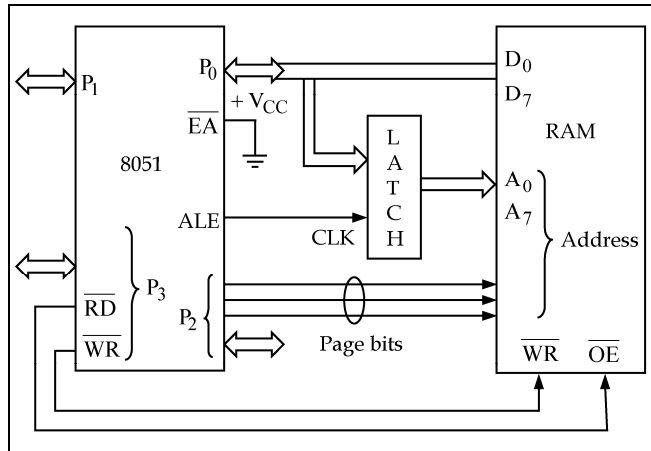


Fig. 2.5(c)

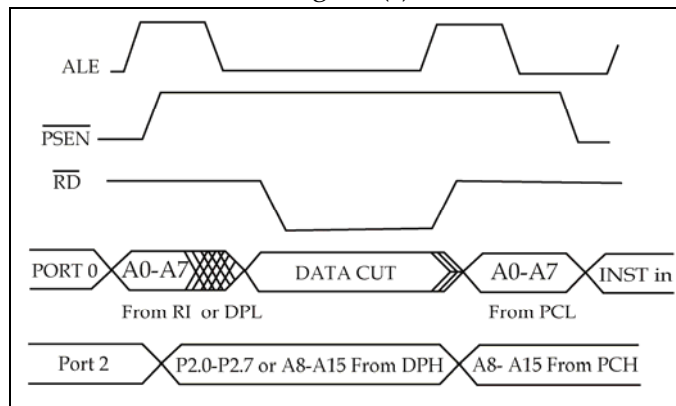


Fig. 2.5(d)

8051 can address up to 64 Kbytes of external data memory. The “MOVX” instruction is used to access the external data memory.

The circuit arrangement for connecting external data memory is shown in fig. 2.5(c).

- The multiplexed address/data bus provided by port 0 is demultiplexed by external latch and ALE signal.

- Port 2 gives the higher order address bus.
- The \overline{RD} and \overline{WR} signals from 8051 selects the memory read and memory write operation, respectively.
- Now the timing waveforms of external data memory for read and write cycles are as shown in fig. 2.5(d) and fig. 2.5(e) respectively.

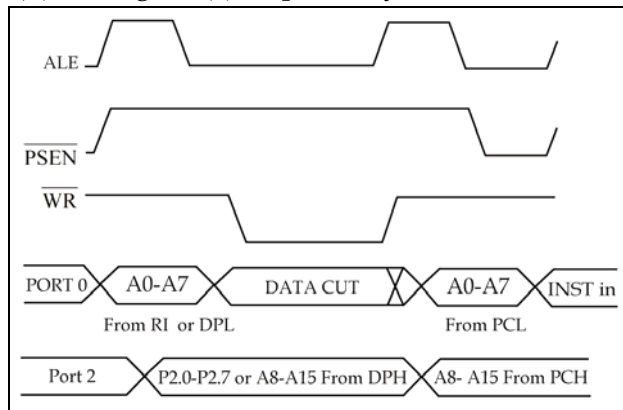


Fig. 2.5(e)

2.6. Define the terms machine language, assembly language, and mnemonics.

When anyone wants to communicate to any other person then a language is followed. So if we want to communicate with computer system then we have to use computer language.

The microcomputer programming languages can be divided into the following three types.

1. Machine language
2. Assembly language
3. High Level language

***Note:** The machine language and assembly language are considered as low level language.

1. Machine language

A program written in the form of 0s & 1s is called machine language also known as binary language as only two bits are being used. To write a program in machine language programmer has to memorize the hundreds of binary instruction codes for a processor. This task is difficult and error prone.

For example, for INTEL 8051 to add the contents of register A and register R7, the binary code is 0010 1111₂(2FH). To move the content of register R7 to register A the binary code is 1110 1111₂(EFH). Thus by observing the two examples, it is very difficult for the programmers, to write a program in machine language.

However this language is preferred for simple programs and control applications where less computation is required..

Characteristics:

1. Easily understood by the computer system.
2. Very fast in execution and no need of hardware/ circuits/ Compiler/ Assemblies.
3. Difficult to write and understand.
4. Difficult to debug.
5. Very slow to enter.
6. Not readable.
7. Machine dependent.

2. Assembly language

A program written in mnemonics is known as assembly language program. Mnemonic is a group of alphabets which suggest operation to be performed by that instruction. A few examples are ADD for addition, XCH for exchange, and MOV A, Rn transfer data from register Rn to accumulator.

The writing of program in assembly language is much easier as compared to the writing of a program in machine language. The machine and assembly language instructions are specific to each microcontroller; they are not transferable to other microcontroller. In some times it is also called as one to one language since each mnemonic refers to unique operation.

Characteristics:

1. Very easy to read
2. Easy to debug
3. Easy to write
4. It is not portable i.e. the programs written for one microcomputer cannot be used for any other because each computer has its own assembly language.
5. It is machine dependent
6. It requires assembler to convert it into machine language.

3. High level language

The disadvantages of assembly languages are overcome using high level languages. Instructions written in high level language are called statements rather mnemonics. In a high level language statements more clearly resemble English and Mathematics than mnemonics. FORTRAN, BASIC, COBOL, C, C++, PASCAL are some of the examples of high level languages.

Programs written in this language for one type of computer can easily be used for any other type of computer. Thus the programs written in high languages are portable. The programs writing in these languages are easier and faster because one statement of

high level language correspond too many instructions of assembly languages. The programs written in high level languages are converted to machine level before executing them. This is done by either compilers or interpreters.

Compiler: A program that translates a high level language program into a machine language program is called a compiler. A compiler is more powerful than assembler.

Interpreter: It is also a translator. It translates a high level language program into object codes, statement wise. It does not translate the entire program at a time. It takes up one statement of a high level language program at a time, translates it and then executes it. It is slower than a compiler.

2.7. Write the differences between machine level and assembly level programming.

S.No.	Machine Language	Assembly Language	High-Level Language
1	Language consists of binary codes which specify the operation.	Language consists of mnemonics which specify the operation.	Language consists of English-like statements which specify more-than one operations.
2	Processor dependent and hence requires knowledge of internal details of processor to write a program.	Processor dependent and hence requires knowledge of internal details of processor to write a program.	Independent of processor.
3	Programs require less memory.	Programs require less memory.	Programs require more memory.
4	Programs have less execution time.	Programs have less execution time.	Programs have more execution time.
5	Program development is difficult.	Program development is similar than machine language.	Program development is easy.
6	It is not user friendly.	It is less user friendly.	It is user friendly.

2.8. Classify the instruction set of 8051.

Different microcontroller architecture use different instruction sets. But all members of the MCS-51 family execute the same instruction set. The MCS-51 instruction set is optimized for 8-bit control applications. The 8051 has 111 instructions: 49 single byte, 45 two byte and 17 three byte.

Based on function the 8051 instruction set is classified into the following 5 groups:

1. Data transfer group	-	28
2. Arithmetic group	-	24
3. Logical group	-	25
4. Branch group	-	17
5. Boolean group	-	17

2.9. Explain the terms operation code and operand and illustrate these terms by writing an instructions

Explain one byte, two byte and three byte instructions of 8051.

- The fig. shows the general format of the instruction. It consists of two parts, they are opcode and operand.

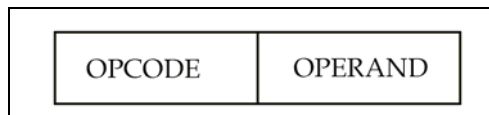


Fig.

- Opcode:** operation code. It is the portion of a instruction, that specifies the operation to be performed.
- Operand:** The second part of the instruction is called the operand, indicating the information needed by the instruction in carrying out its task. Operand consists two fields, such as *Destination and Source*.
- The destination field may be a register or an address or port /port address or a memory location. Destination field specifies the destination for the data that is being copied form the source.
- Based on length the 8051 instruction set is classified into the following 3 groups:
 - One byte instructions - 49
 - Two byte instructions - 45
 - Three byte instructions - 17

a. One byte instructions

A one byte instruction includes the op-code and operand in the same byte. Thus these instructions occupy one memory location as shown in fig. 2.9(a).

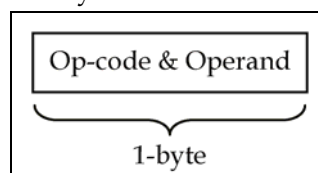


Fig. 2.9(a)

In this format operand may be register/registers and in some cases it may be absented or implicated.

Examples: INC A; ADD A,Rn; NOP.

b. Two Byte Instructions

As the name indicates these instructions occupy two memory locations as shown in fig. 2.9(b).

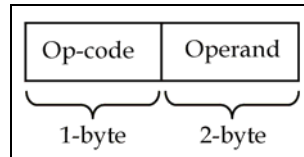


Fig. 2.9(b)

The first byte always indicates the op-code and second byte is either immediate data, direct address, bit address or relative address of the operand. This is illustrated as shown in the below table.

Two byte instruction format		Examples
Op-code	immediate data	ADD A,#data
Op-code	direct address	ADD A,direct
Op-code	bit address	ANL C,bit
Op-code	relative address	DJNZ Rn,rel

c. Three Byte Instructions

This type of instructions occupy three bytes i.e. three memory locations as shown in fig. 2.9(c).

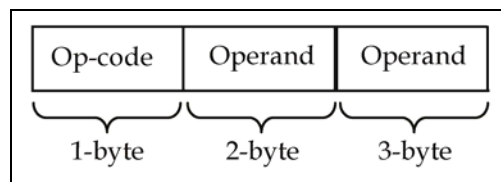


Fig. 2.9(c)

The first byte indicates the op-code. The second and third bytes may indicate address and/or data. This is illustrated as shown in the below table.

Three byte instruction format			Examples
Op-code	immed data15-8	immed data7-0	MOV DPTR,#data16
Op-code	addr15 - addr8	Addr7 - addr0	LCALL addr16

Op-code	dir addr(dest)	dir addr(src)	MOV direct,direct
Op-code	direct address	Immediate data	MOV direct,#data
Op-code	immediate data	Relative address	CJNE A,#data,rel
Op-code	direct address	Relative address	CJNE A,direct,rel
Op-code	bit address	Relative address	JB blt,rel

***Note:**

Except arithmetic group of instructions, remaining group of instructions will have one or two instructions of three byte length.

2.10. List the various addressing modes of 8051 and Explain with examples.

Each instruction requires certain data on which it has to operate. The data could be in register/registers, in memory or an immediate value. The various ways of accessing data are called the addressing modes. The 8051 microcontroller has the following addressing modes.

1. Immediate addressing mode
2. Direct addressing mode
3. Register addressing mode
4. Register indirect addressing mode
5. Indexed addressing mode
6. Register specific addressing mode

1. Immediate Addressing mode

In this addressing mode the source operand is a constant, i.e. immediate data (8-bit or 16-bit).

Examples:

MOV A,#47H	Load 47H in to A.
MOV R4,#67H	Load 67H in to R4.
MOV DPTR,#8500H	DPTR = 5800H.

***Note:**

1. The immediate data must be preceded by the **pround** sign #.
2. Using this addressing mode we can load data into any of the registers, including the DPTR.
3. This is the easiest addressing mode to transfer the data to destination.

4. The general format of immediate addressing mode is

MOV A, #nH : Where 'n' is the 8-bit data

MOV DPTR, #nnH : Where 'nn' is the 16-bit data

MOV R3, #125H	It is illegal instruction because source and destination sizes are mismatched i.e. R3 is a 8-bit register data is 12-bits
MOV 35H, #R2	It is illegal instruction because source should not be register it should be data.
MOV R0, 33H	It is not the example of immediate addressing mode instruction because there is missing of symbol '#'.

2. Direct Addressing mode

In direct addressing the operand is specified by an 8-bit address filed in the instruction.

This mode is used to access data either in the internal RAM (128 bytes) or SFRs.

Examples:

MOV R0,40H	Save content of RAM location 40H in R0.
MOV 56H,A	Save content of A in RAM location 56H.
MOV R4,7FH	Move contents of RAM location 7FH to R4.

*Note:

1. Only internal data RAM and SFRs can be directly addressed. Although the entire 128 bytes of RAM can be accessed using direct addressing mode, it is most often used to access RAM locations 30-7FH.
2. This is due to the fact that register bank locations are accessed by the register names of R0-R7, but there is no such name for other RAM locations 30-7FH.
3. Stack in 8051 uses only direct addressing modes i.e. Only direct addressing mode is allowed for pushing or popping the stack
4. The general format of the Direct Addressing Mode is



PUSH A	It is an invalid instruction because push and pop Instructions must be use direct addressing mode. PUSH 05EH is the valid instruction.
--------	--

3. Register Addressing mode

Register addressing mode involves the use of registers to hold the data to be manipulated.

In this mode the data, which the instruction operates on, A,B, DPTR or is in one of eight registers labeled R0 to R7 (Rn, in general).

Examples:

MOV A,R0	Copy the contents of R0 in to A.
MOV R2,A	Copy the contents of A in to R2.
ADD A,R7	Add the contents of R7 to contents of A.

Do You Know?:

1. The source and destination registers must match in size.
2. The movement of data between Rn registers is not allowed.
3. The general format of Register addressing mode is
4. MOV Rd, Rs where Rd is destination register and Rs is source register
5. MOV DPTR,A; Illegal instruction because the source and destination registers are mismatched in size
6. MOV R1,R2 ; Illegal instruction because, The movement of data between Registers of Register bank is not allowed

4. Register Indirect Addressing mode

In indirect addressing, the instruction specifies a register which contains the address of the operand. Both internal and external RAM can be indirectly addressed.

Examples:

MOV A,@R0	Move contents of RAM location whose Address is held by R0 into A.
MOV @R1,0F0H	Move contents of B (0F0)into RAM location whose address is held by R1.
MOVX @DPTR,A	Writes the contents of the accumulator to the address held by the DPTR register.

***Note:**

1. If the data is inside the CPU, only registers R0 and R1 are used for this purpose. When R0 and R1 are used as pointers, that is, when they hold the address of RAM location, there must be preceded by the @ sign. In the absence of the "@" sign, MOV will be interpreted as an instruction moving the contents of register R0 to A, instead of the contents of the memory location pointed to by R0
2. The external RAM can be addressed indirectly through DPTR.

5. Indexed Addressing mode

Only program memory can be accessed with indexed addressing. Either the DPTR or PC can be used as an index register.

Examples:

MOVC A, @A+DPTR	Copy the code byte found at the ROM address formed by adding A and the DPTR to A.
-----------------	---

MOVC A, @A+PC	Copy the code byte found at the ROM address formed by adding A and the (PC +1).
---------------	---

6. Reregister Specific Addressing mode

In this addressing mode the instructions always operates on implied register such as A or DPTR. Therefore no operands have to be specified.

Examples:

RR A	This instruction operates only on the accumulator. This instruction when executed rotates the contents of accumulator one bit towards right.
DAA	Decimal Adjustment Accumulator.

Additional Information

THE 8051 INSTRUCTION SET

The Inlet 8051 has excellent and most powerful instruction set offers possibilities in control area, serial I/O, arithmetic, byte and bit manipulation. It has 111 instructions.

The most widely used registers of the 8051 are A (address is 0E0, B (0F0), R0, R1, R2, R3, R4, R5, R6, R7. DPTR [DPH (83). DPL (82)]. PC (No address) SP (81).

Based on the operation performed by the instruction, the instruction set is divided into (i) Data move (ii) Arithmetic, (iii) Logic (iv) Call and Jump.

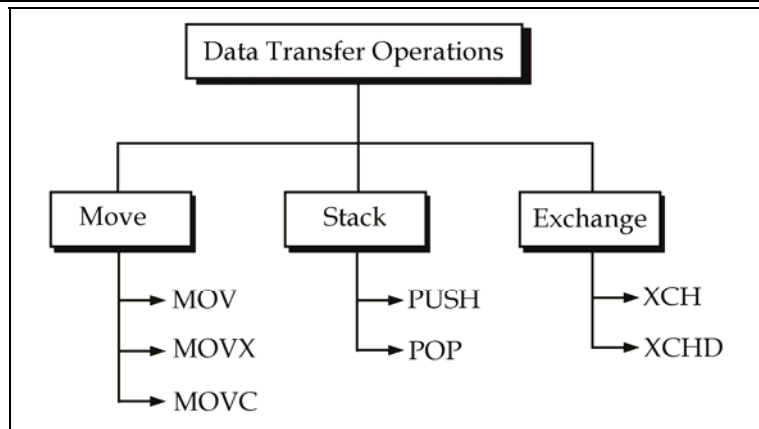
The following symbols and abbreviations are used in the subsequence description of 8051 instructions.

Acronyms

Rn	Register R7-R0 of the currently selected Register Bank.
direct	8-bit internal data location's address. This could be an Internal Data RAM location (0-127) or a SFR [i.e., I/O port, control register, status register, etc., (128-255)].
@Ri	8-bit internal data RAM location (0-255) addressed indirectly through register R1 or R0
#data	8-bit constant included in instruction
# data 16	16-bit constant included in instruction
addr 16	16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K- byte Program Memory address space.
addr 11	11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.
rel	Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is -128 to +127 bytes relative to first byte of the following instruction.

bit	Direct Addressed bit in Internal Data RAM or Special Functional Register.						
CY	The Carry flag						
lsn	Least significant nibble						
msn	Most significant nibble						
[]:	If the condition inside the brackets is true. THEN the action listed will occur, ELSE go to the next instruction						
^	External memory location						
()	Contents of the location inside the parentheses						
Instruction	Flag			Instruction	Flag		
	CY	OV	AC		CY	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	XX	X	CPL C	X		
SUBB	X	X	X	ANL C, bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C, bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C, bit	X		
RLC	X			CJNE	x		
SETB C	1						

2.11. Explain data transfer instructions of 8051.



1. This group of instructions copies data from one location called source to another location called destination.
2. In this operation source content remains unchanged and destination contains the content of source.

3. The data transfer may be between $A \leftrightarrow$ registers, register \leftrightarrow internal memory location, external data (immediate data) to a register or internal memory location and $A \leftrightarrow$ external memory location.
4. During the execution of data transfer instructions “no flags are affected”.

1. MOV A, Rn									
Description	<ul style="list-style-type: none"> • Move the contents of register Rn into accumulator. • Rn presents register from R0 to R7 of currently selected bank. 								
Operation	$(A) \leftarrow (Rn)$								
Example	<p>MOV A, R2</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2" style="text-align: center;">MOV A,R2</th> </tr> <tr> <th style="text-align: center;">Before Execution</th> <th style="text-align: center;">After Execution</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">A = XXH</td> <td style="text-align: center;">A = 37H</td> </tr> <tr> <td style="text-align: center;">R2 = 37H</td> <td style="text-align: center;">R2 = 37H</td> </tr> </tbody> </table> <p>Note: The content of accumulator is changed after execution but contents of R2 remain as it is. This means contents of destination always change after data transfer instruction is executed.</p>	MOV A,R2		Before Execution	After Execution	A = XXH	A = 37H	R2 = 37H	R2 = 37H
MOV A,R2									
Before Execution	After Execution								
A = XXH	A = 37H								
R2 = 37H	R2 = 37H								

2. MOV A, # data							
Description	Move an immediate data into accumulator. Note: <ol style="list-style-type: none"> 1. Immediate data is always presented by # in the instruction 2. The immediate data in this instruction is always an 8-bit. 3. Immediate data can never be a destination. 						
Operation	$(A) \leftarrow \text{data}$						
Example	<p>MOV A,#32H</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">Before Execution</th> <th style="text-align: center;">After Execution</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">A = XXH</td> <td style="text-align: center;">A = 32H</td> </tr> <tr> <td style="text-align: center;">Data = 32H</td> <td></td> </tr> </tbody> </table>	Before Execution	After Execution	A = XXH	A = 32H	Data = 32H	
Before Execution	After Execution						
A = XXH	A = 32H						
Data = 32H							

3. MOV A, direct									
Description	The content of memory location (internal RAM/SFR) whose address is specified directly in the instruction is moved to accumulator.								
Operation	$(A) \leftarrow (\text{direct} = 8\text{-bit address of i-RAM/SFR})$								
Example	<p>MOV A, 40H: The contents of memory location 40 H are moved into accumulator.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2">MOV A,40H</th> </tr> <tr> <th>Before Execution</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td>A = XXH</td> <td>A = 99H</td> </tr> <tr> <td>40H = 99H</td> <td>40H = 99H</td> </tr> </tbody> </table> <p>Note: Parenthesis () means 'contents of'</p>	MOV A,40H		Before Execution	After Execution	A = XXH	A = 99H	40H = 99H	40H = 99H
MOV A,40H									
Before Execution	After Execution								
A = XXH	A = 99H								
40H = 99H	40H = 99H								

4. MOV A, @Ri											
Description	<p>Move the contents of internal RAM memory location whose address is specified by Ri to accumulator.</p> <p>Note: In this indirect addressing mode, the registers used are only R0 and R1.</p>										
Operation	$(A) \leftarrow ((Ri))$										
Example	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2">MOV A,@R0</th> </tr> <tr> <th>Before Execution</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td>A = XXH</td> <td>A = B2H</td> </tr> <tr> <td>R0 = 60H</td> <td>R0 = 60H</td> </tr> <tr> <td>60H = B2H</td> <td>60H = B2H</td> </tr> </tbody> </table>	MOV A,@R0		Before Execution	After Execution	A = XXH	A = B2H	R0 = 60H	R0 = 60H	60H = B2H	60H = B2H
MOV A,@R0											
Before Execution	After Execution										
A = XXH	A = B2H										
R0 = 60H	R0 = 60H										
60H = B2H	60H = B2H										

5. MOV Rn, A	
Description	Move the contents of accumulator into register Rn.
Operation	$(Rn) \leftarrow (A)$
Example	MOV R5, A

6. MOV Rn, #data	
• Description	Move an immediate data into register Rn.
• Operation	$(Rn) \leftarrow \text{data}$
• Example	MOV R2, # 97 H

7. MOV Rn, direct	
Description	The content of memory location (internal RAM/SFR) whose address is directly specified in the instruction is moved to register Rn, where Rn is any one of the 8 register of the currently selected register bank.
Operation	$(Rn) \leftarrow (\text{direct})$
Examples	<ul style="list-style-type: none"> MOV R3, 42: Move the contents of memory location addressed 42 into R3 i.e. $(42) \rightarrow R3$.

8. MOV direct, A									
Description	The content of accumulator is moved to a memory location (internal RAM/SFR) whose address is directly specified in the instruction.								
Operation	$(\text{direct}) \leftarrow (A)$								
Example	<p>MOV 40H, A: The content of accumulator is moved to memory location 40 H.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2">MOV 40H,A</th> </tr> <tr> <th>Before Execution</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td>40H = XXH</td> <td>40H = 17H</td> </tr> <tr> <td>A = 17H</td> <td>A = 17H</td> </tr> </tbody> </table>	MOV 40H,A		Before Execution	After Execution	40H = XXH	40H = 17H	A = 17H	A = 17H
MOV 40H,A									
Before Execution	After Execution								
40H = XXH	40H = 17H								
A = 17H	A = 17H								

9. MOV direct,Rn	
Description	The content of register Rn is moved to memory location (internal RAM/SFR) whose address is directly specified in the instruction.
Operation	$(\text{direct}) \leftarrow (Rn)$
Examples	MOV 55H, R7

10. MOV direct, #data	
Description	An immediate data is moved into internal RAM memory location whose address is specified directly in the instruction. Note: The destination can be SFR also.
Operation	(direct) ← data
Example	MOV 38H, #90H: Move 90 H into RAM location having address 38 H.

11. MOV direct, direct									
Description	The content of one internal RAM/SFR is moved to another internal RAM/SFR. The address of the source and destination are directly specified in the instruction.								
Operation	(direct) ← (data)								
Example	MOV 0A8 H, 78H: Move the contents of RAM location 78H into A8H which is Interrupt Enable (IE) register. <table border="1" style="margin: 10px auto;"> <thead> <tr> <th colspan="2">MOV 0A8H,78H</th> </tr> <tr> <th>Before Execution</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td>A8H = XXH (Address of IE)</td> <td>A8H = 88H</td> </tr> <tr> <td>78H = 88H</td> <td>78H = 88H</td> </tr> </tbody> </table>	MOV 0A8H,78H		Before Execution	After Execution	A8H = XXH (Address of IE)	A8H = 88H	78H = 88H	78H = 88H
MOV 0A8H,78H									
Before Execution	After Execution								
A8H = XXH (Address of IE)	A8H = 88H								
78H = 88H	78H = 88H								

12. MOV direct, @Ri											
Description	The content of internal RAM whose address is specified by Ri is moved to another internal RAM/SFR whose address is directly specified in the instruction. The register Ri can be either R0 or R1.										
Operation	(direct) ← ((Ri))										
Example	<table border="1" style="margin: 10px auto;"> <thead> <tr> <th colspan="2">MOV 74H,@R1</th> </tr> <tr> <th>Before Execution</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td>74H = XXH</td> <td>74H = 85H</td> </tr> <tr> <td>R1 = 47H</td> <td>R1 = 47H</td> </tr> <tr> <td>47H = 85H</td> <td>47H = 85H</td> </tr> </tbody> </table>	MOV 74H,@R1		Before Execution	After Execution	74H = XXH	74H = 85H	R1 = 47H	R1 = 47H	47H = 85H	47H = 85H
MOV 74H,@R1											
Before Execution	After Execution										
74H = XXH	74H = 85H										
R1 = 47H	R1 = 47H										
47H = 85H	47H = 85H										

13. MOV @Ri, A	
Description	The content of accumulator is moved to an internal RAM location whose address is specified by Ri register. The register Ri can be either R0 or R1.
Operation	$((Ri)) \leftarrow (A)$
Example	MOV @ R0,A

14. MOV @Ri, #data	
Description	The immediate data given in the instruction is moved to an internal RAM location whose address is specified by Ri register. The register Ri can be R0 or R1.
Operation	$((Ri)) \leftarrow \text{data}$
Example	MOV @ R0,44H

15. MOV @Ri, direct											
Description	The content of internal RAM/SFR whose address is directly specified in the instruction is moved another internal RAM location whose address is specified by Ri register. The register Ri can be either R0 or R1.										
Operation	$((Ri)) \leftarrow (\text{direct})$										
Example	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2">MOV @R0,3DH</th> </tr> <tr> <th>Before Execution</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td>R0 = 55H</td> <td>R0 = 55H</td> </tr> <tr> <td>55H = XXH</td> <td>55H = 4CH</td> </tr> <tr> <td>3DH = 4CH</td> <td>3DH = 4CH</td> </tr> </tbody> </table>	MOV @R0,3DH		Before Execution	After Execution	R0 = 55H	R0 = 55H	55H = XXH	55H = 4CH	3DH = 4CH	3DH = 4CH
MOV @R0,3DH											
Before Execution	After Execution										
R0 = 55H	R0 = 55H										
55H = XXH	55H = 4CH										
3DH = 4CH	3DH = 4CH										

16. MOV DPTR, #16 bit data	
Description	Move 16 bit immediate data into data pointer register (which is a 16 bit register). Note: This is the instruction which deals with 16 bit data transfer.
Operation	$(DPTR) \leftarrow 16 \text{ bit data}$
Example	MOV DPTR, #8000H

17. MOVX A, @Ri											
Description	<p>Move the contents of external RAM memory location whose address is specified by register Ri into accumulator.</p> <p>Note:</p> <ol style="list-style-type: none"> 1. While accessing external RAM, Ri can address 256 bytes and DPTR can address 64 K bytes. 2. MOVX instruction is used to access external RAM or I/O addresses. 3. There are two sets of RAM addresses between 00H and FFH, one for internal 8051 RAM and another for RAM external to 8051. 										
Operation	$(A) \leftarrow ((Ri))_{Ext}$										
Example	MOVX A, @R1										
18. MOVX A, @DPTR											
Description	<p>Move the contents of the external RAM memory location whose address is specified by data pointer (DPTR) into accumulator.</p> <p>Note: In external moves the data transfer is only between memory and accumulator.</p>										
Operation	$(A) \leftarrow ((DPTR))$										
Example	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2">MOVX A,@DPTR</th> </tr> <tr> <th>Before Execution</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">A = <input type="text" value="XXH"/></td> <td style="text-align: center;">A = <input type="text" value="0FH"/></td> </tr> <tr> <td style="text-align: center;">DPTR = <input type="text" value="8000H"/></td> <td style="text-align: center;">DPTR = <input type="text" value="8000H"/></td> </tr> <tr> <td style="text-align: center;">8000H = <input type="text" value="0FH"/></td> <td style="text-align: center;">8000H = <input type="text" value="0FH"/></td> </tr> </tbody> </table>	MOVX A,@DPTR		Before Execution	After Execution	A = <input type="text" value="XXH"/>	A = <input type="text" value="0FH"/>	DPTR = <input type="text" value="8000H"/>	DPTR = <input type="text" value="8000H"/>	8000H = <input type="text" value="0FH"/>	8000H = <input type="text" value="0FH"/>
MOVX A,@DPTR											
Before Execution	After Execution										
A = <input type="text" value="XXH"/>	A = <input type="text" value="0FH"/>										
DPTR = <input type="text" value="8000H"/>	DPTR = <input type="text" value="8000H"/>										
8000H = <input type="text" value="0FH"/>	8000H = <input type="text" value="0FH"/>										
19. MOVX @ DPTR, A											
Description	Move the contents of accumulator into external RAM memory location whose address is specified by data pointer (DPTR).										
Operation	$((DPTR)) \leftarrow (A)$										
Example	MOVX @ DPTR,A										

20. MOVC A, @A + DPTR											
Description	<p>Move the contents of the external ROM memory location whose address is formed by adding the accumulator contents with the contents of DPTR.</p> <p>Note: This instruction is called as move code byte, the C in MOVC indicates that it's a code byte.</p>										
Operation	$A \leftarrow ((A + DPTR))$										
Example	<p>MOVC A, @ A + DPTR.</p> <p>This instruction copies the code byte found at the external ROM address formed by adding A and the DPTR, i.e. at an address (1200H +61H) 1261 H to A.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2">MOVC A,@A+DPTR</th> </tr> <tr> <th>Before Execution</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td>A = 61H</td> <td>A = 79H</td> </tr> <tr> <td>DPTR = 8000H</td> <td>DPTR = 8000H</td> </tr> <tr> <td>8061H = 79H</td> <td>8061H = 79H</td> </tr> </tbody> </table>	MOVC A,@A+DPTR		Before Execution	After Execution	A = 61H	A = 79H	DPTR = 8000H	DPTR = 8000H	8061H = 79H	8061H = 79H
MOVC A,@A+DPTR											
Before Execution	After Execution										
A = 61H	A = 79H										
DPTR = 8000H	DPTR = 8000H										
8061H = 79H	8061H = 79H										

21. MOV A,@ A+PC	
Description	<p>Move the contents of the external ROM memory location whose address is formed by adding the accumulator contents with the contents of program counter (PC).</p> <p>Note: Destination is always an accumulator.</p>
Operation	$(PC) \leftarrow (PC)+1$ $(A) \leftarrow ((A)+(PC))$
Example	<p>Let the contents of PC are 4000 H and contents of A are 50H. MOVC A, @ A + PC.</p> <p>This instruction copies the code byte found at the external ROM address formed by adding A and the PC, i.e. at an address (4000 H + 50 H) 4050 H to A.</p>

MOVC A,@A+PC	
Before Execution	After Execution
A = <input type="text" value="50H"/>	A = <input type="text" value="39H"/>
PC = <input type="text" value="4000H"/>	PC = <input type="text" value="4000H"/>
4051H = <input type="text" value="39H"/>	4051H = <input type="text" value="39H"/>

22. PUSH direct

Description	<ul style="list-style-type: none"> The stack pointer is incremented by one. PUSH the contents of internal RAM memory location/SFR on to the stack addressed by stack pointer (SP). <p>Note:</p> <ol style="list-style-type: none"> The stack is internal RAM area only. PUSH and POP instructions are used to move the data between internal RAM stack and internal RAM. 										
Operation	$(SP) \leftarrow (SP) + 1$ $((SP)) \leftarrow (\text{direct})$										
Example	<p>PUSH 0F0H</p> <p>This instruction increments the stack pointer by one and stores the contents of register B (0F0H) to the internal RAM location addressed by the stack pointer (SP).</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td colspan="2" style="text-align: center;">PUSH 0F0H</td> </tr> <tr> <td style="text-align: center;">Before Execution</td> <td style="text-align: center;">After Execution</td> </tr> <tr> <td style="text-align: center;">SP = <input type="text" value="07H"/></td> <td style="text-align: center;">SP = <input type="text" value="08H"/></td> </tr> <tr> <td style="text-align: center;">0F0H = <input type="text" value="82H"/></td> <td style="text-align: center;">0F0H = <input type="text" value="82H"/></td> </tr> <tr> <td style="text-align: center;">07H = <input type="text" value="XXH"/></td> <td style="text-align: center;">07H = <input type="text" value="82H"/></td> </tr> </table>	PUSH 0F0H		Before Execution	After Execution	SP = <input type="text" value="07H"/>	SP = <input type="text" value="08H"/>	0F0H = <input type="text" value="82H"/>	0F0H = <input type="text" value="82H"/>	07H = <input type="text" value="XXH"/>	07H = <input type="text" value="82H"/>
PUSH 0F0H											
Before Execution	After Execution										
SP = <input type="text" value="07H"/>	SP = <input type="text" value="08H"/>										
0F0H = <input type="text" value="82H"/>	0F0H = <input type="text" value="82H"/>										
07H = <input type="text" value="XXH"/>	07H = <input type="text" value="82H"/>										

23. POP direct											
Description	<ul style="list-style-type: none"> This instruction moves the contents of stack addressed by stack pointer (SP) to the internal RAM/SFR (Whose address is given within the instruction). Stack pointer is decremented by one. <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2">POP 0E0H</th> </tr> <tr> <th>Before Execution</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td>0E0H = <input type="text" value="XXH"/></td> <td>0E0H = <input type="text" value="XXH"/></td> </tr> <tr> <td>SP = <input type="text" value="08H"/></td> <td>SP = <input type="text" value="07H"/></td> </tr> <tr> <td>08H = <input type="text" value="67H"/></td> <td>08H = <input type="text" value="67H"/></td> </tr> </tbody> </table> <p>Note:</p> <ol style="list-style-type: none"> When stack reaches to FF H it rolls over to 00 H. RAM ends at 7F. So pushing the stack contents above 7F will result in errors. Stack should be normally initialized above the register banks. Only address and not register names are used in the instruction. 	POP 0E0H		Before Execution	After Execution	0E0H = <input type="text" value="XXH"/>	0E0H = <input type="text" value="XXH"/>	SP = <input type="text" value="08H"/>	SP = <input type="text" value="07H"/>	08H = <input type="text" value="67H"/>	08H = <input type="text" value="67H"/>
POP 0E0H											
Before Execution	After Execution										
0E0H = <input type="text" value="XXH"/>	0E0H = <input type="text" value="XXH"/>										
SP = <input type="text" value="08H"/>	SP = <input type="text" value="07H"/>										
08H = <input type="text" value="67H"/>	08H = <input type="text" value="67H"/>										
Operation	$(\text{direct}) \leftarrow ((\text{SP}))$ $(\text{SP}) \leftarrow (\text{SP}) - 1$										
Example	POP 0E0H This instruction copies the contents of the internal RAM location addressed by the stack pointer to the accumulator (0E0H). Then the stack pointer is decremented by one.										

