

MICROCONTROLLERS

IV Sem EC/EE/IT/TC/BM/ML



ARUNKUMAR G
M.Tech Lecturer in
E&CE Dept.,
S.T.J.I.T., Ranebennur.

For suggestions and feedback:

Cell: 9731311770

e-mail: arunkumar.stjit@gmail.com



ARUNKUMAR.G M.Tech, *Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.*

Microprocessors & Microcontroller

- 1) Differentiate between a microprocessor & a microcontroller
- 2) Give the comparison b/w microprocessor & microcontroller
- 3) Bring out the architectural difference b/w a microprocessor & a microcontroller.

Jan-09, 6M

Model, 6M

Jan 06, 6M

SL No	Microprocessor	Microcontroller
1)	<pre> graph TD ALU[Arithmetic and Logic Unit] ACC[Accumulator] REG[Working Register(s)] PC[Program Counter] SP[Stack Pointer] CLK[Clock Circuit] INT[Interrupt Circuits] ALU --- ACC ACC --- REG REG --- PC PC --- SP SP --- CLK CLK --- INT </pre>	<pre> graph TD ALU[ALU] TCU[Timer/Counter] IOP1[I/O Port] IRAM[Internal RAM] IROM[Internal ROM] SP1[Stack Pointer] IOP2[I/O Port] INT[Interrupt Circuits] CLK[Clock Circuit] ALU --- TCU TCU --- IOP1 IOP1 --- IRAM IRAM --- IROM IROM --- SP1 SP1 --- CLK CLK --- INT IOP2 --- INT IOP2 --- PC[Program Counter] </pre>

Fig : Block diagram of Microprocessor

Fig: Block diagram of Microcontroller

Microprocessor Contains ALU, general purpose registers, stack pointer, program counter, clock timing CK & interrupt CK

Microcontroller Contains the circuitry of microprocessor & in addition it has built in ROM, RAM, I/O devices, timers & counters.



3)	It has many instructions to move data b/w memory & CPU	It has one or two instructions to move data b/w memory & CPU.
4)	It has one or two bit handling instructions.	It has many bit handling instructions. {ex:- CLR C, SETB P1.0 etc.,}
5)	Less number of pins are - multifunctioned.	More number of pins are - multifunctioned.
6)	It has Single memory map for data & code (program)	It has separate memory map for data & code (program).
7)	Access time for memory & I/O devices are more.	Less access time for built-in memory & I/O devices.
8)	Microprocessor based System requires more hardware	Microcontroller based System requires less hardware reducing PCB size & increasing the reliability.
9)	Microprocessor based System is more flexible in design point of view.	Less flexible in design point of view.
	(CR)	(OR)
	Designer can decide the amount of ROM, RAM etc., to be connected	Fixed amount of ROM, RAM, I/O ports on chip.
10)	Expansive applications	Applications in which cost, space & power are critical.
11)	versatile	Not versatile.



12)

Large number of instructions with flexible addressing modes

Limited number of instructions with few addressing modes.

Computer architecture OR processor architectures :-

- * Every processor needs to store the code (instructions) & also the data. Depending on how these are stored in memory & how the memory is accessed, the processor architectures are classified into
 - 1) VON- NEUMANN & Princeton architecture
 - 2) HAR VARD architecture.

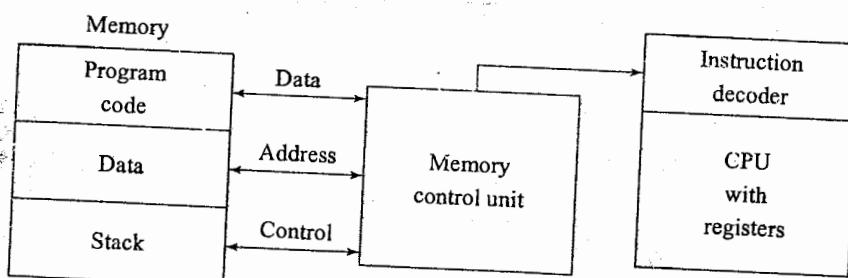


Figure Block diagram of Von Neumann Architecture.

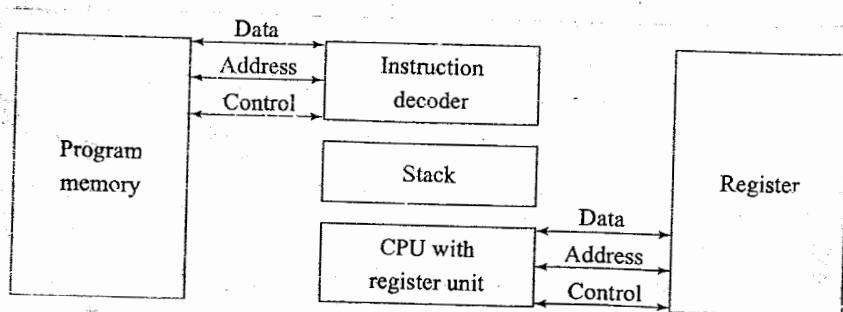


Figure Block diagram of Harvard Architecture.



- 1) Distinguish Harvard & Princeton architecture with diagrams.

Jan-07, 6M

- 2) Explain the difference b/w Harvard & von-Neumann architecture

Jan-09, 4M

SL NO	Von-Neumann & Princeton architecture.	Harvard architecture
1)	Block diagram	Block diagram.
2)	It uses Single memory Space for both Instructions & data. It is also called Shared program Computer.	It has Separate program - memory & data memory.
3)	It is not possible to fetch the instruction Code & data.	Instruction Code & data can be fetched Simultaneously.
4)	Execution of Instruction takes more Instruction (machine) cycle	Execution of Instruction take less instruction (machine) cycle.
5)	Uses CISC processor	Uses RISC processor.
6)	Main feature is "pre-fetching"	Main feature is instruction "parallelism"
7)	The computers based on Princeton architecture are also known as control flow & control driven computers.	They are also called as data flow & data driven processors.



SL No	RISC	CISC
4	Multiple Register Set	Single Register Set
5	Few addressing modes & most instructions have Register to Register addressing mode.	Many addressing modes.
6	Fixed Format Instructions	variable Format Instructions
7	Highly pipelined	Less pipelined
8	Conditional Jump can be based on a bit anywhere in memory.	Conditional Jump is usually based on the Status Register bit.
9	Complexity is in the Compiler	Complexity is in the micro-program.
10	Complex addressing modes are synthesized in Software	Supports complex addressing modes.
11	e.g.: PIC Microcontroller Series	e.g.: 8085, 8086, MC6800, Z-80 & 8051 microcontroller.

Microcontroller :-

Microcontroller contains the circuitry of - microprocessor & in addition it has built in ROM, RAM, I/O devices, timers & counters.



SL No	von-Neumann & Princeton	Harvard
8)	The largest advantage is that it Simplifies the chip design - because only one memory is accessed.	Chip design is complex because of separate memory.
9)	eg:- 8085, 8086, MC6800 etc	eg:- General purpose microcontrollers, Special DSP's.

1) Define microcontroller & differentiate the RISC & CISC processors.

July - 09, 5M

2) Explain the difference b/w RISC & CISC processors.

Jan - 09, 4M

3) What are the advantages of RISC & CISC processor architecture.

Jan - 07, 6M

SL No	RISC	CISC
1)	Simple Instructions taking one cycle	Complex instructions taking multiple cycles.
2)	Very few instructions refer memory	Most of the instructions may refer memory.
3)	only few Instructions	Many Instructions.



Toys, Washing machines, Microwave oven, Remote Controls etc.,

3) OFFICE:-

Security Systems, Fax machine, copiers, printers, paging & Intercom.

3) Auto :-

Engine control, Air bag, Security System centre locking

4) others :-

Cellular phones, Traffic controller, Musical Instruments, Camera, Electronic voting Machines etc.,

Microcontroller Survey:-

i) 4-bit microcontroller :-

* CPU can handle only 4-bit of data at a time.

* These microcontrollers are introduced 1st & are still used in very small appliances.

Applications :- In Toys.

Eg :-

Sl No	Manufacturer	Model	RAM	ROM
1	Hitachi	HMCS40	32-bytes	512 bytes
2	Toshiba	TLCS47	128 bytes	2K-bytes



* What criteria do designers consider in choosing microcontroller?

July-08, 6M

There are a wide variety of microcontrollers available in the market. Program written for one (microcontroller) will not run on others. The choice of the microcontroller is determined by three parameters:

- 1) It must perform the required task efficiently & effectively
 - i) Speed
 - ii) Amount of RAM & ROM on chip
 - iii) Power consumption
 - iv) The number of I/O pins & the timer on the chip
 - v) Cost per unit
 - vi) Ease of upgrading
 - vii) Packaging - The number of pins & the packaging format.

This determines the required space & assembly layout.

- 2) Availability of Software development tools such as compilers, assemblers & debuggers.
- 3) Availability & reliable source for the microcontroller.

1) Mention the applications of (8051) microcontroller?

July-08, 6M

July-06, 4M

Jan-08, 6M

- I) Home :-
- 1) Answering machine
 - 2) TV's
 - 3) VCR
 - 4) Camcorder
 - 5) video games



ii) 8-bit Microcontroller :-

- * CPU can handle 8-bits at a time
- * 8-bit size of data is proven to be very useful data size - because ASCII data is stored in 8-bit formats. This makes 8-bits a good choice for data communication.
- * Most of memory IC's are arranged in 8-bit configuration, which can be interfaced easily to data buses of 8-bits.

Applications :-

Variety of applications that involve limited calculations & simple control operations such as washing machines, TV etc.,.

Eg :-

Manufacturer	Model	No. of pins	RAM	ROM
Intel	8051	40	128	4K-byte
Intel	8052	40	256	8K-byte

iii) 16-bit Microcontroller :-

- * CPU can handle only two bytes at a time.
- * Designed for high speed/high performance applications. These provide large program & data memory spaces for more flexible I/O capabilities, greater speed & less cost than any previous microcontrollers.

Manufacturer	Model	No. of Pins	RAM	ROM
Intel	80C196	64	1-Kbytes	32-Kbytes
Hitachi	H8/532	84	232-Kbytes	8 - Kbytes



32-bit Microcontroller :-

- * CPU can handle 32-bit data at a time
- * Designed for applications such as robotics control, highly intelligent instrumentation, image processing & other high end control systems.

eg:- Intel 80960
ARM processor.

- 1) List the Salient features of 8051 microcontroller
- 2) List the Specific features of 8051 microcontroller

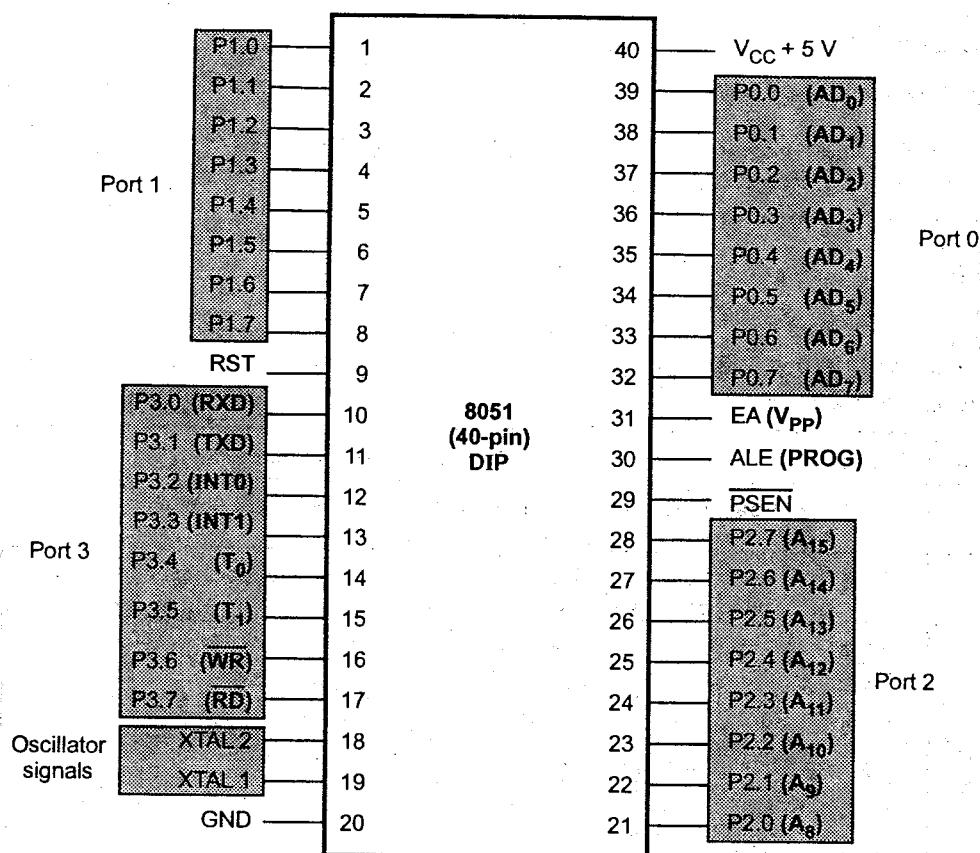
Jan-09, 6M
Jan-07, 5M

The Salient features of 8051 microcontroller are :-

- 1) 8-bit CPU
- 2) Internal ROM of 4-Kbytes
- 3) Internal RAM of 128-bytes
- 4) 32-I/O pins
- 5) Two 16-bit Timers/Counters (T_0 & T_1)
- 6) 8-bit Stack pointer (SP)
- 7) 8-bit program Status Word (PSW)
- 8) 16-bit program Counter (PC) & Data pointer (DPTR)
- 9) 6 Interrupt Sources with priority levels
- 10) Full duplex Serial data Transmitter/ Receiver
- 11) on-Chip oscillator circuits



8051 Pin diagram:-



Pins 1-8 : Port 1 :-

Each of these pins can be configured as I/p or o/p pins.

Pin 9: RST (Reset) :-

It is a active high Signal, When a Pulse (Square Wave) is applied to this pin, microcontroller will terminate all its activities & Reset.

Program Counter is loaded with 0000.

Pins 10-17 : Port 3 :-

Each of these pins can be configured as I/p or o/p pins.

Pins 10 & 11: Rx_D & Tx_D :-

8051 has Serial data Communication Circuits that use 'SBUF' register to hold the data & 'SCON' to Control the data communication.

- * The data is Transmitted out of 8051 through the Tx_D line.
- * The data is Received by 8051 through the Rx_D line.

Pin 12: INT₀ :-

Pin 13: INT₁ :- } } Interrupt 0 & Interrupt 1 are two Interrupt Pins that are triggered by external circuits.

Pins 14 & 15: T₀ & T₁ :- The 8051 has two 16-bit Timers/Counters.

T₀ → Timer₀ Register (16-bit)

T₁ → Timer₁ Register (16-bit)



- * They can be used either as Timers to generate a time delay or as Counters to count events happening outside the microcontroller.

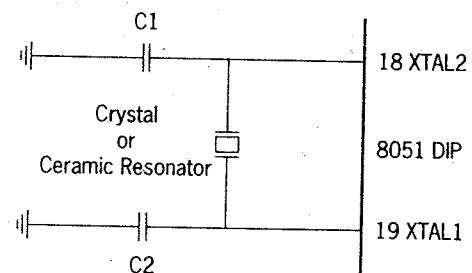
Each 16-bit register can be accessed as two separate 8-bit registers.

Pins 16 & 17: RD & WR :- These are active low pins.

When $\overline{RD} = 0$, microcontroller reads the data from external RAM.

When $\overline{WR} = 0$, microcontroller writes the data into external RAM.

Pins 18 & 19: XTAL2 & XTAL1 :-



Crystal or Ceramic Resonator Oscillator Circuit

- * The 8051 has an on-chip oscillator but requires an external clock to run it. A Quartz Crystal oscillator is connected to I/P's XTAL1 & XTAL2 with two capacitors having values 30 pF.
- * If an external frequency (from AFO) have to be applied, then it must be applied b/w XTAL1 & GND. XTAL2 must be left open.
- * oscillator frequency may vary from 10MHz to 40MHz.

Pin 20: VSS:- It is a ground pin.



Pins 21-28: Port 2 :-

If external memory is not used, these pins can be used as I/p's & O/p's.

- * If external memory is used then the higher address i.e. A₈-A₁₅ will appear on this port.

Pin 29: PSEN (Program Store Enable) :-

If the memory access is for the byte of program code in the ROM, then PSEN Signal goes low & the data byte from the ROM is placed on the data bus.

Pin 30: ALE (Address Latch Enable) :-

When ALE = 1, Port 0 is providing lower order address. (A₀-A₇)
When ALE = 0, Port 0 is used as data lines.

Pin 31: EA (External Access) :-

This Pin is used whenever external memory is used.

- * When EA is connected to Vcc i.e. EA = 1, Code is stored in internal ROM. The program is fetched from address location 0000 to OFFFh.
- * When EA is connected to GND i.e. EA = 0, Code is stored in external ROM. The program is fetched from address location 0000 to FFFFh.

Pins 32-39: Port 0 :-

If external memory is not used, then these pins can be used as I/p's & O/p's.



* If external memory is used then the lower address i.e. A₀-A₇ will appear on this port.

Pin 40: VCC :- DC power Supply +5V is connected to this pin.

1) Explain the function of the following pins of 8051

- i) EA
- ii) ALE
- iii) RST
- iv) PSEN

June-07, 8M

8051 Microcontroller :- It is a 8-bit microcontroller which has got RAM, ROM, 2 timers, 1 Serial port & 4 other ports on a Single Chip.

Features of 8051 :-

Features of the 8051

Feature	Quantity
ROM	4K bytes
RAM	128 bytes
Timer	2
I/O pins	32
Serial port	1
Interrupt sources	6

Table Comparison of 8051 Family Members

Feature	8051	8052	8031
ROM (on-chip program space in bytes)	4K	8K	0K
RAM (bytes)	128	256	128
Timers	2	3	2
I/O pins	32	32	32
Serial port	1	1	1
Interrupt sources	6	8	6

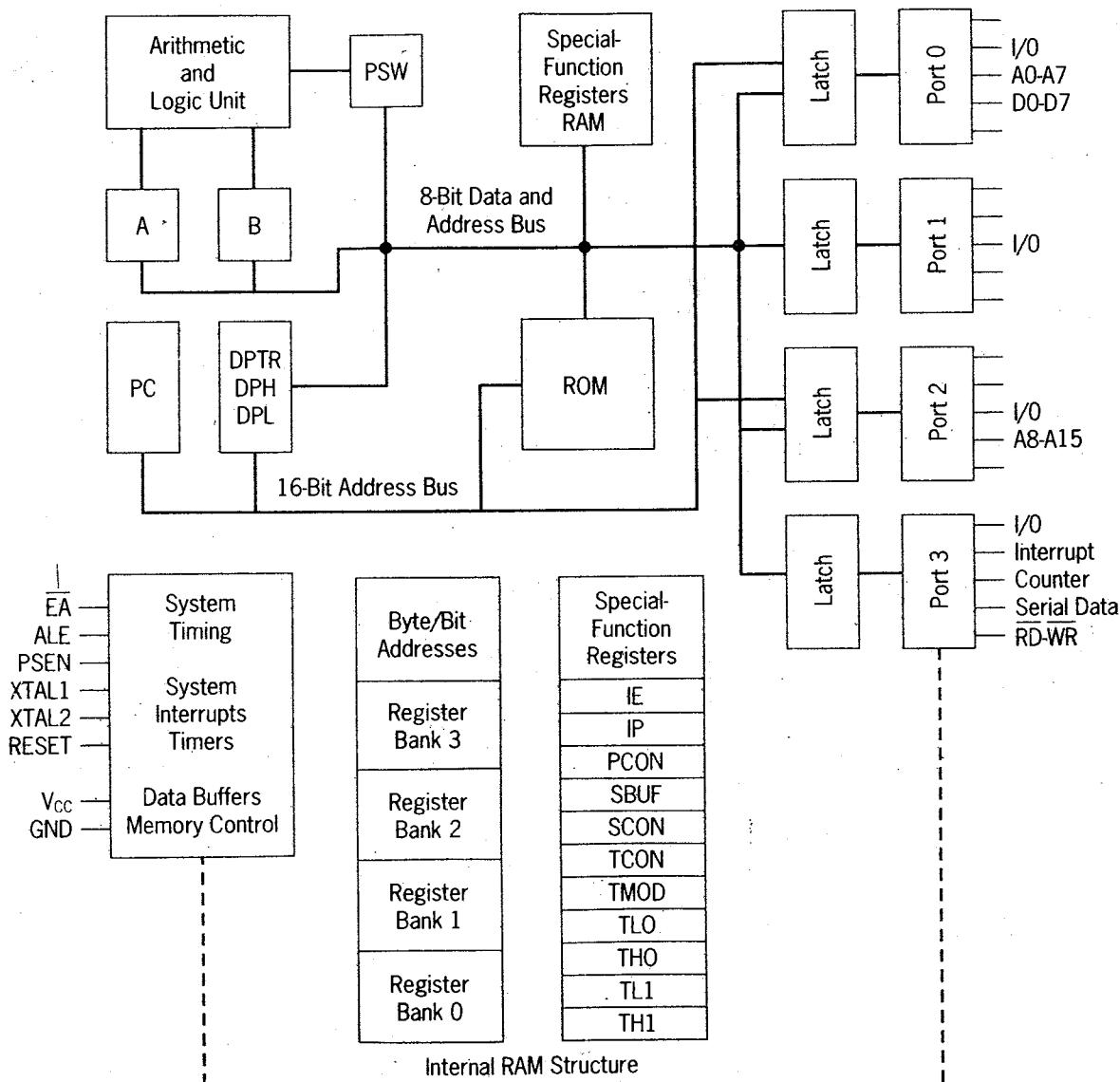


8051 Architecture :-

July - 09, 10M
June - 07, 12M
Model , 10 M

* With the neat block diagram, explain the architecture of 8051.

8051 Block Diagram



Internal RAM Structure



Central processing unit (cpu) :-

- * The 8051 CPU consists of 8-bit Arithmetic & Logic unit with associated registers like A, B, PSW, Sp, 16-bit program counter & "Data pointer registers" (DPTR).
 - * The 8051's ALU can perform Arithmetic & Logic functions on 8-bit variables. The arithmetic unit performs addition, subtraction, multiplication & division.
 - * The Logic unit can perform logical operations such as AND, OR & EX-OR, as well as rotate, clear & complement.

Internal RAM :-

Byte Address		Byte Address	
Bank 3	1F	R7	
	1E	R6	
	1D	R5	
	1C	R4	
	1B	R3	
	1A	R2	
	19	R1	
	18	R0	
	17	R7	
	16	R6	
	15	R5	
	14	R4	
	13	R3	
	12	R2	
	11	R1	
	10	R0	
Bank 2	0F	R7	
	0E	R6	
	0D	R5	
	0C	R4	
	0B	R3	
	0A	R2	
	09	R1	
	08	R0	
	07	R7	
	06	R6	
	05	R5	
	04	R4	
	03	R3	
	02	R2	
	01	R1	
	00	R0	
Bank 1	7F	78	
	77	70	
	6F	68	
	67	60	
	5F	58	
	57	50	
	4F	48	
	47	40	
	3F	38	
	37	30	
	2F	28	
	27	20	
	1F	18	
	17	10	
	0F	08	
	07	00	
Bank 0		7	0
Working Registers		Bit Addressable	
		General Purpose	



- * The 8051 has 128-byte Internal RAM. The Internal RAM of 8051 is organized into three distinct areas
 - 1) Working Registers
 - 2) Bit Addressable Registers
 - 3) General purpose Registers.

Working Register :-

The 1st 32-bytes from address 00h to 1Fh of Internal RAM constitutes 32 Working Registers i.e.

Bank 0 → 8 registers (R₀-R₇) : 00h to 07h

Bank 1 → 8 registers (R₀-R₇) : 08h to 0Fh

Bank 2 → 8 registers (R₀-R₇) : 10h to 17h

Bank 3 → 8 registers (R₀-R₇) : 18h to 1Fh

* Bits RS₀ & RS₁ in the PSW determine which bank of registers is currently in use.

* When 8051 is RESET, the BANK0 is Selected.

Bit addressable register :-

* The 8051 provides 16-bytes of a bit addressable area. It occupies RAM area from 20h to 2Fh, forming a total of 128 addressable bits. ($16 \text{ bytes} \times 8\text{-bits} = 128\text{-bits}$)

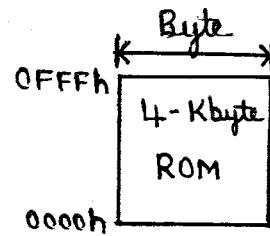
General purpose Registers :-

The RAM area above bit addressable area from 30h to 7Fh is called general purpose RAM. It is - addressable by byte.



INTERNAL ROM :-

- * The 8051 has 4-Kbytes of Internal ROM
With Address Space from 0000h to 0FFFh.
- * The Program address higher than 0FFFh, which exceeds the internal ROM Capacity will cause the 8051 to automatically fetch Code bytes from external program memory.



A Register (Accumulator) :-

- * Accumulator is a 8-bit Register & is widely used for many operations like addition, Subtraction, multiplication, division & boolean bit manipulations.
- * The A-Register is also used for all data transfers b/w the 8051 and any external memory.

B-Register :-

The B-Register is used with the A-Register for multiplication & division operations & has no other function other than as a location where data may be stored.

STACK pointer (8-bit) :-

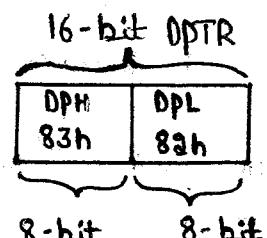
- * The Stack refers to an area of Internal RAM used by the CPU to Store & Retrieve (take back) data quickly.
- * The register used to access the Stack is called the Stack pointer (SP) register.
- * The Stack pointer register is used by the 8051 to hold an -



Internal RAM address that is called the Top of the Stack.

- * When 8051 is RESET, the Sp is Set to 07h.
- * The Storing of a CPU register in the Stack is called a PUSH.
- * Loading the Contents of the Stack back into the CPU register is called a POP.

Data pointer (DPTR) :-



- * DPTR is a 16-bit register, which holds a 16-bit address.
 - * DPTR can be splitted into two parts :
 - DPH → Data pointer high byte having Internal address 83h.
 - DPL → Data pointer low byte having Internal address 83h.
- { The DPTR does not have a Single Internal address }

Program Counter (PC) :-

- * PC is a 16-bit register which holds the address of the next instruction to be executed. The PC is automatically incremented after every instruction byte is fetched.
- * The 8051 has 16-bit PC hence it can address upto 2^{16} bytes
i.e. $2^{16} = 64$ K-bytes of memory.

{ PC is the only register that does not have an Internal address. }

I/P - O/P ports (I/O ports) :-

The 8051 has 32 I/O pins configured as Four 8-bit parallel ports
i.e. P0, P1, P2 & P3.



- * All Four ports are bi-directional i.e. each pin can be configured as I/p or O/p under Software Control.

Timers & Counters :-

- * 8051 use two 16-bit registers namely T_0 & T_1 either for Timer or Counter.
- * The two Timers or Counters are divided into Two 8-bit Registers called Timer Low (TL_0 , TL_1) and Timer High (TH_0 & TH_1).

Program Status Word (PSW) & Flag Register :-

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

Carry Flag (CY) :-

After performing arithmetic & logic operation if there is a carryout from the MSB (D_7 -bit) then $CY = 1$, otherwise $CY = 0$.

Auxiliary Carry Flag (AC) :-

After performing arithmetic & logic operation if a carry is generated from D_3 to D_4 bit then $AC = 1$, otherwise $AC = 0$.

RS1 & RS0 :- Register Bank Selection.

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



overflow Flag (ov) :-

ov flag is Set to 1 if either of the following two conditions occurs:

- 1) There is a Carry from D₆ to D₇ but No carry out of D₇ (cy=0)
- 2) There is a carry out from D₇ bit (cy=1) but No carry from D₆ to D₇ - bit.

Parity Flag (P) :-

parity flag indicates the number of 1's present in the accumulator.

- * If the number of 1's in the accumulator is odd then P=1.
- * If the number of 1's in the accumulator is Even then P=0.

Special Function Registers (SFR) :-

The operations of 8051 are done by a group of specific internal registers, each called a Special Function Register (SFR).

- 1) Explain the Significance of processor Status word. Briefly discuss PSW register of 8051.

Model, 4M

July - 06, 6M

- * The PSW is an 8-bit register. It is also called as Flag Register. out of these only 6-bits of PSW registers are used & 2-bits are unused. (math flag)
- * 4 flags are called Conditional Flags because these 4 flags - Indicates Some Conditions that results after an Instruction is executed. i.e. Carry Flag, Auxiliary carry flag, parity flag & overflow flag.



- * RS0 & RS1 are used to change the bank register.
- * The two unused bits are called user-definable flags

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

Carry Flag (cy) :-

After performing arithmetic & logic operation if there is a carryout from the MSB (D_7 -bit) then $CY=1$, otherwise $CY=0$.

- * carry flag 'CY' can be Set to 1 or 0 directly by an instruction such as "SETB C" & "CLR C"

{ Where SETB C \rightarrow Set bit carry.
 CLR C \rightarrow clear carry. }

Auxiliary carry flag (AC) :-

After performing arithmetic & logic operation if a carry is generated from D_3 to D_4 bit then $AC=1$, otherwise $AC=0$.

RS1 & RS0 :- Register Bank Select.

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



Overflow Flag (OV) :-

* In 8-bit Signed number operations, OV is Set to 1 if either of the following two conditions occurs.

- 1) There is a carry from D₆ to D₇ but No carry out of D₇ ($CY=0$)
- 2) There is a carryout from D₇ ($CY=1$) but No carry from D₆ to D₇.

Eg:-	-128	\longrightarrow	80h	=	1000 0000
	- 2	\longrightarrow	FEh	=	1111 1110
CPU Result	$\rightarrow +126$		7Eh		0111 1110

* OV = 1 because carryout from D₇ & No carry from D₆ to D₇.

* Result is Wrong because CPU Shows answer as +126_d instead of -130_d.

Parity Flag (P) :-

Parity flag indicates the number of 1's present in the accumulator.

* If the number of 1's in the accumulator is odd then P=1.

* If the number of 1's in the accumulator is even then P=0.

Eg:-

* If A = 00011000 \rightarrow P=0 \because Number of 1's are even

* If A = 10001100 \rightarrow P=1 \because Number of 1's are odd



problems :-

- 1) Show the contents of the PSW register after the execution of the following instructions.

MOV A, #9CH

ADD A, #64H

Sol :-

$$\begin{array}{rcl}
 9C H & \longrightarrow & \begin{array}{c} \textcircled{1} \\ \dots \\ 1001\ 1100 \end{array} \\
 + 64 H & \longrightarrow & \begin{array}{c} \textcircled{1} \\ 0110\ 0100 \\ \hline \textcircled{1} \ 0000\ 0000 \end{array}
 \end{array}$$

- ⇒ CY = 1 : Since there is a carry beyond the D₇-bit
 ⇒ AC = 1 : Since there is a carry from D₃ to D₄ bit
 ⇒ P = 0 : Since the accumulator has an even number of 1's i.e. it has zero 1's.
 ⇒ RS0 = 0 : } By default Bank 0 is Selected.
 ⇒ RS1 = 0 : }
 ⇒ F₀ = 0 : unused, hence 0.
 ⇒ OV = 0 : Since there is carry from D₆ to D₇ & a carryout from D₇-bit.
 ⇒ PSW.1 = 0 : Not used, hence 0
 ∴ The contents of PSW is 11000000 i.e. C0h.

- 2) Show the contents of the PSW register after the addition of BFH & 1BH in the following instruction.

MOV A, #0BFH

ADD A, #1BH

Sol :-

$$\begin{array}{rcl}
 BF & \longrightarrow & \begin{array}{c} \textcircled{1} \\ \dots \\ 1011\ 1111 \end{array} \\
 + 1B & \longrightarrow & \begin{array}{c} \textcircled{1} \\ 0001\ 1011 \\ \hline 1101\ 1010 \end{array}
 \end{array}$$



The bits of PSW are as follows :

- 1) CY=0 : Since there is no carry beyond the D₇-bit
- 2) AC=1 : Since there is a carry from the D₃ to the D₄ bit.
- 3) F₀=0 : Unused, hence 0.
- 4) RS1=0 : By default, Bank0 is Selected
- 5) RS0=0 : By default, Bank0 is Selected.
- 6) OV=0 : Since there is NO carry from D₆ to D₇
- 7) PSW.1=0 : Not used, hence 0.
- 8) P=1 : Since there is an odd number of 1's in the accumulator

* The contents of the PSW is thus 01000001 i.e. 41h

Special Function Register :- (SFR)

July - 08, 5M

July - 07, 5M

Name	Function	Internal RAM address (HEX)
A	Accumulator	0E0
B	Arithmetic	0F0
DPH	Addressing external memory	83
DPL	Addressing external memory	82
IE	Interrupt enable control	0A8
IP	Interrupt priority	0B8
P0	Input/output port latch	80
P1	Input/output port latch	90
P2	Input/output port latch	A0
P3	Input/output port latch	0B0
PCON	Power control	87
PSW	Program status word	0D0
SCON	Serial port control	98
SBUF	Serial port data buffer	99
SP	Stack pointer	81
TMOD	Timer/counter mode control	89
TCON	Timer/counter control	88
TL0	Timer 0 low byte	8A
TH0	Timer 0 high byte	8C
TL1	Timer 1 low byte	8B
TH1	Timer 1 high byte	8D

* The 8051 operations that do not use the internal 128-byte RAM - address from 00h to 7Fh.

- * The operations of 8051 are done by a group of specific internal registers, each called a Special Function Register "SFR".
- * The SFR's may be accessed by their names & by using addresses from 80h to FFh.
- * Not all the addresses from 80h to FFh are used for SFR's. If we attempt to use an address that is not defined & empty, results in unpredictable results.

NOTE :- The program counter (pc) is not part of the SFR & has No Internal RAM address.

- * List out the different bit addressable SFR's available in 8051.

Jan - 06, 4M

Bit addressable SFR's available in 8051 are shown below

Symbol	Name	Address
* ACC	accumulator	0E0H
* B	B Register	0F0H
* PSW	Program Status Word	0D0H
* P0	Port 0	80H
* P1	Port 1	90H
* P2	Port 2	0A0H
* P3	Port 3	0B0H
* IP	Interrupt Priority Control	0B8H
* IE	Interrupt Enable Control	0A8H
* TCON	Timmer/Counter Control	88H
* SCON	Serial Control	98H



NOTE :- {Don't remember}

Byte address	Bit address	
FF		
F0	F7 F6 F5 F4 F3 F2 F1 F0	B
E0	E7 E6 E5 E4 E3 E2 E1 E0	ACC
D0	D7 D6 D5 D4 D3 D2 D1 D0	PSW
B8	-- -- BC BB BA B9 B8	IP
B0	B7 B6 B5 B4 B3 B2 B1 B0	P3
A8	AF -- AC AB AA A9 A8	IE
A0	A7 A6 A5 A4 A3 A2 A1 A0	P2
99	not bit-addressable	SBUF
98	9F 9E 9D 9C 9B 9A 99 98	SCON
90	97 96 95 94 93 92 91 90	P1
8D	not bit-addressable	TH1
8C	not bit-addressable	TH0
8B	not bit-addressable	TL1
8A	not bit-addressable	TL0
89	not bit-addressable	TMOD
88	8F 8E 8D 8C 8B 8A 89 88	TCON
87	not bit-addressable	PCON
83	not bit-addressable	DPH
82	not bit-addressable	DPL
81	not bit-addressable	SP
80	87 86 85 84 83 82 81 80	P0
Special Function Registers		

Figure SFR RAM Address (Byte and Bit)



1) Explain the memory organization in 8051 microcontroller

Jan - 09, 8M

2) With neat diagrams, give the details of program memory & data memory of 8051.

Model, 8M

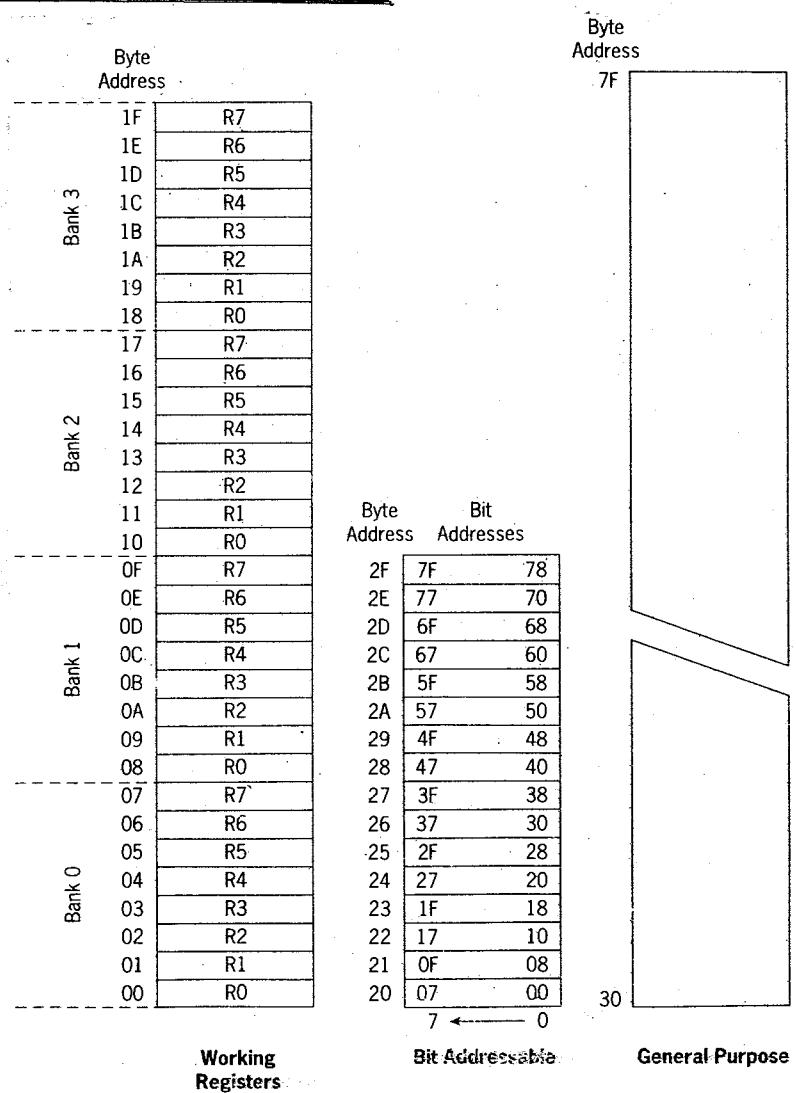
3) Draw & explain the memory structure of 8051 6M

4) Explain the Internal RAM organization of 8051 microcontroller.

* The memory organization of 8051 microcontroller is classified into two types:

- 1) Data Memory (RAM)
- 2) Program Memory (ROM)

1) Data Memory or Internal RAM :-



- * The 8051 has 128-byte Internal RAM. The internal RAM of 8051 is organized into three distinct areas.
 - 1) Working registers
 - 2) Bit addressable registers &
 - 3) General purpose registers.

1) Working Register :-

- * The 1st 32 bytes from address 00h to 1Fh of Internal RAM constitutes 32 Working Registers i.e.

SL No	Bank	No. of registers	Register Name	Address Range
1)	Bank 0	8	R ₀ -R ₇	00h to 07h
2)	Bank 1	8	R ₀ -R ₇	08h to 0Fh
3)	Bank 2	8	R ₀ -R ₇	10h to 17h
4)	Bank 3	8	R ₀ -R ₇	18h to 1Fh

- * Each register can be addressed by name & by its RAM address.
- * only one register bank is in use at a time.
- * Bits RS0 & RS1 in the PSW determine which bank of Registers is currently in use.
- * When 8051 is RESET, the BANK 0 is Selected.

2) Bit addressable register :-

- * The 8051 provides 16-bytes of a bit addressable area. It occupies RAM area from 20h to 2Fh, forming a total of 128 addressable bits (i.e 16 bytes \times 8-bits = 128 bits)
- * The addressable bit may be specified by its bit address of 00h to 7Fh

8) 8-bit may form any byte address from 20h to 2Fh.

{ eg:-

* Bit address 4Fh refers bit 7 of the byte address 29h.

* Bit address 4Eh refers bit 6 of the byte address 29h.

* Mainly used for a binary event such as SWITCH ON, Light OFF etc

}

3) General purpose register:-

The RAM area above bit addressable area from 30h to 7Fh is called general purpose RAM. It is addressable as byte.

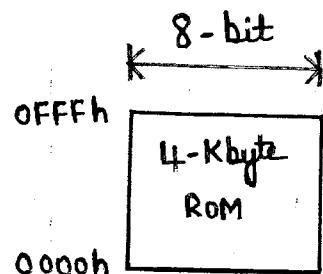
II) Program memory (ROM)

Program memory is classified into :

- 1) Internal ROM
- 2) External ROM

1) Internal ROM :-

- * The 8051 has 4-Kbyte of Internal ROM with address range from 0000h to OFFFh.
- * The 8051 has control pins such as PSEN (program store enable) & EA (External access) that determines whether external memory is accessed or Internal memory is accessed.
- * If EA pin is connected to VCC (usually +5V), then the program memory accessed by the chip is Internal memory.



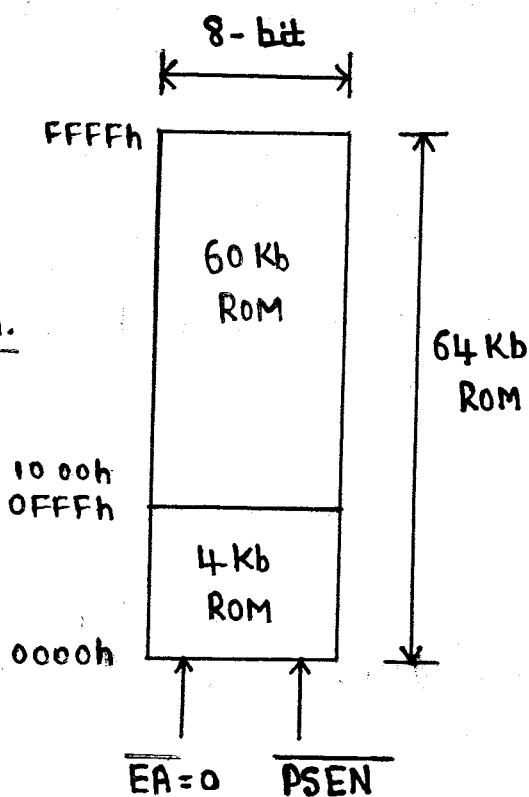
⇒ External memory :-

* When EA Pin is Connected to VSS (0V or ground) then the 8051 can access external memory starting from address 0000h to FFFFh.

NOTE :-

ON-Chip ROM address : 0000h to 0FFFh

OFF-Chip ROM Address : 0000h to FFFFh



Instructions :-

⇒ PUSH direct address :-

Function : Push onto Stack

Description : The Stack pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the Stack pointer.

Flags affected : None

Bytes : 2

Cycles : 2

operation : $(SP) \leftarrow (SP) + 1$



NOTE:- This instruction Supports only direct addressing mode.

∴ The instructions Such as

"PUSH A" & "PUSH R_n" are illegal instructions.

Eg:-

▷ PUSH 0E0h

Where E0h is the RAM address belonging to register A.

▷ PUSH 03h

Where 03h is the RAM address of R₃ of Bank 0.

a) Pop direct address :-

Function : Pop from the Stack.

Description : This Copies the byte pointed to by Stack pointer (sp) to the location where direct address is indicated & decrements Sp by 1.

Flags affected : None

Bytes : 2

Cycles : 2

operation : $(Sp) \leftarrow (Sp) - 1$

NOTE:-

This instruction Supports only direct addressing mode.

∴ The instructions Such as

"pop A" & "pop R_n" are illegal instructions.

Eg:-

▷ POP 0E0h

Where E0h is the RAM address belonging to register A.



⇒ pop 03h

Where 03h is the RAM address of R₃ of Bank 0

Stack :-

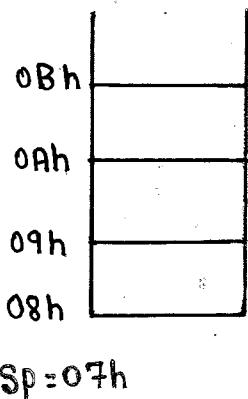
- * Stack refers to an area of internal RAM used by the CPU to Store & Retrievce data (take back) quickly.
- * The Register used to access the Stack is called the Stack pointer (Sp) register.
- * The Stack pointer is a 8-bit register used by the 8051 to hold an internal RAM address that is called the Top of the Stack.
- * When 8051 is RESET, the Sp is Set to 07h
- * When data is to be placed on the Stack, the Sp increments - before Storing data on the Stack i.e. Sp = Sp + 1, so that the Stack grows up as data is stored.
- * As the data is retrieved from the Stack, the byte is read from Stack & then Sp decrements i.e. Sp = Sp - 1 to point to the next available byte of stored data.
- * Storing the data onto the Stack is called a PUSH.
- * Retrieving the Contents of the Stack is called a Pop.
- * RAM location 08h is the 1st location used by the Stack to Store the data.

Eg:- 1) move R₃, #30h
PUSH A

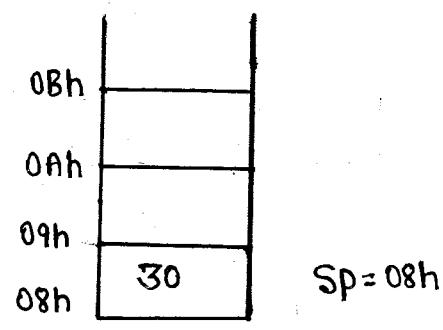
Assume that initially Bank 0 is Selected & Sp = 07h



Before Execution



After Execution

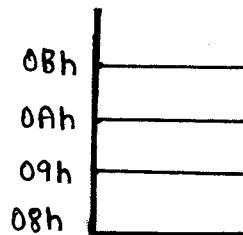


- * Sp is 1st incremented by one i.e. Sp = Sp+1, then the Contents of R₂ is stored in top of Stack i.e. 08h address.

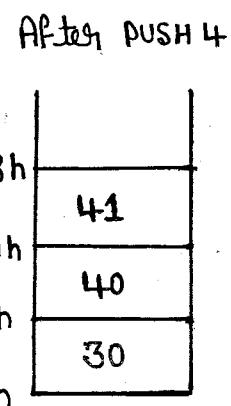
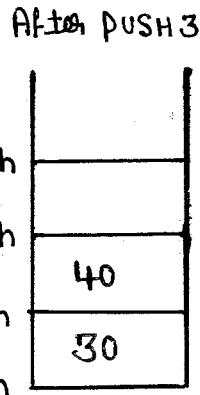
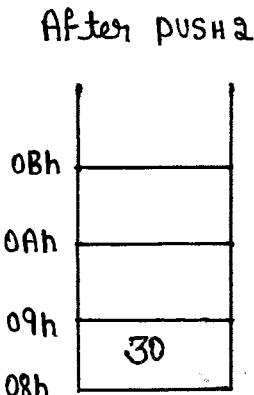
⇒
 move R₂, #30h
 move R₃, #40h
 move R₄, #41h
 PUSH 2
 PUSH 3
 PUSH 4

* Assume Bank 0 is Selected & Sp has initially 07h

Before Execution



After Execution



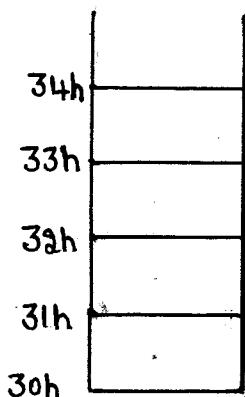
UPPER limit of Stack:-

- * Locations 08h to 1Fh in 8051 RAM can be used for Stack because locations 20h - 2Fh of RAM are reserved for bit addressable memory & must not be used by Stack.
- * If in program, we need more than 24 bytes ($08h \text{ to } 1Fh = 24 \text{ bytes}$) of Stack then we can change SP to point to RAM location 30h - 7Fh. This is done by the instruction "move SP, #xx"

Eg:-

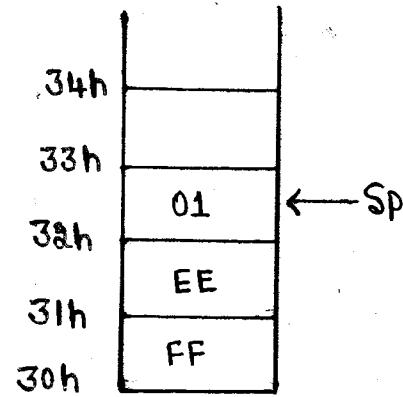
```
    mov SP, #30h ; Now Sp is initialized to 30h
    SETB PSW.3      ; Bank 1 is Selected
    move R0, #0FFh
    move R1, #0EEh
    mov A, #01h
    PUSH 8          ; direct address of R0 (bank 1)
    PUSH 9          ; direct address of R1 (bank 1)
    PUSH 0E0h        ; direct address of A
```

Before Execution



Sp = 30h

After Execution



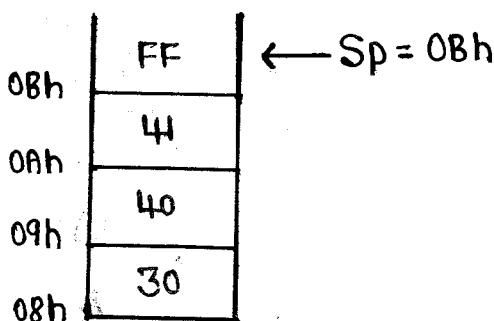
Sp = 32h



Popping from Stack:-

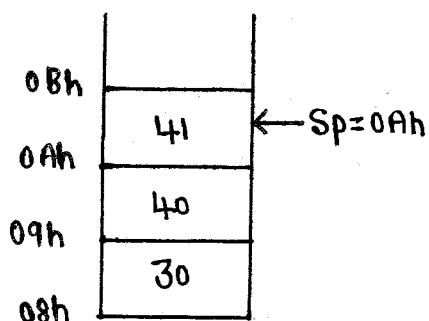
- Eg:- POP 4 ; Pop Stack into R₄ of Banko
 POP 3 ; Pop Stack into R₃ of Banko
 POP 2 ; Pop Stack into R₂ of Banko

Before Execution

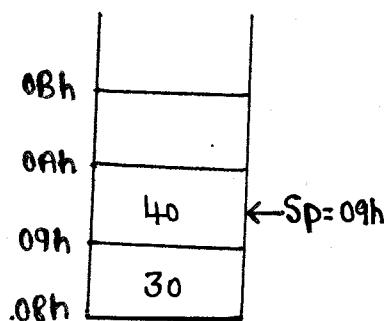


After Execution:-

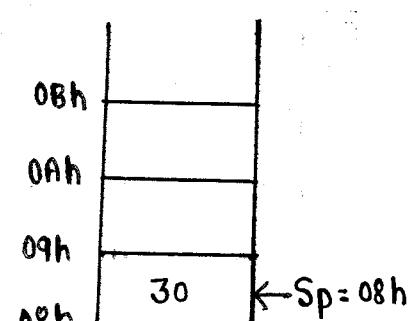
After pop 4 \rightarrow Sp = Sp - 1



After pop 3 \rightarrow Sp = Sp - 1



After pop 2 \rightarrow Sp = Sp - 1



$$(R_4) = FFh$$

$$(R_3) = 41h$$

$$(R_2) = 40h$$



1) What is Stack? Explain with examples the push & pop instructions

Jan-09, 8M

2) Discuss the need for Stack memory in microcontrollers. How Stack is operated in 8051? What is the default location of Stack? How programmer can modify it?

July-06, 7M

3) How Stack operates in 8051 CPU? Mention its relevant instructions.

4) Explain how Stack is implemented in 8051

Jan-08, 5M

Jan-07, 6M

Model, 6M

* Stack refers to an area of internal RAM used by the CPU to store & retrieve data quickly.

* The register used to access the Stack is called the Stack pointer (SP) register.

* The Stack pointer is a 8-bit register used by the 8051 to hold an internal RAM address that is called the Top of the Stack.

* When 8051 is RESET, the SP is set to 07h.

* When data is to be placed on the Stack, the SP increments before Storing data on the Stack i.e. $Sp = Sp + 1$, so that the Stack grows up as data is stored.

* As the data is retrieved from the Stack, the byte is read from Stack & then SP decrements i.e. $Sp = Sp - 1$ to point to the next available byte of stored data.

* Storing of data onto the Stack is called a PUSH

* Retrieving the contents of the Stack is called a POP

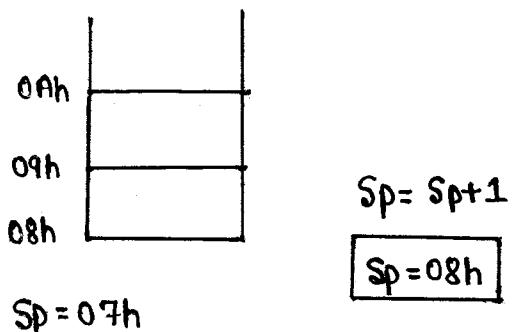
* RAM location 08h is the 1st location used by the Stack to store the data.



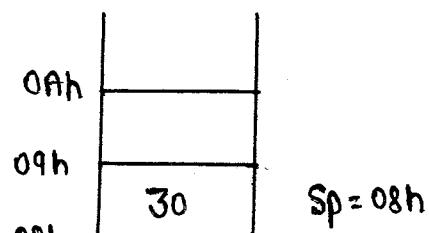
Eg:- \rightarrow move R₂, #30h
PUSH 2

Assume that initially Banko is Selected & Sp = 07h

Before Execution



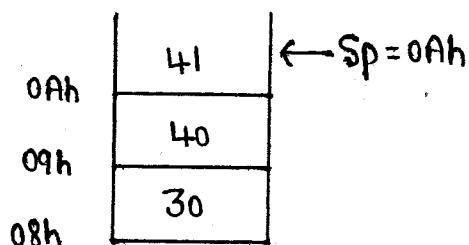
After Execution



* Sp is 1st incremented by one i.e. Sp = Sp+1, then the Contents of R₂ is stored in Top of Stack i.e. 08h address.

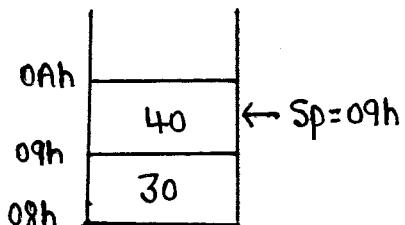
- \Rightarrow pop 3 ; Pop Stack into R₃ of Banko
- pop 2 ; Pop Stack into R₂ of Banko

Before Execution



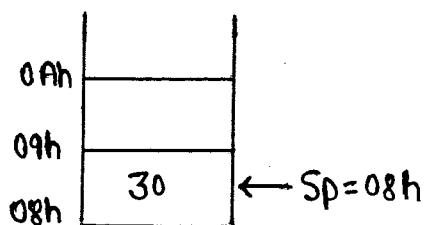
After Execution :

After pop 3 \rightarrow Sp = Sp-1



$$(R_3) = 41h$$

After pop 2 \rightarrow Sp = Sp-1



$$(R_2) = 40h$$



- * The default location of Stack is 07h
- * The programmer can modify the Sp value by using the instruction

mov Sp, # xxh

e.g.: -> mov Sp, # 30h

Now Sp = 30h

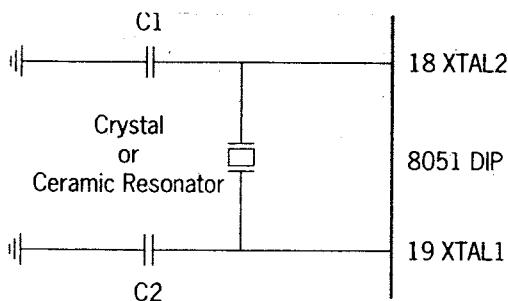
⇒ mov Sp, # 09h

Now Sp = 09h

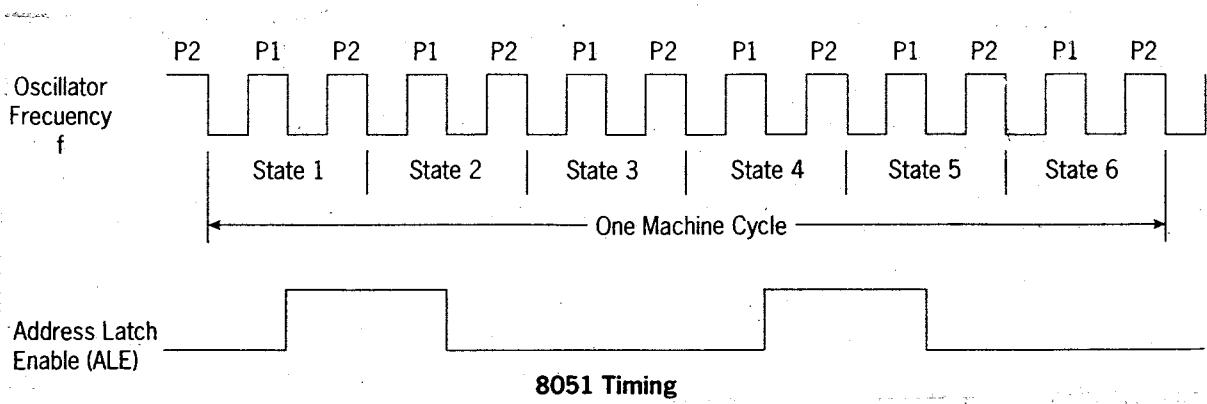
8051 oscillator & Clock:-

- 1) Explain the oscillator circuit & timing of 8051 microcontroller
Jan - 07, 6M
- 2) Explain with neat CKT diagram how the clock circuit works
Jan - 08, 4M

- * All internal operations of the 8051 are synchronized by the clock pulse. These clock pulses are generated by using oscillator CKT.
- * The 8051 provides XTAL1 & XTAL2 pins for connecting a resonant network to form an oscillator as shown in Fig ①.



- * The oscillator circuit consists of Quartz Crystal & capacitors.
- * The Crystal frequency is the basic internal clock frequency of the microcontroller.
- * The minimum & maximum operating frequencies for 8051 are typically 1-MHz to 16-MHz respectively.



- * In 8051 one machine cycle consists of 6 States numbered S₁ to S₆
i.e. $\boxed{\text{one Machine cycle} = 6 \text{ States}}$
- * Each State consists of two oscillator pulses.
i.e. $\boxed{\text{one State} = 2 \text{ oscillator pulses}}$
- * The machine cycle is defined as the smallest interval of time needed to execute (accomplish) any simple instruction.
- * Instructions may require one, two or four machine cycles to execute any instruction depending on the type of instructions.
- * When microcontroller is RESET, instructions are fetched & executed by microcontroller automatically beginning with the instruction located at ROM memory address 0000h.
- * Time needed to execute an instruction is calculated as:

$$T_{\text{inst}} = \frac{C \times I \times d}{\text{Crystal Frequency}}$$



Where

' T_{int} ' is the time for instruction to be executed

'C' is the number of machine cycles &

'f' is the crystal frequency.

- * In Fig ①, there are two ALE pulses per machine cycle. These ALE pulses are used for external memory access.
- * Instructions which are two byte long can be fetched & executed in one machine cycle.
- * Single byte instructions are not executed in a half machine cycle, however Single byte instructions "throw-away" the 2nd byte (which is the 1st byte of the next instruction)
- * The next instruction is then fetched in the following machine cycle. (i.e. in next machine cycle)

1) Calculate the machine cycle if the crystal frequency is

i) 16 MHz ii) 12 MHz

Sol:- i) Time period for one pulse = $\frac{1}{f} = \frac{1}{16 \times 10^6} = 0.0625 \mu\text{sec}$

one machine cycle = 12 pulses

\therefore one machine cycle time = $0.0625 \mu\text{sec} \times 12 = 0.75 \mu\text{sec}$

ii) Time period for one pulse = $\frac{1}{f} = \frac{1}{12 \times 10^6} = 0.0833 \mu\text{sec}$

one machine cycle = 12 pulses

\therefore one machine cycle time = $0.0833 \mu\text{sec} \times 12 = 1 \mu\text{sec}$



Q) In an 8051 System, driven by 11.0592 MHz clock, find the time taken for an instruction which takes 4 machine cycles.

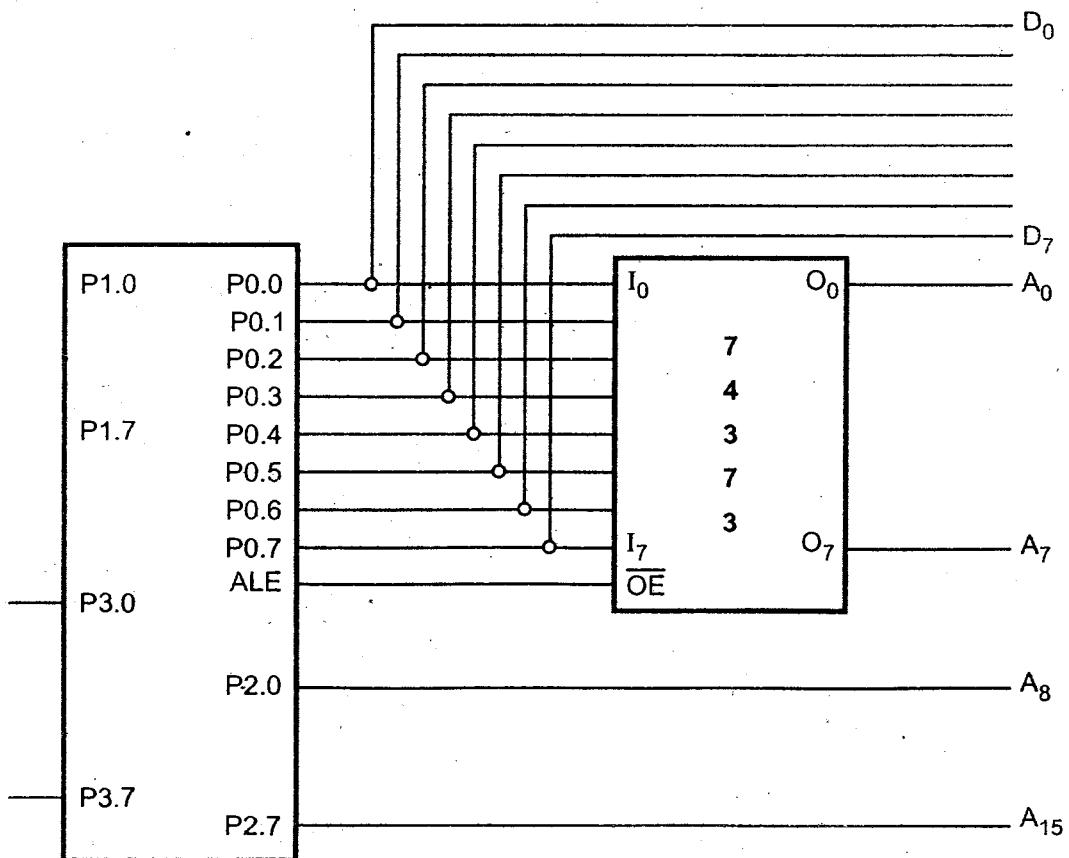
Sol:-

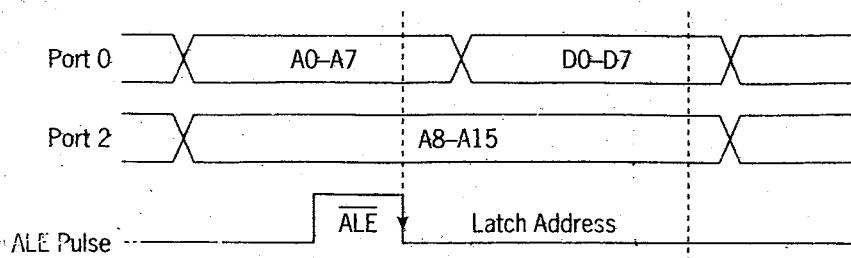
$$T_{inst} = \frac{C \times 1\mu s}{f} = \frac{4 \times 1\mu s}{11.0592 \times 10^6}$$

$T_{inst} = 4.34 \mu s$

* Write the hardware required to demultiplex the address bus of 8051.