24. XCH A, Rn	
Description	<ul style="list-style-type: none"> XCH stands for exchange. Exchange data bytes between register Rn and A. <p>Note: All exchanges are internal to 8051 and with register A only (accumulator)</p>
Operation	$(A) \leftrightarrow (Rn)$
Example	Initially assume Accumulator = 20H and Register R5=45H. After executing the instruction XCH A, R5 the accumulator =45H and

	Register R5=20H. <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2">XCH A,R5</th> </tr> <tr> <th>Before Execution</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td>A = 20H</td> <td>A = 45H</td> </tr> <tr> <td>R5 = 45H</td> <td>R5 = 20H</td> </tr> </tbody> </table>	XCH A,R5		Before Execution	After Execution	A = 20H	A = 45H	R5 = 45H	R5 = 20H
XCH A,R5									
Before Execution	After Execution								
A = 20H	A = 45H								
R5 = 45H	R5 = 20H								

25. XCH A, direct	
Description	Exchange the contents of direct address (8 bit) and accumulator.
Operation	(A) ↔ (direct)
Example	XCH A, 30 H: The contents of 30 H (address of internal RAM) and accumulator are exchanged.

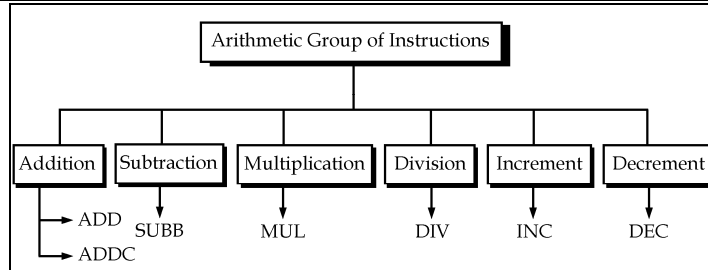
26. XCH A, @Ri	
Description	Exchange the contents of internal RAM location whose address is stored either in Ri and accumulator.
Operation	((Ri)) ↔ (A)
Example	XCH A, @R0

27. XCHD A, @Ri											
Description	Exchange the lower nibble (lower four bits) in accumulator and lower nibble of contents of address in register Ri. Note: The upper nibbles remain un-exchanged.										
Operation	((Ri)) ₀₋₃ ↔ A ₀₋₃										
Example	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2">XCHD A,R0</th> </tr> <tr> <th>Before Execution</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td>R0 = 50H</td> <td>R0 = 50H</td> </tr> <tr> <td>50H = 9CH</td> <td>50H = 92H</td> </tr> <tr> <td>A = 42H</td> <td>A = 4CH</td> </tr> </tbody> </table>	XCHD A,R0		Before Execution	After Execution	R0 = 50H	R0 = 50H	50H = 9CH	50H = 92H	A = 42H	A = 4CH
XCHD A,R0											
Before Execution	After Execution										
R0 = 50H	R0 = 50H										
50H = 9CH	50H = 92H										
A = 42H	A = 4CH										

Summary of Data transfer instructions of 8051

Instructions	Operation	Description
Data Moving Instructions		
1. MOV A,Rn	$(A) \leftarrow (Rn)$	Move register to Accumulator.
2. MOV A,direct	$(A) \leftarrow (\text{direct})$	Move direct byte to Accumulator.
3. MOV A,@Ri	$(A) \leftarrow ((Ri))$	Move indirect RAM to Accumulator.
4. MOV A,#data	$(A) \leftarrow \text{data}$	Move immediate data to Accumulator.
5. MOV Rn,A	$(Rn) \leftarrow (A)$	Move Accumulator to register.
6. MOV direct,@Ri	$(\text{direct}) \leftarrow ((Ri))$	Move indirect RAM to direct byte.
7. MOV DPTR,#data16	$(DPTR) \leftarrow \text{data16}$	Load Data Pointer with a 16-bit constant.
External Data Memory Accessing Instructions		
8. MOVX A,@DPTR	$(A) \leftarrow ((DPTR))$	Move external RAM (16-bit addr) to A.
9. MOVX @DPTR,A	$((DPTR)) \leftarrow (A)$	Move A to external RAM (16-bit addr).
External Program Memory Accessing Instructions		
10. MOVC A,@A+DPTR	$(A) \leftarrow ((A) + (DPTR))$	Move Code byte relative to DPTR to A.
11. MOVC A,@A+PC	$(PC) \leftarrow (PC)+1$ $(A) \leftarrow ((A)+(PC))$	Move Code byte relative to PC to A.
PUSH & POP Instructions		
12. PUSH direct	$(SP) \leftarrow (SP)+1$ $((SP)) \leftarrow (\text{direct})$	Push direct byte onto stack.
13. POP direct	$(\text{direct}) \leftarrow ((SP))$ $(SP) \leftarrow (SP)-1$	Pop direct byte from stack.
Data Exchange Instructions		
14. XCH A,Rn	$(A) \leftrightarrow (Rn)$	Exchange register with Accumulator.
15. XCHD A,@Ri	$((Ri_{0-3}) \leftrightarrow (A_{0-3}))$	Exchange low-order digit indirect RAM with A.

2.12. Explain the arithmetic instructions and recognize the flags that are set or reset for given data conditions.



1. This group of instructions performs arithmetic operations such as addition, subtraction, multiplication, division, increment, decrement and decimal adjustment operations on the 8-bit data present in registers or in memory locations.
2. All the additions are done with the A register as the destination of the result.
3. Add, subtract, increment and decrement instructions use one of the four addressing modes each: register, immediate, direct, and indirect.
4. There are two sets of addition instructions: add without carry and add with carry. There is only one set of subtract with borrow, subtract without borrow can be done in two steps-CLR C and then SBBB.
5. The 8051 has four arithmetic flags: the Carry (CY), Auxiliary Carry (AC), Overflow (OV) and Parity (P). The CY, AC and OV flags are set to 1 or cleared to 0 automatically, depending on the outcomes of the certain instructions.
6. The parity flag is affected by every instruction executed.
 - a) Parity flag will be set to 1 for odd number of 1's in A register and will be set to 0 for even number of 1's in A register.
 - b) All 0's in A register is considered as even number of 1's and set P flag to 0.

1. ADD A, Rn															
Description	The content of register Rn is added with contents of A and result is stored in register A. Note: Flags are affected.														
Operation	$(A) \leftarrow (A) + (Rn)$														
Example	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2" style="text-align: center;">ADD A, R3</th> </tr> <tr> <th style="width: 50%;">Before Execution</th> <th style="width: 50%;">After Execution</th> </tr> </thead> <tbody> <tr> <td>A = FEH → 1111 1110</td> <td>A = 73H</td> </tr> <tr> <td>R3 = 75H → 0111 0101</td> <td>R3 = 75H</td> </tr> <tr> <td style="text-align: center;"> $\begin{array}{r} 11111 \\ 01110101 \\ \hline 101110011 \end{array}$ </td> <td>CY = 1</td> </tr> <tr> <td style="text-align: center;"> $\begin{array}{r} \text{Carry} \rightarrow \textcircled{1}01110011 \\ \hline \underbrace{\hspace{1.5cm}}_7 \quad \underbrace{\hspace{1.5cm}}_3 \end{array}$ </td> <td>AC = 1</td> </tr> <tr> <td></td> <td>P = 1</td> </tr> </tbody> </table>	ADD A, R3		Before Execution	After Execution	A = FEH → 1111 1110	A = 73H	R3 = 75H → 0111 0101	R3 = 75H	$ \begin{array}{r} 11111 \\ 01110101 \\ \hline 101110011 \end{array} $	CY = 1	$ \begin{array}{r} \text{Carry} \rightarrow \textcircled{1}01110011 \\ \hline \underbrace{\hspace{1.5cm}}_7 \quad \underbrace{\hspace{1.5cm}}_3 \end{array} $	AC = 1		P = 1
ADD A, R3															
Before Execution	After Execution														
A = FEH → 1111 1110	A = 73H														
R3 = 75H → 0111 0101	R3 = 75H														
$ \begin{array}{r} 11111 \\ 01110101 \\ \hline 101110011 \end{array} $	CY = 1														
$ \begin{array}{r} \text{Carry} \rightarrow \textcircled{1}01110011 \\ \hline \underbrace{\hspace{1.5cm}}_7 \quad \underbrace{\hspace{1.5cm}}_3 \end{array} $	AC = 1														
	P = 1														

2. ADD A, #data							
Description	An immediate 8 bit number is added with contents of accumulator and result is stored in accumulator.						
Operation	$(A) \leftarrow (A) + \text{data}$						
Example	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2" style="text-align: center;">ADD A,#42H</th> </tr> <tr> <th style="text-align: center;">Before Execution</th> <th style="text-align: center;">After Execution</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">A = 25H Data = 42H</td> <td style="text-align: center;">A = 25+42 = 67H CY = 0, AC = 0, P = 1</td> </tr> </tbody> </table>	ADD A,#42H		Before Execution	After Execution	A = 25H Data = 42H	A = 25+42 = 67H CY = 0, AC = 0, P = 1
ADD A,#42H							
Before Execution	After Execution						
A = 25H Data = 42H	A = 25+42 = 67H CY = 0, AC = 0, P = 1						

3. ADD A, direct							
Description	The contents of internal RAM location specified is added with accumulator and result is stored in accumulator						
Operation	$(A) \leftarrow (A) + (\text{direct})$						
Example	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2" style="text-align: center;">ADD A,41H</th> </tr> <tr> <th style="text-align: center;">Before Execution</th> <th style="text-align: center;">After Execution</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">A = 23H Location 41H = 32H</td> <td style="text-align: center;">A = 23+32 = 55H CY = 0, AC = 0, P = 0</td> </tr> </tbody> </table>	ADD A,41H		Before Execution	After Execution	A = 23H Location 41H = 32H	A = 23+32 = 55H CY = 0, AC = 0, P = 0
ADD A,41H							
Before Execution	After Execution						
A = 23H Location 41H = 32H	A = 23+32 = 55H CY = 0, AC = 0, P = 0						

4. ADD A, @Ri							
Description	The content of memory location whose address is in register Ri is added with contents of register A and result is stored into accumulator.						
Operation	$(A) \leftarrow (A) + ((Ri))$						
Example	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2" style="text-align: center;">ADD A,@R1</th> </tr> <tr> <th style="text-align: center;">Before Execution</th> <th style="text-align: center;">After Execution</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">A = 33H R1 = 45H</td> <td style="text-align: center;">A = 33+45 = 78H CY = 0, AC = 0, P = 0</td> </tr> </tbody> </table>	ADD A,@R1		Before Execution	After Execution	A = 33H R1 = 45H	A = 33+45 = 78H CY = 0, AC = 0, P = 0
ADD A,@R1							
Before Execution	After Execution						
A = 33H R1 = 45H	A = 33+45 = 78H CY = 0, AC = 0, P = 0						

8. ADDC A, @Ri	
Description	Add accumulator, the contents of the address stored in register Ri, and carry and sum is stored in accumulator.
Operation	$(A) \leftarrow (A) + (CY) + ((Ri))$
Example	ADDC A, @R0 A = 00110011; CY = 1; R0 = 45H; @R0 = 01000000 After executing ADDC A; @ R0 instruction A = 01110100; CY = 0

9. SUBB A, Rn							
Description	The contents of register Rn and a carry (borrow) flag are subtracted from accumulator and result is stored in accumulator.						
Operation	$(A) \leftarrow (A) - (CY) - (Rn)$						
Example	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2" style="text-align: center;">SUBB A, R5</th> </tr> <tr> <th style="text-align: center;">Before Execution</th> <th style="text-align: center;">After Execution</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;"> A = 85H → 1 0 0 0 0 1 0 1 R5 = 32H → 0 0 1 1 0 0 1 0 $\begin{array}{r} 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1 \\ -\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \\ \hline 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \\ \hline \end{array}$ <div style="display: flex; justify-content: center; gap: 20px;"> $\underbrace{\hspace{2em}}_5$ $\underbrace{\hspace{2em}}_3$ </div> </td> <td style="text-align: center;"> A = 53H CY = 0 </td> </tr> </tbody> </table>	SUBB A, R5		Before Execution	After Execution	A = 85H → 1 0 0 0 0 1 0 1 R5 = 32H → 0 0 1 1 0 0 1 0 $\begin{array}{r} 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1 \\ -\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \\ \hline 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \\ \hline \end{array}$ <div style="display: flex; justify-content: center; gap: 20px;"> $\underbrace{\hspace{2em}}_5$ $\underbrace{\hspace{2em}}_3$ </div>	A = 53H CY = 0
SUBB A, R5							
Before Execution	After Execution						
A = 85H → 1 0 0 0 0 1 0 1 R5 = 32H → 0 0 1 1 0 0 1 0 $\begin{array}{r} 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1 \\ -\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \\ \hline 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \\ \hline \end{array}$ <div style="display: flex; justify-content: center; gap: 20px;"> $\underbrace{\hspace{2em}}_5$ $\underbrace{\hspace{2em}}_3$ </div>	A = 53H CY = 0						

10. SUBB A, #data	
Description	An immediate number is subtracted from accumulator with carry (borrow) flag.
Operation	$(A) \leftarrow (A) - (CY) - \text{data}$
Example	SUBB A, #32H Initially, A register = 85H and CY = 0. After executing SUBB A, # 32H the accumulator will hold the result 53H, in it and the CY flag is made zero. $\begin{array}{r} A = 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1 \\ CY = \hspace{10em} 0 \\ \hline 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1 \\ 32H = 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \\ \hline = 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1 = 53H \end{array}$

11. SUBB A, direct	
Description	The contents of RAM location and a carry (borrow) flag are subtracted from accumulator and result is stored in accumulator.
Operation	$(A) \leftarrow (A) - (CY) - (\text{direct})$
Example	<p>SUBB A, 32H</p> <p>Let us assume that, A register = 86H and internal memory location 32H contains 30H and CY = 1. After executing SUBB A, 32H instruction the accumulator is loaded with 55H and the memory location content remains same.</p> $ \begin{array}{r} A = 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0 \\ CY = \quad \quad \quad \quad \quad 1 \\ \hline \quad \quad \quad 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1 \\ (32H) = 30H = \quad \quad 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0 \\ \hline \quad \quad \quad \underline{0\ 1\ 0\ 1\ 0\ 1\ 0\ 1} = 55H \end{array} $

12. SUBB A, @Ri	
Description	The contents of address which is stored in Ri and a carry (borrow) flag are subtracted from accumulator and result is stored in accumulator.
Operation	$(A) \leftarrow (A) - (CY) - ((Ri))$
Example	<p>SUBB A, @R0</p> <p>Assume that, A register = 74H, CY = 1, R0 register = 23H, Internal Memory (23H) = 32H. After executing SUBB A, @R0 the accumulator contains 41H, the internal memory content remains same and CY flag is cleared.</p> $ \begin{array}{r} A = 74H = 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0 \\ CY = \quad \quad \quad \quad \quad 1 \\ \hline \quad \quad \quad 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1 \\ (R0) = (23) = 32 = \quad \quad 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \\ \hline A = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1 = 41H \end{array} $

INC A	
Description	The content of accumulator is incremented by 1.
Operation	$(A) \leftarrow (A) + 1$
Example: 1	Assume initially A = 40H. After executing INC A instruction, the accumulator will contain 41H.

Example: 2	INC A	
	Before Execution	After Execution
	A = 7AH	A = 7BH

INC Rn	
Description	The content of specified register is incremented by 1.
Operation	$(Rn) \leftarrow (Rn) + 1$
Example	Assume initially R5 = 22H. After executing INC R5, the register R5 will have 23H.

INC direct	
Description	The content of RAM/SFR (whose address is directly given in the instruction) is incremented by 1.
Operation	$(\text{direct}) \leftarrow (\text{direct}) + 1$
Example	Assume initially the internal memory location 40H contains data 50H. After executing INC 40H, the location will have the updated value 51H.

INC @Ri	
Description	The content of memory location whose address is stored in Ri is incremented by 1.
Operation	$((Ri)) \leftarrow ((Ri)) + 1$
Example	Assume the pointer R0 contains 23H and the internal memory location 23H contains 40H. After executing INC@R0, the internal memory location 23H will have the updated value 41H.

INC DPTR	
Description	The content of 16-bit DPTR (Data Pointer) is incremented by 1.
Operation	$(DPTR) \leftarrow (DPTR) + 1$
Example: 1	Registers DPH and DPL contain 14H and FFH respectively. The execution of INC DPTR instruction will change DPH and DPL to 15H and 00H respectively.

Example: 2	INC DPTR	
	Before Execution	After Execution
	DPTR = 8500H	DPTR = 8501H

DCR A	
Description	The content of accumulator is decremented by 1.
Operation	$(A) \leftarrow (A) - 1$
Example	Let us assume $A = 42H$. After executing the instruction DEC A, the accumulator will contain 41H.

DCR Rn	
Description	The content of specified register is decremented by 1.
Operation	$(Rn) \leftarrow (Rn) - 1$
Example	Let us assume R5 contains 45H. After executing the instruction DEC R5, the register R5 will contain 44H.

DCR direct	
Description	The content of RAM/SFR (whose address is directly given in the instruction) is decremented by 1.
Operation	$(\text{direct}) \leftarrow (\text{direct}) - 1$
Example	Assume initially the internal memory location 41H contains data 51H. After executing DEC 41H, will have the location will have the updated value 50H.

DCR @Ri	
Description	The content of memory location whose address is stored in Ri is decremented by 1.
Operation	$((Ri)) \leftarrow ((Ri)) - 1$
Example	Assume the pointer R1 contains 24H and the internal memory location 24H contains 56H. After execution of DEC@R1, the internal memory location 24H will have the updated value 55H.

MUL AB	
Description	Multiply two unsigned numbers stored in accumulator and register B. Register B is for such purposes only. The lower byte of the result is stored in accumulator and higher byte in register B.
Operation	Low order byte in $A \leftarrow (A) \times (B)$ High order byte in B
Example 1:	MOV A, # 5d MOV B, # 7d MUL AB ; A = 35d = 23H, B= 0 0
Example 2:	MOV A, # 10d MOV B, #15d MUL AB ; A = 150d = 96H, B = 0 0 Note: This instruction always clears the CY flag; however, OV is changed according to the product. If the product is greater than FFH, OV = 1; otherwise, it is cleared (OV = 0).
Example 3:	MOV A, #25H MOV B, # 78H MUL AB ; A = 58H B=11H, CY = 0 and OV = 1 ; (25H x 78H = 1158H)
Example 4:	MOV A, #100d MOV B, #200d MUL AB ; A = 20H B=4EH, OV = 1 and CY = 0 ; (100 x 200 = 20,000 = 4E20H) Note: <ol style="list-style-type: none"> 1. When A = FFH; B = FFH. The largest possible product is obtained i.e., FE01H. 2. Register A contains 01H and register B contains FEH after multiplication of FFH by FFH. 3. The OV flag is set to 1 that register B contains the high-order-byte of the product, the Carry flag is 0. 4. There is no comma between A and B in the MUL mnemonic.

MUL AB	
Before Execution	After Execution
<p>A = FEH</p> <p>B = 02H</p>	<p>A = FCH</p> <p>B = 01H</p> <p>OV = 1</p> <p>B = 0</p>
<p>Multiplication Operation :</p> <p style="text-align: center;">FEH x 02H = 01FCH or 254D x 02D = 508D = 01FCH</p> <p>Since the result is more than FFH, OV flag is set to 1.</p>	

DIV AB	
Description	<p>The unsigned number in accumulator is divided by the unsigned number in register B. The integer part of the quotient is in register A and integer part of remainder is in B.</p> <p>Note:</p> <ol style="list-style-type: none"> 1. The Carry flag (CY) is always cleared. 2. The overflow flag (OV) is set if division by 0 was attempted, otherwise it is cleared.
Operation	<p>$AB \leftarrow (A)/(B)$ $(A) \leftarrow \text{Quotient}$ $(B) \leftarrow \text{Remainder}$</p>
Example 1:	<p>MOV Ax 35d MOV B, #10d DIV AB ; A = 3d and B = 5d</p>
Example 2:	<p>MOV A, #97H MOV B, #12H DIV AB ; A = 8d and B = 7d</p> <p>Notice in this instruction that the carry and OV flags are both cleared, unless we divide A by 0, in which case the result is invalid and OV = 1 to indicate the invalid condition.</p> <p>A = 45H ; B = 00</p>

Example 3:	<p>DIV AB</p> <p>After executing the instruction the value in register A and B are undefined and the OV flag is set to high to indicate an invalid result. Notice that CY is always 0 in this instruction.</p> <p>Note:</p> <ol style="list-style-type: none"> 1. The original contents of A and B are lost 2. There is no comma between A and B in the DIV mnemonic <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2" style="text-align: center;">DIV AB</th> </tr> <tr> <th style="text-align: center;">Before Execution</th> <th style="text-align: center;">After Execution</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">A = 45H</td> <td style="text-align: center;">A = 03H (Quotient)</td> </tr> <tr> <td style="text-align: center;">B = 14H</td> <td style="text-align: center;">B = 09H (Remainder)</td> </tr> </tbody> </table> <p>Division Operation :</p> $45H \div 14H \longrightarrow \left\{ \begin{array}{l} Q = 03H \\ R = 09H \end{array} \right.$ $69D \div 20D \longrightarrow \left\{ \begin{array}{l} Q = 03H \\ R = 09H \end{array} \right.$	DIV AB		Before Execution	After Execution	A = 45H	A = 03H (Quotient)	B = 14H	B = 09H (Remainder)
DIV AB									
Before Execution	After Execution								
A = 45H	A = 03H (Quotient)								
B = 14H	B = 09H (Remainder)								

DAA	
Description	<p>Decimal Adjust Accumulator after addition. When we add two numbers the result is in binary, to adjust the result in BCD (Decimal) DAA is used followed by ADDI or ADDC. As we know that after adding, if the lower four bits is greater than 09 then 06 is added, if upper four bits greater than 09 then 60 is added to adjust the result in BCD</p> <p>(Take example of 8085 DAA instruction) CY, AC and OV flag area affected.</p>
Operation	<p>If $(A_{3-0}) > 9$ or $(AC) = 1$ then $(A_{3-0}) \leftarrow (A_{3-0}) + 06$</p> <p>If $(A_{7-4}) > 9$ or $(CY) = 1$ then $(A_{7-4}) \leftarrow (A_{7-4}) + 06$</p>
Example 1:	<p>Example (i): If A = 23; B = 14. After addition the result is 37. In this case BCD addition and hexadecimal addition results are same.</p>
Example 2:	<p>IF A = 23; B = 48. After addition the result is 6B. This is the hexadecimal result where as BCD result is 7.</p> <p>In order to obtain BCD result there is an instruction available labeled as DAA instruction. A DAAA instruction is useful in decimal arithmetic. The performance of DAA instructions is given below.</p>

	<p>Execution Process :</p> <p>A = 29H → 0010 1001</p> <p>Data = 18H → 0001 1000</p> $\begin{array}{r} 111 \\ \underline{0100\ 0001} \\ 0100\ 0001 \\ \underbrace{}_4 \quad \underbrace{}_1 \end{array}$ <p>Since AC = 1, after addition, DAA will add 6 to lower 4 bits. The final result is in BCD format.</p> <p>41H → 0100 0001</p> <p>06H → 0 0110</p> $\begin{array}{r} 0110 \\ \underline{0100\ 0111} \\ 0100\ 0111 \\ \underbrace{}_4 \quad \underbrace{}_7 \end{array}$	
--	--	--

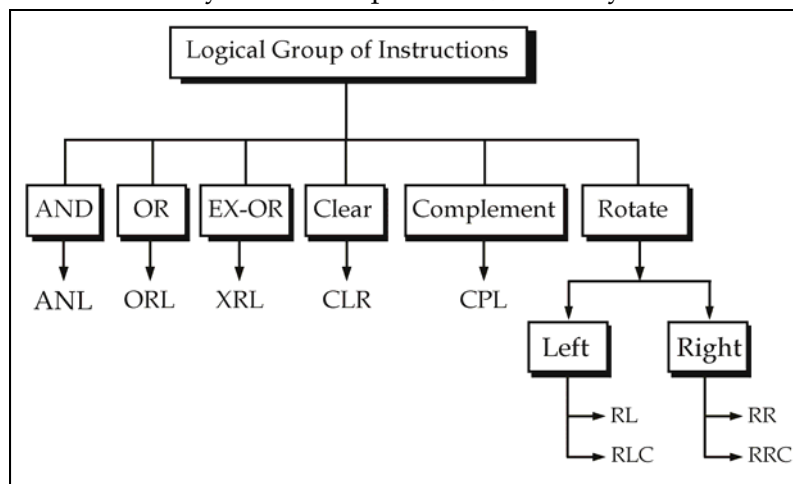
Summary of Arithmetic instructions of 8051

Instructions	Operation	Description
Addition Instructions		
1. ADD A,Rn	$(A) \leftarrow (A) + (Rn)$	Add register to Accumulator.
2. ADD A,#data	$(A) \leftarrow (A) + \text{data}$	Add immediate data to Accumulator.
3. ADDC A,Rn	$(A) \leftarrow (A) + (CY) + (Rn)$	Add register to Accumulator with carry.
Subtraction Instructions		
4. SUBB A,Rn	$(A) \leftarrow (A) - (CY) - (Rn)$	Subtract Register from ACC with borrow.
5. SUBB A,#data	$(A) \leftarrow (A) - (CY) - \text{data}$	Subtract immediate data from ACC with borrow.
Incrementing & Decrementing Instructions		
6. INC A	$(A) \leftarrow (A) + 1$	Increment Accumulator.
7. DEC A	$(A) \leftarrow (A) - 1$	Decrement Accumulator.
8. INC DPTR	$(DPTR) \leftarrow (DPTR) + 1$	Increment Data Pointer.
Multiplication & Division Instructions		
9. MUL AB	Low order byte in $A \leftarrow (A) \times (B)$ High order byte in B	Multiply A and B.

10. DIV AB	Quotient in $A \leftarrow (A) \div (B)$ Reminder in B	Divide A by B.
Decimal Arithmetic Operation Instruction		
11. DA A	If $(A_{3-0}) > 9$ or $(AC) = 1$ then $(A_{3-0}) \leftarrow (A_{3-0}) + 06$ If $(A_{7-4}) > 9$ or $(CY) = 1$ then $(A_{7-4}) \leftarrow (A_{7-4}) + 06$	Decimal Adjust Accumulator.

2.13. Explain the logical instructions and recognize the flags that are set or reset for given data conditions.

1. This group of instructions perform various logical operations AND, OR, EX-OR, rotate, clear and complement.
2. For the instructions RLC, RRC, and CPL only carry flag is affected.
3. For the instruction CLR the carry flag is set to 0.
4. These instructions use one of the four addressing modes each: register, immediate, direct and indirect.
5. The A register or a direct address in internal RAM is the destination of the logical operation result.
6. Keep in mind that all such operations are done using each individual bit of the destination and source bytes. These operations, called byte-level Boolean operations.



ANL A, Rn	
Description	The content of register Rn is ANDed with accumulator and result is stored in accumulator.
Operation	$(A) \leftarrow (A) \text{ AND } (Rn)$

Example: 1	A = 00100010; R0 = 00100010 ANL A, R0 After execution $A \leftarrow 00100000$												
Example: 2	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="3">ANL A, R3</th> </tr> <tr> <th>Before Execution</th> <th>AND Operation</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td>A = 15H</td> <td>15H \rightarrow 0 0 0 1 0 1 0 1</td> <td>A = 00</td> </tr> <tr> <td>R3 = E2H</td> <td>E2H \rightarrow <u>1 1 1 0 0 0 1 0</u> <u>0 0 0 0 0 0 0 0</u></td> <td>R3 = E2H</td> </tr> </tbody> </table>	ANL A, R3			Before Execution	AND Operation	After Execution	A = 15H	15H \rightarrow 0 0 0 1 0 1 0 1	A = 00	R3 = E2H	E2H \rightarrow <u>1 1 1 0 0 0 1 0</u> <u>0 0 0 0 0 0 0 0</u>	R3 = E2H
ANL A, R3													
Before Execution	AND Operation	After Execution											
A = 15H	15H \rightarrow 0 0 0 1 0 1 0 1	A = 00											
R3 = E2H	E2H \rightarrow <u>1 1 1 0 0 0 1 0</u> <u>0 0 0 0 0 0 0 0</u>	R3 = E2H											

ANL A, direct

Description	The content of accumulator is ANDed with the content of RAM location whose address is specified in the instruction. The result is stored in accumulator.
Operation	$(A) \leftarrow (A) \text{ AND (direct)}$
Example: 1	A = 00100010 & Location 41 = 00110100 ANL A, 41H After execution $A \leftarrow 00100000$ The content of location 41H remains same.

Example: 2

ANL A, 76H		
Before Execution	AND Operation	After Execution
A = 27H	27H \rightarrow 0 0 1 0 0 1 1 1	A = 04H
76H = 4CH (ML=Memory Location)	4CH \rightarrow <u>0 1 0 0 1 1 0 0</u> <u>0 0 0 0 0 1 0 0</u>	76H = 4CH (ML=Memory Location)

ANL A, @ Ri

Description	The content of accumulator is ANDed with the content of memory location whose address is stored in either R0 or R1 register. The result is stored in accumulator.
Operation	$(A) \leftarrow (A) \text{ AND } ((Ri))$
Example	A = 00110010

	<p>Content of R0 = 45 Content of RAM address at 45 = 00010010 ANL A, @ R0 After execution, A = 00010010 The content of memory location pointed by R0 remains same.</p>
--	---

ANL A, # data

Description	An immediate data is ANDed with contents of accumulator and result is stored in accumulator.
Operation	$(A) \leftarrow (A) \text{ AND data}$
Example	<p>A = 32H = 00110010 Data = 31H = 00110001 ANL A, #31H After execution of this instruction, $A \leftarrow 00110000 \leftarrow 30H$</p>

ANL direct, A

Description	The content of accumulator is ANDed with direct memory location contents and result is stored in direct memory location.
Operation	$(\text{direct}) \leftarrow (\text{direct}) \text{ AND } (A)$
Example	<p>Acc = 01000001 = 41H; Location 23 = 00110010 = 32H ANL 23H, A After execution of this instruction, the location 23H contains zero. In this instruction, the specified internal memory location 23H is the destination.</p>

ANL direct, # data

Description	An immediate data is ANDed with direct memory location contents and result is stored in direct memory location.
Operation	$(\text{direct}) \leftarrow (\text{direct}) \text{ AND data}$
Example	<p>Location 51H = 01000001 = 41H; data = 01000101 = 45 ANL 51H, #45 After execution of this instruction, the location 51H contains the result 41H.</p>

ORL A, Rn	
Description	The content of register Rn is ORed with accumulator and result is stored in accumulator.
Operation	$(A) \leftarrow (A) \text{ OR } (Rn)$
Example	Acc = 01011010 R1 = 10110000 ORLA, R1 After execution of this instruction Acc = 1111010 = FA.
ORL A, direct	
Description	The content of accumulator is ORed with the content of RAM location whose address is specified in the instruction. The result is stored in accumulator.
Operation	$(A) \leftarrow (A) \text{ OR } (\text{direct})$
Example	Acc = 01011100 = 5C Direct = 43 Content of 43 = 00001100 = 0C ORLA, 43 After execution of this instruction A becomes 01011100 = 5C.
ORL A, @ Ri	
Description	The content of accumulator is ORed with the content of memory location whose address is stored in either R0 or R1 register. The result is stored in accumulator.
Operation	$(A) \leftarrow (A) \text{ OR } ((Ri))$
Example: 1	Acc = 10101010; R0 = 43 Content of RAM address 43 = 00001000 ORLA, @ R0 After execution of this instruction, A=10101010 = AA
Example: 2	

ORL A, @R0		
Before Execution	OR Operation	After Execution
A = 04H	04H → 0000 0100	A = 04H
R0 = 74H	7AH → 0111 1010	R0 = 74H
74H = 7AH	<u>0111 1110</u>	74H = 7AH

ORL A, # data																
Description	An immediate data is ORed with contents of accumulator and result is stored in accumulator.															
Operation	(A) ← (A) OR data															
Example: 1	Acc = 01001100 = 4C Data n = 01000011 = 43 ORL A, # 43 After execution of this instruction A becomes 01001111 = 4F															
Example: 2																
<table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="3">ORL A, #1BH</th> </tr> <tr> <th style="width: 33%;">Before Execution</th> <th style="width: 33%;">OR Operation</th> <th style="width: 33%;">After Execution</th> </tr> </thead> <tbody> <tr> <td>A = 45H</td> <td>45H → 0100 0101</td> <td>A = 5FH</td> </tr> <tr> <td>Data = 1BH</td> <td>1BH → 0001 1011</td> <td></td> </tr> <tr> <td></td> <td><u>0101 1111</u></td> <td></td> </tr> </tbody> </table>		ORL A, #1BH			Before Execution	OR Operation	After Execution	A = 45H	45H → 0100 0101	A = 5FH	Data = 1BH	1BH → 0001 1011			<u>0101 1111</u>	
ORL A, #1BH																
Before Execution	OR Operation	After Execution														
A = 45H	45H → 0100 0101	A = 5FH														
Data = 1BH	1BH → 0001 1011															
	<u>0101 1111</u>															

ORL direct, A	
Description	The content of accumulator is ORed with direct memory location contents and result is stored in direct memory location.
Operation	(direct) ← (direct) OR (A)
Example	A = 67H = 0110 0111 RAM address F0 content = 44H = 0100 0100 ORL F0, A After executing this instruction F0 = 01100111 = 67H

ORL direct, # data	
Description	An immediate data is ORed with direct memory location contents and result is stored in direct memory location.
Operation	(direct) ← (direct) OR data
Example	Assuming that RAM location 32H has the value 67H, ORL32H, #29 After execution: RAM location 32H: 6F

XRL A, Rn	
Description	The content of register Rn is XORed with accumulator and result is stored in accumulator.
Operation	(A) ← (A) XOR (Rn)
Example	XRL A, R5

XRL A, direct	
Description	The content of accumulator is XORed with the content of RAM location whose address is specified in the instruction. The result is stored in accumulator.
Operation	(A) ← (A) XOR (direct)
Example	XRL A, 75H

XRL A, @Ri	
Description	The content of accumulator is XORed with the content of memory location whose address is stored in either R0 or R1 register. The result is stored in accumulator.
Operation	(A) ← (A) XOR ((Ri))
Example	XRL A, @R0

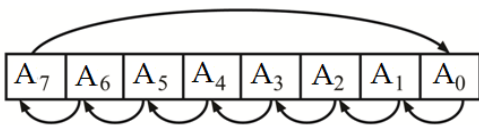
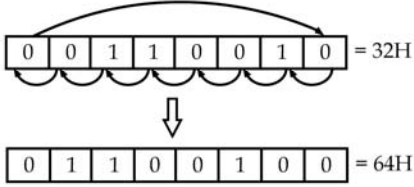
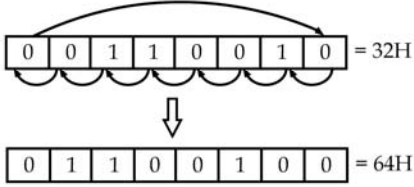
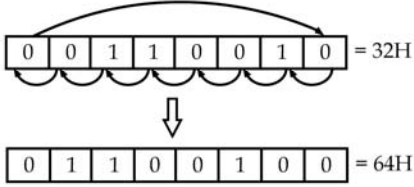
XRL A, # data	
Description	An immediate data is XORed with contents of accumulator and result is stored in accumulator.
Operation	(A) ← (A) XOR data
Example	XRL A, 0FH

XRL direct, A		
Description	The content of accumulator is XORed with direct memory location contents and result is stored in direct memory location.	
Operation	(direct) ← (direct) XOR (A)	
Example		
XRL 72H, A		
Before Execution	X-OR Operation	After Execution
A = 2AH 67H = 54H	2AH → 0010 1010 54H → 0101 0100 <hr style="width: 50%; margin: 0 auto;"/> 0111 1110	A = 7EH 67H = 54H

XRL direct, # data		
Description	An immediate data is XORed with direct memory location contents and result is stored in direct memory location.	
Operation	(direct) ← (direct) XOR data	
Example:		
XRL 0E0, #29H		
Before Execution	X-OR Operation	After Execution
0E0H = 48H Data = 29H	48H → 0100 1000 29H → 0010 1001 <hr style="width: 50%; margin: 0 auto;"/> 0110 0001	0E0H = 61H

CLR A	
Description	<ul style="list-style-type: none"> • Clear accumulator. • All eight bits of accumulator are cleared to zero.
Operation	(A) ← 0
Example	Suppose A = 52 H, then after CLR A, A = 00 H

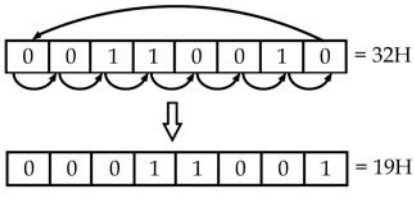
CPL A	
Description	<ul style="list-style-type: none"> Complement accumulator. The contents of accumulator are complemented i.e. 0 to 1 and 1 to 0. Here all eight bits are complemented.
Operation	$(A) \leftarrow (\bar{A})$
Example	Suppose A = 55 H then after CPL, A = AA

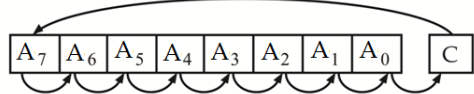
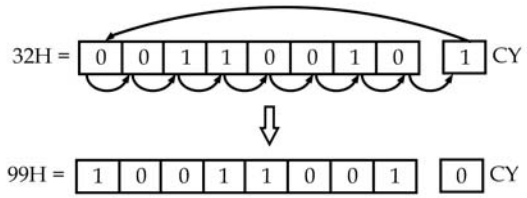
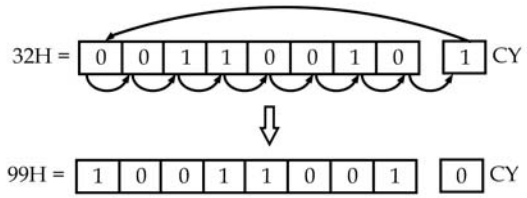
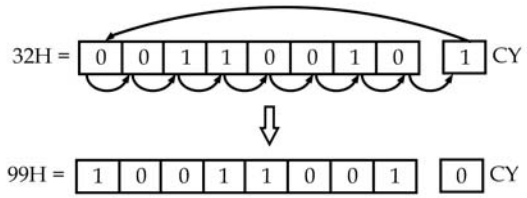
RL A									
Description	<ul style="list-style-type: none"> Rotation accumulator left. Rotate one bit in A (i.e. accumulator) left. This means A7 comes in A0, A0 in A1, A1 in A2 and so on. 								
Operation	$(A_{n+1}) \leftarrow (A_n)_{n=0-6} (A_0) \leftarrow (A_7)$								
Example: 1	Suppose A = 1C = 0001 1100 After RL A = A = 0011 1000 = 38 H								
Example: 2	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2" style="text-align: center;">RL A</th> </tr> <tr> <th style="width: 50%;">Before Execution</th> <th style="width: 50%;">After Execution</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">A = 32H</td> <td style="text-align: center;">A = 64H</td> </tr> <tr> <td colspan="2" style="text-align: center;"> Rotate Operation :  </td> </tr> </tbody> </table>	RL A		Before Execution	After Execution	A = 32H	A = 64H	Rotate Operation : 	
RL A									
Before Execution	After Execution								
A = 32H	A = 64H								
Rotate Operation : 									

RLC A	
Description	<ul style="list-style-type: none"> Rotate accumulator left through the carry flag. Rotate accumulator left by one bit through carry. This means A7 goes into carry, carry into A0, A0 to A1, A1 to A2 and so on.

Operation	$(A_{n+1}) \leftarrow (A_n)_{n=0-6} (A_0) \leftarrow (C) (C) \leftarrow (A_7)$								
Example: 1	Suppose A = 3A = 0011 1010 and CY = 1 Then, After RLC A 0111 0101 and CY = 0								
Example: 2	<table border="1" style="width: 100%; text-align: center;"> <thead> <tr> <th colspan="2">RLC A</th> </tr> <tr> <th>Before Execution</th> <th>After Execution</th> </tr> </thead> <tbody> <tr> <td>A = 32H</td> <td>A = 65H</td> </tr> <tr> <td>CY = 1</td> <td>CY = 0</td> </tr> </tbody> </table> <p>Rotate Operation :</p>	RLC A		Before Execution	After Execution	A = 32H	A = 65H	CY = 1	CY = 0
RLC A									
Before Execution	After Execution								
A = 32H	A = 65H								
CY = 1	CY = 0								

RR A	
Description	<ul style="list-style-type: none"> • Rotate accumulator right. • Rotate register A contents by one bit towards right. This means A0 goes into A7, A7 into A6, A1 into A0 and so on.
Operation	$(A_n) \leftarrow (A_{n+1})_{n=0-6} (A_7) \leftarrow (A_0)$
Example: 1	Suppose A = 1C = 0001 1100 Then, after RR A, A = 0000 1110 = 0E H

Example: 2	RR A	
	Before Execution	After Execution
	A = 32H	A = 19H
Rotate Operation : 		

RRC A											
Description	<ul style="list-style-type: none"> • Rotate accumulator right through carry flag. • Rotate contents of accumulator right by one bit through carry. A0 goes into carry; carry into A7, A7 to A6 and so on. 										
	$(A_n) \leftarrow (A_{n+1})_{n=0-6} \quad (A_7) \leftarrow (C) \quad (C) \leftarrow (A_0)$										
Example: 1	Suppose A = 3A = 0011 1010 and CY = 0 Then after RRC A = 0001 1101 and CY = 0										
Example: 2	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th colspan="2" style="text-align: center;">RRC A</th> </tr> <tr> <td style="text-align: center;">Before Execution</td> <td style="text-align: center;">After Execution</td> </tr> <tr> <td style="text-align: center;">A = 32H</td> <td style="text-align: center;">A = 99H</td> </tr> <tr> <td style="text-align: center;">CY = 1</td> <td style="text-align: center;">CY = 0</td> </tr> <tr> <td colspan="2"> Rotate Operation :  </td> </tr> </table>	RRC A		Before Execution	After Execution	A = 32H	A = 99H	CY = 1	CY = 0	Rotate Operation : 	
RRC A											
Before Execution	After Execution										
A = 32H	A = 99H										
CY = 1	CY = 0										
Rotate Operation : 											

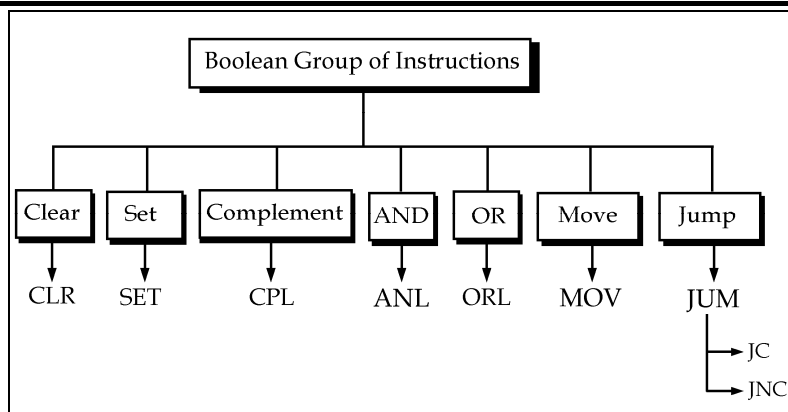
SWAP A							
Description	<ul style="list-style-type: none"> • Swap nibbles within the accumulator. • Interchange the upper 4 bits (nibbles) and lower four bits in the accumulator. This means D7 to D4 will go at D3 to D0 and D3 to D0 will go in place of D7 to D4. 						
Operation	$(A3-0) \leftrightarrow (A7-4)$						
Example: 1	Suppose A = 7D then after swap A = D7						
Example: 2	<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2" style="text-align: center;">SWAP A</th> </tr> <tr> <th style="text-align: center;">Before Execution</th> <th style="text-align: center;">After Execution</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">A = 7DH</td> <td style="text-align: center;">A = D7H</td> </tr> </tbody> </table>	SWAP A		Before Execution	After Execution	A = 7DH	A = D7H
SWAP A							
Before Execution	After Execution						
A = 7DH	A = D7H						

Summary of Logic instructions of 8051

Instructions	Operation	Description
Byte level Logical Operations		
1. ANL A,Rn	$(A) \leftarrow (A) \text{ AND } (Rn)$	AND Register to Accumulator.
2. ANL A,direct	$(A) \leftarrow (A) \text{ AND } (\text{direct})$	AND direct byte to Accumulator.
3. ANL A,@Ri	$(A) \leftarrow (A) \text{ AND } ((Ri))$	AND indirect RAM to Accumulator.
4. ANL A,#data	$(A) \leftarrow (A) \text{ AND } \text{data}$	AND immediate data to Accumulator.
5. ANL direct,A	$(\text{direct}) \leftarrow (\text{direct}) \text{ AND } (A)$	AND Accumulator to direct byte.
6. ANL direct,#data	$(\text{direct}) \leftarrow (\text{direct}) \text{ AND } \text{data}$	AND immediate data to direct byte.
7. ORL A,Rn	$(A) \leftarrow (A) \text{ OR } (Rn)$	OR register to Accumulator.
8. ORL A,direct	$(A) \leftarrow (A) \text{ OR } (\text{direct})$	OR direct byte to Accumulator.
9. ORL A,@Ri	$(A) \leftarrow (A) \text{ OR } ((Ri))$	OR indirect RAM to Accumulator.
10. ORL A,#data	$(A) \leftarrow (A) \text{ OR } \text{data}$	OR immediate data to Accumulator.
11. ORL direct,A	$(\text{direct}) \leftarrow (\text{direct}) \text{ OR } (A)$	OR Accumulator to direct byte.
12. ORL direct,#data	$(\text{direct}) \leftarrow (\text{direct}) \text{ OR } \text{data}$	OR immediate data to direct byte.
13. XRL A,Rn	$(A) \leftarrow (A) \text{ XOR } (Rn)$	Exclusive-OR register to Accumulator.
14. XRL A,direct	$(A) \leftarrow (A) \text{ XOR } (\text{direct})$	Exclusive-OR direct byte to Accumulator.
15. XRL A,@Ri	$(A) \leftarrow (A) \text{ XOR } ((Ri))$	Exclusive-OR indirect RAM to Accumulator.

16. XRL A,#data	$(A) \leftarrow (A) \text{ XOR } \text{data}$	Exclusive-OR immediate data to Accumulator.
17. XRL direct,A	$(\text{direct}) \leftarrow (\text{direct}) \text{ XOR } (A)$	Exclusive-OR Accumulator to direct byte.
18. XRL direct,#data	$\text{direct}) \leftarrow (\text{direct}) \text{ XOR } \text{data}$	Exclusive-OR immediate data to direct byte.
19. CLR A	$(A) \leftarrow 0$	Clear Accumulator.
20. CPL A	$(A) \leftarrow (\bar{A})$	Complement Accumulator.
Rotate & Swap Instructions		
21. RL A	$(A_{n+1}) \leftarrow (A_n)_{n=0-6}$ $(A_0) \leftarrow (A_7)$	Rotate Accumulator left.
22. RLC A	$A_{n+1} \leftarrow (A_n)_{n=0-6}$ $(A_0) \leftarrow (CY) \ \& \ (CY) \leftarrow (A_7)$	Rotate Accumulator left through the carry.
23. RR A	$(A_n) \leftarrow (A_{n+1})_{n=0-6}$ $(A_7) \leftarrow (A_0)$	Rotate Accumulator right.
24. RRC A	$(A_n) \leftarrow (A_{n+1})_{n=0-6}$ $(A_7) \leftarrow (CY) \ \& \ (CY) \leftarrow (A_0)$	Rotate Accumulator right through the carry.
25. SWAP A	$(A_{3-0}) \leftrightarrow (A_{7-4})$	Swap nibbles within the Accumulator.

2.15. Explain Boolean group of instructions.



1. In 8051, these instructions are more powerful and can perform clear, complement, set, move operations on bit wise rather than on type of given data.
2. Certain internal RAM (bytes 20 to 2F) and SFRs (A, B, IE, IP, P0, P1, P2, P3, PSW, TCON, and SCON) can be addressed by their byte addresses or by the address of each bit within a byte.
3. The bit-level Boolean logical op-codes operate on any addressable RAM or SFR bit.
4. The carry flag (CY) in the PSW special function register is the destination for most of the op-codes because the flag can be tested and the program flow is changed using instructions.

Mnemonic	Operation	Description
1. CLR C	$(C) \leftarrow 0$	Clear carry.
2. CLR bit	$(\text{bit}) \leftarrow 0$ Note: bit=Address of particular bit of RAM/SFR.	Clear direct bit.
3. SETB C	$(C) \leftarrow 1$	Set carry.
4. SETB bit	$(\text{bit}) \leftarrow 1$	Set direct bit.
5. CPL C	$(C) \leftarrow (\overline{C})$	Complement carry.
6. CPL bit	$\text{bit} \leftarrow (\overline{\text{bit}})$	Complement direct bit.
7. ANL C,bit	$(C) \leftarrow (C) \text{ AND } (\text{bit})$	AND direct bit to carry.
8. ANL C, $\overline{\text{bit}}$	$(C) \leftarrow (C) \text{ AND } \overline{\text{bit}}$	AND complement of direct bit to carry.
9. ORL C,bit	$(C) \leftarrow (C) \text{ OR } (\text{bit})$	OR direct bit to carry.
10. ORL C, $\overline{\text{bit}}$	$(C) \leftarrow (C) \text{ OR } \overline{\text{bit}}$	OR complement of direct bit to carry.
11. MOV C,bit	$(C) \leftarrow (\text{bit})$	Move direct bit to carry.
12. MOV bit,C	$(\text{bit}) \leftarrow (C)$	Move carry to direct bit.
13. JC rel	$(PC) \leftarrow (PC)+2$ If C= 1 then, $(PC) \leftarrow (PC)+\text{rel}$	Jump if carry is set.
14. JNC rel	$(PC) \leftarrow (PC)+2$ If C= 0 then, $(PC) \leftarrow (PC)+\text{rel}$	Jump if carry not set.
15. JB bit,rel	$(PC) \leftarrow (PC)+3$ If (bit)= 1 then, $(PC) \leftarrow (PC)+\text{rel}$	Jump if direct bit is set.
16. JNB bit,rel	$(PC) \leftarrow (PC)+3$ If (bit)= 0 then, $(PC) \leftarrow (PC)+\text{rel}$	Jump if direct bit is not set.
17. JBC bit,rel	$(PC) \leftarrow (PC)+3$ If (bit)= 1 then, $(\text{bit}) \leftarrow 0$ $(PC) \leftarrow (PC)+\text{rel}$	Jump if direct bit is set and clear bit.

CLR C	
• Description	Clear carry bit, means carry flag is made 0.

CLR Bit	
• Description	Clear the addressed bit to 0. This works only with bit oriented register.
• Example	CLR P 2.0 bit 0 in port 2 cleared. Eg. CLR 7F, means clear Bit 7 of RAM byte 2F from 16 bytes of Bit addressable RAM area.

SETB C	
• Description	Set Carry flag to 1.

SETB bit	
• Description	Set the addressed bit to 1.
• Example	SETB 01, i.e. Bit 1 of RAM byte to 20 H. SETB P1.2 i.e. bit D2 in port 1 is set to 1.

MOV C, b	
• Description	Copy the addressed bit in C.
• Example	MOV C, PO.1 Copy bit 1 port 0 to Carry flag. The contents of bit remains uncharged.

MOV b, C	
• Description	Move (copy) the carry to the addressed bit.

2.14. Explain bit-level logical instructions.

CPL C	
• Description	Complement the carry flag. If 1, make it 0 and vice versa.

CPL bit	
----------------	--

• Description	Complement the addressed bit.
• Example	CPL P1.0 : Complements the bit 0 from port 1.

ANL C, bit	
• Description	AND the carry bit and the addressed bit and put the result in carry bit.
• Example	ANL C, 03 AND carry with bit 3 in 20 H RAM location.

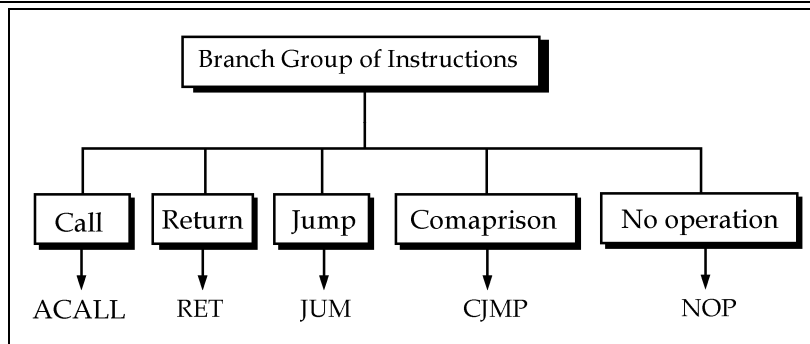
ANL C, bit	
• Description	/bit means complement of bit. The carry bit and complement of addressed bit is ANDed and result is stored in carry
• Example	ANL C, P1.2 ; the second bit in port 1 is complemented and then ANDed with carry bit.

ORL C, bit	
• Description	OR carry with the addressed bit and result of ORing stored in carry.
• Example	ORL C, 07, i.e. bit 07 in location 20 H

ORL C, / bit	
• Description	Carry is logically ORed with complement of addressed bit and result is back stored in C.
• Example	ORL C, 07

Additional Information

Branch Control group of instructions of 8051



1. These instructions are used to change the normal sequence of a program, either unconditionally or under certain test conditions.

2. These instructions sometimes make use of the flags to change the sequence of program.
3. Flags are not affected by any instruction.
4. For the instruction CJNE the carry flag is only affected.
5. The branch group mnemonics (17) are grouped in to the following types.

6. **Jump and Call Instructions:**

- The jump and call instructions are decision control group of instructions that alter the flow of the program.
- A jump or call instruction can replace the contents of the program counter with a new program address that causes program execution to begin at the code at the new address.
- A jump permanently changes the contents of the program counter if certain program conditions exist.
- A call temporarily changes the program counter to allow another part of the program to run.
- Program has the following types of decision op-codes while writing program to do certain task using microcontroller 8051.
 - Jump on bit conditions
 - Compare bytes and jump if not equal
 - Decrement byte and jump if zero
 - Jump unconditionally
 - Call a subroutine
 - Return from a subroutine

7. **Range:**

The difference, in bytes, between new address from the address in the program where the jump or call is located is called the range of the jump or call.

Example: For example, if a jump instruction is located at program address 0100H, and the jump causes the program counter to become 0130H, then the range of the jump is 30H bytes.

8. **Classification based on range:**

- **Relative range:** Maximum of +127d, -128d bytes from the instruction following the jump or call instruction
- **Absolute range:** Maximum of 2 K bytes from +127, -128d bytes from the instruction following the jump or call instruction.
- **3) Long range:** Maximum of 64K byte (any address from 0000H to FFFFH). Figure 3.1 shows the relative range of all the jump instructions.

Mnemonic	Operation	Description
CALL & Subroutines		
1. ACALL addr11	$(PC) \leftarrow (PC)+2$ $(SP) \leftarrow (SP)+1$ $((SP)) \leftarrow PCL$ $(SP) \leftarrow (SP)+1$ $((SP)) \leftarrow PCH$ $(PC_{10-0}) \leftarrow \text{addr11}$	Absolute subroutine call.
2. LCALL addr16	$(PC) \leftarrow (PC)+3$ $(SP) \leftarrow (SP)+1$ $((SP)) \leftarrow (PCL)$ $(SP) \leftarrow (SP)+1$ $((SP)) \leftarrow (PCH)$ $(PC) \leftarrow \text{addr16}$	Long subroutine call.
3. RET	$(PCH) \leftarrow ((SP))$ $(SP) \leftarrow (SP)-1$ $(PCL) \leftarrow ((SP))$ $(SP) \leftarrow (SP)-1$	Return from subroutine.
4. RETI	$(PCH) \leftarrow ((SP))$ $(SP) \leftarrow (SP)-1$ $(PCL) \leftarrow ((SP))$ $(SP) \leftarrow (SP)-1$	Return from interrupt.
5. AJMP addr11	$(PC) \leftarrow (PC)+2$ $(PC_{10-0}) \leftarrow \text{addr11}$	Absolute jump
Jump Instructions		
6. LJMP addr16	$(PC) \leftarrow \text{addr16}$	Long jump.
7. SJMP rel	$(PC) \leftarrow (PC)+2$ $(PC) \leftarrow (PC)+\text{offset}$	Short jump (relative addr).
8. JMP @A+DPTR	$(PC) \leftarrow ((A)+(DPTR))$	Jump indirect relative to the DPTR.
9. JZ rel	$(PC) \leftarrow (PC)+2$ If $(A)=0$ then, $(PC) \leftarrow (PC)+\text{rel}$	Jump if Accumulator is zero.
10. JNZ rel	$(PC) \leftarrow (PC)+2$ If $(A) \neq 0$ then, $(PC) \leftarrow (PC)+\text{rel}$	Jump if Accumulator is not zero.

11. CJNE A,direct,rel	(PC) \leftarrow (PC)+3 If (A) \neq (direct) then, (PC) \leftarrow (PC)+offset If (A)<(direct) then, (C) \leftarrow 1 If (A)>(direct) then, (C) \leftarrow 0	Compare direct byte to A and jump if not equal.
12. CJNE A,#data,rel	(PC) \leftarrow (PC)+3 If (A) \neq #data then, (PC) \leftarrow (PC)+offset If (A)<#data then, (C) \leftarrow 1 If (A)>#data then, (C) \leftarrow 0	Compare immediate to A and jump if not equal.
13. CJNE Rn,#data,rel	(PC) \leftarrow (PC)+3 If (Rn) \neq #data then, (PC) \leftarrow (PC)+offset If (Rn)<#data then, (C) \leftarrow 1 If (A)>#data then, (C) \leftarrow 0	Compare immediate to register and jump if not equal.
14. CJNE @Ri,#data,rel	(PC) \leftarrow (PC)+3 If ((Ri)) \neq #data then, (PC) \leftarrow (PC)+offset If ((Ri))<#data then, (C) \leftarrow 1 If (A)>#data then, (C) \leftarrow 0	Compare immediate to indirect and jump if not equal.
15. DJNZ Rn,rel	(PC) \leftarrow (PC)+2 (Rn) \leftarrow (Rn)-1 If (Rn) \neq 0 then, (PC) \leftarrow (PC)+rel	Decrement register and jump if not zero.
16. DJNZ direct,rel	(PC) \leftarrow (PC)+2 (direct) \leftarrow (direct)-1 If direct \neq 0 then, (PC) \leftarrow (PC)+rel	Decrement direct byte and jump if not zero.
17. NOP	(PC) \leftarrow (PC)+1	No operation.

2.16. Explain unconditional & conditional jump instructions.

2.16.1. Unconditional jump

The unconditional jump is a jump in which control is transferred unconditionally to the target location.

In 8051 there are 4 types of unconditional jump instructions. These are:

- AJMP adddr(11)
- LJMP adddr(16)

- SJMP addr(rel)
- JMP @A+DPTR

AJMP address 11	
Description	(Absolute jump) Jump to absolute (i.e in the range of 2k) as discussed earlier. This jump is unconditional jump. This is two byte instruction in which absolute address is written. After execution, program counter is loaded by the jump address.
Operation	(PC) ← (PC)+2 (PC ₁₀₋₀) ← addr11

LJMP address 16	
Description	Jump to long address range (i.e. 16bit address). This is an unconditional jump. This is three byte instruction. First is op-code followed by lower and higher bytes of 16 bit address.
Operation	(PC) ← addr16

SJMP rel	
Description	Jump to relative address (in the range of + 127 to - 128) specified in the instruction. This is an unconditional jump and two byte instruction. First is opcode and another specifies the address often called as short jump.
Operation	(PC) ← (PC)+2 (PC) ← (PC)+offset

JMP @ A + DPTR	
Description	Jump to the address formed by adding A to the DPTR. This is single byte unconditional jump. The address can be anywhere in program memory (i.e. 16 bit add). In this instruction neither A nor DTPR in changed.
Operation	(PC) ← ((A)+(DPTR))

2.16.2. Conditional jump

JC rel	
Description	In the JC instruction, if CY = 1 it jumps to the target address. If CY = 0, it will not jump but will execute the next instruction below JC.
Operation	(PC) ← (PC)+2 If (CY)=1 then, (PC) ← (PC)+rel

JNC rel	
Description	In the JNC instruction, if CY = 0 it jumps to the target address. If CY = 1, it will not jump but will execute the next instruction below JNC.
Operation	(PC) ← (PC)+2 If (CY)=0 then, (PC) ← (PC)+rel

JZ rel.	
Description	Jump to the relative address if a content of register A is 00H (Accumulator is zero).
Operation	(PC) ← (PC)+2 If (A)=0 then, (PC) ← (PC)+rel

JNZ rel	
Description	Jump to the relative address if accumulator is not zero.
Operation	(PC) ← (PC)+2 If (A) ≠ 0 then, (PC) ← (PC)+rel

CJNE A, direct, rel.	
Description	Compare contents of accumulator with the contents of direct address and if not equal, then jump to relative address. Here the status of carry flag is checked. If accumulator contents are less than direct address contents, then carry flag is set otherwise it is reset.
Operation	(PC) ← (PC)+3 If (A) ≠ (direct) then, (PC) ← (PC)+offset If (A)<(direct) then, (C) ← 1 If (A)>(direct) then, (C) ← 0

CJNE A, # data, rel.	
Description	The contents of register A is compared with immediate data and if not equal then jump to relative address specified in the instruction. Carry flag as explained earlier.

	$(PC) \leftarrow (PC)+3$ If $(A) \neq \#data$ then, $(PC) \leftarrow (PC)+offset$ If $(A) < \#data$ then, $(C) \leftarrow 1$ If $(A) > \#data$ then, $(C) \leftarrow 0$
--	---

CJNE Rn, # data, rel.	
Description	Compare the contents of register Rn (R0 - R7) with immediate data specified in the instruction and if not equal jump to specified relative address. The status of carry flag is as explained in previous instructions.
Operation	$(PC) \leftarrow (PC)+3$ If $(Rn) \neq \#data$ then, $(PC) \leftarrow (PC)+offset$ If $(Rn) < \#data$ then, $(C) \leftarrow 1$ If $(Rn) > \#data$ then, $(C) \leftarrow 0$

CJNE @ Ri, # data, rel.	
Description	Compare contents of the address contained in Ri with immediate number (data) specified in the instruction and if not equal then jump to specified address (relative).
Operation	$(PC) \leftarrow (PC)+3$ If $((Ri)) \neq \#data$ then, $(PC) \leftarrow (PC)+offset$ If $((Ri)) < \#data$ then, $(C) \leftarrow 1$ If $((Ri)) > \#data$ then, $(C) \leftarrow 0$

DJNZ Rn, rel.	
Description	Decrement the contents of register Rn (R0... R7) specified in the instruction by one and jump to the relative address if register is not zero. Here decrement and jump is performed in a single instruction. Whereas in processor like 8085 two instructions are written.
Operation	$(PC) \leftarrow (PC)+2$ $(Rn) \leftarrow (Rn)-1$ If $(Rn) \neq 0$ then, $(PC) \leftarrow (PC)+rel$

DJNZ direct, rel.	
Description	Decrement the contents of direct address by one and if not zero jump to relative address specified in the instruction.
Operation	$(PC) \leftarrow (PC)+2$ $(direct) \leftarrow (direct)-1$ If $direct \neq 0$ then, $(PC) \leftarrow (PC)+rel$

Additional Information

Conditional Jump Instructions

Instruction	Action
JZ	Jumps if A = 0
JNZ	Jumps if A \neq 0
DJNZ	Decrement and Jumps if register \neq 0
CJNE A, data	Jumps to the target address if A \neq 0
CJNE reg, #data	Jumps if byte \neq #data
JC	Jumps if CY = 1
JNC	Jumps if CY = 0
JB	Jumps if bit = 1
JNB	Jumps if bit = 0
JBC	Jumps if bit = 1 and clear bit

- All conditional jumps are short jumps
- It must be noted that all conditional jumps are short jumps, **meaning** that the address of the target must be within -128 to +127 bytes of the contents of the program counter (PC). This very important concept is discussed at the end of this section.

CHAPTER 3


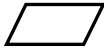





8051 Programming Concepts

OBJECTIVES

- 3.1. List the various symbols used in drawing flow charts.
- 3.2. Draw flow charts for simple problems.
- 3.3. Write programs in mnemonics to illustrate the application of data copy instructions.
- 3.4. Illustrate the application of jump instruction in the program.
- 3.5. Write a program using counter techniques.
- 3.6. Write programs of instructions to perform single byte, double byte and multi byte addition and subtraction.
- 3.7. Define a subroutine and explain its use.
- 3.8. Explain the sequence of program when subroutine is called and executed.
- 3.9. Explain information exchange between the program counter and the stack and identification of stack pointer register when a subroutine is called.
- 3.10. Explain PUSH & POP instructions.
- 3.11. Illustrate the concept of nesting, multiple ending and common ending in subroutines.
- 3.12. Use input/output, machine related statements in writing assembly language programs.
- 3.13. Explain the term debugging a program.
- 3.14. List the important steps in writing and trouble shooting a simple program.
- 3.15. Explain the principles of single step and break point debugging techniques.
- 3.16. Write simple programs to setup time delay using counter & a single register.
- 3.17. Calculate the time delay in the program given the clock frequency.

3.1. List the various symbols used in drawing flow charts.

- Flowchart is a graphical or pictorial representation of an algorithm. It is used to represent algorithm steps into graphical format.
- A flowchart is a set of symbols that indicate various operations in the program.
- For every process, there is a corresponding symbol in the flowchart. Once an algorithm is written, its pictorial representation can be done using flowchart symbols.
- The flowchart symbols as given below are as per connection followed by International Standard Organization (ISO).

rff	Name	Symbol	Description
1	Terminator Symbol		<ul style="list-style-type: none"> • This symbol looks like a flat oval or is egg shaped. • It is the first symbol and last symbol of the program logic of flowchart.
2	Input/Output Symbol		<ul style="list-style-type: none"> • The input/output symbol looks like a parallelogram. • These symbols are used to denote any function of an I/O device in the program.
3	Process Symbol		<ul style="list-style-type: none"> • The process symbol looks like a rectangle. • This symbol is used primarily for calculations and initialization of memory locations. All the arithmetic operations, data movements, initialization operations.
4	Decision Symbol		<ul style="list-style-type: none"> • The decision symbol looks like a diamond shape. • It is used when we want to take any decision in the program.
5	Connector Symbol		<ul style="list-style-type: none"> • This symbol is used to connect the various portion of a flowchart. • This is normally used when the flow chart is split between two pages.
6	Flow Lines Symbol		<ul style="list-style-type: none"> • These lines show the flow of control through the program.
7	Predefined Process		<ul style="list-style-type: none"> • This symbol indicates a call to a module that is not included with in the current program.