July-06, 5M





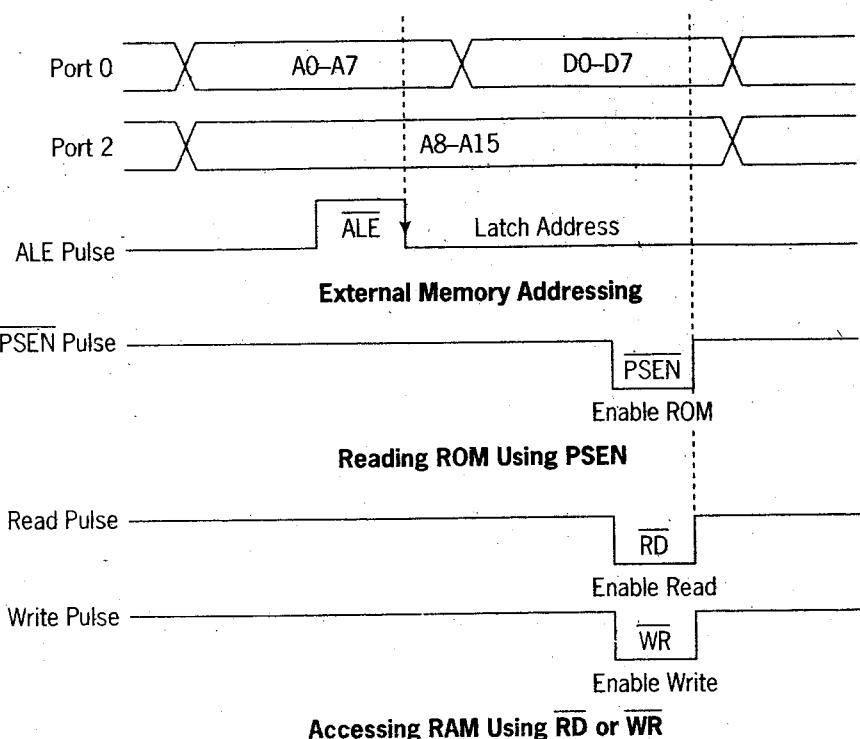
- * In 8051 microcontroller, the port has multiplexed address/data lines i.e. A₀-A₇.
- * The ALE Signal is used to demultiplex (Separate) lower order address bus (A₀-A₇) & data bus Signal (D₀-D₇).
- * When ALE=1, port 0 contains address bus &
When ALE=0, port 0 Contains data bus.
So we use 74LS373 to demultiplex address & data bus.
- * The 74LS373 will be enabled when ALE is high, So O/p of 74LS373 has lower order address A₀-A₇ as shown in fig 0.



- * Draw a Schematic to interface external ROM & RAM to 8051.
How to access them?

Jan-08, 8M

July - 07, 8M



Accessing RAM :-

- * The 8051 can address upto 64 K-bytes of external data memory.
- * The "MOVX" instruction is used to access the external data memory.

- { * The Internal data memory is divided into

Sl No	Internal RAM	Address range	Registers
1>	Lower 128 byte	00h - 7Fh	Working register, bit addressable register & general purpose register
2>	Upper 128 byte	80h - FFh	SFR registers.

- * The SFR Registers are accessed by Indirect addressing only.
- * The Lower 128 bytes are accessed either by direct addressing or



by indirect addressing.

}

RD, WR :-

RD & WR pins are used when a RAM has to be - accessed. When RD=0, a data byte can be read from a RAM location.

When WR=0, a data byte can be written into a RAM - location

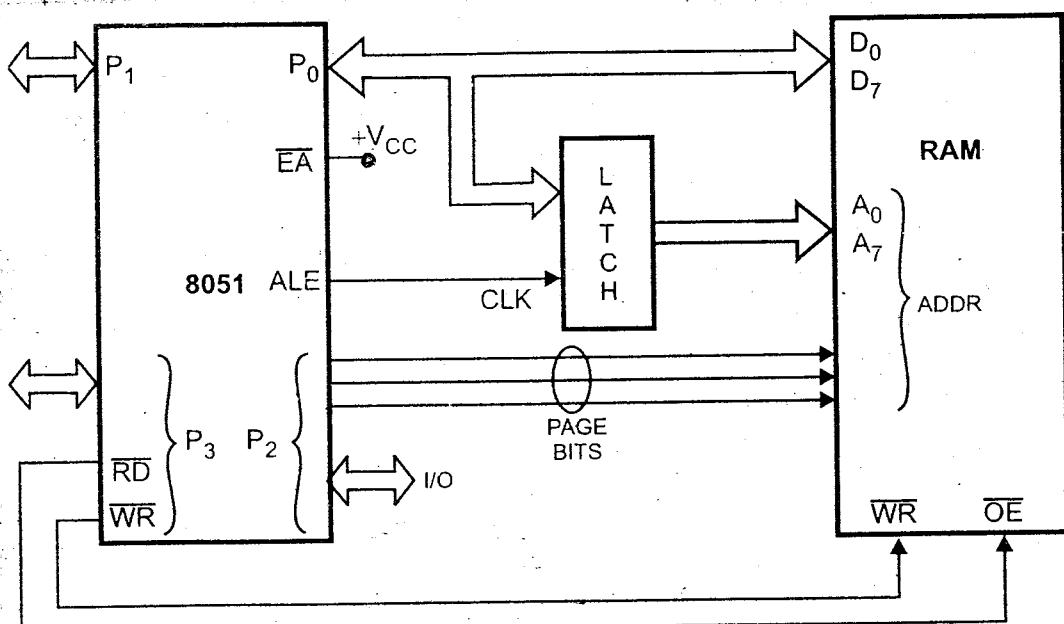


Fig. Accessing external data memory

Accessing ROM :-

- * In 8051, when the EA pin is connected to VCC, the 8051 - fetches addresses 0000h through OFFFh & are directed to internal ROM.
- * When EA=0 (GND), the 8051 fetches address 0000h through FFFFh & are directed to external ROM/EPRAM.



* P₀ is a multiplexed address/data bus.

When ALE=1, P₀ contains address bus (A₀-A₇)

When ALE=0, P₀ contains data bus (D₀-D₇)

* The 74LS373 is used to demultiplex address & data bus.

* The 74LS373 will be enabled when ALE is high, So o/p of 74LS373 has lower order address A₀-A₇ as shown in figure.

* program Store Enable (PSEN) pin is an active low pin, which is used to activate o/p enable signal of the external Rom/EPROM as shown in figure ③.

* When 8051 has to access program code from an external Rom, PSEN is connected to the enable pin (OE) of the Rom chip

* To access the program code, EA must be grounded then - PSEN will go low, to enable the external ROM to place a byte of program code on the data bus.

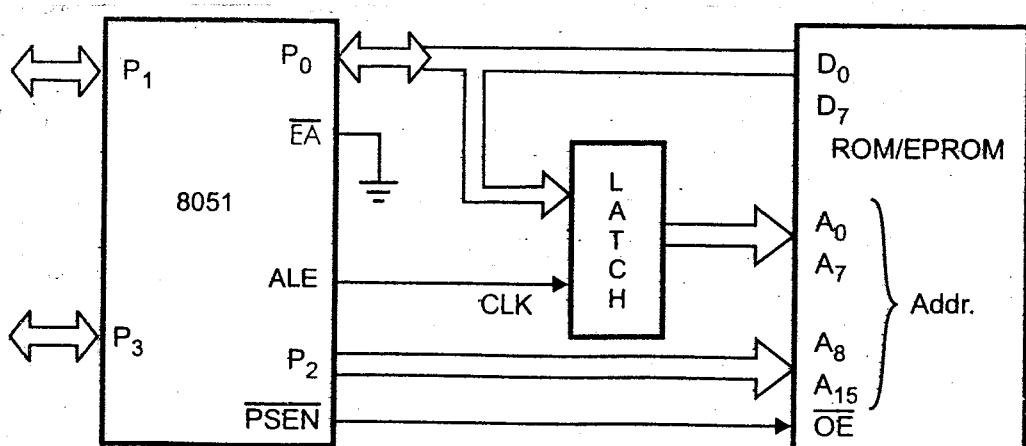


Fig. Accessing external program memory



- * Indicate the organization of internal RAM of 8051. How much maximum RAM & ROM can be used in a 8051 System & What are their - Addresser?

Jan-09, 6M

Byte Address	Byte Address
1F R7	7F
1E R6	
1D R5	
1C R4	
1B R3	
1A R2	
19 R1	
18 R0	
17 R7	
16 R6	
15 R5	
14 R4	
13 R3	
12 R2	
11 R1	
10 R0	
0F R7	2F 7F 78
0E R6	2E 77 70
0D R5	2D 6F 68
0C R4	2C 67 60
0B R3	2B 5F 58
0A R2	2A 57 50
09 R1	29 4F 48
08 R0	28 47 40
07 R7	27 3F 38
06 R6	26 37 30
05 R5	25 2F 28
04 R4	24 27 20
03 R3	23 1F 18
02 R2	22 17 10
01 R1	21 0F 08
00 R0	20 07 00
	30
	7 ← → 0
Working Registers	
Bit Addressable	
General Purpose	

The 8051 has 128-byte of Internal RAM & is divided into

S.No	Registers	Address Range
1	Working Registers	00h - 1Fh
2	Bit addressable Registers	20h - 2Fh
3	General purpose Registers	30h - 7Fh

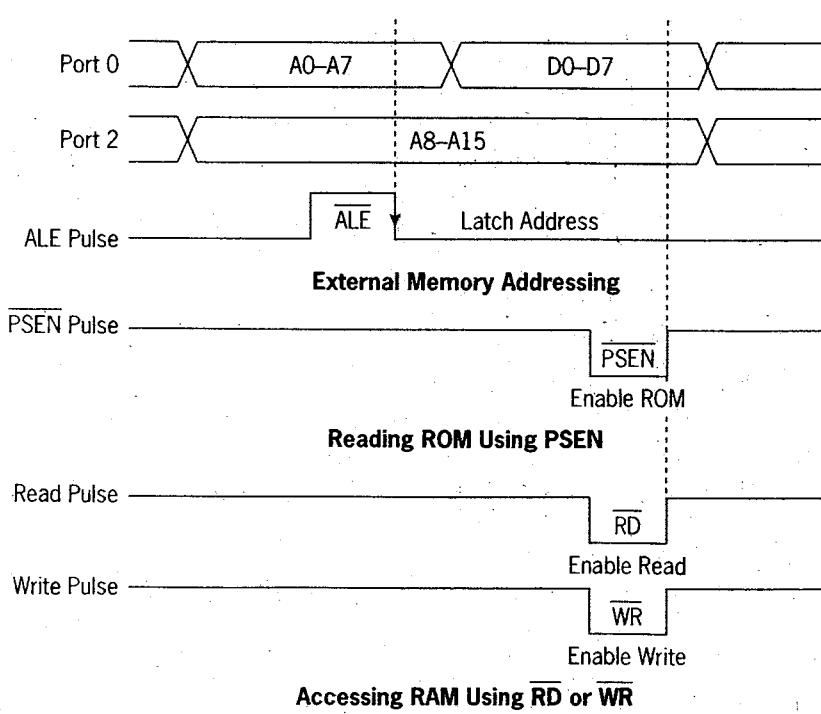


- * The SFR registers have address ranges from 80h to FFh.
- * The 8051 has 4Kbytes of Internal program memory & 256 bytes of Internal data memory.
- * The 8051 can access upto 64Kbytes of external data memory (RAM) & program memory (ROM).

SL No	Memory	Bytes	Address Range
1	Internal RAM	128 bytes } 256 128 bytes } bytes	00h-1Fh 80h-FFh → SFRs
2	Internal ROM	4Kbytes	0000h - 0FFFh
3	External RAM	64Kbytes	0000h - FFFFh
4	External ROM	64Kbytes	0000h - FFFFh

- * Explain the timing diagram of external memory cycle of 8051 microcontroller

July - 06, '7M



- * The 8051 address bus is 16-bit ($A_0 - A_{15}$)
- * p8051 has multiplexed address/data bus ($AD_0 - AD_7$), it provides lower byte address ($A_0 - A_7$) of 16-bit address.
- * The ALE Signal is used to demultiplex (Separate) lower order address bus ($A_0 - A_7$) & data bus Signal ($D_0 - D_7$).
- * When ALE=1, p8051 Contains address bus &
When ALE=0, p8051 Contains data bus.
So we use 74LS373 to demultiplex address & data bus.
- * When external ROM is to access (read),
PSEN=0, ROM is enabled & a byte of data is placed on the data bus.
- * RD & WR pins are used when a RAM has to be accessed.
When RD=0, a data byte can be read from a RAM location.
When WR=0, a data byte can be written into a RAM location.



Addressing modes :-

- * The CPU can access data in various ways. The data could be in a register, or in memory or be provided as an immediate value.
- * The various ways of accessing data are called addressing modes.

There are 5 addressing modes in 8051

- 1) Immediate addressing mode.
- 2) Register addressing mode.
- 3) Direct addressing mode.
- 4) Register indirect addressing mode.
- 5) Indexed addressing mode.

1) Immediate addressing mode :-

- * In this addressing mode, the source operand is a constant. The immediate data must be preceded by the pound sign "#".
- * This addressing mode can be used to load information into any of the registers, including the DPTR register & 8051 ports.

- ex:-
- 1) MOV A, #FFh.
 - 2) MOV R4, #0Ah.
 - 3) MOV B, #10h.
 - 4) MOV DPTR, #1234h.
 - 5) MOV P1, #55h.



①

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

2) Register addressing mode :-

- * Register addressing mode involves the use of registers to hold the data to be manipulated.
- * The registers A, DPTR, & R₀ to R₇ may be used as source as well as destination.

Ex:- 1) MOV A, R₀

2) MOV R₂, A

3) ADD A, R₅

4) ADD A, R₇

5) MOV R₆, A

6) MOV DPTR, #0123h.

7) MOV R₇, DPL

8) MOV R₆, DPH

NOTE:-

- * We can move data between the accumulator & R_n register (n=0 to 7) but movement of data b/w R_n register is not allowed.

Ex:- MOV R₄, R₇ is Invalid.

3) Direct addressing mode :-

{

- 1) RAM locations 00-1Fh are assigned to the register banks & stack.
- 2) RAM locations 20-2Fh are set aside as bit addressable space to save single-bit data.



②

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

3) RAM locations 30-7Fh are available as a place to save byte-sized data.

- g
- * The entire 128 bytes of RAM can be accessed using direct addressing mode. The RAM locations 30h to 7Fh are most often used.
 - * In direct addressing mode, the data is in a RAM memory location whose address is known, & this address is given as a part of the instructions.

NOTE:- The '#' sign distinguishes b/w immediate & direct addressing.

- Ex's :-
- 1) MOV R0, 40h.
 - 2) MOV 56h, A
 - 3) MOV R7, 01h.
 - 4) PUSH 0EOh.
 - 5) POP 03h etc.

{ RAM locations 0 to 7 are allocated to bank 0 register R0 - R7. These registers can be accessed in two ways.

- 1) MOV A, R4 is same as MOV A, 4
- 2) mov A, R7 is same as mov A, 7
- 3) mov A, R0 is same as mov A, 0.
- 4) mov R2, R3 is same as mov 2,3 but Invalid inst

g.



3

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

4) Register Indirect addressing mode :-

- * In register indirect addressing mode a register is used to hold the address of the data.
- * the register itself is not the address, but rather the number in the register.
- * The instruction for indirect addressing uses MOV opcodes along with registers R0 or R1. Register R0 or R1 will hold the RAM address ranging from 00h to FFh.
- * The mnemonic symbol used for indirect addressing is the "at" sign i.e., "@".

Ex:- 1) MOV A,@R0.

2) MOV @R1,B.

3) MOV @R1,A

4) MOV 20h,@R1

5) MOV @R0,03h.

Limitations of register indirect addressing mode :-

- * R0 & R1 are the only registers that can be used for pointers in register indirect addressing mode. Since R0 & R1 are 8-bit wide, their use is limited to accessing any information in the internal RAM.
- * By using DPTR register we can access external memory.



5) Indexed addressing mode :-

- * Indexed addressing mode is widely used in accessing data elements of look-up table entries located in the program ROM space.
- * The instruction's used for this purpose is
 - i) MOVC A, @A + D PTR
 - ii) MOVC A, @A + PC
- {
- i) MOVC A, @A + D PTR : Add the contents of the accumulator with the contents of the D PTR register to form a program code memory location address. Move the contents of this external memory address to the accumulator.
- ii) MOVC A, @A + PC :-
Add the contents of the accumulator with the contents of the PC register to form a program code memory location address. Move the contents of this external memory address to the accumulator.

4

OR

- * only program memory can be accessed in the Index addressing. Either the D PTR or PC can be used as an index register.



5

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

INSTRUCTION SET

Based on the operations performed, the instruction set of 8051 are classified as

- 1) Data transfer instructions.
- 2) Arithmetic instructions.
- 3) Logical instructions.
- 4) Boolean instructions.
- 5) Program branching or Machine control instructions.

* Each instruction has two parts :

operation code & operands.

I) Data Transfer group :-

The instruction in this group are

MOV, PUSH, POP, XCH.

MOV :-

MOV instruction copies data from one location to another location.

Syntax :- MOV operand 1, operand 2

format :- MOV Destination, Source ;

copy from source to destination.



①

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

1) MOV A, Rn.

Bytes : 1

Cycles : 1

Status flags affected : None.

Operation : MOV

$$(A) \leftarrow (R_n)$$

Description :- This instruction moves the contents of Rn register to accumulator. The Rn register is not affected.

Note :-

Rn may be R0 to R7 register of the currently selected bank.

Ex :- MOV A, R3

Before Execution $\rightarrow R_3 = 58h$, A = Any value say 10h.

After Execution $\rightarrow A = 58h$. & $R_3 = 58h$.

2) MOV A, direct (Direct address)

Bytes : 2.

Cycles : 1

Operation : $(A) \leftarrow (\text{direct})$

Description : This instruction moves the contents of the address into A register.

Flags affected : None



②

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

ex:- MOV A, 40h.

Before Execution.

$$A \leftarrow 'xx'$$

$$40h \leftarrow FF$$

After Execution.

$$A \leftarrow FF$$

$$40h \leftarrow FF$$

NOTE:- Here 40h is a direct address. The direct address content 'FF' is moved into Accumulator.

3) MOV A, @Ri

Bytes : 1

Cycles : 1.

operation : $(A) \leftarrow ((R_i))$

Description :

Flags affected: None.

ex:-

MOV A, @R0.

Say $R_0 = 40h \leftarrow$ address.

$40h = FF \leftarrow$ data.

* The address 40h is in register R0.

* The data FF is in address 40h.

Before execution.

$$R_0 \leftarrow 40h.$$

$$40h \leftarrow FF.$$

$$A \leftarrow 'xx'$$

After execution.

$$R_0 \leftarrow 40h.$$

$$40h \leftarrow FF$$

$$A \leftarrow FF.$$



③

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

4) $\text{MOV A}, \# \text{data}$.

Bytes : 2.

Cycles : 1

Operation: $(A) \leftarrow \# \text{data}$.

Flags affected: None.

Description: 8-bit data is moved into accumulator.

Ex:- $\text{MOV A}, \# 28$

Before execution

$A \leftarrow 'xx'$

After execution.

$A \leftarrow 28$.

5) $\text{MOV Rn}, A$

Bytes : 1.

Cycles : 1.

Operation : $(R_n) \leftarrow (A)$.

R_n may be any Register
i.e. R₀-R₇ of the currently
selected bank.

Description : The contents of accumulator is moved
to register R_n.

Flags affected : None.

Ex:- $\text{MOV R}_7, A$

Before Execution

$R_7 = 'xx'$

$A = FF$

After execution.

$R_7 = FF$

$A = FF$.



④

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

6} MOV Rn, Direct

Rn may be any register

Bytes : 2.

(R₀ to R₇).

Cycles : 2.

Operation : (R_n) \leftarrow (Direct)

Flags affected : None.

Ex :-

MOV R₁, 40h.

Before execution.

$$40h = FF$$

$$R_1 = 'xx'$$

After execution.

$$40h = FF$$

$$R_1 = FF$$

7} MOV Rn, # data.

Rn \rightarrow may be any register
(R₀ to R₇)

Bytes : 2.

Cycles : 1.

Operation : (R_n) \leftarrow # data.

Flags affected : None

Ex :-

MOV R₅, #00

Before Execution

$$R_5 = 'xx'$$

After Execution.

$$R_5 = 00.$$

8) MOV direct, A

Bytes : 2.

Cycles : 1.

Operation : (direct) \leftarrow (A)

Flags affected : None.



⑤

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

ex:- MOV 50h, R₃.

Before execution

$$50h = FF$$

$$R_3 = 00$$

After execution.

$$50h = 00$$

$$R_3 = 00$$

10) MOV direct, direct

Bytes : 3

Cycles : 2

Operation: (direct) \leftarrow (direct)

Flags affected : None.

ex:- MOV 30h, 50h.

Before execution.

$$30h = 11$$

$$50h = 00$$

After execution.

$$30h = 00$$

$$50h = 00.$$

11) MOV direct, @ R_i

Bytes : 2

Cycles : 2

Operation : (direct) \leftarrow (R_i)

Flags affected : None.

ex:- MOV 70h, @ R₀

Before execution

$$70h = 00$$

$$R_0 = 40h.$$

$$40h = FF.$$

After execution.

$$70h = FF$$

$$R_0 = 40h.$$

$$40h = FF$$



⑥

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

12) $\text{MOV direct, \#data.}$

Bytes : 3.

Cycles : 2.

Operation: $(\text{direct}) \leftarrow \#\text{data.}$

Flags affected: None.

Ex:- MOV 70h, \#FF

Before execution.

$70h = 'xx'$

After execution.

$70h = FF$

13) MOV @Ri, direct

Bytes : 2.

Cycles : 2.

Operation: $((Ri)) \leftarrow (\text{direct})$

Flags affected: None.

Ex:- MOV @R1, 70h

Before execution

$R_1 = 40h$

$40h = 'xx'$

$70h = FF$

After execution

$R_1 = 40h$

$40h = FF$

$70h = FF$

14) MOV @Ri, \#data.

Bytes : 2

Cycles : 1

Operation: $((Ri)) \leftarrow \#\text{data.}$

Flags affected: None.



ex:- MOV @ R0, #00

Before execution

$$R0 = 40h.$$

$$40h = 'xx'$$

After execution.

$$R0 = 40h.$$

$$40h = 00.$$

* MOV dest-bit, source-bit

function : Move bit data from source bit to destination bit.

NOTE:-

one of the operands must be the carry flag ;
the other may be any directly addressable bit.

Flags affected : carry flag.

i) MOV C, bit

Bytes : 2

Cycles : 1.

operation : $(C) \leftarrow (bit)$

Flags affected : carry flag (CY)

ex:- i) MOV C, P1.4

Before execution

$$C = 'x'$$

$$P1.4 = 1$$

After execution.

$$C = 1$$

$$P1.4 = 1.$$

ii) MOV C, P0.7

Before execution.

$$C = 'x'$$

$$P0.7 = 0.$$

After execution.

$$C = 0$$

$$P0.7 = 0.$$



⑧

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

16) MOV bit, C

Bytes : 2.

Cycles : 2.

Operation : $(\text{bit}) \leftarrow (\text{C})$

Flags affected : None.

Ex:- i) MOV P1.2, C

Before execution

$P1.2 = 'X'$

$C = 1.$

After execution.

$P1.2 = 1$

$C = 1.$

ii) MOV P3.7, C

Before execution

$P3.7 = 'X'$

$C = 0$

After execution

$P3.7 = 0$

$C = 0.$



⑨

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Arithmetic Group Op. Instructions :-

* ADD A, source.

Syntax : ADD A, operand.

Function : Add

Description : Adds the contents of source with
Accumulator contents & result is stored
in accumulator.

Flags affected : C, AC & OV.

NOTE:- OV is set if there is a carry-out of
bit 6 but no carry out from bit 7 or a
carry out of bit 7 but no carry out from
bit 6 ; otherwise OV = 0.

i) ADD A, Rn.

where $R_n = R_0$ to R_7 .

Bytes : 1.

Cycles : 1

Operation : $(A) \leftarrow (A) + (\text{direct})$

Flags affected : CY, OV & AC.

Ex:-

i) ADD A, R4.

Before execution.

$$A = '11'$$

$$R_4 = 11$$

After execution.

$$A = 22$$

$$R_4 = 11$$



⑩

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

i) ADD A, R₇

Before execution.

$$A = 00$$

$$R_7 = FF$$

After execution

$$A = FF$$

$$R_7 = FF$$

ii) ADD A, direct

Bytes : 2

Cycles : 1

Operation : $(A) \leftarrow (A) + (\text{direct})$

Flags affected: CY, AC, OV.

Ex:-

ADD A, 40h.

Before execution.

$$A = 11$$

$$40h = 02$$

After execution.

$$A = 33$$

$$40h = 22$$

3) ADD A, @R_i

Bytes : 1

Cycles : 1

Operation: $(A) \leftarrow (A) + ((R_i))$

Flags affected: CY, AC, OV.

Where $R_i \rightarrow$ may be R₀ or R₁.

Program:-

MOV @R_i, #70h

MOV A, #30h

ADD A, @R_i.

Ex:- ADD A, @R₁

Before execution.

$$A = 30$$

$$R_1 = 70h$$

$$70h = 20$$

After execution.

$$A = 50$$

$$R_1 = 70h$$

$$70h = 20$$



11

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

4) ADD A, # data.

Bytes : 2.

Cycles : 1.

Operation : $(A) \leftarrow (A) + \# \text{data}$.

Flags affected : CY, OV, AC.

Ex:- ADD A, #01h.

Before execution.

A = 01.

After execution.

A = 02.

5) ADDC A, Rn.

Bytes : 1.

Cycles : 1

Operation: $(A) \leftarrow (A) + (C) + (R_n)$

Description: Simultaneously adds the contents of Rn register, the carry flag & the accumulator contents, result is stored in accumulator.

Flags affected: OV, CY, AC.

Ex:- 2) ADDC A, R7

Before execution

A = 01

C = 1

R7 = 01.

After execution

A = 03

C = 1

R7 = 01.



12

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

ii) ADDC A, R0

Before execution

$$A = 01$$

$$C = 0$$

$$R0 = 01$$

After execution.

$$A = 02$$

$$C = 0$$

$$R0 = 01.$$

6) ADDC A, direct

Bytes : 2

Cycles : 1.

Operation : $(A) \leftarrow (A) + (C) + (\text{direct})$

Flags affected : CY, AC, OV.

Ex: ADDC A, 80h.

Before execution

$$A = 02$$

$$C = 1$$

$$80h = 02$$

After execution

$$A = 05$$

$$C = 1$$

$$80h = 02.$$

7) ADDC A, @Ri

where $Ri = R_0 \text{ or } R_1$

Bytes : 1

Cycles : 1.

Operation : $(A) \leftarrow (A) + (C) + ((Ri))$

Flags affected : CY, AC, OV.



13

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

ex:- ADDC A, @R0.

Before execution

$$A = 01$$

$$C = 1$$

$$R_0 = 70h$$

$$70h = 01$$

After execution.

$$A = 03$$

$$C = 1$$

$$R_0 = 70h$$

$$70h = 01$$

8) ADDC A, #data.

Bytes : 2

Cycles : 1

operation: $(A) \leftarrow (A) + (C) + \# \text{data}.$

Flags affected: CY, AC, OV

ex:- ADDC A, #01h

Before execution.

$$A = 01$$

$$C = 1.$$

After execution.

$$A = 03$$

$$C = 1.$$

9) MUL AB.

Function : Multiply. A \times B.

Description: MUL AB multiplies the unsigned eight-bit integer i.e. the contents of accumulator is multiplied with the contents of register B.

Bytes : 1

Cycles : 4.



14

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

operation : $(A)_{7-0}$ }
 $(B)_{15-8}$ } $\leftarrow (A) \times (B)$.

Flags affected : c, ov

ex:- mov A, #5

$$5 \times 5 = 35 = 23h.$$

mov B, #7

$$A = 35 = 23h.$$

MUL AB

$$B = 00$$

Before execution.

$$50 \times A0 = 3200h.$$

$$A = 50h$$

$$B = A0h.$$

After execution.

$$A = 00h \leftarrow (0-7) \text{ lower byte}$$

$$B = 32h \leftarrow (15-8) \text{ higher byte}$$

ex:- A = 100, B = 200.

$$\rightarrow 100 \times 200 = 20,000.
= 4E20h.$$

$$A \times B = 4E20h.$$

$$A = 20h, B = 4E$$

10) DIV AB

Function : Divide.

Bytes : 1.

Cycles : 4.

operation : $(A)_{15-8}$ } $\leftarrow (A) / (B)$
 $(B)_{7-0}$ }

Flags affected : cy, ov.



15

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Description : DIV AB divides the unsigned 8-bit integer content of accumulator by the unsigned 8-bit integer content of B-register.

The accumulator receives the integer part of the quotient ; register B receives the integer remainder.

The carry & overflow flags will be cleared.

Ex:- Before execution	After execution.	
$A = FBh$ (251 dec)	$A = 0dh$ (13 dec)	$18 \overline{) 251}$
$B = 12h$ (18 dec).	$B = 11h$ (17 dec)	$\underline{234}$ $17 \leftarrow (B)$

- * the quotient 13 (decimal) 0dh is stored in accumulator & the remainder (17 decimal) 11h is stored in B-register.

Ex:- $\rightarrow A = 35, B = 10$

DIV AB.

A = 3, B = 5

$$\begin{array}{r} 3 \leftarrow \text{Quotient} \\ 10) 35 \\ \underline{30} \\ 05 \\ \uparrow \\ \text{Remainder.} \end{array}$$

Q) $A = 97h = 151$

$B = 12h = 18$

DIV AB

A = 8, B = 7

$$\begin{array}{r} 8 \\ 18) 151 \\ \underline{144} \\ 7 \end{array}$$



16

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

11) INC A.

Bytes : 1.

Cycles : 1.

Operation : $(A) \leftarrow (A) + 1.$

Flags affected : None.

Before execution.

$A = 01$

After execution.

$A = 02.$

12) INC Rn.

Bytes : 1.

Cycles : 1.

Operation : $(R_n) \leftarrow (R_n) + 1.$

Flags affected : None.

Ex: INC R0.

Before execution.

$R_0 = 30h.$

After execution.

$R_0 = 31h.$

13) INC direct

Bytes : 2

Cycles : 1

Operation : $(\text{direct}) \leftarrow (\text{direct}) + 1$

Flags affected : None.



Ex:- INC 40h.

Before execution. After execution.
 $40h = 01$ $40h = 02$.

14) INC @ Ri where, $Ri \rightarrow R_0$ or R_1

Bytes : 1.

Cycles : 1.

Operation : $((R_i)) \leftarrow ((R_i)) + 1$.

Flags : None.

Function : Increment.

Ex: INC @ R0.

Before execution. After execution.
 $R_0 = 80h$. $R_0 = 80h$.
 $80h = 01$. $80h = 02$.

15) INC DPTR.

Function : Increment data pointer.

Bytes : 1.

Cycles : 2.

Operation : $(DPTR) \leftarrow (DPTR) + 1$.

Description : This instruction increments the 16-bit register content (DPTR) by 1. This is the only 16-bit register that can be incremented.



⑧

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Flags : None.

CX : INC DPTR.

i) Before Execution.

DPTR = 16 FFh.

(DPH = 16, DPL = FF)

i) After execution.

DPTR = 1700h.

(DPH = 17, DPL = 00)

ii) Before Execution.

DPTR = 00FFh.

DPH = 00h.

DPL = FFh.

B After execution.

DPTR = 0100h.

DPH = 01h

DPL = 00h.

16) DEC A.

Function : Decrement.

Bytes : 1.

Cycles : 1.

Operation: $(A) \leftarrow (A) - 1$

Flags : None.

Before execution.

A = 05h.

After execution.

A = 04h.

17) DEC Rn.

Bytes : 1.

Cycles : 1.

Operation: $(Rn) \leftarrow (Rn) - 1$.

Flags : None.



ex:- DEC R0.

Before execution.

$$R_0 = 80h.$$

$$80h = 05h.$$

After execution.

$$R_0 = 80h.$$

$$80h = 04h.$$

18) DEC direct

Byte : 2.

Cycles : 1.

Operation : $(\text{direct}) \leftarrow (\text{direct}) - 1$

Flags : None.

ex:- DEC 40h.

Before execution

$$40h = 05h.$$

After execution.

$$40h = 04h.$$

19) DEC @ Ri

Byte : 1.

Cycles : 1.

Operation : $((R_i)) \leftarrow ((R_i)) - 1$

Flags : None.

ex:- DEC @ R0.

Before execution.

$$R_0 = 80h$$

$$80h = 04h.$$

After execution

$$R_0 = 80h$$

$$80h = 03h.$$



20

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

20) DA A

Function : Decimal - adjust accumulator after addition.

Flags : CY.

Description : This instruction is used after addition of BCD numbers to convert the result back to BCD. The data is adjusted in the following two possible cases.

- ↳ It adds 6 to the lower 4-bits of A if it is greater than 9 or if AC=1.
- ↳ It also adds 6 to the upper 4-bits of A if it is greater than 9 or if CY=1.

Bytes : 1.

Cycles : 1.

Operation : If $[(A_{3-0}) > 9] \vee [AC = 1]$ Then
 $(A_{3-0}) \leftarrow (A_{3-0}) + 6.$

If $[(A_{7-4}) > 9] \vee [CY = 1]$ Then.

$$(A_{7-4}) \leftarrow (A_{7-4}) + 6.$$

Ex:- MOV A, #47h. A = 47h.

ADD A, #38h ; A = 47h + 38h = 7Fh, invalid BCD.

DA A ; A = 85h, valid BCD.



21

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

i.e. $47h$

$+38h$

$\overline{7Fh}$. Invalid BCD

$6h$. After DA A

$\boxed{85h}$ Valid BCD

$\boxed{AC=1}$

* since the lower nibble is greater than 9, DA added 6 to A. If the lower nibble is less than 9 but $AC=1$, it also adds 6 to the lower nibble.

ii) $MOV A, \#29h$.

$29h$

$ADD A, \#18h$.

$+18h$

$\overline{41h} \leftarrow$ (incorrect result in BCD)

DA A

$+6$

$\overline{47h} \leftarrow$ (correct result in BCD)

$\boxed{AC=1}$

iii) $MOV A, \#52h$.

$52h$

$ADD A, \#91h$.

$+91h$

DA A

$E3h$. Invalid BCD.

$+6$ upper nibble is >9 , so added 6.

$\boxed{CY=1}$

$\boxed{143h}$ Valid BCD

iv) $MOV A, \#54h$.

$54h$

$ADD A, \#87h$.

$87h$

DA A

$\overline{① \leftarrow d.b.}$ Invalid BCD

66

$\overline{141}$. BCD.

$\boxed{AC=1, CY=1}$



22

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

21) SUBB A, source byte.

Function : Subtract with borrow.

Flags : OV, AC, CY.

Operation : $(A) = (A) - (\text{byte}) - (C)$ or $(A) = (A - \text{byte} - C)$

Description : SUBB subtracts the source byte & the carry flag from the accumulator & puts the result in the accumulator.

SUBB instruction sets the carry flag according to the following.

CY

- i) destination byte > source byte. 0 the result is +ve.
- ii) destination byte = source byte. 0 the result is 0.
- iii) destination byte < source byte. 1 the result is -ve in 2's compliment.

Ex:- XCH A, Rn.

Bytes : 1

Cycles : 1.

Operation : $(A) \leftrightarrow (R_n)$.

Flags : None.

Ex:- XCH A, R2.

Before execution.

$$A = FFh.$$

$$R2 = 11h.$$

After execution.

$$A = 11h.$$

$$R2 = FFh.$$



23

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Ex:- MOV A, #FFh.
MOV R2, #11h.
XCH A, R2.

Q3) XCH A, direct

Bytes : 2
cycles : 1
operation : $(A) \leftarrow (\text{direct})$
Flags : None

Ex:- XCH A, 40h.

Before execution. After execution.
 $A = FFh$. $A = 11h$.
 $40h = 11h$. $40h = FFh$.

Q4) XCH A, @Ri

Bytes : 1
cycles : 1
operation : $(A) \longleftrightarrow ((R_i))$
Flags : None

Ex:- XCH A, @R1

MOV 40h, #11h	Before execution	After execution.
MOV R1, #40h.	$40h = 11h$.	$40h = FFh$.
MOV A, #FFh.	$A = FFh$.	$A = 11h$.
XCH A, @R1.		



24

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

25} XCHD A, @ R_i

Function : Exchange Digit

Description : The XCHD instruction exchanges only the lower nibble of accumulator (bit 3-0), with the lower nibble of the RAM location pointed to by R_i.

* The higher-order nibbles (bits 7-4) of each register are not affected.

Flags : None.

Bytes : 1.

Cycles : 1.

Operation : A₍₃₋₀₎ \leftrightarrow ((R_i 3-0))

Ex:- XCHD A, @ R_i.

MOV A, # FFh.

Before execution.

After execution.

A = FFh.

MOV R1, # 50h.

A = FFh.

MOV 50h, # 00h.

R1 = 50h.

A = F0h.

XCHD A, @ R1.

50h = 00h.

R1 = 50h.

50h = 0Fh.

26} MOV DPTR, # 16-bit value.

Function : Load data pointer with a 16-bit constant.

Description : The data pointer is loaded with the 16-bit constant indicated.

The DPH register holds high-order byte, while DPL holds the low-order byte.



25

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Flags : None.

Bytes : 3.

Cycles : 2.

Operation: $(DPTR) \leftarrow \# \text{data}_{(0-15)}$

$DPH \leftarrow \# \text{data}_{15-8}$

$DPL \leftarrow \# \text{data}_{7-0}$

Ex:- $\text{MOV DPTR}, \#1234$.

After execution

$DPH = 12 \text{ h.}$

$DPL = 34 \text{ h.}$

27) $\text{MOVX dest-byte; source-byte.}$

Function : Move External.

Description: * This instruction transfers data b/w external memory & register A, hence the "x" appended to MOV.

* The address of external memory location being accessed can be 16-bit or 8-bit.

MOVX A, @Ri

Bytes : 1.

Cycles : 2.

Operation: $(A) \leftarrow ((Ri))$

Flags : None.



Ex:- MOVX A, @ R0.

MOV R0, #50h.

Before execution After execution.

MOV 50h, #FFh

R0 = 50h.

A = FFh.

MOVX A, @ R0.

50h = FFh.

R0 = 50h.

A = 'xx'

50h = FFh.

28) MOV A, @ DPTR.

Bytes : 1.

Cycles : 2

Operation: (A) \leftarrow ((DPTR))

Flags : None.

Ex:- MOVXA, @ DPTR.

MOV DPTR, #1234

Before execution

After execution.

MOV 1234h, #FFh.

1234h = FFh.

A = FFh.

MOVX A, @ DPTR.

DPTR = 1234.

DPTR = 1234h.

A = 'xx'

1234h = ffh.

29) MOVX @ Ri, A

Flags : None

Bytes : 1

Cycles : 2

Operation: ((Ri)) \leftarrow (A)



27

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Ex:- MOVX @R1, A

MOV R1, #80h.

Before Execution.

After Execution.

MOV 80h, #AAh.

R1 = 80h.

A = AAh.

MOVX @R1, A.

80h = AAh.

R1 = 80h.

A = 'xx'

80h = AAh.

30) MOVX @DPTR, A.

Bytes : 1.

Cycles : 2.

Operation: ((DPTR)) \leftarrow (A)

Flags : None.

Ex:- MOVX @ DPTR, A.

MOV DPTR, #1234

Before execution

After execution.

A = FFh.

MOV 1234, #FFh.

DPTR \leftarrow 1234h

MOVX @DPTR, A.

1234h \leftarrow FFh.

DPTR = 1234h.

A \leftarrow 'xx'

1234 = FFh.



Logical Instructions :-

* ANL dest-byte, src-byte.

function : Logical - AND for byte variables.

Description : ANL performs the bit wise logical - AND operation b/w the variables indicated & stores the result in the destination variable.

Flags : None.

A	B	A AND B.
0	0	0
0	1	0
1	0	0
1	1	1

i) ANL A, Rn

Bytes : 1.

Cycles : 1.

operation : $(A) \leftarrow (A) \wedge (R_n)$

function : Logical AND for byte variables.

Flags : None.

ex:- ANL A, R5.

Before execution

$A = 39$

$R5 = 09$.

After execution

$A = 09$

$R5 = 09$.

0011 1001	$\leftarrow 39$
0000 1001	$\leftarrow 09$
0000 1001	09



2) ANL A, direct

Bytes : 1

Cycles : 1.

Operation: $(A) \leftarrow (A) \wedge (\text{direct})$

Flags : None.

Ex:- ANL A, 40h.

Before execution

$A = 39$

$40h = 09$

After Execution.

$A = 09$

$40h = 09$

$\begin{array}{r} 39 \\ \wedge 09 \\ \hline 09 \end{array}$

$0011\ 1001$

$0000\ 1001$

$0000\ 1001$

3) ANL A, @ Ri

$Ri \rightarrow R0 \text{ or } R1$

Bytes : 1.

Cycles : 1.

Operation: $(A) \leftarrow (A) \wedge ((Ri))$

Flags : None.

Ex:- ANL A, @ R0.

Before execution.

$A = 39$

$R0 = 80h.$

$80h = 09.$

After execution.

$A = 09$

$R0 = 80h.$

$80h = 09.$

$A = 39$

$\begin{array}{r} 39 \\ - 09 \\ \hline 09 \end{array}$

$0011\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\ 1001$

$0000\$

4) ANL A, #data.

Bytes : 2.

Cycles : 1

Operation : $(A) \leftarrow (A) \wedge \#data$.

Flags : None.

Ex:- ANL A, #09h.

Before execution.

A = 39

After execution.

A = 09

39	0011 1001
09	0000 1001
09	0000 1001

5) ANL direct, A

Bytes : 2

Cycles : 1.

Operation : (Direct) \leftarrow (Direct) \wedge (A)

Flags : None

Ex:- ANL 40h, A

40h = 39

A = 09

After execution.

40h =

A =

Ex:-

ANL P1, #1111110 B ; Mask P1.0 (i.e D0 of port 1)

ANL P1, #0111111 B ; Mask P1.7 (i.e D7 of port 1)

ANL P1, #11110111 B ; Mask P1.3 (i.e D3 of port 1)

ANL P1, #11111100 B ; Mask P1.0 & P1.1.



* ANL C, source-bit

Function : Logical AND for bit variables.

Flag : CY.

Description : In this instruction carry flag bit is ANDed with a source bit & the result is placed in carry flag.

i.e if source bit = 0, then CY=0 otherwise CY=1.

i) ANL C, bit

Bytes : 2

Cycles : 2

Operation : $(C) \leftarrow (C) \wedge (\text{bit})$

Flags : CY.

Ex's:-

i) ANL C, ACC.7

Before execution. After execution

C = 1

C = 0

A = 13 = 00010011.

A = 13.

A = 13.

C = 1 = 1

Result C = 00010011.

ii) ANL C, P2.2.

Before execution.

After execution.

C = 1.

C = 0

P2 = 00001011.

P2 = 00001011 B.

P2 = 0Bh.

C = 1

or

P2 = 0Bh.

Result C = 0



⇒ ANL C, 1 bit

bytes : 2

cycles : 2

operation: $(C) \leftarrow (C) \wedge T(\text{bit})$

Flag : CY.

Description: AND carry bit with inverse of bit data
↑
(complement)

ex:- ii) ANL C, 1 ACC. F

Before execution

C = 1

ACC = 0111111B.

or

AC = 7Fh.

After execution

C = 1

ACC = 7Fh.

ACC.F = 0111111

1 ACC.F = 1000000

$\begin{array}{r} \wedge C \\ C \quad \boxed{\text{Result}} \quad \boxed{1} \leftarrow C \end{array}$

ii) ANL C, 1 P2.0.

Before execution

C = 1.

P2 = 00010000 B.

or

P2 = 10h.

After execution.

C = 1.

P2 = 10h.

P2.0 = 00010000

1 P2.0 = 00010001

$\begin{array}{r} \wedge C \\ C \quad \boxed{\text{Result}} \quad \boxed{1} \leftarrow C \end{array}$



8) SWAP A

Bytes: 1

Cycles: 1

Operation: $(A_{3-0}) \leftrightarrow (A_{7-4})$

Function: Swap nibbles within the accumulator.

Flags: None

Description: The swap instruction interchanges the lower nibble ($A_0 - A_3$) with the upper nibble ($A_4 - A_7$) inside register A.

Ex:- MOV A, #59H

SWAP A

Before execution

$A = 59H$

i.e $A = 0101\ 1001$

After execution

$A = 95$

i.e $A = 1001\ 0101$



Logical OR for the byte variables :-

ORL < dest-byte >, < src-byte > or

ORL dest-byte, source-byte.

function : Logical OR for byte variables.

Description: ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte.

(The ORL instruction can be used to set certain bits of an operand to 1).

1) ORL A, Rn.

Bytes : 1

Cycles : 1

Operation: $(A) \leftarrow (A) \vee (R_n)$

Flags: None

Description: This instruction performs a logical OR on the byte operands, bit by bit, and stores the result in the destination.

Ex:- ORL A, R5

MOV A, #32h	Before execution.	A = 00110010
MOV R4, #50h	A = 32h	R5 = 01010000
ORL A, R4	R5 = 50h	A \leftarrow <u>01110010</u> 72h



2) ORL A, direct

Bytes : 2

Cycles : 1

Operation: $(A) \leftarrow (A) \vee (\text{direct})$

Flags: None

Ex:- ORL A, 30h.

Before execution.

A = 32h.

30h = 50h.

After execution

A = 72h.

30h = 50h.

i.e.

MOV A, # 32h.

MOV 30h, # 50h.

ORL A, 30h.

3) ORL A, @ R_i

R_i \rightarrow R₀ or R₁ only.

Bytes : 1

Cycles : 1

Operation: $(A) \leftarrow (A) \vee ((R_i))$

Flags: None

Ex:- ORL A, @ R₁.

MOV 30h, # 50h.

Before execution

A = 32h.

A = 72h.

After execution

A = 72h.

MOV R₁, # 30h

R₁ = 30h

R₁ = 30h.

MOV A, # 32h.

R₁ = 30h

R₁ = 30h.

ORL A, @ R₁.

30h = 50h.

30h = 50h.

4) ORL A, # data.

Bytes : 2

Cycles : 1

Operation: $(A) \leftarrow (A) \vee \# \text{data}$



Flags : None.

ex:- ORL A, #50h.

Before execution.

A = 32h.

data = 50h.

After execution.

A = 72h.

5) ORL direct, A

Bytes : 2

Cycles : 1.

Operation: (direct) \leftarrow (direct) V (A)

Flags : None

ex:- ORL 30h, A.

MOV A, #32h

Before execution

After execution.

A = 32h.

A = 72h.

MOV 30h, #50h

30h = 50h.

30h = 50h.

6) ORL direct, #data.

Bytes : 3.

Cycles : 2.

Operation: (direct) \leftarrow (direct) V #data

Flags : None

ex:- ORL 30h, #32h.

Before execution.

After execution

MOV 30h, #50h

30h = 50h

30h = 72h.

ORL 30h, #32h.

data = 32h.



ORL C, source-bit

Function : Logical OR for bit variables.

Description : The carry flag bit is ORed with a source bit and the result is placed in the carry flag.

∴ If the source bit is 1, CY is set;
Otherwise, the CY flag remains unchanged.

Flags : CY

* ORL C, bit

Bytes : 2

Cycles : 2

Operation : $(C) \leftarrow (C) \vee (\text{bit})$.

Flags : CY

Ex:- ORL C, ACC.3

i) Before execution.

A = FF.

CY = 0.

i.e. A = 11110111

After execution.

A = FF.

CY = 1

ii) Before Execution

A = F7

CY = 0

After execution.

A = F7

CY = 0

A = 11110111



38

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

8) ORL C, 1bit

1bit \rightarrow NOT of bit OR bit

bytes : 2

Cycles : 2

Operation: $(C) \leftarrow (C) \vee (\overline{\text{bit}})$

Flag: CY

Ex:- ORL C, 1ACC.3

i) Before execution

$$A = 00011000 \text{ b.}$$

i.e $A = 18h$.

$$CY = 0$$

After execution.

$$CY = 0$$

$$A = 00010000 \text{ bit } 3$$

$$\begin{array}{r} + CY = \\ \hline \text{Result} \quad CY \end{array} \quad \begin{array}{r} + 0 \\ \hline 0 \end{array}$$

ii) Before execution.

$$A = 00010000 \text{ b}$$

i.e $A = 10h$

$$CY = 0$$

After execution

$$A = 00011000 \text{ bit } 3$$

$$\begin{array}{r} + 0. \\ \hline CY = \end{array} \quad \begin{array}{r} 1. \\ \hline \end{array}$$

$$CY = 1$$



LOGICAL XOR :-

XRL dest-byte, source-byte

Function : Logical exclusive-OR for byte variables.

Flags : None

Description : performe the bitwise logical exclusive-OR operation b/w the indicated variables, storing the results in the destination.

1) XRL A, Rn

A \rightarrow XOR

Bytes : 1

Rn \rightarrow R0 to R7

Cycles : 1

A B. A XOR B.

0 0 0

0 1 1

1 0 1

1 1 0

operation: (A) \leftarrow (A) \vee (Rn)

Flags : None

Ex:- XRL A, R3.

MOV A, #39h

Before execution.

After execution.

MOV R3, #09h

A = 39h.

00111001 = A

A = 30h

XRL A, R3

R3 = 09h.

00001001 = R

R3 = 09h.

00110000 = A

2) XRL A, direct.

Bytes : 2

Cycles : 1

operation: (A) \leftarrow (A) \vee (direct)

Flags : None



40

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Ex:- XRL A, 30h.

MOV A, #39h

Before execution.

After execution.

MOV 30h, #09h.

A = 39h.

A = 30h.

XRL A, 30h.

30h = 09h.

30h = 09h.

$$\begin{array}{r} A = 39 \\ \text{#} \\ 30h = 09 \\ \hline A = 30 \end{array} \quad \begin{array}{l} = 0011\ 1001 \\ = 0000\ 1001 \\ \hline = 0011\ 0000 \end{array}$$

3) XRL A, @Ri

Ri \rightarrow R1 or R2

Bytes : 1

Cycles : 1

Operation: $(A) \leftarrow (A) \# ((R_i))$

Flags: None

Ex:- XRL A, @R1

MOV A, #39h.

Before execution.

After execution.

A = 39h

A = 30h.

MOV R1, #50h

R1 = 50h

MOV 50h, #09h.

50h = 09h.

XRL A, @R1.

4) XRL A, #data.

Bytes : 2

Cycles : 1

Operation: $(A) \leftarrow (A) \# \# \text{data}$

Flags: None



Ex:- XRL A, #09h.

mov A, #39h

XRL A, #09h.

Before execution.

A = 39h.

data = 09h.

After execution.

A = 30h.

5) XRL direct, A

Bytes : 2

Cycles : 1

Operation : (direct) \leftarrow (direct) $\text{A} \oplus (\text{A})$

Flags : None

Ex:- XRL 50h, A

mov A, #39h

mov 50h, #09h.

XRL 50h, A

Before execution.

After execution.

A = 39h

A = 30h.

50h = 09h.

6) XRL direct, #data.

Bytes : 3.

Cycles : 2.

Operation : (direct) \leftarrow (direct) $\text{A} \oplus \#data$

Flags : None.

Ex:- XRL 50h, #09h.

mov 50h, #39h.

XRL 50h, #09h.

Before execution

After execution.

50h = 39h.

50h = 30h.

data = 09h.



NOTE :-

XRL is used to check whether the two register have same value. If the value is same then '0' is placed in accumulator or location (destination).

ROTATE Instructions :-

1) RL A.

Function : Rotate Accumulator left.

Description : The eight bits in the accumulator are rotated one-bit to the left.

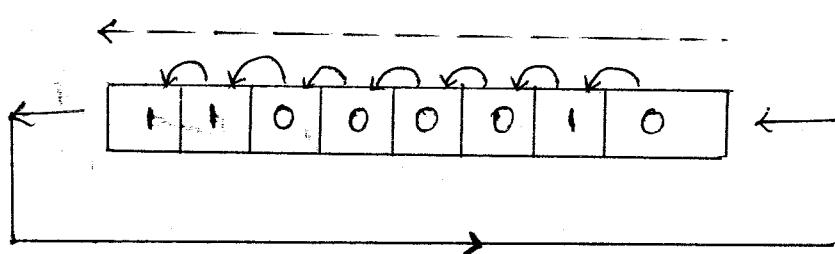
Bit - 7 is rotated into the bit - 0 position.

Flags : None

Bytes : 1.

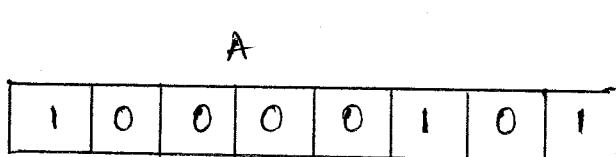
Cycles : 1.

Operation : $(A_{n+1}) \leftarrow (A_n)$ where $n=0-6$ } NOT clear
 $(A_0) \leftarrow (A_7)$



Before Execution.

$A = C2h$.



After execution.

$A = 85h$.

i.e. $A = 10000101b$.



2} RRA.

function : Rotate Accumulator Right

Bytes : 1

Cycles : 1

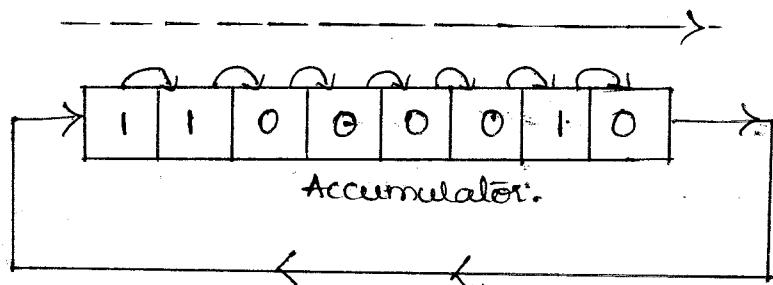
Flags : None

Description : The eight bits in the accumulator are rotated one bit to the right. Bit-0 is rotated into the bit-7 position.

operation : $(A_n) \leftarrow (A_{n+1})$, $n=0-6$

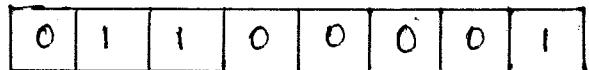
$(A_7) \leftarrow (A_0)$

Before Execution :-



After execution:-

$$A = 61h = 01100001$$



44

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

3) RLC A

Function : Rotate Accumulator left through the carry flag.

Flags : CY

Bytes : 1

Cycles : 1

Operation : $(A_{n+1}) \leftarrow (A_n)$,
 $(A_0) \leftarrow (C)$
 $(C) \leftarrow (A_7)$

Description: The 8-bits in the accumulator & the carry flag are together rotated one bit to the left.

Bit-7 moves into the carry flag.

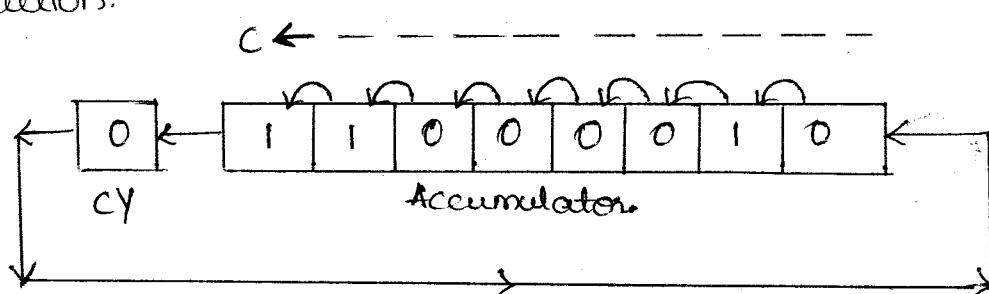
The original state of the carry flag moves into the bit-0 position.

Before execution.

$$A = 23h$$

$$A = 11000010b$$

$$C = 0$$

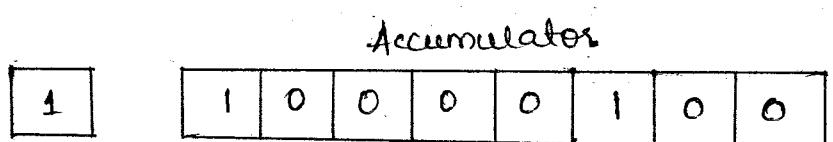


After execution:-

$$A = 84h$$

$$A = 10000100$$

$$C = 1$$



4) RRC A

Function : Rotate accumulator Right through the carry flag.

Bytes : 1.

Cycles : 1

Operation : $(A_n) \leftarrow (A_{n+1})$

$(A_7) \leftarrow (C)$

$(C) \leftarrow (A_0)$

Flags : CY

Description : The 8-bits in the Accumulator and the carry flag are together rotated 1-bit to the right.

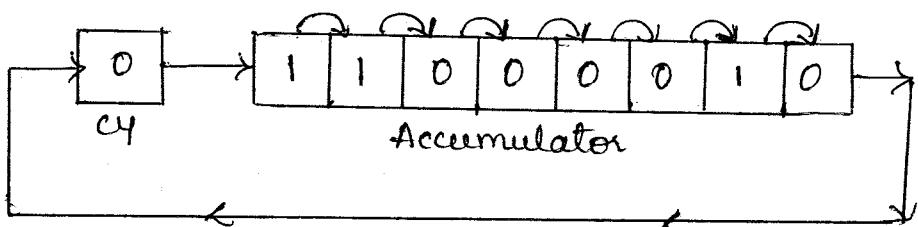
Bit-0 moves into the carry flag.

The original state of the carry flag moves into the bit-7 position.

Before execution:-

$$A = C2h = 11000010b$$

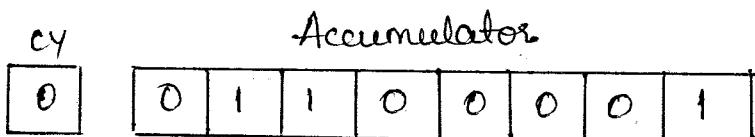
$CY = 0$



After execution:-

$$A = 01100001b = 61h$$

$CY = 1$



5) NOP

Function : No operation.

Flags : None

Description : * This instruction performs no operation & execution continues with the next instruction.

- * It is sometimes used for timing delays to waste clock cycles.
- * This instruction only updates the program counter (PC) to point to the next instruction following NOP.

Bytes : 1

Cycles : 1

Operation : $(PC) \leftarrow (PC) + 1$

Boolean Instructions :-

1) CLR A

Function : clear Accumulator

Description : The Accumulator is cleared ($A = 00h$)
All bits of the accumulator are set
to 0.

Bytes : 1

Cycles : 1



47

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

operation : $(A) \leftarrow 0$

Flags : None

Ex:-

MOV A, #FFh

Before execution

A = FFh.

After execution.

A = 00h.

CLR A

2) CLR bit

Function : clear bit

Description : This instruction clears a indicated bit.

* The bit can be the carry flag, or
any bit-addressable location in 8051.

Bytes : 1.

Cycles : 1.

operation : $(C) \leftarrow 0$

Flags : CY.

Ex:-

CLR C ; CY = 0

CLR P2.4; clear P2.5 (port 2's 5th pin is cleared)⁽⁶⁾

CLR P1.2; clear P1.2 (P1.2 = 0)

CLR ACC.7; clear D7 of accumulator (ACC.7 = 0)

Ex: 9) MOV A, #FFh.

CLR ACC.7

Before execution.

ACC = 11111111

After execution

ACC = 01111111



ii) CLR C

Before execution.

$$CY = 1$$

After execution.

$$CY = 0$$

iii) CLR P1.2

Before execution.

$$\text{port 1} = 0010\ 0111$$

After execution.

$$\text{port 1} = 0010\ 0011$$

3) CPL A

Function : complement Accumulator.

Description : This instruction complements the contents of the Accumulator.

- * The result is 1's complement of the accumulator i.e. 0's become 1's & 1's become zero.

Bytes : 1.

Cycles : 1.

Operation : $(A) \leftarrow \overline{A}$

$$\begin{array}{l} \overline{A} \\ \overline{\overline{A}} \\ \overline{\overline{\overline{A}}} \end{array}$$

Ex:- CPL A.

i) MOV A, #FFh. Before execution After execution.

CPL A

$$A = FFh$$

$$A = 00h$$

ii) MOV A, #00h.

CPL A

$$A = 00h$$

$$A = FFh$$



49

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

4} CPL bit

Bytes : 2.

Cycles : 1

Function : complement bit

Description : * This instruction complements a specified single bit.

* The bit can be any bit addressable location in 8051.

Flags : None.

Ex :- CPL P0.3.

Before execution.

P0.3 = 1 (I/p pin)

After Execution.