3.2. Draw flow charts for simple problems.

Program - 1: Draw a flow chart to load the 32H into R0, R1, R2 and Accumulator.

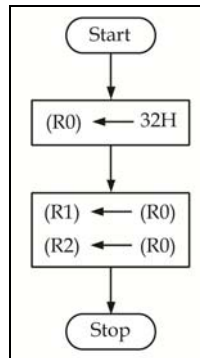


Fig. 3.2.(a)

Program - 2: Draw a flow chart to divide the 87H by 23H, save the quotient in register R5 and remainder in R4.

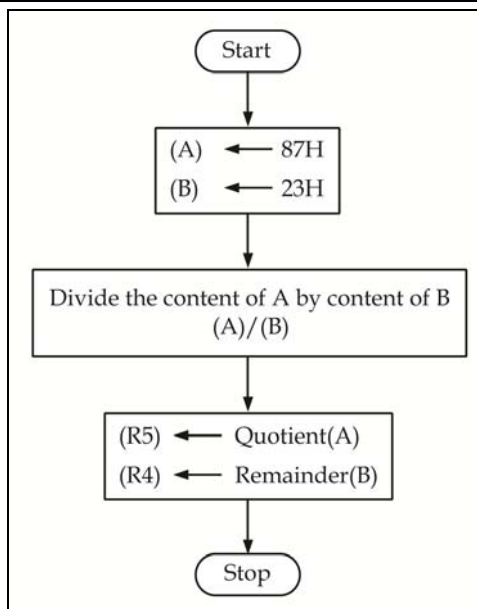


Fig.3.2(c)

Programme - 3: Draw a flow chart to add 65H and 87H, if carry is generated, preserve the carry into register R0 and sum into the register R1.

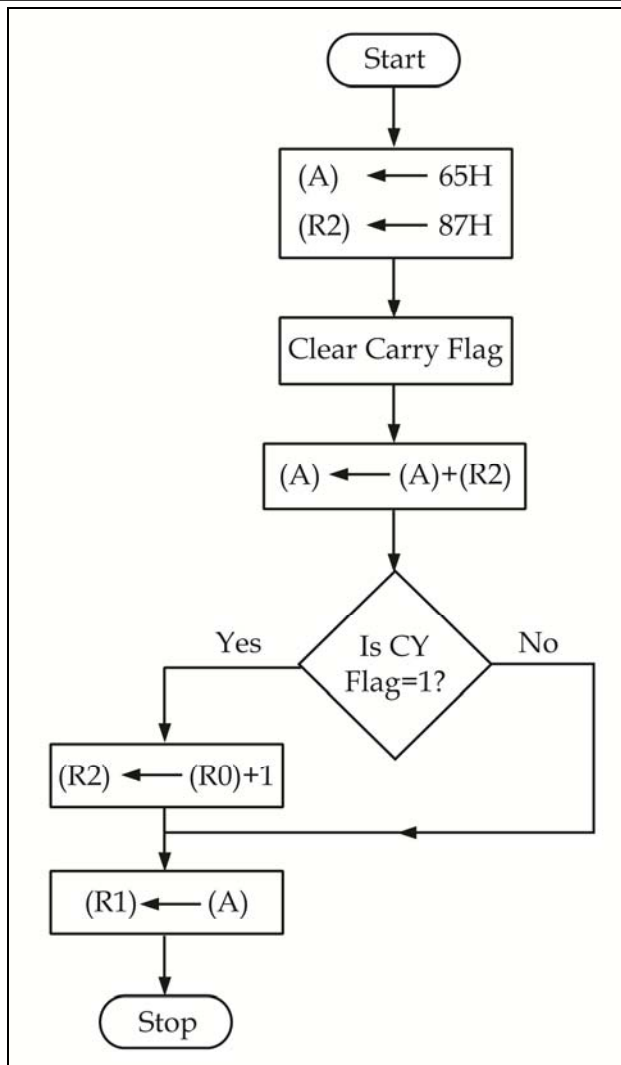


Fig. 3.2(b)

3.3. Write programs in mnemonics to illustrate the application of data copy instructions.

Program - 1: Write a program to load Accumulator, Register B, register R0 and with 99H.

Memory Address	Opcode	Hex Code	Comment
0000	MOV A,#99H	74	Load the Accumulator with 99H
0001		99	
0002	MOV B,A	F5	Get the data 99H into register B

0003	MOV R0,A	F8	Get the data 99H into register R0
0004	HERE: SJMP HERE	80	Stop the execution of the program
0005		FE	

Program- 2: Write a program to transfer the data at the RAM locations 45H, 46H, 47H into the registers R3, R4 and R5 respectively.

Memory Address	Opcode	Hex Code	Comment
0000	MOV R0,#45H	78	Load the Register R0 with 45H
0001		45	
0002	MOV A,@R0	E2	Get the data at the RAM location 45H into A
0003	MOV R3,A	FB	Transfer the Accumulator data into Register R3
0004	INC R0	08	Increment the pointer; (R0)=(R0)+1=46H
0005	MOV A,@R0	E2	Get the data at the RAM location 46H into A
0006	MOV R4,A	FC	Transfer the Accumulator data into Register R4
0007	INC R0	08	Increment the pointer; (R0)=(R0)+1=47H
Memory Address	Opcode	Hex Code	Comment
0008	MOV A,@R0	E2	Get the data at the RAM location 47H into A
0009	MOV R5,A	FD	Transfer the Accumulator data into Register R5
000A	HERE: SJMP HERE	80	Stop the execution of the program
		FE	

Program - 3: Write a program to transfer the data at the External RAM locations 4500H, 4501H, and 4502H into the registers R0, R1 and R2 respectively.

Memory Address	Opcode	Hex Code	Comment
0000	MOV DPTR,#4500H	90	Load the DPTR with 4500H
0001		00	
0002		45	

0003	MOVX A,@DPTR	E0	Get the data at the RAM location 4500H into A
0004	MOV R0,A	F8	Transfer the Accumulator data into Register R0
0005	INC DPTR	A3	Increment the data pointer;
0006	MOVX A,@DPTR	E0	Get the data at the RAM location 4501H into A
0007	MOV R1,A	F9	Transfer the Accumulator data into Register R1
0008	INC DPTR	A3	Increment the data pointer;
0009	MOVX A,@DPTR	E0	Get the data at the RAM location 4502H into A
000A	MOV R2,A	FA	Transfer the Accumulator data into Register R2
000B	HERE: SJMP HERE	80	Stop the execution of the program
000C		FE	

Program 4: write a program to move an array (block) of data from one location to another location of internal RAM.

Memory Address	Opcode	Hex Code	Comment
0000	MOV R0,#50H	78	Initialize the Source memory location with 50H
0001		50	
0002	MOV R1,#60H	79	Initialize the Destination memory location with 60H
0003		60	
0004	MOV R6,#0AH	7E	Load the count value into the register R2
0005	L1:MOV A,@R0	E2	Transfer the source data into Accumulator
0006	MOV @R1,A	F7	Transfer the source data to destination location
0007	INC R0	08	Increment the source location pointer; (R0)=(R0)+1
0008	INC R1	09	Increment the destination location pointer; (R1)=(R1)+1
0009	DJNZ R6,L1	DE	Decrement and compare the content of R6 with zero
000A		**06	
000B	HERE: SJMP HERE	80	Stop the execution of the program
000C		FE	

** Calculation of Offset address of DJNZ: Offset address of Next memory location of DJNZ (0B) – Offset address of target address L1 (05) = 06.

Programme - 5: Write a program to move an array (block) of data from external memory location to internal RAM locations.

Memory Address	Opcode	Hex Code	Comment
0000	MOV R0,#70H	78	Initialize the Destination memory location with 70H
0001		70	
0002	MOV DPTR,#3300H	90	Initialize the Data pointer with 3300H
0003		00	
0004		33	
0005	MOV R6,#0AH	7E	Load the count value into the register R2
0006		0A	Transfer the source data into Accumulator
0007	L1:MOVX A,@DPTR	E0	Transfer the source data to destination location
0008	MOV @R0,A	F6	Increment the source location pointer; (R0)=(R0)+1
0009	INC DPTR	A3	Increment the destination location pointer; (R1)=(R1)+1
000A	INC R0	08	Decrement and compare the content of R6 with zero
000B	DJNZ R6,L1	DE	
000C		**06	
000D	HERE:SJMP HERE	80	Stop the execution of the program
000E		FE	

** Calculation of Offset address of DJNZ: Offset address of Next memory location of DJNZ (0D) – Offset address of target address L1 (07) =06

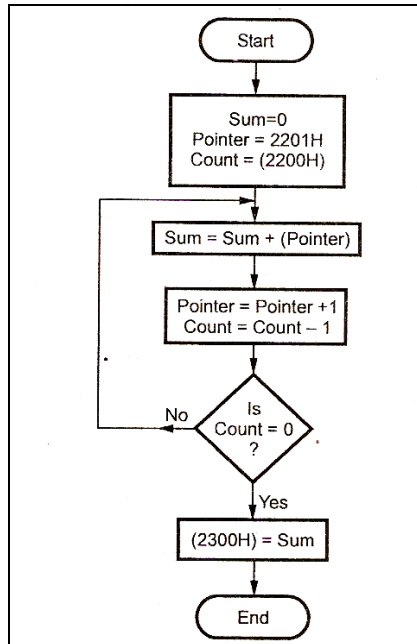
3.4. Illustrate the application of jump instruction in the program.

Program 1: Write a program to clear 16 RAM locations starting at RAM address 60H.

Label	Instruction	Comment
	CLR A	A=0
	MOV R1,#60H	Load pointer, R1=60H.
	MOV R7,#10H	Load counter, R7=16 (10 in hex).
UP	MOV @R1,A	Clear RAM location R1 points to.
	INC R1	Increment R1 pointer.
	DJNZ R7,UP	Loop until counter=zero.

Program 2: Write a Program to sum of given N numbers (series of numbers). The length of the series is in memory location 2200H and the series itself begins from memory location 2201H. Assume the sum to be 8-bit number so you can ignore carries. Store the sum at memory location 2300H.

a) Flow chart

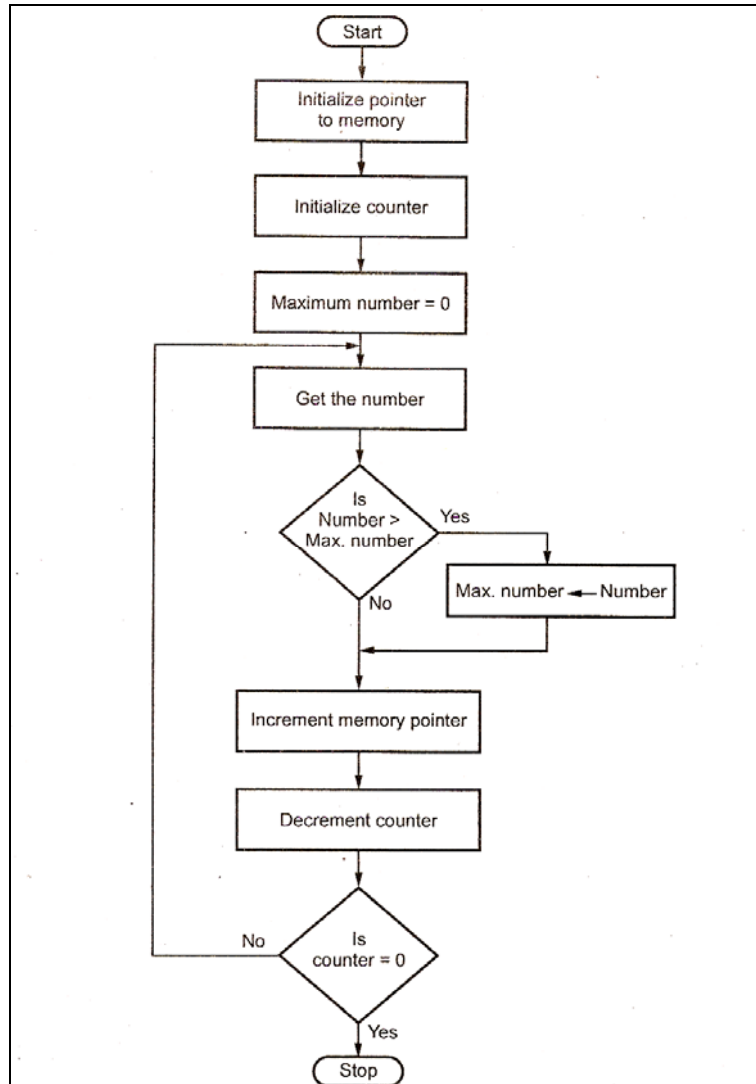


b) Program

Label	Instructions	Comments
	MOV DPTR,#2200H	Initialize memory pointer.
	MOVX A,@DPTR	Get the count.
	MOV R0,A	Initialize the iteration counter.
	INC DPTR	Initialize pointer to array of numbers.
	MOV R1,#00H	Result = 0.
UP	MOVX A,@DPTR	Get the number.
	ADD A,R1	(A)← Result + (A).
	MOV R1,A	Result ← (A).
	INC DPTR	Increment the array pointer.
	DJNZ R0,UP	Decrement iteration count if not zero repeat.
	MOV DPTR,#2300H	Initialize memory pointer.
	MOV A,R1	Get the result.
	MOVX @DPTR,A	Store the result.
STOP	SJMP STOP	Stop execution.

Program 3: Write a Program to find biggest data value in given data array.

a) Flow chart



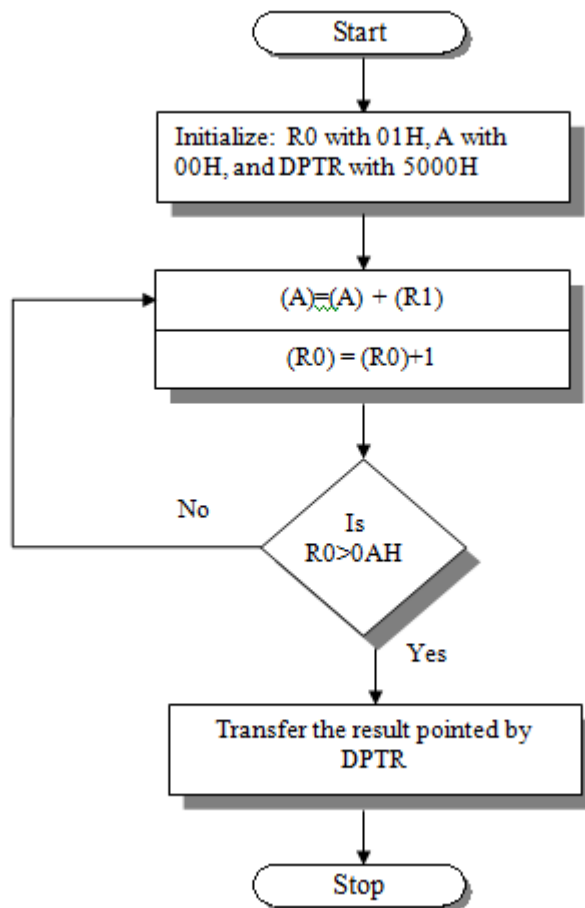
b) Program

Label	Instructions	Comments
	MOV DPTR,# 2000	Initialize pointer to memory where numbers are stored.
	MOV R0,#0AH	Initialize counter.
	MOV R3,#00H	Maximum=0.

AGAIN	MOVX A,@DPTR	Get the number from memory.
	CJNE A, R3,NE	Compare number with maximum number.
	AJMP SKIP	If equal to SKIP.
NE	JC SKIP	If not equal check for carry. If carry go to SKIP.
	MOV R3,A	Otherwise maximum=number
SKIP	INC DPTR	Increment memory pointer.
	DJNZ R0,AGAIN	Decrement count, if count = 0 stop. Otherwise go to AGAIN.
STOP	SJMP STOP	Stop execution.

Program 4: Write a Program to find the sum of first N natural numbers.

a) Flow chart



b) Program

Label	Instructions	Comments
	MOV A, #00H	A is initialized with 00H.
	MOV R0,#01H	Register R0 is initialized with 01H.
	MOV R2,#00H	Register R2 is initialized with 00H.
Top1	MOV A,R2	Bring the partial result to accumulator.
	ADD A,R0	Add the generated number to accumulator.
	MOV R2,A	Transfer the result to R2 register.
	INC R0	Do the generated natural number is 11.
	CJNE R0,#0BH,Top1	If not go to the label Top1.
	MOVDPTR,5000H	Move the address 5000H into DPTR.
	MOVX @DPTR,A	Result is moved into memory location.
STOP	SJMP STOP	Stop execution.

Note: Above program adds the generated natural numbers as follows $10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 55$

Additional Information

8051 timer as an event counter

When timer/counter is used as an event counter, the source of clock pulses is outside the 8051. If $C/\bar{T} = 1$ in the TMOD register, timer/counter is used as an event counter and the source of clock pulses is outside the 8051. The pulses are fed from T0(P3.4) and T1(P3.5).

In the TMOD register, if $C/\bar{T} = 0$, the timer gets pulses from the crystal, when $C/\bar{T} = 1$ the timer is used as counter and gets its pulsed from outside the 8051. Therefore, when $C/\bar{T} = 1$ the counter count up as pulses are fed from pin 14 and 15. These pins are called T0 and T1, notice that these two pins belong to part 3. In the case of timer 0, when $C/\bar{T} = 1$, pin 3.4 provides the clock pulse and the counter counts up for each clock pulse coming from that pin. Similarly for timer 1, when $C/\bar{T} = 1$ each clock pulse coming in from pin 3.5 makes the counter count up.

a) Counter 0 in Mode 1

The simplified block diagram of Counter 0 in Mode 1 is shown in 3. The counter counts up when the logic signal on pin T0 (P3.4) goes from high level to low level (1 to 0 transition).

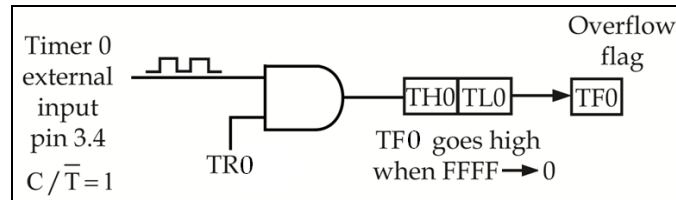


Fig. Counter 0 in mode 1

Timer/counter is used as an event counter by making $C/\bar{T}=1$ in the TMOD register. TR0 is used to start and stop the counter.

Counter 0 starts counting up at the rate at which the transitions occur at pin T0 (P3.4) of 8051. The counting will start from the 16-bit initial value present in the TH0 and TL0 registers. When the counter value exceeds FFFFH, timer overflow flag TF0 is set to 1.

b) Counter 1 in Mode 1

The simplified block diagram of Counter 1 in Mode 1 is shown in fig. The counter counts up when the logical signal on pin T1 (P3.4) goes from high level to low level.

Timer/counter is used as an event counter by making $C/\bar{T}=1$ in the TMOD register. TR1 is used to start and stop the counter.

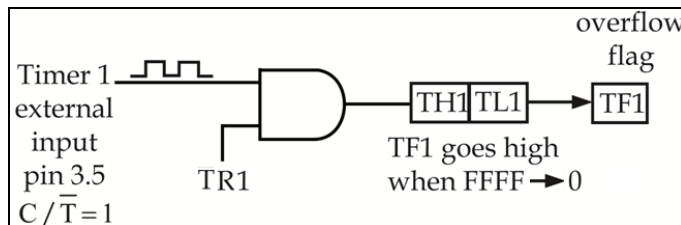


Fig. Counter 1 in mode 1

Counter 1 starts counting up at the rate at which the transitions occur at pin T1 (P3.4) of 8051. The counting will start from the 16-bit initial value present in the TH1 and TL1 registers. When the counter value exceeds FFFFH, timer overflow flag TF1 is set to 1.

3.5. Write a program using counter techniques.

Program 6: Write a program to count the pulses of an input signal every second and display the count value on ports 1 and 2.

Solution:

The external input clock pulses can be fed to one of the timers say timer '0'. Hence timer '0' is acting as counter ($C/\bar{T}=1$) and the mode of timer '0' can be mode 1 ($M_1M_0=01$) for a very large count (16-bit counter-counts from 0000 to a maximum value of FFFFH). The external input is to be fed to pin P3.5. We need a 1 second timing to be generated. This can be done by using Timer '1' in mode 1 (refer example 8.5).

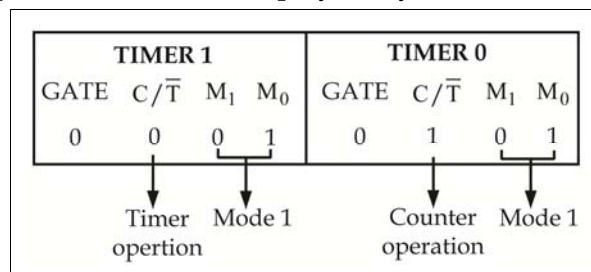
Maximum delay by timer 1 in mode 1

$$= (\text{FFFFH}) \times \frac{12}{\text{Crystal frequency}} = \frac{65535 \times 12}{11.0592 \text{ MHz}} = 0.0711 \text{ seconds}$$

This is repeated 14 times (register R0 is used as counter) to get $14 \times 0.0711 \approx 1$ second delay.

Algorithm:

1. Initialize timer 1 as timer, mode 1 and timer 0 as counter, mode 1.
2. Set P3.5 as input pin (to feed external pulses to be counted).
3. Initialize the timer 0 registers as 0000 (to count from 0) & start timer 0.
4. Initialize counter (R0) with 14 (for 1 second delay).
5. Initialize timer 1 registers with initial count (here 0000H for maximum delay of 0.0711 seconds).
6. Start timer1.
7. Wait till timer 1 overflows ($TF1=1$).
8. Stop timer 1 and clear TF1.
9. Decrement counter R0 and if not zero, repeat from step 5.
10. Display the count accumulated in TH0:TL0 on port 2 and 1.
11. Repeat from step 3 for continuous display every second.

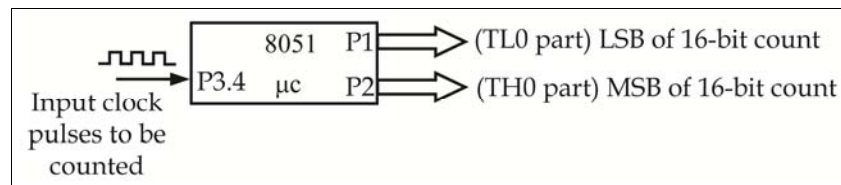


Hence TMOD = 15 H

Assembly Language Program:

	ORG 00H	
	MOV TMOD, #15H	; Timer 1 -- timer, mode 1 and ; timer 0 -- counter, mode 1
	SETB P3.4	; make port pin as input to accept external clock
rept:	MOV TL0, #00	; initialize counter (timer0) to count from zero
	MOV TH0, #00	
	SETB TR0	; start timer 0
	MOV R0, #14	; to repeat timer 1 14 times for 1 second delay
again:	MOV TL1, #00	; Initialize timer 1
	MOV TH1, #00	; start timer 1
wait:	JNB TF1, wait	; wait till timer 1 overflows
	CLR TR1	; stop timer 1
	CLR TF1	; clear timer flag
	DJNZ R0, again	; repeat till R0 becomes zero
	CLR TR0	; stop timer 0
	MOV A, TL0	The count in timer 0 ; (since 1 second is over)
	MOV P1, A	; is displayed on port P1
	MOV A, TH0	; upper part of count on port P2
	MOV P2, A	
	SJMP rept	; repeat for next 1 second count
	END	

A schematic of the hardware set-up for the above example is shown in fig.

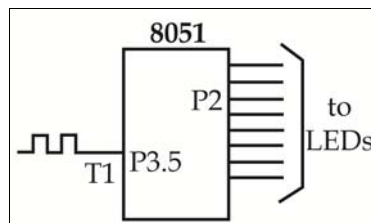


Program 7: Assuming that clock pulses are fed into pin T1, write a program for counter 1 in mode 2 to count the pulses and display the state of the TL1 count on P2.

Solution:

	MOV TMOD, #01100000B	counter 1, mode 2, C/T = 1 external pulses
	MOV TH1, #0	clear TH1
	SETB P3.5	make T1 input
AGAIN:	SETB TR1	start the counter
BACK:	MOV A, TL1	get copy of count TL1
	MOV P2, A	display it on port 2
	JNB TF1, Back	keep doing it if TF=0
	CLR TR1	stop the counter 1
	CLR TF1	make the counter 1
	SJMP AGAIN	keep doing it

Notice in the above program the role of the instruction “SETB P3.5”. Since ports are set up for output when the 8051 is powered up, we make P3.5 an input port by making it high. In other words, we must configure (set high) the T1 pin (pin P3.5) to allow pulses to be fed into it.



P2 is connected to 8 LEDs and input T1 to pulse.

In example 4.15 we are using timer 1 as an event counter where it counts up as clock pulses are fed into pin P3.5. These clock pulses could represent the number of people passing through an entrance, or the number of wheel rotations, or any other event that can be converted to pulses.

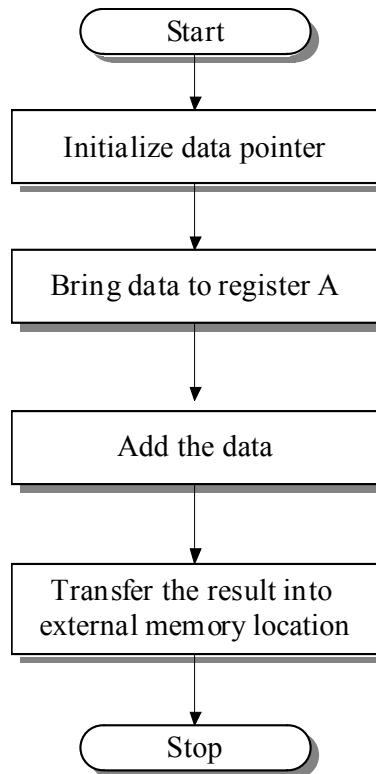
3.6. Write programs of instructions to perform single byte, double byte and multi byte addition and subtraction.

Single byte and Multi byte Addition

1. Addition of Two 8-bit (Two 1-byte) Numbers

Program 1: Write a program to add two 8-bit numbers 14H and 23H and store the result in the location 8500H.

a) Flow chart



b) Program

Label	Instructions	Comments
	MOV DPTR,#8500H	Initialize the data pointer.
	MOV A,#14H	Move 14H to A.
	ADD A,#23H	Add A and 23H.
	MOVX @DPTR,A	Result is moved into memory location.
STO	SJMP STOP	Stop execution.

Program 2: Write a program to ADD given two 8-bit numbers and store the result in a specified external memory location.

Label	Instructions	Comments
	MOV DPTR,#8500H	Initialize the data pointer.
	MOVX A,@DPTR	Data1 is copied into accumulator.
	MOV R2,A	Data1 is copied into R2 register.
	INC DPTR	Update the pointer.
	MOVX A,@DPTR	Data2 is copied into accumulator.
	ADD A,R2	Data1 & Data2 are added
	INC DPTR	Update the pointer.
	MOVX @DPTR,A	Result is transferred into memory location.
STOP	SJMP STOP	Stop execution.

2. Addition of Three 8-bit (Three 2-byte) Numbers

Program 3: Write a program to find the sum of three data bytes and store the result in a specified external memory location.

Label	Instructions	Comments
	MOV DPTR,#8500H	Initialize the data pointer
	MOV A,#29H	Move 29H to A.
	ADD A,#35H	Add A & 35H.
	ADDC A,#47H	Add A, 47H and CY.
	MOVX @DPTR,A	Result is moved into memory location.
STOP	SJMP STOP	Stop execution.

3. Addition of Two 16-bit (Two 2-byte) Numbers

Program 4: Write a program to ADD given two 16-bit numbers and store the result in a specified external memory location.

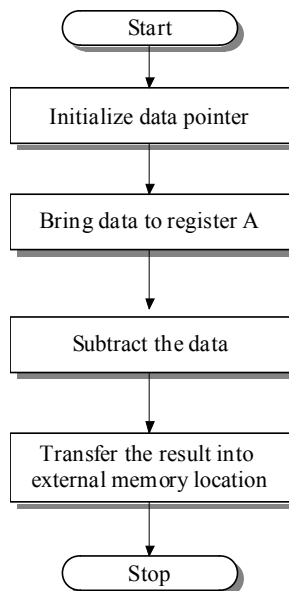
Label	Instructions	Comments
	CLR C	Clear the C flag to zero.
	MOV A,#DATA L1	Load DATA1 LSD to A.
	MOV A,#DATA L2	Add the content of A and DATA 2 LSD.

	MOV DPTR,#4500H	Load DPTR = 4150H.
	MOVX @DPTR,A	Move the content in A to external memory whose address is 4150H.
	INC DPTR	Increment the DPTR.
	MOV A,#DATA M1	Load DATA1 MSD to A.
	ADDC A,#DATA M2	Add A, DATA 2 MSD and CY.
	MOVX @DPTR,A	Move the sum in A to external memory address 4151H.
HERE	SJMP HERE	Stay in this loop.

4. Subtraction of Two 8-bit Numbers

Program 5: Write a program to subtract two 8-bit numbers 23H and 14H and store the result in the location 8500H.

a) Flow chart



b) Program

Label	Instructions	Comments
	MOV DPTR, #8000H	Initialize the data pointer.
	MOV A,#23H	Move 23H to A.
	CLR C	Clear carry flag.
	SUBB A,#14H	Subtract 14H from A.

	MOVX @DPTR,A	Result is moved into memory location.
STOP	SJMP STOP	Stop execution of the program.

Program 6: Subtract an 8-bit number from another 8-bit number and store the result in memory

Label	Instructions	Comments
	MOV DPTR, #8500H	Initialize the data pointer.
	MOVX A,@DPTR	Data 1 is loaded into accumulator.
	MOV R2,A	Data 1 is copied into R2 register.
	INC DPTR	Update the pointer.
	MOVX A,@DPTR	Data 2 is loaded into accumulator.
	CLR C	Clear the carry flag.
	SUBB A,R2	Subtract data1 from data2.
	JNC DOWN	If no carry goto DOWN.
	CPL A	1's Complement
	INC A	2's Complement
DOWN	INC DPTR	Update the pointer
	MOVX @DPTR,A	Result is transferred into memory
STOP	SJMP STOP	Stop execution.

5. Subtraction of two 16-bit numbers

Program 7: Subtract a 16-bit data from another 16-bit data and store the result in memory.

Label	Instructions	Comments
	CLR C	Clear the C (initial borrow as zero).
	MOV A,#DATA L1	Load DATA1 LSD to A.

	SUBB A,#DATA L2	Subtract with borrow the content of A and Data L2.
	MOV DPTR,#4500	Load DPTR = 4500H.
	MOVX @DPTR,A	Move the content in A to external memory whose address is 4500H.
	INC DPTR	Add one to DATA pointer.
	MOV A,# DATA M1	Load DATA1 MSD to A.
	SUBB A,#DATA M2	Subtract with borrow the content of A DATA2 MSD.
	MOVX @DPTR,A	Move the result in A into memory whose address is at data pointer.
HLT	SJMP HLT	Stay in the loop.

3.7. Define a subroutine and explain its use.

During the execution of a program, certain operations have to be repeated at different times within the program operating on different parameters. This would be a waste of memory space. Instead of such repeated operations, a min program may be written for the operation as a sub-program or routine in the memory and it may be called into main program whenever necessary. Then such sub-program is known as 'subroutine'.

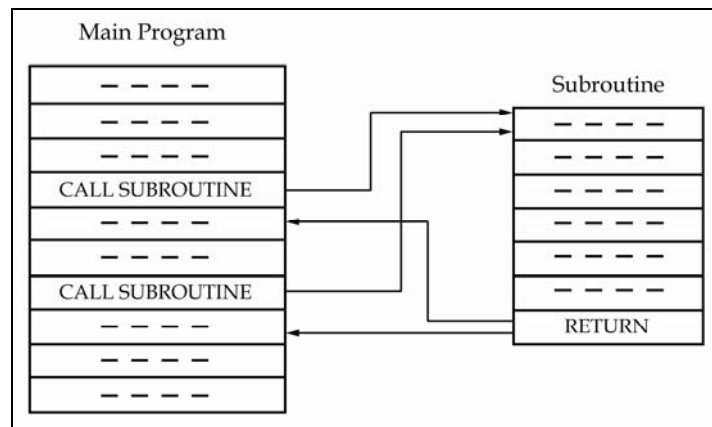


Fig. Structure of Subroutine

Thus a subroutine is defined as a group of instructions written separately from the main program to perform a function that occurs repeatedly from the main program to perform a function that occurs repeatedly in the main program.

Note that the concept of subroutine is mainly used to avoid the repetition of small programs. The structure of the subroutine is shown in Fig. in which they are called at various points of the main program by CALL instruction where ever they required. The subroutine ends with RETURN instruction.

The concept of subroutine is mainly used to avoid repetition of smaller programs. Subroutine are written separately and stored in the memory. They are called at various points of the program using CALL instruction.

3.8. Explain the sequence of program when subroutine is called and executed.

LCALL is a 3-byte instruction, in which one byte is the opcode, and the other two bytes are the 16-bit address of the target subroutine. LCALL calls a program subroutine.

The following is the sequence of the LCALL instruction.