P0.3 = 0 (O/p pin).

CPL P3.3.

Before execution.

P3.3 = 0 (O/p pin)

After Execution.

P3.3 = 1 (I/p pin).

5} CPL C

Function : complements carry bit

Operation : $(C) \leftarrow \neg(C)$

Bytes : 1

Cycles : 1

Flags : CY



50

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Before execution.

i) CY = 0

ii) CY = 1.

After execution.

i) CY = 1.

ii) CY = 0.

6} SETB C

Bytes : 1.

Cycles : 1

Operation : (C) \leftarrow 1.

Function : Sets the carry bit

Flags : CY.

Ex:-

Before execution.

i) CY = 0

ii) CY = 1.

After execution.

i) CY = 1

ii) CY = 1

7} SETB bit

Bytes : 2

Cycles : 1

Operation : (bit) \leftarrow 1.

Function : Set specified bit.

Ex:- SETB P1.0

Before execution.

P1.0 = 0 (0LP pin)

After execution.

P1.0 = 1 (1LP pin).



JUMP & CALL Instructions

- * Jump & call instructions change the flow of the program by changing the contents of program counter.
- * A jump permanently changes the program flow whereas call temporarily changes the program flow to allow another part of the program to run.
- * Jump instructions are classified into
 - 1) conditional Jump.
 - 2) Unconditional Jump.
- * The jump instruction which changes the program flow if certain condition exists is called conditional Jump.
- * The jump instruction which changes the program flow irrespective of the condition (NOT depends on any condition) is called Unconditional Jump.



compare Instruction :- CJNE (compare & Jump if not equal)

This instruction compares the magnitude of the first two operands, & changes program flow if their values are not equal.

The instructions that change the program flow are :-

- * Jump on bit conditions.
- * compare byte & Jump if not equal.
- * Decrement byte & Jump if zero.
- * Jump unconditionally.
- * Call a subroutine
- * Return from a subroutine.

1) JB bit, rel. (rel \rightarrow relative address)

Function : Jump if Bit set.

Description : If the indicated bit is a one (1), Jump to the address indicated ; otherwise proceed with the next instruction.

The bit tested is not modified.

Bytes : 3

Cycles : 2

Operation : $(PC) \leftarrow (PC) + 3$.

If (bit) = 1

Then, $(PC) \leftarrow (PC) + \text{rel}$.



Flags : None.

Ex :- Address.

0300 JB ACC.0, down.
0301
0302.
0304. — down : INC 90 ←
0305

i) Before execution.

PC = 0300h.

Say ACC.0 = 1.

After execution.

ACC.0 = 1

PC = 0304h.

ii) PC = 0300h.

Say ACC.0 = 0.

ACC.0 = 0,

PC = 0301

NOTE:

i) If condition is false then, processor executes next instruction.

ii) If condition is True. then, processor jumps to the specified address (label) & executes that address instruction.

iii) SETB P1.5

Here : JB P1.B, Here

MOV P2, #55h

iii) JB ACC.0, Next

INC A

Next : DEC A

iv) SETB P1.4

Here : JB P1.A, Here

MOV A, B.



2) JNB bit, rel

Function : Jump if bit NOT set.

Description : If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction.

The bit tested is not modified.

Flags : None

Bytes : 3.

Cycles : 2.

Operation : $(PC) \leftarrow (PC) + 3$

If $(bit) = 0$

then,

$(PC) \leftarrow (PC) + \text{rel.}$

Ex:-

Say address

0300 JNB ACC.0 , down.

0301 INC R0.

0302

0303. down:dec R1.

Before execution.

After execution.

i) $PC = 0300h.$

ACC.0 = 1

$PC = 0301h.$

Say ACC.0 = 1.

Here $ACC.0 = 1$, the condition is false, so processor executes next instruction ie INC R0.



ii) $PC = 0300h$. $ACC.0 \neq 0$
say $ACC.0 = 0$. $PC = 0303$.

Here the condition is True, so processor jumps to the address specified by the label & executes that instruction.

ii) SETB ACC.0	iii) JNB ACC.0, Next	iv) CLR A
here: JNB ACC.0, here	INC A	here: JNB ACC.0, here
MOV R2, R3.	Next : DEC A.	MOV R2, R3.

3) JBC bit, sel.

function : Jump if bit is set & clear bit.

Description: If the desired bit is high it will jump to the target address (specified address ie. label), at the same time the bit is cleared to zero.

* If the desired bit is low, then processor proceeds with the next instruction ie, executes next instruction.

Bytes : 3.

Cycles : 2.

operation : $(PC) \leftarrow (PC) + 3$.

If $(bit) = 1$

Then $(bit) \leftarrow 0$.



56

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

$(PC) \leftarrow (PC) + \text{rel.}$

ex:-

0300	SETB	ACC.F
0301	JBC	ACC.F, Next
0302	MOV	P1,A
0303.		
Next : INC R0.		

Before execution.

ACC.F = 1
PC = 0301.

After execution.

ACC.F = 0
PC = 0303.

4) JC target or JC rel

function : Jump if carry is set i.e, CY=1.

Flags : None.

Description : If the carry flag is set, branch to the address indicated ; otherwise proceed with the next instruction.

Bytes : 2.

Cycles : 2.

operation : $(PC) \leftarrow (PC) + 2$

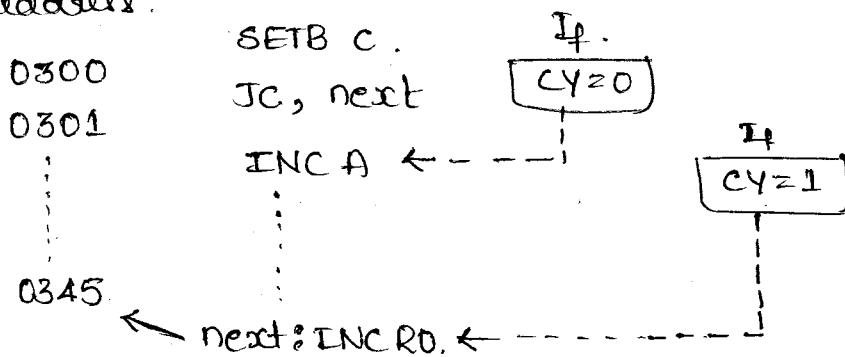
If $(CY)=1$

then.

$(PC) \leftarrow (PC) + \text{rel.}$



Ex:- Say address.



- * If $\boxed{CY=1}$, then Jumps to address specified by label.
- * If $\boxed{CY=0}$, then executes next instruction.

Before execution

$\boxed{CY=1}$

PC = 0301

After execution.

$CY=1$

PC = 0345.

5) JNC target or JNC sel.

Function : Jump if NO carry ($CC=0$)

Description : This instruction checks the carry flag, & if $CC=0$. it will jump to the target address.

Bytes : 2.

Cycles : 2.

Operation : $(PC) \leftarrow (PC) + 2$

If $(CC)=0$

Then

$(PC) \leftarrow (PC) + \text{sel}$

Flags : None

Ex:- MOV A, #0Fh.

MOV B, #0F0h.

Add a,b

JNC, down.

INC R0.

Addc a, #30h.

JNC, down

down: mov 40h, a.

- * This instruction checks the carry flag, if $CY=0$ (condition true), then jumps to the specified label. If $CY=1$ (condition false), then executes next instruction.

6) JZ target or JZ rel

function : Jump if A=0 (Jump if Accumulator zero).

Flags : None.

Description : If all bits of the accumulator are zero, branch to the indicated address; otherwise proceed with the next inst.

Bytes : 2.

Cycles : 2

Operation : $(PC) \leftarrow (PC) + 2$

If $A=0$.

then, $(PC) \leftarrow (PC) + \text{rel}.$



Ex:- mov A, #01h.
 mov B, #02h.
 add A, B.
 jz , docon.
 inc R0.

down: mov 40h, A
 end

* If A=0, (condition is True) then Jumps to the address specified by the label down.
ie. mov 40h, A in above ex.

* If A≠0, (condition is false). then proceed with the next instruction.
ie. INC R0 in above ex.

7) JNZ target or JNZ rel.

Function : Jump if accumulator is NOT zero.

Flags : None.

Description : If any bit of the accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction.

i.e, If A≠0 (True condition), then branch (Jump) to the indicated address.

If A=0 (False condition), then proceed with the next instruction.



Bytes : 2

Cycles : 2.

Operation : $(PC) \leftarrow (PC) + 2$.

If $A \neq 0$

Then $(PC) \leftarrow (PC) + \text{rel.}$

Ex :- MOV A, #0Fh.

MOV B, #0E0h.

ADD A, B.

JNZ, down.

INC R0

ADDC A, #01h.

JNZ, down.

INC RL.

down : MOV 40h, A.

- * If $A \neq 0$ (true condition), then jump to the address specified by the label down i.e. $\text{MOV } 40h, A$.
- * If $A = 0$ (false condition), then proceed with the next instruction i.e. $\text{INC } R0$ and $\text{INC } RL$.

3) DJNZ byte, target or DJNZ byte, rel - address.

Function : Decrement and jump if not zero.

Flags : None.

Description : In this instruction a byte is decremented, & if the result is NOT zero it will jump to the target address.



DJNZ Rn, target (where n=0 to 7)

Bytes : 2

Cycles : 2.

Operation : $(PC) \leftarrow (PC) + 2$

$(R_n) \leftarrow (R_n) - 1$

If $(R_n) \neq 0$ i.e. $(R_n) > 0$ or $(R_n) < 0$.

Then.

$(PC) \leftarrow (PC) + \text{rel.}$

Ex:-

MOV R0, #30h

MOV R1, #40h.

MOV R3, #05h.

up: MOV A, @R0

ADDC A, #01h

MOV @R1, A

INC R1

INC R0.

DJNZ R3, up

end.

- * 1st decrements the contents of R3 register & then checks the condition i.e. R3 ≠ 0. If R3 contents is not zero, then jumps to the specified address indicated by the label.
- * If R3 = 0, (condition is false) then executes the next instruction. i.e. end.



9) DJNZ direct, rel

Bytes : 2

Cycles : 2

Flags : None

operation: $(PC) \leftarrow (PC) + 2$

$(\text{direct}) \leftarrow (\text{direct}) - 1$

If $\text{direct} \neq 0$ ie. $(\text{direct}) > 0$ or $(\text{direct}) < 0$.

Then.

$(PC) \leftarrow (PC) + \text{rel.}$

Ex:- mov R0, #30h.

mov R1, #40h.

mov 40h, #05h

up: mov A, @R0

add A, #0h.

mov @R1, A

INC R1

INC R0.

DJNZ 40h, up

end.

* 1st decrements the contents of 40h (address), then checks the condition ie $(40) \neq 0$ if contents of 40h address is NOT zero, then jumps to the specified address indicated by the label.

* If $(40)h = 0$ (condition is false) then executes the next instruction i.e., end.



NOTE :-

- * The target address can be no more than 128 bytes backward or 127 bytes forward, since it is a 2-byte instruction.

10) SJMP rel OR SJMP 8-bit address

Function : short Jump

Description : program control branches unconditionally to the address indicated.

Bytes : 2

Cycles : 2

Operation : $(PC) \leftarrow (PC) + 2$.

$(PC) \leftarrow (PC) + \text{rel}$

Range : -128 bytes to +127 bytes.

Flags : None.

Ex:-

0a9 00h.

0a9 00h.

SJMP OVER

SJMP 30h.

0a9 30h.

0a9 30h.

OVER : MOV A, #10h.

11) LJMP 16-bit address.

Function : Long Jump

Description : LJMP causes an unconditional branch to the indicated address, by loading the high order & low order bytes of the PC respectively.



Bytes : 3.

Cycles : 2.

Operation : $(PC) \leftarrow \text{address}_{(0-15)}$

Flags : None

Range : -32768 bytes to +32767 bytes.

NOTE :-

The destination may therefore be anywhere in the full 64K bytes program address space.

12. JMP @A + DPTR

Function : Jump indirect.

Description : The JMP instruction is an unconditional jump to a target address. The target address is provided by the total sum of register A & the DPTR register.

Bytes : 1.

Cycles : 2.

Operation : $(PC) \leftarrow (A) + (DPTR)$

Flags : None.

NOTE : This instruction is not widely used.

* Jump Instructions are classified into.

1) conditional Jump &

2) Unconditional Jump.



1) Conditional Jump :-

- * Depending upon the condition i.e. True or False, program branches (Jumps) to the specified address.
i.e., If condition is true \rightarrow then JMP to specified label.
If condition is false \rightarrow NO Jump. Executes Next instruction.
- * All conditional Jumps are short Jumps, meaning that the target address cannot be more than -128 bytes backward to +127 bytes forward of the PC of the instruction following the Jump.
- * If the target address is beyond the -128 to +127 byte range, the assembler gives an error.
- * If the target address is beyond the -128 to +127 byte range, the assembler gives an error.

2) Unconditional Jump :-

Jumps to the specified address without any condition (unconditionally Jump to the specified address).

The unconditional Jump instructions are

- 1) SJMP 8-bit address.
- 2) LJMP 16-bit address.
- 3) JMP @A+DPTR.



1) SJMP :-

- * This is a 2-byte instruction. The 1st byte is the opcode & the second byte is the signed number, which is added to the PC of the instruction following the SJMP to get the target address.
- * ∵ In this Jump the target address must be within -128 to +127 bytes of the PC of the instruction.

2) LJMP :-

- * This is a 3-byte instruction. The 1st byte is the opcode & the next two bytes are the target address.
- * The LJMP is used to jump to any address location within the 64 kbytes code space of 8051.

3) JMP @ A + DPTR :-

- * The target address is provided by the total sum of register A & the DPTR register.
- * This instruction is not widely used.

conditional Jump's :-

465 - Mazidi



(67)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

1) PUSH direct :-

Function : Push onto stack.

Description : The stack pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location, addressed by the stack pointer.

Flags : None.

Bytes : 2

Cycles : 2.

Operation : $(SP) \leftarrow (SP) + 1$
 $((SP)) \leftarrow (\text{direct})$

NOTE :-

This instruction supports only direct addressing mode. ∵ The instruction such as "PUSH A" or "PUSH R3" are illegal instructions.

Ex:- i) PUSH 0E0h.

where 0E0h is the RAM address belonging to register A.

ii) PUSH 03h.

where 03h is the RAM address of R3 or Bank 0.



(68)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

2} POP direct.

Bytes : 2.

Cycles : 2.

Function : POP from the stack.

Operation : This copies the byte pointed to by SP (stack pointer) to the location where direct address is indicated & decrements SP by 1.

Flags : None.

Operation : $(\text{direct}) \leftarrow (\text{SP})$

$(\text{SP}) \leftarrow (\text{SP}) - 1$.

NOTE :-

This instruction supports only direct addressing mode. ∴ The instructions such as "POP A" or "POP R3" are illegal instructions.

Ex:- i} POP 0E0h.

where 0E0h is the RAM address belonging to register A.

ii} POP 03h.

where 03h is the RAM address of R3 of Bank 0.



1) ACALL target address.

Function : Absolute call. Transfers control to a Subroutine.

Description : * Acall unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes 16-bit result onto the stack (low order byte 1st) & increment the stack pointer to store high-order byte.

- * Acall is a 2-byte instruction, in which 5-bits are used for the opcode & the remaining 11-bits are used for the target subroutine address.
- * A 11-bit address limits the range to 2K-bytes.

Bytes : 2.

Cycles : 2.

Flags : None.

Operation : $(PC) \leftarrow (PC) + 2.$

$(SP) \leftarrow (SP) + 1.$

$(SP) \leftarrow (PC_{15-0})$

$(SP) \leftarrow (SP) + 1$

$(SP) \leftarrow (PC_{15-8})$

$(PC_{10-0}) \leftarrow \text{page address.}$



70

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

2) LCALL 16-bit address.

Function : Long call. Transfers control to a subroutine.

Bytes : 3.

Cycles : 2

Flags : None.

Description : Lcall calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction & then pushes the 16-bit result onto the stack (1st lower byte) incrementing the stack pointer by 2 & pushes higher byte.

- * The subroutine may ∵ begin anywhere in the full 64 K-byte program memory address space.

Operation : $(PC) \leftarrow (PC) + 3.$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC_{7-0})$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC_{15-0})$

$(PC) \leftarrow \text{address}_{0-15}.$



CALL Instructions :- (Lcall & Acall).

* call instruction transfers control to a subroutine.

There are two types of call:

1) ACALL &

2) LCALL.

1) ACALL :- * ACALL is a 2-byte instruction.

* In Acall, the target address is within 8K bytes of the current program counter (PC).

* If a subroutine is called, the PC (which has the address of the instruction after the Acall) is pushed onto the stack, & the stack pointer (SP) is incremented by 2.

* Then the program counter (PC) is loaded with the new address & control is transferred to the subroutine.

* At the end of the procedure (subroutine), when RET instruction is executed, PC is popped off the stack, which returns control to the instruction after the CALL.

2) LCALL :-

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



⇒ RET :-

Function : Return from subroutine.

Description : This instruction is used to return from a subroutine previously entered by inst LCALL or ACALL. The top two bytes of the stack are popped in the program counter (PC) & program execution continues at this new address.

- * After popping the top two bytes of the stack into the program counter, the stack pointer (SP) is decremented by 2.

Flags : None

Bytes : 1

Cycles : 2.

Operation : $(PC_{15-8}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

$(PC_{7-0}) \leftarrow ((SP))$

$(SP) \leftarrow (SP) - 1$

⇒ RETI :-

Function : Return from interrupt.

Bytes : 1.

Cycles : 2

Flags : None



Description : This is used at the end of an interrupt service routine (interrupt handler). The top two bytes of the stack are popped into the program counter (PC), the stack pointer (SP) is decremented by 2.

operation :

$$(PC_{15-8}) \leftarrow ((SP))$$
$$(SP) \leftarrow (SP) - 1$$
$$(PC_{7-0}) \leftarrow ((SP))$$
$$(SP) \leftarrow (SP) - 1.$$

NOTE:

The RET instruction is used to at the end of a subroutine associated with the ACALL & LCALL instructions, RETI must be used for the interrupt service subroutine.



Compare & Jump Instructions

CJNE dest-byte, source-byte, target.

Function : compare and jump if not equal.

Description : The magnitudes of the source byte & destination byte are compared. If they are not equal, it jumps to the target.

Flag : CY.

1) CJNE A, direct, rel address.

Bytes : 3.

Cycles : 2.

Operation: $(PC) \leftarrow (PC) + 3$

If $(A) \neq (\text{direct})$

then.

$(PC) \leftarrow (PC) + \text{relative address.}$

If $(A) \neq (\text{direct})$

then.

$(C) \leftarrow 1$

Else.

$(C) \leftarrow 0$

Flags : CY.

2) CJNE A, #data, rel address.

Bytes : 3.

Cycles : 2

Flags : CY.



operation : $(PC) \leftarrow (PC) + 3.$

If $(A) \neq \text{data}$.

Then.

$(PC) \leftarrow (PC) + \text{relative address}.$

If $(A) = \text{data}$

Then.

$(C) \leftarrow 1.$

Else

$(C) \leftarrow 0$

3) CJNE Rn, #data, rel.

Bytes : 3.

Cycles : 2.

operation: $(PC) \leftarrow (PC) + 3.$

If $(Rn) \neq \text{data}$

THEN

$(PC) \leftarrow (PC) + \text{relative address}$

If $(Rn) = \text{data}$.

THEN

$(C) \leftarrow 1$

ELSE

$(C) \leftarrow 0$

Flags : Cy.



4) CJNE @R_i, #data, rel.

Bytes : 3

Cycles : 2

Flags : CY

Operation : (PC) \leftarrow (PC) + 3

IF ((R_i) \neq data)

THEN

(PC) \leftarrow (PC) + relative address

IF (R_i) \neq data

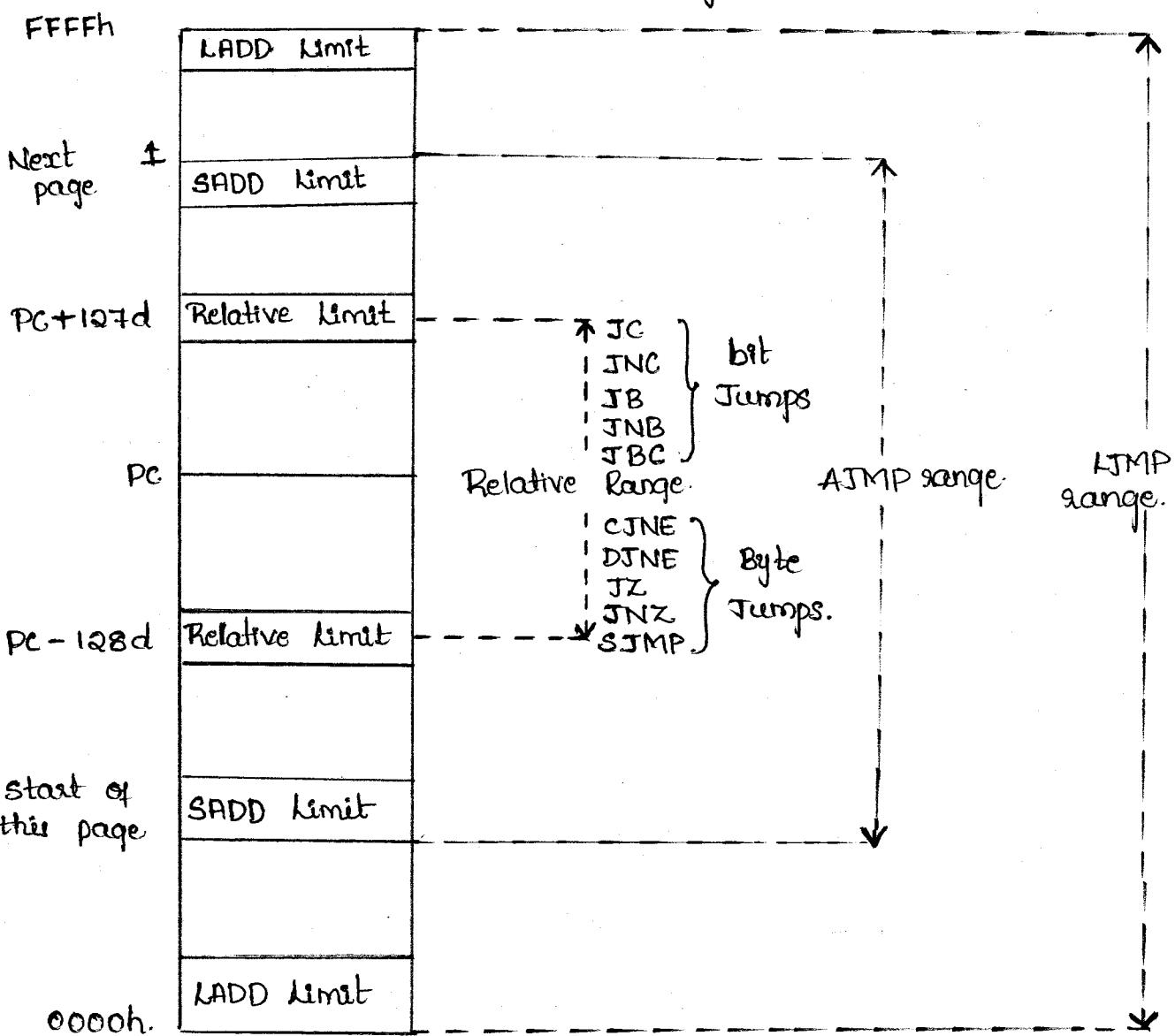
Then.

(C) \leftarrow 1

Else (C) \leftarrow 0.

JUMP & CALL INSTRUCTIONS :-

- * Jump & call instructions replaces the contents of program counter (PC) with New address & program execution to start from that new address.
- * The difference (in bytes) of this new address from address in program where jump or call instruction is called range of jump or call.



①

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

fig ① Jump Instruction Range.

SADD → short address.

LADD → Long address.

- * Jump or call instructions may have one of the 3 ranges :-

i) Relative range - +127d to -128d.

ii) Absolute range - within a page (2Kbyte)

iii) Long range - 0000h to FFFFh.

i) Relative range :-

- * The jump can be written within -128 bytes (for backward jump) or +127 bytes (for forward jump) of memory relative to the address of current program counter (PC).

- * Jump or call instruction with relative range will be of 2-byte instructions. The 1st byte is opcode & second byte is relative address of target location.

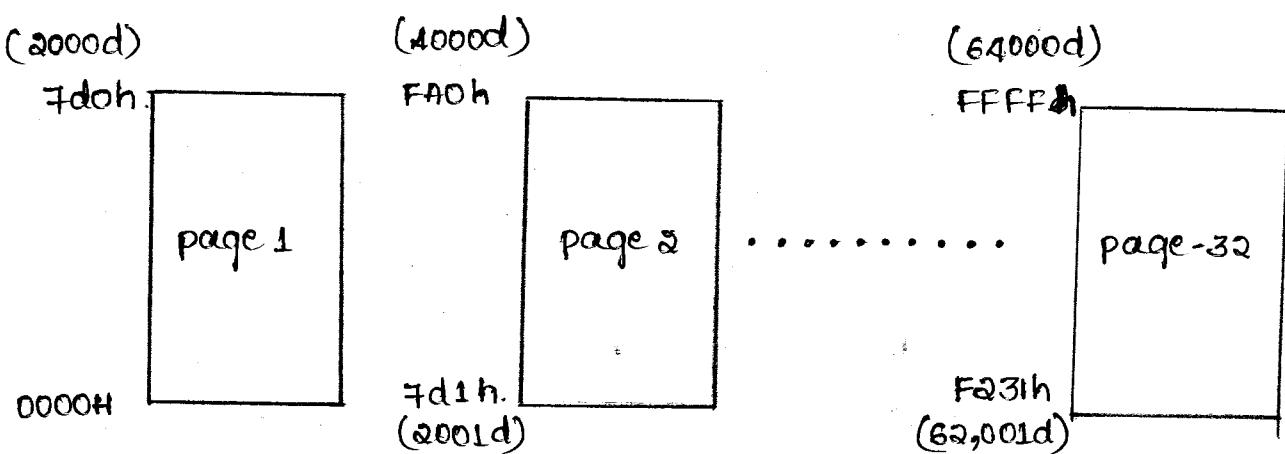
ii) Absolute Range :-

- * In 8051, program memory is divided into logical divisions called pages each of 2Kbyte.

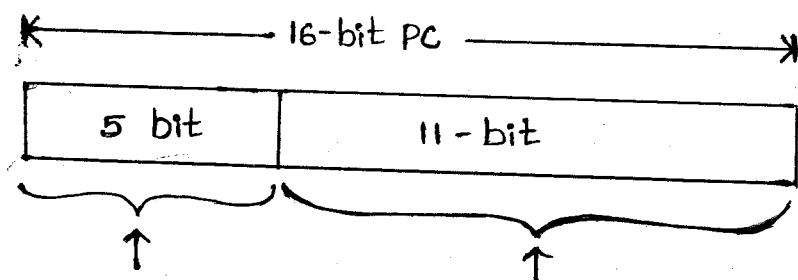
- * Maximum size program memory is 64Kbytes.
Size of each page is 2Kbytes.



∴ Maximum number of pages = $\frac{64 \text{ Kb}}{2 \text{ Kb}} = 32 \text{ pages.}$



- * In absolute range, Jump can be within a single page.
- * The upper 5-bits of PC holds the page number & lower 11-bits holds the address within that page.
i.e., $2^5 \rightarrow 32 \text{ page.}$
 $2^{11} \rightarrow 2 \text{ Kb range.}$



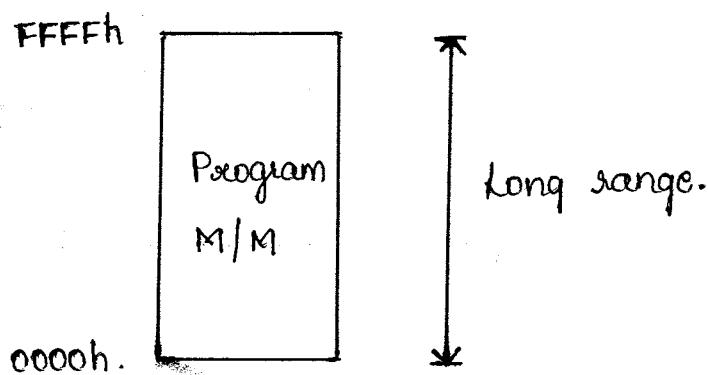
Holds page Number Holds address within page.



③

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

iii) Long Absolute Range :-



- * This range allows the jump to any where in the memory location from 0000h to FFFFh.
- * The Jump or call instructions with this range will be of 3 byte instructions in which 1st byte is opcode & 2nd & 3rd bytes represents the 16-bit address of target location.

Type of Jump or CALL	Range	No of bytes	example.
Relative Range	-128d to +127d.	2-byte instructions.	JC , JNC , JB , JNB , JBC , JZ , JNZ , DJNZ , CJNE .
Absolute range	Within a page (2Kbyte)	2- byte instructions.	ACALL .
Long Range.	Anywhere within program memory (0-FFFFh)	3 - byte instructions.	LCALL



④

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

I/O ports :-

- * A port is a pin where data can be transferred between 8051 and an external device. The 8051 has four 8-bit ports P₀, P₁, P₂ & P₃.
- * Each port has a D-type flip-flop for each pin & each port's pins can be either used as I/p or o/p pin under Software Control.

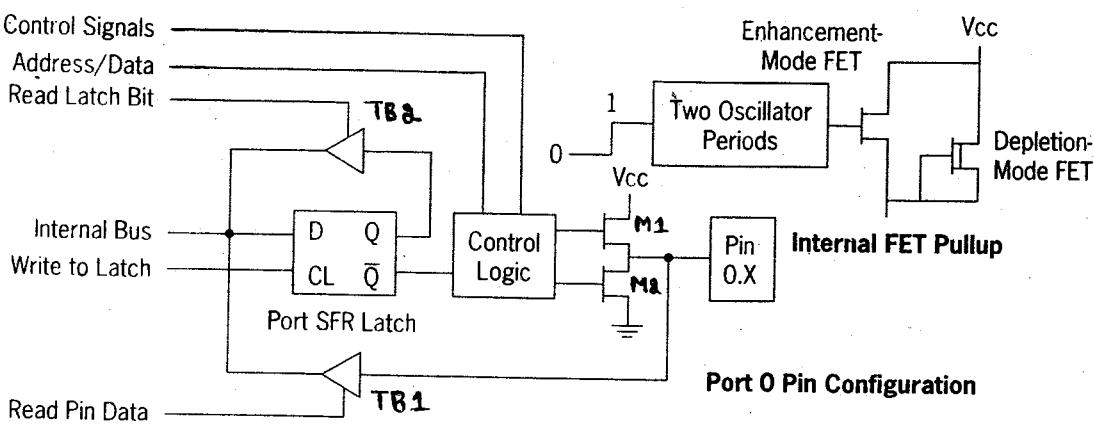
Port 0 :-

June - 07, 8M

- * Draw the sketch of the Pin Configuration of Port 0 pins & explain the various operations performed by the pins of Port 0.
- * Hardware Configuration of Port 0

June - 06, 5M

Jan - 07, 6M



- * Port 0 is an 8-bit, bit addressable I/p - o/p port. It is also used as a bi-directional low-order address & data bus for external memory.

I/p port:-

- * To use each pin as an I/p pin, 1st we must write '1' (logic



high) to that bit i.e.

- 1) Writing a 1 to the port 0 pin, the D-FF o/p is high i.e $Q=1$ & $\bar{Q}=0$.
- 2) Since $\bar{Q}=0$ is connected to the FET's gate M_1 & M_2 , thus turning OFF the both FET's.
- 3) When M_1 & M_2 are OFF, it acts like open circuit & there will be no connection between port 0 pin & ground, thus I/p Signal is directed to the tristate buffer TB1.

ex:- MOV A, #0FFh
 MOV P0, A.

o/p port :-

- * To use each pins' as an o/p pin, 1st we must write a '0' (logic zero) to that bit i.e.
- 1) Writing a '0' to the port 0 pin, the D-FF o/p is low i.e. $Q=0$ & $\bar{Q}=1$.
- 2) Since $\bar{Q}=1$ is connected to the FET's gate M_1 & M_2 , thus turning ON the both FET's.
- 3) When M_1 & M_2 are ON, it acts like short circuit, thus port pin is connected to the ground.
 \therefore Any attempt to read the I/p pin will always get the low ground Signal regardless of the status of the I/p pin.

ex:- MOV A, #00h
 MOV P0, A



Address / Data bus operation :-

- * Port 0 is also used to carry the multiplexed address / data bus during access to external memory.
- * When the 8051 access external memory, Port 0 pin carry the low-order address whenever the ALE has a Rising edge Signal i.e. 
- * port 0 lines are further used to carry the bi-directional data bus to Read & Write to external memory.

NOTE:-

* Port 0 Address \rightarrow 80h

* Port 0 bit address.

P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
87	86	85	84	83	82	81	80

P.T.O

NOTE :-

- To use the pins of port 0 as both I/O & O/P ports, each pin must be connected externally to a 10 k Ω resistor called Pull up Resistor.

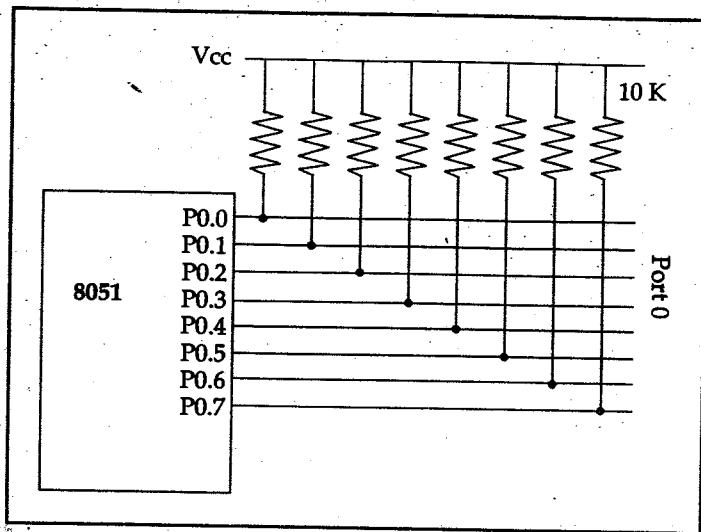


Figure Port 0 with Pull-Up Resistors

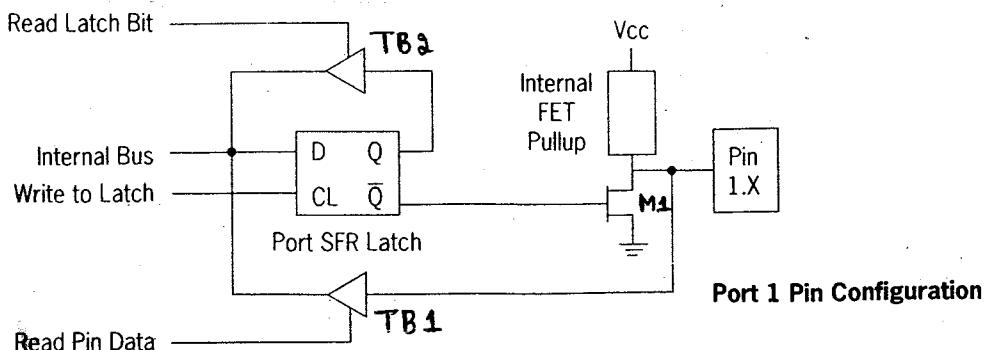
- Port 0 does not need a pull up resistor when used to access external memory.

Port 1 :-

Jan-09, 8M

- * Sketch the internal circuit diagram of port 1 of 8051 & briefly explain how to use it as I/O & O/P port. How does this port differ from other ports?
- * Explain with diagram the feature & operation of port 1

Jan-06, 6M



Port 1 Pin Configuration

I/o port :-

- * To use each pin has an I/p pin, first we must write a '1' (logic high) to that bit i.e.
- 1) Writing a '1' to the port 1 pin, the D-FF o/p is high i.e. $A=1$ & $\bar{A}=0$.
- 2) Since $\bar{A}=0$ & is connected to the FET gate M_1 , thus turning OFF the FET.
- 3) When M_1 is OFF, it acts like open circuit (it blocks any path to the ground) thus I/p Signal is directed to the tristate buffer TB1.

ex:-
Mov A, #0FFh
Mov P1, A

O/p port :-

- * To use each pin has an o/p pin, 1st we must write a '0' (logic low) to that bit i.e.
- 1) Writing a '0' to the port 1 pin, the D-FF o/p $A=0 \& \bar{A}=1$
- 2) Since $\bar{A}=1$ & is directly connected to the FET gate M_1 , thus turning ON the FET.
- 3) When M_1 is ON, it acts like short circuit, thus port pin is connected to the ground.
 \therefore Any attempt to read the I/p pin will always get the low ground Signal regardless of the status of the I/p pin.

ex:-
Mov A, #00h
Mov P1, A



NOTE :-

* Port 1 address \rightarrow 90h

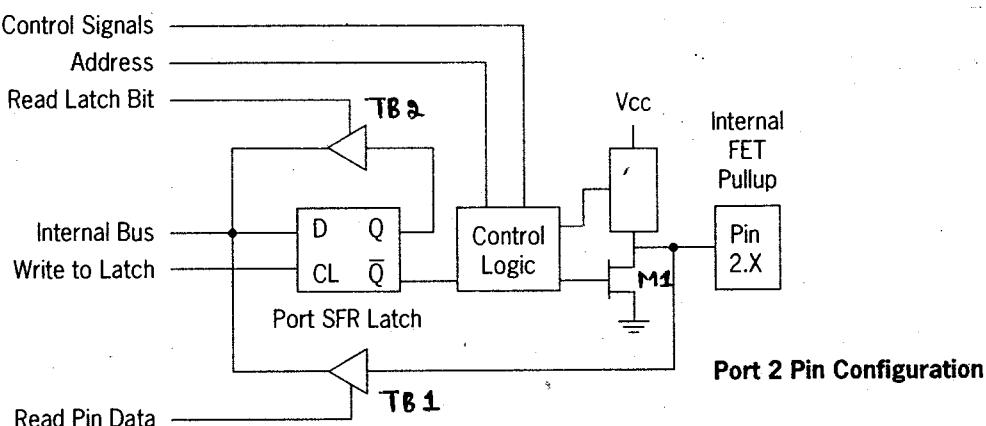
* Port 1 bit address :

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
97	96	95	94	93	92	91	90

* Port 1 pins don't have any dual function (only I/O & output function)

* Port 1 pins has Internal pull-up resistors.

Port 2 :-



* Port 2 pins can be used as

 > I/O & output port

 > Higher order address ($A_8 - A_{15}$)

I/O Port :-

O/I Port :-



⑥

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

- * Port 2 pins are used to carry the higher order address ($A_8 - A_{15}$) bus during external memory access.

i.e. during Rising edge of ALE Signal, Port 2 provides $A_8 - A_{15}$ address lines

NOTE :-

- * Port 2 address is $A_0 h$

- * Port 2 bit address

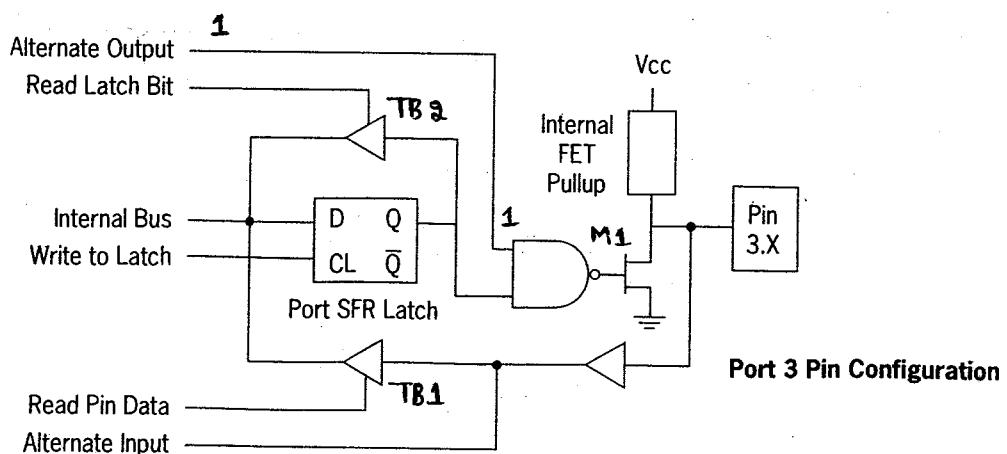
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
A7	A6	A5	A4	A3	A2	A1	A0

Port 3 :-

July-08, 10M

- * With the sketch of Pin Configuration of Port 3 pins & explain the various operations performed by the pins of Port 3.
- * Explain the functions of Port 3.

Jan-07, 6M



(7)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

I/p port :-

- * To use each pin as I/p pin, 1st we must write a '1'(logic high) to that bit i.e.
- 1) Writing a '1' to the port 3 pins, the D-FF o/p is high i.e. Q=1
- 2) Since Q=1 & is connected to one I/p of NAND gate & another I/p is '1' thus o/p of NAND gate is '0' & Turns OFF the FET.
- 3) When FET is OFF, it acts like open circuit, thus I/p Signal is directed to the tristate buffer TB1.

ex:- Mov A, #0FFh
 Mov P3, A

O/p port :-

- * To use each pin has an o/p pin, 1st we must write a '0' (logic low) to that bit i.e.
- 1) Writing a '0' to the port 3 pin, the D-FF o/p Q=0, thus o/p of NAND gate is '1' (high) & Turns ON the FET.
- 2) When FET is ON, it acts like short circuit, thus it provides the path to ground to the I/p pin.

∴ Any attempt to read the I/p pin will always gets the low ground Signal regardless of the State of the I/p pin.

ex:- Mov A, #0FFh
 Mov P3, A

NOTE :- For I/p - o/p operation, alternate o/p is always '1'

P.T.O.

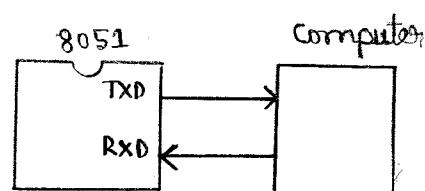


⑧

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Alternate Function of port 3:

Pin	Alternate Use
P3.0-RXD	Serial data input
P3.1-TXD	Serial data output
P3.2-INT0	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 pulse
P3.7-RD	External memory read pulse



RXD & TXD :-

* In 8051 mc RXD & TXD pins are used for Serial communication.

TXD :- The data is Transmitted out of 8051 through TXD Pin

RXD :- The data is Received by 8051 through the RXD pin.

INT0 & INT1 :- Interrupt 0 & Interrupt 1 are two Interrupt pins that are triggered by external circuits.

T₀ & T₁ :- The 8051 has two 16-bit Timers / counters.

T₀ → Timer 0 Register (16-bit)

T₁ → Timer 1 Register (16-bit)

* T₀ & T₁ can be used either as Timers to generate a time delay or as Counters to count events happening outside the microcontroller.

RD & WR :-

When RD = 0, microcontroller Reads the data from external RAM.

When WR = 0, microcontroller, Writes the data into external RAM.



NOTE :-

* Port 3 address is B0h

* Port 3 bit address

P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
B7	B6	B5	B4	B3	B2	B1	B0



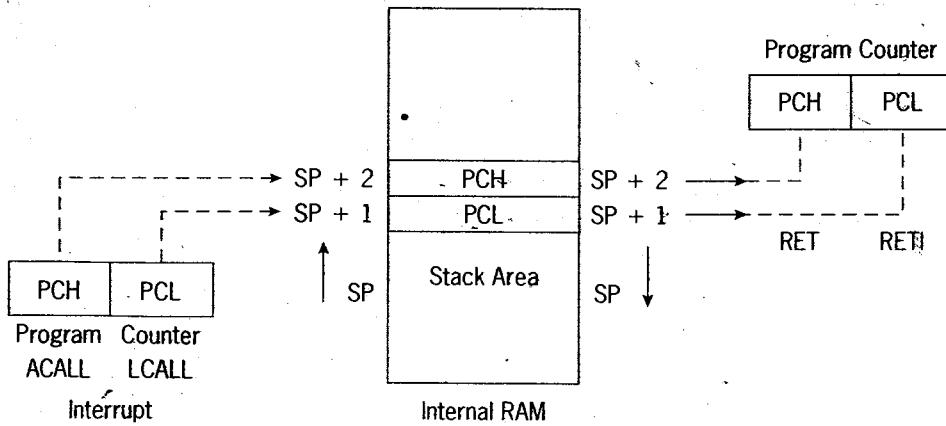
Subroutine :-

Subroutine is a program that may be used many times in the execution of a larger program.

(OR)

Subroutine are the programs that are often used to perform tasks that need to be performed frequently.

- * With the relevant Figure, write a Sequence of events that occurs in 8051 microcontroller when the CALL & RET instructions are executed Jan-10, 6M
- * Explain with a neat diagram, the Significance of Stack memory, whenever a CALL instruction is executed by the 8051 uc. June-08, 5M June-07, 5M



When call instruction is executed the following Sequence of events occurs:

- ⇒ The return address of the next instruction after the call instruction will be in the program counter (pc).
- ⇒ The return address are PUSHed onto the Stack i.e 1st lower



11

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

- byte then higher byte.
- 3) The Stack pointer is incremented for each push on the stack (thus Sp is incremented by 2)
 - 4) Then pc fetches the Subroutine address.
 - 5) The Subroutine is executed.
 - 6) After executing RET instruction at the end of the Subroutine, the return address is popped from the Stack & restored in PC.
 - 7) Then Stack pointer is decremented for each pop (thus Sp is decremented by 2).

* Differentiate between Jump & call Instructions.

July-06, 4M

Sl No	Jump Instruction	Call Instruction
1)	Jump Instructions permanently changes the program flow	call Instructions temporarily changes the program flow
2)	Jump Instructions won't store return address on Stack.	call Instructions store the return address on Stack.
3)	Jump Instructions branches (Jump) to the target address	call Instructions are used to call a Subroutine.
4)	Conditional Jump Instructions are available	Conditional Call Instructions are NOT available.
5)	Jump ranges i) Relative \rightarrow -128d to +127d ii) Absolute \rightarrow Within a page (3kb) iii) Long \rightarrow Anywhere within 64Kbyte	Call ranges i) A call \rightarrow within a page (3kb) ii) L call \rightarrow Anywhere within 64 Kb.



6>	ex:- SJMP, JNC, JNZ, JZ, JC JB etc	ex:- ACall Lcall
----	--	------------------------

- * Differentiate between Conditional Jump & unconditional Jump Inst's.

Sl No	Conditional Jump	Unconditional Jump
1>	All Conditional Jumps are Short Jumps i.e. -128d to +127d	Unconditional Jump may be LJMP & SJMP
2>	The Conditional Jump Inst's changes the program flow if certain condition exists	The unconditional Jump Inst's changes the program flow irrespective of the condition i.e. NOT depends on any condition
3>	ex:- JNZ, JZ, JNC, JC, JB etc	ex:- SJMP, LJMP.

- * Specify the memory area for bit level logical Instructions used in 8051 & list bit level logical Inst's. June-09, 5M
- * 20h to 2Fh is the memory area used for bit level logical Inst's
- * List of bit level logical Inst's are:

ANL C, bit	OR C, bit	CPL C
ANL C,/bit	OR C,/bit	



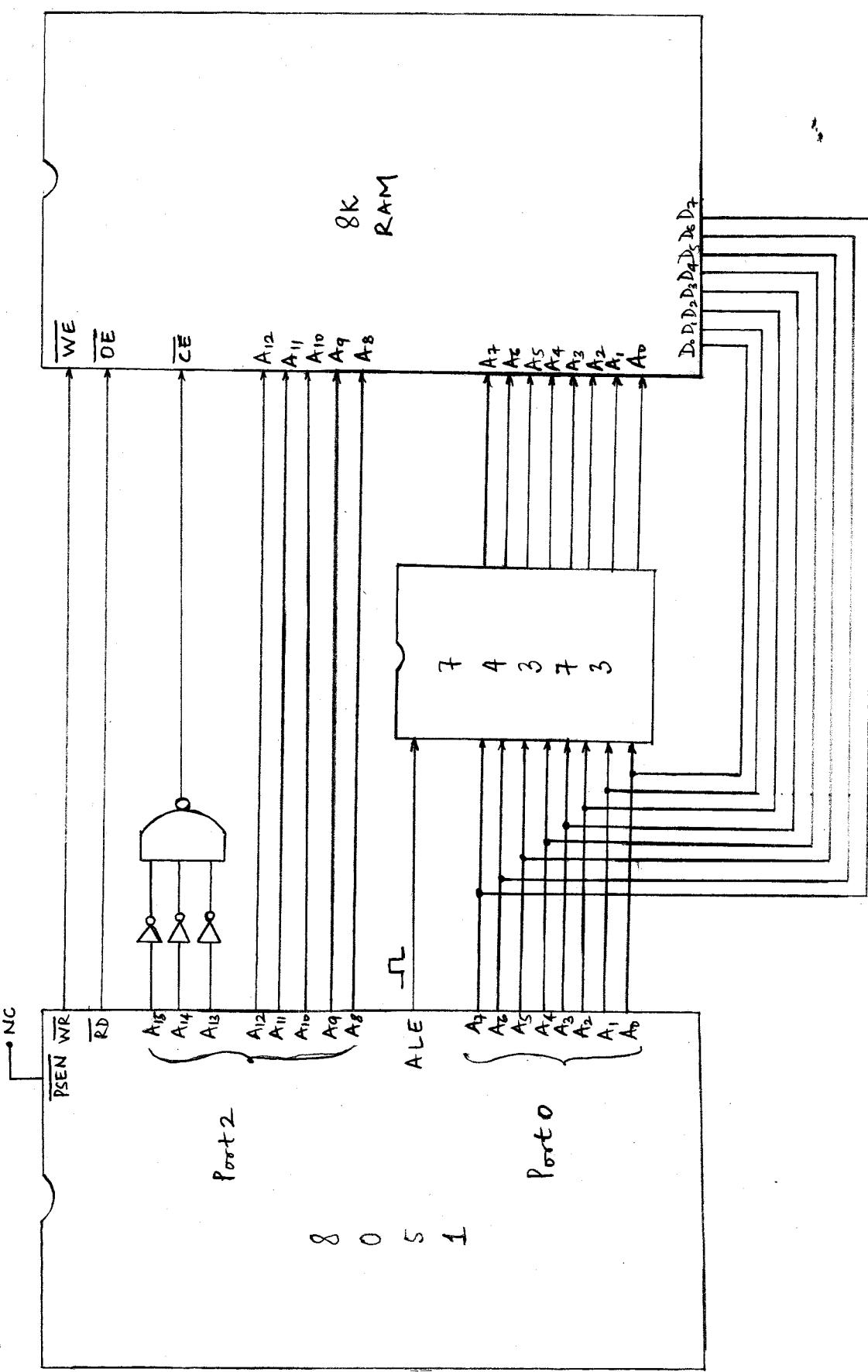


Fig: Interfacing 8K RAM to 8051 Microcontroller.



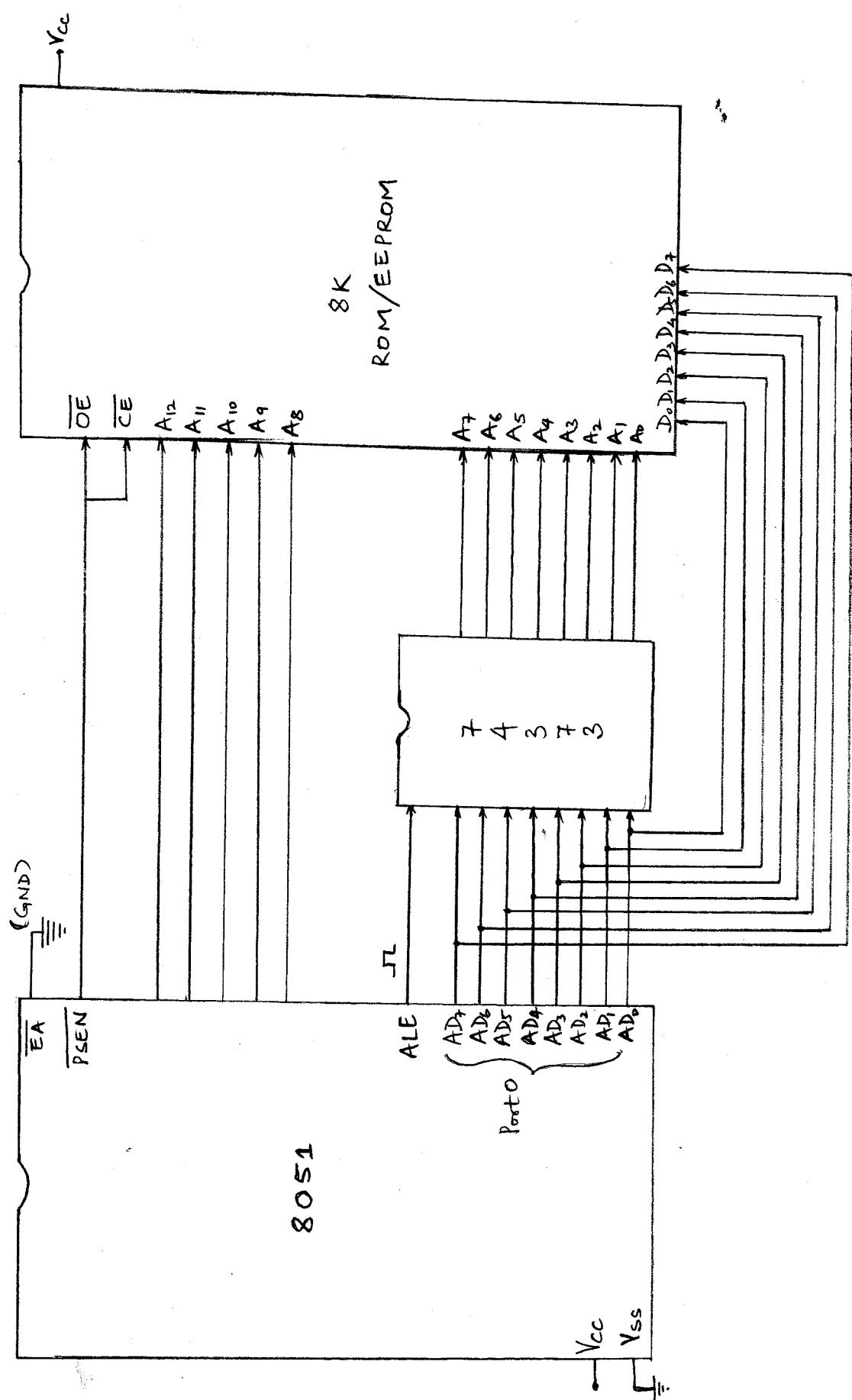


fig: Interfacing 8K ROM to 8051 microcontroller.



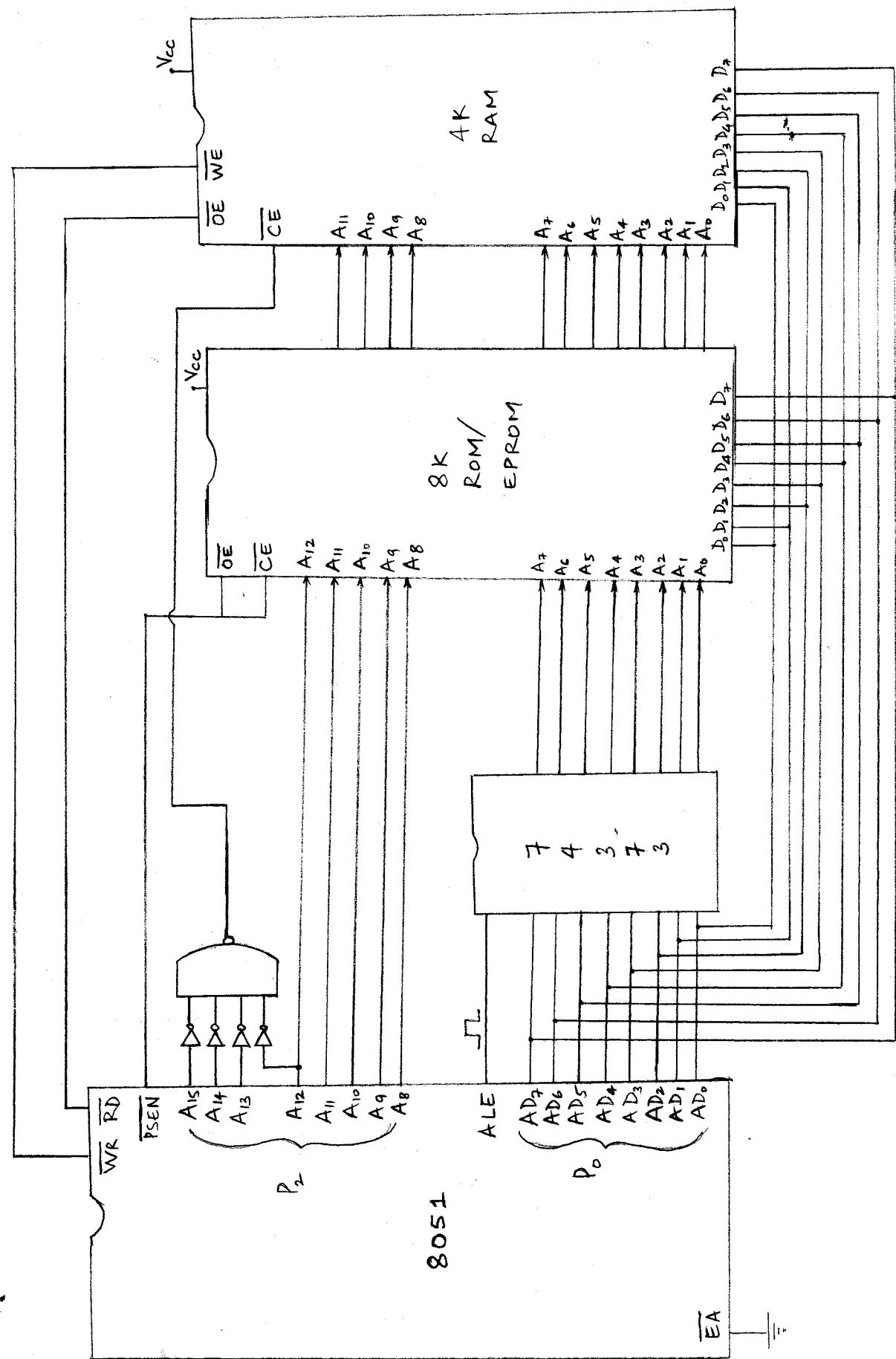


Fig: Interfacing 8K ROM + 4K RAM to 8051 Microcontroller.



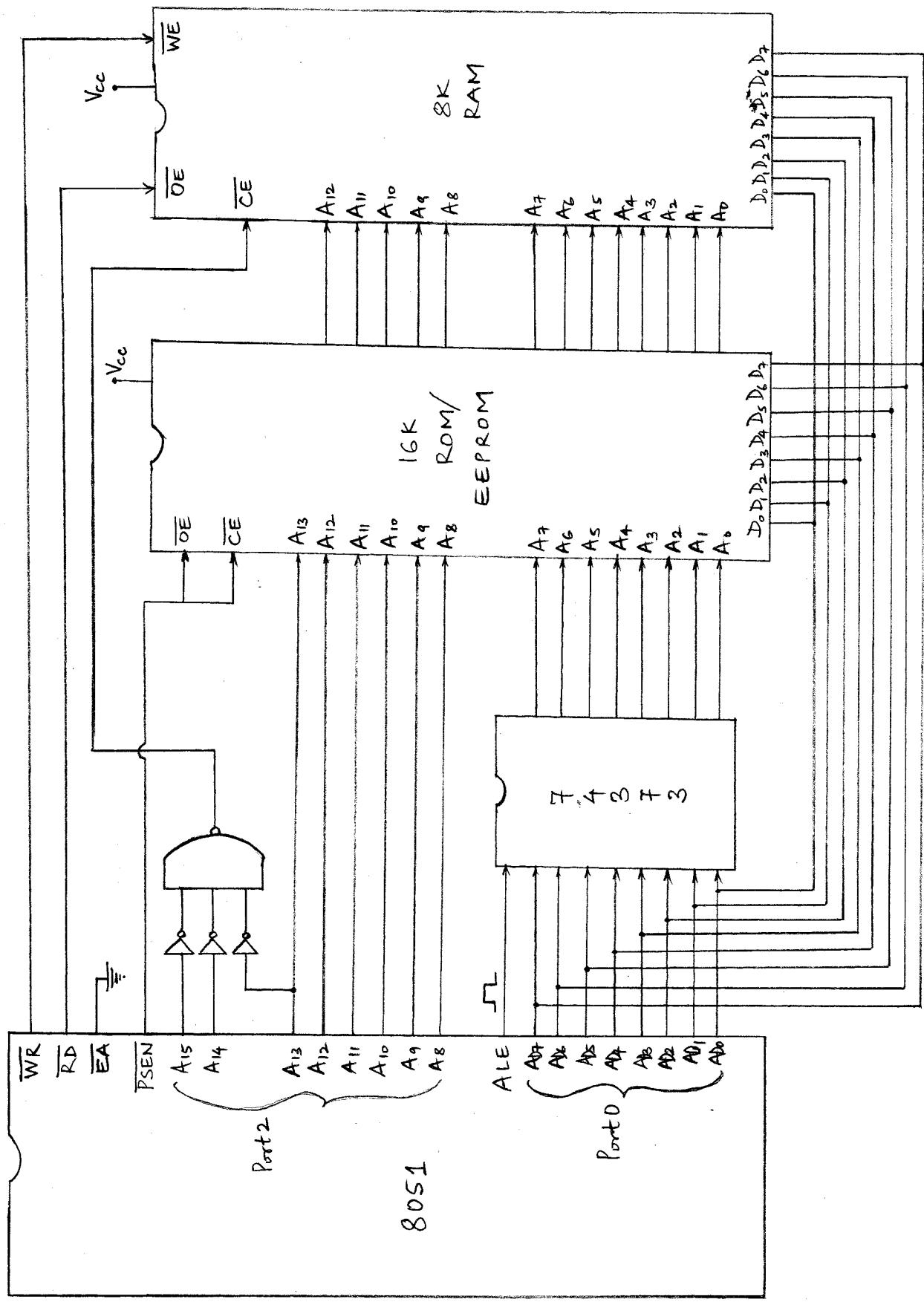
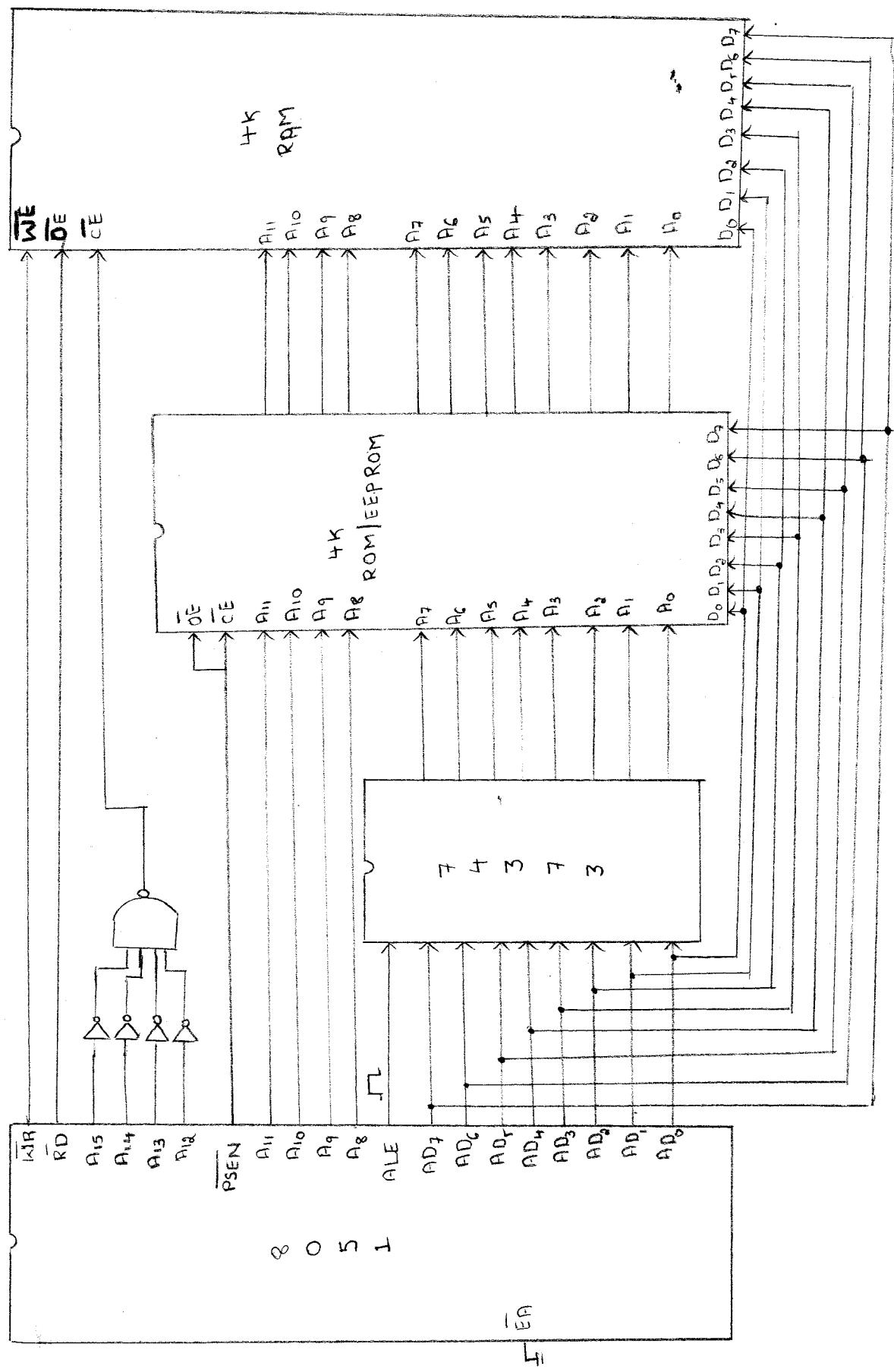


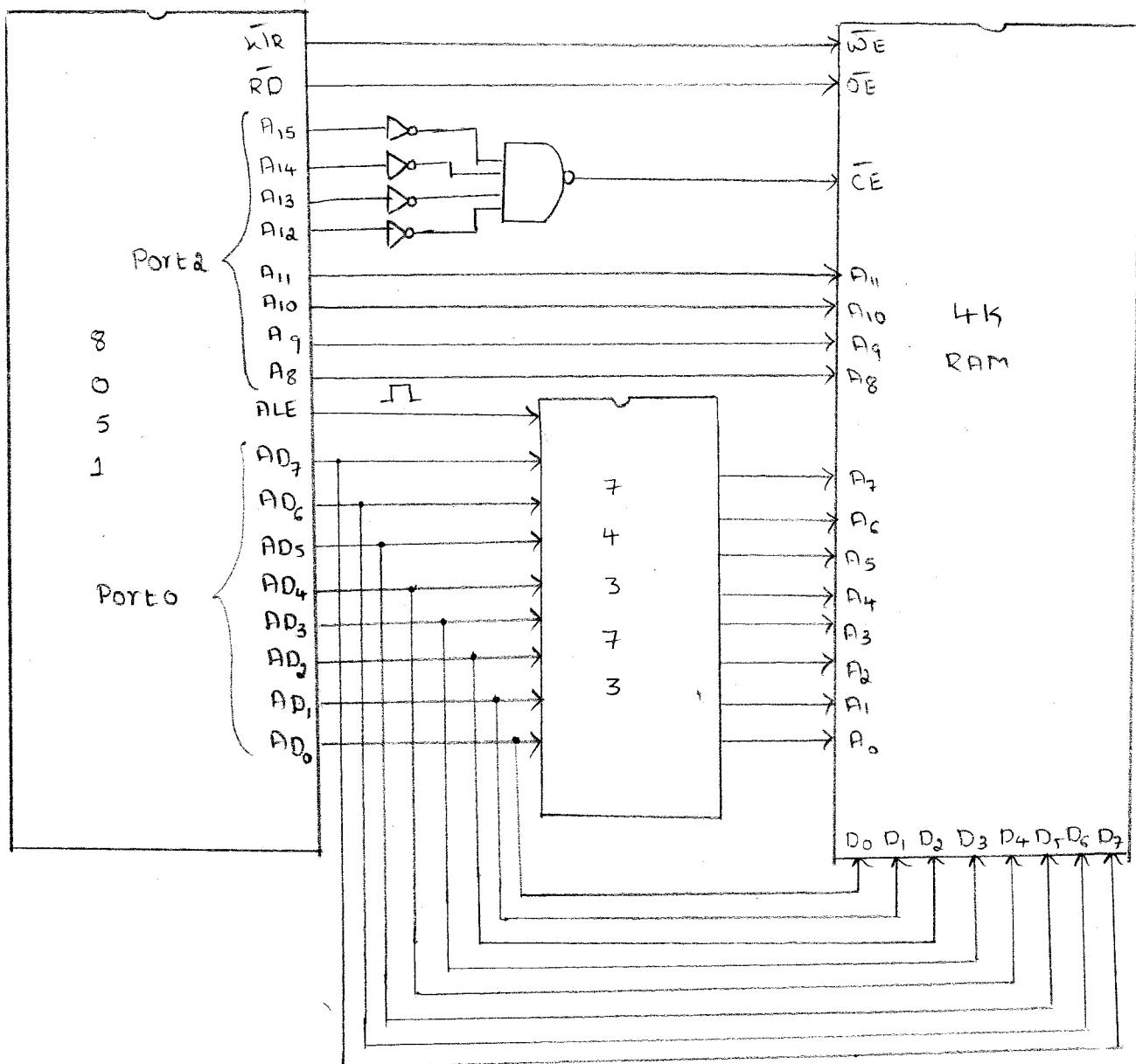
Fig: Interfacing 16K ROM & 8K RAM to 8051 Microcontroller.



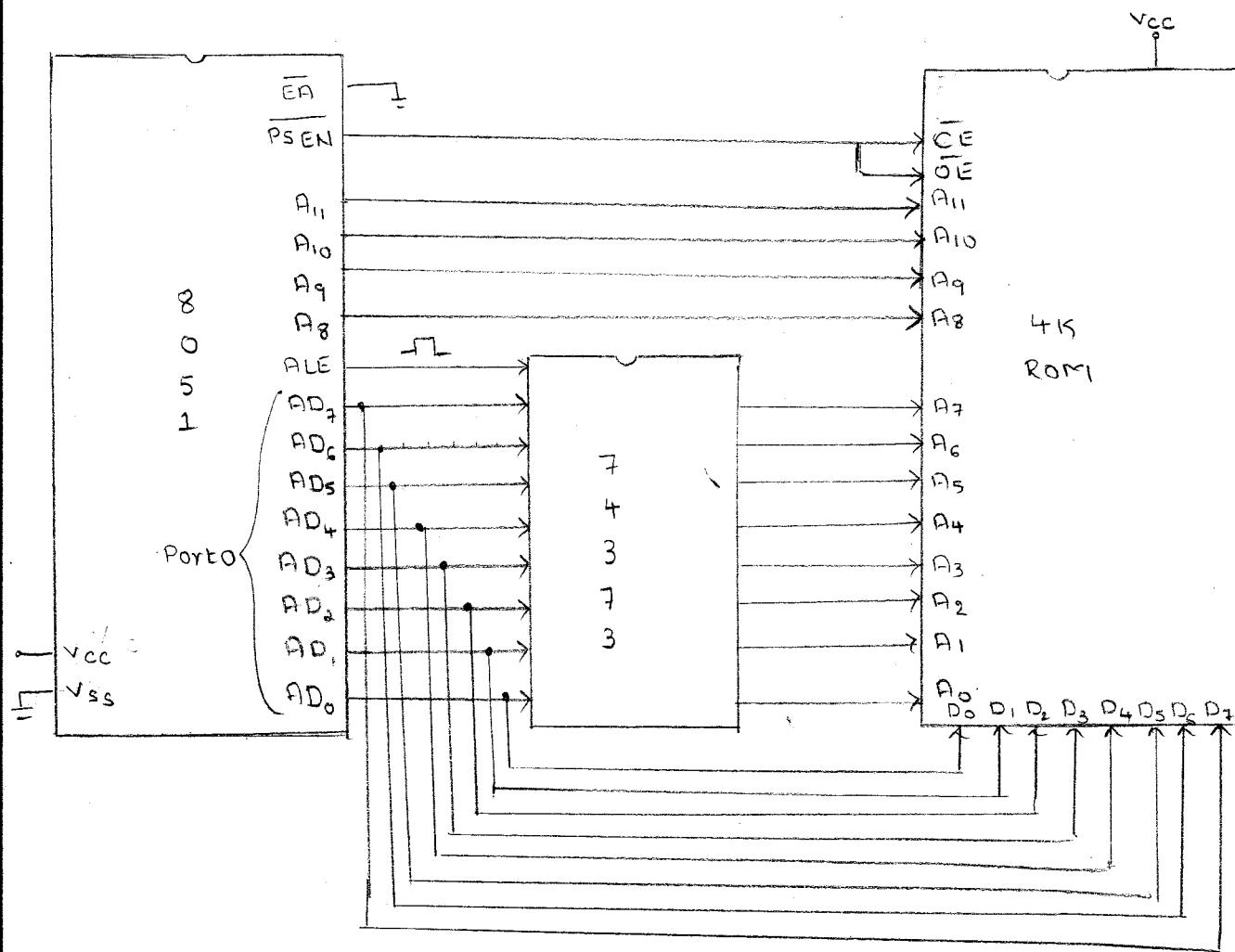
3. Interface 4K-ROM and 4K-RAM to 8081.



2. Interface 4K-RAM to 8051.



1. Interface 4K-ROM to 8051



Important Instructions :-

1) SWAP & SWAP A :-

Function : Swap nibbles within the accumulator.

Description : The Swap instruction interchanges the lower nibble ($A_0 - A_3$) with the upper nibble ($A_4 - A_7$) inside the register A.

Bytes : 1

Cycles : 1
operation : $(A_0-3) \leftrightarrow (A_4-7)$

Flags affected : None

Eg:-

MOV A, #59H

SWAP A

Before Execution	After Execution
$A = 59H$	$A = 95H$
i.e. $A = 01011001$	i.e. $A = 10010101$

2) XCHG :- XCH A, Byte

Function : Exchange Accumulator (A) with byte variable.

Description : This instruction swaps (exchange) the contents of Register A & the source byte. The source byte can be any register or RAM location.

Bytes : 1
cycles : 1
operation : $(A) \leftrightarrow (\text{Byte})$

Flags affected : None

Eg :-

i) $\boxed{\text{XCH A, R}_n}$

$\text{mov A, } \#0FFh$
 $\text{move R}_2, \#11h$
 XCH A, R_2

Before Execution	After Execution
$A = FFh$ $R_2 = 11h$	$A = 11h$ $R_2 = FFh$

ii) XCH A, direct

$\text{move A, } \#0FFh$
 $\text{move 40h, } \#11h$
 XCH A, 40h

Before Execution	After Execution
$A = FFh$ $R_2 = 11h$	$A = 11h$ $R_2 = FFh$

3) DAA :-

Function : Decimal - adjust accumulator after addition

Description : This instruction is used after addition of BCD numbers to convert the result back to BCD.

The data is adjusted in the following two possible cases:

- ▷ It adds 6 to the lower 4-bits of accumulator if it is greater than 9 & if $AC=1$
- ▷ It also adds 6 to the upper 4-bits of accumulator if it is greater than 9 & if $CY=1$.

Bytes : 1

$V \rightarrow 09$

Cycles : 1

operation : If

$$(A_{3-0}) > 9 \quad V \quad AC = 1, \text{ then } (A_{3-0}) \leftarrow (A_{3-0}) + 6$$

If

$$(A_{7-4}) > 9 \quad V \quad CY = 1, \text{ then } (A_{7-4}) \leftarrow (A_{7-4}) + 6$$

Flags affected : CY, AC.

Eg:- ▷ mov A, #47h
ADD A, #38h
DAA

$$\begin{array}{r} 47h \\ + 38h \\ \hline 7Fh \\ 6 \\ \hline 85h \end{array}$$

ii) $\begin{array}{r} 55h \\ + 68h \\ \hline 8dh \end{array}$

Here $B > 9$ & $D > 9$, So add 6 to each digit

$$\begin{array}{r} 1^1 \\ 1^1 \\ 66h \\ \hline 23h \end{array}$$

$CY=1$



(3)

4) $XCHD A, @R_p \text{ OR } XCHD A, @R_i$

Function : Exchange digit

Description : The XCHD instruction exchanges only the lower nibble of accumulator (A_{3-0}), with the lower nibble of the RAM location pointed by R_p register.

The higher-order nibble (bits 7-4) of each registers are not affected.

Bytes : 1

Cycles : 1

Operation : $(A_{3-0}) \leftrightarrow ((R_p(3-0)))$

Flags affected : None

Eq:-

mov A, #0FFh

mov R1, #50h

mov 50h, #00h

XCHD A, @R1

Before Execution	After Execution
$A = FFh$	$A = F0h$
$R_1 = 50h$	$R_1 = 50h$
$50h = 00h$	$50h = 0Fh$



5) MUL AB :-

Function : Multiply $\rightarrow (A \times B)$

Description : The 8-bit Contents of A-Register is multiplied with the 8-bit Contents of B-Register.

The 8-bit multiplication results in 16-bit result.

After multiplication, the lower byte of the result is available in A-Register & higher byte of the result is available in B-Register.

Bytes : 1

Cycles : 4

operation : $(A) \times (B) \rightarrow \begin{cases} \text{Higher byte } \uparrow \text{ in A-Register} \\ \text{Lower byte } \uparrow \text{ in B-Register} \end{cases}$

Flags affected : CY, OV.

Eg:-

mov A, #05h

mov B, #07h

MUL AB

$$\begin{array}{r} 05 \times 07h \\ \hline 00, 32h \\ \quad B \quad A \end{array}$$

Before Execution	After Execution
A = 05h B = 07h	A = 32h B = 00h



6) SUBB A, Src :-

Function : Subtract with borrow

Description : SUBB Subtracts the Source byte & the Carry Flag from the accumulator & puts the result in the accumulator.

Bytes : 1

Cycles : 1

Operation : $(A) = (A) - (\text{byte}) - (\text{cy})$

Flags affected : CY, AC, OV

Subb instruction Sets the Carry Flag according to the following

- i) destination byte > Source byte \rightarrow CY=0, result is +ve
- ii) destination byte = Source byte \rightarrow CY=0, result is 0
- iii) destination byte < Source byte \rightarrow CY=1, result is -ve & in 2's complement.

Eg:-

SUBB A, #data \rightarrow SUBB A, #25h \rightarrow $(A) = (A) - (25h) - (\text{cy})$

SUBB A, R_n \rightarrow SUBB A, R₀ \rightarrow $(A) = (A) - (R_0) - (\text{cy})$

SUBB A, direct \rightarrow SUBB A, 50h \rightarrow $(A) = (A) - (50h) - (\text{cy})$

SUBB A, @R_i \rightarrow SUBB A, @R₀ \rightarrow $(A) = (A) - ((R_0)) - (\text{cy})$

⇒ MOV C, b :- i) MOV C, bit ii) MOV dest-bit, Source-bit

Function : Move bit data from Source bit to destination bit.
one of the operands must be the carry flag ; the
other may be any directly addressable bit.

Description: The Single bit is moved to the Carry Flag.

Bytes : 2

Cycles : 1

operations : $(C) \leftarrow (\text{bit})$

Flags affected : CY

Eg :-

i) MOV C, P1.4

Before Execution	After Execution
$C = 'X'$	$C = 1$
$P1.4 = 1$	$P1.4 = 1$

ii) MOV C, P1.4

Before Execution	After Execution
$C = 'X'$	$C = 0$
$P1.4 = 0$	$P1.4 = 0$



8) DIV AB :-

Function : Divide $\rightarrow (A)/(B)$

Description : Div AB divides the contents of accumulator by the contents of B-Register.

The accumulator receives the quotient & B-Register receives the remainder.

Bytes : 1

Cycles : 4

operation : $(A)/(B) \rightarrow \begin{cases} A = \text{quotient} \\ B = \text{remainder} \end{cases}$

Flags affected : CY, OV

Eg:-
mov A, #0FBh
mov B, #12h
DIV AB

FBh \rightarrow 251d
12h \rightarrow 18d

$$\begin{array}{r} 13 \rightarrow (A) \\ \hline 18) 251 \\ 14 \\ \hline 11 \\ 11) 17 \\ 17 \\ \hline 0 \\ 17 \rightarrow (B) \end{array}$$

Before Execution	After Execution
A = FBh (251d)	A = 0dh (13d)
B = 12h (18d)	B = 11h (17d)

After executing the instruction, the Quotient is stored in accumulator & the remainder is stored in B-Register.



⑧

q) MOVX :- MOVX dest-byte, Source-byte

Function : MOVE external

Description : This instruction transfers data between external memory & register A, hence the 'X' is appended to MOV.

The address of external memory location being accessed can be 16-bit or 8-bit.

Bytes : 1

Cycles : 2

operation : $(A) \leftarrow (R_i)$

Flags affected : None

Eg :- MOVX A, @R_p MOVX A, @R_i

Where,
 $R_i = R_p = R_0 \text{ or } R_1$

i) MOVX A, @R₀

mov R₀, #50h

mov 50h, #FFh

movx A, @R₀

Before Execution	After Execution
$A = 'xx'$ $R_0 = 50h$ $50h = FFh$	$A = FFh$ $R_0 = 50h$ $50h = FFh$

ii) MOVX A, @DPTR

mov DPTR, #1234h

mov 1234h, #0FFh

movx A, @DPTR

Before Execution	After Execution
$A = 'xx'$ $1234 = FFh$ $DPTR = 1234h$	$A = FFh$ $1234h = FFh$ $DPTR = 1234h$



⑨

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

10) MOVC

Function : Move Code

Description : This instruction loads the accumulator with a code byte or constant from program memory.

Bytes : 1

Cycles : 2

Flags affected : None

Eg:- \Rightarrow MOVC A, @ A + DPTR
 \Rightarrow MOVC A, @ A + PC

\Rightarrow MOVC A, @ A + DPTR

Before Execution	After Execution
A = 00h	A = FFh
DPTR = 1234h	DPTR = 1234h
1234h = FFh	1234h = FFh

↑ Code memory address ↑ Data

11) SETB :-

\Rightarrow SETB C

Function : Sets the Carry Flag bit

Bytes : 1

Cycles : 1

Operation : $(C) \leftarrow 1$

Flags affected : CY

Before Execution	After Execution
$\Rightarrow CY = 0$	$\Rightarrow CY = 1$
$\Rightarrow CY = 1$	$\Rightarrow CY = 1$



10

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

8051 Programming in C

Compilers produce HEX files that we download into the ROM of the microcontroller. The size of the HEX file produced by the compilers is one of the main concerns of microcontroller programmers, for two reasons:

- Microcontrollers have limited on-chip ROM.
- The code space for the 8051 is limited to 64Kbytes.

Advantages of programming in ‘C’ for Microcontrollers:

1. It is easier & less time consuming to write programs in ‘C’ than assembly. (Assembly language is TEDIOUS & time consuming)
2. C is easier to modify & update.
3. C programs are portable to other microcontrollers with little or NO modification.
4. Programming in ‘C’ allows using code available in function libraries.

Disadvantages of programming in ‘C’:

1. Assembly language programs have fixed size for the HEX files produced, whereas for the same ‘C’ programs, different ‘C’ compiler produces different HEX code sizes.
2. The 8051 general purpose registers, such as **R₀-R₇, A & B** are under the control of the ‘C’ compiler & are not accessed by ‘C’ statements.
3. It is difficult to calculate exact delay for ‘C’ programs
4. Microcontrollers have limited on-chip ROM & the code space (to store program codes) is also limited. A misuse of data types by the programmer in writing ‘C’ programs, for ex using ‘int’(16-bit) data type instead of ‘unsigned char’ (8-bit) can lead to a large size HEX files.

The same problem does not arise in assembly language programs.



DATA TYPES IN C:

1. Unsigned char:

- ❖ Since 8051 is an 8-bit microcontroller, the character data type is also 8-bit. So char data type is most widely used in 8051 C.
- ❖ The unsigned char is an 8-bit data type that takes a value in the range of 0-255(00-FFh).
- ❖ Used for setting a counter value, to represent ASCII character etc.

Note:

- ❖ The C compilers use the **signed char** as the default if we do not put the keyword **unsigned** in front of the **char**.

2. Signed char:

- ❖ The Signed char is an 8-bit data type that uses the MSB bit (Most Significant Bit) '**D₇**' to represent the **positive** or **Negative** value.
- ❖ So only 7-bits are used to represent the magnitude of the number.
- ❖ Signed char ranges from -128 to +127.
- ❖ Used to represent a quantity having negative values such as Temperature etc.

3. Unsigned int:

- ❖ The unsigned int is a **16-bit** data type that ranges from **0 to 65635 (0000-FFFFh)**.
- ❖ Unsigned int is used to define a 16-bit variables such as **memory addresses**. It is also used to set counter values of more than **256**.
- ❖ Since 8051 is a 8-bit microcontroller & the int data type takes **two bytes of RAM**. (So we must not use int data type unless we have to)
- ❖ The misuse of int variables will result in a **larger HEX file**.

Note:

- ❖ The C compiler uses **signed int** as a default if we do not use the keyword **unsigned**.



4. Signed int:

- ❖ Signed int is a 16-bit data type that uses the **MSB bit D₁₅** to represent **positive or negative value**.
- ❖ So only 15-bits are used to represent the **magnitude** of the number.
- ❖ Signed int ranges from **-32,768** to **+32,767**.
- ❖ Used to represent **positive or negative** value.

5. sbit (Single bit):

- ❖ sbit keyword is used to access **single-bit addressable registers**.
- ❖ It allows access to the **single bit** of the **SFR registers**.
- ❖ Its size is **1-bit** i.e. either **0** or **1**.

6. Bit:

- ❖ The bit data type allows to access to **single bits** of **bit-addressable memory spaces 20h-2Fh**.
- ❖ Its size is **1-bit** i.e. either **0** or **1**.

Note:

- ❖ The sbit data type is used for **bit addressable SFR's**.

7. sfr:

- ❖ sfr data type is used to access the **byte-size SFR registers**.
- ❖ Its size is **8-bits (1-Byte)**.
- ❖ RAM address **80h-ffh** is used.



Data Types in C:

Sl No	Data Types	Bits	Bytes	Data Range
1	Unsigned char	8	1	0 to 255
2	Signed char	8	1	-128 to +127
3	Unsigned int	16	2	0 to 65535
4	Signed int	18	2	-32768 to +32767
5	Bit	1	-	0 to 1
6	sbit	1	-	0 to 1
7	sfr	8	1	0 to 255

TIME DELAYS IN 8051 ‘C’:

There are two ways to create a time delay in 8051 ‘C’:

1. Using a simple for loop.
2. Using the 8051 Timers.

We cannot get the exact delays using simple for loops because of following reasons.

1. The **instruction execution speed varies** according to the number of clock periods per machine cycle i.e. different versions of microcontroller uses different machine cycles to execute instructions.

{ **eg:** The 8051/8052 uses 12-clock periods per machine cycle & newer generation microcontrollers uses fewer clocks per machine cycle.

For example:

- ❖ DS500 uses 4-clock periods per machine cycle.
- ❖ DS89C420 uses only one clock periods per machine cycle.

}



2. The crystal frequency connected to the X_1 & X_2 input pins.
3. In C programs, the C compiler converts the 'C' statements & functions to **assembly language instructions**.

If we compile a given 8051 C-programs with different compilers, each compiler produces different **HEX codes**.

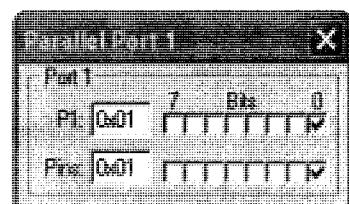
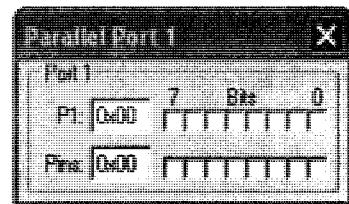
1. Write an 8051 C program to send values 00-FF to port 1 only once.

Algorithm:

1. Initialize a counter (FFh)
2. Using a for loop, send data to port P1

Program:

```
#include<reg51.h>
void main()
{unsigned char i;
for(i=0;i<=0xFF;i++)          // using HEXA decimal value
P1=i;
}
```



OR



```
#include<reg51.h>
void main()
{unsigned char i;
for(i=0;i<=225;i++)      // using decimal value
P1=i;
}
```

Note:

1. P_1 is declared in the reg51.h file.
2. In for statement, we have $i \leq 0xFF$; implying that $i = FF_{16}$. If it was written as $i \leq 255$; then $i = 255$ in decimal.
3. P_1 must be **UPPER case** Letter. If p_1 is **LOWER case** then it will not support in 'C'.

2. Write an 8051 C program to continuously display numbers from 00-FF to port 1.

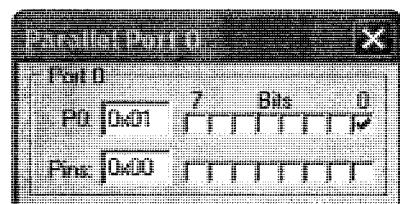
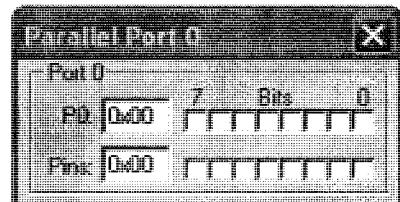
```
#include<reg51.h>
void main()
{
while(1)      //repeat forever
{unsigned char i;
for(i=0;i<=0xFF;i++)      //      for(i=0;i<=255;i++)
P1=i;
}}
```



Using sfr data type:

3. Write a C program to display number from 00 to FFh only once using **sfr** data type.

```
sfr P0=0x80;  
void main()  
{  
    unsigned char i;  
    for(i=0;i<=0xFF;i++)  
        // for(i=0;i<=255;i++)  
        P0=i;  
}
```



Note:

1. The program doesn't use the `#include<reg51.h>`, but the port address **P0** is declared using **sfr** keyword.
2. **sfr P0=0x80;** implies **P0** is a variable of **Byte size**, which has a definite **RAM address of 80H** in the **RAM area**.



4. Write a 8051 c program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C & D to port 1.

Note:

The ASCII characters can be concatenated to form a string & then we can access all the characters serially one after the other using an array indexing.

Eg;

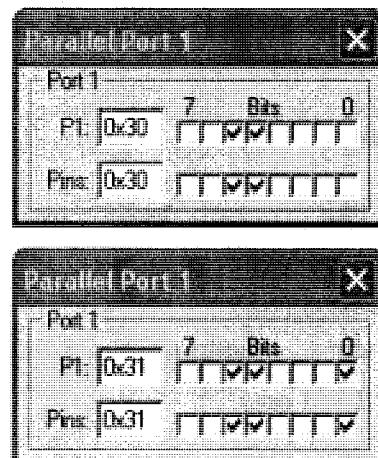
- ❖ To access '3', we can use x[3].
- ❖ Similarly to access 'A', we can use x[6].
- ❖ Before that we have to define as

```
unsigned char x[ ]="012345ABCD";
```

(Program to displays only once)

```
#include<reg51.h>

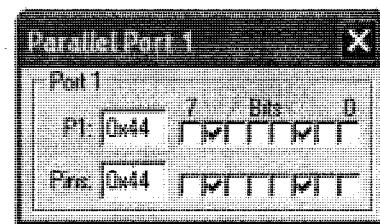
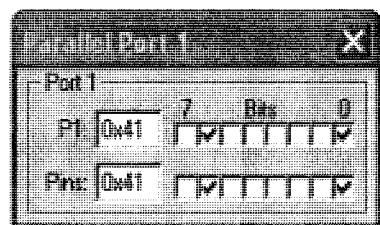
void main()
{
    unsigned char x[ ]="012345ABCD";
    unsigned char i;
    for(i=0;i<10;i++)
        P1=x[i];
}
```



(Program displays Continuously)

```
#include<reg51.h>

void main()
{
    while(1)
    {
        unsigned char x[]={012345ABCD"};
        unsigned char i;
        for(i=0;i<10;i++)
            P1=x[i];
    }
}
```

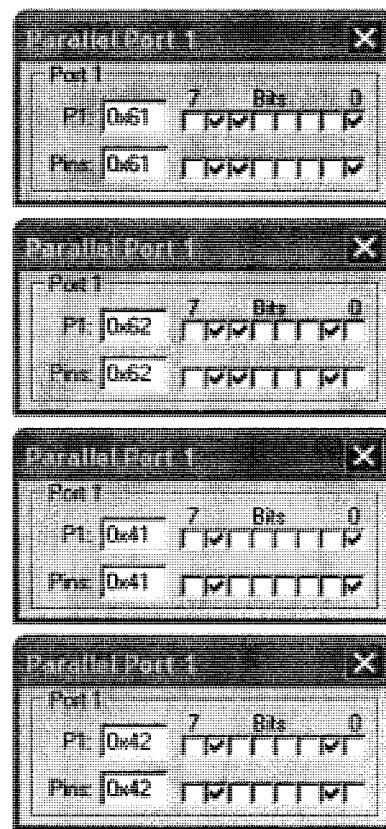


5. Write a 8051 c program to continuously send hex values for ASCII characters of **a**, **b**, **A**, **B** to port 1.

```
#include<reg51.h>

void main()
{
    while(1)          // repeat forever

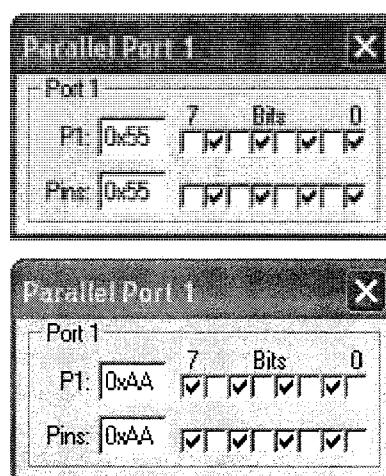
    {
        unsigned char x[]={ "abcABC"};
        unsigned char i;
        for(i=0;i<6;i++)
            P1=x[i];
    }
}
```



6. Write an 8051 C program to toggle all the bits of **P₁** continuously.

```
#include<reg51.h>

void main()
{
    for(;;)          // repeat forever
    {
        P1=0x55;
        P1=0xAA;
    }
}
```



7. Write an 8051 C program to send values of -4 to +4 to port P₁.

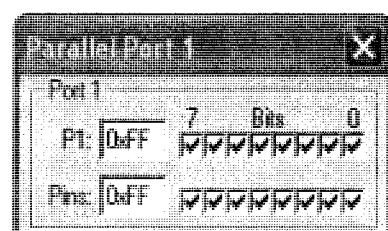
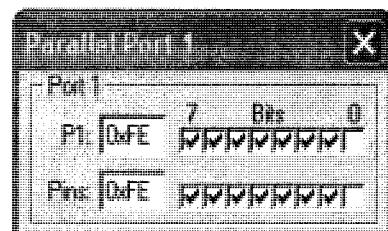
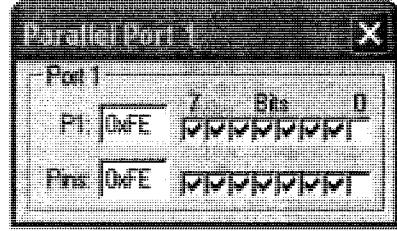
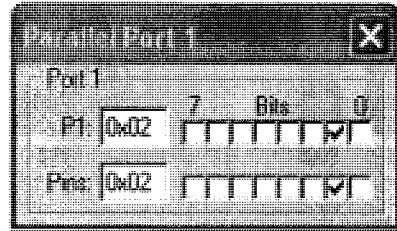
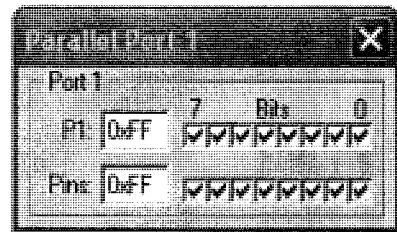
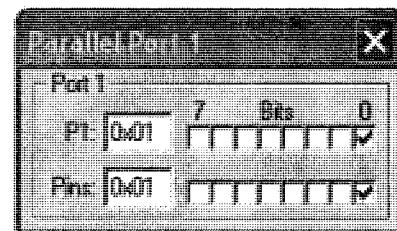
```
#include<reg51.h>
void main()
{
    char num[ ]={1,-1,2,-2,3,-3,4,-4};
    unsigned char i;
    for(i=0;i<8;i++)
        P1=num[i];
}
```

Note:

char num[] is a **signed char**.

8. Write an 8051 C program to toggle bit D₀ of the port P₁ (P_{1.0}) 50,000 times **without any time delay**

```
#include<reg51.h>
sbit port1=P1^0;
void main()
{
    unsigned int i;
    for(i=0;i<50000;i++)
    {
        port1=0;
        port1=1;
    }
}
```



Note:

- ❖ sbit is declared **outside of main**.
- ❖ P_{1.0} is defined as port1 i.e. **single bit**.

With Delay:

```
#include<reg51.h>

sbit pin1= P1^0;

void main()
{
    unsigned int i,j;
    for(i=0;i<50000;i++)
    {
        for(j=0;j<4000;j++)
            pin1=0;
        for(j=0;j<4000;j++)
            pin1=1;
    }
}
```



OR

```
#include<reg51.h>

sbit pin1= P1^0;

void main()

{

    unsigned int i,j;

    for(i=0;i<50000;i++)

    {

        pin1=0;

        for(j=0;j<4000;j++);

        pin1=1;

        for(j=0;j<4000;j++);

    }

}
```

9. Write an 8051 C program to toggle P1 continuously forever with some delay

```
#include<reg51.h>

void main()

{

    unsigned int i;

    for(;;)

    {

        for(i=0;i<4000;i++)

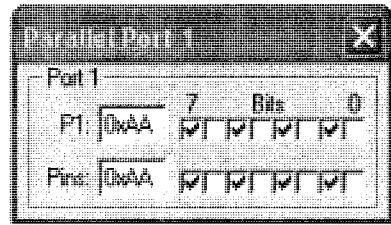
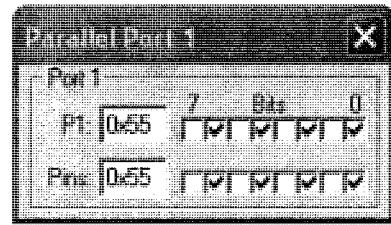
            P1=0x55;

        for(i=0;i<4000;i++)

            P1=0xAA;

    }

}
```



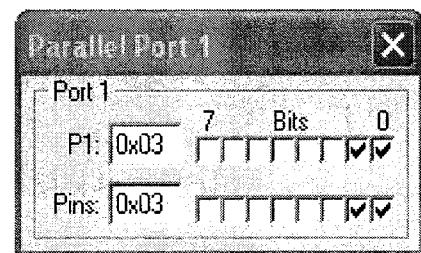
OR

```
#include<reg51.h>

void main()
{
    unsigned int i;
    for(;;)
    {
        P1=0x55;
        for(i=0;i<4000;i++);
        P1=0xAA;
        for(i=0;i<4000;i++);
    }
}
```

10. Write an 8051 C program to send the sum of values -12 and 15 to port P1.

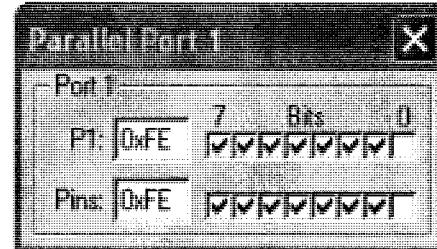
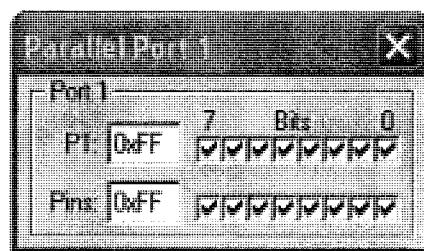
```
#include<reg51.h>
void main()
{
    signed char i,j;
    i=-12;
    j=15;
    P1=i+j;
}
```



11. Write an 8051 C program to toggle bit 0 of port P₁ (P_{1.0}) 20,000 times.

```
#include<reg51.h>
sbit port1bit = P1^0;

void main()
{
    unsigned int i;
    for(i=0;i<20000;i++)
    {
        port1bit = 0;
        port1bit = 1;
    }
}
```



Note:

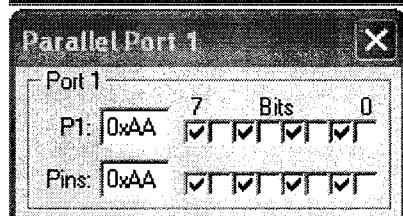
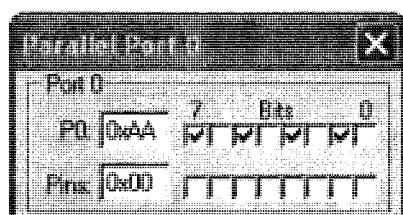
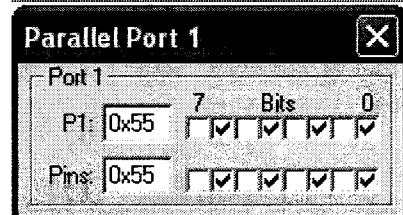
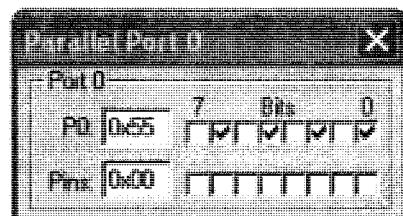
- ❖ sbit is declared outside of main.



12. Write an 8051 C program to Toggle all the bits of P0 & P1 continuously with a 1 ms delay

```
#include<reg51.h>

void main()
{
    unsigned int i;
    while(1)          //      repeat forever
    {
        P0=0x55;
        P1=0x55;
        for(i=0;i<1275;i++);
        P0=0xAA;
        P1=0xAA;
        for(i=0;i<1275;i++);
    }
}
```



Note:

The delay program is tested for the **DS89C420** Microcontroller using **11.0592 MHz** crystal oscillator.



13. Write an 8051 C program to toggle the bits of P1 Ports continuously with a 250ms delay.

```
#include<reg51.h>

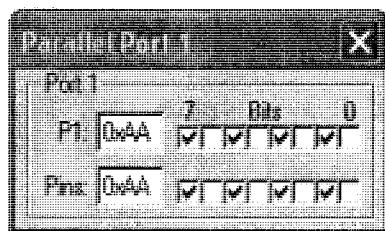
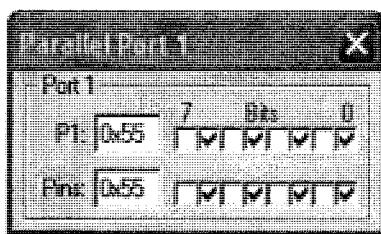
void delay(unsigned int);

void main()
{
    while(1)          //      repeat forever
    {
        P1=0x55;
        delay (250);
        P1=0xAA;
        delay(250);
    }
}

void delay(unsigned int count)
{
    unsigned int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<1275;j++);
}
```

Note:

The delay program is tested for the **DS89C420** Microcontroller using **11.0592 MHz** crystal oscillator.



14. Write an 8051 C program to toggle all the bits of P₀ and P₂ continuously with a 250ms delay.

Note:

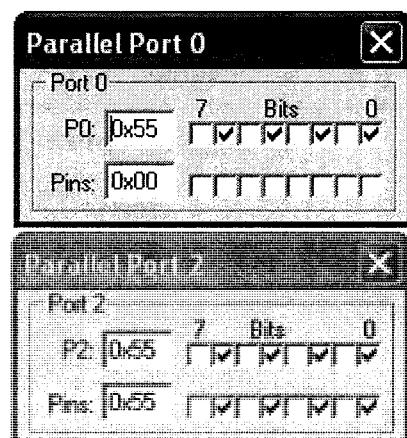
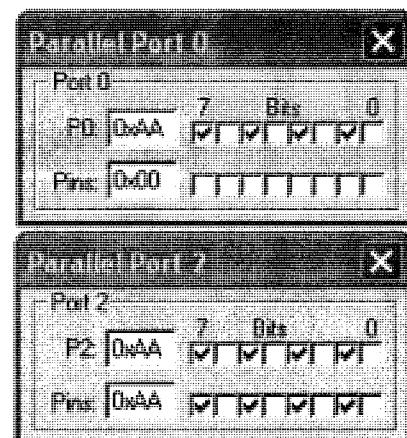
The delay program is tested for the DS89C420 Microcontroller using 11.0592 MHz crystal oscillator.

```
#include<reg51.h>

void delay(unsigned int);

void main()
{
    while(1)      // repeat forever
    {
        P0=0x55;
        P2=0x55;
        delay (250);
        P0=0xAA;
        P2=0xAA;
        delay(250);
    }
}

void delay(unsigned int count)
{
    unsigned int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<1275;j++);
}
```

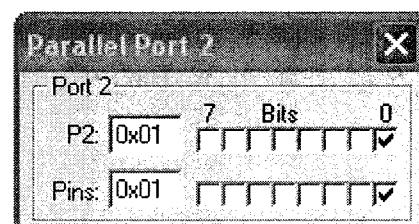
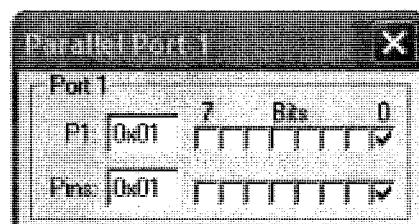
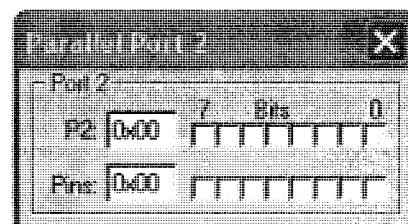
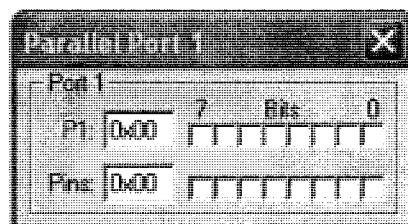


BYTE SIZE I/O:

15. LED's are connected to bits P₁ & P₂. Write an 8051 C program that shows the count from 0 to FFh on the LED's. (0000 0000 to 1111 1111)

```
#include<reg51.h>
#define LED P2
void main()
{
    P1=0;          //clear port 1
    LED=0;          //clear port 2

    for(;;)
    {
        P1++;
        LED++;
    }
}
```

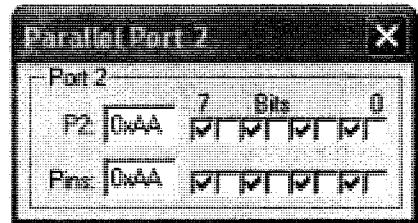
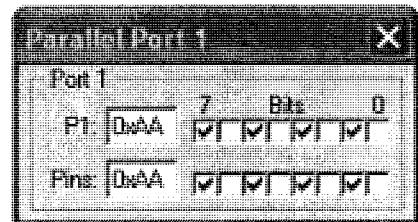


16. Write an 8051 C program to get a byte of data from P₁, wait for ½ second, and then send it to P₂.

```
#include<reg51.h>
void delay(unsigned int);
void main()
{
    unsigned char mydata;
    P1=0xFF;           // port1 acts as I/P port
    P1=0xAA;           // AA is an I/P data to port 1
    while(1)
    {
        mydata=P1;
        delay(500);
        P2=mydata;
    }
}

void delay(unsigned int count)

{
    unsigned int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<1275;j++);
}
```



17. Write an 8051 C program to get data from P₀. If it is less than 100, send it to P₁; otherwise, send it to P₂.

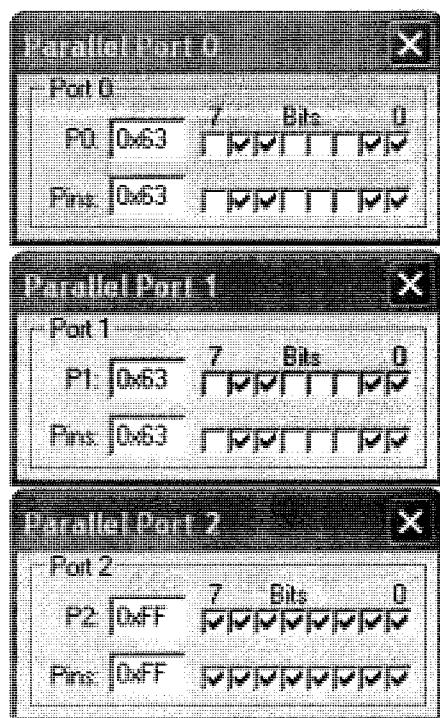
```
#include<reg51.h>
void main()
{
    unsigned char mydata;
    P0=0xFF; // port 0 as I/P port.

    P0=99; // Less than 100 so send data
              // to P1

    while(1)
    {
        mydata=P0;
        if(mydata<100)
            P1=mydata;
        else
            P2=mydata;
    }
}
```

Note:

- 99d=63h



OR

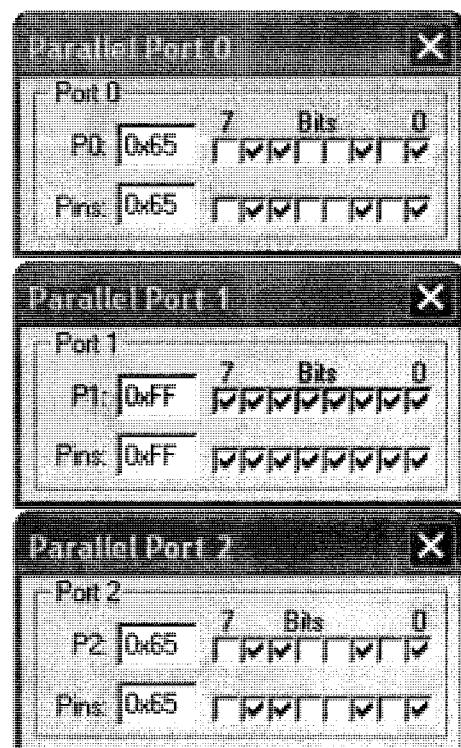
```
#include<reg51.h>
void main()
{
    unsigned char mydata;
    P0=0xFF; // port 0 as I/P port.

    P0=101; // More than 100 so send data
              // to P2

    while(1)
    {
        mydata=P0;
        if(mydata<100)
            P1=mydata;
        else
            P2=mydata;
    }
}
```

Note:

- 101d=65h

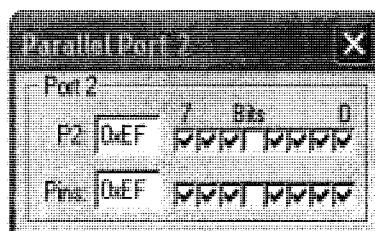
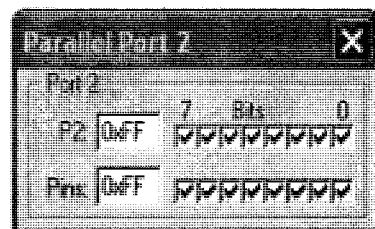


BIT ADDRESSABLE I/O PROGRAMMING.

- ❖ **sbit** data type is used to access a single bit of P₀-P₃.
18. Write an 8051 C program to toggle only one bit P_{2.4} continuously without disturbing the rest of the bits of P₂.

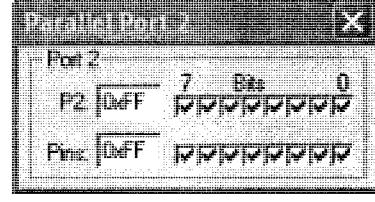
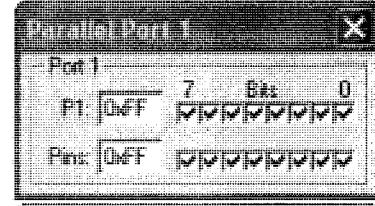
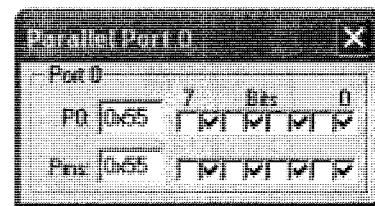
```
#include<reg51.h>
sbit onebit=P2^4;

void main()
{
while(1)
{
    onebit=1;
    onebit=0;
}
}
```



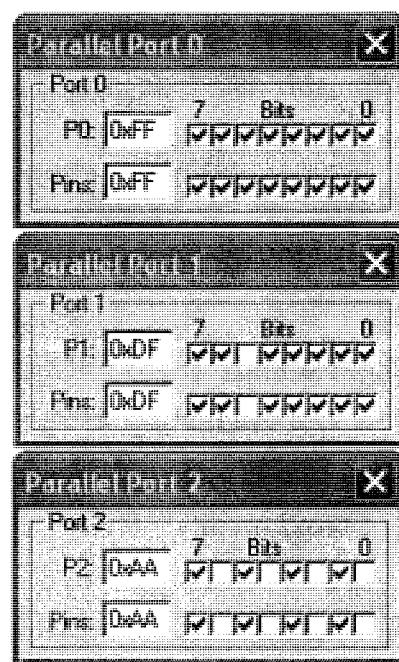
19. Write an 8051 C program to monitor bit P_{1.5}. If it is **high**, send **55h** to P₀; otherwise send **AAh** to P₂.

```
#include<reg51.h>
sbit mybit=P1^5;
void main()
{
mybit=1;
while(1)
{
    if(mybit==1)
        P0=0x55;
    else
        P2=0xAA;
}
}
```



II)

```
#include<reg51.h>
sbit mybit=P1^5;
void main()
{
    mybit=0;
    while(1)
    {
        if(mybit==1)
            P0=0x55;
        else
            P2=0xAA;
    }
}
```



BIT ADDRESSES OF PORTS:

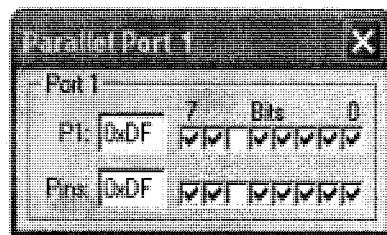
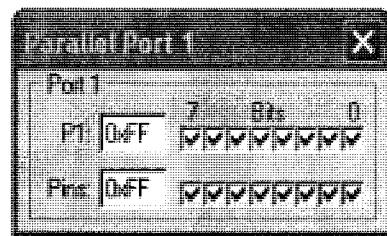
Table Single Bit Addresses of Ports

P0	Addr	P1	Addr	P2	Addr	P3	Addr	Port's Bit
P0.0	80H	P1.0	90H	P2.0	A0H	P3.0	B0H	D0
P0.1	81H	P1.1	91H	P2.1	A1H	P3.1	B1H	D1
P0.2	82H	P1.2	92H	P2.2	A2H	P3.2	B2H	D2
P0.3	83H	P1.3	93H	P2.3	A3H	P3.3	B3H	D3
P0.4	84H	P1.4	94H	P2.4	A4H	P3.4	B4H	D4
P0.5	85H	P1.5	95H	P2.5	A5H	P3.5	B5H	D5
P0.6	86H	P1.6	96H	P2.6	A6H	P3.6	B6H	D6
P0.7	87H	P1.7	97H	P2.7	A7H	P3.7	B7H	D7

20. Write an 8051 C program to turn bit P_{1.5} ON and OFF 50,000 times.

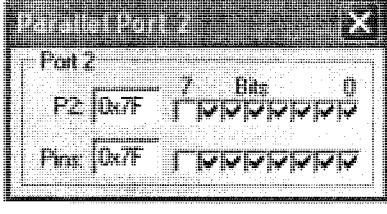
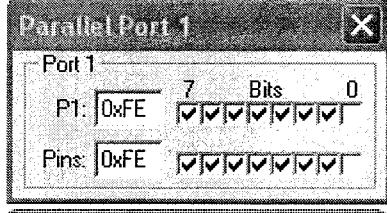
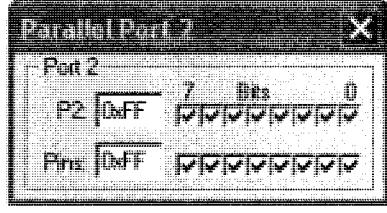
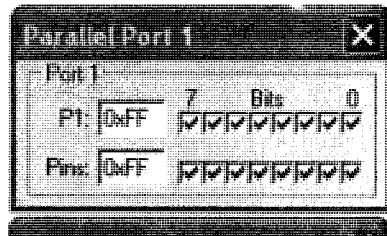
```
#include<reg51.h>
sbit portbit=0x95; // another way to declare bit P1.5

void main()
{
    unsigned int i;
    for(i=0;i<50000;i++)
    {
        portbit =1;
        portbit =0;
    }
}
```



21. Write an 8051 C to get the status of bit P_{1.0}, save it, and send it to P_{2.7} continuously.

```
#include<reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit membit;
void main()
{
    inbit=0;      // only 1-bit
    while(1)
    {
        membit=inbit;
        outbit=membit;
    }
}
```



Error:

```
#include<reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit membit;
void main()
{
    inbit=0x00; // here 00h is a byte data so this
                 // program is not working.
    while(1)
    {
        membit=inbit;
        outbit=membit;
    }
}
```



22. Write an 8051 C program to toggle all the bits of P₀, P₁, and P₂ continuously with a 250ms delay. Use the **sfr keyword** to declare the port addresses.

```
sfr P0=0x80;  
sfr P1=0x90;  
sfr P2=0xA0;
```

```
void delay(unsigned int);
```

```
void main()  
{
```

```
    while (1)
```

```
    {
```

```
        P0=0x55;  
        P1=0x55;  
        P2=0x55;
```

```
        delay(250);
```

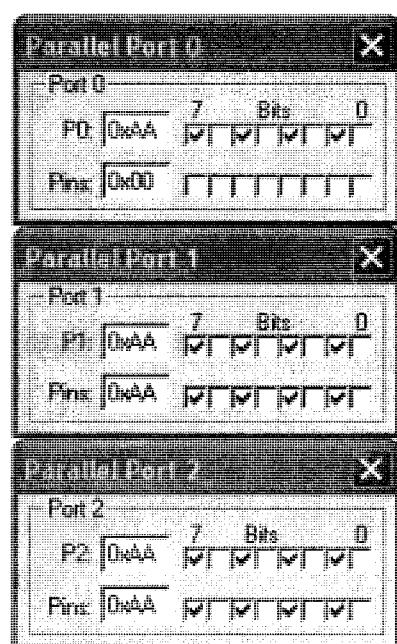
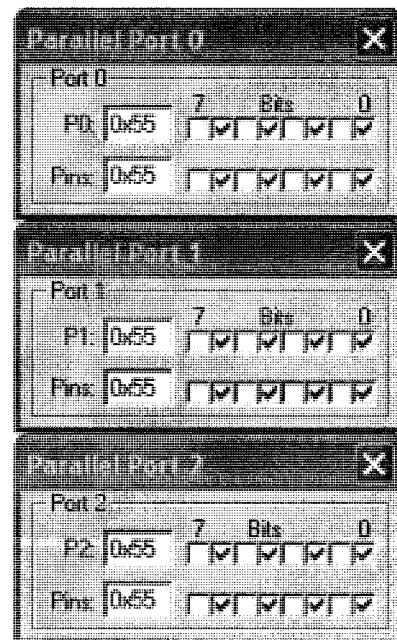
```
        P0=0xAA;  
        P1=0xAA;  
        P2=0xAA;
```

```
        delay(250);
```

```
}
```

```
}
```

```
void delay(unsigned int count)  
{  
    unsigned int i,j;  
    for(i=0;i<count;i++)  
        for(j=0;j<count;j++);  
}
```

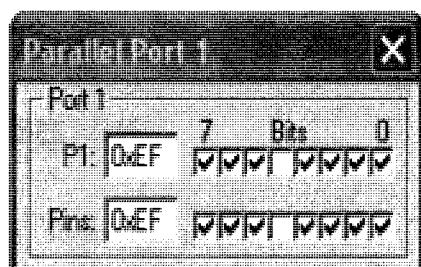
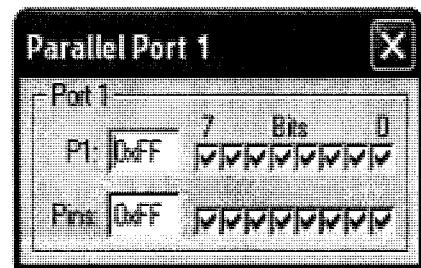


23. Write a program to generate a square wave of 250ms ON-time, 50% duty cycle on pin P1.4

- Making a pin **LOW & HIGH** continuously generates a square wave. **50% duty cycle** means **ON time = OFF time = 250ms**.

```
#include<reg51.h>
sbit port1pin = P1^4;
void delay(unsigned int);
void main()
{
    while (1)
    {
        port1pin=0;
        delay(250);
        port1pin=1;
        delay(250);
    }
}

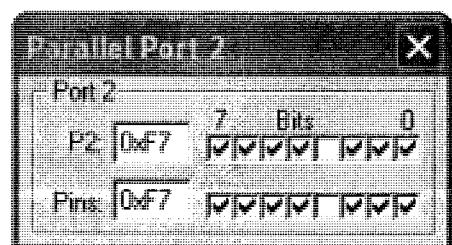
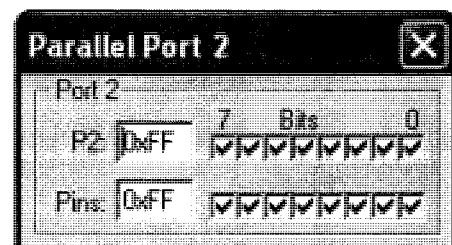
void delay(unsigned int count)
{
    unsigned int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<count;j++);
}
```



23. Write a program to generate a square wave of 75% duty cycle with ON time of 300ms on pin P2.3

```
#include<reg51.h>
sbit port2pin = P2^3;
void delay(unsigned int);
void main()
{
    while (1)
    {
        port2pin=0;
        delay(100);
        port2pin=1;
        delay(300);
    }
}

void delay(unsigned int count)
{
    unsigned int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<count;j++);
}
```



Error: Follow Previous Program Method

```
#include<reg51.h>

sbit P1^5=0x95;           //      Error

void delay(unsigned int);
void main()
{
    while (1)
    {
        P1^5=0;           //      Error
        delay(100);
        P1^5=1;           //      Error
        delay(300);
    }
}

void delay(unsigned int count)
{
    unsigned int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<count;j++);
}
```



24. A Door sensor is connected to the P_{1.1} pin, and a buzzer is connected to P_{1.7}. Write an 8051 C program to monitor the door sensor, and when it opens, sound the buzzer. You can sound the buzzer by sending a square wave of a few hundred Hz.

```
#include<reg51.h>
void delay(unsigned int);
sbit Dsensor = P1^1;
sbit Buzzer = P1^7;

void main()
{
    Dsensor = 1;          //      P1.1 as an Input
    while (Dsensor==1)
    {
        Buzzer=0;
        delay(200);     //      Square wave Low to High
        Buzzer=1;
        delay(200);
    }
}

void delay(unsigned int count)
{
    unsigned int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<count;j++);
}
```



25. The data pins of an LCD are connected to P1. The information is latched into the LCD whenever it's Enable pin goes from high to low. Write an 8051 C program to send "My Students" to this LCD.

```
#include<reg51.h>
#define LCDData P1;
sbit En=P2^0;
void main()
{
    unsigned char message[ ]= "My Students";
    unsigned char i;
    for(i=0;i<11;i++)
    {
        LCDData = message[i];
        En=1;
        En=0;
    }
}
```



26. Write an 8051 C program to get the status of bit P_{1.0}, save it at bit D₁ of 24h RAM location and send it to P_{2.7} continuously.