1. The content of PC is incremented by three
2. Stack pointer is incremented by one
3. The low order PC is pushed on to the stack.
4. Stack pointer is incremented by one
5. The high order PC is pushed on to the stack
6. The second byte of the instruction is loaded to PCH and the that byte of the instruction is loaded to PCL
7. Now, the instruction at (PC) is executed.
8. The last instruction of the called subroutine must be RET. After executing the called subroutine, the address from the stack is loaded into PC. The subroutine may therefore begin anywhere in the full 64K byte program memory address space. No flags are affected.

3.9. Explain information exchange between the program counter and the stack and identification of stack pointer register when a subroutine is called.

- A set of instructions written separately from the main program to perform a function that occurs repeatedly in the main program is called subroutine or subprogram or routine or procedure or function.
- The subroutine may be required by the main program or another subroutine as many times as required.

- Subroutine can be called from the main program either conditionally or unconditionally.
- Subroutine is called to main program by the CALL instructions. As an when the subroutine is called, the execution of main program is stopped and the program counter is loaded with starting memory address of the subroutine and the subroutine is running up to the RET instruction encountered by the controller, the controller return to the main program and starts to execute the remaining program.

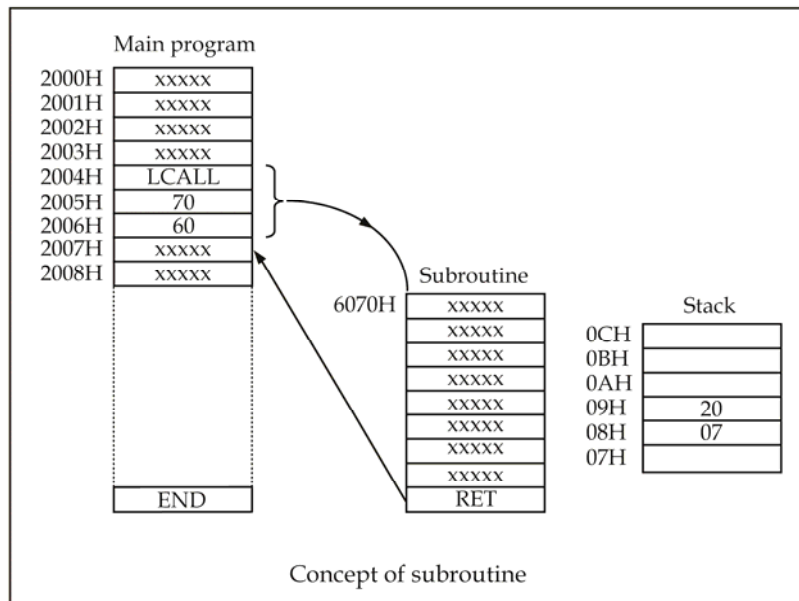


Fig. 3.9

There are so many advantages to subroutine, they are

- The main program becomes simple.
- Execution speed increases.
- It takes less up space in the ROM.
- Debugging is very easy.
- Size of the main program is reduced.
- The disadvantage of the subroutine is that need of stack
- If a subroutine is made up of few instructions, it may be advisable not to create it, since the call and return mechanism may make it slower execution instructions to place directly in the main program.

The concept of subroutine can be understood from the following diagram.

- The main program starts at 2000 H; default value of the stack is 07H.
- The controller encounters the subroutine at 2004H in the main program.

- Immediately the program counter is loaded with the starting address of the subroutine (6070H).
- The return address 2007H is loaded into the stack as the lower byte of the return address is (07) at 08H location of the stack and higher byte (20) of the return address is at 09H location of the stack.
- Now the subprogram at the location 6070H is starts to executed.
- Whenever the controller encounters the RET instruction in the subprogram, it will return back to the return address 2007H of the main program , at the same time the program counter content becomes 2007H. And controller starts to execute the remaining program.

3.10. Explain PUSH & POP instructions.

PUSH and POP are stack related instruction. These instructions must be used in direct addressing mode. No flag is affected. The data moves between areas of internal RAM are called as stack. The starting address of the stack is stored in stack pointer register (SP). By default the stack point register loaded with 07H. After execution of each PUSH instruction, the contents of the register are saved on to the stack. After transfer of data the stack pointer is incremented by 1. It is a two byte instruction and it needs two machine cycles for execution. The format of PUSH instruction is 'PUSH direct'.

For example PUSH A is invalid if we are writing the same instruction as 'PUSH 0E0H' is valid, where 0E0H is address of the register A. Similarly considering 2nd example, PUSH R5 is invalid instruction, if we are writing the same instruction as 'PUSH 05 H' is valid instruction, where 05H is the address of the register R5.

Similarly the function of the POP instruction is that getting the contents back from the stack into a given registers. For every POP instruction, the top data (content) of stack is copied to the register specified by the instruction and the stack pointer is decremented once.

- 08H to 1FH (24 locations) of the internal RAM is used for the stack.
- If a programme need more than 24 locations for stack , we have to change the SP to point to RAM
- Locations 30-7FH. This is done with the instruction "MOV SP, # nn".
- 20H to 2FH locations of RAM should not use for RAM.
- In a Program the number of Pushes and number of POPs must be equal in number

Example 1: With the following example we can understand the function of PUSH INSTRUCTION.

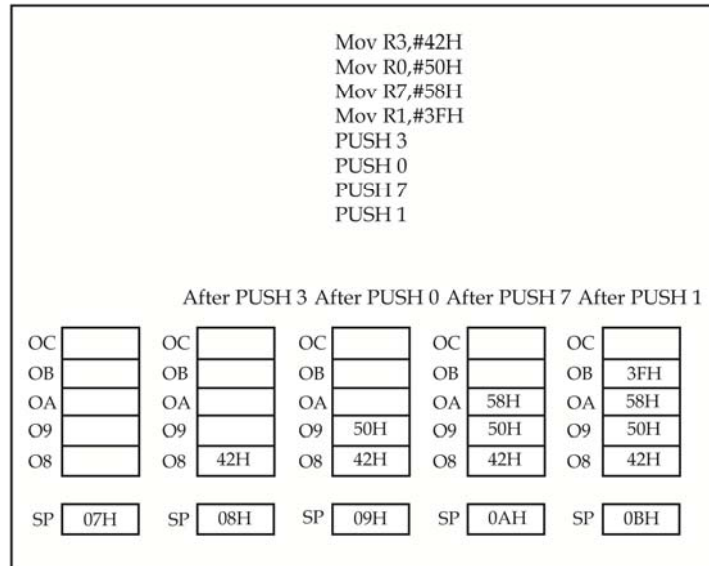


Fig 3.10(a)

Example 2:

With the following example we can understand the function of POP INSTRUCTION.

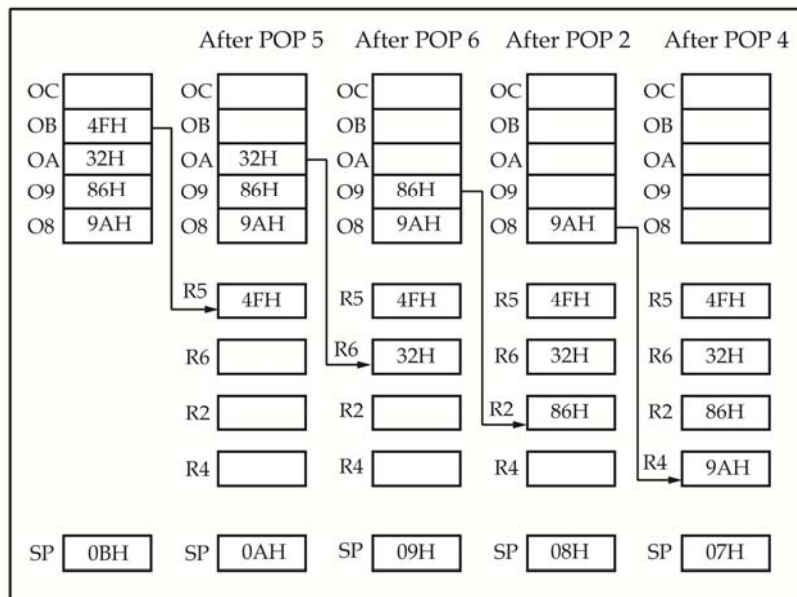


Fig. 3.10(b)

3.11. Illustrate the concept of nesting, multiple ending and common ending in subroutines.

Subroutines are three types, they are:

1. Multiple-calling of a subroutine.
2. Nesting of subroutines.
3. Multiple ending subroutines.

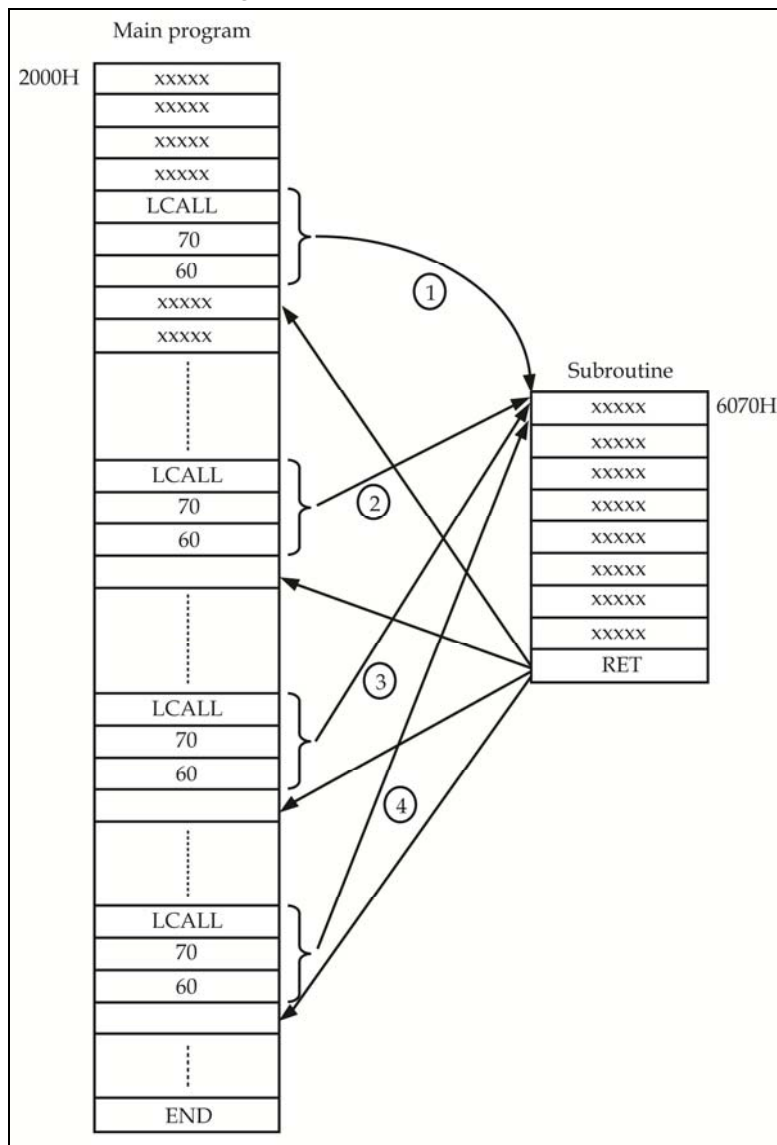


Fig. 3.11.(a)

1. Multiple-calling of a subroutine

Subroutines are normally called by the main program, calling a subroutine more than once by the main program is called "multiple calling of a subroutine."

The concept of multiple calling can be understood from the fig. 3.11.(a).

2. Nesting of subroutines

- If a subroutine is called by another subroutine and so-on. Then such programming is known as Nesting.
- The process of a subroutine calling a second subroutine and the second subroutine in its turn calling a third one and so on is called nesting of subroutines.
- Theoretically speaking, the number of subroutines that can be called by this process is infinite but, it will be limited by the size of memory.
- The nesting process limited by the available stack capacity.
- When one subroutine calls another subroutine, all the return addresses are stored on the stack.
- The fig illustrates the Concept of Nesting.

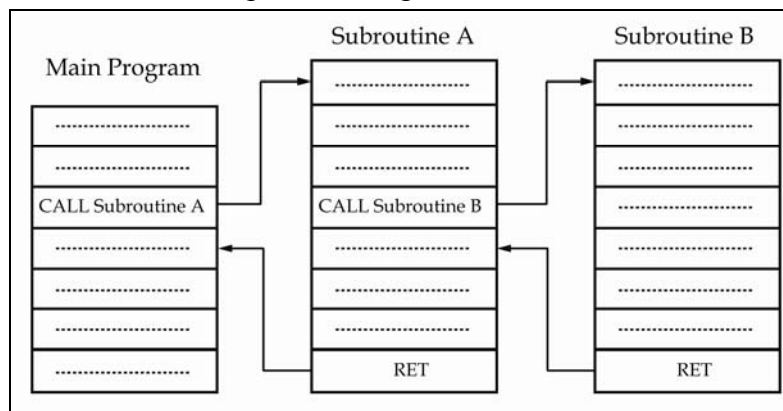


Fig. 3.11.(b)

Additional Information

- The process of nesting can be under from the above diagram.
- The main program starts at 1000H; default value of the stack is 07H.
- The controllers encounter the subroutine at 100CH in the main program.
- The address of the next instruction i.e. 100FH is placed on the stack and program control is transferred to the starting address of subroutine-1 i.e. 3100H.
- Subroutine-1 calls the subroutine-2 from the location 3102H, The address 3105H is placed on the stack and the program control is transferred to the subroutine-2.

- The subroutine-2 encounters the subroutine-3 at the location 8101H. The address 8104H is placed on the stack and the program control is transferred to the starting address (9300H) the subroutine-3. The sequence of execution return to the main program.

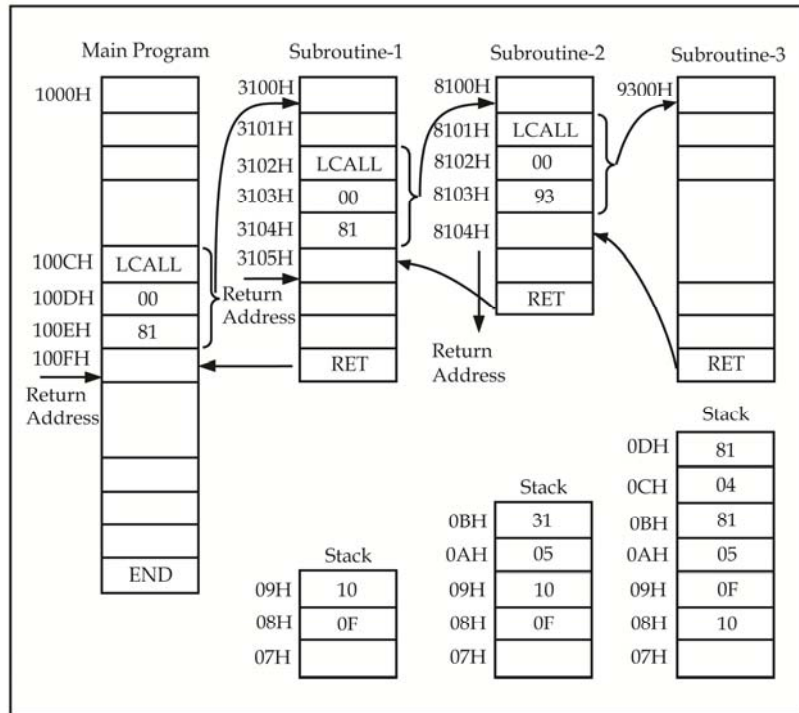


Fig.

3. Multiple ending of a subroutine

- The CALL instruction has three possible endings. In which two endings are conditional and one is unconditional.
- The two conditional returns are RZ (return zero) and RC (return carry) and the unconditional return is RET.
- From the above diagram we can understand the concept of multiple ending of subroutine.
- If the subroutine encounters the conditional return RC (return carry), the program control transfer to the main program from 1003H.
- If the subroutine encounters the conditional return RZ (return zero), the program control transfer to the main program from 1006H.
- If neither RC nor RZ encountered, the program control transferred to main program from 1008H location of the subroutine.

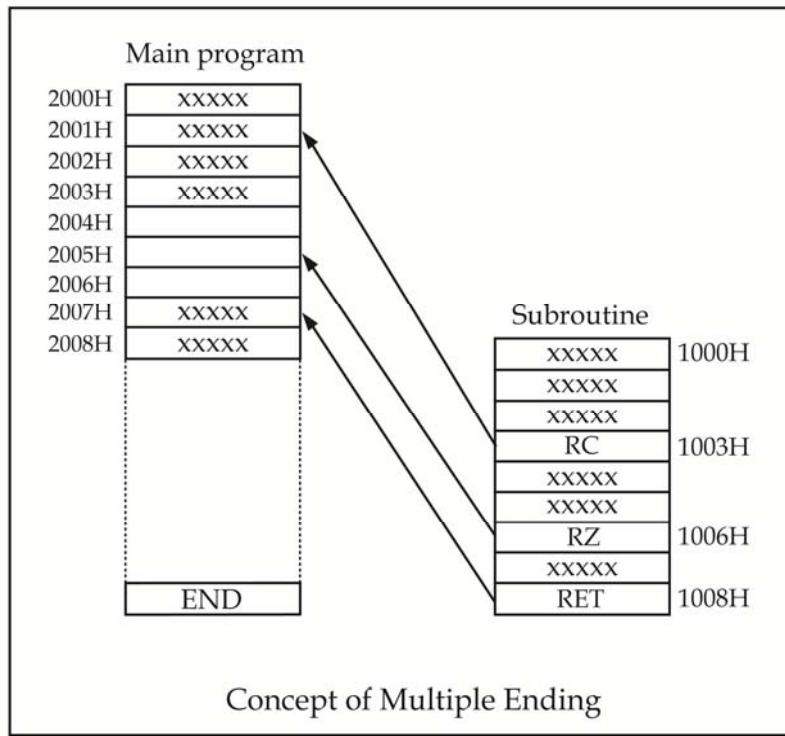


Fig.

3.12. Use input/output, machine related statements in writing assembly language programs.

3.12.1. Port 0 (P0)

Port 0 occupies a total of 8 pins named as P0.0 to P0.7 is used as:

- Input port for transferring data from peripherals to microcontroller.
(or)
- Output port for transferring data from microcontroller to peripherals.
(or)
- Lower order address bus (A0 to A7) for external memory.
(or)
- Bidirectional data bus (D0 to D7) for external memory.

Example 1: A program to toggle all the bits of Port 0

```
BACK:      MOV A, #55H      ; A = 55H
           MOV P0, A      ; Send 55H to Port 0
           ACALL DELAY    ; Wait
```

```

MOV A, #0AAH ; A = AAH
MOV P0, A    ; Send AAH to Port 0
ACALL DELAY ; Wait
SJMP BACK   ; Keep doing it

```

AAH (=10101010₂) is the complement of 55H (=01010101₂). Therefore by sending 55H and AAH continually to Port 0 we can toggle all the bits of that port.

3.12.2. Port 1 (P1)

- Port 1 occupies a total of 8 pins named as P1.0 to P1.7.
- It can be used as input/output.

Example 2: A program to get a byte from Port 0 and send it to Port 1.

```

MOV A, #0FFH; A = FFH
MOV P0, A ; Write '1's to all the bits of port 0 and make
it as an input port
BACK: MOV A, P0 ; Get data from port 0
      MOV P1, A ; Send data to port 1
      SJMP BACK ; Keep doing it

```

In the above program, the port 0 is configured as an input port by writing '1's to all the bits. Then the data is received from the port 0 and sent to port 1.

Example 3: A Program to toggle all the bits of Port 1 continuously.

```

MOV A, #55H ; A = 55H
BACK: MOV P1, A ; Send 55H to Port 1
      ACALL DELAY ; Wait
      CPL A ; Complement A-register
           (i.e., A = AAH)
      SJMP BACK ; Keep doing it

```

The above program will continuously send out to port 1, the alternating values 55H and AAH.

Example 4: A Program to get data from Port 1 and save it in registers R7 and R6.

```

MOV A, #0FFH ; A = FFH
MOV P1, A ; Write 1's to all the bits of port 1
and make it as an input port
MOV A, P1 ; Get data from port 1
MOV R7, A ; Save it in register R7

```

```

ACALL DELAY ; Wait
MOV A, P1    ; Get another data from port 1
MOV R6, A    ; Save it in register R6

```

3.12.3. Port 2 (P2)

A total of 8 pins of Port2 named as P2.0 to P2.7 are used as:

- Input port
(or)
- Output port
(or)
- High-order address bus (A8 to A15) for external memory.

Example 5: A Program to toggle all the bits of port 2 continuously.

```

MOV A, #55H ; A = 55H
BACK:      MOV P2, A ; Send 55H to port 2
          ACALL DELAY ; Wait
          CPL A ; Complement A-register
          (i.e., A =AAH)
          SJMP BACK ; Keep doing it

```

The above program will continuously send out to port 2, the alternating values of 55H and AAH.

Example 6: A Program to get a byte (Data) from port 2 and send it to port 1 continuously.

```

MOV A, #0FFH ; A = FFH
MOV P2, A ; Write 1's to all the bits of port 2
and make it as an input Port
BACK:      MOV A, P2 ; Get data from port 2
          MOV P1, A ; Send data to port 1
          SJMP BACK ; Keep doing it

```

3.12.3. Port 3

Port 0 occupies a total of 8 pins named as P0.0 to P0.7. These 8 numbers of pins are used:

- as input port
(or)
- as output port
(or)

- Alternate uses are:

***Note:** The ports P1, P2 and P3 do not require any pull-up resistors since they already have pull-up resistors internally.

3.13. Explain the term debugging a program.

Debugging is the process of finding and eliminating errors in a program. In the process of debugging, the program is tested thoroughly to find out the errors and to remove them. During debugging process the errors in program logic, machine codes and execution process are detected and eliminated.

The debugging process can be divided into two parts: static debugging and dynamic debugging. Static debugging is similar to visual inspection of a circuit board. It is done by paper and pencil check of a flowchart and machine code. Dynamic debugging involves observing the output or register contents, following the execution of each instruction or of a group of instructions.

3.13.1. Common Error occurred in Assembly Language Program

The assembly language program is translated in to machine codes before entering the program to microprocessor/ microcontroller kit. The following common errors are occurred in assembly language program.

1. Selecting a wrong code
2. Forgetting the second and third byte of an instruction
3. Specifying the wrong jump location
4. Not reversing the order of high and low bytes in a jump instruction
5. Writing memory addresses in decimal thus specifying wrong jump locations.

These common errors can be checked and corrected before entering the program in the memory of microprocessor/ microcontroller kit.

3.14. List the important steps in writing and trouble shooting a simple program.

- An assembly language program is a sequence of instructions written in mnemonics to perform a specific task. These instructions are selected from the instruction set of the microprocessor being used.
- To write a program, we need to divide a given problem into small steps and translate these steps into the operations the 8051 can perform.
- For example, the Z80 does not have an instruction that can multiply two' binary numbers, but it can add. Therefore, the multiplication problem can be written as a series of additions.
- After writing the instructions in mnemonics, you should translate them in to binary machine code; this process of translation is called assembling the code Quite often,

this process involves intermediate steps, such as translating mnemonics into Hex code and then into binary code.

- To execute a program, the binary code should be entered and stored in the R/W memory of a microcomputer so that the microprocessor can read and execute the binary instructions written in memory.
- In a single-board microcomputer the instructions are, generally, entered using a Hex keyboard. This is one of the reasons why we translate mnemonics into Hex code as an intermediate step rather than into binary code directly.
- When the Hex code is entered, the keyboard program, residing in the microcomputer system, translates the Hex code into binary code.

3.15. Explain the principles of single step and break point debugging techniques.

Dynamic debugging again can be divided into two common methods as single step control technique and break point technique.

3.15.1. Single step control technique

The process which involves the observing of the output, or register contents, following the execution of each instruction, is known as single step control technique of dynamic debugging. The single step control allows the user to execute the program one step at a time and check whether the intermediate results are correct or not. Hence this method is very slow and not suitable for lengthy programs. This method cannot find timing errors and errors in DMA or interrupt.

3.15.2. Break point technique

The process which involves the observing of output, or register contents, following the execution of a group of instructions, is known as break point technique of dynamic debugging. In the microcomputer, this location or point in the program by using RST software instructions.

Additional information

Introduction to Time Delay and Initial Count Calculations

8051 timers can be used to generate the required time delay. Timers 0 or 1 are used to count the clock pulses.

$$\text{Timer clock frequency} = \frac{\text{Crystal frequency}}{12} \quad \dots (1)$$

$$\text{Timer clock period} = \frac{12}{\text{Crystal frequency}} \quad \dots (2)$$

a) Time Delay

The time delay introduced between starting and stopping of the timer is calculated using the following formula:

$$\text{Time delay} = [\text{Maximum value} - \text{Initial count} + 1] \times \text{Timer Clock period}$$

or

$$\text{Time delay} = \left([\text{Maximum value} - \text{Initial count} + 1] \times \frac{12}{\text{Crystal frequency}} \right)$$

b) Initial Count

The initial count value is computed as follows:

$$\text{Initial count} = (\text{Maximum value} + 1) - (\text{Time delay} \times \text{Timer clock frequency})$$

or

$$\text{Initial count} = (\text{Maximum value} + 1) - \left(\frac{\text{Time delay} \times \text{Crystal frequency}}{12} \right)$$

c) Maximum Count Value

The maximum value depends on the mode of the timer as shown in table.

Mode	Timer Size	Maximum Count Value	
		Hexadecimal	Decimal
Mode 0	13-bit	1FFFH	8191D
Mode 1	16-bit	FFFFH	65535D
Mode 2	8-bit	FFH	255D

- 8051 timers can be used to generate the required time delay.
- In order to generate certain time delay timer (0 or 1) register should be loaded with an appropriate number. Then run the timer and it starts counts up and reaches to its maximum value.
- While reaching the maximum value timer rolls over and stops the counting process. The following procedure to be adopted while finding the number to be loaded into the timer registers.

Procedure for Time Delay Generation:

Step 1: Initialize TMOD for the required Timer (0 or 1) and mode (0 or 1 or 2)

Step 2: Load the initial count value in registers TL and TH (of timer 0 or 1)

Step 3: Start the timer (SET TR1 or TR0 bit of TCON register)

Step 4: Wait until the timer flag (TF1 or TF0 of TCON) is set. (Remember TFX=1 when timer X overflows the maximum value)

Step 5: Stop the timer (TR1 or TR0 is made zero)

Step 6: Clear TF flag

Step 7: Repeat from step 2 for the next delay.

The initial count value is computed as shown below:

3.16. Write simple programs to setup time delay using counter & a single register.

- For example, if the microcontroller clock frequency is 12 MHz.
- The time period of one T state = $\left(\frac{1}{12 \times 10^6}\right)$ second .
- The time period of one machine cycle = $12 \times T_{\text{state}} = 1$ microsecond.

1. Delay program with one loop

Mnemonics	Number of machine cycles	Number of times executed	Total number of machine cycles
DELAY MOV R0, #FFH	1	1	1
LOOP DJNZ R0, LOOP	2	255D	510D
RET	3	1	2D
Total			513D

- Total number of machine cycles needed to execute this delay subroutine = 513 D.
- Total time period produced by this delay subroutine = $513 \times 1 \mu\text{s} = 513 \mu\text{s}$.
- This delay program can produce a maximum delay period of 513 μs .

2. Delay program with two loops

Mnemonics	Number of machine cycles	Number of times executed	Total number of machine cycles
DELAY MOV R0, #FFH	1	1	1
LOOP MOV R1, #FFH	1	255	255
LOOP1 DJNZ R1, LOOP1	2	255 x 255	130050
LOOP2 DJNZ R0, LOOP2	2	255	510
RET	2	1	2
Total			130818D

- Total number of machine cycles needed to execute this delay subroutine = 130818 D.
- Total time period produced by this delay subroutine $130818 \times 1\mu\text{s} = 130818\mu\text{s}$.
- The delay period of delay subroutine can be increased to increase the number of loops. For changing the initial value loaded in the registers, we can able to change the delay period of the time delay subroutines.

Program 1: Write a 8051 assembly language program to generate a delay of 5ms using timer1 in mode 1 with XTAL frequency of 11.0592 MHz.

Solution: In mode 1, the timer counts up till FFFFH=65535D (maximum value for the 16-bit timer). Then it rolls over from FFFFH to 0000H and sets the flag bit TF=1.

Step 1: The timer mode register TMOD is programmed as

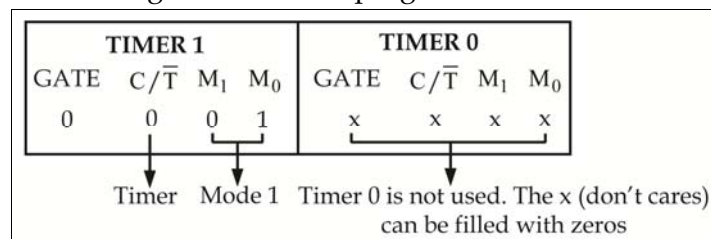


Fig.3.16

Here TMOD = 10H \Rightarrow Timer 1 in mode 1

Step 2: The initial count value (in hexadecimal) to be loaded into TL1 and TH1 is computed as,

$$\text{Initial count} = (\text{Maximum value} + 1) - (\text{Time delay} \times \text{Timer clock frequency})$$

$$= (65535 + 1) - \frac{5\text{m} \times 11.059\text{M}}{12} = 65536 - 4608 = 60,928\text{D} = \text{EE00H}$$

The initial value (16-bit) should be loaded into the 16-bit timer register T1 as TH1 = 0EEH (MSB) and TL1 = 00H (LSB).

Assembly Program:

	MOV TMOD, #10H	Timer 1, mode 1
AGAIN:	MOV TL1, #00H	Initial value in TL & TH
	MOV TH1, #0EEH	
	SETB TR1	START Timer 1
WAIT:	JNB TF1, WAIT	wait until timer 1 counts upto maximum i.e., jump if TF1 = 0
	CLR TR1	stop timer 1
	CLR TF1	clear TF1 so that it can be set again
	SJMP AGAIN	by the timer1 roll over

Problem: Find the initial count value to be loaded into TL1 and TH1 to generate 5ms time delay

The following procedure to be adopted while finding the number to be loaded into the timer registers.

1. Assume the XTAL frequency of the 8051 (f).
2. Calculate the clock period ($t = \frac{1}{f}$).
3. Divide the desired time delay by clock period. Result subtracted from maximum count i.e.; for 8-bit : 256 & 16-bit: 65 536.
4. Convert the result (obtained in step-3) into hexa decimal.
5. Load the hexa decimal number in timer register (TH & TL).

Solution:

$$\text{Clock period}(t) = \frac{1}{11.0592 \text{ MHz}} = 1.085 \text{ microsec.}$$

$$\text{Number of clocks} = \frac{5 \text{ millisecc}}{1.085 \text{ microsec}} = 4608$$

Number to be loaded in to timer - 0: 65536-4608=60,928D=EE00H.

3.17. Calculate the time delay in the program given the clock frequency.

Program 2: Find the time delay generated by timer 1 for the following program.

```

MOV TMOD, #10H
AGAIN: MOV TL1, #00H
MOV TH1, #0EEH
SETB TR1
WAIT:  JNB TF1, WAIT
CLR TR1
CLR TF1
SJMP AGAIN

```

Solution:

- The timer 1 counts from the **initial value** EE00H to the **maximum value** (FFFFH).
- TMOD = 10H, $C/\bar{T} = 0$, implies timer 1 is in timer mode and is counting the internal clock pulses.

$$\begin{aligned} \text{Timer clock period} &= \frac{12}{\text{crystal frequency}} \\ &= \frac{12}{11.0592 \text{ MHz}} = 1.085 \mu\text{s} \end{aligned}$$

- Total time taken to roll over, that is set the TF1, is

$$\text{Time delay} = [\text{Maximum value} - \text{Initial count} + 1] \times \text{Timer clock period}$$

$$\begin{aligned} \therefore \text{Time delay} &= (\text{FFFF} - \text{EE00} + 1) \times 1.085 \mu\text{s} = 1200\text{H} \times 1.085 \mu\text{s} \\ &= 4608 \text{d} \times 1.085 \mu = 4996.425 \mu\text{s} = 4.996 \text{ms} \end{aligned}$$

Note: + 1 is added in the above formula because of the extra clock pulse needed when the timer rolls over from FFFF to 0 and raises the TF flag.

Program 3: Find the time delay obtained for the following program.

Solution:

	Instruction	Machine Cycles
	MOV TMOD, #10H	2
AGAIN:	MOV TL1, #0EAH	2
	MOV TH1, #0FFH	2
	SETB TR1	1
WAIT:	JNB TF1, WAIT	22 (depends on the initial value in the timer = max. value - init. Value + 1)
	CLR TR1	1 (FFFF - FFEA + 1)
	CLR TF1	1
	SJMP AGAIN	2
	Total	33

$$\text{Total delay} = \text{No. of machine cycles} \times \frac{12}{\text{Crtal frequency}}$$

$$\text{Total delay} = 33 \times \frac{12}{11.0592 \text{M}} = 33 \times 1.085 \mu = 35.805 \mu\text{s}$$

Note:

$$\begin{aligned} \text{Maximum value} &\rightarrow \text{FFFFH} \\ \text{Intially value} &\rightarrow - \text{FFEAH} \\ &+ 1 \\ &16\text{H} = 22\text{D} \end{aligned}$$

CHAPTER 4

Interfacing Simple I/O Devices

OBJECTIVES

- 4.1 Describe the Interfacing of push button switches and LEDs
- 4.2 Write instructions to access data for the above
- 4.3 Describe the Seven segment display interface – static and dynamic types
- 4.4 List reasons for the popularity of LCDs
- 4.5 Describe the functions of pins of LCD
- 4.6 List instruction command code for programming a LCD
- 4.7 Explain Interfacing LCD to 8051
- 4.8 Program LCD in assembly language
- 4.9 Explain the basic operations of keyboard
- 4.10 Explain key press and detection mechanisms
- 4.11 Describe key bouncing problem and de-bouncing solutions
- 4.12 Explain Interfacing of a 4x4 Matrix Key Board.
- 4.13 Write a program to access key code from matrix key board

4.0. Introduction to push button switches

Switch is an electrical component that can break an electrical circuit, interrupting the current or diverting it from one conductor to another. A switch can act as an input device to the microcontroller unit (MCU). Switches come in many different shapes and sizes. The main idea or function is to either enable or disable something by simply turning it on/off. Thus it acts as an input device that users can use to control certain parts of the system.

There are different conventional methods available to interface the switch to the microcontroller.

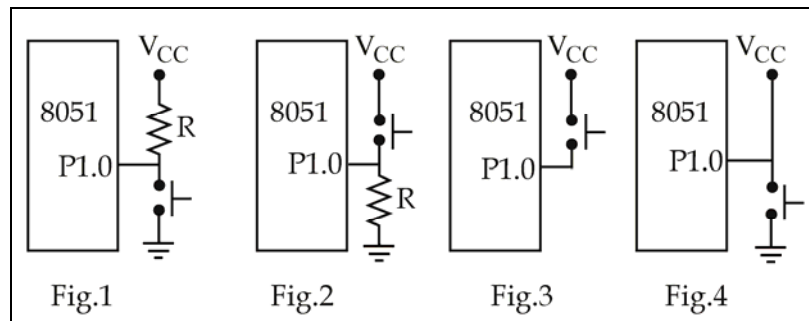


Fig. 4.0

From the above diagrams, fig.1 and fig.2 are good and fair, but fig.3 and fig.4 not preferable because, the microcontroller draws more current if we are connecting the switch to the microcontroller as per fig.3, and the microcontroller shorted when the switch is closed as per fig.4.

4.1. Explain the Interfacing of push button switches and LEDs.

1. Interfacing of pushbutton switches to 8051

- The fig. 4.1 shows the interfacing of switches to microcontroller.
- Switch (pushbutton) can be operated in any of two states (ON or OFF).
- The switch is used as input to a microcontroller.
- In the fig. 4.1(a) the switch 1 is connected to the P0.0 of the microcontroller through a pull-up resistor (10K). If the SW1 is open, the P0.0 will be driven to high and read as logic 1. When the SW1 is closed, P0.0 is shorted to ground and P0.0 pulled to 0V and P0.0 will read as logic 0.
- As per fig. 4.1(b), if SW2 is open, the P0.1 is pulled to ground and the P0.1 will be read as logic 0. If SW2 is closed, the Vcc is connected to the P0.1 and the P0.1 will be read as logic 1.

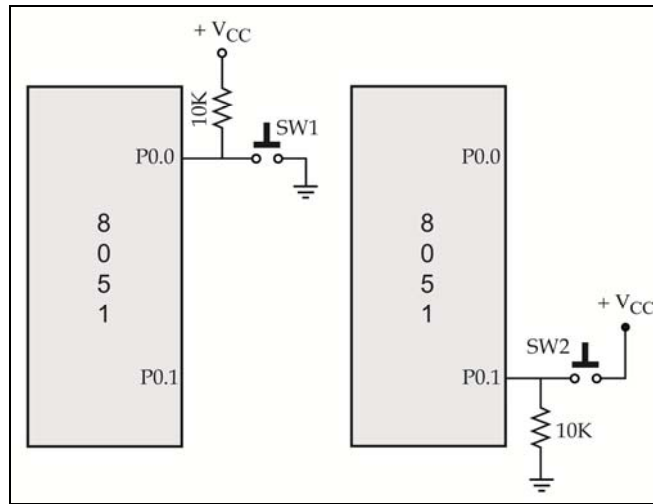


Fig. 4.1(a)

Fig. 4.1(b)

2. Interfacing of LED to 8051

- The fig. 4.1 shows interfacing of LED to 8051.
- LED can be connected directly to the microcontroller or through a buffer (current driver).
- The anode (A) terminal of the LED connected to the Vcc through a suitable resistor and cathode (K) terminal of the LED connected to the port pin.
- The LED glows when port bit is reset. The microcontroller lacks 20-30mA driving capability; hence a buffer or current driver can be connected in series with the LED to the port pin.

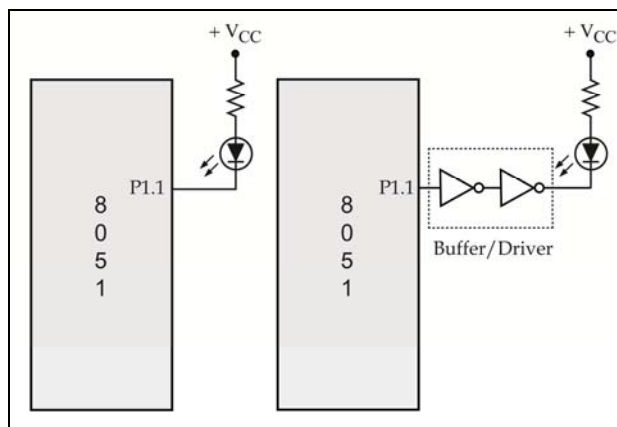


Fig. 4.1(a)

Fig. 4.1(b)

Code for run a LED

- Program for turn ON/OFF an LED is given below.

```

ORG 0000h

LED_RUN: CLR P1.1    : LED glows by clear the output port bit (P1.1)

LED_OFF: SETB P1.1  : LED turned off by set the output port bit(P1.1)

HLT:      SJMP HLT

END

```

3. Interfacing of push button switches and LEDs

Now, if in any application, it is required to light an LED when a pushbutton is pressed, then a circuit shown in fig. may serve the purpose. The operation is very simple. Under normal conditions, when the pushbutton is not pressed, the port pin is internally pulled high through the weak internal pull-up. The LED will not light, because its cathode is also high. However, when the push button is pressed, the port pin is grounded; current will flow through the LED and the current limiting resistor that will cause the LED to glow. Normally, a resistance of 220–330Ω is suitable as a current limiting resistor.

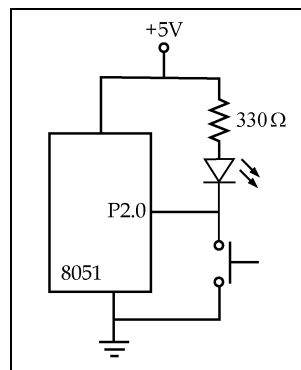


Fig. 4.1(c)

4.2. Explain the interfacing of Seven segment display.

There are two types SSDs available.

1. Common cathodes i.e. the cathode of all LEDs are given as a common pin. In this case the anode is connected to the port pins. For method requires port pins to source large current. But 8051 cannot source current beyond 2mA.
2. Common anodes i.e. the anode of all the LEDs are given as a common pin. In this case the cathode is connected to the port pins. Hence port pins have to sink current of 20 mA.

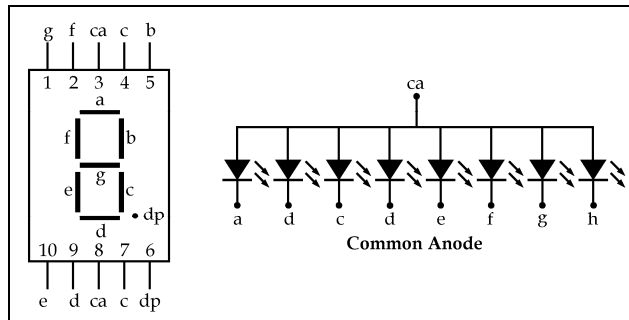


Fig. 4.2(a)

Hence we use common anode, SSD, and connect it to 8051 as shown in fig. 4.2(a).

4.2.1. Interfacing Common anode SSD to 8051

- Here, common anode seven segment display is used to interface with 8051.
- Pins 'a' to 'h' of the display are connected to the port 1 of microcontroller and common pin is connected to +V_{CC}.
- Each segment of the display is connected to the port 1 pins through a suitable resistor (220E).
- To display the different digits from 0 to 9 on the seven segment display, we need different logic combinations to the corresponding LED segments.

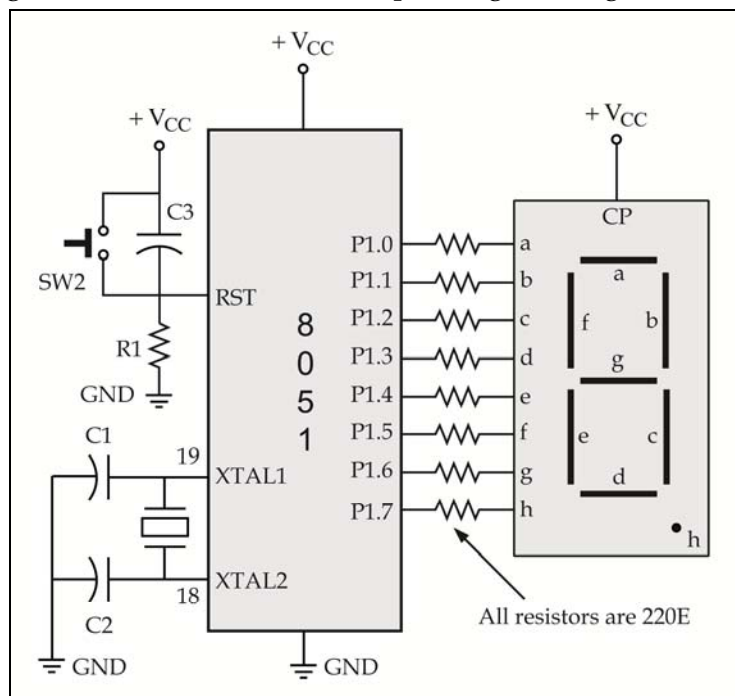


Fig. 4.2.1(a)

1. Hex code for the decimals from 0 to 9 for common Anode

The table illustrates the binary and hexadecimal data given to the pins of SSD, to display different digits.

Decimal Digit	h	g	f	e	d	c	b	a	Hex code
0	1	1	0	0	0	0	0	0	C0H
1	1	1	1	1	1	0	0	1	F9H
2	1	0	1	0	0	1	0	0	A4H
3	1	0	1	1	0	0	0	0	B0H
4	1	0	0	1	1	0	0	1	99H
5	1	0	0	1	0	0	1	0	92H
6	1	0	0	0	0	0	1	0	82H
7	1	1	1	1	1	0	0	0	F8H
8	1	0	0	0	0	0	0	0	80h
9	1	0	0	1	0	0	0	0	90H

2. Program to display digit '9'

```

ORG 0000H
CLR P1.0    : Turn on the segment 'a'
CLR P1.1    : Turn on the segment 'b'
CLR P1.2    : Turn on the segment 'c'
CLR P1.3    : Turn on the segment 'd'
SETB P1.4   : Turn off the segment 'e'
CLR P1.5    : Turn on the segment 'f'
CLR P1.6    : Turn on the segment 'g'
HLT: SJMP HLT : Temporary stop the program
END

```

OR

Single Step Program to display the digit '9':

```

ORG 0000H
MOV P1,#90H
HLT: SJMP HLT
END

```

4.3. Mention the reasons for the popularity of LCDs

LCD is a flat display and it is an electronic visual display. LCD uses the light modulating properties of liquid crystals. These crystals do not emit light directly. CRT screen, LEDs and seven segment displays are replaced by the LCDs because of

- Low cost prices of LCDs.
- LCDs are small in size compare with CRT displays.
- LCDs can display numerical, characters, symbols and other graphics.
- Refreshing is so easy.
- Easy to program for characters and graphics.
- Easy to interface with CPU.
- It consumes less electric power therefore they can operate with batteries.
- LCD screen is more energy efficient than all other display screens i.e. CRT screens.
- LCDs are available in a wider range of screen sizes.
- LCDs have wide range of applications such as computer monitors, televisions, instruments panels and in other consumer products such DVD players, clocks etc.

4.4. Explain the functions of pins of LCD

- The pin configuration of 16 by 2 LCD is shown in fig. 4.4(c).

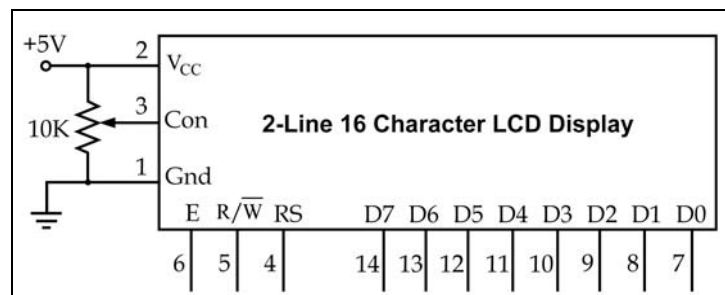


Fig. 4.4(c).

- The functions of the pins of LCD are listed in the table.

Pin	Symbol	I/O	Description
1.	V_{SS}	-	Ground
2.	V_{CC}	-	+5V power supply
3.	V_{EE}	-	Used for controlling LCD contrast
4.	RS	I	Register select input is used to select either of the two

			available registers in the module: Data register or command register. When RS=0 Data register is selected when RS=1 command register is selected.
5.	R/\overline{W}	I	Allows the user to write information in the LCD or read information from it. For reading $R/W = 1$ and for writing $R/\overline{W} = 0$.
6.	E	I	This pin is used by LCD to latch information available at its data pins.
7-14	$DB_0 - DB_7$	I/O	This 8-bit data bus is used to send information to the LCD or read the contents of the internal registers of the LCD.

Additional Information

Functions of pins of LCD

RS: Registers select

- There are two registers of LCD viz. instruction command code register and data register. The RS pin is used to select one of these registers.
- If RS=0 the instruction command code register is selected and if RS=1 the data register is selected, allowing user to send data to be displayed on the LCD.

R/\overline{W} : Read/Write

- R/\overline{W} Pin is used to read or write to LCD.
- If $R/\overline{W} = 0$ the user can write information to the LCD and if $R/\overline{W} = 1$ the user can read information.

Enable: E (it can also be denoted by EN)

- The enable pin E, is used to latch the data into the data or command register. When data is supplied, a high-to-low (negative edge) is required for LCD latch the data.

D0 -D7 or DBO - DB7

- The data pin D0 - D7 are used to send information to the LCD or read the contents of LCD internal registers.
- To display letters and numbers we send ASCII codes for letters A -Z, a -z and numbers 0-9 while making RS=1.

- The ASCII code that is to be displayed is of 8bits. It is send to the LCD in either nibbles or bytes i.e. 4 or 8 bits at a time.
- The two primary modes of operation to send parallel data are 4 or 8 bits.
- If four bit mode is used two nibbles of data are sent to do an 8 bit transfer. The “E” clock is used to initiate the data transfer. Atleast 6 I/O pins must be available for 4 bit mode.
- In 8 bit mode atleast 10 I/O pins must be used. This mode is used when application needs speed.

V_{CC} , V_{SS} and V_{EE}

V_{CC} – Provides +5V supply

V_{SS} – Ground

V_{EE} is used for controlling LCD contrast.

4.5. List instruction command code for programming a LCD

16x2 LCD module has a group of designed command instructions. Each command code has particular task. Commonly used commands are listed below.

Hexa code	Instruction/Description
01	Clear display screen
02	Return cursor to home
04	Decrement cursor (Shift cursor to left)
06	Increment cursor (Shift cursor to right)
05	Shift display right
07	Shift display left
08	Display OFF, cursor OFF
0A	Display OFF, cursor ON
0C	Display ON, cursor OFF
0E	Display ON, cursor ON
0F	Display ON, cursor Blinking
10	Shift cursor position to left
14	Shift cursor position to right

18	Shift the entire display to the left
1C	Shift the entire display to the right
80	Force cursor to the beginning of first line
C0	Force cursor to the beginning of the second line
38	2 line and 5 X 7 matrix

4.6. Explain Interfacing LCD to 8051

LCD displays are dot matrix displays where displays elements in the form of dots are arranged as row and columns. Common matrix sizes are 5×7 or 7×9 . By switching on some and off some dots, any character can be formed. Inside the LCD there is a ROM which stores the required dot matrix code (for on/off of the dots) for the corresponding ASCII input. Using the dot matrices text, pictogram and graphic displays can be displayed.

4.6.1 Interfacing 16 x 2 LCD to 8051

Fig. 4.6.1(a) shows the interfacing of a 16 x 2 line LCD module with the microcontroller 8051.

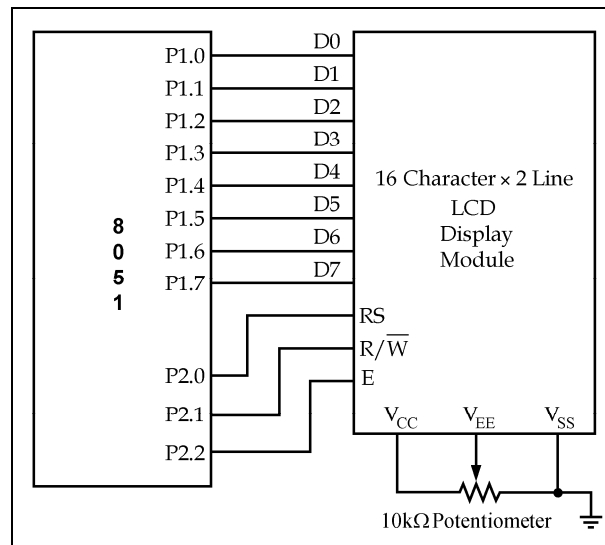


Fig. 4.6.1(a)

- Pins P1.0 to P1.7 of the microcontroller is connected to the D0 to D7 pins of LCD module, through these pins the data transfer to the LCD module.
- Pins P2.0, P2.1, and P2.2 are connected to the RS, R/\overline{W} and E respectively, through these pins control signals are transferred to the LCD module.

- A potentiometer is connected between 1, 2 and 3rd pins of the LCD. It is used to adjust the contrast of the LCD module.

To display data on the LCD the following steps are executed:

- Initialized the LCD with a set of command words. The command words are sent on the D0-D7 data lines with RS=0, R/W=0 and a high-to-low pulse on the E pin.
- The ASCII value of the character to be displayed is sent on the D0-D7 data lines with RS=1, R/W=0 and a high-to-low pulse on the E pin.

Table. lists some commonly used commands in LCD. In the LCD, the data can be displayed at any location. The command words 80H sent to the LCD, positions the cursor at position 0 on line 1. Similarly 86H, command word positions the cursor at position 6 on line 1. The data byte sent to the LCD after the command word C0H is sent, is displayed at position 0, line2.

LCD command	Function
01H	Clear LCD display
06H	Shift cursor right after every data write to display
0EH	LCD on, cursor ON
0CH	LCD on, cursor OFF
0FH	LCD on, blinking cursor
80H	Cursor in position '0' line 1
C0H	Cursor in position '0' line 2
38H	2 line 5×7 matrix display

4.7. Explain assembly language Program for interfacing LCD.

Write an 8051 assembly language program (ALP) to display HELLO on the LCD connected to the 8051

Solution: Consider that the LCD is interfaced to the 8051 as shown in Fig. 4.7.

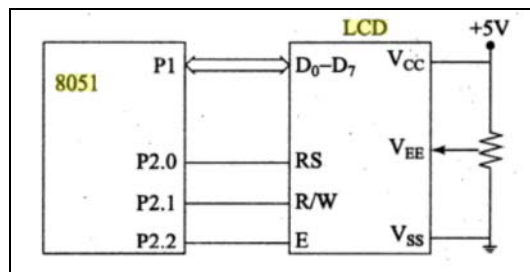


Fig. 4.7

Algorithm:

1. Initialize the LCD using the init LCD subroutine
2. Send the ASCII value of 'H' to port P1 and call datawrt subroutine
3. Repeat step 2 for 'E', 'L', 'L' and 'O'
4. Wait here.

Init LCD subroutine:

1. Send command 38H to port P1 and call cmdwrt subroutine
2. Repeat step 1 for commands 0EH, 01, 06 and 80H (refer Table 10.3 for details)

Cmdwrt subroutine (to latch data on P1 into command register):

1. Make RS (i.e., P2.0) and R/W (i.e., P2.1) pins low (for command & write)
2. Make enable pin E (P2.2) high, for a lms delay.
3. Make enable pin low
4. Call delay of 250ms
5. Return to calling program

Datawrt subroutine (to latch ASCII data on P1 into data register (RAM on LCD) for display on LCD)

1. Make RS=1 (data register) and R/W=0 (write function)
2. Make E=1 for 1ms approximately
3. Make E=0
4. Call 250ms delay
5. Return to calling program.

ALP for LCD interfacing:

	ORG	0H	;Reset vector
	SJMP	START	
	ORG	30H	;start main program here
START:	ACALL	INITLCD	;initialize LCD
	MOV	A, #'H'	;ASCII value of H
	ACALL	DATAWRT	; display on LCD
	MOV	A, #'E'	
	ACALL	DATAWRT	
	MOV	A, #'L'	
	ACALL	DATAWRT	
	MOV	A, #'L'	
	ACALL	DATAWRT	
	MOV	A, #'O'	

HERE:	ACALL DATAWRT SJMP HERE	;WAIT HERE
-------	----------------------------	------------

DATAWRT Subroutine:

DATAWRT:	MOV P1, A SETB P2.0 CLR P2.1 SETB P2.2 ACALL DELAY1 CLR P2.2 ACALL DELAY RET	; send the data on P1 ; RS=1 for data ; R/W=0 for write ; E=1 enable high ; small delay ; E=0 enable low ; high delay
----------	---	---

CMDWRT Subroutine:

CMDWRT:	MOV P1, A CLR P2.0 CLR P2.1 SETB P2.2 ACALL DELAY1 CLR P2.2 ACALL DELAY RET	; send command on P1 ; RS=0 for data ; R/W=0 for write to LCD ; E=1 ; small delay ; E=0 ; high delay
---------	--	--

INITLCD Subroutine:

INITLCD:	MOV A, #38H ACALL CMDWRT MOV A, #0EH ACALL CMDWRT MOV A, #01 ACALL CMDWRT MOV A, #06; ACALL CMDWRT MOV A, #80H ACALL CMDWRT RET	;2 lines, 5 x 7 ; display & cursor ON ; clear LCD ; shift cursor right ; line 1 display
----------	---	---

Delay:

DELAY:	MOV	R3, #250	; high delay
LOOP1:	MOV	R4, #255	
LOOP2:	DJNZ	R4, LPPP2	
	DJNZ	R3, LOOP1	
	RET		

Delay 1:

DELAY1:	MOV	R3, #10	; small delay
LOOP3:	MOV	R4, #255	
LOOP4:	DJNZ	R4, LOOP4	
	DJNZ	R3, LOOP3	
	RET		
	END		

NOTE: DATAWRT and CMDWRT subroutines are almost same except for RS, which is low (RS=0) for command & high (RS=1) for data.

4.8. Explain the basic operations of keyboard

A common method of entering programs into a microcomputer is through a keyboard which consists of a set of switches. Basically each switch will have two normally open metal contacts. These two contacts can be shorted by a metal plate supported by a spring as shown in Fig.4.8 (a) On pressing the key, the metal plate will short the contacts and on releasing the key, again the contacts will be open. The processor has to perform the following three major task to get a meaningful data from a keyboard.

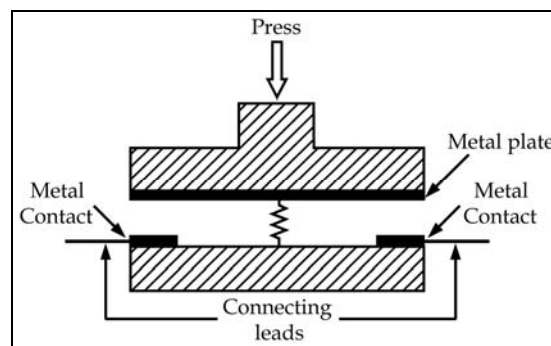


Fig. 4.8(a)

- The input keyboard is composed of a set of labeled push button switches. Each switch makes electrical contact when pressed.

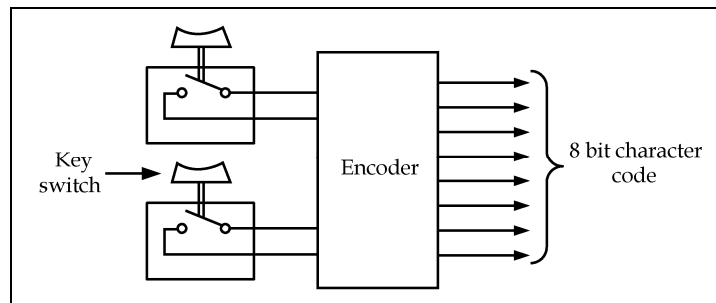


Fig.4.8 (b)

- Fig. (b) shows the general operation of a keyboard.
- The aim of this mechanism is to generate and transmit a code each time key is pressed.
Note: The mechanism should send one and only proper code, when the key is pressed.
- The input keyboard is composed of a set of a labeled pushbutton switches. Each switch makes electrical contact when pressed.

4.9. Explain key press and detection mechanisms

The steps required to identify the pressed key are,

1. To identify if any key is pressed or not.
 - All the column lines are made zero by sending low on all the output lines. i.e. all the keys in the keyboard matrix are activated.
 - Read the status of rows i.e. return lines. If the status of all lines logic high, the key is not pressed. Otherwise if the status of all lines is logic low, the key is pressed.
2. Debouncing the key. (Using software debouncing as explained earlier)
3. Identifying the pressed key.
 - Activate the keys from one column by making one column line zero.
 - Read the status of returns lines. The zero or any return line indicates that key is pressed.
 - Activate the keys from next column and repeat steps (b) and (c) for all the columns.

Key bouncing problem and de-bouncing solutions

1. Key bouncing

For interfacing keyboard to the microprocessors/ microcontroller based systems, usually push button keys are used. These push button keys when pressed, bounces a few times (10 - 20 ms), closing and opening the contacts before providing a steady reading.

Normally a key is interfaced to a microcontroller through a port line as shown in fig. 4.9(a). the port line is pulled up to HIGH level when the key is not pressed and contact is open. When the key is pressed, the port line is connected to Low level through the key. However, because of the key-bounce problem, the port line eventually attains ground level, after making several transitions between ground and V_{cc} levels as shown in fig.4.9 (b) Similar situation arises when the key is released.

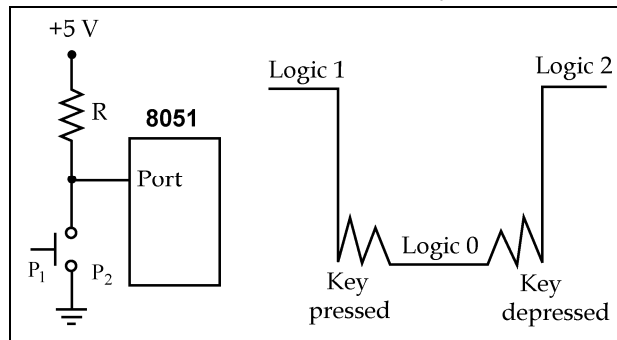


Fig.4.9 (a)

Do You Know?

1. The problem of a key bounce arises when a pushbutton key is pressed or released.
2. Reading taken during bouncing period may be faulty. Therefore, microprocess/microcontroller must wait until the key reach to a steady state; this is known as key debounce.

2. De-bouncing methods

In case of a push button key, the metal contact bounces few times; hence the voltage across the switch fluctuates and generates spikes in the signal. Therefore, it is necessary to debounce the mechanical switches. The key debouncing is done through hardware and software.

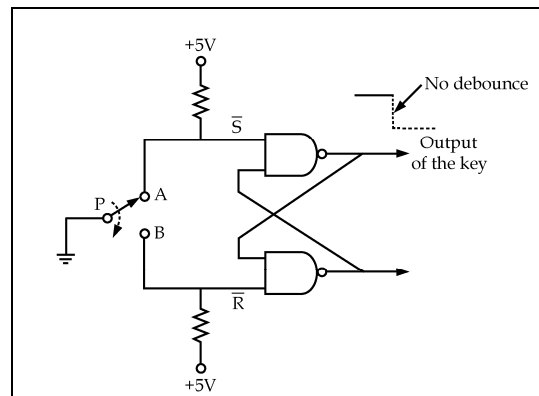


Fig. 4.9(c)

a) Hardware key Debouncing

- It is implemented by using flip-flop or latch. Fig.4.9 (c) shows a circuit diagram of hardware key debouncing.

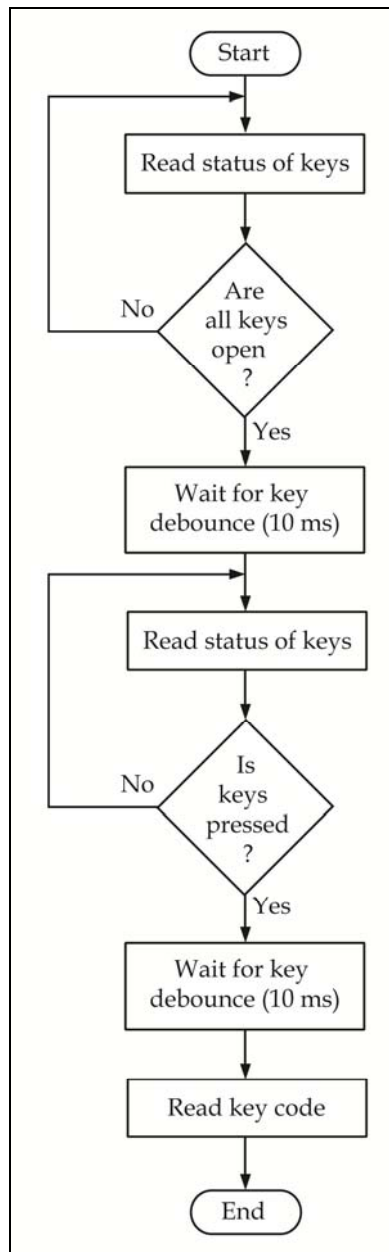


Fig.4.9 (d)

- When the switch is connected to A, the output of the latch goes high. When the key makes contact with B, the output changes from logic 1 to logic 0. The wiper bounces many times on contact B, but the output does not fluctuate between logic 1 and logic 0. When the wiper is not connected either to A or B, the output of the latch remains constant.

b) Software key Debouncing

- In the software technique the microcontroller waits for 20 ms before it accepts the key as an input. If after 20ms the key is pressed the key is accepted by microcontroller. The process of software key debouncing is as shown in fig.4.9 (d).

4.10. Explain Interfacing a 4 X 4 Matrix Key Board.

A common method of entering programs into a microcomputer is through a keyboard. The microprocessor/ microcontroller has to perform the following three major tasks to get a meaningful data from a keyboard.

1. Sense a key actuation
2. Debounce the key
3. Decode the key

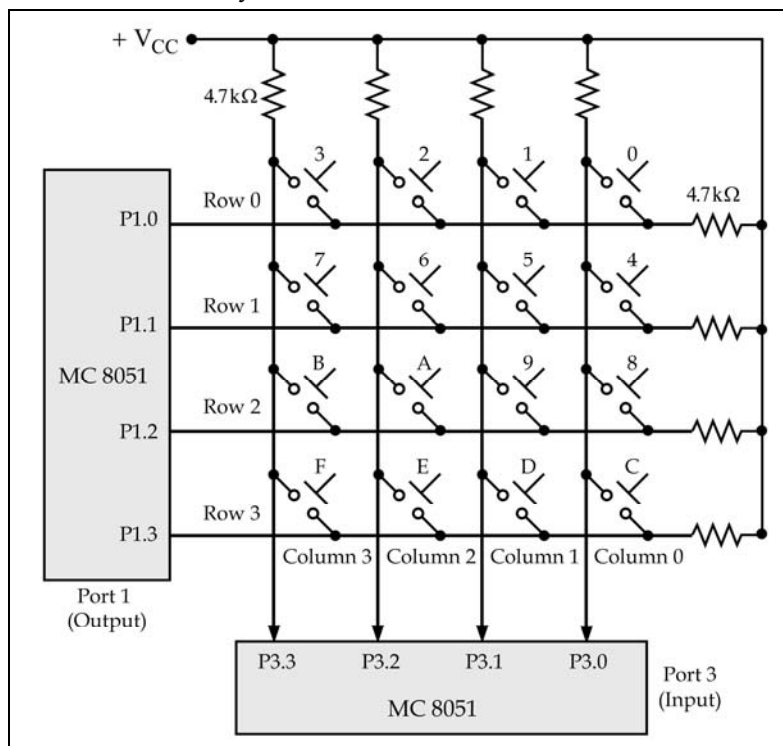


Fig. 4.10(a)

The three major tasks mentioned above can be performed by software. Matrix keyboards are scanned by bringing each X row low (or high) in sequence and detecting a Y column low (or high) to identify each key in the matrix X-Y scanning can be done by using dedicated keyboard circuitry or by using microcontroller ports under program control.

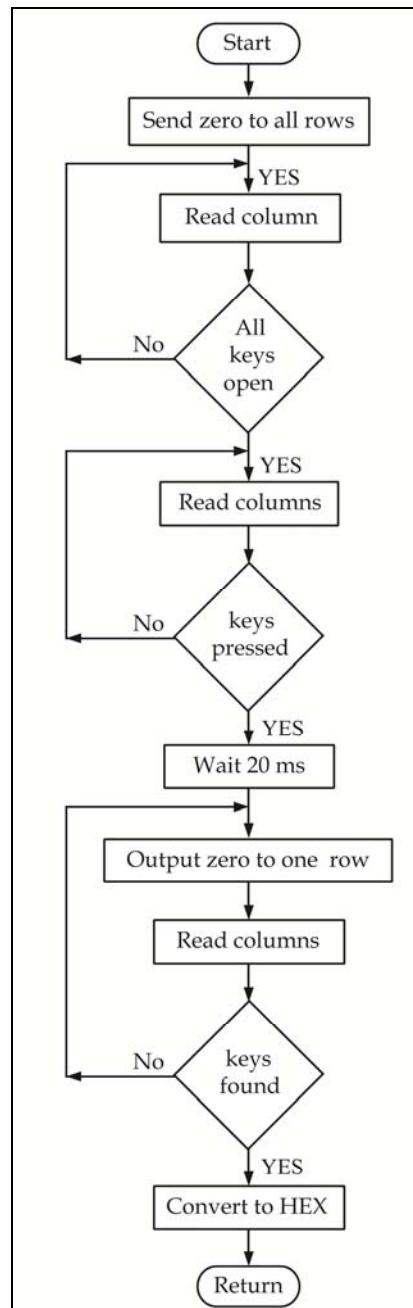


Fig. 4.10(b)

Consider a simple matrix keyboard in which the keys are arranged in rows and columns are shown in the fig. 4.10(a).