(Not Sure about this program.)

```
#include<reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit membit = 0X21;
void main()
{
    inbit =1;
    while(1)
    {
        membit = inbit;
        outbit = membit;
    }
}
```



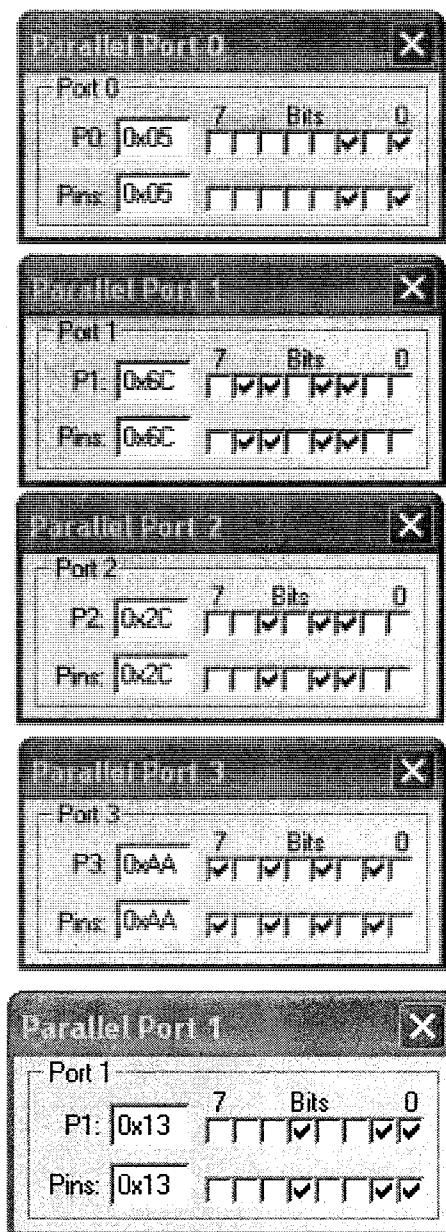
BITWISE OPERATOR IN C:

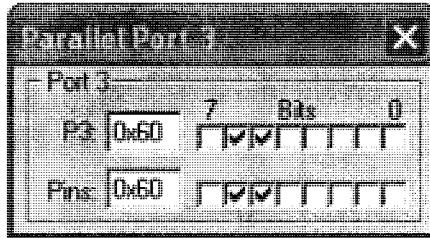
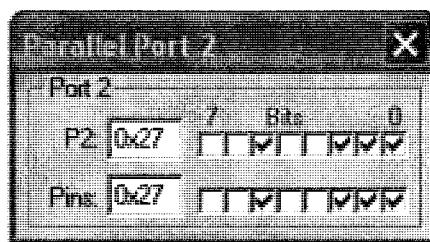
Table Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	B	A&B	A B	A^B	Y=~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

27. Write a 8051 C program to perform different bitwise operations.

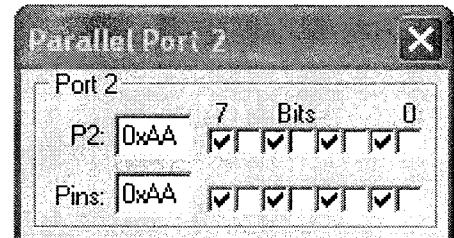
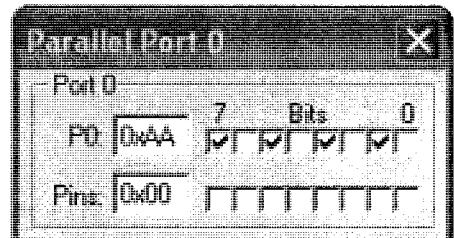
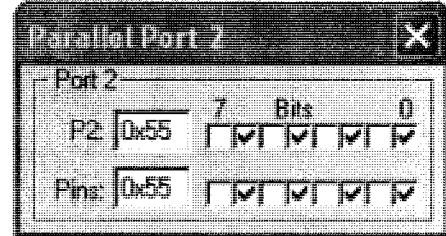
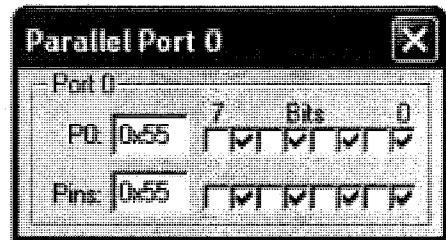
```
#include<reg51.h>
void main()
{
    P0=0x55 & 0x0F;
    P1=0x04 | 0x68;
    P2=0x54 ^ 0x78;
    P3=~ 0x55;
    P1=0x9A >> 3;
    P2=0x77 >> 4;
    P3=0x6 << 4;
}
```





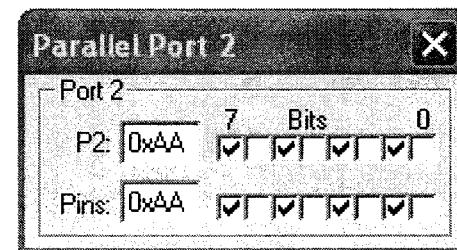
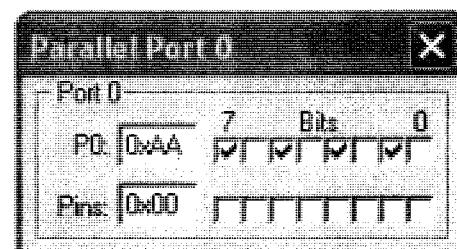
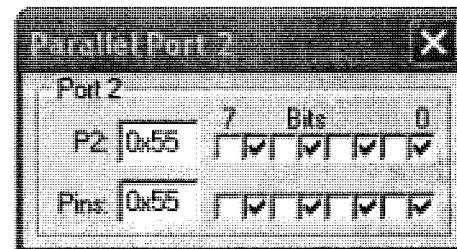
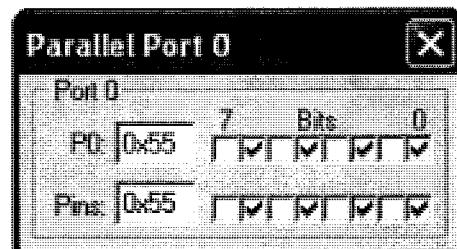
28. Write an 8051 C program to toggle all the bits of P₀ & P₂ Continuously with 250ms delay. Use the inverting operator.

```
#include<reg51.h>
void delay(unsigned int);
void main()
{
    P0=0x55;
    P2=0x55;
    while(1)
    {
        P0=~P0;
        P2=~P2;
        delay(250);
    }
}
void delay(unsigned int count)
{
    unsigned int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<1275;j++);
}
```



29. Write an 8051 C program to toggle all the bits of P₀ & P₂ Continuously with 250ms delay. Use the Ex-OR operator.

```
#include<reg51.h>
void delay(unsigned int);
void main()
{
    P0=0x55;
    P2=0x55;
    while(1)
    {
        P0=P0 ^ 0xFF;
        P2=P2 ^ 0xFF;
        delay(250);
    }
}
void delay(unsigned int count)
{
    unsigned int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<1275;j++);
}
```



30. Write an 8051 C program to get bit $P_1.0$ and send it to $P_2.7$ after Inverting.

```
#include<reg51.h>
sbit inbit = P1^0;
sbit outbit=P2^7;
bit membit;
void main()
{
    while(1)
    {
        //inbit = 1;          // 20h =01h
        //inbit = 0;          // 20h =00h
        membit = inbit;
        outbit = ~ membit;
    }
}
```

Address:		i:20h
I :0x20:	01	00 00
I :0x33:	00	00 00
I :0x46:	00	00 00

Address:		i:20h
I :0x20:	00	00 00
I :0x33:	00	00 00
I :0x46:	00	00 00

31. Write an 8051 C program to read the $P_1.0$ and $P_1.1$ bits and issue an ASCII character to P_0 according to the following table:

$P_1.1$	$P_1.0$	Operations
0	0	Send '0' to P_0
0	1	Send '1' to P_0
1	0	Send '2' to P_0
1	1	Send '3' to P_0



```
#include<reg51.h>
void main()
{
    unsigned char i;
    i=P1;
    i=i&0x3;           // mask unused bits

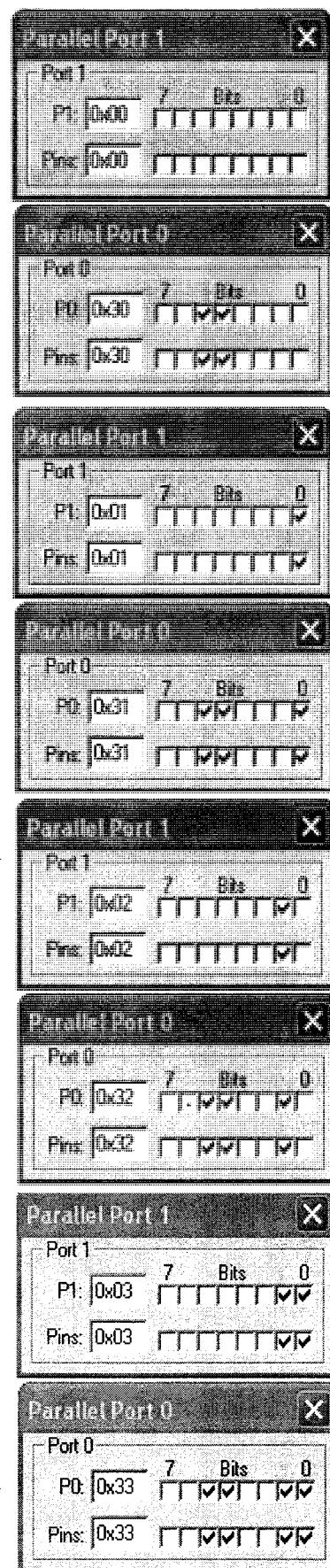
    switch(i)
    {
        case(0):
            P0='0'; // ASCII 30
            break;

        case(1):
            {
                P0='1'; // ASCII 31
                break;
            }

        case(2):
            {
                P0='2'; // ASCII 32
                break;
            }

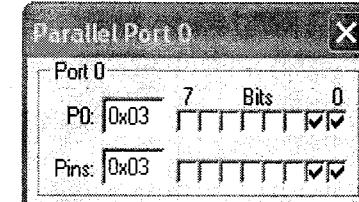
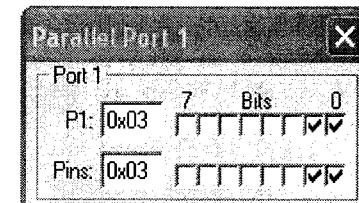
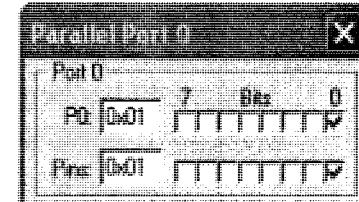
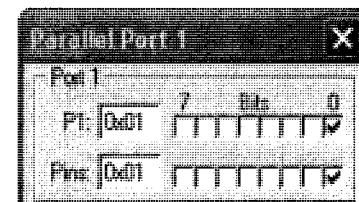
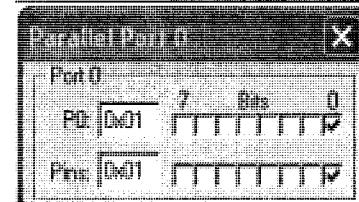
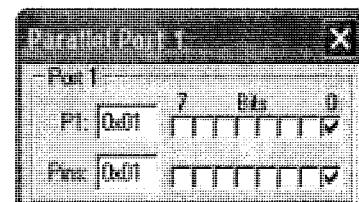
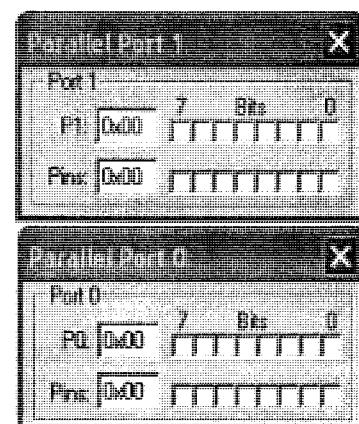
        case(3):
            {
                P0='3'; // ASCII 33
                break;
            }
    }
}
```

NOTE:



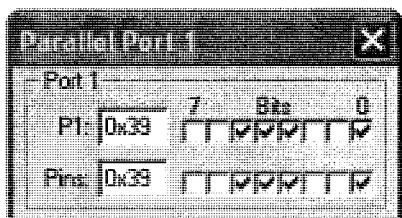
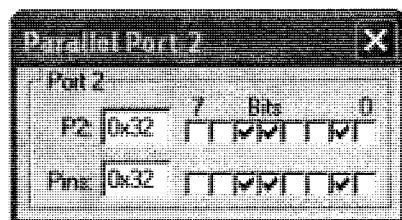
```
#include<reg51.h>
void main()
{
    unsigned char i;
    i=P1;
    i=i&0x3;           // mask unused bits

    switch(i)
    {
        case(0):
            {
                P0=0; // Decimal 00
                break;
            }
        case(1):
            {
                P0=1; // Decimal 01
                break;
            }
        case(2):
            {
                P0=2; // Decimal 02
                break;
            }
        case(3):
            {
                P0=3; // Decimal 03
                break;
            }
    }
}
```



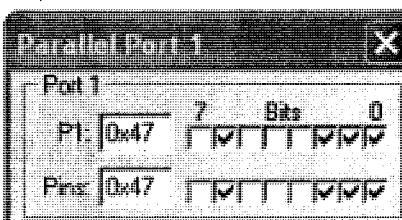
32. Write an 8051 C program to convert packed BCD 0x29 to ASCII & display the bytes on P₁ & P₂.

```
#include<reg51.h>
void main()
{
    unsigned char x,y,z;
    unsigned char mydata=0x29;
        // dont use data. It is keyword
    x = mydata & 0x0F;      // Mask lower 4 bits
    P1= x | 0x30;          // Add 30
    y=mydata & 0xF0; //Mask upper 4 bits
    y = y >> 4;           // Swap y
    P2 = y | 0x30;         // Add 30
}
```



33. Write an 8051 C program to convert ASCII digits of '4' and '7' to packed BCD and display them on P₁.

```
#include<reg51.h>
void main()
{
    unsigned char bcddata;
    unsigned char w = '4';
    unsigned char z = '7';
    w = w & 0x0F;
    w = w << 4;
    z = z & 0x0F;
    bcddata = w | z;
    P1 = bcddata;
}
```



(Same Program but displaying each result in Port 1.)

```
#include<reg51.h>
void main()
{
    unsigned char bcddata;
    unsigned char w = '4';
    unsigned char z = '7';
```

```
P1 = w;
```

```
P1 = z;
```

```
w = w & 0x0F;
```

```
P1 = w;
```

```
w = w << 4;
```

```
P1 = w;
```

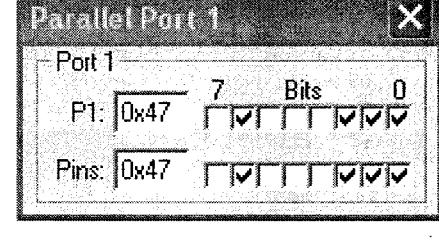
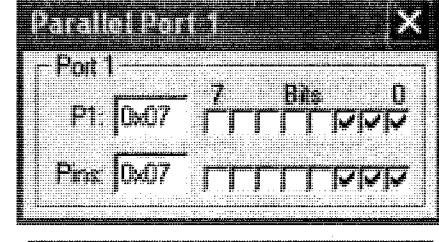
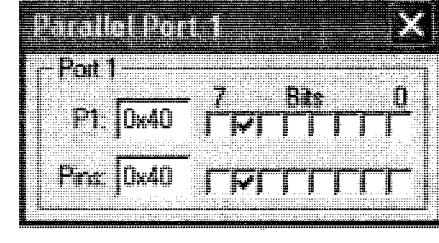
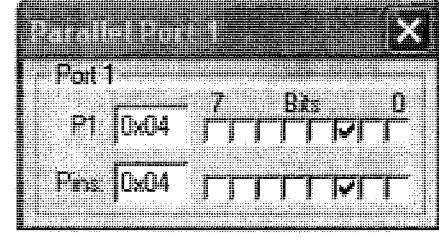
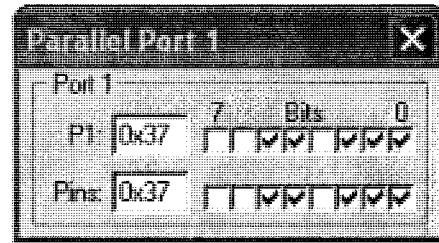
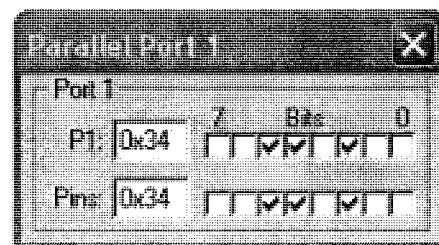
```
z = z & 0x0F;
```

```
P1 = z;
```

```
bcddata = w | z;
```

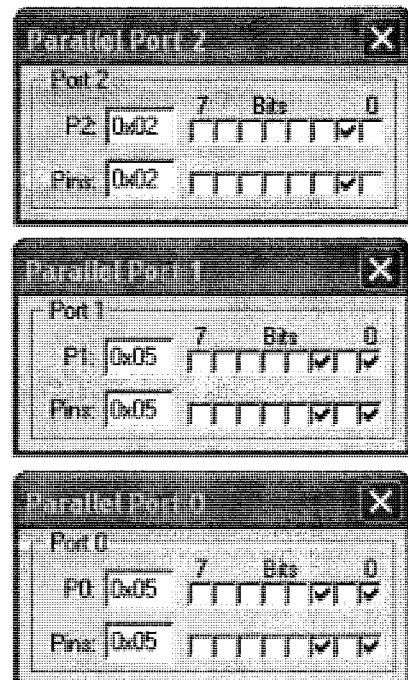
```
P1 = bcddata;
```

```
}
```



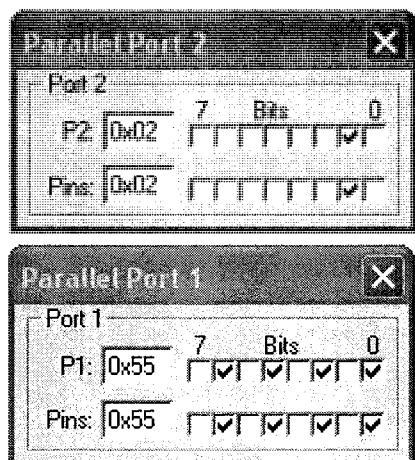
34. Write an 8051 C program to convert 11111111 (FF Hex) to decimal and display the digits on **P₀, P₁, & P₂**.

```
#include<reg51.h>
void main()
{
    unsigned char x, bindata, d1,d2,d3;
    bindata = 0xFF;
    x = bindata / 10;
    d1 = bindata % 10;
    d2 = x % 10;
    d3 = x / 10;
    P0 = d1;
    P1 = d2;
    P2 = d3;
}
```



OR

```
#include<reg51.h>
void main()
{
    unsigned char x, bindata, d1,d2,d3;
    bindata = 0xFF;
    x = bindata / 10;
    d1 = bindata % 10;
    d2 = x % 10;
    d3 = x / 10;
    d2 = d2 << 4;
    P1 = d2;
    P1 = d2 + d1;
    P2 = d3;
}
```



Accessing CODE ROM space & RAM Data space

Accessing RAM Memory:

35.

```
#include<reg51.h>
```

```
void main()
```

```
{
```

```
    unsigned char mydata[] = "ABCDEF";
    unsigned char i;
    for(i=0; i<=6; i++)
        P1 = mydata[i];
}
```

Memory #1	
Address:	00h
I : 0x00:	0A 08 00
I : 0x13:	00 00 00

Memory #1	
Address:	00h
I : 0x00:	0B 08 00
I : 0x13:	00 00 00

Memory #1	
Address:	00h
I : 0x00:	0C 08 00
I : 0x13:	00 00 00

Result:

After single-step the array numbers are stored in address 00h one by one.

00h = A

00h = B

:

00h = F

Memory #1	
Address:	00h
I : 0x00:	0F 08 00
I : 0x13:	00 00 00

A = 41h (ASCII) stored in address 09h of Internal Memory.

B = 42h (ASCII) stored in address 0Ah of Internal Memory.

:

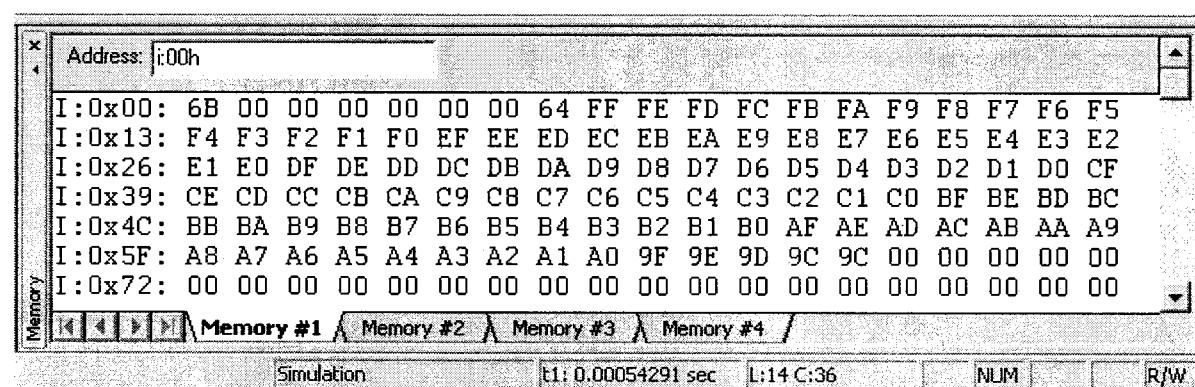
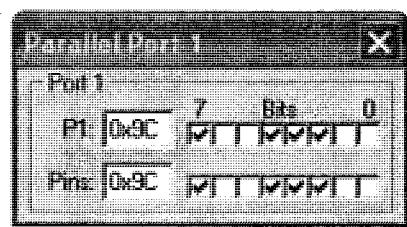
F = 46h (ASCII) stored in address 0Eh of Internal Memory.

Memory #1		Memory #2		Memory #3		Memory #4	
Address:	00h						
I : 0x00:	0E 08 00	00 00 00	00 00 00	01 07 41	42 43 44 45 46	00 09 01	
I : 0x13:	00 00 00	00 00 00	00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	



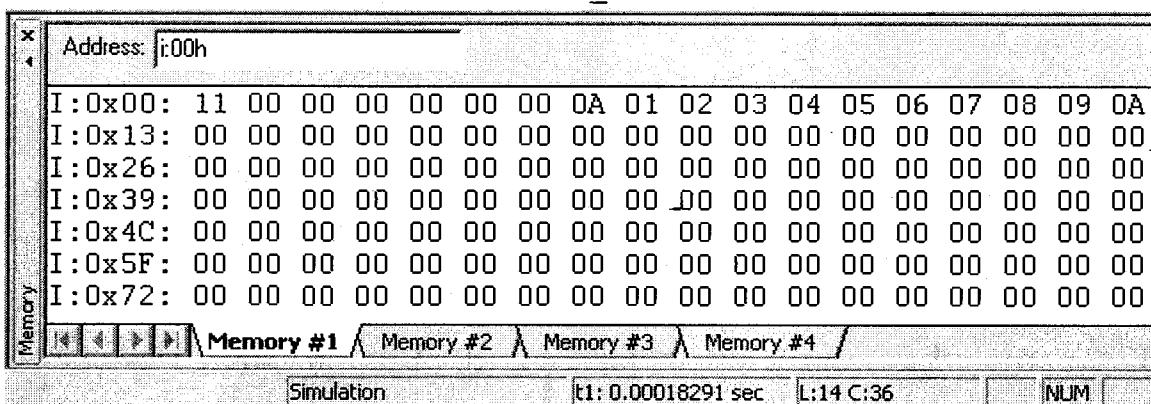
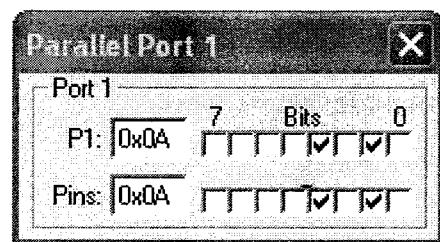
36. Down counter.

```
#include<reg51.h>
void main()
{
    unsigned char mydata[100];
    unsigned char i,z;
    for(i=0; i<100; i++)
    {
        z--;
        mydata[i] = z;
        P1 = z;
    }
}
```



37. UP counter.

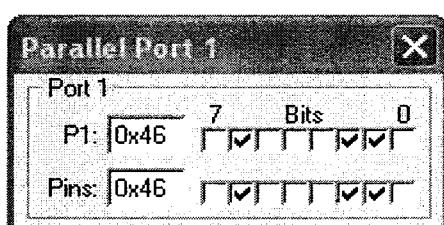
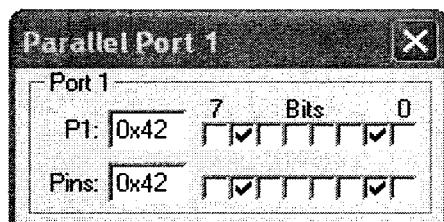
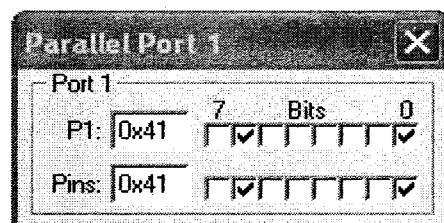
```
#include<reg51.h>
void main()
{
    unsigned char mydata[10];
    unsigned char i,z;
    for(i=0; i<10; i++)
    {
        z++;
        mydata[i] = z;
        P1 = z;
    }
}
```



Accessing CODE Memory:

38.

```
#include<reg51.h>
void main()
{
    code unsigned char mydata[] = "ABCDEF";
    unsigned char i;
    for(i=0; i<=6; i++)
        P1 = mydata[i];
}
```



Data Serialization using 8051 C:

* Data Serialization is a way of sending a byte of data one-bit at a time through a single pin of microcontroller.

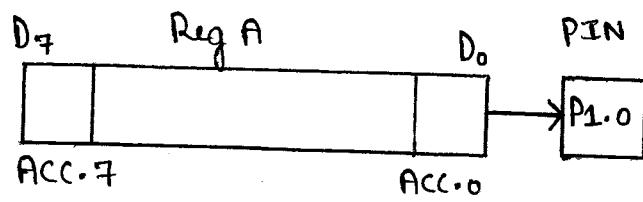
There are two ways to transfer a byte of data serially:

- 1) We can transfer data serially using Serial port & Supported hardware.
- 2) We can transfer data serially via P1.0 pin using Software control.



39. Write a C program to send out the value **44h** serially one bit at a time via **P_{1.0}**. The LSB should go out first.

```
#include<reg51.h>
sbit P1b0 = P1^0;
sbit regALSB = ACC^0;
void main()
{
    unsigned char mydata = 0x44;
    unsigned char i;
    ACC = mydata;
    for (i=0; i<8; i++)
    {
        P1b0 = regALSB;
        ACC = ACC >> 1;
    }
}
```



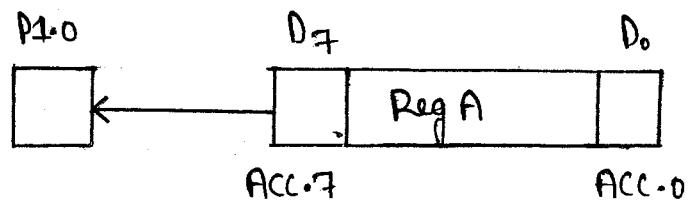
No. of Right Shifts	Accumulator Contents	P _{1.0} Contents
0	(0100 0100 = 44H) D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀ (Shift one bit position)	P _{1.0} = ACC.0 P _{1.0} = D ₀
1	0 D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁	P _{1.0} = D ₁
2	0 0 D ₇ D ₆ D ₅ D ₄ D ₃ D ₂	P _{1.0} = D ₂
3	0 0 0 D ₇ D ₆ D ₅ D ₄ D ₃	P _{1.0} = D ₃
4	0 0 0 0 D ₇ D ₆ D ₅ D ₄	P _{1.0} = D ₄
5	0 0 0 0 0 D ₇ D ₆ D ₅	P _{1.0} = D ₅
6	0 0 0 0 0 0 D ₇ D ₆	P _{1.0} = D ₆
7	0 0 0 0 0 0 0 D ₇	P _{1.0} = D ₇

- * For transmitting data serially with LSB 1st, we have to shift the data right by one-bit position eight times so that each bit will come & occupy the LSB of the register, say Accumulator is used & this LSB will be sent on the port pin P_{1.0} as shown in the table.



39. Write a C program to send out the value **44h** serially one bit at a time via P_{1.0}. The MSB should go out first.

```
#include<reg51.h>
sbit P1b0 = P1^0;
sbit regAMSB = ACC^7;
void main()
{
    unsigned char mydata = 0x44;
    unsigned char i;
    ACC = mydata;
    for (i=0; i<8; i++)
    {
        P1b0 = regAMSB;
        ACC = ACC << 1;
    }
}
```



F81 40th program:-

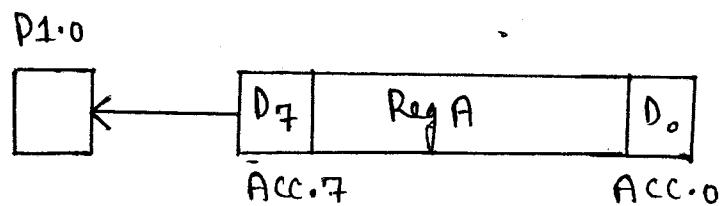
No of right shifts of accumulator	Received data bit on P1.0	Accumulator contents (Acc.7 = P1.0 after shifting)
0	D ₀ (LSB first)	D ₀ × × × . × × × → Initially D ₀ can be filled in at D ₇
1	D ₁	D ₁ D ₀ × × × × × }
2	D ₂	D ₂ D ₁ D ₀ × × × × }
3	D ₃	D ₃ D ₂ D ₁ D ₀ × × × × }
4	D ₄	D ₄ D ₃ D ₂ D ₁ D ₀ × × × }
5	D ₅	D ₅ D ₄ D ₃ D ₂ D ₁ D ₀ × × }
6	D ₆	D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀ × }
7	D ₇	D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀ }

- * The LSB of the data is received 1st, So feed it at D₇ of accumulator. So when the next bit is received, Shift the LSB received in accumulator by one right shift & enter the new bit again at D₇ position of accumulator. Repeat this for all 8-data bits received.



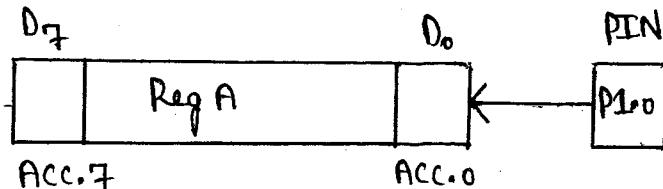
40. Write a C program to bring in a byte of data serially one bit at a time via P_{1.0}. The LSB should come in first.

```
#include<reg51.h>
sbit P1b0 = P1^0;
sbit ACCMSB = ACC^7;
void main()
{
    unsigned char mydata = 0x44;
    unsigned char i;
    ACC = mydata;
    for (i=0; i<8; i++)
    {
        ACCMSB = P1b0;
        ACC = ACC >> 1;
    }
    P2 = ACC;
}
```



41. Write a C program to bring in a byte of data serially one bit at a time via P_{1.0}. The MSB should come in first.

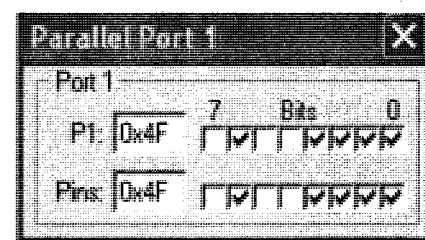
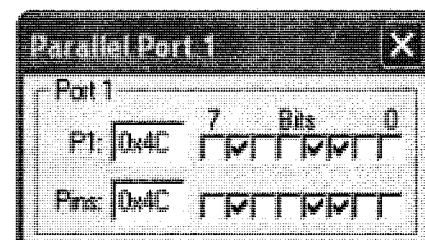
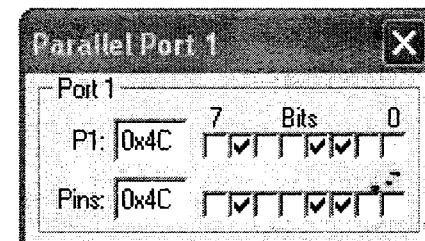
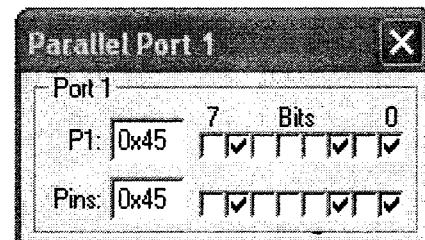
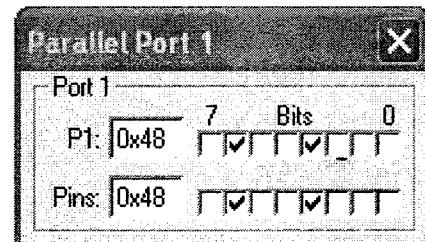
```
#include<reg51.h>
sbit P1b0 = P1^0;
sbit regALSB = ACC^0;
void main()
{
    unsigned char i;
    for (i=0; i<8; i++)
    {
        regALSB = P1b0;
        ACC = ACC << 1;
    }
    P2 = ACC;
}
```



42. Compare and contrast the following programs and discuss the advantages of each one.

I)

```
#include<reg51.h>
void main()
{
    P1='H';
    P1='E';
    P1='L';
    P1='L';
    P1='O';
}
```



II)

```
#include<reg51.h>
void main()
{
    unsigned char mydata[ ] = "HELLO";
    unsigned char i;
    for(i=0; i<=5; i++)
        P1 = mydata[i];
}
```

III)

```
#include<reg51.h>
void main()
{
    code unsigned char mydata[ ] = "HELLO";
    unsigned char i;
    for(i=0; i<=5; i++)
        P1 = mydata[i];
}
```



Sol:-

All the programs Sends out "HELLO" to P₁, one character at a time, but they do it in different ways.

- I) The 1st program is Short & Simple, but the Separate Statements are written to Send each Character to Port P₁. Thus, if we want to Change the Character of message, the Whole program changes. So this method is NOT Good.
- II) The Second Program uses array to Store message in the RAM data Space, therefore the Size of the array is limited.
- III) The third Program uses array to Store message, but it uses CODE data Space. This allows the Size of the array to be as long as you want if you have the on-chip ROM. However, the more Space you use for data, the less Space is left for your program code.

NOTE:-

- * If we want to Change the String & make it longer, then we can easily upgrade in 2nd & 3rd program.
- * Program 1 cannot be easily upgradeable.



UNIT - V

TIMERS / COUNTERS IN 8051

* The 8051 has two 16-bit Timers / counters. They can be used either as Timers to generate a time delay or as counters to count events happening outside the microcontroller.

i.e.

- 1) Timer-0 register (16-bit)
- 2) Timer-1 register (16-bit)

Each 16-bit registers can be accessed as two separate 8-bit registers of low byte & high byte.

Timer-0 register :-

- * The 16-bit register of Timer 0 is accessed as low byte & high byte.
- * The low byte register is called TLO & the high byte register is called THO.
- * These registers can be accessed like any other registers, such as A, B, R0, R7 etc.

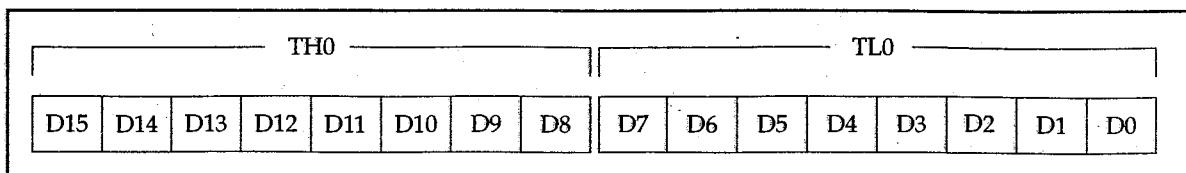
ex:- MOV TLO, #4Fh ; move the value 4Fh into TLO register.

MOV R5, THO ; save the contents of THO in R5 register.



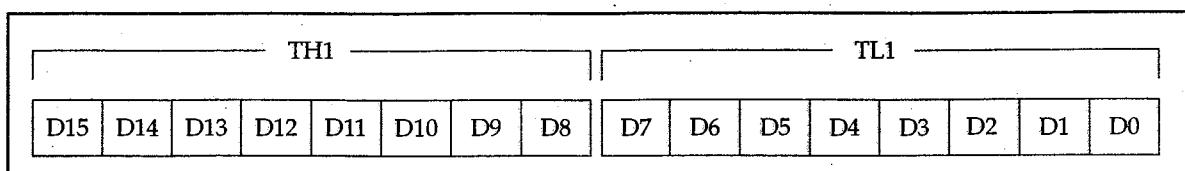
①

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.



Timer 0 Registers

Timer 1 register :-



Timer 1 Registers

- * Timer 1 is also 16-bit register & is accessed as two 8-bit registers, i.e. TL1 (Timer 1 low byte) & TH1 (Timer 1 high byte).
- * These registers can be accessed like any other registers such as A, B, R0, R3 etc.

ex:-
MOV TL1, #4FH ; move the value 4FH into TL1
MOV R0, #TH1 ; move the contents of TH1 into R0.
register.

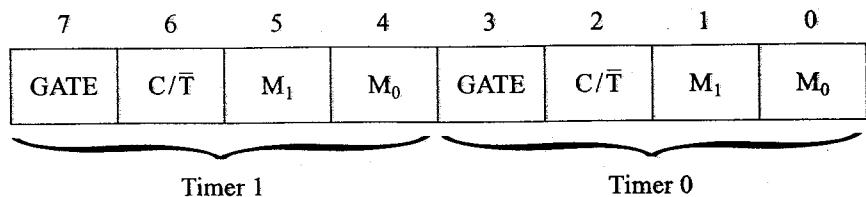
TMOD (Timer mode) register :-

- * Timer 0 & Timer 1 use the same register, called TMOD, to set the various timer operation modes.
- * TMOD is an 8-bit register in which the lower 4-bits are set aside for Timer 0 & the upper 4-bits for Timer 1



②

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.



For Timer 1 :-

Bit 7 : Gate :-

- * When Gate=1, timer 1 will run only when $\overline{INT1}$ (P3.3) Pin is high.
- * When Gate=0, timer 1 will run, regardless of the state of $\overline{INT1}$ pin.
- * TR1 in TCON register must be set to 1 for timer 1 to Run. (i.e. TR1=1 to run Timer 1)

Bit 6: C/T :-

- * When C/T=1, counter mode.
In counter mode, Timer 1 will count events (pulses) on T1 pin (P3.5).
- * When C/T=0, Timer mode.
In Timer mode, Timer 1 will increment every machine cycle.



(3)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Bit 5 & 4 : M₁ & M₀ :-

M ₁	M ₀	Mode	Operating Mode.
0	0	0	13-bit Timer / counter ie (8-bit \rightarrow TH ₁ 5-bit \rightarrow TL ₁)
0	1	1	16-bit Timer / counter (8-bit \rightarrow TH ₁ 8-bit \rightarrow TL ₁)
1	0	2	8-bit auto-reload Timer / counter (TH ₁ holds a value that is to be reloaded into TL ₁ each time when TF ₁ overflows)
1	1	3	Split timer mode.

For Timer 0 :-

Bit - 3 : Gate :-

- * When Gate = 1, Timer 0 will run, only when $\overline{\text{INTO}}$ (P3.2) Pin is high.
- * When Gate = 0, Timer 0 will run, regardless of the state of $\overline{\text{INTO}}$ pin.
- * TRO in TCON register must be set to 1 for Timer 0



to Run. (i.e $TRO=0$ to run Timer 0).

Bit 2 : C/T :-

- * When $C/T=1$, counter mode.

In counter mode, Timer 0 will count events (pulse) on T0 pin (P3.4).

- * When $C/T=0$, Timer mode.

In Timer mode, Timer 0 will increment every machine cycle.

Bit 1 & 0 : M1 & M0 :-

M1	M0	Mode	Operating Mode.
0	0	0	13-bit Timer/counter (8-bit \rightarrow TH0 & 5-bit \rightarrow TL0)
0	1	1	16-bit Timer/counter (8-bit \rightarrow TH0 & 8-bit \rightarrow TL0)
1	0	2.	8-bit auto-reload Timer/counter (TH0 holds a value that is to be reloaded into TL0 each time when TFO overflows).
1	1	3.	split timer mode.



NOTE:-

- * The only difference between counter & timer is the source of the clock pulses to the timer/counter.
- * When used as a timer, the clock pulses are sourced from the oscillator through the divide by 12-d circuit.
- * When used as a counter, Pin T0 (P3.4) supplies pulses to counter 0, & pin T1 (P3.5) supplies pulses to counter-1.
- * The C/T bit in TMOD must be set to 1 to use as a counter & set to 0 to use as a timer.



TMOD Register (Timer Mode)

This register selects mode of the timers T0 and T1. The lower 4 bits (bit0 - bit3) refer to the timer 0, while the higher 4 bits (bit4 - bit7) refer to the timer 1.

									Value after reset
									Bit name
TMOD	GATE1	C/T1	T1M1	T1M0	GATE0	C/T0	T0M1	T0M0	
	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	

- ❖ **GATE1** starts and stops Timer 1 by means of a signal provided to the pin INT1 (P3.3):
 - 1 - Timer 1 operates only if the bit INT1 is set
 - 0 - Timer 1 operates regardless of the state of the bit INT1

- ❖ **C/T1** selects which pulses are to be counted up by the timer/counter 1:
 - 1 - Timer counts pulses provided to the pin T1 (P3.5)
 - 0 - Timer counts pulses from internal oscillator

- ❖ **T1M1, T1M0** These two bit selects the Timer 1 operating mode.

T1M1	T1M0	Mode	Description
0	0	0	13-bit timer
0	1	1	16-bit timer
1	0	2	8-bit auto-reload
1	1	3	Split mode

- ❖ **GATE0** starts and stops Timer 1, using a signal provided to the pin INT0 (P3.2):
 - 1 - Timer 0 operates only if the bit INT0 is set
 - 0 - Timer 0 operates regardless of the state of the bit INT0

- ❖ **C/T0** selects which pulses are to be counted up by the timer/counter 0:
 - 1 - Timer counts pulses provided to the pin T0(P3.4)
 - 0 - Timer counts pulses from internal oscillator

- ❖ **T0M1, T0M0** These two bits select the Timer 0 operating mode.

T0M1	T0M0	Mode	Description
0	0	0	13-bit timer
0	1	1	16-bit timer
1	0	2	8-bit auto-reload
1	1	3	Split mode

Examples :-

I) Configure Timer 0 to run as a.

A 16 bit timer with only Internal control.

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1

Timer 1 not used

TMOD = 01H

2) 16 - bit timer with external control.

0	0	0	0	1	0	0	1
---	---	---	---	---	---	---	---

TMOD = 09H

3) 16 - bit counter with Internal control.

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

TMOD = 05H

4) 16 - bit counter with external control.

0	0	0	0	1	1	0	1
---	---	---	---	---	---	---	---

TMOD = 0DH

II) Find the values of TMOD to operate as timers in the following modes.

i) Mode 1 Timer 1

→ TMOD is $00010000 = 10H$.



ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

ii) Mode 2 Timer 0, Mode 2 Timer 1

→ TMOD is $01010010 = 52H$.

iii) Mode 0 Timer 1

→ TMOD is $00000000 = 00H$.

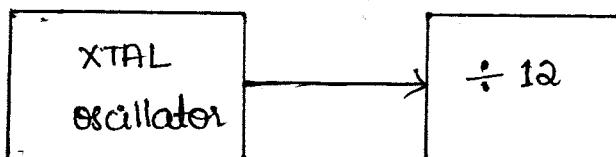
III) Find the timer's clock frequency & its period for various 8051 - based systems, with the following crystal frequencies.

a) 12 MHz.

b) 16 MHz.

c) 11.0592 MHz.

Sol:-



a) $\frac{12 \text{ MHz}}{12} = 1 \text{ MHz}$ clock frequency.

b) period $T = \frac{1}{1 \text{ MHz}} = 1 \mu\text{sec.}$

b) $\frac{16 \text{ MHz}}{12} = 1.33 \text{ MHz}$ clock frequency.

period $T = \frac{1}{1.33 \text{ MHz}} = 0.75 \mu\text{sec.}$

c) clock frequency. $\frac{11.0592 \text{ MHz}}{12} = 921.6 \text{ KHz.}$

period $T = \frac{1}{921.6 \text{ KHz}} = 1.085 \mu\text{sec.}$



⑧

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

TCON (Timer Control SFR) :-

7	6	5	4	3	2	1	0
TF1	TR1	TFO	TRO	IE1	IT1	IEO	ITO

Fig ① TCON Register format.

TF1 : Timer 1 overflow flag.

$TF_1 = 1$, when timer rolls from all 1's to 0.

$TF_1 = 0$ (cleared), when processor vectors to execute interrupt service routine located at program address 001Bh.

TR1 :- Timer 1 run control bit

Set to 1 by program to enable timer to count.
Cleared to 0 by program to halt timer. Does not reset timer.

TFO :- Timer 0 overflow flag.

$TFO = 1$ (set), when timer rolls from all 1's to 0.

$TFO = 0$ (cleared), when processor vectors to executed interrupt service routine located at program address 000Bh.

TRO :- Timer 0 run control bit.

Set to 1 by program to enable timer to count
Cleared to 0 by program to halt timer. Does not reset Timer.



IE1 : External Interrupt 1 Edge flag.

IE1 = 1 (set), when a high-to-low edge signal is received on port 3 pin 3.3 ($\overline{\text{INT1}}$).

IE1 = 0, (cleared), when processor vectors to interrupt service routine located at program address 0013h. Not related to timer operation.

IT1 : External Interrupt 1 signal type control bit.

IT1 = 1 (set), by program to enable external interrupt 1 to be triggered by a falling edge signal.

IT1 = 0 (cleared) by program to enable a low-level signal state on external interrupt 1 to generate an interrupt.

IE0 : External Interrupt 0 Edge flag.

IE0 is set to 1 when a high-to-low edge signal is received on port 3 pin 3.2 ($\overline{\text{INT0}}$)

IE0 is cleared when processor vectors to interrupt service routine located at program address 0003h. Not related to timer operation.



10

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

IT0 : External Interrupt 0 signal type control bit.

Set to 1 by program to enable external interrupt 0 to be triggered by a falling edge signal.

Set to 0 by program to enable a low-level signal on external interrupt 0 to generate an interrupt.

Timer Modes of operation:-

The timer may operate in any one of four modes (i.e. Mode 0, Mode 1, Mode 2 & Mode 3) that are determined by the mode bits M1 & M0 in the TMOD register.

Timer in Mode 0 :-

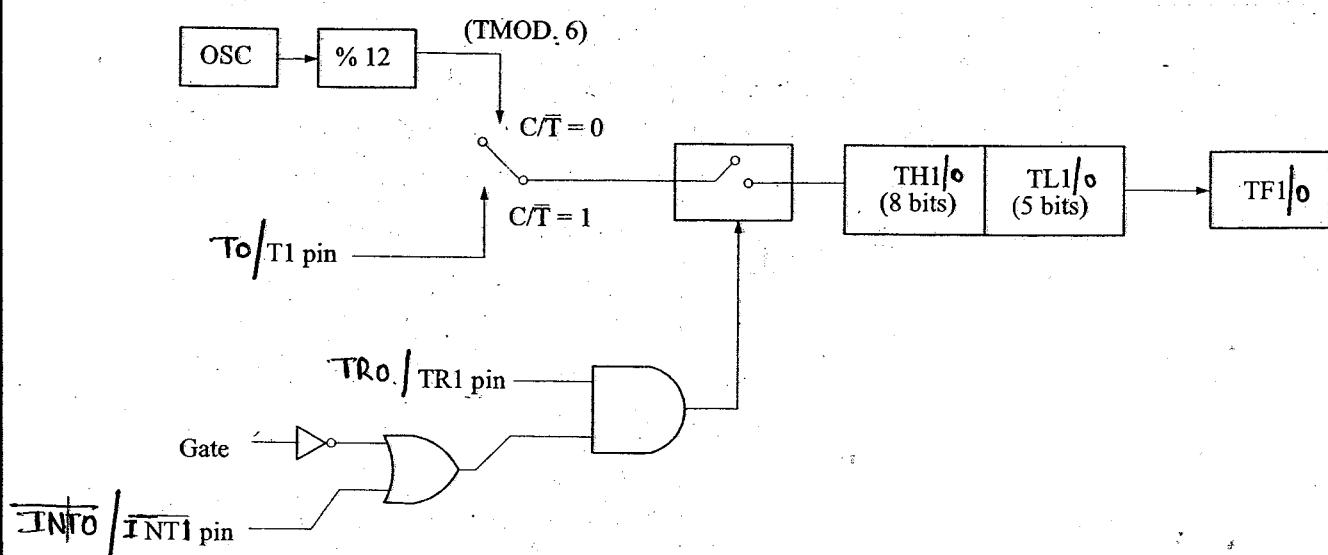


Fig ① shows the Timer 0/1 in Mode 0.



- * In this mode the registers TH0|1 & TL0|1 operate as a 13-bit timer.
- * The lower 5-bits are in TL0|1 register & the upper 8-bits are in TH0|1 register.
- * The timer interrupt flag TFO|TF1 bit is set, when the 13-bit value in the TH0:TL0|TH1:TL1 rolls over from 1FFFFh to 0000h.
 - * The TFO|TF1 bit can then be used to interrupt the 8051 if needed. ?

The following settings are important to operate Timer in Mode 0.

- 1) C1T bit in TMOD register is cleared to '0' to allow Timer mode operation.
- 2) M1:M0 bits are cleared to 0:0 in TMOD register to select mode 0.
- 3) When Gate=0 in TMOD register, the timer will operate only if TR=1 (ie TRO|TR1=1).
- 4) When Gate=1 in TMOD register, the timer will run only if TR=1 & INT0|1=1.



NOTE :-

- * Mode 0 for both Timer 0 & Timer 1.
- * For Timer T₀, we have to use T_{H0}, T_{L0} & T_{R0} register & lower 4-bits of TMOD register.
- * For Timer T₁, we have to use T_{H1}, T_{L1} & T_{R1} register & higher 4-bits of TMOD register.

- {
- 1) Timer x = Timer 0 or Timer 1)
 - 2) Mode x = Mode 0 or Mode 1 or Mode 2 or Mode 3)
 - 3) TF x = T_{F0} or T_{F1}
 - 4) TR x = T_{L0} or T_{L1}.
 - 5) TH_x = T_{H0} or T_{H1}.
 - 6) TR_x = T_{R0} or T_{R1}.
- g.

Timer in Mode 1 :- (Timer 0 & Timer 1 in Mode 1)

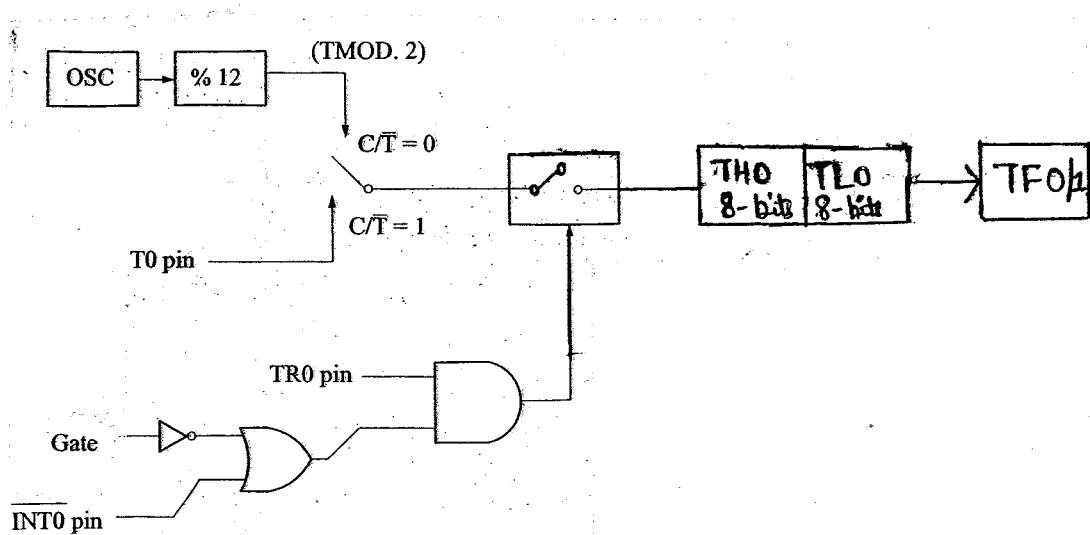


Fig shows the block diagram of Timer 0/1 in Mode 1.



(13)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Mode 1 programming :-

The following are the characteristic & operation of mode 1:

- 1) It is a 16-bit timer, so 0000 to FFFFh can be loaded into timer's register TH & TL.
- 2) After TH & TL are loaded with a 16-bit initial value, the timer must be started. This is done by "SETB TR0" instruction for Timer 0 & "SETB TR1" instruction for Timer 1.
- 3) After the timer is started, it starts to count up until it reaches its maximum value of FFFFh. When it rolls over from FFFFh to 0000h, it sets high a flag bit called TF (Timer Flag).
- 4) When TF is set, we must stop the timer by using instructions "CLR TR0" for Timer 0 & "CLR TR1" for Timer 1.
- 5) When Timer rolls over, in order to repeat the process the TF must be reset to 0. & TH & TL must be reloaded with the original value (initial value).



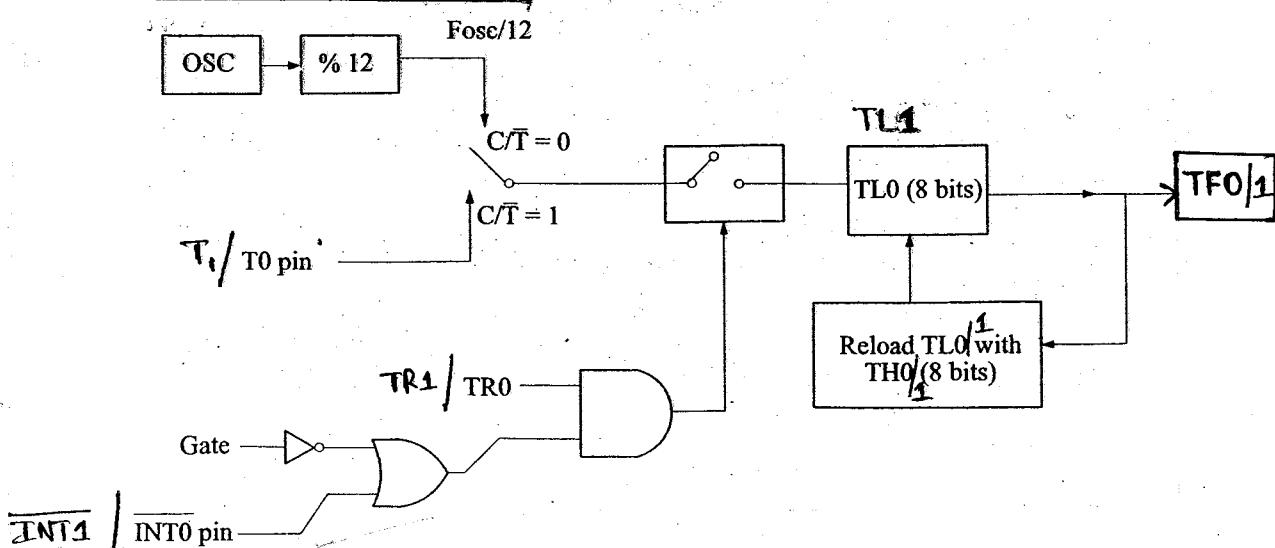
14

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Steps to program in mode 1 :-

- 1) Load the TMOD register value indicating which timer to be used (timer 0 or timer 1) and which timer mode is selected (mode 0 or 1).
- 2) Load registers TH & TL with initial count values.
- 3) Start the Timer (set TR bit in TCON register)
- 4) Keep monitoring the timer flag (TF) with the "JNB TFx", here instruction until the timer flag is set. Get out of the loop when TF becomes high.
- 5) Stop the timer (set TRX = 0).
- 6) clear the TF flag for the next sound (TFX = 0)
- 7) Repeat from step 2 for the next delay.

Timer X in Mode 2 :-



(IS)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

{

- * In this mode the timer act as an 8-bit timer using register TLX for counting up.
- * The value in THX gets loaded into TLX whenever the value in TLX overflows by exceeding the count of FFh.
This mode is called the 8-bit auto-reload mode.
- * Overflow from the TLX register will result in setting the TFX flag.

g.

Mode 2 programming :-

The following are the characteristics & operations of mode 2.

- 1) It is an 8-bit timer.
 \therefore It allows only values of 00 to FFh to be loaded into the timer's register TH.
- 2) After TH is loaded with the 8-bit value, the 8051 gives a copy of TH to TL. Then the timer must be started. This is done by the instruction "SETB TRX" for Timer X.



⑯

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

3) After the timer is started, TH register starts to count up by incrementing the TL register.

When TH register overflows from FFh to 00h, it set high the TF flag ($TF=1$).

4) When $TF=1$, TH register is automatically reloaded with the original value kept by the TH register.

To repeat the process, we must clear TF register & programmer no need to reload the original value. This makes mode 2 an auto-reload.

Steps to program in Mode 2:-

To generate a time delay using the timer's mode 2, take the following steps:

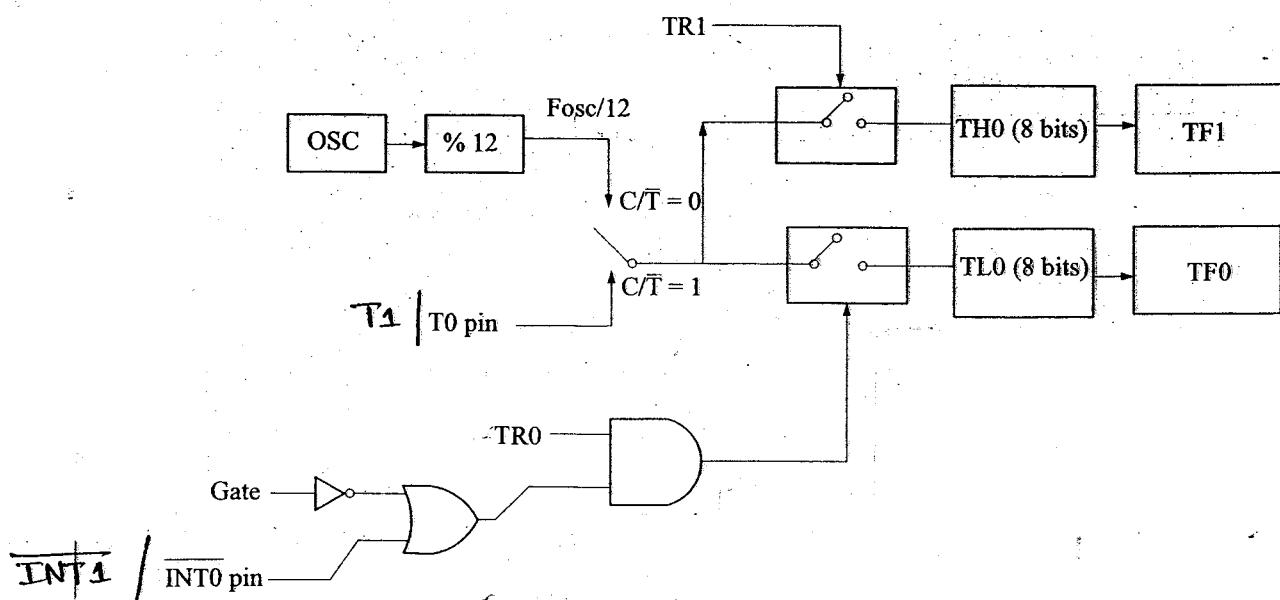
- 1) Load the TMOD register value indicating which timer (Timer 0 or Timer 1) is to be used & then select the timer mode (Mode 2).
- 2) Load the TH register with the initial count value.
- 3) Start the timer (set TR bit in TCON register).
- 4) Keep monitoring the timer flag (TF) with the "JNB TFX; here" instruction until the timer flag is



set. Get out of the loop when TF becomes high.

- 5) Stop the timer (clear TRX=0).
- 6) Clear the TF flag.
- 7) Go back to step 4, since mode 2 is Auto-reload.

Timer X in Mode 3:-



* Timer X in mode 3 is setup as two independent timers as shown in fig.

The features of Timer X in Mode 3 are illustrated below:

- 1) TRX is an 8-bit timer counting up at a rate of Fosc/12.
- 2) Timer X is started by setting TRX bit.



- 3) Overflow from TLX will set the TFO flag.
 - 4) THX is an independent 8-bit timer counting at the rate of $f_{osc}/12$.
 - 5) TRX=0 stops the timer & holds the contents in THX.
 - 6) Overflow from THX will set the timer overflow flag TFI bit.
-
- 1) Obtain the table to configure Timer 0 in all possible operating modes.
- * Timer 0 is turned ON/OFF by setting/clearing bit TRO in TCON register, using an instruction. (Software control).
- * When external control is used, Timer 0 is turned ON/OFF by the high to low transition on INT0 pin (P3.2) when TRO=1 (Hardware control).

NOTE:- We assume that Timer 1 is not used.
Timer 0 operation.

Mode	Timer 0 function	TMOD Internal control (GATE = 0)	External control (GATE = 1)
0	13-bit timer	00H	08H
1	16-bit timer	01H	09H
2	8-bit auto-reload timer	02H	0AH
3	Two 8-bit timers	03H	0BH
0	13-bit counter	04H	0CH
1	8-bit auto-reload counter	05H	0DH
2	8-bit auto-reload counter	06H	0EH
3	One 8-bit counter	07H	0FH



2) Obtain the table to configure Timer 1 in all possible operating modes.

- * Timer 1 is turned ON/OFF by setting/clearing bit TR1 in TCON register, using an instruction. (Software control).
- * When external control is used, Timer 1 is turned ON/OFF by the high to low (1 to 0) transition on INT1 pin (P3.3) when TR0=1. (Hardware control).

NOTE:-

We assume Timer 0 is NOT running.

Timer 1 operation.

Mode	Timer 1 function	TMOD Internal control	External control
0	13-bit timer	00H	08H
1	16-bit timer	10H	09H
2	8-bit auto-reload timer	20H	0AH
3	does not run	30H	0BH
0	13-bit counter	40H	0CH
1	16-bit counter	50H	0DH
2	8-bit auto-reload counter	60H	0EH
3	Not available	70H	0FH

How to use Timer 0 & Timer 1 simultaneously.

- * If we desire to run both Timers simultaneously, then the value of TMOD for Timer 0 is ORed with the value of Timer 1.



Ex:-

- 1) what is the value of TMOD to run Timer 0 in Mode 2 & Timer 1 as a counter in Mode 1 ?
- * The TMOD value for Timer 0 in Mode 2 is $\rightarrow 02h$.
 - * The TMOD value for Timer 1 as counter in mode 1 is $\rightarrow 50h$.
 - * The value of Timer 0 is ORed with the value of Timer 1 i.e., $TMOD = 50h + 02h = 52h$.
 - * Then set $TR0=1$ & $TR1=1$
- 2) what is the value of TMOD to run Timer 0 in Mode 1 with external control & Timer 1 in Mode 2 as a counter ?
- * The TMOD value for Timer 0 in Mode 1 with external control is $09h$.
 - * The TMOD value for Timer 1 in mode 2 as counter is $60h$.
 - * The value of Timer 0 is ORed with the value of Timer 1 ie.
- $$TMOD = 60h + 09h = 69h.$$
- * Then set $TR0=1$ & $TR1=1$.