The rows are connected to port 1 lines of 8051 and the columns are connected to port 3 lines of 8051 also. The rows and columns are normally tied 'high'. At the

intersection of a row and column. A key is placed such that pressing a key will short circuit the row and the column.

A key actuation is sensed by sending a 'low' to each row once at a time via port 1. The columns are then read via port 3 to see whether any of the normally high columns is pulled low by a key actuation. For finding the key actuation, the rows can be checked individually to determine the row in which key is down. The row and column code in which the key is pressed can thus be found.

The next step is to debounce the key. Normally the key bounces, When it is pressed or released. When this bounce occurs, it may appear to the microcontroller that the same key has been actuated several times instead of just one. The problem can be eliminated by reading the keyboard after 20 ms and then verifying to see if it is still down. If it is, then the key actuation is valid. This process is called key debouncing.

The next step is to translate the row and column code into its equivalent hexadecimal code or ASCII code. This can be easily accomplished by a program. The flow chart for the software required for the keyboard interfacing with microcontroller is shown in the fig. 4.10(b).

In keyboard interfacing there are two methods of handling multiple key press. They are two key lock out and N key rollover. The two key lock out takes into account only one key pressed. An additional key pressed and released does not generate any codes. The N key rollover will detect all the keys pressed in the order of entry and generates corresponding key code.

The key board interfacing using microcontroller through ports is that most of the controller time is utilized (or wasted) in keyboard scanning and debouncing.

4.10. A program to access key code matrix key board

Assembly language program for detection and identification of key activation
; keyboard subroutine. This program sends the ASCII code
; for pressed key to P0.1
; P1.0-P1.3 connected to rows P2.0-P2.3 connected to columns

	MOV P2,#0FFH	; make P2 an input port.
K1:	MOV P1,#0	; ground all rows at once.
	MOV A, P2	; read all col. (ensure all keys open).
	ANL A, 00001111B	; masked unused bits.
	CINE A,#00001111B,K1	; check till all keys released
K2:	ACALL DELAY	; call 20 msec delay
	MOV A, P2	; see if any key is pressed

	ANL A, #00001111B	;mask unused bits
	CJNE A, # 00001111B, OVER	; key pressed, await closure
	SJMP K2	; check till key pressed
OVER:	ACALL, DELAY	; wait 20 msec debounce time
	MOV A, P2	; Check key closure
	ANL A, #00001111B	;mask unused bits
	CJNE A, #00001111B, OVER1	;key pressed find row
	SJMP K2	;if none, keep polling
OVER1	MOV P1, #11111110B	;ground row 0
	MOV A, P2	; read all columns
	ANL A,#00001111B	;mask unused bits
	CJNE A1,#00001111B, ROW_0	;key row 0, find the col.
	MOV P1,#11111101B	; ground row1
	MOV A, P2	; read all columns
	ANL A, #00001111B	;mask unused bits
	CJNE A, #00001111B, ROW_1	;keyrow 1, find the col.
	MOV P1,#11111011B	; ground row2
	MOV A, P2	;read all columns.
	ANL A, # 00001111B	;make unused bits.
	CJNE A, #00001111B, ROW_2	; keyrow 2, find the col.
	MOV P1, #1111011B	; ground row 3
	MOV A, P2	;read all cloumns.
	ANL A, #00001111B	;mask unused bits
	CJNE A,#00001111B, ROW_3	;keyrow 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 col. Key belongs to
ROW_1:	MOV DPTR, # KCODE1	; set DPTR= start of row 1
	SJMP FIND	;find col. Key belongs to
ROW_2:	MOV DPTR, # KCODE2	; set DPRT= Start of row 2
	SJMP FIND	; find col. Key belongs to

ROW_3:	MOV DPTR, # KCODE 3	; set DPTR=start of row 3
FIND:	RRC A	: see if any CY bit 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).
	MOVC A,@A+DPTR	; get ASCII code from table.
	MOV P0, A	Display pressed key.
	LJMP K1	
; ASCII	LOOK-UP TABLE FOR EACH ROW	
	ORG 300H	
KCODE0:	DB '0', '1', '2', '3'	; ROW 0
KCODE1:	DB '4', '5', '6', '7'	; ROW 1
KCODE2:	DB '8', '9', 'A', 'B'	; ROW 2
KCODE3:	DB 'C', 'D', 'E', 'F'	; ROW 3
	END	

CHAPTER 5

Programming 8051 Timers, Serial port & Simple Applications

OBJECTIVES

- 5.1 Explain how to program 8051 timers to create time delays.
- 5.2 Write programs to generate a square wave of given frequency and duty cycle using timer.
- 5.3 Explain how to use an 8051 timer as an event counter.
- 5.4 Write a program to count the number of events using timer.
- 5.5 Explain how to program 8051 serial port to transmit and receive serial data.
- 5.6 Write a program to transmit a message serially using serial port.
- 5.7 Write a program to receive a message serially and store it in memory.
- 5.8 Explain RS232 standards
- 5.9 List RS232 pins of DB 25 and DB 9 connectors
- 5.10 Explain MAX 232 and 233 and interfacing
- 5.11 Explain the need of relays and opto-couplers for interfacing
- 5.12 Interface 8051 with relay to drive a lamp
- 5.13 Interface a solid state relay to drive a mains operated motor
- 5.14 Explain the working of a stepper motor.
- 5.15 Draw and explain a driver circuit required to run a stepper motor
- 5.16 Interface a stepper motor
- 5.17 Write a program to run stepper motor continuously
- 5.18 Explain pulse width modulation for controlling the speed of small DC motor.

5.0. Introduction

8051 timers can be used to generate the required time delay. Timers 0 or 1 are used to count the clock pulses.

$$\text{Timer clock frequency} = \frac{\text{Crystal frequency}}{12} \quad \dots (1)$$

$$\text{Timer clock period} = \frac{12}{\text{Crystal frequency}} \quad \dots (2)$$

a) Time Delay

The time delay introduced between starting and stopping of the timer is calculated using the following formula:

$$\text{Time delay} = [\text{Maximum value} - \text{Initial count} + 1] \times \text{Timer Clock period}$$

or

$$\text{Time delay} = \left([\text{Maximum value} - \text{Initial count} + 1] \times \frac{12}{\text{Crystal frequency}} \right)$$

b) Initial Count

The initial count value is computed as follows:

$$\text{Initial count} = (\text{Maximum value} + 1) - (\text{Time delay} \times \text{Timer clock frequency})$$

or

$$\text{Initial count} = (\text{Maximum value} + 1) - \left(\frac{\text{Time delay} \times \text{Crystal frequency}}{12} \right)$$

c) Maximum Count Value

The maximum value depends on the mode of the timer as shown in table.

Mode	Timer Size	Maximum Count Value	
		Hexadecimal	Decimal
Mode 0	13-bit	1FFFH	8191D
Mode 1	16-bit	FFFFH	65535D
Mode 2	8-bit	FFH	255D

5.1 Explain how to program 8051 timers to create time delays.

- The 8051 has two built in timers i.e., timer-0 and timer-1. These are controlled by two SFRs TMOD and TCON.
- 8051 timers can be used to generate the required time delay.

- In order to generate certain time delay timer (0 or 1) register should be loaded with an appropriate number. Then run the timer and it starts counts up and reaches to its maximum value.
- While reaching the maximum value timer rolls over and stops the counting process. The following procedure to be adopted while finding the number to be loaded into the timer registers.

Procedure for Time Delay Generation:

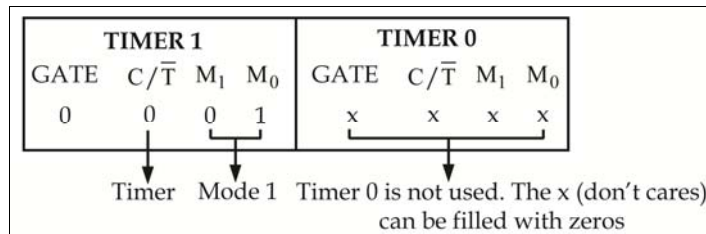
- Step 1:** Initialize TMOD for the required Timer (0 or 1) and mode (0 or 1 or 2)
- Step 2:** Load the initial count value in registers TL and TH (of timer 0 or 1)
- Step 3:** Start the timer (SET TR1 or TR0 bit of TCON register)
- Step 4:** Wait until the timer flag (TF1 or TF0 of TCON) is set. (Remember TFX=1 when timer X overflows the maximum value)
- Step 5:** Stop the timer (TR1 or TR0 is made zero)
- Step 6:** Clear TF flag
- Step 7:** Repeat from step 2 for the next delay.

The initial count value is computed as shown below:

Program 1: Write a 8051 assembly language program to generate a delay of 5ms using timer1 in mode 1 with XTAL frequency of 11.0592 MHz.

Solution: In mode 1, the timer counts up till FFFFH=65535D (maximum value for the 16-bit timer). Then it rolls over from FFFFH to 0000H and sets the flag bit TF=1.

Step 1: The timer mode register TMOD is programmed as



Here TMOD = 10H ⇒ Timer 1 in mode 1

Step 2: The initial count value (in hexadecimal) to be loaded into TL1 and TH1 is computed as,

$$\text{Initial count} = (\text{Maximum value} + 1) - (\text{Time delay} \times \text{Timer clock frequency})$$

$$= (65535 + 1) - \frac{5\text{m} \times 11.059\text{M}}{12} = 65536 - 4608 = 60,928\text{D} = \text{EE00H}$$

The initial value (16-bit) should be loaded into the 16-bit timer register T1 as TH1 = 0EEH (MSB) and TL1 = 00H (LSB).

Assembly Program:

	MOV TMOD, #10H	Timer 1, mode 1
AGAIN:	MOV TL1, #00H	Initial value in TL & TH
	MOV TH1, #0EEH	
	SETB TR1	START Timer 1
WAIT:	JNB TF1, WAIT	wait until timer 1 counts upto maximum i.e., jump if TF1 = 0
	CLR TR1	stop timer 1
	CLR TF1	clear TF1 so that it can be set again
	SJMP AGAIN	by the timer1 roll over

Additional Information

Problem: Find the initial count value to be loaded into TL1 and TH1 to generate 5ms time delay

The following procedure to be adopted while finding the number to be loaded into the timer registers.

1. Assume the XTAL frequency of the 8051 (f).
2. Calculate the clock period ($t = \frac{1}{f}$).
3. Divide the desired time delay by clock period. Result subtracted from maximum count i.e.; for 8-bit : 256 & 16-bit: 65 536.
4. Convert the result (obtained in step-3) into hexa decimal.
5. Load the hexa decimal number in timer register (TH & TL).

Solution:

$$\text{Clock period}(t) = \frac{1}{11.0592 \text{ MHz}} = 1.085 \text{ microsec.}$$

$$\text{Number of clocks} = \frac{5 \text{ millisecc}}{1.085 \text{ microsec}} = 4608$$

Number to be loaded in to timer - 0: 65536-4608=60,928D=EE00H.

Program 2: Find the time delay generated by timer 1 for the following program.

```

MOV TMOD, #10H
AGAIN:  MOV TL1, #00H
        MOV TH1, #0EEH

```

```

SETB TR1
WAIT:  JNB TF1, WAIT
      CLR TR1
      CLR TF1
      SJMP AGAIN
    
```

Solution:

- The timer 1 counts from the **initial value** EE00H to the **maximum value** (FFFFH).
- TMOD = 10H, C/T = 0, implies timer 1 is in timer mode and is counting the internal clock pulses.

- Timer clock period = $\frac{12}{\text{crystal frequency}}$
 $= \frac{12}{11.0592 \text{ MHz}} = 1.085 \mu\text{s}$

- Total time taken to roll over, that is set the TF1, is

$$\text{Time delay} = [\text{Maximum value} - \text{Initial count} + 1] \times \text{Timer clock period}$$

$$\therefore \text{Time delay} = (FFFF - EE00 + 1) \times 1.085 \mu\text{s} = 1200H \times 1.085 \mu\text{s}$$

$$= 4608d \times 1.085 \mu = 4996.425 \mu\text{s} = 4.996 \text{ ms}$$

Note: + 1 is added in the above formula because of the extra clock pulse needed when the timer rolls over from FFFF to 0 and raises the TF flag.

Program 3: Find the time delay obtained for the following program.

Solution:

	Instruction	Machine Cycles
	MOV TMOD, #10H	2
AGAIN:	MOV TL1, #0EAH	2
	MOV TH1, #0FFH	2
	SETB TR1	1
WAIT:	JNB TF1, WAIT	22 (depends on the initial value in the timer = max. value - init. Value + 1)
	CLR TR1	1 (FFFF - FFEA + 1)
	CLR TF1	1
	SJMP AGAIN	2

	Total	33
--	--------------	-----------

$$\text{Total delay} = \text{No. of machine cycles} \times \frac{12}{\text{Crtal frequency}}$$

$$\text{Total delay} = 33 \times \frac{12}{11.0592 \text{ M}} = 33 \times 1.085 \mu = 35.805 \mu\text{s}$$

Note:

$$\begin{aligned} \text{Maximum value} &\rightarrow \text{FFFFH} \\ \text{Intially value} &\rightarrow - \text{FEEAH} \\ &+ 1 \\ &16\text{H} = 22\text{D} \end{aligned}$$

5.2 Write programs to generate a square wave of given frequency and duty cycle using timer.

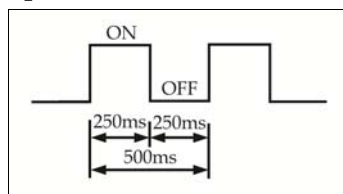
The time delays generated by the timers are used in many applications. One of the most widely used is in the generation of square wave. This is done by toggling (complementing) one or more port pins at a repeated fixed intervals of time (depending on the duty cycle).

Program 4: Write a program using timer 0 and mode 1 to generate a square wave of frequency 2KHz with 50% duty cycle on P1.5. Assume clock frequency is 11.0592 MHz .

Solution:

1. Clock frequency = 11.0592 MHz;
Hence period = $1/11.0592 \text{ MHz} = 1.085 \text{ microsec}$
2. Square wave frequency = 2 KHz;
Hence period = $1/2 \text{ kHz} = 500 \text{ microsec}$

Since 500 microsec is the period of square wave, 250 microsec will be ON period and 250 microsec will be OFF period.



3. Calculation of the initial value:
Initial value = Period of the pulse output/Internal clock period
 $= 250 \text{ microsec} / 1.085 \text{ microsec} = 230$
4. Subtract this value (230) from 65536

$$65536 - 230 = 65306$$

5. Convert this value (65306) into HEX

$$65536d = FF1AH$$

6. Load 1A in TL0 and FF in TH0.

Main Program:

Label	Instruction	Comment
START	MOV TMOD, #01H	Timer-0 under mode-1
REPEAT	MOV TL0, #1AH	Load TL0 with 1AH value
	MOV TH0, #0FFH	Load TH0 with 0FFH value
	CPL P1.4	Complement or Toggle P1.4
	ACALL DELAY	Delay subroutine called
	SJMP REPEAT	Repeat again to generate LOW/HIGH portion of square wave

Delay Subroutine:

Label	Instruction	Comment
DELAY	SETB TR0	Start the timer
STAY	JNB TF0, STAY	Stay and Monitor the timer flag '1' until it rolls over
	CLR TR0	Stop the timer
	CLR TF0	Clear TF0 flag
	RET	Return to the main program

Additional Information

The delay can be generated by a subroutine.

The **algorithm** for the square wave generation is given below.

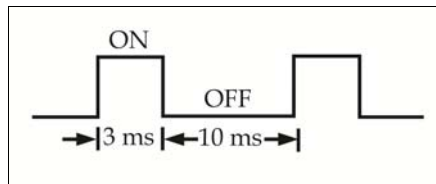
Main Program:

1. Initialize the timer 0 in mode 1.
2. Load the initial value into TL0 and TH0.
3. Complement (toggle) P1.4 (that is a 1 becomes a zero and vice versa).
4. Call timer delay subroutine.
5. Repeat from step 2.

Subroutine Algorithm:

1. Start timer.
2. Wait until timer flag is set.
3. Stop timer.
4. Clear timer flag.
5. Return to the main program.

Program 5: Generate a square wave with an ON time of 3ms and an OFF time of 10ms on pin P3.4. Assume a crystal frequency of 11.0592 MHz. Use timer 0 in mode 1.

Solution:

$$\text{The clock period} = \frac{12}{11.0592 \text{ MHz}} = 1.085 \mu\text{sec}$$

Case 1: ON time initial value computation:

$$\text{Number of clocks for ON time: } \frac{3 \text{ ms}}{1.085 \mu\text{s}} = 2765$$

$$\text{Initial value to be loaded: } 65536 - 2765 = 62271 = \text{F533H}$$

$$\boxed{\text{TH0} = \text{0F5H} \ \& \ \text{TL0} = \text{33H}}$$

Case 2: OFF time initial value computation:

$$\text{Number of clocks for OFF time: } \frac{10 \text{ ms}}{1.085 \mu\text{s}} = 9216$$

$$\text{Initial value to be loaded: } 65536 - 9216 = 56320 = \text{DC00H}$$

$$\boxed{\text{TH0} = \text{0DCH} \ \& \ \text{TL0} = \text{00H}}$$

Main Program:

Label	Instruction	Comment
START	MOV TMOD, # 01 H	Timer 0 in mode 1
BACK	MOV TL0, # 00H	To generate the OFF time, load TL0
	MOV TH0, # 0DCH	Load OFF time value in TH0
	CLR P3.4	Make port pin Low
	ACALL, DELAY	Call delay routine
	MOV TL0, # 33H	To generate on time, load TL0
	MOV TH0, # 0F5H	Load ON time value in TH0

	SETB P3.4	Make port pin HIGH
	ACALL DELAY	Call delay
	SJMP BACK	Repeat

Delay Subroutine:

Label	Instruction	Comment
DELAY	SETB TR0	Start the counter
AGAIN	JNB TF0, AGAIN	Check times overflow
	CLR TR0	When TF0 is set, stop the timer
	CLR TF0	Clear times flag
	RET	

5.3 Explain how to use an 8051 timer as an event counter.

5.3.1. Counter Programming

When timer/counter is used as an event counter, the source of clock pulses is outside the 8051. If $C/\bar{T} = 1$ in the TMOD register, timer/counter is used as an event counter and the source of clock pulses is outside the 8051. The pulses are fed from T0(P3.4) and T1(P3.5).

In the TMOD register, if $C/\bar{T} = 0$, the timer gets pulses from the crystal, when $C/\bar{T} = 1$ the timer is used as counter and gets its pulsed from outside the 8051. Therefore, when $C/\bar{T} = 1$ the counter count up as pulses are fed from pin 14 and 15. These pins are called T0 and T1, notice that these two pins belong to part 3. In the case of timer 0, when $C/\bar{T} = 1$, pin 3.4 provides the clock pulse and the counter counts up for each clock pulse coming from that pin. Similarly for timer 1, when $C/\bar{T} = 1$ each clock pulse coming in from pin 3.5 makes the counter count up.

a) Counter 0 in Mode 1

The simplified block diagram of Counter 0 in Mode 1 is shown in fig. The counter counts up when the logic signal on pin T0 (P3.4) goes from high level to low level (1 to 0 transition).

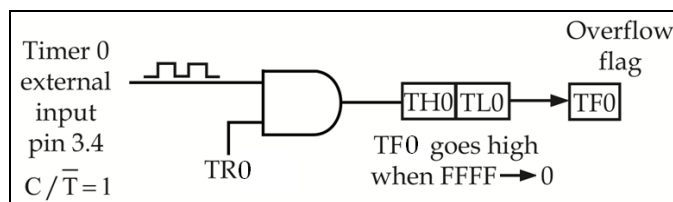


Fig. Counter 0 in mode 1

Timer/counter is used as an event counter by making $C/\bar{T}=1$ in the TMOD register. TR0 is used to start and stop the counter.

Counter 0 starts counting up at the rate at which the transitions occur at pin T0 (P3.4) of 8051. The counting will start from the 16-bit initial value present in the TH0 and TL0 registers. When the counter value exceeds FFFFH, timer overflow flag TF0 is set to 1.

b) Counter 1 in Mode 1

The simplified block diagram of Counter 1 in Mode 1 is shown in fig. The counter counts up when the logical signal on pin T1 (P3.5) goes from high level to low level.

Timer/counter is used as an event counter by making $C/\bar{T}=1$ in the TMOD register. TR1 is used to start and stop the counter.

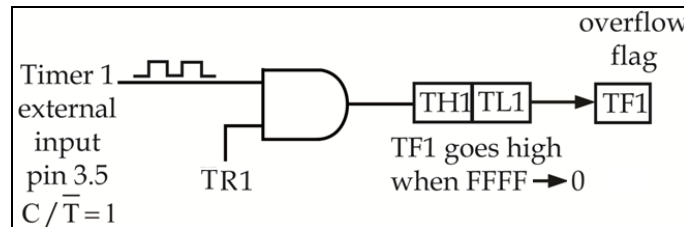


Fig. Counter 1 in mode 1

Counter 1 starts counting up at the rate at which the transitions occur at pin T1 (P3.5) of 8051. The counting will start from the 16-bit initial value present in the TH1 and TL1 registers. When the counter value exceeds FFFFH, timer overflow flag TF1 is set to 1.

5.4 Write a program to count the number of events using timer.

Program 6: Write a program to count the pulses of an input signal every second and display the count value on ports 1 and 2.

Solution:

The external input clock pulses can be fed to one of the timers say timer '0'. Hence timer '0' is acting as counter ($C/\bar{T}=1$) and the mode of timer '0' can be mode 1 ($M_1M_0=01$) for a very large count (16-bit counter-counts from 0000 to a maximum value of FFFFH). The external input is to be fed to pin P3.4. We need a 1 second timing to be generated. This can be done by using Timer '1' in mode 1 (refer example 8.5).

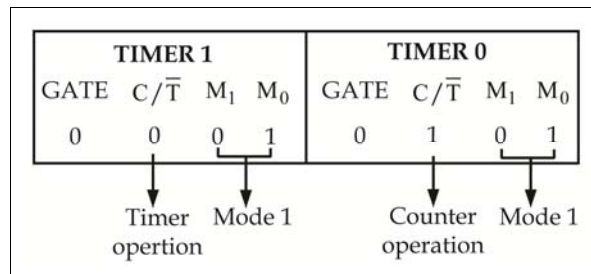
Maximum delay by timer 1 in mode 1

$$= (\text{FFFFH}) \times \frac{12}{\text{Crystal frequency}} = \frac{65535 \times 12}{11.0592 \text{ MHz}} = 0.0711 \text{ seconds}$$

This is repeated 14 times (register R0 is used as counter) to get $14 \times 0.0711 \approx 1$ second delay.

Algorithm:

1. Initialize timer 1 as timer, mode 1 and timer 0 as counter, mode 1.
2. Set P3.4 as input pin (to feed external pulses to be counted).
3. Initialize the timer 0 registers as 0000 (to count from 0) & start timer 0.
4. Initialize counter (R0) with 14 (for 1 second delay).
5. Initialize timer 1 registers with initial count (here 0000H for maximum delay of 0.0711 seconds).
6. Start timer1.
7. Wait till timer 1 overflows (TF1 = 1).
8. Stop timer 1 and clear TF1.
9. Decrement counter R0 and if not zero, repeat from step 5.
10. Display the count accumulated in TH0:TL0 on port 2 and 1.
11. Repeat from step 3 for continuous display every second.



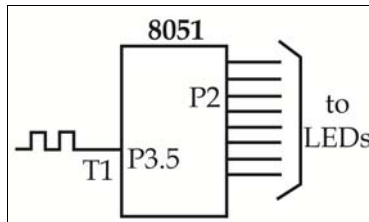
Hence TMOD = 15 H

Assembly Language Program:

	ORG 00H	
	MOV TMOD, #15H	; Timer 1 - - timer, mode 1 and ; timer 0 - - counter, mode 1
	SETB P3.4	; make port pin as input to accept external clock
rept:	MOV TL0, #00	; initialize counter (timer0) to count from zero
	MOV TH0, #00	
	SETB TR0	; start timer 0
	MOV R0, #14	; to repeat timer 1 14 times for 1 second delay

	CLR TR1	stop the counter 1
	CLR TF1	make the counter 1
	SJMP AGAIN	keep doing it

Notice in the above program the role of the instruction “SETB P3.5”. Since ports are set up for output when the 8051 is powered up, we make P3.5 an input port by making it high. In other words, we must configure (set high) the T1 pin (pin P3.5) to allow pulses to be fed into it.



P2 is connected to 8 LEDs and input T1 to pulse.

In example 4.15 we are using timer 1 as an event counter where it counts up as clock pulses are fed into pin P3.5. These clock pulses could represent the number of people passing through an entrance, or the number of wheel rotations, or any other event that can be converted to pulses.

5.5 Explain how to program 8051 serial port to transmit and receive serial data.

5.5.1. Procedure to Program the 8051 to Transfer Data Serially

The following steps give the steps to program 8051 for serial data transfer.

1. Initialize TMOD with 20H, to make use of timer 1 in mode 2 (auto reload) to set the baud rate.
2. Load the initial value into TH1 for the required baud rate.
3. Initialize SCON register, generally 50H for serial mode 1, 8-bit data, start and stop bits.
4. Start timer (TR1 = 1).
5. Clear TI flag.
6. Move the 8-bit data to be transmitted serially into SBUF register.
7. Wait until TI flag is set (remember TI flag is set after the contents of SBUF register, along with START and STOP bits have been transmitted serially on Tx pin).
8. For transferring another 8-bit data (character), repeat from step 5.

5.5.2. Procedure for Programming the 8051 to Receive Data Serially

All the above programs involved transmission of serial data. In this section we deal with the reception of character bytes serially. The following steps are needed.

1. Load TMOD register with 20H (for Timer 1, in mode 2) to generate serial clock.
2. Load initial value in TH1 for the required baud rate.
3. Load SCON with 50H (for serial mode 1, 8-bit data, start & stop bits and receive enable turned on) (refer example 8.16 for expansion of SCON).
4. Start timer 1 (TR1 = 1).
5. Clear RI flag (to enable reception).
6. Wait till RI flag goes high. RI = 1 when a character is received completely in SBUF.
7. Read SBUF for the received character.
8. Repeat from step 5 for receiving next character.

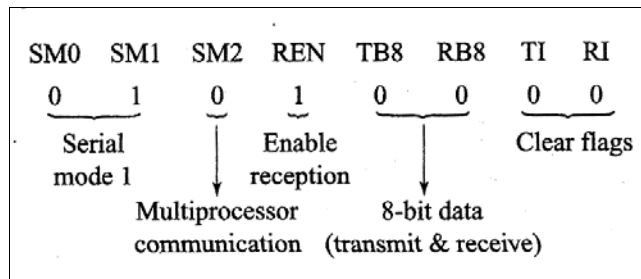
5.6 Write a program to transmit a message serially using serial port.

Program 8: Write a program to transfer a letter 'H' serially at 9600 baud continuously.

Solution:

TMOD register = 20H → Timer 1, Mode 2

SCON register initialization → SCON = 50H as shown below.



Assembly Language Program

	MOV TMOD, #20H	Timer 1, Mode 2 (auto reload)
	MOV TH1, #-3	9600 baud rate
	MOV SCON, #50H	serial mode 1, 8-bit data
	SETB TR1	start timer
AGAIN:	MOV SBUF, #'H'	move the character to SBUF register for serial transfer
WAIT:	JNB TI, wait	wait until serial transmission is over
	CLR TI	clear transmit flag
	SJMP AGAIN	repeat again

Program 9: Write an 8051 assembly language program to transfer the message "HELLO" serially at 9600,baud, 8-bit data, 1 stop bit.

Solution:

Main program:

	MOV TMOD,#20H	Timer 1, mode 2
	MOV TH1, #FDH	9600 baud rate
	MOV SCON, #50H	8-bit, 1 stop bit, REN enabled
	SETB TR1	Start timer 1
START	MOV A, #"H"	Transfer "H"
	ACALL TRANS	
	MOV A, #"E"	Transfer "E"
	ACALL TRANS	
	MOV A, #"L"	Transfer "L"
	ACALL TRANS	
	MOV A, #"L"	Transfer "L"
	ACALL TRANS	
	MOV A, #"0"	Transfer "0"
	ACALL TRANS	Serial data transfer subroutine

Delay Subroutine:

TRANS	MOV SBUF, A	Load SBUF
HERE	JNB TI, HERE	Wait for the last bit to
		Transfer
	CLR TI	Clear TI for the next Character
	RET	

5.7 Write a program to receive a message serially and store it in memory.

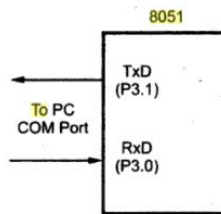
Program 9: Program the 8051 to receive bytes of data serially, and put them in PI. Set the baud rate at 4800, 8-bit data, and 1 stop bit

Solution:

	MOV TMOD, #20H	Timer 1, mode 2(auto-reload)
	MOV TH1, #-6	4800 baud
	MOV SCON, #50H	8-bit, 1stop, REN enabled
	SETB TR1	Start Timer 1
HERE	JNB RI, HERE	Wait for char to come in
	MOV A,SUBUF	Save incoming byte in A
	MOV P1,A	Send to port 1
	CLR RI	Get ready to receive next byte
	SJMP HERE	Keep getting data

Program 9: write a program to receive message from PC to 8051. Message string is "Hello". After this micro controller sends message to PC "Fine".

Solution: The fig. shows the connection between 8051 and PC.



	MOV TMOD, #20H	Initialize timer 1 in mode 2
	MOV TH1, #0FDH	Load count to get 9600 baud rate
	MOV SCON, #50H	8-bit, 1 stop, REN enabled
	SETB TR1	Start timer 1
	MOV DPTR, #2000H	Initialize memory pointer to save received data
	MOV R0, #05H	Initialize counter to read 5 characters
RECV	JNB RI, RECV	Wait for character
	MOV A, SBUF	Read the character
	MOVX @ DPTR, A	Save it in memory
	INC DPTR	Increment memory pointer
	CLR RI	Get ready for next character

	DJNZ R0, RECV	If not last character repeat
	MOV DPTR, #MY DATA	Initialize pointer for message
	MOV R0, #4H	Initialize counter to send 4 characters
	MOVC A, @A+DPTR	Get the character
	MOV SBUF, A	Load the data
	JNB TI, HERE	Wait for complete byte transfer
	CLR TI	Get ready for next character
MYDATA	DB "Fine", 0	
	END	

Program 12: Write a program to receive serial data and place it in RAM memory location 62H and also send it to port P2.

Solution:

Assembly Language Program

	MOV TMOD #20H	; Timer 1, mode 2
	MOV TH1, # - 3	; 9600 Baud rate
	MOV SCON, #50H	; serial mode 1, receive enable, 8-bit, START & Stop bit
	SETB TR1	; start timer
again:	CLR RI	; clear flag
wait:	JNB RI, wait	; wait till character is received
	MOV A, SBUF	; put received character into A
	MOV 62H, A	; copy into memory location 62H
	MOV P2, A	; output to port P2
	SJMP again	; repeat for next character

5.8 Explain RS232 standards

RS 232 is the most widely used serial I/O interfacing standard. The RS232 standard was published by the Electronic Industry Association (EIA) in 1960. The COM1 and COM2 ports in IBMPC are RS 232 compatible ports. In RS 232, 1 is represented by - 3 to - 25V and 0 is represented by + 3 to + 25V .

In a microcontroller, serial TXD and RXD lines are TTL compatible i.e. 1 and 0 are represented by +5V and 0V. For this reason, in order to connect a microcontroller to RS 232 bus, voltage converters are used. MAX 232 IC is commonly used to convert the TTL logic levels to the RS 232 voltage levels. The significance of the number 232 is that 2 is transmission line, 3 is receiving line, and 7(2+3+2) is signal ground line. In RS 232, ground line is common to the transmitter and receiver, and they are usable up to one meter without any shield.

5.8.1. RS 232 Pin CONNECTORS

Basic data communication link is shown in fig. The communication link consists of Data Terminal Equipment (DTE) and an associated modem (DCE) at each end. The function of modem is to process digital information received from the computer into a form suitable for analog transmission. Also, it receives analog signal and processes it into digital information.

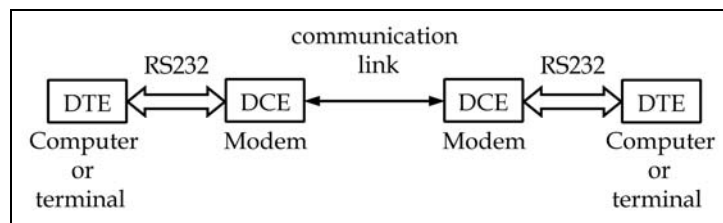


Fig. Data communication system

RS232 uses a 25 pin plug connector for all interface circuits and is commonly referred to as the DB-25 pin connector. DB-25P refers to the plug connector (male) and DB-25S refers to the socket connector (female). Since all the 25 pins are not used in PC, IBM introduced DB-9 connector. These plug connectors are shown in fig.

There are basically two standards for serial data transmission between data terminal equipment (DTE) and data communication equipment (DCE). The transmitted data either in current or voltage form. When data are transmitted as a voltage form, the commonly used standard is known as RS-232.

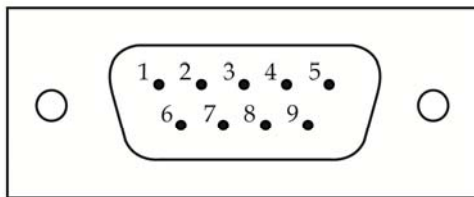
RS-232 means Recommended Standard-232. It is a single ended, bipolar voltage, 25-pin electrical interface. It was introduced in 1960. It was developed by Electronics Industries Association (EIA). There are some versions of RS-232 are developed, they are RS-232A, RS-232B and RS-232C etc. In which RS-232C is the most widely used serial I/O interfacing standard.