21

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Procedure for time delay generation :-

- 1) Initialize TMOD for the required Timer (0 or 1) & Mode (0 or 1 or 2).
- 2) Load the initial count value in registers TH & TL (of timer 0 or 1).
- 3) Start the Timer (SET TR1 or TR0 bit of TCON register).
- 4) Wait until the Timer flag is set (TF1 or TF0 of TCON).
- 5) Stop the Timer (TR0 or TR1 is cleared).
- 6) clear TF flag.
- 7) Repeat from step 2 for the next delay.

Formula to compute initial value of Timer (TH & TL) :-

$$(\text{Initial value} - 1) = \frac{\text{Maximum Value}}{\text{delay}} - \frac{\text{Required} \times \text{crystal frequency}}{12}$$

NOTE:-

- * The Maximum value depends on the mode of the timer i.e,
 - 1) Mode 0 \rightarrow 13-bit timer $= 2^{13} = 1FFF\text{h}$.
 - 2) Mode 1 \rightarrow 16-bit timer $= 2^{16} = FFFF\text{h}$.
 - 3) Mode 2 \rightarrow 8-bit timer $= 2^8 = FF\text{h}$.
 - 4) Mode 3 \rightarrow 8-bit split timer $= 2^8 = FF\text{h}$.



Range of different Modes :-

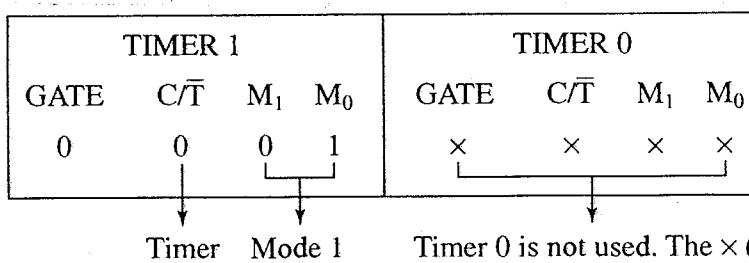
Mode	Initial value in hex	Maximum Value in hex	Maximum delay.
Mode 0	0000h	1FFFh.	
Mode 1	0000h.	FFFFh.	
Mode 2	00h	FFh	
Mode 3.	00h	FFh.	

MODE 1 :

- * Write a 8051 assembly & C program to generate a delay of 12 μsec using Timer 1 in Mode 1 with XTAL frequency of 22MHz.

Sol:- * In Mode 1, the timer counts up till FFFFh. (maximum value for 16-bit timer) then it rolls over from FFFFh to 0000h & sets the TF flag bit ($TF = 1$).

- * The timer mode register TMOD is programmed as



Q3

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Hence TMOD value is 10h for Timer 1 in Mode 1.

- * The initial count value (in hex) to be loaded into TH1 & TL1 is computed as.

$$[\text{Initial value} - 1] = \frac{\text{Maximum value}}{\text{Time delay} \times \frac{\text{Crystal frequency}}{12}}$$

$$= \text{FFFFh.} - \frac{12 \mu\text{sec} \times 22 \text{MHz}}{12}$$

$$(65535d)$$

$$= 65535d - 22.$$

$$[\text{Initial value} - 1] = 65513 \text{ in decimal}$$

$$\text{Initial value} = 65513 + 1$$

$$= 65514d$$

$$\boxed{\text{Initial value} = \text{FFEAh.}}$$

- * The initial 16-bit value should be loaded into 16-bit timer register TH1L = FFEAh i.e.
TH = FFh (MSB) &
TL = EAh (LSB).

Assembly program:- program 1

ORG 00h.

MOV TMOD, #10h ; Timer 1, Mode 1

up: MOV TL1, #0EAh ; Initial value LSB.

MOV TH1, #0FFh ; Initial value MSB.

SETB TR1 ; Start Timer 1

here: JNB TF1, here ; wait until Timer 1 counts upto the



(24)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Maximum i.e. Jump if $TF1=0$

CLR TR1 ; Stop Timer 1.

SJMP up ; clear TF1 so that it can be set again.

END ; Repeat again.

8051 C program :-

```
#include <reg51.h>
```

```
Void main( )
```

```
{
```

```
TMOD = 0x10 ;           // Timer 1, Mode 1.
```

```
while(1)
```

```
{
```

```
TR1 = 0xEA ;           // initial value is loaded in TL1 & TH1
```

```
TH1 = 0xFF ;
```

```
TR1 = 1 ;             // start timer 1.
```

```
while (TF1 == 0);
```

// The semicolon (;) implies that
while loop is executed until
 $TF1=0$ & comes out when
 $TF1=1$.

```
TR1 = 0 ;             // stop Timer 1.
```

```
TF1 = 0 ;             // clear TF1 for next roll over.
```

```
}
```

```
}
```



- * write a 8051 assembly & c program to find the delay generated by Timer1 in mode 1 with XTAL frequency of 11.0592 MHz with initial value FFEAh by
 - i) NOT considering the overhead due to instructions.
 - ii) overhead due to instructions.

Sol:-

- i) Refer program 1.

$$[\text{Initial value} - 1] = \frac{\text{Maximum value}}{\text{Time delay} \times \frac{\text{crystal freq}}{12}}$$

$$\frac{\text{Time delay} \times \text{crystal freq}}{12} = \frac{\text{Maximum value}}{\text{Time delay}} - \text{Initial value} + 1$$

$$\text{Time delay} = \left[\frac{\text{Maximum value} - \text{Initial value} + 1}{\text{crystal freq.}} \right] \times \frac{12}{\text{crystal freq.}}$$

$$= (FFFFh - FFEAh + 1) \times \frac{12}{11.0592 \text{ MHz.}}$$

$$= 22d \times 1.085 \mu\text{sec.}$$

$\text{Time delay} = 23.87 \mu\text{sec.}$



NOTE:-

- 1) (Initial value + 1) is added in the above formula because of the extra clock pulse is needed when the timer rolls over from FFFFh to 0000h & raises the TF flag.
- 2) The same calculations holds good for the delay generated by the timer in the 'c' program.
- ii) Delay calculations including overheads due to instructions :- c program :-
 - * The delay generated by the Timer 1 for while loop is same for both 'c' & assembly programs by using formula.
 - * The time taken to execute c-code ie, initialize TMOD, TL, TH etc, making TR1 = 1 | 0 etc cannot be exactly computed.
Hence the time delay duration for the entire 'c' program is not easy to calculate.

Assembly program :-

The total delay including the overhead due to the instructions for the assembly language program is calculated as shown below.



04

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

	<u>Instruction</u>	<u>Machine cycles</u>	
AGAIN:	MOV TMOD, #10H	2	
	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	

Formula :-

$$T = \frac{c \times 12d}{\text{crystal freq.}}$$

where $c \rightarrow$ Machine cycles.

$$\begin{aligned} \text{Total delay (including overhead)} &= 33 \text{ machine cycles} \times \frac{12d}{\text{crystal freq.}} \\ &= 33 \times 1.085 \mu\text{sec} \\ &= \underline{\underline{35.805 \mu\text{sec.}}} \end{aligned}$$

NOTE :-

- 1) Time delay without including overhead = 23.87 μsec
- 2) Time delay including overhead = 35.805 $\mu\text{sec.}$



Generation of Square Wave :-

- * The time delays generated by the timers are used in many applications. one of the most widely used application is in generation of the square wave.

This is done by toggling (complementing) port pins at a repeated fixed interval of time depending on the duty cycle.

- * Generate a square wave of 50% duty cycle on P2.3
Use timer0 in Mode 1 to generate the delay. Use a initial value of FF4E for the timer & a crystal frequency of the square wave generated by
 - i) NOT considering the overhead due to instructions
 - ii) Considering the overhead.

Sol :-

The algorithm for the square wave generation is given below:

- 1) Initialize the Timer0 in Mode 1.
- 2) Load the initial value FF4E into TLO & TH0.
- 3) Complement pin P2.3 (Toggle).
- 4) Call Timer delay subroutine.
- 5) Repeat from step 2.



Q9

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

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.

TIMER 1				TIMER 0				TMOD = 01H
GATE	C/T	M ₁	M ₀	GATE	C/T	M ₁	M ₀	
x	x	x	x	0	0	0	1	

↓ ↓
Timer Mode 1

The assembly language program is given below.

	PROGRAM	COMMENTS	MACHINE CYCLE
AGAIN:	MOV TMOD, #01H	;Timer 0, mode 1	2
	MOV TL0, #4EH	;Initial value	2
	MOV TH0, #FFH		2
	CPL P2.3	;Toggle the port pin	1
DELAY:	LCALL DELAY	;call delay	3
	SJMP AGAIN	;repeat	2
WAIT:	SETB TR0	;Start timer	1
	JNB TF0, WAIT	;wait till TF0 = 1	Timer delay cycles
	CLR TR0	;Stop timer	1
	CLR TF0	;clear TF	1
	RET		2
<hr/>			
Total = Timer delay + 17 cycles			



30

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

The corresponding 'C' program is given below.

```
#include <reg51.h>
void delay (void);
sbit squrpin=P2^3; //assign a variable name to the pin
void main ()
{TMOD=0x01; //initialize Timer-mode 1, Timer 0
 while(1)
 {sqrpin=~sqrpin, //Toggle pin
  TL0=0x4E; //load initial count
  TH0=0xFF;
  }delay ();
}
void delay (void) //start timer
{TR0=1; //wait till timer 0 overflow
 while (TF0==1);
 TR0=0; //stop timer
 TF0=0; //clear TF
}
```

The square wave is generated at pin P2.3 as shown below.

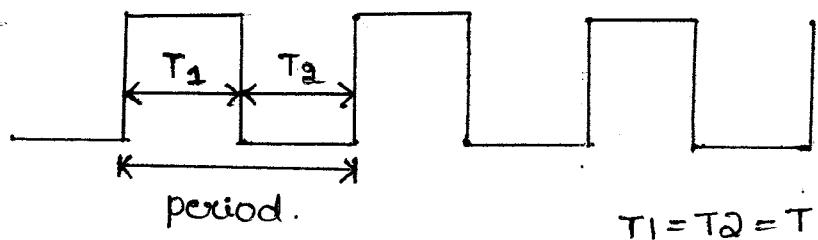


fig ① Square wave

* Total period of square wave is $T = T_1 + T_2$

$$T = 2T$$

* Frequency = $\frac{1}{\text{Total period}} = \frac{1}{2T}$



(31)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

$$* \text{ Duty cycle} = \frac{T_{ON}}{T_{ON} + T_{OFF}} = \frac{T}{T+T} = \frac{T}{2T} = \frac{1}{2} = 0.5$$

% duty cycle = 50%.

NOTE :-

Since in the above program the same delay is used for both ON time & OFF time, we have $T_1 = T_2 = T$ & the total period of square wave = $2T$ & the frequency is $\frac{1}{2T}$.

i) The frequency of the generated square wave can be calculated. by calculating the delay 'T'.

$$\text{Time delay} = \frac{12}{\text{crystal freq}} \times \left[\frac{\text{Maximum value} - \text{Initial value}}{16} + 1 \right]$$

$$= \frac{12}{11.0592 \text{ MHz}} \times [FFFFh - FF4E + 1]$$

$$= 1.085 \mu\text{sec} \times 178$$

$$\boxed{\text{Time delay} = 193.13 \mu\text{sec}}$$

$$f = \frac{1}{2T} = \frac{1}{2 \times 193.13 \mu\text{sec}}$$

$$\boxed{f = 2.589 \text{ kHz.}}$$



ii) The total time delay is computed as timer delay cycles + 17 machine cycles of the assembly language program.

i.e

$$\text{Timer delay cycles} = \left[\frac{\text{Maximum Value}}{\text{Initial Value}} - 1 \right] = 178.$$
$$= \underline{\underline{178}}$$

$$\therefore \text{Total Time delay} = 178 + 17$$
$$= 195 \text{ Machine cycles}$$

$$T = 195 \times \frac{12}{11.0592 \mu\text{sec}}$$

$$T = 195 \times 1.0850 \times 10^6$$

$$T = 211.575 \mu\text{sec.}$$

$$\therefore f = \frac{1}{2T} = \frac{1}{2 \times 211.575 \mu\text{sec.}}$$

$$f = 2.36 \text{ KHz.}$$

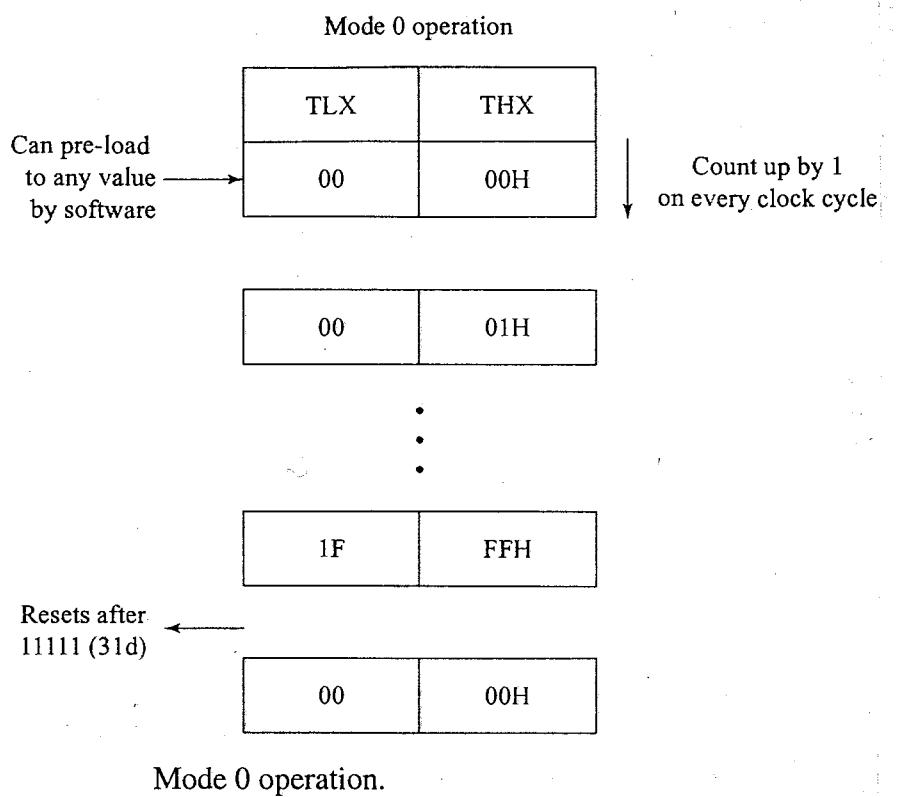
Mode 0 programs :-

- * In mode 0, the timers are used as 13 bit timers. The timer will increment from 0000h to 1FFFh, then it will reset to 0000h as shown below.



(33)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.



NOTE :-

8051 is NOT used in this mode.

- * Generate a square wave with an ON time of 4 msec & an OFF time of 3 msec on pin P3.4. Assume a crystal frequency of 22 MHz. Use Timer 0 in Mode 0.

Sol:- In this example, the ON time & OFF time are different, hence the initial value for ON time & OFF time have to be computed separately.

ON Time initial value computation :-

ON Time required is 4 msec.



34

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

$$(\text{Initial value} - 1) = \frac{\text{Maximum value in.} - \text{Required delay} \times \frac{\text{Crystal freq}}{12}}{\text{Mode 0.}}$$

$$= 1FFFh - \frac{4 \times 10^3 \times 22 \times 10^6}{12}$$

$$= 1FFFh - 7333d$$

$$= 8191 - 7333.$$

Initial value - 1 = 858 d
↓
= 859 d.

Initial value = 35Bh.

* Mode 0 is a 13-bit timer i.e. 5 bits in TH register & 8-bits in TL register.

$$35Bh = \underbrace{00011}_{8\text{-bits}} \underbrace{0101}_{\text{ }} \underbrace{1011}_{5\text{-bits}} B$$

* The lower 5-bits "11011" are located to TLO(1Bh) & upper 8 bits "00011010" to THO(1Ah).

i.e. $\boxed{THO = 1Ah \quad \& \quad TLO = 1Bh.}$

OFF-Time initial value computation :-

OFF-time required = 3msec.



$(\text{Initial value} - 1) = \text{Maximum value in Mode 0.} - \left(\frac{\text{Required delay} \times \text{crystal freq}}{12} \right)$

$$(\text{Initial value} - 1) = 1FFF_h - \frac{3 \times 10^3 \times 22 \times 10^6}{12}$$

$$= 8191 - 5500.$$

$$\begin{aligned} \text{Initial value} - 1. &= 2691 \\ &\quad \curvearrowup \\ &= 2692 \end{aligned}$$

$\therefore \text{Initial value} = A84H.$

- * Mode 0 is a 13-bit timer ie 5-bits in TH & 8 bits in TL register.

$A844 = \underbrace{01010}_{8\text{-bit}} \underbrace{10000100}_{5\text{-bit}} B.$

$TLO = 00000100 B = 04H$

$THO = 01010100 B = 54H$

TMOD register is initialized as.

TIMER 1				TIMER 0			
GATE	C/T	M ₁	M ₀	GATE	C/T	M ₁	M ₀
x	x	x	x	0	0	0	0

TMOD = 00H

↓ ↓

Timer Mode 0
operation



Assembly language program

```
        MOV     TMOD, #00H    ;Timer 0 in mode 0
AGAIN:  MOV     TL0, #01BH   ;Initial value for 4ms in mode 0
        MOV     TH0, #1AH
        SETB    P3.4         ;Make port pin HIGH
        ACALL   DELAY       ;For this delay period (of 4ms),
                            ;port pin is High
        MOV     TL0, #04 H    ;Initial value for 3ms in mode 0
        MOV     TH0, #054H
        CLR     P3.4         ;Make port pin Low
        ACALL   DELAY
        SJMP   AGAIN        ;Repeat loading the initial
                            ;values for continuous
                            ;square wave
DELAY:  SETB    TR0        ;Delay subroutine, start Timer
WAIT:   JNB     TF0, WAIT  ;Wait till timer overflows
        CLR     TR0        ;Clear timer flag and stop timer
        CLR     TF0
        RET
```

Corresponding C program

```
#include <reg51.h>
void delay (void);
sbit outpin=P3^4;
void main()
{TMOD=0x00;                                //Timer 0, mode 0
 while (1)
 {TL0=0x1B;                                 //initial value for 4ms in mode 0
  TH0=0x1A;
  outpin=1;                                  //make port pin High
  delay ();
  TL0=0x04;                                 //initial value for 3ms in mode 0
  TH0=0x54;
  outpin=0;                                  //make port pin Low
  delay();
 }
}
void delay (void)
{TR0=1;                                       //start timer
 while (TF0==1);                            //wait till TF is set
 TR0=0;                                       //stop timer
 TF0=0;                                       //clear TF
}
```

Alternate C program

```
#include <reg51.h>
void delayon (void);
void delayoff (void);
sbit outpin=P3^4;
void main()
{TMOD=0x00;           //Timer 0 & Timer 1 in mode 0
 while (1)
 {outpin=1;           //make port pin High
 delayon ()
 outpin = 0;          //make port pin low
 delayoff ();
 }
}
void delayon (void)
{TL0 = 0x5B;          //Initial value in Timer 0 registers
 TH0 = 0x03;
 TR0 = 1;              //Start Timer 0
 while (TF0 == 1);    //wait till timer 0 overflows
 TR0 = 0;              //stop timer 0
 TF0 = 0;              //clear timer 0 flag
}
void delayoff (void)
{TL1 = 0x84;           //Initial value in Timer 1 registers
 TH1 = 0x0A;
 TR1 = 1;              //start timer 1
 while (TF1 == 1);    //wait till timer 1 overflows
 TR1 = 1;              //stop timer 1
 TF1 = 1;              //clear timer 1 flag
}
```



Generation of time delays in different Modes :-

With initial value = 0000H & XTAL = 11.0592 MHz.

Mode 0 :- (13 bit Timer)

$$\begin{aligned}\text{Maximum time delay} &= 1FFFh \times \frac{12}{\text{crystal freq}} \\ \text{from Timer 0/1 in Mode 0} &= 1FFFh \times \frac{12}{11.0592 \text{ MHz.}} \\ &= 8191 \times 1.085 \times 10^6 \\ &= 8.8 \text{ msec.} \\ &= \underline{\underline{8 \text{ msec.}}}\end{aligned}$$

Mode 1 :- (16-bit Timer)

$$\begin{aligned}\text{Maximum time delay} &= FFFFh \times \frac{12}{11.0592 \text{ MHz.}} \\ \text{from timer 0/1 in mode 1} &= 65535 d \times 1.085 \times 10^6 \\ &= 0.071 \text{ sec.}\end{aligned}$$

Mode 2 :- (8 bit Timer)

$$\begin{aligned}\text{Maximum time delay} &= FFh \times \frac{12}{11.0592 \text{ MHz.}} \\ \text{from Timer 0/1 in Mode 2} &= 255d \times 1.085 \times 10^6 \\ &= 0.278 \text{ msec.} \\ &\approx 0.2 \text{ msec or } 278 \mu\text{sec.}\end{aligned}$$



Maximum Time delay generated by the Timer 0/1 in different modes :-

For crystal frequency • XTAL = 11.0592 MHz.

For Timer 0/1:

Modes	Initial & Maximum value	Maximum Time delay
Mode 0	0000h - 1FFFh.	8.8 msec
Mode 1	0000h - FFFFh	0.071 sec
Mode 2	00h - FFh	276 μsec
Mode 3.	00h - FFh	276 μsec.

Generation of Larger Time delays:-

- * Use Timer 1 in Mode 1 to generate a delay of 2sec. with XTAL = 11.0592 MHz.

Sol:- In Mode 1, the maximum delay can be generated by the timer 1/0 is 0.071 Sec.

so to get higher delays, repeat the timer to get the required time delay.

i.e. 1st program the Timer to generate a delay of 5msec & repeat the timer to get the required delay.

In above example the required delay is 2sec so

$$\frac{2\text{sec}}{5\text{msec}} = 400 \text{ Times.}$$



NOTE :-

- * 1st program the Timer to generate a 5 msec delay.
- * Repeat the number of times to generate the required delay.

i.e.,

$$\text{Required delay} = 5 \text{ msec} \times 400 = 2 \text{ sec.}$$

- * Generate an delay of 50 msec. use Timer 0 in Mode 0 with a crystal frequency of 22MHz.

Sol:

$$\frac{\text{Maximum Time}}{\text{delay in Mode 0}} = \frac{\text{Maximum value of Mode 0}}{\text{crystal freq.}} \times \frac{12}{1}$$

$$= 1FFFh \times \frac{12}{22 \text{ MHz}}$$

$$= \frac{8191 \times 12}{22 \times 10^6}$$

$$\frac{\text{Maximum Time}}{\text{delay in Mode 0.}} = 4.467 \text{ msec.}$$

Now required delay is 50 msec > 4.467 msec.

$$\therefore 1) \frac{50 \text{ msec}}{14} = 3.57 \text{ msec which is less than } 4.467 \text{ msec}$$

(max. delay)

$$2) \frac{50 \text{ msec}}{12} = 4.166 \text{ msec which is less than } 4.467 \text{ msec}$$

(max. delay)



42

In 1st case :- first generate a delay of 4.166 msec.
then repeat the timer (loop) for 12 times to get required delay.

$$12 \times 4.166 \text{ msec} = 50 \text{ msec}$$

In 2nd case :- 1st generate a delay of 4.166 msec.
then repeat the timer (loop) for 12 times to get required delay.

$$12 \times 4.166 \text{ msec} = 50 \text{ msec.}$$

NOTE:-

- * $\frac{50 \text{ msec}}{10} = 5 \text{ msec}$, which is greater than 4.467 msec (max. delay value) care should be taken while generating the smaller delay i.e 1st generated smaller delay must be within the maximum delay generated by the Timer.

$$5 \text{ msec} > 4.467 \text{ msec.}$$



* Toggle P1.5 every 1 second. Use Timer 1 in Mode 1.

Sol: Assume XTAL = 11.0592 MHz.

* WKT the maximum delay that can be generated by the Timer. in Mode 1, with XTAL = 11.0592 MHz is 0.0711 sec.

i.e., 1 sec > 0.0711 sec.

$$*\frac{1\text{sec}}{14} = 0.9954 \text{ sec} \quad \text{or} \quad \frac{1\text{sec}}{0.0711 \text{ sec}} = \underline{\underline{14.0646}}$$

* Repeat the maximum delay 14 times to get a 1 second delay. (The exact delay generated is $14 \times 0.0711 \text{ sec} = 0.9954 \text{ sec}$).

Assembly language program for 1 second delay.

```
        MOV TMOD, #10H ;Timer 1, mode 1
AGAIN:  MOV R0, #14   ;Counter to repeat the delay
        CPL P1.5    ;Toggle pin
AGAIN2: MOV TL1, #00H ;Initial value=00 for maximum delay
        MOV TH1, #00H ;These 2 lines can be skipped
                  ;as timer overflows to 0000
        SETB TR1   ;START timer
WAIT:   JNB TF1, WAIT ;wait until timer1 overflows
        CLR TR1   ;stop timer
        CLR TF1   ;clear timer flag
        DJNZ R0, AGAIN2 ;Decrement counter (R0) & repeat
                  ;till zero
        SJMP AGAIN ;stop
```

NOTE :-

- * To generate the exact 1 second delay, we divide 1sec by 20 to get a 0.05sec.
- * Now, N=20 is the counter value required to repeat the timer delay of 0.05sec.
- * So 1st generate a timer delay of 0.05sec and also calculate the initial value to be loaded into timer to generate a delay of 0.05sec. Then repeat the timer by 20 times.
- * Write a program to generate a pulse train of 50msec on P2.3. use Timer0 in Mode0 with a crystal frequency of 22MHz.

Sol:- The maximum delay possible in Mode0 with a XTAL frequency of 22MHz is

$$\text{Max. delay} = \frac{\text{Maximum value of mode 0}}{\text{crystal freq}} \times \frac{12}{\text{crystal freq}}$$

$$= 1FFF \times \frac{12}{22 \text{ MHz}}$$

$$= 8191 \times \frac{12}{22 \times 10^6}$$

Maximum. delay = 4.467 msec.



45

* Required delay is 50 msec > 4.467 msec.

$$\therefore \frac{50\text{msec}}{14} = 3.57 \text{ msec} < 4.467 \text{ msec.}$$

* Hence counter value = 14 & Timer 0 has to be programmed with an initial value to generate 3.57 msec in mode 0.

Computation of initial value :-

$$[\text{Initial value} - 1] = \text{Maximum value} - \left(\frac{\text{Required delay} \times \text{crystal freq}}{12} \right)$$
$$= 1FFFh - \left(3.57 \times 10^3 \times \frac{22 \times 10^6}{12} \right)$$

$$[\text{Initial value} - 1] = 8191 - 6547$$
$$= 1644 + 1$$
$$= 1645 \text{ d}$$

Initial value = 66 DH.

$$\therefore 66DH = \underbrace{00110}_{8 \text{ bits}} \underbrace{0110}_{5 \text{ bits}} \underbrace{1101}_D \text{ B}$$

$$TL = 00001101 = 0DH$$

$$TH = 00110011 = 33H$$



(46)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Assembly language program

```

        MOV TMOD, #00H ;Timer 0, mode 0
AGAIN:   MOV R0, #14    ;counter to repeat the delay
          CPL P2.3    ;toggle pin to get continuous pulses
AGAIN2:  MOV TL0, #0DH  ;Initial value in Timer registers
          MOV TH0, #33H ;
          SETB TR0     ;Start timer
Wait:    JNB TF0, Wait
          CLR TR0
          CLR TF0
          DJNZ R0, AGAIN2
          SJMP AGAIN

```

The corresponding C program is given below.

```

#include <reg51.h>
Sbit mypin=P2^3;
void main()
{unsigned char i;                                //counter 'i'
while(1)
    {mypin=~mypin;                            //toggle port pin
     for (i=0; i<14; i++)
         {TMOD=0x00;                          //timer 0, mode 0
          TL0=0x0D;                           //initial value in timer
          TH0=0x33;
          TR0=1;                             //start timer
          while (TF0==0);                    //wait till TF0 becomes 1
                                              //((while is true when TF0 =0)
          TR=0;                               //stop timer
          TF0=0 ;                            //clear timer flag
         }
    }
}

```

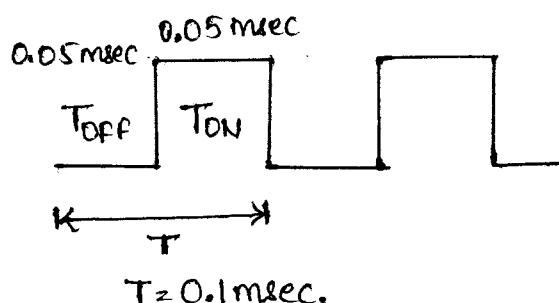
Mode 2 : programs :-

- 1) Write a program to generate a square wave of frequency 10KHz. on P1.4. Use Timer 0 in Mode 2 with a crystal frequency of 22MHz.

Sol:- Given :

$$\text{frequency} = 10\text{KHz}$$

$$\text{XTAL} = 22\text{MHz}$$



$$\text{W.K.T, } T = \frac{1}{f} = \frac{1}{10\text{KHz}} = 0.1\text{msec.}$$

$$\therefore T = T_{ON} + T_{OFF} = 0.1\text{msec}$$

$$T_{ON} = T_{OFF} = \frac{0.1\text{msec}}{2} = 0.05\text{msec}$$

(50% duty cycle)

Hence Required delay = 0.05msec.

Initial value computation :

$$\begin{aligned} [\text{Initial value - 1}] &= \text{Maximum value of mode 2} - \left(\frac{\text{Required delay} \times \text{crystal freq}}{12} \right) \\ &= (\text{FFh}) - \left(0.05 \times 10^3 \times \frac{22 \times 10^6}{12} \right) \\ &= (255) - 91.6 \end{aligned}$$

$$[\text{Initial value - 1}] = 163.33.$$



48

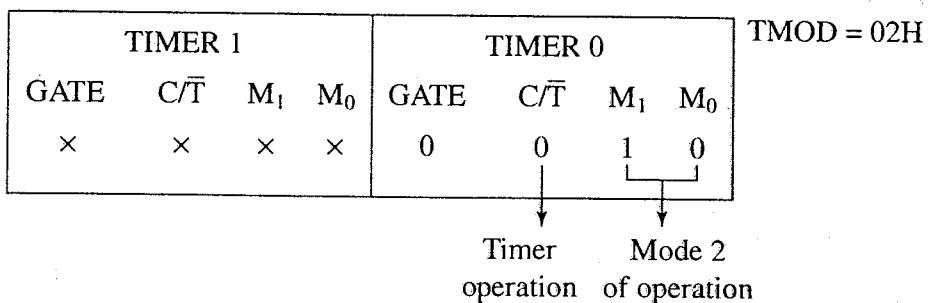
$$\begin{aligned}\text{Initial value} - 1 &= 163.33 \\ &= 163 + 1 \\ &= 164 \text{ d}\end{aligned}$$

Initial value = A4H

$$\therefore \text{THO} = \text{A4H}$$

NOTE :- In program the initial value is loaded only once into the TH register.

TMOD register initialization.



Assembly Language Program

```
MOV TMOD, #02H ;Timer 0 in mode 2
MOV TH0, #A4H ;initial value
SETB TR0 ;start timer 0
Wait: JNB TF0, Wait ;wait until timer 0 overflows
      CPL P1.4 ;complement pin (i.e., toggle make low
                  ;as high & vice versa)
      CLR TF0 ;Clear timer flag
      SJMP Wait ;repeat
```



49

Corresponding C program

```
#include <reg51.h>
sbit outpin=P1^4;
void main()
{
    TMOD=0x02;           //initialize timer 0 in mode 2
    TH0=0xA4;            //load initial value
    TR0=1;                //start timer
    while(1)
    {
        outpin=~outpin;   //toggle outpin
        while (TF0==0);   //wait until timer overflow
        TF0=0;              //clear timer flag
    }                      //end of while loop
}                      //end of main
```

- Q) Write a program to toggle all bits of P2 continuously every 50msec. use Timer 1, mode 2 with a crystal frequency of 11.0592 MHz to compute the delay.

Sol: The Maximum delay possible in Mode 2 is :

$$\begin{aligned}\text{Maximum delay} &= \frac{\text{Maximum value of } \times \frac{12}{\text{crystal freq}}}{\text{Mode 2}} \\ &= \text{FFh} \times \frac{12}{11.0592 \text{MHz.}} \\ &= 255 \times 1.085 \mu\text{sec.}\end{aligned}$$

$$\begin{aligned}\text{Maximum delay} &= 276.6 \mu\text{sec} = 0.276 \text{ msec.}\end{aligned}$$



$$\therefore \frac{50 \text{ msec}}{200} = 0.25 \text{ msec.}$$

i.e. $0.25 \text{ msec} < 0.276 \text{ msec.}$

- * 1st generate a smaller delay i.e. 0.25 msec, then repeat the timer for 200 times to get a required delay.

$$\therefore \boxed{\text{Counter} = 200}$$

Computation of Initial Value :-

$$\left[\begin{array}{c} \text{Initial} \\ \text{value} \end{array} - 1 \right] = \left[\begin{array}{c} \text{Maximum value} \\ \text{of Mode 2} \end{array} \right] - \left(\begin{array}{c} \text{Required} \\ \text{delay} \times \frac{\text{Crystal freq}}{12} \end{array} \right)$$

$$= (\text{FFh}) - \left(0.25 \times 10^3 \times \frac{11.059 \times 10^6}{12} \right)$$

$$\left[\begin{array}{c} \text{Initial} \\ \text{value} \end{array} - 1 \right] = 255 - 230.4$$
$$= 25 + 1$$

$$\boxed{\text{Initial value} = 1AH.}$$



Assembly language program

```
MOV TMOD, #20H ;Timer 1 in mode 2
MOV TH1, #1AH ;Initial value to generate 0.25ms
;in mode 2 (1AH)
AGAIN: MOV P2, #00H ;make all bits of port P2 = 00
; (NOTE: This can be any number,
;say 55 H, we just have to toggle the
;bits, i.e., get their complement)
SETB TR1 ;start timer
MOV R0, #200 ;load R0 with the count to get 50ms
Wait: JNB TF1, wait ;wait till timer overflows
CLR TF1 ;clear timer flag
DJNZ R0, wait ;repeat for 50ms
CLR TR1 ;stop timer
MOV P2, #FFH ;make all bits of port P2 = FF
MOV R0, #200 ;load R0 again for 50 ms delay
SETB TR1
Wait1: JNB TF1, Wait1,
CLR TF1
DJNZ R0, Wait1
CLR TR1
SJMP AGAIN
```

C Program to toggle all bits of P2 every 50ms

```
#include <reg51.h>
void delay1 (void);
void main()
{TMOD=0×20 ; //Timer 1 in mode 2
TH1=0×19 ; //Initial value – *
P2=0×58 ; //some value on port P2, which has to
//be toggled
while(1) //repeat forever
{P2=~P2 ; //toggle port pins
delay1(); //call delay
}
}
void delay1()
{unsigned char i;
for (i=0; i<200; i++) //repeat 200 times to get 50ms
{TR1=1; //start timer
while(TF1==0); //wait till timer overflow
TR1=0; //stop timer & clear timer flag
TF1=0;
}
}
//end of for & delay subroutine
```

9.10. Assume that XTAL = 11.0592 MHz What value do we need to load into the timer's registers if we want to have a time delay of 5 ms? Show the program for timer 0 to create a pulse width of 5 ms on P2.3.

Sol :-

Timer 0 : Mode 1 \rightarrow 16 bit timer.

$$\begin{aligned} \left[\text{Initial value} - 1 \right] &= \text{Maximum Value in mode 1} - \left[\frac{\text{Time delay} \times \text{crystal freq}}{12} \right] \\ &= \text{FFFF(h)} - \left[5 \times 10^3 \times \frac{11.0592 \text{MHz.}}{12} \right] \end{aligned}$$

$$\left[\text{Initial value} - 1 \right] = 65535(\text{d}) - 4608$$

$$\begin{aligned} \text{Initial value} &= 65535(\text{d}) - 4608 + 1 \\ &= 60928(\text{d}) \end{aligned}$$

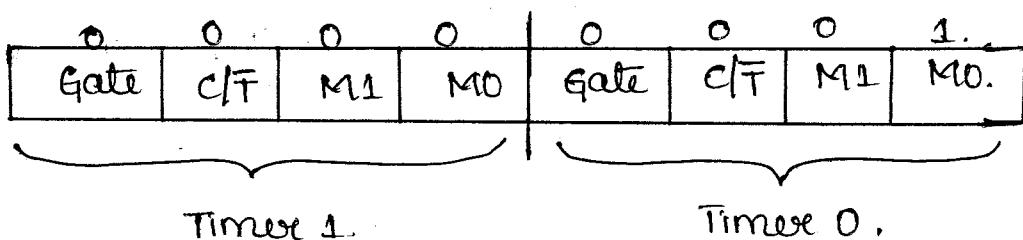
$$\text{Initial value} = 60928(\text{d})$$

$$\boxed{\text{Initial value} = \text{EE00h}}$$

$$\therefore \text{THOTLO} = \text{EE00h.}$$

$$\text{ie. } \text{THO} = \text{EEh.}$$

$$\text{TLO} = \text{00h.}$$



$$\boxed{\text{TMOD} = 01h}$$



(53)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

```
ORG 00H  
CLR P2.3      ; CLEAR P2.3  
MOV TMOD, #01H ; TIMERO, MODE 1(16-BIT TIMER)
```

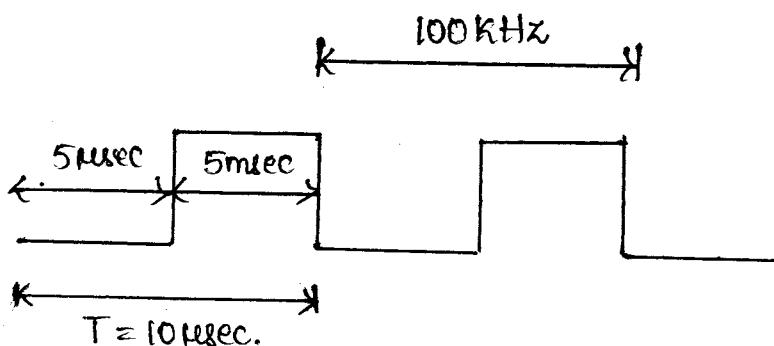
HERE:

```
MOV TL0, #00H      ; TL0 = 00H, LOW BYTE  
MOV TH0, #0EEH      ; TH0 = EEH, HIGH BYTE  
SETB P2.3          ; SET P2.3 HIGH  
SETB TR0            ; START TIMER  
AGAIN:          ; MONITOR TIMER 0 FLAG UNTIL  
JNB TF0, AGAIN      ; IT ROLLS OVER  
CLR P2.3          ; CLEAR P2.3  
CLR TR0            ; STOP TIMER  
CLR TF0            ; CLEAR TIMER 0 FLAG  
SJMP HERE  
END
```

9.11) with a frequency of 22MHz, generate a frequency of 100 KHz on pin

P2.3. Use Timer 1 in mode 1

Sol :-



$$W.K.T \cdot T = \frac{1}{F} = \frac{1}{100\text{KHz}}$$

$$T = 10 \mu\text{sec}$$

$$T = T_{ON} + T_{OFF}$$

$$T = 5 \mu\text{sec} + 5 \mu\text{sec}$$

$$(Initial \text{ value} - 1) = \frac{\text{Max Value}}{\text{in mode 1}} - \left(\frac{\text{Required Time delay} \times \text{crystal freq}}{12} \right)$$

$$= FFFF \text{ h} - \left(5 \mu\text{sec} \times \frac{22 \text{MHz}}{12} \right)$$

$$\begin{aligned} \text{Initial value} - 1 &= 65535 \text{ (d)} - 9166 \text{ (d)} \\ &= 65526 \text{ (d)} + 1 \\ &= 65527 \text{ (d)}. \end{aligned}$$

$$\boxed{\text{Initial value} = FFF7 \text{ (h)}}$$

$$\Rightarrow TH1TL1 = FFFF$$

$$\boxed{\begin{aligned} \text{ie } TH1 &= FF \\ TL1 &= FF \end{aligned}}$$



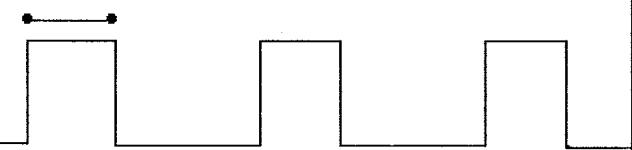
ORG 00H

```
MOV TMOD, #10H      ; Timer 1, mode 1
BACK:  MOV TL1, #0F7H    ; TL1=F7H
       MOV TH1, #0FFH    ; TH1=FFH
       SETB TR1        ; start Timer
AGAIN: JNB TF1, AGAIN   ; wait for timer rollover
       CLR TR1        ; stop Timer 1
       CPL P2.3       ; complement p2.3
       CLR TF1        ; clear timer flag
       SJMP BACK      ; reload Timer
END
```

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

Sol :- In mode 1, Maximum possible 3 ms

delay is given by



$$\text{Max possible time delay in Mode 1} = \frac{\text{Max Value in Mode 1}}{\text{crystal freq}} \times \frac{12}{12}$$

10 ms

12

crystal freq

$$= \text{FFFF(h)} \times \frac{12}{22 \text{ MHz}} = 0.0357 \text{ sec.}$$

\therefore Maximum possible time = 35.7463 μsec.

delay in Mode 1

i) ON - Time Initial value computation :-

$$\begin{aligned} (\text{Initial value} - 1) &= \text{Max value in mode 1} - \left(\frac{\text{Required time delay} \times \text{crystal freq}}{12} \right) \\ &= \text{FFFF(h)} - \left(3 \text{ msec} \times \frac{22 \text{ MHz}}{12} \right) \\ &= 65535 - (5500) + 1 = 60036(\text{d}). \end{aligned}$$

Initial value = EA84(h).

ii) OFF - Time Initial value computation :-

$$\begin{aligned} (\text{Initial value} - 1) &= \text{Max. value in Mode 1} - \left(\frac{\text{Required time delay} \times \text{crystal freq}}{12} \right) \\ &= \text{FFFF(h)} - \left(10 \text{ msec} \times \frac{22 \text{ MHz}}{12} \right) \\ &= 65,535 - 18333 + 1 \end{aligned}$$

Initial value = 47203(d). = B863(h).

* Use Timer 0 : Mode 1 :-

$$TR0D = 01h.$$



ORG 00H

MOV TMOD, #01H ; Timer 0 in mode 1

BACK: MOV TL0, #63H ; to generate the OFF time, load TL0

MOV TH0, #0B8H ; load OFF time value in TH0

MOV P0, #00H ; make port bits low

ACALL **DELAY** ; call delay routine

MOV TL0, #84H ; to generate the ON time, load TLO

MOV TH0, #0EAH ; load ON time value in TH0

MOV P0, #0FFH ; make port bite high

ACALL **DELAY** ; call delay

SJMP **BACK** ; repeat for reloading counters to get a
; Continuous square wave

DELAY: SETB TR0 ; start counter

AGAIN: JNB TF0, AGAIN ; check time over flow

CLR TR0 ; when TF0 is set, stops the timer

CLR TF0 ; clear time flag

RET

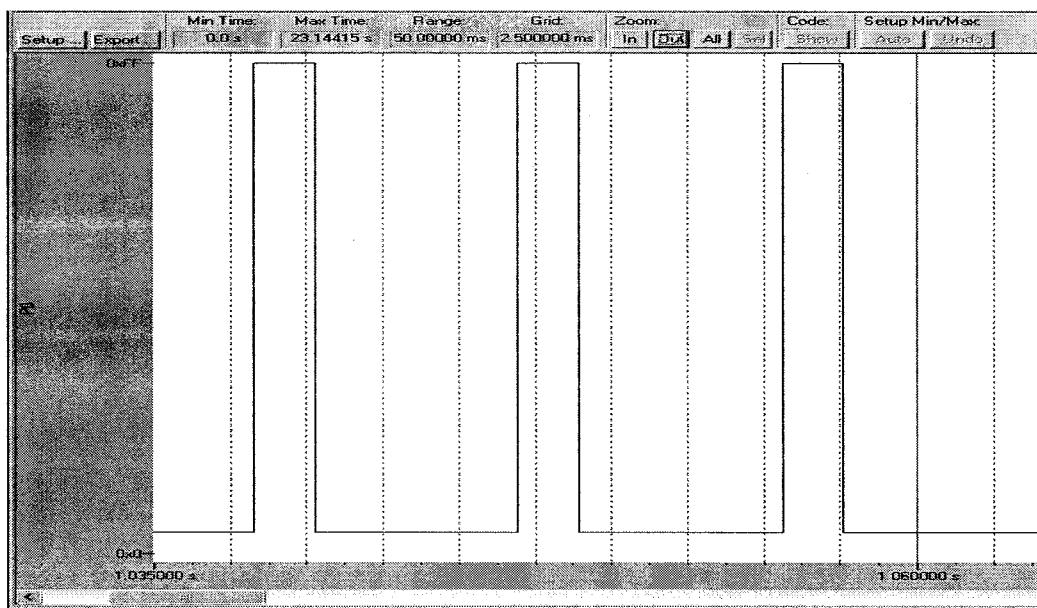
END ; end of the file



58

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Result:



59

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

9.13 Assuming XTAL=22MHz, write a program to generate a pulse train of 2 seconds period on pin P2.4.use Timer 1 in Mode 1.

Sol :- * Maximum delay in Mode 1 with XTAL=22MHz is.

$$\text{Maximum Time delay} = \frac{12}{\text{crystal freq.}} \quad (\text{Maximum Value in Mode 1})$$

$$= \frac{12}{22 \text{ MHz}} \times \text{FFFF(h)}$$

$$= \frac{12}{22 \text{ MHz}} \times 65535(\text{d}) = 0.035746 \text{ sec.}$$

$$\begin{array}{lcl} \text{Max. Time} & = & 35.746 \text{ msec.} \\ \text{delay in Mode 1} & & \end{array}$$

* To get the required time delay of 1sec

$$\text{i.e., } \frac{1 \text{ sec}}{35.746 \text{ msec}} = 27.9748.$$

Or

$$\text{Counter 'N'} = \frac{\text{Required Time delay}}{\text{Generated Time delay.}} = \frac{1 \text{ sec.}}{35.746 \text{ msec}}$$

in Mode 1

$$= 27.9748$$

$$\text{Counter 'N'} = 28$$

* If we repeat the maximum time delay by 28 times then we will get a delay of 1sec.



ORG 00H

```
MOV TMOD, #10H ; Timer 1, mode 1
REPT: MOV R0, #28 ; counter for multiple delays

CPL P2.4 ; complement p2.4
BACK: MOV TL1, #00H ; load count value in TL1

MOV TH1, #00H ; load count value in TH1

SETB TR1 ; start timer

AGAIN: JNB TF1, AGAIN ; stay until timer rolls over

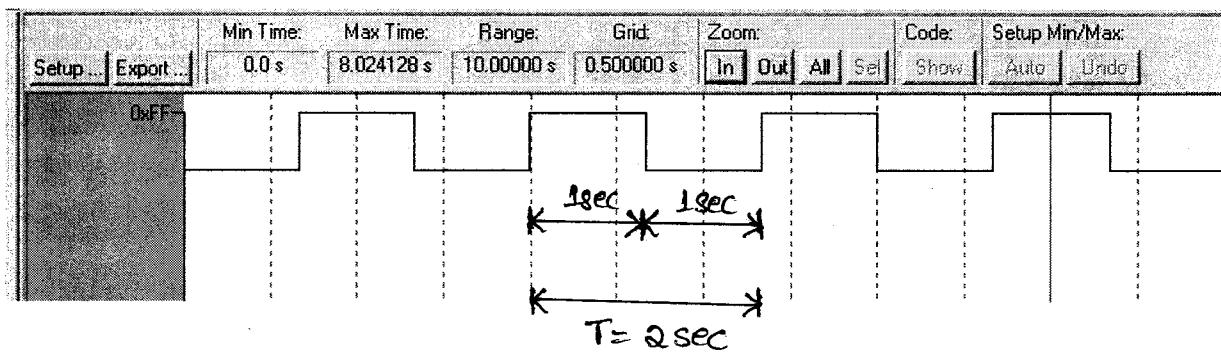
CLR TR1 ; stop timer

CLR TF1 ; clear timer flag

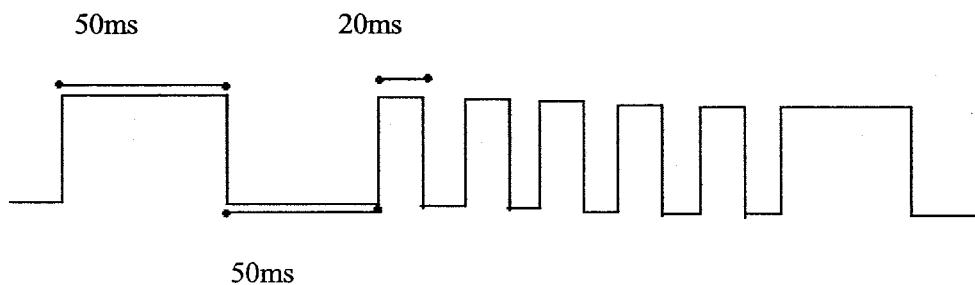
DJNZ R0, BACK ; if R0 is not zero, reload timer

SJMP REPT ; repeat for continuous pulse generation

END
```



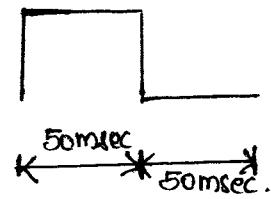
9-13a Generate the following waveform on P1.2. XTAL=22MHz



Sol:- In above waveform, 1st it has 50msec ON time & 50msec OFF time, followed by Five repetitions of 10msec ON time & 10msec OFF time.

To generate 50msec delay,

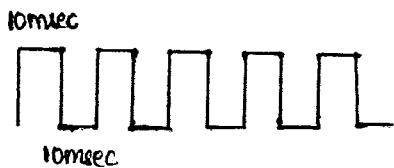
- * 1st generate a delay of 10msec & repeat 10msec five times i.e $10\text{msec} \times 5 = 50\text{msec}$.
- 1) First set P1.2=1, then generate a delay of 50msec.
- 2) Second set P1.2=0, then generate a delay of 50msec.



- * Now set P1.2=1 & generate a delay of 10msec & again set P1.2=0, generate a delay of 10msec.

Repeat this process for 5 times

to get above waveform.



Computation of Initial value to generate a delay of 10ms:

$$(\text{Initial value} - 1) = \text{Max value in Mode 1} - \left[\frac{\text{Required Time delay} \times \text{Crystal freq}}{12} \right]$$

$$\begin{aligned} \text{Initial value} - 1 &= \text{FFFF(h)} - \left(10\text{ms} \times \frac{22\text{MHz}}{12} \right) \\ &= 65535 - 18333.33 + 1 \\ &= 47203(d). \end{aligned}$$

Initial value = B863(h). i.e \downarrow $\text{TH1} = 88(h)$ & $\text{TMOD} = 10h$.
 $\text{TL1} = 63(h)$. Timer 1: Mode 1

ORG 00H

```

START: SETB P1.5           ; set port pin for generating the ON time
        MOV R1, #02          ; one ON time and one OFF time
        MOV TMOD, #10H         ; timer 1, mode 1
BACK:  MOV R0, #05          ; repeat the 10 ms delay 5 times
RPT:   MOV TL1, #75H
        MOV TH1, #0B8H
        SETB TR1
AGAIN: JNB TF1, AGAIN
        CLR TR1
        CLR TF1
        DJNZ R0, RPT
        CPL P1.5
        DJNZ R1, BACK          ; stop after 2 periods of 50 ms are over
    
```



(63)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

```
SETB P1.5 ; start the sequence of 5 pulses of T=20 ms  
MOV R1, #10 ; this part for 10 ON-OFF periods of 10 ms  
BACK1: MOV TL1, #75H  
MOV TH1, #0B8H  
SETB TR1  
AGAIN1: JNB TF1, AGAIN1  
CLR TR1  
CPL P1.5  
CLR TF1  
DJNZ R1, BACK1  
SJMP START ; repeat  
END
```

9.14 Assuming that XTAL=11.0592MHz, find (a) the frequency of the square wave generated on pin P1.0 in the following program, and (b) the smallest frequency achievable in this program ,and the TH value to do that.

```

ORG 00H

MOV TMOD, #20H      ; T1/mode 2/8-bit/auto-reload

MOV TH1, #5          ; TH1=5

SETB TR1             ; start Timer 1

BACK: JNB TF1, BACK  ; stay until timer rolls over

CPL P1.0              ; comp. P1.0 to get hi, lo

SJMP BACK            ; mode 2 is auto-reload

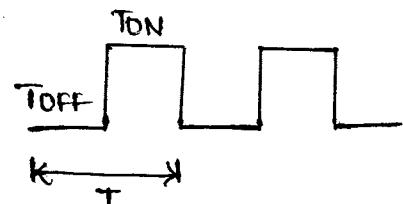
END

```

Sol:-

* In Mode 2, we do not need to reload TH since it is auto-reload. From above program, Initial value = 05h.

* Since this program generates a 50% duty cycle square wave.



Formula:-

$$\begin{aligned}
 \text{Time delay} &= \frac{12}{\text{crystal freq}} - \left[\frac{\text{Max. Value}}{\text{in Mode 2.}} - \frac{\text{Initial Value}}{} \right] \\
 &= \frac{12}{11.0592 \text{ MHz}} - [FF(h) - 05]
 \end{aligned}$$



(65)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

$$= \frac{12}{11.0592 \text{ MHz}} [255 - 05]$$

$$= 1.085069 \times 10^6 [250]$$

Time delay = 271.26 μsec.

* From program WKT : $T_{ON} + T_{OFF} = 271.26 \mu\text{sec}$.

$$T = T_{ON} + T_{OFF} = 271.26 \mu\text{sec} + 271.26 \mu\text{sec}$$

$$T = 544.67 \mu\text{sec.}$$

$$f = \frac{1}{T} = \frac{1}{544.67 \mu\text{sec}} = \underline{\underline{1.83597 \text{ kHz}}}$$

b) To get smallest frequency, we need the largest time delay 'T'.

* Maximum Time delay can be achieved when initial value = 00h.

Formula:- Maximum Time delay = $\frac{12}{\text{crystal freq}}$ [Maximum Value in Mode 2]

$$= \frac{12}{11.0592 \text{ MHz}} [\text{FF(h)}]$$

$$= 1.085069 \times 10^6 (255 d)$$

Max. Time delay = 276.85092 μsec.

W.K.T $T = T_{ON} + T_{OFF}$
 $= 276.85 \mu\text{sec} + 276.85 \mu\text{sec}$
 $= 553.7019 \mu\text{sec}$

$$f = \frac{1}{T} = \frac{1}{553.7019 \mu\text{sec}} = \underline{\underline{1.806025 \times 10^3 \text{ Hz.}}}$$

* This is the smallest frequency that can be generated in Mode 2.

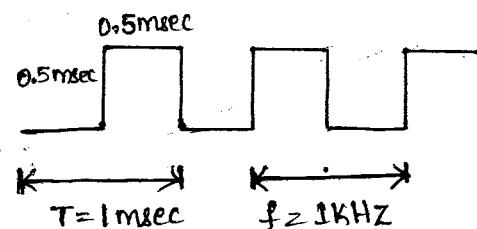


9.15 Assuming that XTAL=22MHz, write a program to generate a square wave of frequency 1 KHz on pin P1.2.

Sol:- * Use Mode 2:

XTAL = 22MHz, $f = 1\text{ KHz}$ on P1.2 pin.

$$\text{W.K.T} \Rightarrow T = \frac{1}{f} = \frac{1}{1\text{ KHz}} \quad \boxed{T = 1\text{ msec}}$$



$$T = T_{ON} + T_{OFF} = 1\text{ msec}$$

$$\therefore T_{ON} = T_{OFF} = 0.5\text{ msec}$$

* Maximum possible delay in Mode 2

$$= \text{FF(h)} \times \frac{12}{\text{crystal freq}}$$

$$= 255(d) \times \frac{12}{22\text{ KHz.}} = 0.1390909\text{ msec}$$

But required delay is 0.5msec.

* So 1st generate a delay of 0.1msec & repeat the loop for 5 times

$$\text{i.e., } 0.1\text{ msec} \times 5 = 0.5\text{ msec}$$

$$\text{counter 'N'} = \frac{0.5\text{ msec}}{0.1\text{ msec}} = 5.$$

Computation of Initial value for 0.1 msec:-

$$\left(\text{Initial value} - 1 \right) = \text{Max. value in Mode 2} - \left[\frac{\text{Required time delay} \times \text{crystal freq}}{12} \right]$$

$$= \text{FF(h)} - \left[0.1\text{ msec} \times \frac{22\text{ MHz}}{12} \right]$$

$$\left(\text{Initial value} - 1 \right) = 255(d) - [183.33]$$

$$= 72(d) + 1 = 73(d)$$

$$\boxed{\text{Initial value} = 49(h).}$$



67

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

ORG OOH

MOV TMOD, #02H ; Timer 0, mode 2

REPT: CPL P1.2 ; complement P1.2

MOV R0, #05 ; count for multiple delays

AGAIN: MOV TH0, #48H ; load TH0 value

SETB TR0 ; start Timer 0

BACK: JNB TF0, **BACK** ; stay until timer rolls over

CLR TR0 ; stop timer

CLR TF0 ; clear timer flag

DJNZ R0, **AGAIN** ; repeat until R0=0

SJMP **REPT** ; repeat to get a train of pulses

END

9-17 find (a) the frequency of the square wave generated in the following code, and (b) the duty cycle of this wave

ORG 00H

MOV THOD, #02H ; Timer 0, mode 2

; (8-bit, auto-reload)

MOV TH0, #-150 ; TH0=6AH=2'S comp of -150

AGAIN: SETB P1.3 ; P1.3=1

ACALL **DELAY**

ACALL **DELAY**

CLR P1.3 ; P1.3

ACALL **DELAY**

SJMP **AGAIN**

DELAY:

SETB TR0 ; start Timer roller over



BACK:	JNB TF0, BACK	; stay until timer rolls over
	CLR TR0	; stop timer 0
	CLR TF0	; clear TF for next round
	RET	
	END	

- Sol:-
- * In this program, Initial value = -150(d). If we convert this negative number, we will get this equivalent hex value. = 6A(h).
 - * 6A(h) is the initial value of TWO register
 $6A(h) = 106(d)$

Formula:

$$\begin{aligned}\text{Required time delay} &= \frac{12}{\text{crystal freq}} \left(\frac{\text{final value} - \text{initial value}}{255(d) - 106(d)} + 1 \right) \\ &= \frac{12}{11.0592 \text{ MHz}} [255(d) - 106(d) + 1] \\ &= \frac{12}{11.0592 \text{ MHz}} [148]\end{aligned}$$

$$\begin{aligned}\text{Required Time delay} &= 160.5902 \text{ μsec.}\end{aligned}$$

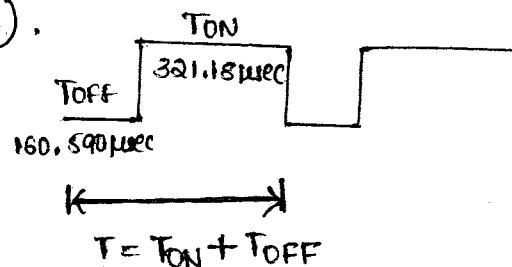


* In this program TON is Twice that of TOFF.

$$\text{ie } \text{TOFF} = 160.5902 \mu\text{sec}$$

$$\text{TON} = 2(160.5902 \mu\text{sec})$$

$$\text{TON} = 321.18 \mu\text{sec}$$



$$\therefore T = \text{TON} + \text{TOFF}$$

$$= 321.18 \mu\text{sec} + 160.590 \mu\text{sec}$$

$$T = 481.77 \mu\text{sec}$$

frequency ,

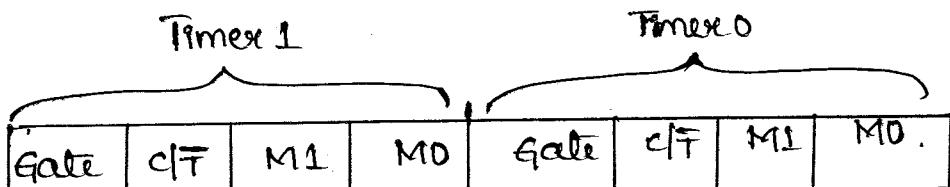
$$f = \frac{1}{T} = \frac{1}{481.77 \mu\text{sec}}$$

$$f = 2.0756 \text{ kHz}$$



9-18 Design a counter for counting the pulses of an input signal. The pulses to be counted are fed to pin P3.4. XTAL=22MHz.

Sol:-



$$TMOD = 15(h).$$

* Timer 1 \rightarrow Mode 1

$C/T = 1 \rightarrow$ counter

* Timer 0 is used as counter in Mode 1.

$$\text{Maximum possible time delay in Mode 1} = \frac{\text{Max value in mode 1}}{\text{crystal freq.}} \times \frac{12}{\text{crystal freq.}}$$

$$= FFFF(h) \times \frac{12}{22 \times 10^6}$$

$$= 65535(d) \times 0.545454 \times 10^{-6}$$

$$\begin{aligned} \text{Maximum possible time delay in Mode 1} &= 35.74636 \text{ msec.} \\ \text{Required time delay} &= 1 \text{ sec.} \end{aligned}$$

$$\therefore \text{Counter 'N'} = \frac{1 \text{ sec}}{\text{Max. time delay}} = \frac{1 \text{ sec}}{35.746 \text{ msec}} \\ = 27.77$$

$$\text{Counter 'N'} = 28.$$

- * 1st generate a delay of 35.746 msec.
- * Repeat the generated time delay by 28 times to get a delay of 1 sec i.e $35.746 \text{ msec} \times 28 = 1 \text{ sec.}$



71

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

- * Timer 0 is used as a counter, with Ifp pulses fed into Pin P3.4. At the end of 1 second, the values in TL0 & TH0 gives the number of pulses that were received at Pin P3.4 during this 1 second.

ORG 0000H

RPT: MOV TMOD, #15H ; timer 1 as timer and Timer 0 as counter
SETB P3.4 ; make port p3.4 an input port
MOV TL0, #00 ; clear TL0
MOV TH0, #00 ; clear TH0
SETB TR0 ; start counter
MOV R0, #28 ; R0=28, to timer 1 second

AGAIN: MOV TL1, #00H ; TL1=0
MOV TH1, #00H ; TH1=0
SETB TR1 ; start timer 1

BACK: JNB TF1, BACK ; test timer 1 flag
CLR TF1 ; clear timer 1 flag
CLR TR1 ; stop timer 1
DJNZ R0, AGAIN ; repeat the loop until R0=0
MOV A, TL0 ; since 1 sec has elapsed, check TL0
MOV P2, A ; move TL0 to port 2
MOV A, TH0 ; move TH0 to ACC
MOV P1, A ; move it to port 1
SJMP RPT ; repeat

END



9-19 assume that a 1Hz frequency pulse is connected to input pin

3.4. Write a program to display counter 0 on an LCD. Set the initial value of TH0 to -60

ORG 00H

ACALL LCD_SET_UP ; initialize the LCD
MOV TMOD, #00000110B ; counter 0, mode 2, C/T=1
MOV TH0, #-60 ; counting 60 pulses
SETB P3.4 ; make T0 as input
AGAIN: SETB TR0 ; start the counter
BACK: MOV A, TL0 ; get copy of count TL0
ACALL CONV ; convert in R2, R3, R4
ACALL DISPLAY ; display on LCD
BACK: JNB TF0, BACK ; loop if TF0=0
CLR TR0 ; stop the counter 0
CLR TF0 ; make TF0=0
SJMP AGAIN ; keep doing it

; converting 8-bit binary to ASCII

; Upon return, R4, R3, R2 have ASCII data (R2 has LSD)

CON: MOV B, #10 ; divide by 10
DIV AB
MOV R2, B ; save low digit
MOV B, #10 ; divide by 10 once more
DIV AB
ORL A, #30H ; make it ASCII
MOV R4, A ; save MSD
MOV A, B



```
ORL A, #30H  
MOV R4, A  
MOV A, B  
ORL A, #30H ; make 2nd digit an ASCII  
MOV R3, A ; save it  
MOV A, R2  
ORL A, #30H ; make 3rd digit ASCII  
MOV R2, A ; save the ASCII  
RET  
END
```

9,20) write a 8051 c program to toggle all the bits of port P1 continuously with initial value of 3500h. Use Timer 0, 16-bit mode to generate the delay.

Initial value = 3500h.

Sol:-

$$\begin{aligned} \text{Time delay} &= \frac{12}{\text{crystal freq}} \left[\frac{\text{Max Value}}{\text{in Mode 1}} - \frac{\text{Initial Value}}{} + 1 \right] \\ &= \frac{12}{11.0592 \text{ MHz}} \left[\text{FFFF (h)} - 3500 \text{h} + 1 \right] \\ &= 1.08506 \times 10^6 [\text{CAFF h}] \\ &\approx 1.08506 \times 10^6 \times 51967 (\text{d}). \end{aligned}$$

Time delay = 56.4202 msec.

TMOD = 01h.

Assume :-

XTAL = 11.0592 MHz

```
#include <reg51.h>
void delay (void);
void main(void)
{
    while(1) //repeat forever
    {
        P1=0x55; //toggle all bits of P1
        delay(); //delay size unknown
        P1=0xAA; //toggle all bits of P1
        delay();
    }
}
```



```
void delay ()  
{  
    TMOD=0x01;           //Timer 0, Mode 1  
    TL0=0x00;             //load TL0  
    TH0=0x35;             //load TH0  
    TR0=1;                //turn on T0  
    while (TF0==0);       //wait for TF0 to roll over  
    TR0=0;                //turn off T0  
    TF0=0;                //clear TF0  
}
```

9.21 write an 8051 c program to toggle only bit P1.5 continuously every 50ms. Use Timer 0, mode 1 (16-bit) to create the delay. Test the program (a) on the AT89C51 and (b) on the ds89c420.

Assume XTAL = 11.0592 MHz

$$\begin{aligned}\text{Initial value} - 1 &= \frac{\text{Max value}}{\text{in Mode 1}} - \left[\frac{\text{Required time delay}}{\text{crystal freq}} \right] \\ &= \text{FFFF h} - \left[50\text{msec} \times \frac{11.0592 \times 10^6}{12} \right] \\ &= \text{FFFF h} - 46080 \text{ d} \\ &= 19455 \text{ d} + 1 \\ &= 19456 \text{ d}.\end{aligned}$$

Initial value = 4000h.

TL0 = 00h
TH0 = 4ch.



```
#include <reg51.h>
void delay(void);
Sbit mybit=P1^5;
void main(void)
{
    while(1)
    {
        mybit=~mybit;           //toggle P1.5
        delay();                //Timer 0,mode 1 (16-bit)
    }
}
```

(a) Tested for AT89C51,XTAL=11.0592 MHz, using the proview32 compiler

```
void delay(void)
{
    TMOD=0x01;           //Timer 0, mode 1 (16-bit)
    TL0=0xFD;            //load TL0
    TH0=0x4B;            //load TH0
    TR0=1;               //turn on T0
    while (TF0==0);      //wait for TF0 to roll over
    TR0=0;               //turn off T0
    TF0=0;               //clear TF0
}
```

(B) Tested for DS89C420, XTAL=11.0592MHz, using the Proview32 compiler

```
void delay(void)
{
    TMOD=0x01;           //Timer 0, mode 1 (16-bit)
    TL0=0xFD;            //load TL0
```



```

TH0=0x4B;           //load TH0
TR0=1;             //turn on T0
while (TF0==);     //wait for TF0 to roll over
TR0=0;             //turn off T0
TF0=0;             //Clear TF0
}

```

9.22) write an 8051 c program to toggle all bits of P2 continuously every 500ms. Use Timer 1, mode 1 to create the delay.

Sol:- Maximum possible Time delay in Mode 1 is $\approx 1.110 \text{ msec}$.

$$\text{Case 1: counter 'N' } = \frac{500 \text{ msec}}{50 \text{ msec}} = 10.$$

$$\begin{aligned}
[\text{Initial value} - 1] &= \text{Max. value in mode 1} - \left[\frac{\text{Required Time delay} \times \text{Crystal freq}}{12} \right] \\
&= \text{FFFF(h)} - \left[50 \text{ msec} \times \frac{11.0592 \times 10^6}{12} \right] \\
&= 65535(d) - 46080(d) \\
&= 19455(d) + 1
\end{aligned}$$

i.e. $\text{TH1} = 00h$

$\text{TL1} = 4ch$ &

$\text{TMOD} = 01h$.

Case 2:-

$$\text{counter 'N' } = \frac{500 \text{ msec}}{25 \text{ msec}} = 20$$

$$\begin{aligned}
[\text{Initial value} - 1] &= \text{Max. Value in mode 1} - \left[\frac{\text{Required Time delay} \times \text{Crystal freq}}{12} \right]
\end{aligned}$$



(48)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

$$= \text{FFFF(h)} - \left[25 \text{ msec} \times \frac{11.0592 \times 10^6}{12} \right]$$

$$= 65535(\text{d}) - 23040(\text{d}) + 1$$

$$= 42496 \text{ d.}$$

Initial value = A600h.

i.e TL1 = 00h.

TH1 = A6h.

- * In this program we are using case 1.
i.e, 1st generating time delay of 50msec &
repeating the process for 10 times.

$$\underline{10 \times 50 \text{ msec} = 500 \text{ msec.}}$$

//tested for DS89C420,XTAL=11.0592MHz,using the Proview32 complier

```
#include <reg51.h>
void delay(void);
void main(void)
{
    unsigned char X;
    P2=0x55;
    while(1)
    {
        P2=~P2; //Toggle all bits of P2
        for(x=0; x<20; x++)
            delay();
    }
}
void delay(void)
```



79

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

```
{  
    TMOD=0x10;           //Timer 1, mode 1 (16-bit)  
    TL1=0xFE;            //load TL1  
    TH1=0xA5;            //load TH1  
    TR1=1;               //turn on T1  
    while (TF1==0);      //wait for TF1 to roll over  
    TR1=0;                //turn off T1  
    TF1=0;                //clear TF1  
}  
}
```

9.23) Write an 8051 c program to toggle only pin P1.5 continuously every 250ms. Use Timer 0, mode 2 (8-bit auto-reload) to create the delay.

Sol :- Assume XTAL = 11.0592 MHz.

$$\begin{aligned} \text{Maximum possible time delay in Mode 2} &= \frac{\text{Maximum value in Mode 2}}{\text{crystal freq}} \times \frac{12}{10^6} \\ &= FF(b) \times \frac{12}{11.0592 \times 10^6} \\ &= 276.6927 \text{ usec.} \end{aligned}$$

Required time delay is greater than maximum time delay ie $250 \text{ msec} > 276.6927 \text{ usec.}$

$$\text{counter 'N'} = \frac{250 \text{ msec}}{276.692 \text{ usec}} = 903.52$$

In the above counter value, we are getting fraction value so change the counter value.

Case 1 : Counter 'N' = $\frac{250 \text{ msec}}{50 \text{ msec}} = 5000.$



- * 1st Generate a time delay of 50μsec & repeat the process for 5000 times i.e. $50\mu\text{sec} \times 100 \times 50 = 250\text{msec}$.
- * It can be written as $100 \times 50 = 5000$.

Computation of Initial value :-

$$\begin{aligned} [\text{Initial value} - 1] &= \frac{\text{Maximum value in Mode 2}}{\text{Required Time delay}} \times \frac{\text{crystal freq}}{12} \\ &= \text{FF(h)} - \left[50\mu\text{sec} \times \frac{11.0592\text{MHz}}{12} \right] \\ &= 255(d) - 46.08 + 1 \\ &= 210(d) \end{aligned}$$

Initial value. = Dab.

Case 2 : * counter 'N' = $\frac{250\text{msec}}{25\mu\text{sec}} = 10000$.

$$\boxed{N = 10,000}$$

* It can be written as $250 \times 40 = 10000$.

i.e. $25\mu\text{sec} \times 250 \times 40 = 250\text{msec}$.

$$\begin{aligned} [\text{Initial value} - 1] &= \frac{\text{Max. value in Mode 2}}{\text{Required Time delay}} \times \frac{\text{crystal freq}}{12} \\ &= \text{FF(h)} - \left[25\mu\text{sec} \times \frac{11.0592 \times 10^6}{12} \right] \\ &= 255(1) - 23.04 d + 1 \\ &= 233. \end{aligned}$$

Initial value. = E9h.

* we can also load equivalent -ve value of E9h.

$$\text{i.e. } \boxed{E9h = -23}$$



//tested for DS89C420, XTAL = 11.0592MHz, using the Proview32 compiler

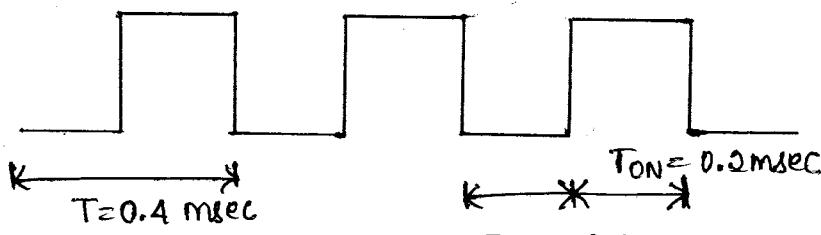
```
#include <reg51.h>
void delay(void);
Sbit mybit = P1^5;
void main(void)
{
    unsigned char X, Y;
    while(1)
    {
        mybit=~mybit;           //toggle P1.5
        for(X=0; X=250; X++)
            for(Y=0; Y=40; Y++)
                delay();
    }
}

void delay(void)
{
    TMOD=0x02;               //Timer 0, mode 2 (8-bit auto-reload
    TH0=-23;                 //load TH0 (auto-reload value)
    TR0=1;                   //turn on T0
    While(TF0==0);           //wait for TF0 to roll over
    TR0=0;                   //turn off T0
    TF0=0;                   //clear TF0
}
```



9.24) Write an 8051 C program to create a frequency of 2500 Hz on pin P2.7. Use Timer 1, mode 2 to create the delay.

Sol:- $\xrightarrow{2500 \text{ Hz}}$



$$T = \frac{1}{F}$$

$$\frac{1}{2500 \text{ Hz}}$$

$$T = 0.4 \text{ msec}$$

$$T = 0.4 \text{ msec}$$

$$T = T_{ON} + T_{OFF}$$

$$T = 0.2 \text{ msec} + 0.2 \text{ msec.}$$

$$T_{ON} = T_{OFF} = 0.2 \text{ msec.}$$

Computation of Initial value :-

$$\begin{aligned} [\text{Initial value} - 1] &= \text{Max. Value in Mode 1.} - \left[\frac{\text{Required Time delay} \times \text{crystal freq}}{12} \right] \\ &= \text{FF(h)} - \left[0.2 \text{ msec} \times \frac{11.0592 \times 10^6}{12} \right] \\ &\approx 255(d) - [184.32(d)] + 1 \\ &\approx 72(d) \end{aligned}$$

$$\boxed{\begin{array}{l} \text{Initial} = 48 \text{ h.} \\ \text{Value} \end{array}}$$



//tested for DS89C420, XTAL = 11.0592MHz, using the Proview32 compiler

```
#include <reg 51.h>

void T1M2Delay(void);
Sbit mybit=P2^7;
void main(void)
{
    unsigned char X;
    while(1)
    {
        mybit=~mybit;      //toggle P2.7
        T1M2Delay ();
    }
}

void T1M2Delay(void)
{
    TMOL=0x20;          //Timer 1,mode 2 (8-bit auto-reload)
    THL=-184;           //load TH1(auto-reload value)
    TR1=1;               //turn on T1
    while (TF1==0);     //wait for TF1 to roll over
    TR1=0;               //turn off T1
    TF1=0;               //clear TF1
}
```

9.25) A switch is connected to pin P1.2. Write an 8051 C program to monitor SW and create the following frequencies on pin P1.7:

Sw=0: 500Hz

Sw=1: 750Hz

Use Timer 0, mode 1 for both of them.



Sol: Case 1: When $SW=0 : 500\text{Hz}$.

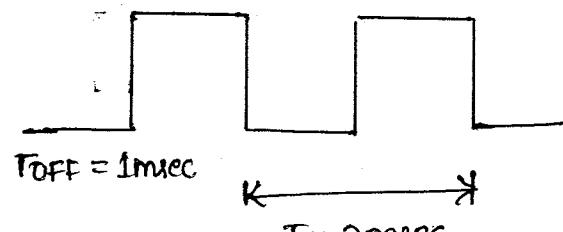
$$W.K.T \quad T = \frac{1}{F} = \frac{1}{500\text{Hz}}$$

$$T = 2\text{msec}$$

$$T = T_{ON} + T_{OFF} = 2\text{msec}$$

$$T = 1\text{msec} + 1\text{msec}$$

$$T = T_{ON} + T_{OFF} = \underline{\underline{2\text{msec}}}$$



$$\text{Initial value} - 1 = \frac{\text{Max value in Mode 1}}{\text{Required Time delay}} \times \frac{\text{crystal freq}}{12}$$

$$= FFFF\text{h} - \left[1\text{msec} \times \frac{11.0592 \times 10^6}{12} \right]$$

$$= 65535(\text{d}) - 921.6(\text{d}) + 1$$

$$= 64614(\text{d})$$

$\text{Initial value.} = FC66\text{h}$



Case 2: When $SW=1 : 750\text{Hz}$.

$$T = \frac{1}{F} = \frac{1}{750\text{Hz}} = 1.33\text{msec.}$$

$$T = T_{ON} + T_{OFF} = 0.666\text{msec} + 0.666\text{msec.}$$

Initial value computation :-

$$\text{Initial value} - 1 = \frac{\text{Max value in Mode 1}}{\text{Required Time delay}} \times \frac{\text{crystal freq}}{12}$$



(85)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

$$\begin{aligned} &= \text{FFFF (h)} - \left(0.666 \text{ msec} \times \frac{11.0592 \times 10^6}{12} \right) \\ &= 65535(\text{d}) - 614(\text{d}) + 1 \\ &= 65020(\text{d}) \end{aligned}$$

Initial value = FDFC(h).

TLO = FCh &

THO = FDh.

* TMOD = 01h .

// tested for AT89C51/52, XTAL-11.0592MHz, using the Proview32 compiler

```
#include <reg51.h>
sbit mybit=P1^5;
sbit SW=P1^7;
void delay(unsigned char);
void main(void)
{
    SW=1;                                //make P1.7 an input
    while(1)
    {
        mybit=~mybit;                      //toggle P1.5
        if(SW==0)                          //check switch
            delay(0);
        else
            delay(1);
    }
}
```

void delay(unsigned char c)



```
{  
    TMOD=0x01;  
    if(c==0)  
    {  
        TL0=0x67;           //FC67  
        TH0=0xFC;  
    }  
    else  
    {  
        TL0=0x9A;           //FD9A  
        TH0=0xFD;  
    }  
    TR0=1;  
    while(TF0==0);  
    TR0=0;  
    TF0=0;  
}
```



OR

```
#include <reg51.h>
sbit mybit=P1^5;
sbit SW=P1^7;
void delay();
void main(void)
{
    SW=1;                                //make P1.7 an input
    TMOD=0x01;
    while(1)
    {
        mybit=~mybit;                  //toggle P1.5
        if(SW==0)                      //check switch
        {
            TL0=0x67;
            TH0=0xFC;
            delay ();
        }
        else
        {
            TL0=0x9A;
            TH0=0xFC;
            delay ();
        }
    }
}

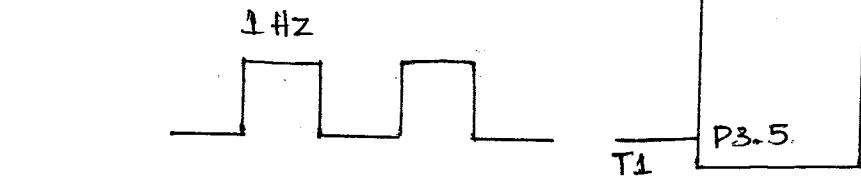
void delay()
{
```



```
TR0=1;  
while (TF0==0);  
TR0=0;  
TF0=0;  
}
```

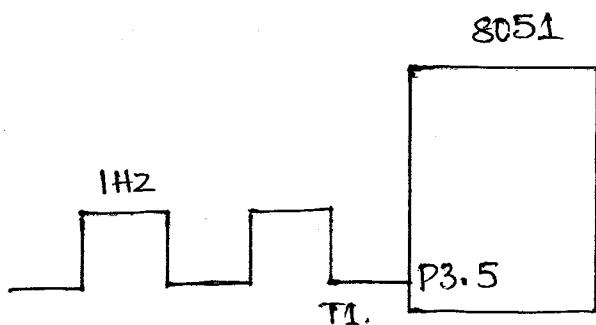
9.26) Assume that 1-Hz external clock is being fed into pin T1 (P3.5).Write a C program for counter 1 in mode 2 (8-bit auto reload) to count up and display the state of the TL1 count on P1.Start the count at 0H

```
#include <reg51.h>  
Sbit T1 = P3^5;  
void main(void)  
{  
    T1=1;                                //make T1 an input  
    TMOD=0x60;  
    TH0=0;                                //set count to 0  
    while(1)                               //repeat for ever  
    {  
        do  
        {  
            TR1=1;                          //start timer  
            P1=TL1;                          //place value on pins  
        }  
        while(TF0==0);                    //wait here  
        TR1=0;                            //stop timer  
        TF1=0;                            //clear flag  
    }  
}
```



9.27) Assume that a 1-Hz external clock is being fed into pin T0(P3.4).Write a C program for counter 0 in mode 1(16-bit) to count the pulses and display the TH0 and TL0 registers on P2 and P1,respectively.

```
#include <reg51.h>
void main(void)
{
    T0=1;                                //make T0 an input
    TMOD=0x05;
    TL0=0;                                //set count to 0
    TH0=0;                                //set count to 0
    While (1)                             //repeat forever
    {
        do
        {
            TR0=1;                          //start timer
            P1=TL0;                         //place value on pins
            P2=TH0;
        }
        while (TF0==0);                  //wait here
        TR0=0;                            //stop timer
        TF0=0;
    }
}
```



90

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

Two switches are connected to pin P3.0 & P3.1 respectively. Write an 8051 C program to monitor switches and create the following frequencies on pin P0.0 according to switch positions.

SW1	SW2	FREQUENCY
0	0	100Hz
0	1	200Hz
1	0	300Hz
1	1	400Hz

```
#include <reg51.h>
sbit SW0=P3^2;
sbit SW1=P3^3;
sbit pin=P1^5;
void delay();
void main(void)
{
    SW0=1;                                //make P3.2 an input
    SW1=1;                                //make P3.3 an input
    TMOD=0x01;
    while (1)
    {
        pin=~pin;                         //toggle P1.5
        if(SW1==0 & SW0==0)                //check switch
        {
            TL0=0x03;
            TH0=0xEE;
        }
    }
}
```



91

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

```
delay();  
}  
if(SW1==0 & SW0==1) //check switch  
{  
    TL0=0x00;  
    TH0=0xF7;  
    delay();  
}  
if(SW1==1 & SW0==0) //check switch  
{  
    TL0=0x00;  
    TH0=0xFA;  
    delay();  
}  
if(SW1==1 & SW0==1) //check switch  
{  
    TL0=0x80;  
    TH0=0xFB;  
    delay();  
}  
}  
}  
void delay()  
{  
    TR0=1;  
    while (TF0==0);  
    TR0=0;  
    TF0=0;  
}
```



Using SWITCH condition

```
#include <reg51.h>
sbit SW0=P3^0;
sbit SW1=P3^1;
sbit pin=P0^0;
void delay(void);
void main()
{
    unsigned char i;
    SW0=1;
    SW1=1;
    TMOD=0x01;
    while(1)
    {
        pin=~pin;           //toggle P1.5
        i=P3;
        i=i & 0x03;

        switch(i)
        {
            case(0):          //check switch
            {
                TL0=0x03;
                TH0=0xEE;
                delay();
                break;
            }
            case(1):          //check switch
            {
            }
        }
    }
}
```



(93)

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

```
{  
    TL0=0x00;  
    TH0=0xF7;  
    delay();  
    break;  
}  
case(2): //check switch  
{  
    TL0=0x00;  
    TH0=0xFA;  
    delay();  
    break;  
}  
case(3): //check switch  
{  
    TL0=0x80;  
    TH0=0xFB;  
    delay();  
    break;  
}  
}  
}
```

```
void delay(void)  
{  
    TR0=1;  
    while (TF0==0);  
    TR0=0;  
    TF0=0;  
}
```



9.28) Assume that a 2-Hz external clock is being fed into pin T1 (P3.5). Write a C program for counter 0 in mode 2 (8-bit auto reload) to display the count in ASCII. The 8-bit binary count must be converted to ASCII. Display the ASCII digits (in binary) on P0, P1, and P2 where P0 has the least significant digit. Set the intial value of TH0 to 0.

```
#include <reg51.h>

void BinToASCII (unsigned char);

void main ( )
{
    unsigned char value;
    T1=1;
    TMOD=0x06;
    TH0=0;
    while (1)
    {
        do
        {
            TR0=1;
            value =TL0;
            BinToASCII (value);
        }
        while (TF0==0);
        TR0=0;
        TF0=0;
    }
}

void BinToASCII (unsigned char value)
{
    unsigned char x,d1,d2,d3;
```



95

ARUNKUMAR.G M.Tech, Lecturer in E&CE Dept. S.T.J.I.T, Ranebennur.

```
x = value / 10;  
d1 = value % 10  
d2 = x % 10;  
d3 = x / 10;  
P0 = 30 | d1;  
P1 = 30 | d2;  
P2 = 30 | d3;  
}
```

9.29. Assume that a 60-Hz external clock is being fed into pin T0 (P3.4). Write a C program for counter 0 in mode 2 (8-bit auto-reload) to display the seconds and minutes on P1 and P2, respectively.

```
#include <reg.51>  
void ToTime (unsigned char);  
void main ()  
{  
    unsigned char val;  
    T0=1;  
    TMOD=0x06;           // T0, mode 2, couter  
    TH0=-60;             // sec = 60 pulses  
    while (1)  
    {  
        do  
        {  
            TR0=1;  
            sec = TL0;  
            ToTime (val);
```



```
    }  
  
    while( TF0==0);  
  
    TR0=0;  
  
    TF0=0;  
  
}  
}  
  
void ToTime(unsigned char val)  
{  
  
    unsigned char sec,min;  
  
    min = value / 60;  
  
    sec = value % 60;  
  
    P1 = sec;  
  
    P2 = min;  
}
```



Chapter - 6

ARUNKUMAR G M.Tech
Lecturer in E&CE Department
S.T.J.I.T., Ranebennur

Serial Communication

Communication:

Communication means exchange of meaningful information between transmitter & receiver i.e. source & destination.

Communication are mainly classified into

1. **Serial communication**
2. **Parallel communication**

Serial communication:

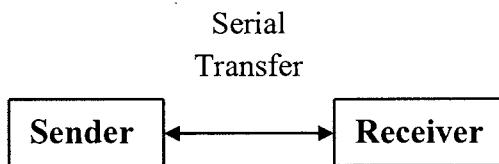


Fig 1 shows serial communication

- ❖ Serial communication uses **single data line** to transfer data.
- ❖ In serial communication **one bit is transferred at a time over a single data line**.
- ❖ Serial communication **enables two computers located in two different cities to communicate over the telephone**.
- ❖ Serial communication uses **single data line** instead of the **8-bit data line** of parallel communication makes it **much cheaper**.
- ❖ Serial communication is used for **long distance communication**



- ❖ In serial communication **data byte** (8-bit data) must be **converted** to **serial bits** using **parallel-in-serial out shift register**, and then it can be transmitted over a single data line.
- ❖ At the receiving end there must be a **serial-in-parallel out shift register** to receive the serial data & **pack** them into **a byte**.
- ❖ Serial communication is **slower than parallel** communication
- ❖ If the **data** is to be **transferred** on the **telephone line**, it must be converted from 1's & 0's to **AUDIO tones**, which are sinusoidal shape signals. This conversion is performed by a peripheral device called a **MODEM** i.e. "**MODULATOR / DEMODULATOR**".
- ❖ When the communication **distance is short**, the digital signal can be transferred it on a **simple wire** & requires **NO modulation**

Eg: - IBM keyboards transfer data to the motherboard.

Parallel communication:

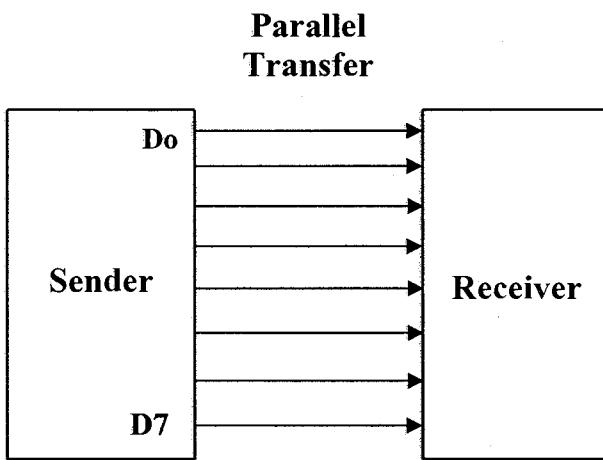


Fig 2 Parallel communication

- ❖ In parallel communication **number of lines required to transfer data depends on the number of bits to be transferred simultaneously**.
- ❖ The information is simply grabbed from the **8-bit data bus** of the **sender** presented (transferred) to the **8-bit data bus** of the **receiver**.



- ❖ Parallel communication works only for **shorter distance**.
- ❖ For **longer distance** communication long **cables** diminish & even **distorts signals**.
- ❖ The data transmission over a **long distance** using parallel communication is **impractical** due to **increase in cost of cabling**.
- ❖ Parallel communication is **faster than serial communication**.

Eg: - **Data transmission from computer to printer.**

Serial Data transmission formats:-

The data in serial communication may be sent in two ways:

1. ASYNCHRONOUS
2. SYNCHRONOUS

ASYNCHRONOUS Serial communication:-

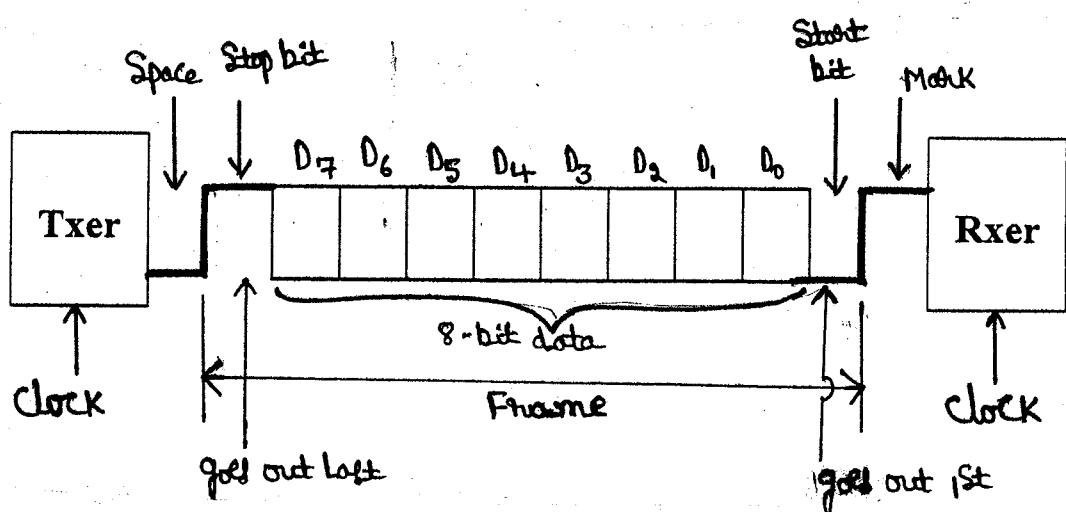
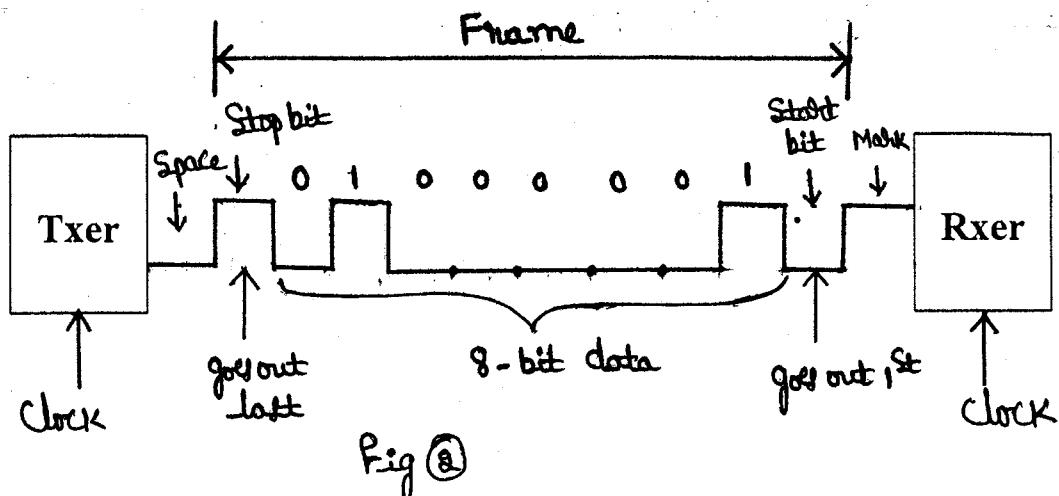


Fig 1 Asynchronous format.

- ❖ Asynchronous serial data communication is widely used for character-oriented transmission.
- ❖ Each character (data) is placed between start & stop bits as shown in figure. This is called **Framing**.



- ❖ The start bit is always one-bit, but the stop bit can be one or two-bits.
- ❖ The start bit is always a 0 (low, called Space) followed by a character & one or two stop bits (High, called Mark).



- ❖ In fig 2 the ASCII character "A" (8-bit binary $01000001 = 41h$) is framed between the **start bit** & a single **stop bit**. The **LSB** is sent out first.
- ❖ When there is **No data transfer**, the signal is high (1), which is referred to as **MARK**.
- ❖ The **low** (0) is referred to as **SPACE**.
- ❖ In synchronous serial communication, peripheral chips & MODEMS can be **programmed** for data that is **7-bit** or **8-bit** wide.
- ❖ While in older systems ASCII characters were 7-bit & in recent years ASCII characters are 8-bit.
- ❖ Assuming that we are transferring an **8-bit** ASCII characters using **1- Stop bit** & **1- Start bit** i.e. total of **10-bits** for **each character**. Therefore for each 8-bit character there are **extra 2-bits**, which give **20% overhead**.
- ❖ In some systems, the parity bit of the character byte is included in the data frame in order to maintain **data Integrity**.
i.e. for each 8-bit character we have a **single parity bit** in **addition** to **start & stop bits**.
The **parity bit** is **ODD** or **EVEN**.



- ❖ In case of **ODD-parity** bit the number of **data bits**, including the **parity bit**, has an **odd number of 1's**.
- ❖ In case of **EVEN-parity** bit the number of **data bits**, including the **parity bit**, has an **even number of 1's**.

Data Transfer Rates:

- ❖ The rate of data transfer in serial communication is stated in **bps** (bits per second). Another widely used terminology for bps is **Baud rate**.
- ❖ The **baud rate & bps** are **not same**. The **baud rate** is the **MODEM** terminology & is defined as the **number of signal changes per second**.
- ❖ In **modem** a single change of signal sometimes transfers several bits of data.
- ❖ For **conductor wire**, the **baud rate & bps** are the **same**. So for this reason we use the term **bps** & **baud** interchangeably.

Baud rate in the 8051:-

- ❖ The 8051 transfers & receives data serially at many different baud rates. The baud rate in the 8051 is programmable.
- ❖ A standard crystal frequency, XTAL=11.0592 MHz is used to generate the baud rate.
- ❖ The 8051 divides the crystal frequency by 12 to get the machine cycle frequency.
i.e. $\text{XTAL}/12 = 11.0592\text{MHz}/12 = \mathbf{921.6\text{ KHz}}$.

The 8051's serial communication **UART circuitry divides** the machine cycle frequency of 921.6 KHz by **32** i.e. $921.6\text{ KHz}/32 = \mathbf{28,800\text{ Hz}}$, then fed to the Timer1 to set the baud rate as shown in below figure.

- ❖ The 8051 **baud rate is set** by **Timer1** using **Mode 2** (8-bit auto reload).
- ❖ To get the baud rates compatible with the PC, we must load TH1 with the values shown in below table.

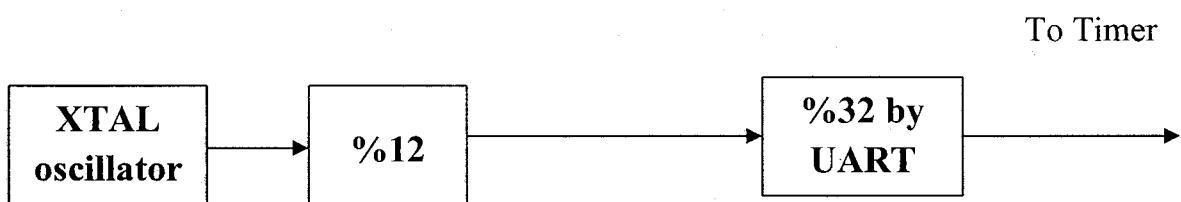


Table1: Timer1 TH1 register values for various Baud Rates.

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

Note: XTAL = 11.0592 MHz

Note:



SYNCHRONOUS serial communication:-

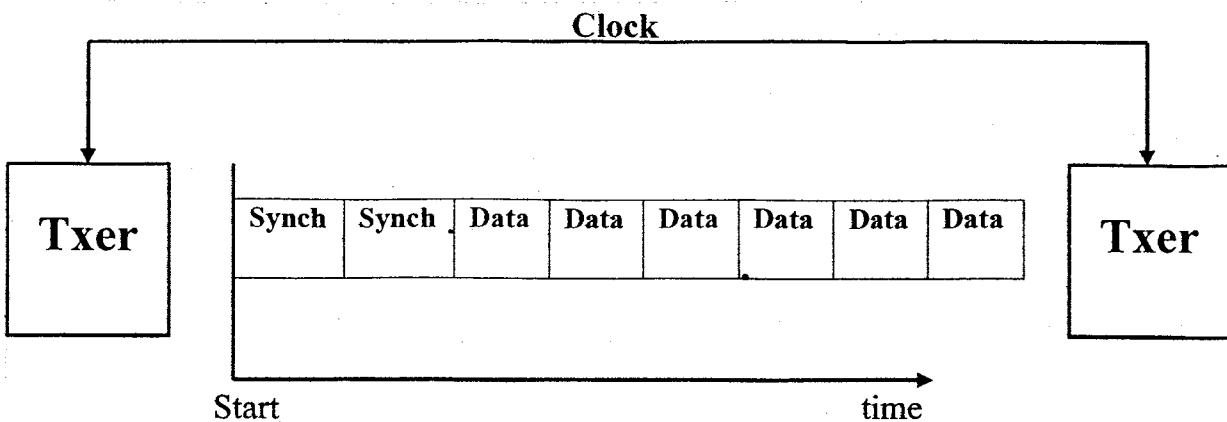


Fig1: Synchronous Transmission format.

- ❖ The synchronous method **transfers a block of data (Character) at a time.**
- ❖ The start & stop bits in each frame of asynchronous format represents wasted overhead bytes that **reduces the overall character rate.**
- ❖ These start & stop bits can be eliminated by **synchronizing receiver & transmitter** i.e. by having a **common clock signal.**
- ❖ In synchronous transmission **synchronous bits are inserted instead of start & stop bits.**



Comparison between Asynchronous & Synchronous serial communication

Sl No.	Asynchronous serial communication	Synchronous serial communication
1	<i>Transmitter and Receiver are not synchronized by clock</i>	<i>Transmitter and Receiver are synchronized by clock</i>
2	<i>Bits of data are transmitted at constant rate</i>	<i>Data bits are transmitted with synchronization of clock</i>
3	<i>Character may arrive at any rate at receiver</i>	<i>Character is received at constant rate</i>
4	<i>Data transfer is character oriented</i>	<i>Data transfer takes place in blocks</i>
5	<i>Start & Stop bits are required to establish communication of each character</i>	<i>Start & Stop bits are not required to establish communication of each character; however synchronization bits are required to transfer the data block.</i>
6	<i>Used in LOW-SPEED transmission at about speed less than 20 Kbs</i>	<i>Used in HIGH-SPEED transmission</i>

Baud Rate & Transmission Rate:-

Baud rate is defined as the number of bits transmitted per second.

Eg: Consider a baud rate of 1200 then

$$\text{Transmission rate} = 1 \text{ Sec} / \text{Baud rate}.$$

$$1 \text{ bit} = 1 \text{ sec} / 1200 = 0.83\text{ms}.$$

0.83ms is the delay between two bits



PC Baud Rates:

Note:-

Some of the Baud rates supported by 486/Pentium IBM PC BIOS

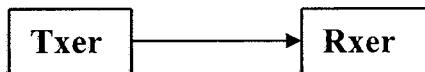
110	150	300	600	1200	2400	4800	9600	19200
-----	-----	-----	-----	------	------	------	------	-------

Serial Data Transmission classification:-

Serial data transmission can be classified on the basis how transmission occurs:

- 1) **SIMPLEX**
- 2) **HALF DUPLEX**
- 3) **FULL DUPLEX**

1. Simplex:-



In simplex transmission data is transmitted in only one direction. There is no possibility of data transfer in other direction.

Eg: - **From computer to printer.**

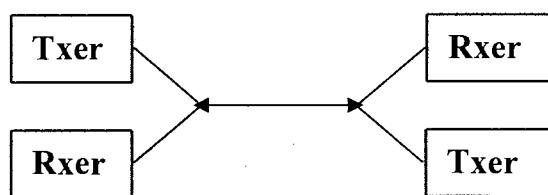
{

Duplex: - In data transmission if the data can be transmitted and received, it is a duplex transmission.

}



2. Half Duplex:-

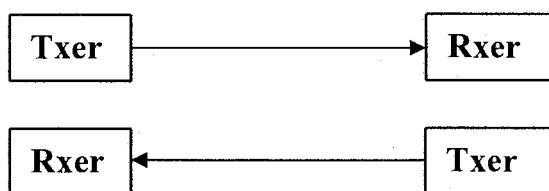


In half duplex, data transmission is possible only one way at a time.

OR

The half duplex transmission allows the data transfer in both direction, but not simultaneously. Eg: - **Walkie-Talkie**

3. Full Duplex:-



Full duplex transmission allows the data transfer in both directions simultaneously.

Eg: - **Transmission through Telephone lines.**



SCON (Serial control) Register:

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

SM0:SM1:- Serial mode specifier

SM0	SM1	MODE	Description	Baud Rate
0	0	MODE0	Shift Register	Baud=f/12
0	1	MODE1	8-bit UART	Baud= variable (Set by mode1,2)
1	0	MODE2	9-bit UART	Baud=f/12 of f/32
1	1	MODE3	9-bit UART	Baud= variable

Where **f** is the crystal frequency.

SM2:- Used for **multiprocessor communication**. In 8051 we are not using multiprocessor communication so SM2 is made 0 i.e. **SM2=0**

REN: - Receive Enable bit.

REN is set to 1 to **enable reception** (REN=1)

REN is cleared to 0 to **disable reception** (REN=0)

TB8:- Transmit bit 8

Set/cleared by hardware to **determine state** of the **9th data bit** transmitted in **9-bit UART** (In mode 2 & 3)

RB8:- Receive bit 8.

Set/cleared by hardware to **indicate state** of **9th data bit received**. (In mode 2 & 3)

In 8051 mode is used, so these two bits are cleared i.e. **TB8=RB8=0**



TI: - Transmit interrupt flag

Set by hardware at the beginning of the stop bit in Mode 1. (i.e. set by hardware whenever byte is transmitted)

TI must be cleared by software.

RI: - Receive interrupt flag.

Set by hardware at the beginning of the stop bit in Mode 1. (i.e. set by hardware whenever byte is received)

RI must be cleared by software.

PCON (Power Mode Control) Special function register:-

SMOD	-	-	-	GP1	GP0	PD	IDL
------	---	---	---	-----	-----	----	-----

(NOT bit addressable register)

SMOD:- Serial Baud Rate modify bit.

Set to 1 by program to **double the baud rate** using Timer1 for mode 1, 2 & 3. (**SMOD=1**)

Cleared to 0 by program to use Timer1 Baud Rate (**SMOD=0**)

Bit 6-4: Not implemented

GF1:- General purpose user flag bit 1

Set/cleared by program

GF0:- General purpose user flag bit 0

Set/cleared by program.

PD: - Power Down bit

Set to 1 by program to enter **power down** configuration for CHMOS processors.



IDL: - Idle mode bit.

Set to 1 by program to enter **IDLE mode** configuration for CMOS processors.

Formula to compute BAUD Rate:-

$$\text{Baud Rate} = \frac{\text{Crystal Frequency}}{12 \times 32} \times \frac{1}{256 - \text{TH1}}$$

Formula to compute Initial value for Particular Baud rate:-

$$\text{TH1} = 256 - \frac{\text{Crystal Frequency}}{12 \times 32 \times \text{Baud rate}}$$

Doubling the Baud rate in 8051:-

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

1. Use a Higher frequency crystal(**Not Feasible**)
2. **Change a bit in the PCON register.** (used by 8051 Microcontroller)

Procedure for Doubling the baud rate in the 8051:-

- ❖ When the 8051 is **powered up**, **SMOD bit(D7) of the PCON register is zero** (i.e. $\text{SMOD}=0$)
- ❖ We can set it to **high** by software & thereby **double the baud rate**.

When

$$\text{SMOD} = 0: \quad \text{Baud Rate} = \frac{\text{Crystal Frequency}}{12 \times 32} \times \frac{1}{256 - \text{TH1}}$$

$$\text{SMOD} = 1: \quad \text{Baud Rate} = \frac{\text{Crystal Frequency}}{12 \times 16} \times \frac{1}{256 - \text{TH1}}$$



}

- ❖ The following sequence of instructions must be used to set high SMOD (D7) of PCON register.

```
MOV A, PCON      ; Place a copy of PCON in ACC  
SETB ACC.7      ; Make D7=1  
MOV PCON, A      ; Now SMOD=1, without changing any other bits
```

Note:

SETB ACC.7

MOV PCON, A

- Now only SMOD is set & all other bits of PCON is cleared.
- Only we have to set the PCON D7 bit & we must not alter the other bits of PCON register.

SBUF Register:-

- ❖ SBUF is an **8-bit** register used only for **serial communication** in the 8051.
- ❖ Whenever 8051 wants a **byte of data** to be transferred via **TxD** line, it must be placed in the **SBUF register**.
- ❖ Similarly **SBUF holds the byte of data**, when it is **received** by the 8051's **RxD** line.
- ❖ SBUF can be accessed like any other registers in the 8051.

Eg:-

1. MOV SBUF,#'D' ; load SBUF = 44h ASCII for 'D'
2. MOV SBUF,A ; copy Accumulator into SBUF
3. MOV A,SBUF ; copy SBUF into accumulator



NOTE:-

- ❖ When a byte is written into SBUF, it is **framed** with the **start & stop bits** & transferred serially via the TxD pin
- ❖ Similarly when the bits are received serially via RxD, the 8051 **deframes** it by eliminating the **stop & start bits**, making a byte out of the data received, & then placing it in SBUF.

Different Baud Rates:

Baud rate comparison for **SMOD=0** & **SMOD=1**

TH1		Baud Rate	
DECIMAL	HEX	SMOD = 0	SMOD = 1
-3	FD	9,600	19,200
-6	FA	4,800	9,600
-12	F4	2,400	4,800
-24	E8	1,200	2,400
<u>Note: XTAL = 11.0592 MHz</u>			



Procedure to program the 8051 to TRANSFER data serially:

In programming the 8051 to transfer character bytes serially, the following steps must be taken:

1. The **TMOD** register is loaded with the value **20H**, indicating the use of **TIMER1** in **mode 2** (8-bit auto-reload) **to set the baud rate**.
2. The **TH1** is loaded with one of the values four values **to set the baud rate** for serial data transfer (Assuming **XTAL=11.0592MHz**).
3. The **SCON** register is loaded with the value **50H**, indicating serial mode 1, where an **8-bit data** is framed with **start and stop bits**.
4. **TR1** is **set to 1** to start Timer1.
5. **TI** is cleared by the “**CLR TI**” instruction.
6. The character byte to be transferred serially is written into **SBUF** register.
7. The **TI** flag bit is monitored with the use of the instruction “**JNB TI, label**” to see if the character has been transferred completely.
8. To transfer the next character, go to step 5.

Importance of the TI Flag:

To understand the importance of the role of **TI**, look at the following sequence of steps that the 8051 goes through in transmitting a character via **TxD**

1. The **byte character to be transmitted** is written into **SBUF** register.
2. The **start bit is transferred**.
3. The **8-bit character is transferred one bit at a time**.
4. The **stop bit is transferred**. It is during the transfer of the stop bit that the 8051 **raises the TI flag (TI=1)**, indicating that the **last character was transmitted** and it is **ready to transfer the next character**.
5. By **monitoring the TI flag**, we make sure that we are **not overloading** the **SBUF** register. If we write another byte into **SBUF** register before **TI** is raised, the untransmitted portion



of the previous byte will be lost. In other words, when the 8051 finishes transferring a byte, it raises the TI flag to indicate it is ready for the next character.

6. After SBUF is loaded with a **new byte**, the **TI flag** bit must be forced to **0** by the “**CLR TI**” instruction in order for this **new byte** to be **transferred**.

Procedure to program the 8051 to RECEIVE data serially:

In programming the 8051 to receive character bytes serially, the following steps must be taken:

1. The **TMOD** register is loaded with the value **20H**, indicating the use of **TIMER1** in **mode 2** (8-bit auto-reload) to set the **baud rate**.
2. The **TH1** is loaded with one of the values four values to **set the baud rate** for serial data transfer (Assuming **XTAL=11.0592MHz**).
3. The **SCON** register is loaded with the value **50H**, indicating serial mode 1, where an **8-bit data** is framed with **start** and **stop** bits.
4. **TR1** is **set to 1** to start Timer1.
5. **RI** is cleared by the “**CLR RI**” instruction.
6. The **RI** flag bit is monitored with the use of the transmission “**JNB RI, label**” to see if an entire character has been received yet.
7. **When RI is raised**, SBUF has the byte. Its contents are moved into a safe place.
8. To receive the next character, go to step 5.



Importance of the RI Flag:

In receiving bits via its **RxD pin**, the 8051 goes through the following steps:

1. It receives the **start bit indicating** that the **next bit is first bit of the character byte it is about to receive.**
2. The **8-bit character is received one bit at a time.** When the last bit is received, a byte it is about to receive.
3. The **stop bit is received.** When receiving the stop bit the 8051 makes **RI=1**, indicating that an entire character byte has been received and must be placed up before it gets overwritten by an incoming character.
4. By checking the **RI flag bit** when it is raised, we know that a character has been received and is **sitting in the SBUF register.** We copy the **SBUF** contents to a safe place in some other register or memory before it is lost.
5. After the **SBUF** contents are **copied** into a **safe place**, the RI flag bit must be forced to 0 by the “**CLR RI**” instruction in order to allow the next received character byte to be placed in **SBUF**. **Failure to do this causes loss of the received character.**

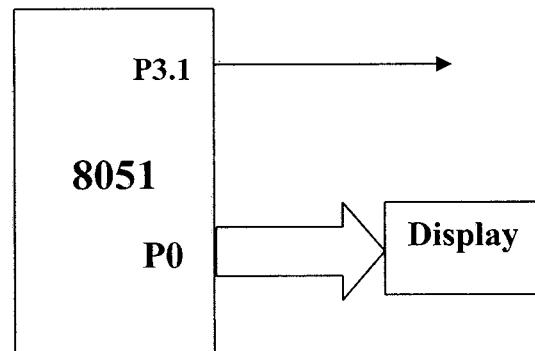


1. Write a program to transfer a letter 'Y' serially at 9600 baud continuously, and also to send a letter 'N' through port0, which is connected to a display device.

```
ORG 00h  
MOV TMOD, #20H      ; timer 1, mode 2  
MOV TH1, #-3        ; 9600 baud rate  
MOV SCON, #50H      ; 8 bits, 1stop, REN enabled  
SETB TR1            ; START TIMER 1
```

AGAIN:

```
MOV SBUF, #'Y'    ; transfer 'Y' serially  
HERE: JNB TI, HERE      ; WAIT FOR TRANSMISSION TO BE OVER  
       CLR TI           ; clear TI for next transmission  
       MOV P0, #'N'        ; move 'N' to p0 for parallel transfer  
       SJMP AGAIN         ; repeat  
END
```



2. Take a data in through ports 0, 1 and 2, one after the other and transfer this data serial, continuously.

```
ORG 00h  
MOV TMOD, #20H  
MOV TH1, #-6  
MOV SCON, #50H  
MOV P0, #0FFH  
MOV P1, #0FFH  
MOV P2, #0FFH  
SETB TR1  
RPT: MOV A, P0  
      ACALL SEND  
      MOV A, P1  
      ACALL SEND  
      MOV A, P2  
      ACALL SEND  
      SJMP RPT
```

; -----**TRANSFERRING SERIALLY-----**

```
SEND: MOV SBUF, A  
HERE: JNB TI#, HERE  
      CLR TI  
      RET  
      END
```



3. Write a program to receive the data which has been sent in serial form and send it out to port0 in parallel form. Also save the data at RAM location 60h.

Sol:-

```
ORG 00h  
MOV TMOD, #20H  
MOV TH1, #-3  
MOV SCON, #50H  
SETB TR1  
CLR RI  
RPT: JNB RI, RPT  
MOV A, SBUF  
MOV P0, A  
MOV 60H, A  
END
```

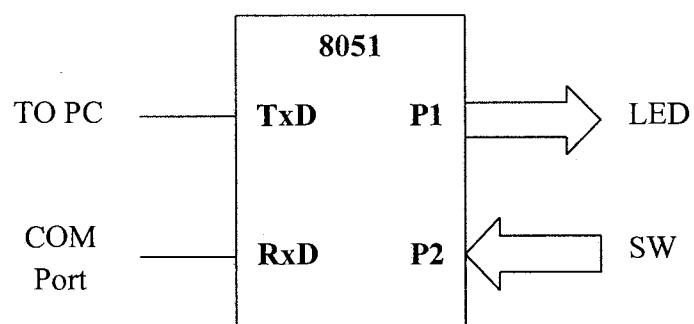
4. Assume that the 8051 serial port is connected to the COM port of the IBM PC, and on the C we are using the hyperterminal program to send and receive data serially. P1 & P2 of the 8051 are connected to LEDs and switches, respectively. Write an 8051 program to (a) send to the PC the message " We are ready", (b) receive any data sent by the PC and put it on the LEDs connected to the P1, and (c) get data on switches connected to P2 and send it to the PC serially. The program should perform part (a) once, but parts (b) and (c) continuously. Use the 4800 baud rate.

Sol:-

```
ORG 00h  
MOV P2, #0FFH  
MOV TMOD, #20H  
MOV TH1, #0FAH  
MOV SCON, #50H  
SETB TR1  
MOV DPTR, #MYDATA
```

H_1: CLR A

MOVC A,@A+DPTR1



```
JZ B_1  
ACALL SEND  
INC DPTR  
SJMP H_1  
B_1: MOV A, P2  
ACALL SEND  
ACALL RECV  
MOV P1, A  
SJMP B_1
```

; -----**SERIAL DATA TRANSFER-----**

```
SEND: MOV SBUF, A
```

```
H_2: MOV T1, H_2
```

```
CLR T1
```

```
RET
```

; -----**RECIEVE DATA SERIALLY-----**

```
RECV: JNB R1, RECV
```

```
MOV A, SBUF
```

```
CLR R1
```

```
RET
```

; -----**THE MESSAGE-----**

```
MYDATA: DB "WE ARE READY", 0
```

```
END
```



5) Assuming that XTAL=11.0592MHz for the following program, state(a)what this program does,(b)compute the frequency used by Timer 1 to set the baud rate, and (c) find the baud rate of the data transfer.

```
MOV A, PCON  
SETB ACC.7  
MOV PCON, A  
MOV TMOD, #20H  
MOV TH1,-3  
MOV SCON, #50H  
SETB TR1  
MOV A,"B"  
A_1: CLR TI  
MOV SBUF, A  
H_1: JNB TI H_1  
SJMP A_1
```

Ex10.9 Port 0 of an 8051 is used to monitor a parameter in an industrial environment. If the parameter gives the reading above 0FH,a message 'HI' is to be sent serially. Otherwise, a message 'OK' is to be sent. The words 'HI' and 'OK' are burned into program ROM locations

```
ORG 0000H  
MOV P0, #0FFH ; make P0 an input port  
MOV TMOD, #20H  
MOV TH1, #-3  
MOV SCON, #50H  
SETB TR1  
CHECK: MOV A, P0 ; move P0 into A  
CJNE A, #0FH, TEST ; check if it is equal to 0FH, if not to go test  
SJMP OK ; if equal to 0FH, go to OK  
  
TEST: JNC HI ; if it is greater than 0FH, go to HI  
OK: MOV DPTR, #00A0H ; let DPTR point to message OK  
ACALL ACCESS ; call subroutine to access message ROM area  
SJMP CHECK ; continue monitoring P0  
HI: MOV DPTR, #0090H ; i.e. DPTR pointer to message HI
```



ACALL ACCESS ; call subroutine to access message area

SJMP CHECK ; continue monitoring P0

----- Subroutine to access message ROM area where messages are stored-----

ACCESS: CLR A

MOVC A,@A+DPTR

ACALL SEND

INC DPTR

CLR A

MOVC A,@A+DPTR

ACALL SEND

RET

-----Subroutine to send data serially-----

SEND: MOV SBUF, A

HERE: JNB T1, HERE

CLR T1

RET

ORG 0090H

MES1: DB "HI"

ORG 00A0H

MES2: DB "OK"

END



Ex 10.10 A Square wave is being generated at pin P1.2 this square wave is to be sent to a receiver connected in serial form to this 8051. Write a program for this.

Solution:

Timer 0 in mode 2 is used to generate the square wave on P1.2. Whenever this pin is high, a data FFH is transmitted serially, and when this pin is low, a data 00H is transmitted. This data can be converted into parallel form at the receiver side to regenerate the square Wave there.

```
ORG 0000H
MOV TMOD, #22H      ; timer 0 and timer 1 in mode 2
MOV SCON, #50H
MOV TH1, #-3
MOV TH0, #00H ; count value for timer 0
SETB TR1      ; start timer 1
MOV A, #00H      ; move A=00
CLR P1.2
REPT: SETB TRO      ; start timer 0
BACK: JNB TF0, BACK; wait for timer 0 rollover
CPL A          ; complement A
CPL P1.2      ; complement P1.2
MOV SBUF, A      ; move A to SBUF for transmission
CLR TR0      ; stop timer 0
CLR TF0      ; clear Timer 0 flag
HERE: JNB TI, HERE      ; check for TI flag
CLR T1          ; clear TI to enable next transmission
SJMP REPT      ; repeat the whole process
END
```



Eg10-12) Write a program to send the text string “Hello” to serial #1. Set the baud rate at 9600, 8-bit data, and 1 stop bit.

Solution:

```
        SCON1    EQU 0C0H
        SBUF1    EQU 0CIH
        TI1      BIT  0C1H
        ORG      00H           ; starting position
        MOV      TMOD, #20H
        MOV      TH1, #-3       ; 9600 baud rate
        MOV      SCON1, #50H
        SETB    TR1
        MOV      DPTR, #MESS1   ; display "Hello"
FN:    CLR     A
        MOVC   A,@A+DPTR    ; read value
        JZ     S1             ; check for end of line
        ACALL  SENDCOM2     ; send to serial port
        INC    DPTR          ; move to next value
        SJMP   FN
S1:    SJMP   S1
```

SENDOM2:

```
        MOV     SBUF1, A       ; place value in buffer
HERE1: JNB    TI1, HERE1     ; wait until transmitted
        CLR    TI1            ; clear
        RET
MESS1: DB    "Hello", 0
        END
```



Eg10-13) Program the second serial port of the DS89C4x0 to receive bytes of data serially and output them on P1. Set the band rate at 4800,8-bit data, and 1 stop bit.

Solution:

```
SBUF1    EQU  0C1H      ; second serial SBUF addr
SCON1    EQU  0C0H      ; second serial SCON addr
RI1      BIT   0C0H      ; second serial RI bit addr
ORG      0H              ; starting position
MOV      TMOD, #20H     ; COM2 uses Timer 1 upon reset
MOV      TH1, #-6        ; 4800 baud rate
MOV      SCON1, #50H     ; COM2 has its own SCON1
SETB    TR1             ; start Timer 1
HERE:   JNB    RI1, HERE  ; wait for data to come in
        MOV    A, SBUF1     ; save data
        MOV    P1, A         ; display on P1
        CLR    RI1
        SJMP   HERE
END
```



Eg10-14) Assume that a switch is connected to pin P2.0. Write a program to monitor the switch and perform to monitor the switch and program the following:

- (a) If SW=0 send the message “Hello” to the serial #0 port.
- (b) If SW=1 send the message “Good bye” to the serial #1 port.

Solution:

```
        SCON1      EQU  0C0H
        TI1       BIT   0C1H
        SW1       BIT   P2.0
        ORG        OH           ; starting position
        MOV        TMOD, #20H
        MOV        TH1, #-3      ; 9600 baud rate
        MOV        SCON, #50H
        MOV        SCON1, #50H
        SETB      TR1
        SETB      SW1          ; make sw1 an input
S1:   JB       SW1, NEXT    ; check SW1 an input
        MOV        DPTR, #MESS1  ; if sw1=0 display “Hello”
FN:   CLR      A
        MOVC     A,@A+DPTR   ; read value
        JZ       S1           ; check for end of line
        ACALL    DELAY      ; send to serial port
        INC      DPTR         ; move to next value
        SJMP    FN
NEXT:  MOV      DPTR, #MESS2 ; if SW1=1 display “Goodbye”
LN:   CLR      A
        MOVC     A,@A+DPTR   ; read value
        JZ       S1           ; check for end of line
        ACALL    SENDCOM2   ; send to serial port
        INC      DPTR         ; move to next value
        SJMP    LN
SENDCOM1:
HERE:  MOV      SBUF, A    ; place value in buffer
        JNB      TI, HERE    ; wait until transmitted
        CLR      TI           ; clear
        RET
```



SEDCOM2:

MOV	SBUF1, A	; place value in buffer	
HERE1:	JNB	TI1, HERE1	; wait till transmitted
	CLR	TI1	; clear
	RET		
MESS1:	DB	“Hello”, 0	
MESS2:	DB	“Goodbye”, 0	
	END		

10-15) write a C program for the 8051 to transfer the letter “A” serially at 4800 baud continuously. Use 8-bit data and 1 stop bit.

```
#include<reg51.h>
void main (void)
{
    TMOD=0x20;           //use Timer 1, 8-BIT auto-reload
    TH1=0XFA;            //4800 baud rate
    SCON=0x50;
    TR1=1;
    while (1)
    {
        SBUF='A';        //place value in buffer
        while (T1==0);
        TI=0;
    }
}
```



10-16) write an 8051 C program to transfer the message “YES” serially at 9600 baud, 8-bit data, 1 stop bit. Do this continuously.

```
#include<reg51.h>

void SerTx (unsigned char);

void main (void)

{
    TMOD=0x20;
    TH1=0XF0;           //9600 baud rate
    SCON=0x50;
    TR1=1;              //start timer
    while (1)
    {
        SerTx ('Y');
        SerTx ('E');
        SerTx ('S');
    }
}

void SerTx (unsigned char x)
{
    SBUF=x;             //place value in buffer
    while (TI==0);      //wait until transmitted
    TI=0;
}
```



10-17) Program the 8051 in C to receive bytes of data serially and put them in P1.