RS-232C is widely used for direct connection between data acquisition and computer systems. Where the data acquisition system is referred as DCE (data communication equipment) such as Printer, Modem, Data storage units etc., and PC is the example for DTE. The rate of data transmission in RS-232C is restricted to a maximum of 20kbaud and a distance of 50ft.

5.9 List RS232 pins of DB 25 and DB 9 connectors

The RS-232 specifies 25 signals for the bus used for serial data transfer. In practice first 9 signals are sufficient for most of the serial data transmission; RS-232 serial bus is terminated on D-type 9-pin connector. In case all 25 signals are used, then RS-232 serial bus is terminated on D-type 25-pin connector. DB9 and DB25 connectors of RS-232 are shown below.

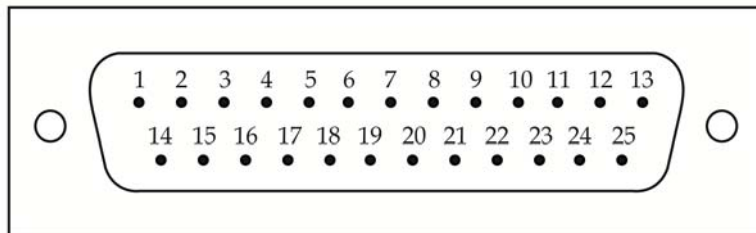
a) DB- 9P connector details



DB9 Connector

Pin Number	Common Name	Details
1	\overline{DCD}	Data Carrier Detector
2	RxD	Receive Data
3	TxD	Transmit Data
4	DTR	Data Terminal Ready
5	GND	Signal Ground
6	\overline{DSR}	Data set ready
7	RTS	Request to send
8	CTS	Clear to send
9	RI	Ring Indicator

b) DB 25P connector details



DB25 Connector

Pin Number	Common Name	RS-232C Name	Details	Direction of signal on DCE
1		AA	Protective ground	
2	TxD	BA	Transmit data through this pin	IN
3	RxD	BB	Data received through this pin	OUT
4	$\overline{\text{RTS}}$	CA	Request signal to send data	IN
5	$\overline{\text{CTS}}$	CB	Clear to send	OUT
6	$\overline{\text{DSR}}$	CC	Data set ready	OUT
7	GND	AB	Ground pin	--
8	$\overline{\text{CD}}$	CF	Received line signal detector	OUT
9	-	-	Reserved for Data set testing	-
10		-	Reserved for Data set testing	-
11		-	Unassigned	-
12		SCF	Secondary Received line signal detector	OUT
13		SCB	Secondary clear to send	OUT
14		SBA	Secondary Transmitted Data	IN
15		DB	Transmission signal element timing(DCE source)	OUT
16		SBF	Secondary received data	OUT
17		DD	Receiver signal element timing(DCE source) Unassigned	OUT
18		-	Unassigned	-
19		SCA	Secondary request to send	IN
20	$\overline{\text{DTR}}$	CD	Data terminal ready	IN
21		CG	Signal quality detector	OUT
22		CE	Ring indicator	OUT
23		CH/CI	Data signal rate selector (DTE/DCE Source)	IN/OUT
24		DA	Transmit signal element timing (DTE source)	IN
25		-	Unassigned	-

5.10 Explain MAX 232 and 233 and interfacing

5.10.1. MAX232

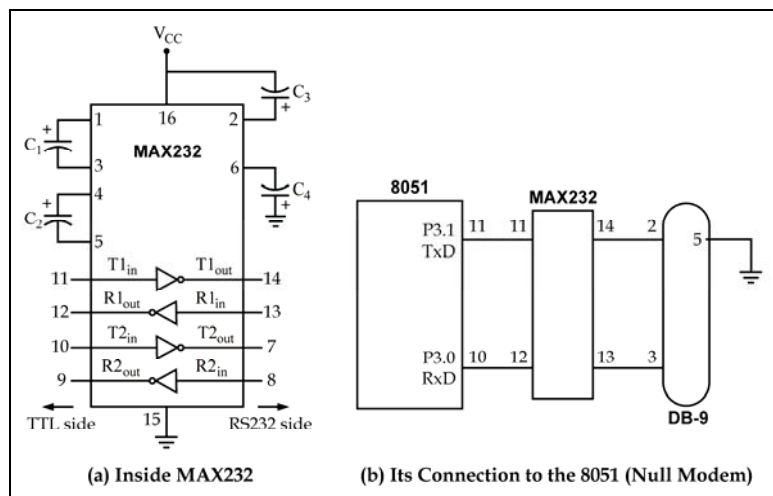
The MAX232 converts RS232 voltage levels to TTL voltage levels and vice versa. One advantage of the MAX232 chip is that it uses a +5V power source which is the same as the source voltage for the 8051. In other words, with a single +5V power supply we can power both the 8051 and MAX232, with no need for the dual power supplies that are common in many older systems.

The MAX232 has two sets of line drivers for transferring and receiving data, as shown in fig. The line drivers used for TxD are called T_1 and T_2 , while the line drivers for RxD are designated as R_1 and R_2 . In many applications only one of each is used. For example, T_1 and R_1 are used together for TxD and RxD of the 8051, and the second set is left unused.

Circuit Implementation:

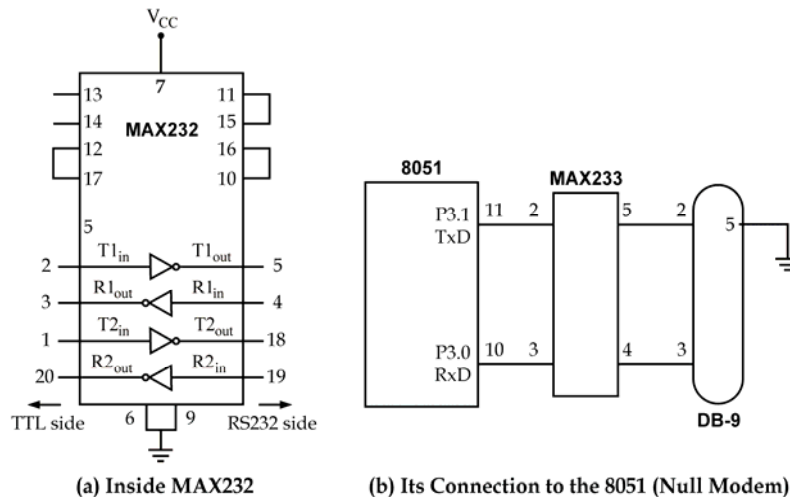
Notice in MAX232 that the T1 line driver has a designation of $T1_{in}$ and $T1_{out}$ on pin numbers 11 and 14, respectively. The $T1_{in}$ pin is the TTL side and is connected to TxD of the microcontroller, while $T1_{out}$ is the RS232 side that is connected to RxD pin of the RS232 DB connector. The R1 line driver has a designation of $R1_{in}$ and $R1_{out}$ on pin numbers 13 and 12, respectively. The $R1_{in}$ (pin 13) is the RS232 side that is connected to the TxD pin of the RS232 DB connector, and $R1_{out}$ (pin 12) is the TTL side that is connected to the RxD pin of the microcontroller. See fig. Notice the null modem connection where RxD for one is TxD for the other.

MAX232 requires four capacitors ranging from 1 to 22 μ F. The most widely used value for these capacitors is 22 μ F.



5.10.2. MAX233

To save board space, some designers use the MAX233 chip from Maxim. The MAX233 performs the same job as the MAX232 but eliminates the need for capacitors. However, the MAX233 chip is much more expensive than the MAX232. Notice that MAX233 and MAX232 are not pin compatible. You cannot take a MAX232 out of a board and replace it with a MAX233. See fig. for MAX233 with no capacitor used.



Additional Information

* Interrupts of 8051 *

A single microcontroller can serve several devices. There are two methods to do this. The first method is called polling wherein the CPU continuously checks whether any device needs to be serviced. In the interrupt method; the external devices sends a signal to interrupt whatever the microcontroller is doing, and serve the device. Polling method does not use the microcontroller efficiently and precious CPU time is wasted. Further assigning priority is not possible.

Advantages of Interrupt Method over Polling Method:

1. In polling method, the microcontroller is not used efficiently. When this method is used to service more than one device, it cannot assign priority, but serves in round robin fashion, and hence cannot ignore (mask) a device request for service.
2. In interrupt method, the microcontroller is free from checking the status of the devices. Only when the devices are ready, they interrupt the microcontroller and get the service. In the meantime the microcontroller can be programmed for some useful tasks.
3. The other advantage is that priority can be assigned to the interrupts and the interrupts can be masked (ignored).

5.11 Explain the need of relays and opto-couplers for interfacing

5.11.1 Need of relays for Interfacing

- A relay is an electrically controllable switch widely used in industrial controls, automobiles, and appliances.
- It allows the isolation of two separate sections of a system with two different voltage sources.

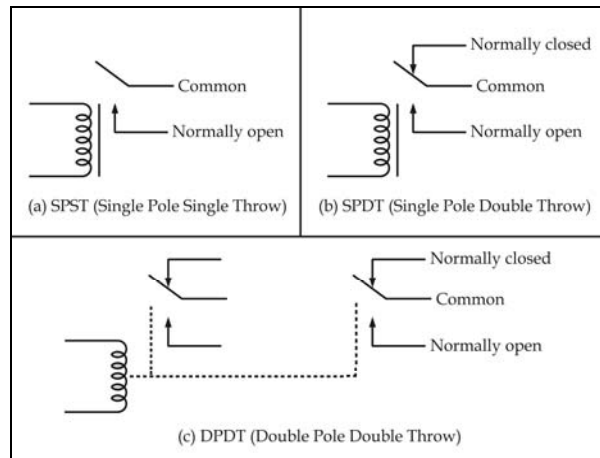


Fig.

- Example, a +5V system can be isolated from a 120V system by placing a relay between them. An electromechanical (or electromagnetic) relay (EMR) is shown in fig.
- The EMRs have three components: the coil, spring and contacts. In fig. a digital +5V on the left side can control a 12V motor on the right side without any physical contact between them.
- When current flows through the coil, a magnetic field is created around the coil (the coil is energized), which cause the armature to be attracted to the coil.
- The armature's contact acts like a switch and closes or opens the circuit. When the coil is not energized, a spring pulls the armature to its normal state of open or closed.

5.11.2. Need Of Opto Couplers For Interfacing

- An optocouplers is also known as optoisolator.
- An optoisolator is a combined package of a LED and photo sensor (photo diode or photo transistor).

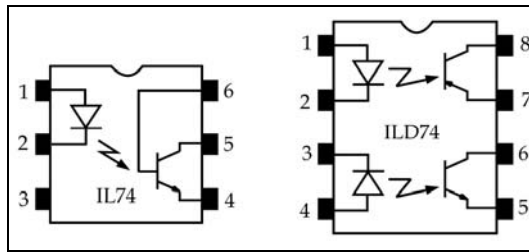


Fig.

- An optoisolator is used to isolate two parts of a system.
- An example is driving a motor. Motors can produce what is called back EMF, a high voltage spike produced by a sudden change of current as indicated in the $V = L di/dt$ formula.
- In situations such as printed circuit board design, we can reduce the effect of this unwanted voltage spike (called ground bounce) by using decoupling capacitors (see Appendix C).
- In systems that have inductors (coil winding), such as motors, decoupling capacitor or a diode will not do the job. In such cases we use optoisolators.
- Optoisolators are widely used in communication equipment such as MODEMS. This allows a computer to be connected to a telephone line without risk of damage from power surges.
- Fig. shows the optoisolator packages, IL 74 and ILD 74. IL 74 contains only one optoisolator. ILD 74 contains two optoisolators. ILQ 74 contains four optoisolators (not shown in fig).

5.12 Interface 8051 with relay to drive a lamp

- Fig. shows the interfacing of a relay 106462 with microcontroller 8051. Here the port pin P1.0 is connected to the input side of relay. P1.0 acts as an output port. A lamp is connected to the output side of the relay. The 8051 microcontroller controls the lamp output through the relay 106462.

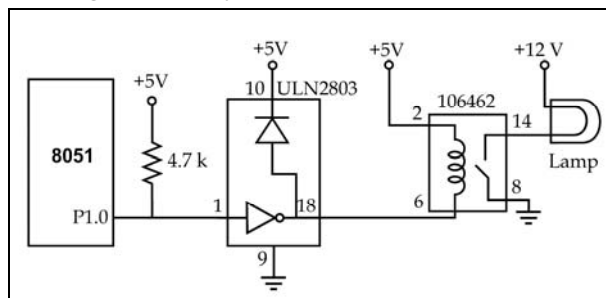


Fig.

- The relay given in the fig is a SPST-NO, (single pole, signal throw with normally opened). When +5V DC is applied to its input, the coil is energized and the NO contact is closed.
- The output current at the port pin of the microcontroller is insufficient to drive the relay. It can provide a maximum of 1 to 2mA current. But the relay coil requires around 10mA current to be energized. Therefore a driver (ULN 2803) is used between the microcontroller and the relay.

Note: The driver is used to raise the current, for driving the relay.

ALP to turn ON and OFF the lamp via the Relay

The following program turns the lamp ON and OFF by energizing and de-energizing the relay every second.

	ORG 0H;	Origin at 0H
MAIN:	SETB P1.0	;Make port pin P 1.0 as high
	ACALL DELAY	;Call delay subroutine
	CLR P 1.0	;Make port pin P1.0 as low
	ACALL DELAY	;Call delay subroutine
	SJMP MAIN	;Repeat the process
	END	;End of program
DELAY:	MOV R1, # 37H	;Delay subroutine
L3	MOV R2, # 64H	
L2	MOV R3, # FD H	
L1	DJNZ R3, L1	
	DJNZ R2, L2	
	DJNZ R1, L3	
	RET	
	END	

Additional Information:

Fig. illustrates the symbol of relay and interfacing of a relay with 8051. Simply making the pin '0' or '1' will switch ON/OFF the relay.

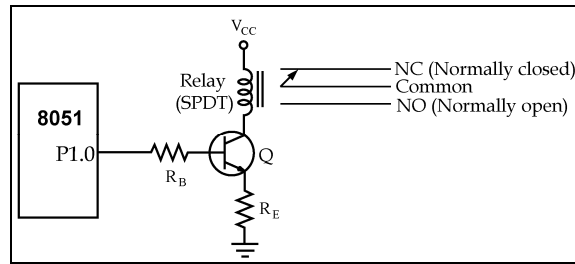


Fig.

5.13 Interface a solid state relay to drive mains operated motor

5.13.1. Solid state relay

- A solid state relay (SSR) is a solid state electronic component that provides a similar function to an electromechanical relay but does not have any moving components.
- In this relay, there is no coil, spring, or mechanical contact switch. The entire relay is made out of semiconductor materials. Because no mechanical parts are involved in solid- state relays, their switching response time is much faster than of electromechanical relays.
- Another problem with the electromechanical relay is its life expectancy. The life cycle for the electromechanical relay can vary from a few hundred thousand to few million operations. Wear and tear on the contact points can cause the relay of malfunction after a while. Solid-state relay have no such limitations.
- Extremely low input current and small packaging make solid-state relays ideal for microprocessor and logical control switching.
- They are widely used in controlling pumps, solenoids, alarms, and other power applications.
- Some solid-state relays have a phase control option, which is ideal for motor-speed control and light- dimming applications.

5.13.2. Interface a solid state relay to drive mains operated motor

- Fig illustrates the internal structure of a solid state relay and its interfacing with 8051.
- A photo-coupled SSR is controlled by a low voltage signal which is isolated optically from the load. The control signal in a photo-coupled SSR typically energises an LED which activates a photo-sensitive diode.
- The diode turns on a back-to-back thyristor, silicon controlled rectifier, or MOSFET transistor to switch the load.

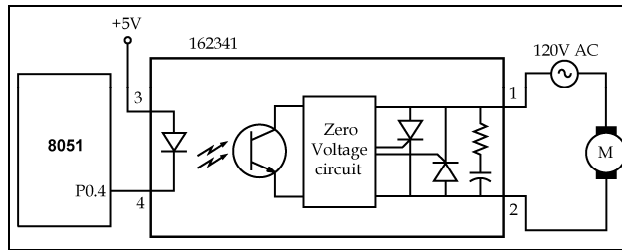
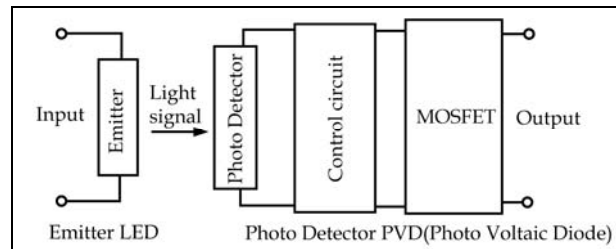


Fig.



Additional Fig.

Additional Information

*** Stepper motors ***

a) Introduction to Stepper motor

- Basically stepper motors are electromechanical converters that translate electrical pulses into mechanical movement. They differ from constant speed motors in the shaft movement i.e., the movement of the shaft will be in steps.
- The angle through which the motor rotates for each input pulse is called the step angle (α). It is given by

$$\text{Step angle, } \alpha = \frac{360^\circ}{\text{number of steps per revolution}}$$

- If the number of steps per revolution is 180, then step angle is

$$\alpha = \frac{360^\circ}{180} = 2^\circ$$

- The step angle can be adjusted by changing one of the following.
 - a) Number of poles on the rotor
 - b) Number of phases (windings) on the stator
 - c) Sequence of excitation of stator windings.

Note: The below table gives some step angles.

Step angle	Steps per revolution
0.72	500
1.8	200
2.0	180
2.5	144
5.0	72
7.5	48
15	24

b) Construction of stepper motor

Fig. (a) shows the constructional details of permanent magnet stepper motor. It consists of:

- a) Rotor without windings
- b) Stator with windings

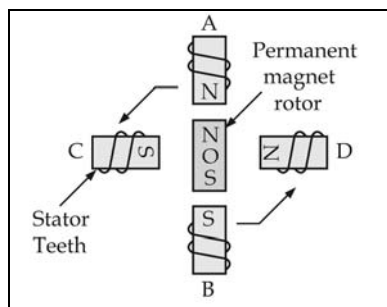


fig. (a)

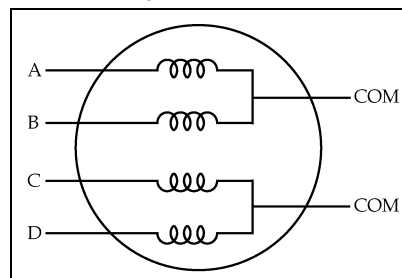


Fig. (b)

- The stepper motor discussed here has a total of 6 leads: 4 leads representing the four stator windings and 2 commons for the center tapped leads (see fig. (b)).
- Such a stepper motor is called as four phase or uni-polar stepper motor.

1. Rotor

- Stepper motor has a permanent magnet rotor.
- It is made of alnico magnetic material

- The number of teeth on the rotor is referred to as “pole”.

2. Stator

- The fig. (b) shows stator winding configuration.
- Stator is made of low retentivity steel. The number of windings in the stator is referred to as “phase”. Phase windings are placed on the projected portions of the stator.
- The stator windings (A, b, C and D) must be excited in a particular sequence. This makes the motor to move to next step. This sequence is called as “Excitation sequence”.
- A stepper motor requires an electronic controller to energize its windings in a proper sequence thereby causing it to step. The stepper motor controller requires:
 - Logic sequencer
 - Power drivers

Logic sequencer	Power drivers
<ul style="list-style-type: none"> • The logic sequencer generates the following sequence in which the stator windings are to be excited. 	<ul style="list-style-type: none"> • The power drivers supply necessary current to energize the motor windings.

5.14 Explain the working of a stepper motor.

- A permanent magnet stepper motor has a rotor, which is a permanent magnet and a stator made up of electromagnets.
- The rotor will move to align itself to the energized electromagnet (field pole). If the field magnets are energized one after the other around the circle, the rotor can move making a complete rotation.
- Four-phase unipolar stepper motors are commonly used. The term phase refers to the number of separate winding circuits. In case of four-phase stepper motor, there are four field windings energized independently. Unipolar means that the current always flows in the same direction in the coils.

Case - 1: Four step wave drive sequence

- Fig. shows the timing diagram for exciting a four-phase unipolar stepper motor.
- Wave drive 4-step sequence has only one winding energized at a given instant of time. The motor is made to rotate in steps of 90° in the clockwise direction by exciting the stator phases.

- When phase A is excited the magnetic field is setup along A. Rotor aligns itself along A. When phase B is excited, the magnetic field is setup along B. Rotor aligns itself along B and so on.
- Thus the motor is made to rotate in steps of 90° in clockwise direction by exciting the stator phases sequentially in the following order A, B, C and D.
- The motor is made to rotate in steps of 90° in the counter clockwise direction by exciting the stator phases sequentially in the following order: D, C, B and A. Table shows the rotation of the rotor in a stepper motor along with the wave drive 4-step sequence.

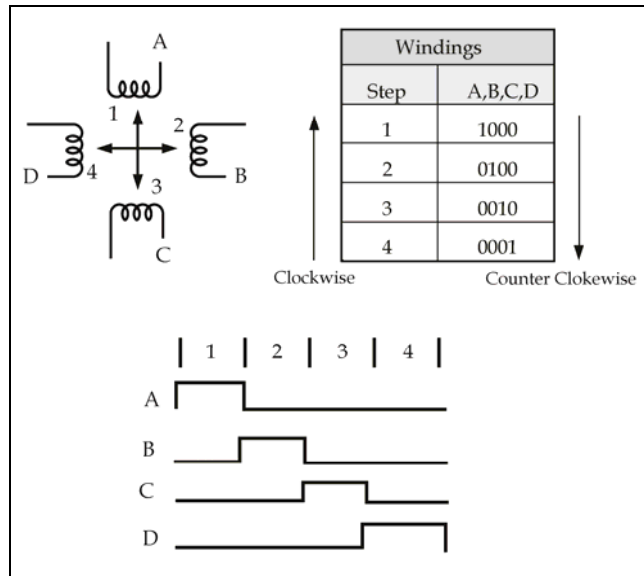


Fig.

Case - 2: Half step sequence

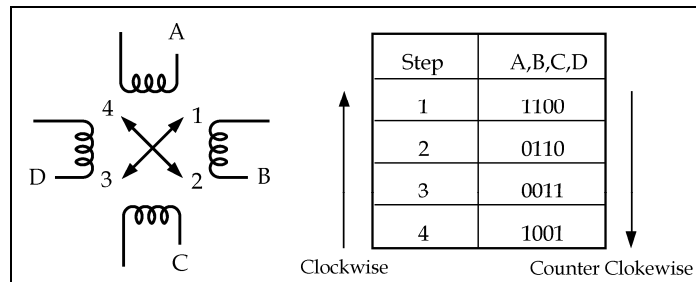


Fig.

- When phase A and B are excited, the magnetic field is setup between A and B. Rotor aligns itself between A and B. The motor is made to rotate in steps of 90° in the clockwise direction by exciting the stator phases sequentially in the following order: AB, BC, CD and AD.

- The motor is made to rotate in steps of 90° in the counter clockwise direction by exciting the stator phases sequentially in the following order: AD, CD, BC and AB. Table. 10.5 shows the rotation of the rotor in a stepper motor, along with the winding energization sequence.

Note:

- The four step wave drive sequence is also called as “full stepping” sequence.
- To allow finer resolutions all stepper motors use an 8 step switching sequence. It is also called as half stepping as follows.
- Each step is the half of the normal step angle.

Case – 3: Half step 8-step sequence

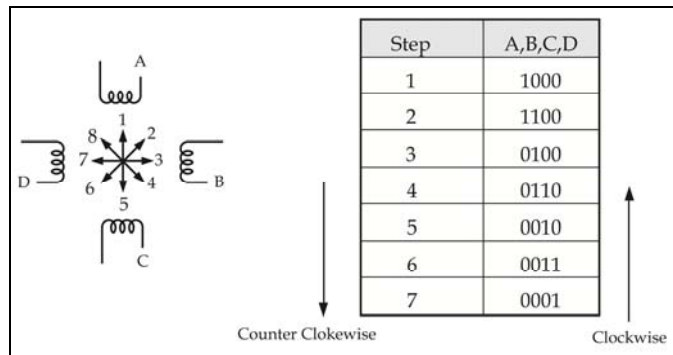
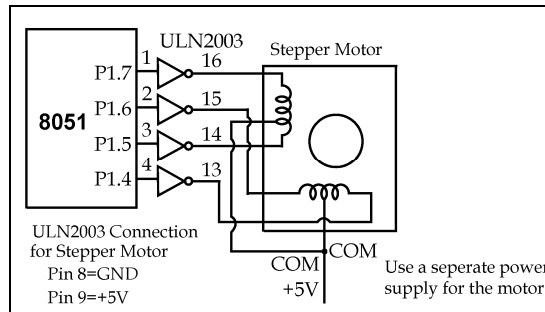


Fig.

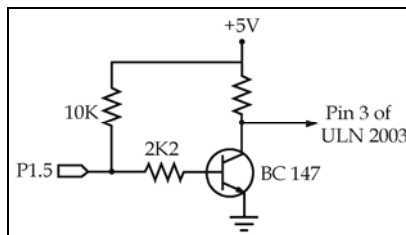
To double the number of steps/revolution, 8-step sequence is followed as shown in Table. With this method, the step size is half the original step size and hence the 8-step sequence is called ‘half stepping’. The motor is made to rotate in step of 45° in the clockwise direction by exciting the stator phases sequentially in the following order: A, AB, B, BC, C, CD, D, and AD. The motor is made to rotate in steps of 45° in the counter clockwise direction by exciting the stator phases sequentially in the following order: AD, D, CD, C, BC, B, AB and A.

5.15 Draw and explain a driver circuit required to run a stepper motor

- The stepper motor requires high power where as the output power level of the sequential signal produced from the microcontroller are small. Hence microcontroller cannot be used directly to drive the motor. Power drivers are inserted in between the microcontroller and the motor.
- The power drivers supply necessary current to energized the motor windings.



- Fig. shows the method of connecting stepper motor to microcontroller 8051. ULN 2003 is used as a power driver to energize the stator. The ULN 2003 has an internal diode to take care of back EMF.



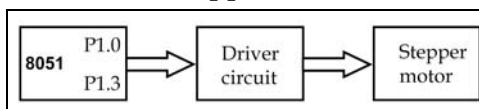
- The micro controller port lines P1.4, P1.5, P1.6, and P1.7 are used to control the four windings of the stepper motor.
- After power on reset, the port lines of the microcontroller will be in the high level, which may energize all the windings at the same time. In order to avoid this error condition, signal from the port line is inverted using with the switching transistor BC147 and then given to the power driver. ULN 2003.
- In the fig only P1.5 port alone is shown. Similar invertors are used for the other ports P1.4, P1.6 and P1.7.

Circuit Operation:

- Sending '0' level to the connected port line can energize a motor winding.
- The signal gets inverted in the transistor BC147 and the signal becomes a '1' level. This level will make the power driver IC ULN 2003 to energize the motor winding.
- The windings must be excited in a sequence and the excitation sequence is generated by the microcontroller.

5.16 Interface a stepper motor

- The fig. shows block schematic of stepper motor interfaced to 8051.



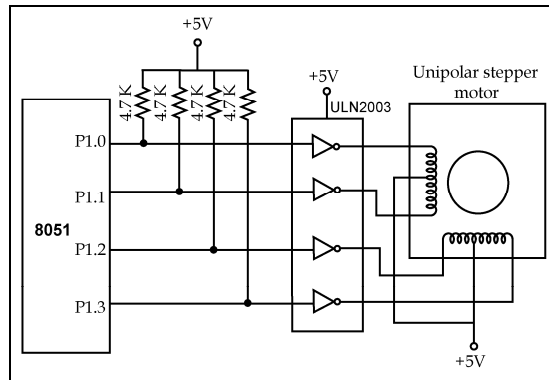


Fig.

- Fig. shows 8051 connection to stepper motor.
- The four leads of the stator winding are controlled by port 1 bits P1.0 - P1.3.
- The 8051 does not have sufficient current to drive the stepper motor windings.
- Hence, a drive like ULN 2003 is required to energize the stator.

5.17 Write a program to run stepper motor continuously

Program 1: Write an assembly language program to rotate a stepper motor continuously

Algorithm:

Step:1 Load initial value 99H into P1 for half step sequence

Step:2 Call delay

Step:3 Rotate the initial value right for clockwise rotation (left for counter clockwise direction)

Step:4 Repeat from step 2

Assembly language program

Main program:

	MOV A,# 99h	;initial step sequence
NEXT:	MOV P1, A	;output on P1 for motor to rotate
	RR A	;rotate clockwise (use RL A for counter clockwise)
	ACALL DELAY	;DELAY controls the speed of the stepper motor
	SJMP NEXT	;repeat continuously

Delay program:

DELAY:	MOV R2, #10H	;delay subroutine. Change R2 value for speed variation
LOOP2:	MOV R1, #FFH	
LOOP1:	DJNZ R1, LOOP1	
	DJNZ R2, LOOP2	
	RET	

Additional Information

Program 2: Write an assembly language program to rotate a stepper motor 100° in the clock wise direction. The motor has a step angle of with the normal 4-step sequence.

Solution:

$$\text{Number of steps} = \frac{100^\circ}{\text{step angle}} = \frac{100^\circ}{2^\circ} = 50$$

Algorithm:

Step:1 Initialize a counter with 50 steps

Step:2 Send initial phase sequence, i.e., 99H to P1 port

Step:3 Call delay

Step:4 Rotate the phase sequence right (for clock-wise direction)

Step:5 Decrement counter and repeat from step 1 if counter is not zero

Step:6 If counter is zero, wait here.

Assembly language program

Main program:

	ORG 00H	
	MOV A, #99H	;initial phase sequence
	MOV R0, #50	;counter for 50 steps for 100° rotation
BACK:	RR A	;rotate sequence right for clockwise rotation
	MOV P1, A	;output to port for stepper motor to rotate

	ACALL DELAY	;small delay between steps
	DJNZ R0, BACK	
HERE:	SJMP HERE	;wait here after 100° rotation

Delay program:

DELAY:	MOV R2, #10H	;delay subroutine. Change R2 value for speed variation
LOOP2:	MOV R1, #FFH	
LOOP1:	DJNZ R1, LOOP1	
	DJNZ R2, LOOP2	
	RET	

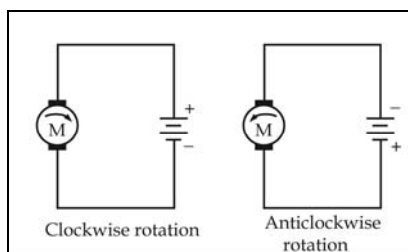
5.18 Explain pulse width modulation for controlling the speed of small DC motor.

5.18.1. DC motor interfacing

Unlike stepper motor, the DC (direct current) motor rotates continuously. It has two terminals positive and negative. Connecting DC power supply to these terminals rotates motor in one direct and reversing the polarity of the power supply reverses the direction of rotation. The speed of the DC motor is measured in revolutions per minute (RPM).

To speed of the DC motor increases with increase in the supply voltage. However, we cannot exceed supply voltage beyond the rated voltage. The speed of the DC motor also depends on the load. At no-load speed is highest. As we increase the load, the speed decreases. The overloading the DC motor can damage it because of excessive heat generated due to high current consumption.

Direction control:



- The fig shows how dc motor changes its direction of rotation when the polarity of power supply reverses.
- By using switches for changing the power supply polarity we can control the direction of the rotation of the DC motor. This is illustrated in the fig.

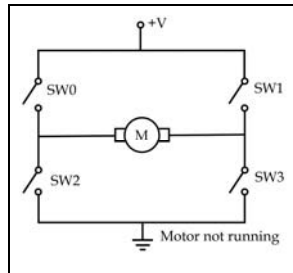


Fig: Bidirectional control of DC motor.

SW3	SW2	SW1	SW0	Motor rotation
ON	OFF	OFF	ON	Colckwise
OFF	ON	ON	OFF	Anticlockwise

Table: Switch configurations.

- Any other switch configurations, for example, SW0 and SW2 ON is invalid, because it creates short circuit of the power supply.

Example: Write an 8051 ALP to control the direction of the DC motor according to the status of bit P1.10. Assume port pins P2.0, P2.1, P2.2 and P2.3 controls the switches SW0, SW1, SW2 and SW3 respectively.

Solution:

	ORG OH	
	CLR P2.0	;make all switches OFF
	CLR P2.1	
	CLR P2.2	
	CLR P2.3	
	SETB P1.0	;configure P1.0 as input
Check:	JNB P1.0	;Clockwise
	CLR P2.0	;make SW0 OFF
	SWTB P2.1	;make SW1 ON
	SWTB P2.2	;make SW2 ON

	CLR P2.3	;make SW3 OFF
	SJMP check	
Clockwise:	SETB P2.0	;make SW0 ON
	CLR P2.1	;make SW1 OFF
	CLR P2.2	;make SW2 OFF
	SETB P2.3	;make SW3 ON
	SJTB check	
	END	

5.18.2. Pulse width modulation (PWM)

DC motor's speed is varied by varying the dc voltage supplied to it. Pulse Width Modulation (PWM) method can be used to provide variable output. By changing the width of the pulse, the average dc output voltage V_{out} can be varied. The relationship between pulse width and the average output voltage is as follows:

$$V_{out} = \text{Duty cycle} \times V_{max}$$

Where
$$\text{Duty cycle} = \frac{T_{on}}{T_{off} + T_{on}}$$

We know that the speed of the dc motor depends on the applied voltage. The average applied Dc voltage and hence the power can be varied using technique called pulse width modulation. In this technique, the dc power supply is not a voltage of fixed amplitude; however it is a pulsating DC voltage. By changing pulse width we can change the applied power. This is illustrated in the fig.

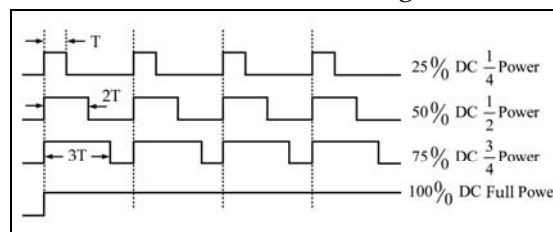


Fig: Applied power variation using pulse width modulation.