Set the baud rate at 4800, 8-bit data, and 1 stop bit.

```
#include<reg51.h>
void main (void)
{
    unsigned char mybyte;
    TMOD=0x20;           //use Timer 1, 8-BIT auto-reload
    TH1=0xFA;             //4800 baud rate
    SCON=0x50;
    TR1=1;                //start timer
    while (1)              //repeat forever
    {
        while (RI==0);      //wait to receive
        mybyte=SBUF;         //save value
        P1=mybyte;            //write value to port
        RI=0;
    }
}
```



10-18) Write an 8051 C program to send two different strings to the serial port. Assuming that SW is connected to pin P2.0, monitor its status and make a decision as follows:

SW=0: send your first name

SW=1: send your last name

Assume XTAL=11.0592 MHz, baud rate of 9600,8-bit data, 1 stop bit.

```
#include<reg51.h>
sbit MYSW=P2^0;                                //input switch
void main( void)
{
    unsigned char z;
    unsigned char fname [] = "ALI";
    unsigned char lname [] = "SWITCH";
    TMOD=0x20;                                     //use Timer 1, 8-BIT auto-reload
    TH1=0xFD;                                       //9600 baud rate
    SCON=0x50;
    TR1=1;                                         //START TIMER
    if (MYSW==0)                                    //check switch
    {
        for (z=0;z<3;z++)
        {
            SBUF=fname[z];
            while (TI==0);
            TI=0;
        }
    }
    else
    {
        for (z=0 ; z<5 ; z++)                      //write name
        {
            SBUF=lname[z];                         //place value in buffer
            while (TI==0);                          //wait for transmit
            TI=0;
        }
    }
}
```



10.19 Write an 8051 C program to send two messages “Normal Speed” and “High Speed” to the serial port. Assuming that SW is connected to pin P2.0, monitor its status and set the baud rate as follows:

SW=0 28,800 baud rate

SW=1 56K baud rate

Assume that XTAL=11.0592MHz for both cases.

Solution

```
#include<reg 51.h>
sbit MYSW=P2^0;           //input switch
void main(void)
{
    unsigned char z;
    unsigned char mess1 [] = "Normal Speed";
    unsigned char mess2 [] = "High Speed";
    TMOD=0X20;             //use timer1, 8bit auto-reload
    TH1=0xFF;               //28,800 for normal speed
    SCON=0X50;
    TR1=1;                  //start timer
    if (MYSW==0)
    {
        for (z=0;z<12;z++)
        {
            SBUF=Mess1 [z];      //place value in buffer
            while (TI==0);       //wait for transmit
            TI=0;
        }
    }
    else
    {
        PCON=PCON|0X80;        //for high speed of 56K
        for (z=0;z<10;z++)
        {
            SBUF=Mess2 [z];      //place value in buffer
            while (TI==0);       //wait for transmit
            TI=0;
        }
    }
}
```



10.20 Write a C program for the DS894x0 to transfer letter "A" serially at 4800 baud continuously. Use the 2nd serial port with 8 bit data and 1 stop bit. We can only use timer1 to set the baud rate.

Solution

```
#include<reg51.h>
sfr SBUF=0XC1;
sfr SCON=0XC0;
sbit TI1=0XC1;
void main(void)
{
    TMOD=0X20;           //use timer 1 for 2nd serial port
    TH1=0XFA;            //4800 baud rate
    SCON1=0X51;           //use 2nd serial port SCON1 register
    TR1=1;                //start timer
    while (1)
    {
        SBUF1='A';       //use 2nd serial port SBUF1 register
        while (TI1==0);   // wait for transmit
    }
}
```



10.21 Program the DS89C4x0 in C to receive bytes of data serially via the 2nd serial port and put them in P1. Set the baud rate at 9600, 8bit data, and 1 stop bit. Use timer 1 for baud rate generation.

Solution:

```
#include<reg51.h>
sfr SBUF1=0xC1;
sfr SCON=0xC0;
sbit RI1=0xC0;
void main(void)
{
    unsigned char mybyte;
    TMOD=0X20;
    TH1=0XFD;
    SCON1=0X50;
    TR1=1;
    while (1)
    {
        while (RI1==0);
        mybyte = SBUF1;
        P2 = mybyte;
        RI1=0;
    }
}
```



1) Write an 8051 assembly language program to transfer letter "G" serially at 9600 baud rate, continuously.

```
ORG 00H
MOV TMOD, #20H          ; timer 1, mode 2
MOV TH1, #-3             ; 9600 baud rate or FDh
MOV SCON, #50H           ; 8 bits, 1stop, REN enabled
SETB TR1                 ; START TIMER 1

AGAIN:
MOV SBUF, #'G'           ; transfer 'Y' serially
HERE: JNB TI, HERE       ; WAIT FOR TRANSMISSION TO BE OVER
CLR TI                   ; clear TI for next transmission
SJMP AGAIN               ; repeat
END
```

C- Program:

```
#include<reg51.h>
Void main ()
{
    TMOD=0x20;
    TH1=0xFD;
    SCON=0x50;
    TR1=1;
    while (1)
    {
        SBUF = 'G';
        while (TI==0);
        TI=0;
    }
}
```



2) Write an 8051 assembly language program to transfer the message "HELLO" serially at 9600 baud rate, 8-bit data, 1 stop bit continuously.

ORG 00H

MOV TMOD, #20H ; timer 1, mode 2
MOV TH1, #3 ; 9600 baud rate
MOV SCON, #50H ; 8 bits, 1stop, REN enabled
SETB TR1 ; START TIMER 1

START:

MOV A, #'H'
ACALL TRANS
MOV A, #'E'
ACALL TRANS
MOV A, #'L'
ACALL TRANS
MOV A, #'L'
ACALL TRANS
MOV A, #'O'
ACALL TRANS
SJMP START ; repeat

TRANS:

MOV SBUF, A

HERE:

JNB TI, HERE ; WAIT FOR TRANSMISSION TO BE
; OVER
CLR TI ; clear TI for next transmission
RET
END



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

```
ORG 00H  
MOV TMOD, #20H ; timer 1, mode 2  
MOV TH1, #-3 ; 9600 baud rate  
MOV SCON, #50H ; 8 bits, 1stop, REN enabled  
SETB TR1 ; START TIMER 1
```

START:

```
MOV A, #'G'  
ACALL TRANS  
MOV A, #'O'  
ACALL TRANS  
MOV A, #'O'  
ACALL TRANS  
MOV A, #'D'  
ACALL TRANS  
MOV A, #'L'  
ACALL TRANS  
MOV A, #'U'  
ACALL TRANS  
MOV A, #'C'  
ACALL TRANS  
MOV A, #'K'  
ACALL TRANS  
SJMP AGAIN ; repeat
```

-----Serial data transfer subroutine-----

TRANS:

```
MOV SBUF, A
```

HERE:

```
JNB TI, HERE ; WAIT FOR TRANSMISSION TO BE OVER  
CLR TI ; clear TI for next transmission  
RET  
END
```



4. Write an 8051 assembly language program to receive bytes serially with baud rate 9600, 8-bit data & 1 stop bit. Simultaneously send received bytes to port 2.

```
ORG 00H  
MOV TMOD, #20H  
MOV TH1, #FDH  
MOV SCON, #50H  
SETB TR1
```

HERE:

```
JNB RI, HERE  
MOV A, SBUF  
MOV P2, A  
CLR RI  
SJMP HERE  
END
```

5. Write an assembly level program for 8051 to transmit ten bytes of data stored in internal memory serially by selecting proper BAUD rate, data length and stop bit.

```
ORG 00H  
MOV TMOD, #20H ; timer 1, mode 2  
MOV TH1, #-3 ; 9600 baud rate or FDh  
MOV SCON, #50H ; 8 bits, 1stop, REN enabled  
MOV R1, #10 ; initial counter  
MOV R0, #20H ; Initial memory pointer  
SETB TR1 ; START TIMER 1
```

START:

```
MOV A,@R0  
ACALL TRANS  
INC R0  
DJNZ R1, START
```

-----Serial data transfer subroutine-----

TRANS:

```
MOV SBUF, A
```



HERE:

JNB TI, HERE	; WAIT FOR TRANSMISSION TO BE over
CLR TI	; clear TI for next transmission
RET	
END	

6. Write an 8051 assembly language program to transfer letter "H" serially at 9600 baud rate, continuously.

```
ORG 00H
MOV TMOD, #20H           ; timer 1, mode 2
MOV TH1, #3               ; 9600 baud rate or FDh
MOV SCON, #50H            ; 8 bits, 1stop, REN enabled
SETB TR1                 ; START TIMER 1

AGAIN:
MOV SBUF, #'H'            ; transfer 'Y' serially
WAIT: JNB TI, WAIT        ; WAIT FOR TRANSMISSION TO BE OVER
CLR TI                   ; clear TI for next transmission
SJMP AGAIN                ; repeat
END
```

C- Program:

```
#include<reg51.h>
Void main ()
{
    TMOD=0x20;
    TH1=0xFD;
    SCON=0x50;
    TR1=1;
    while (1)
    {
        SBUF = 'H';
        while (TI==0);
        TI=0;
    }
}
```



7. Read port P1 data and transfer this data serially continuously at a baud rate of 4800.

```
ORG 00H  
MOV TMOD, #20H  
MOV TH1, #-6 ; 4800 baud rate or FA  
MOV SCON, #50H  
MOV P1, #0FFH  
SETB TR1
```

START:

```
MOV A, P1  
MOV SBUF, A
```

WAIT:

```
JNB TI, WAIT  
CLR TI  
SJMP START  
END
```

C- Program:

```
#include<reg51.h>  
void main ()  
{  
    unsigned char x;  
    P1=0xFF;  
    TMOD=0x20;  
    TH1=0xFA; ; 4800 baud rate or -6  
    SCON=0x50;  
    TR1=1;  
    while (1)  
    {  
        x=P1; // read port P1  
        SBUF = x; // place in SBUF  
        while (TI==0);  
        TI=0;  
    }  
}
```



8. Consider that a switch SW is connected to pin P2.3. Monitor the SW status and if SW=0: send "HELLO" and if SW=1: send "WORLD" serially. Assume XTAL=11.0592MHz, baud rate of 9600, 8-bit data, and 1 stop bit.

Assembly language program

```
ORG    00
MOV    TMOD, #20H      ;serial port initialization
MOV    TH1, #-3        ;9600 Baud
MOV    SCON, #50H
SETB   P2.3            ;pin as input
SETB   TR1             ;start timer
AGAIN: JNB   P2.3, next ;if P2.3=0 go to next
       MOV   DPTR, #PHIGH ;else display 'world'
       ACALL SEND         ;send subroutine
       SJMP  AGAIN
next:  MOV   DPTR, #PLOW ;display 'hello'
       ACALL SEND
       SJMP  AGAIN
       ;SEND subroutine to display the strings
SEND:  CLR   A           ;points to first character
       MOVC  A, @A+DPTR   ;Get character pointed by DFTR+A
       JZ    END1          ;if A=0, implies string over
       MOV   SBUF, A        ;character to transmit
wait:  JNB   TI, wait    ;wait till transmission over
       CLR   TI
       INC   DPTR          ;next character
       SJMP  SEND
END1:  RET               ;storing of strings in code memory
PHIGH: DB    "WORLD", 0
PLOW:  DB    "HIGH", 0
END
```



C- Program:

```
#include<reg51.h>
Sbit SW=P2^3;
void main()
{
    unsigned char i;
    unsigned char Phigh [ ] = "WORLD"
    unsigned char Plow [ ] = "HELLO"
    TMOD = 0x20;
    TH1=0xFD;
    SCON = 0x50;
    TR1=1;

    while (1)
    {
        if (SW==0)
        {
            for (i=0;i<5;i++)
            {
                SBUF=Plow[i];
                while (TI==0);
                TI=0;
            }
        }
        else
        {
            for (i=0;i<6;i++)
            {
                SBUF=Phigh[i];
                while (TI==0);
                TI=0;
            }
        } //end of else
    } //end of while
} //end of main
```



9. The 8-bit digital output of a signal conditioning circuit is connected to port 2 of the 8051. If the data at P2 is above 0Fh, a message 'HIGH' is to be sent serially; else a message 'FINE' is to be sent. The words 'HIGH' & 'FINE' are burned into program ROM locations.

```
ORG    00
MOV    TMOD, #20H      ;Timer 1, mode 2
MOV    TH1, #-3        ;Baud rate of 9600
MOV    SCON, #50H      ;Serial mode 1, start/stop
                      ;8-bit data
MOV    P2, #0FFH      ;make port P2 as input port
SETB   TR1            ;start timer
AGAIN: MOV   A, P2      ;Read port data
CJNE   A, #0FH, TEST  ;if not equal to 0FH goto test
SJMP   OK              ;if equal goto OK, to display FINE
TEST:  JNC   HIGH         ;if its greater than 0FH, goto HIGH
OK:    MOV   DPTR, #00A0H  ;DPTR points to message FINE,
                      ;(we can as in previous program
ACALL  SEND            ;have a label to 00A0H, say FINEL
                      ;and use
SJMP   AGAIN           ;MOV DPTR, #FINEL instruction)
HIGH:  MOV   DPTR, #0090H  ;ROM location where message 'HIGH'
                      ;is stored
ACALL  SEND            ;SEND subroutine to get message
SJMP   AGAIN           ;from ROM area & transmit serially
SEND:  CLR   A
MOVC   A, @A+DPTR     ;Get character pointed by DPTR+A
JZ    END1             ;Terminate if last character,
                      ;i.e., zero
MOV    SBUF, A          ;load SBUF to transmit
wait:  JNB   TI, wait    ;wait till transmission is complete
CLR   TI               ;clear for next transmission
INC   DPTR             ;point to next character
SJMP   SEND             ;ROM locations to store the
END1:  RET              ;message strings
ORG    00A0H            ;assembler directive ORG places
                      ;the next data in ROM
                      ;location 00A0H
FINEL: DB    ``FINE'', 0  ;FINEL is a label
                      ;(now #FINEL=00A0H)
                      ;(the array size is 5 bytes
                      ;i.e., 4 ASCII values of FINE, +0)
ORG    0090H
MESG: DB    ``HIGH'', 0
END
```



C- Program:

```
#include<reg51.h>
void main()
{
    unsigned char i, temp;
    code unsigned char h [ ] = "HIGH"
    code unsigned char f [ ] = "FINE"
    TMOD = 0x20;
    TH1=0xFD;
    SCON = 0x50;
    P2=0xFF;
    TR1=1;

    while (1)

    {
        Temp = P2;
        if (temp>0x0F)
        {
            for (i=0;i<4;i++)
            {
                SBUF=h [i];
                while (TI==0);
                TI=0;
            }
        }
        else
        {
            for (i=0;i<4;i++)
            {
                SBUF=f [i];
                while (TI==0);
                TI=0;
            }
        }
    }
}
```



10. Write a program to serially transmit the message “HELLO” continuously at a baud rate of 9600, 8-bit data and 1 stop bit.

```
ORG 00H
MOV TH1, #-3
MOV SCON, #50H
SETB TR1
AGAIN: MOV A, #'H'
        ACALL SEND
        MOV A, #'E'
        ACALL SEND
        MOV A, #'L'
        ACALL SEND
        MOV A, #'L'
        ACALL SEND
        MOV A, #'O'
        ACALL SEND
        SJMP AGAIN
SEND:  MOV SBUF, A
WAIT:  JNB TI, WAIT
        CLR TI
        RET
        END
```



C- Program:

```
#include<reg51.h>
void main()
{
    unsigned char i;
    unsigned char msg [ ] = "HELLO"
    TMOD = 0x20;
    TH1=0xFD;
    SCON = 0x50;
    TR1=1;

    while (1)
    {
        for (i=0;i<5;i++)
        {
            SBUF = msg[i];
            while (TI==0);
            TI=0;
        }
    }
}
```



Alternate C program:

```
#include<reg51.h>
void send(unsigned char);
void main()
{
    TMOD = 0x20;
    TH1=0xFD;
    SCON = 0x50;
    TR1=1;

    while (1)
    {
        send ('H');
        send ('E');
        send ('L');
        send('L');
        send ('O');
    }
}

void send (unsigned char value)
{
    SBUF = value;
    while (TI==0);
    TI=0;
}
```



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

```
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
```

C Program:

```
#include<reg51.h>
void main()
{
    unsigned char val, receivedata;
    TMOD = 0x20;
    TH1=0xFD;
    SCON = 0x50;
    TR1=1;

    while (1)
    {
        while (RI==0);
        val = SBUF;
        P2=val;
        receivedata=val;
        RI=0;
    }
}
```



12. Write a program to transfer an ASCII character 'B' with a baud rate of 19,200 with a crystal frequency of 11.0592 MHz.

```
MOV A, PCON      ;Get PCON contents into Accumulator
SETB ACC.7       ;set accumulator's 7th bit
                 ;(=SMOD in PCON)
MOV PCON, A      ;write back to PCON
MOV TMOD, #20H
MOV TH1, #-3
MOV SCON, #50H
SETB TR1
MOV A, #'B'      ;load accumulator with ASCII value 'B'
AGAIN: MOV SBUF, A ;move into SBUF for serial transmission
wait: JNB TI, wait
CLR TI
SJMP AGAIN
```

C program:

```
#include<reg51.h>
void main ()
{
    TMOD = 0x20;
    TH1=0xFD;
    SCON = 0x50;
    TR1=1;
    PCON = PCON | 0x80;

    while (1)
    {
        SBUF = 'B';
        while (TI==0);
        TI=0;
    }
}
```



13. Write a program to generate a square wave on Pin P1.3. Transmit this square wave to receiver connected in serial form.

```
ORG    00
MOV    TMOD, #22H ;Timer 0, 1 in mode 2
MOV    SCON, #50H ;
MOV    TH1, #-3   ;9600 baud
MOV    TH0, #00H  ;square wave freq (minimum)
                ;(delay is maximum)
SETB   TR1        ;start timer1 for serial port
MOV    A, #00      ;Initialize accumulator
CLR    P1.3        ;and pin level
AGAIN: SETB   TR0      ;start timer for square wave
WAIT:  JNB    TF0, WAIT
       CLR    TR0      ;stop timer 0
       CLR    TF0
       CPL    P1.3      ;toggle pin
       CPL    A          ;complement accumulator
       MOV    SBUF, A    ;transmit data
HERE:  JNB    TI, HERE
       CLR    TI
       SJMP  AGAIN
END
```

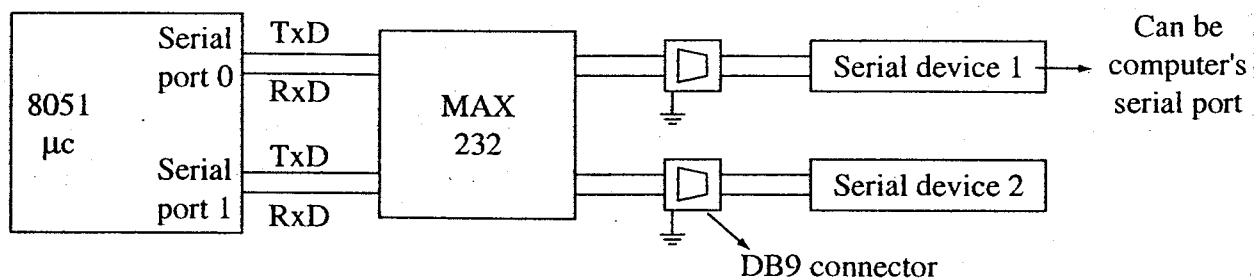
C program:

```
#include<reg51.h>
void main ()
{
    unsigned char I;
    TMOD = 0x22;
    SCON = 0x50;
    TH1=0xFD;
    TH0=00;
    TR1=1;
    i=0xFF;
    P1^3 = 1;
    while (1)
    {
        TR0=1;
```



```
while (TF0==0);  
TR0=0;  
TF0=0;  
  
P1^3 = ~ P1^3;  
i = ~ i;  
SBUF = i;  
while (TI==0);  
TI=0;  
}  
}
```

SECOND SERIAL PORT PROGRAMMING:



8051 connected to MAX232.



14. Write a program to continuously transfer letter 'A' serially using the second serial port of the 8051 at 4800 baud, 8-bit data & 1 stop bit. Use timer 1.

Assembly language program

```
SBUF1 EQU 0C1H ;SFR's byte addresses
SCON1 EQU 0C0H
TI1 BIT 0C1H ;flag's bit address
RI1 BIT 0C0H
ORG 00H ;main program
MOV TMOD, #20H ;timer 1, mode 2
MOV TH1, #-6 ;4800 baud rate
MOV SCON1, #50H ;8-bit, 1 stop bit
SETB TR1 ;start timer 1
AGAIN: MOV A, #'A' ;move character to accumulator
        MOV SBUF1, A ;move to SBUF1 for transmission
HERE: JNB TI1, HERE ;wait till transmission is over
      CLR TI1 ;clear transmit interrupt flag
      SJMP AGAIN
```

C Program:

```
#include<reg51.h>
sfr SBUF1=0XC1;
sfr SCON1=0XC0;
sbit TI1 = 0xC1;
sbit RI1 = 0xC0;
void main ()
{
    TMOD = 0x20;
    TH1=0xFA; // (-6) 4800 Baud rate.
    SCON1=0x50;
    TR1=1
    while (1)
    {
        SBUF1 = 'A';
        while (TI1==0);
        TI1 = 0;
    }
}
```



15. Write a program to receive bytes of data serially through the second serial port & display them in P1. Set the baud rate at 9600, 8-bit data and 1 stop bit. Use timer 1 for baud rate generation.

Assembly language program

```
SBUF1 EQU 0C1H      ;second serial port
SCON1 EQU 0C0H      ;SFR's addresses
RI1 BIT 0C0H
ORG 00H             ;main program
MOV TMOD, #20H
MOV TH1, #-3         ;9600 baud
MOV SCON1, #50H
SETB TR1            ;start timer
HERE: JNB RI1, HERE ;wait till byte is received
      MOV A, SBUF1   ;get data
      MOV P1, A       ;display at P1
      CLR RI1        ;clear flag
      SJMP HERE
END
```

C Program:

```
#include<reg51.h>
sfr SBUF1=0XC1;
sfr SCON1=0XC0;
sbit RI1=0xC0;
void main ()
{
    unsigned char i;
    TMOD = 0x20;
    TH1=0xFD;
    SCON1=0x50;
    TR1=1
    while (1)
    {
        while (RI1==0);
        i=SBUF1;
        P1=i;
        RI1=0;
    }
}
```



```
for (i=0; i<16; i++)  
{  
    SBUF1 = mess[i];  
    while (TI1==0);  
    TI1=0;  
}  
while (1);  
}
```



16. Write a program to transmit the string “beautiful earth” to serial #1.
Set the baud rate at 9600, 8-bit data and 1 stop bit.

Assembly language program

```
SBUF1 EQU 0C1H
SCON1 EQU 0C0H
TF1 BIT 0C1H
ORG 00H
MOV TMOD, #20H
MOV TH1, #-3          ;9600 baud
MOV SCON1, #50H       ;for second serial port
SETB TR1              ; start timer
MOV DPTR, #MESSAGE   ;address of string
AGAIN: CLR A
        MOVC A, @A+DPTR    ;get character
        JZ END1             ;check if '0' end of string
        MOV SBUF1, A         ;move character to SBUF1
                           ;for serial transmission
wait:  JNB TI1, wait      ;wait till complete
        CLR TI1              ;clear flag
        INC DPTR              ;get next character
END1:  SJMP END1           ;wait endlessly
                           ;storing string in memory
MESSAGE: DB "Beautiful Earth", 0
```

C Program:

```
#include<reg51.h>
sfr SBUF1= 0XC1;
sfr SCON1 = 0XC0;
sbit TI1 = 0Xc1;
void main ()
{
    unsigned char i, mess[] = "beautiful earth"; // 15 characters
    TMOD = 0x20;
    TH1=0xFD;
    SCON1=0x50;
    TR1=1
```



17. Write a program to perform the following.

- i) If SW=0, send the message "FINE" to the serial #0 port
- ii) If SW=1, send the message "BEAUTIFUL" to the serial #1 port where SW is a switch connected to pin P2.2

```
SBUF1 EQU 0C1H;
SCON1 EQU 0C0H
TF1 BIT 0C1H
ORG OOH
MOV TMOD, #20H
MOV TH1, #-3      ;9600 baud
MOV SCON, #50H    ;SCON for both serial ports
MOV SCON1, #50H,
SETB P2.2          ;configure for input pin (SW)
SETB TR1           ;start timer
Again: JB P2.2, high ;go to display 'beautiful' if P2.2=1
       MOV DPTR, #DISP1 ;SW=0, hence display fine at serial
                           ;port #0
next:  CLR A
       MOVC A, @A+DPTR ;get value from ROM
       JZ Again          ;if character is zero, break
                           ;the loop (end of string)
       MOV SBUF, A        ;send to serial port #0 for
                           ;transmission
wait:  JNB TI, wait   ;wait if TI=0, i.e., until end of
                           ;transmission
       CLR TI             ;reset flag for next character
       INC DPTR           ;increment DPTR for next consecutive
                           ;address in string
       SJMP next
high:  MOV DPTR, #DISP2
next2: CLR A
       MOVC A, @A+DPTR ;get character from string 2
       JZ Again          ;end of string 2, break
       MOV SBUF1, A        ;transmit using serial port #1
wait1: JNB TI1, wait1 ;wait till transmission is complete
       CLR TI1
       INC DPTR
       SJMP next2
DISP1: DB  "FINE", 0
DISP2: DB  "BEAUTIFUL", 0
END
```



C Program:

```
#include<reg51.h>
sbit SW = P2^2;
sfr SBUF1= 0XC1;
sfr SCON1 = 0XC0;
sbit TI1 = 0Xc1;
void main ()
{
    unsigned char i, mess1[] = "FINE" , mess2 = "BEAUTIFUL";
    TMOD = 0x20;
    TH1=0xFD;
    SCON=0x50;
    SCON1=0x50;
    SW=1;
    TR1=1

    while (1)
    {
        if (SW==0)
        {
            for (i=0;i<4;i++)
            {
                SBUF = mess1 [i];
                while (TI==0);
                TI=0;
            }
        }
        else
        {
            for (i=0;i<9;i++)
            {
                SBUF1 = mess2 [i];
                while (TI1==0);
                TI1=0;
            }
        }
    }
}
```



INTERFACING

Stepper Motor

1. Write a program to rotate stepper motor continuously.

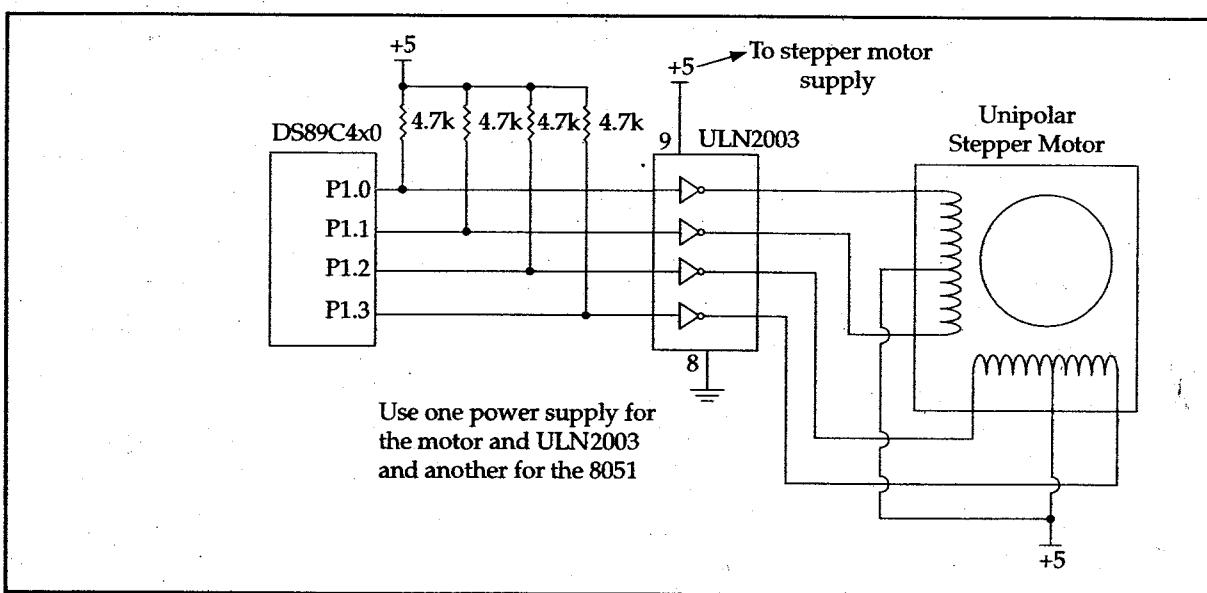


Figure 8051 Connection to Stepper Motor

ORG 00h
MOV A, #66H

UP:

MOV P1, A
RR A
ACALL DELAY
SJMP UP

DELAY:

MOV R2, #100
MOV R3, #255

HERE:

DJNZ R3, HERE



DJNZ R2, HERE

RET

END

OR

ORG 00h

UP:

MOV A, #66H

MOV P1, A

ACALL DELAY

MOV A, #**33** H

MOV P1, A

ACALL DELAY

MOV A, #99H

MOV P1, A

ACALL DELAY

MOV A, #0**CC**H

MOV P1, A

ACALL DELAY

SJMP UP

DELAY:

MOV R2, #100

MOV R3, #255

HERE:

DJNZ R3, HERE

DJNZ R2, HERE

RET

END

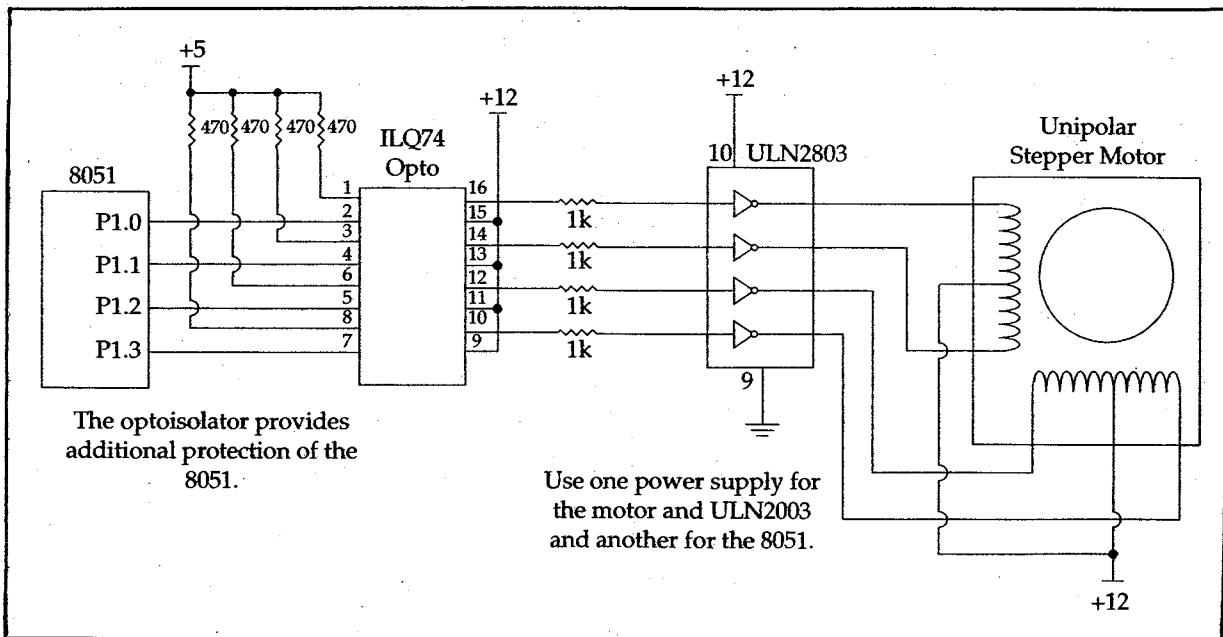


2. Write a program to rotate a motor 64° in the clockwise direction. The motor has a step angle of 2° . Use the 4 step sequence.

Solution:

- ❖ Step angle : 2°
- ❖ One Revolution = 360°
- ❖ Movement per 4-step sequence : $4 \times 2^\circ = 8^\circ$
- ❖ Steps per revolution = $360^\circ / 2^\circ = 180$
- ❖ To rotate a motor 64° , $64^\circ / 8^\circ = 8$ (Counter)

Or $64^\circ / 2^\circ = 32$ (Counter)



Controlling Stepper Motor via Optoisolator

```

ORG 00h
MOV R0, #32
MOV A, #66H

```

UP:

```

MOV P1, A
RR A

```



ACALL DELAY

DJNZ R0, UP

DELAY:

MOV R2, #100

MOV R3, #255

HERE:

DJNZ R3, HERE

DJNZ R2, HERE

RET

END

OR

ORG 00h

MOV R0, #8

UP:

MOV A, #66H

MOV P1, A

ACALL DELAY

MOV A, #**33** H

MOV P1, A

ACALL DELAY

MOV A, #99H

MOV P1, A

ACALL DELAY

MOV A, #**0CCH**



```
MOV P1, A  
ACALL DELAY  
DJNZ R0, UP
```

DELAY:

```
MOV R2, #100  
MOV R3, #255
```

HERE:

```
DJNZ R3, HERE  
DJNZ R2, HERE  
RET  
END
```



C-Program:

```
#include<reg51xd2.h>
```

```
void delay (unsigned char i);
```

```
void main ()
```

```
{
```

```
unsigned char i;
```

```
{
```

```
for(i=0;i<8;i++)
```

```
{
```

```
P1= 0x66;
```

```
delay (1);
```

```
P1= 0x33;
```

```
delay (1);
```

```
P1= 0x99;
```

```
delay (1);
```

```
P1= 0xc0;
```

```
delay (1);
```

```
}
```

```
}
```

```
}
```

```
void delay(unsigned int count)
```

```
{
```

```
unsigned int i,j;
```

```
for (i=0;i<count;i++)
```

```
    for (j=0;j<1275;j++);
```

```
}
```



3. Write a program to rotate a motor 180° in the clockwise direction. The motor has a step angle of 1.8° . Use the 4 step sequence.

Solution:

- ❖ Step angle : 1.8°
- ❖ One Revolution = 360°
- ❖ Movement per 4-step sequence : $4 \times 1.8^\circ = 7.2^\circ$
- ❖ Steps per revolution = $360^\circ / 1.8^\circ = 200$
- ❖ To rotate a motor 180° , $180^\circ / 7.2^\circ = 25$ (Counter)

Or $180^\circ / 1.8^\circ = 100$ (Counter)

```
ORG 00h  
MOV R0, #100  
MOV A, #66H
```

UP:

```
MOV P1, A  
RR A  
ACALL DELAY  
DJNZ R0, UP
```

DELAY:

```
MOV R2, #100  
MOV R3, #255
```

HERE:

```
DJNZ R3, HERE  
DJNZ R2, HERE  
RET  
END
```



OR

ORG 00h

MOV R0, #25

UP:

MOV A, #66H

MOV P1, A

ACALL DELAY

MOV A, #0CCH

MOV P1, A

ACALL DELAY

MOV A, #99H

MOV P1, A

ACALL DELAY

MOV A, #033H

MOV P1, A

ACALL DELAY

DJNZ R0, UP

DELAY:

MOV R2, #100

MOV R3, #255

HERE:

DJNZ R3, HERE

DJNZ R2, HERE

RET

END



C- Program:

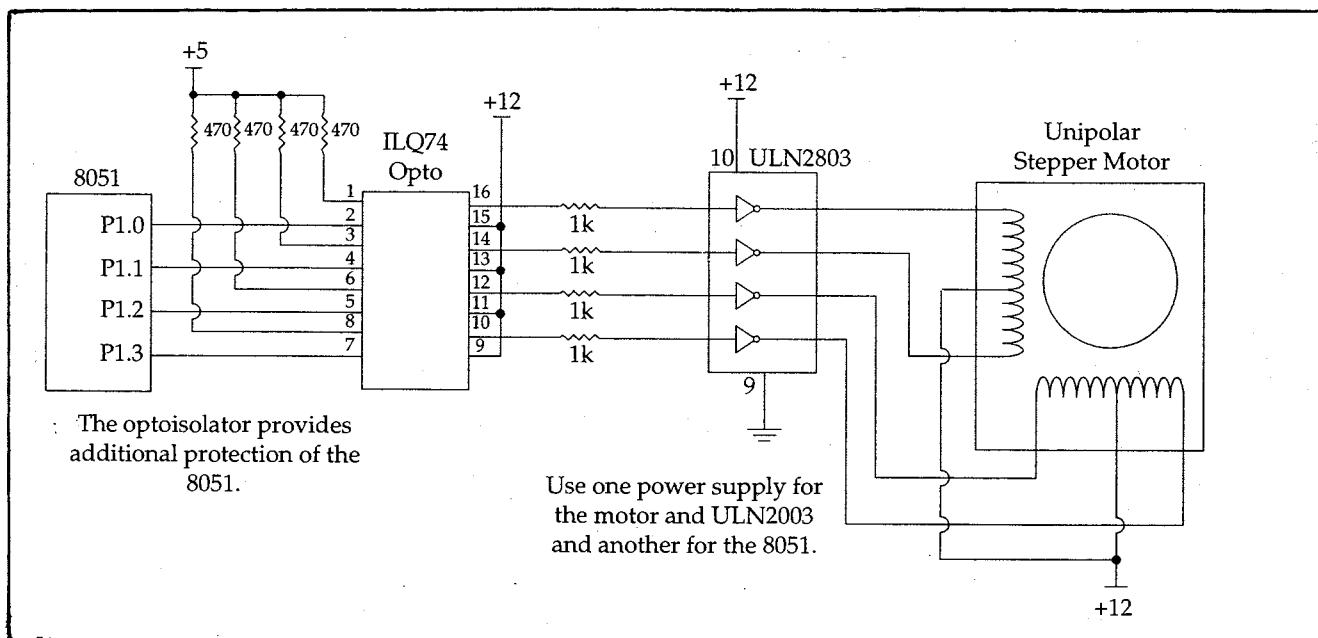
```
#include<reg51xd2.h>
void delay (unsigned char i);
void main ()
{
    unsigned char i;
    {
        for (i=0;i<25;i++)
        {
            P1= 0x66;
            delay (1);
            P1= 0xcc;
            delay (1);
            P1= 0x99;
            delay (1);
            P1= 0x33;
            delay (1);
        }
    }
}

void delay(unsigned int count)
{
    unsigned int i,j;
    for (i=0;i<count;i++)
        for (j=0;j<1275;j++);
}
```



4. A Switch is connected to pin P2.7. Write a program to monitor the status and perform the following:

- a) If SW=0, the stepper motor moves clockwise.
- b) If SW=1, the stepper motor moves counterclockwise.



Controlling Stepper Motor via Optoisolator

ORG 00h

SETB P2.7 // MAKE INPUT PIN

MOV A, #66H

MOV P1, A

UP:

JNB P2.7, CLOCK

RL A

ACALL DELAY

MOV P1, A

SJMP UP

CLOCK:

RR A

ACALL **DELAY**

MOV P1, A

SJMP **UP**

DELAY:

MOV R2, #100

MOV R3, #255

HERE:

DJNZ R3, **HERE**

DJNZ R2, **HERE**

RET

END

OR

ORG 00h

SETB P2.7 // **MAKE INPUT PIN**

UP:

JNB P2.7, **CLOCK**

MOV A, #66H

MOV P1, A

ACALL **DELAY**

MOV A, #33H

MOV P1, A

ACALL **DELAY**



```
MOV A, #99H  
MOV P1, A  
ACALL DELAY
```

```
MOV A, #0CCH  
MOV P1, A  
ACALL DELAY  
SJMP UP
```

CLOCK:

```
MOV A, #66H  
MOV P1, A  
ACALL DELAY
```

```
MOV A, #0CCH  
MOV P1, A  
ACALL DELAY
```

```
MOV A, #99H  
MOV P1, A  
ACALL DELAY
```

```
MOV A, #33H  
MOV P1, A  
ACALL DELAY  
SJMP UP
```



DELAY:

MOV R2, #100

MOV R3, #255

HERE:

DJNZ R3, HERE

DJNZ R2, HERE

RET

END



C-Program:

```
#include <REG51xD2.H>
void delay (unsigned int i);
sbit SW=P2^7;
void main()
{
    SW=1;
    while (1)
    {
        if(SW==0)
        {
            P0=0x66;
            delay (1);
            P0=0xCC;
            delay (1);
            P0=0x99;
            delay (1);
            P0=0x33;
            delay (1);
        }
        else
        {
            P0=0x66;
            delay (1);
            P0=0x33;
            delay (1);
            P0=0x99;
            delay (1);
            P0=0xCC;
            delay (1);
        }
    }
}
```



```
void delay(unsigned int count)
{
    unsigned int i,j;
    for (i=0;i<count;i++)
        for (j=0;j<1275;j++);
}
```

OR

Alternate C- Program:

```
#include <REG51xD2.H>
void delay (unsigned int i);
sbit SW=P2^7;
void main()
{
    unsigned char clock[]={0x66,0xCC,0x99,0x33};
    unsigned char anticlock[]={0x66,0x33,0x99,0xCC};
    SW=1;

    while (1)
    {
        unsigned char i;
        for (i=0;i<4;i++)
        {
            if(SW==0)
            {
                P1=clock[i];
                delay (1);
            }
            else
            {
                P1=anticlock[i];
                delay (1);
            }
        }
    }
}
```



```
void delay(unsigned int count)
{
    unsigned int i,j;
    for (i=0;i<count;i++)
        for (j=0;j<1275;j++);
}
```

5. Rotate the stepper motor continuously

- a) Clockwise using the wave drive 4-step sequence
- b) Clockwise using the half-step 8-step sequence.

Use the sequence values saved in program ROM locations (100H & 200H)

a) Solution

ORG 00h

BACK:

MOV R0, #04
MOV DPTR, #0100H

UP:

CLR A
MOVC A,@A+DPTR
MOV P1, A
ACALL DELAY
INC DPTR
DJNZ R0, UP
SJMP BACK
ORG 0100H
DB 8,4,2,1



DELAY:

MOV R2, #100

MOV R3, #255

HERE:

DJNZ R3, **HERE**

DJNZ R2, **HERE**

RET

END



C- Program:

```
#include<reg51xd2.h>
void delay (unsigned char i);
void main ()
{
    code unsigned char clock[]={8,4,2,1};
    unsigned char i;
    while (1)
    {
        for (i=0;i<4;i++)
        {
            P1= clock[i];
            delay (1);

        }
    }
}

void delay(unsigned int count)
{
    unsigned int i,j;
    for (i=0;i<count;i++)
        for (j=0;j<1275;j++);
}
```



b) Solution:

ORG 00h

BACK:

```
MOV R0, #08  
MOV DPTR, #0200H  
CLR A
```

UP:

```
MOVC A,@A+DPTR  
MOV P1, A  
ACALL DELAY  
INC DPTR  
DJNZ R0, UP  
SJMP BACK
```

ORG 0200H

DB 09, 08, 0CH, 04, 06, 02, 03, 01

DELAY:

```
MOV R2, #100  
MOV R3, #255
```

HERE:

```
DJNZ R3, HERE  
DJNZ R2, HERE  
RET
```

END



C- Program:

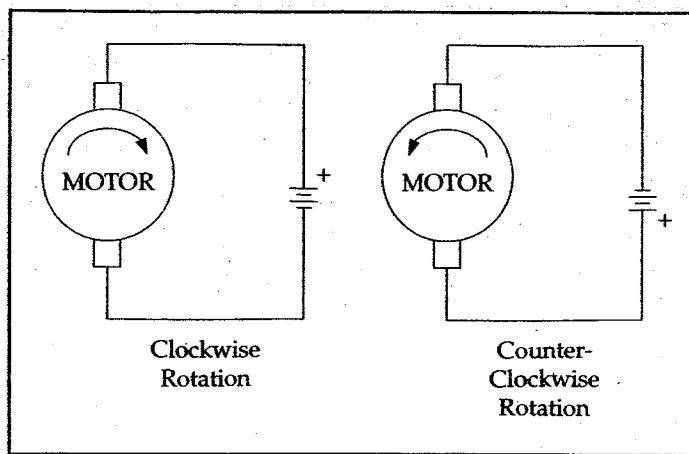
```
#include<reg51xd2.h>
void delay (unsigned char i);
void main ()
{
    code unsigned char clock[]={9,8,0CH,4,6,2,3,1};
    unsigned char i;
    while (1)
    {
        for (i=0;i<8;i++)
        {
            P1= clock[i];
            delay (1);

        }
    }
}

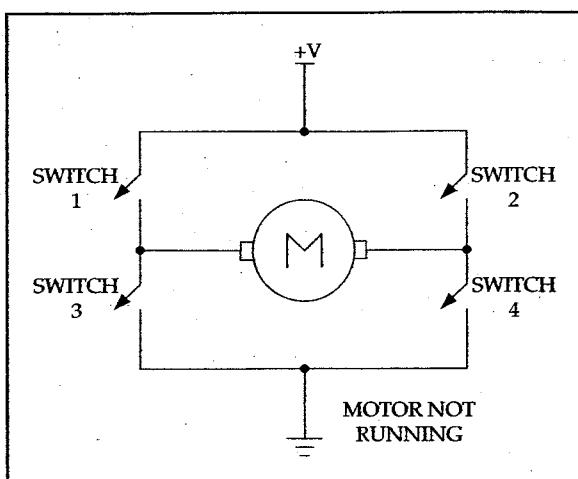
void delay(unsigned int count)
{
    unsigned int i,j;
    for (i=0;i<count;i++)
        for (j=0;j<1275;j++);
}
```



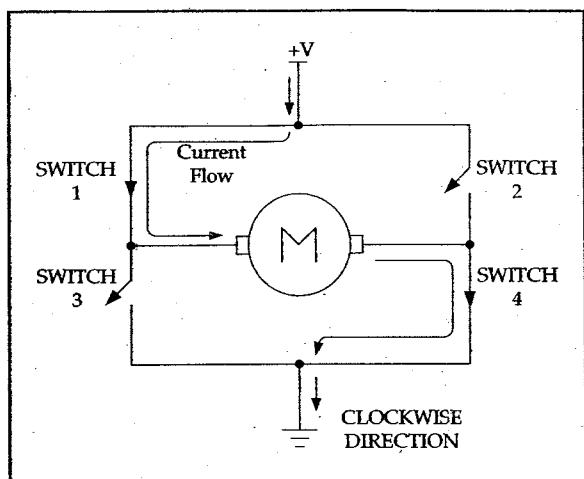
DC Motors



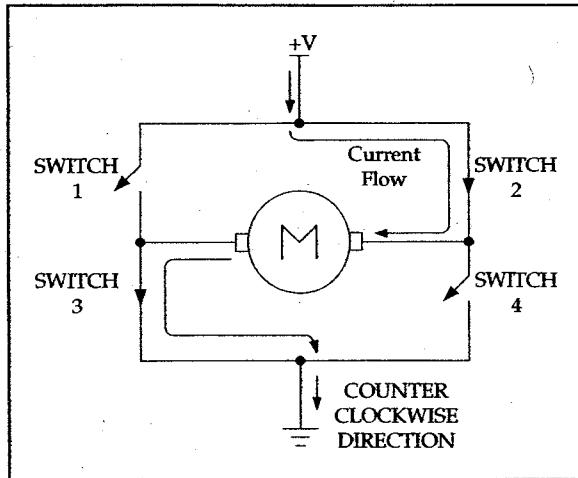
DC Motor Rotation



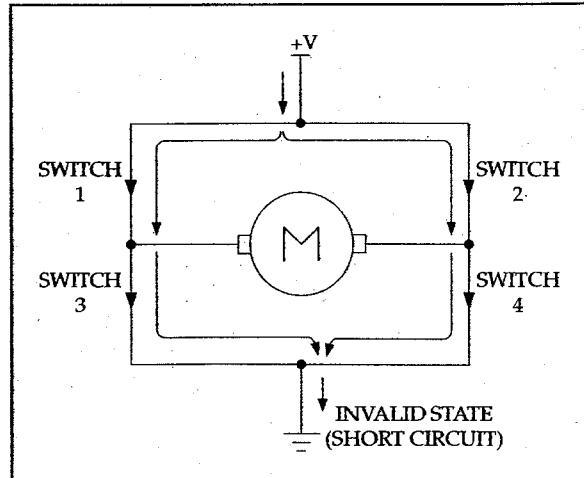
H-Bridge Motor Configuration



H-Bridge Motor Clockwise Configuration



H-Bridge Motor Counterclockwise Configuration



H-Bridge in an Invalid Configuration

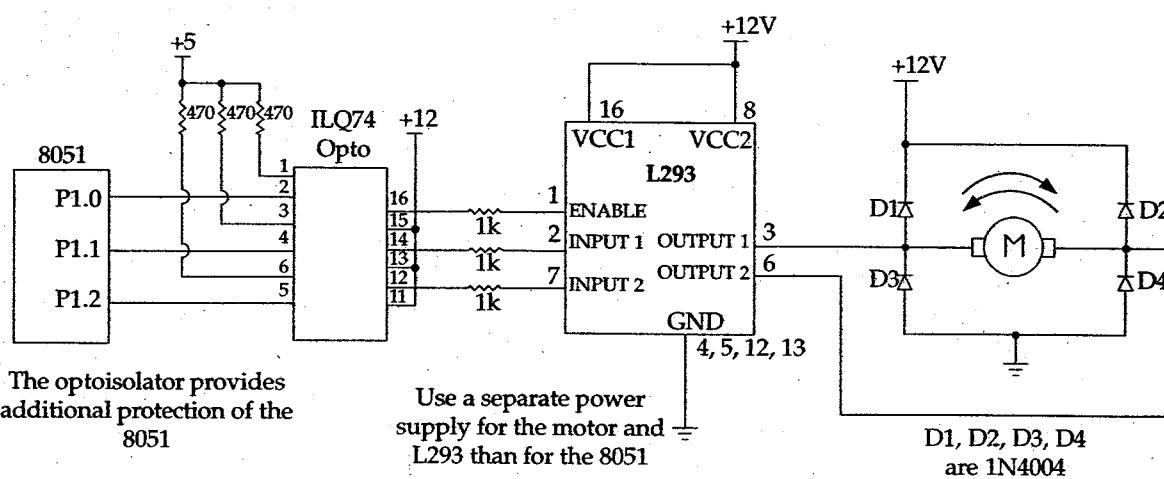
Some H-Bridge Logic Configurations for Figure

Motor Operation	SW1	SW2	SW3	SW4
Off	Open	Open	Open	Open
Clockwise	Closed	Open	Open	Closed
Counter Clockwise	Open	Closed	Closed	Open
Invalid	Closed	Closed	Closed	Closed



1. A switch is connected to pin P2.7. Write a program to monitor the status of SW & perform the following.

- 1) If SW=0, the DC motor moves Clockwise.
- 2) If SW=1, the DC motor moves counterclockwise.



ORG 00h

MAIN:

```

CLR P1.0          ; Switch 1 open
CLR P1.1          ; Switch 2 open
CLR P1.2          ; Switch 3 open
CLR P1.3          ; Switch 4 open
SETB P2.7
    
```

MONITOR:

```

JNB P2.7, CLKWISE
SETB P1.0          ; Switch 1 closed
CLR P1.1          ; Switch 2 open
CLR P1.2          ; Switch 3 open
    
```



SETB P1.3 ; Switch 4 closed

SJMP MONITOR

CLKWISE:

CLR P1.0 ; Switch 1 open

SETB P1.1 ; Switch 2 closed

SETB P1.2 ; Switch 2 closed

CLR P1.3 ; Switch 2 open

SJMP MONITOR

END

C-Program:

```
#include <REG51.H>
sbit SW=P2^7;
sbit ENABLE=P1^0;
sbit clkwise=P1^1;
sbit anticlkwise=P1^2;
void main()
{
    SW=1;
    ENABLE=0;
    clkwise=0;
    anticlkwise=0;

    while(1)
    {
        ENABLE=1;
        if(SW==1)
        {

```

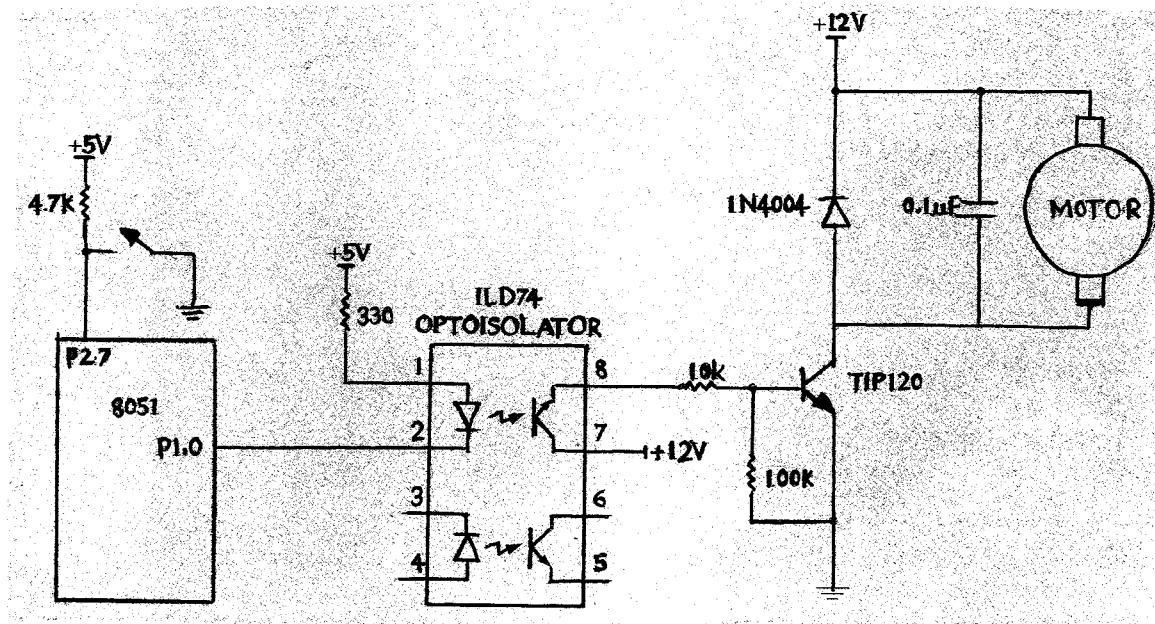


```
clkwise=0;  
anticlkwise=1;  
}  
else  
{  
clkwise=1;  
anticlkwise=0;  
}  
}  
}
```



2. Refer to the figure in this example. Write a C program to monitor the status of SW and perform the following

- If SW=0, the DC motor moves with 50% duty cycle pulse.
- If SW=1, the DC motor moves with 25% duty cycle pulse.



```
#include<reg51.h>
sbit SW=P2^7;
sbit pulse=P1^0;

void delay(unsigned int value);
void main()
{
    SW=1;
    pulse=0;
    while(1)
    {
        if(SW==1)
        {
            pulse=1;
```



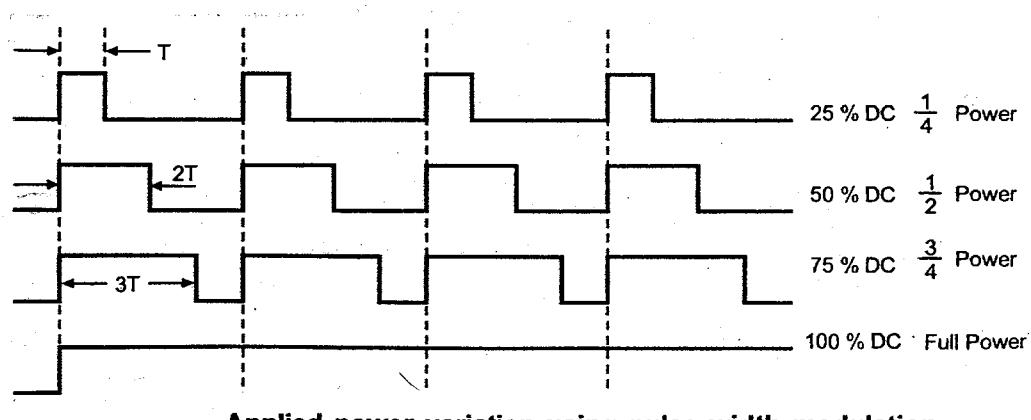
```
delay(25);  
pulse=0;  
delay(75);  
}  
else  
{  
pulse=1;  
delay(50);  
pulse=0;  
delay(50);  
}  
}  
  
void delay( unsigned int count)  
{  
unsigned char i,j;  
for(i=0; i<count;i++)  
    for(j=0;j<1275;j++);  
}
```



3. Refer to figure 17-20 for connection to the monitor. Two switches are connected to pins P2.0 & P2.1. Write a C program to monitor the status of both switches & perform the following:

SW2 (P2.7)	SW1 (P2.6)	Speed of the motor
0	0	DC motor moves slowly (25% duty cycle)
0	1	DC motor moves moderately (50% duty cycle)
1	0	DC motor moves fast (75% duty cycle)
1	1	DC motor moves very fast (100% duty cycle)

Sol:-



$$\text{Duty cycle} = \frac{T_{ON}}{T_{ON} + T_{OFF}}$$



```
#include<reg51.h>

sbit SW2=P2^1;
sbit SW1=P2^0;
sbit pulse=P1^0;
void delay(unsigned int value);
void main()
{
    unsigned char i;
    SW2=1;
    SW1=1;
    pulse=0;
    P2=0xFF;           ; Port 2 used as input port
    i=i&0x03;          ; Masking pins P7-P2(0)

    while(1)
    {
        switch (i)
        {
            case (0):
            {
                pulse=1;
                delay (25);
                pulse=0;
                delay (75);
            }

            case (1):
            {
                pulse=1;
                delay (50);
                pulse=0;
                delay (50);
            }
        }
    }
}
```



```
case (2):
{
    pulse=1;
    delay (75);
    pulse=0;
    delay (25);
}
default:
    pulse=1;
}
}

void delay( unsigned int count)
{
    unsigned char i,j;
    for (i=0; i<count;i++)
        for (j=0;j<1275;j++);
}
```

Without using Switch

```
#include<reg51.h>
sbit SW2=P2^1;
sbit SW1=P2^0;
sbit pulse=P1^0;

void delay(unsigned int value);
void main()
{
    unsigned char i;
    SW2=1;
    SW1=1;
```



```
pulse=0;
while (1)
{
    if (SW2==0 & SW1==0)
    {
        pulse=1;
        delay (25);
        pulse=0;
        delay (75);
    }

    if (SW2==0 & SW1==1)
    {
        pulse=1;
        delay (50);
        pulse=0;
        delay (50);
    }

    if(SW2==1 & SW1==0)
    {
        pulse=1;
        delay (75);
        pulse=0;
        delay (25);
    }

    if (SW2==1 & SW1==1)
        pulse=1;

}

void delay ( unsigned int count)
{
```



```
unsigned char i,j;  
for (i=0; i<count;i++)  
    for (j=0;j<1275;j++);  
}
```



LCD Display:

LCD Command Codes :-

LCD Command Codes

Code (Hex)	Command to LCD Instruction Register
1	Clear display screen
2	Return home
4	Decrement cursor (shift cursor to left)
6	Increment cursor (shift cursor to right)
5	Shift display right
7	Shift display left
8	Display off, cursor off
A	Display off, cursor on
C	Display on, cursor off
E	Display on, cursor blinking
F	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 beginning of 1st line
C0	Force cursor to beginning of 2nd line
38	2 lines and 5x7 matrix

LCD Pin Functions :-

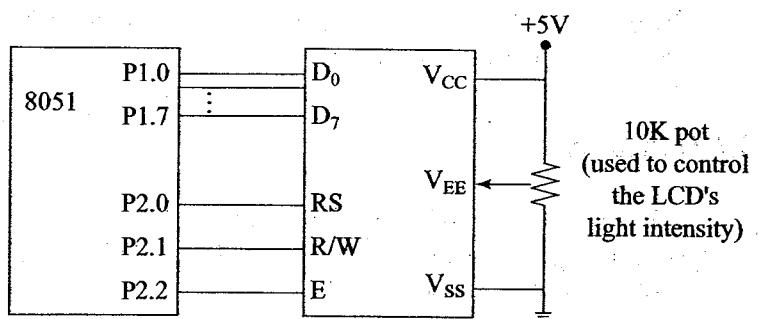
Functions		
Pins	V _{SS} , V _{CC} V _{EE}	Ground, +5V Contrast Control pins
CONTROL PINS	RS R/W E	RS = 0 – select command register RS = 1 – select data register R/W = 0 – write to LCD R/W = 1 – Read from LCD Enable – high-to-low pulse (450 ns wide) to latch data D ₀ –D ₇ into LCD
DATA PINS	D ₀ –D ₇ (bidirectional)	8 data pins, D ₀ –D ₇ , used to send data/command byte to LCD or read from LCD



1. Write An 8051 C Program To Send Letter 'E' & 'C' To LCD Using Delays

```
#include<reg51xd2.h>
sfr ldata = 0x90;
sbit RS  = P2^0;      //Register select
sbit RW  = P2^1;      //Read (1) or Write (0)
sbit E   = P2^2;      //LCD Enable
void lcdcmd (unsigned char i);
void lcddata (unsigned char i);
void delay (unsigned char i);
void main ()
{
    lcdcmd (0x38);
    delay (250);
    lcdcmd (0x0E);
    delay (250);
    lcdcmd (0x01);
    delay (250);
    lcdcmd (0x06);
    delay (250);
    lcdcmd (0x86);
    delay (250);
    lcddata('E');
    delay (250);
    lcddata('C');
    delay (250);
}
```

```
void lcdcmd(unsigned char value)
{
    ldata=value;
    RS=0;
    RW=0;
```



Interfacing of LCD to 8051.



```
E=1;
delay (1);
E=0;
return;
}

void lcddata(unsigned char value)
{
    ldata=value;
    RS=1;
    RW=0;
    E=1;
    delay (1);
    E=0;
    return;
}

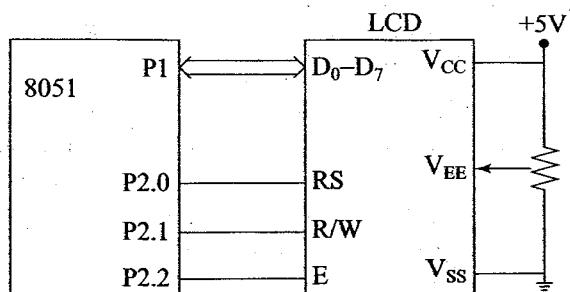
void delay(unsigned int count)
{
    unsigned int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<1275;j++);
}
```

2. Write An 8051 C Program to Send message “HELLO” to LCD without Using delays

```
#include<reg51xd2.h>
sfr ldata = 0x90;
sbit RS  = P2^0;      //Register select
sbit RW  = P2^1;      //Read (1) or Write (0)
sbit E   = P2^2;      //LCD Enable
void lcdcmd (unsigned char i);
void lcddata (unsigned char i);
void delay (unsigned char i);
```



```
void main ()  
{  
    lcdcmd (0x38);  
    lcdcmd (0x0E);  
    lcdcmd (0x01);  
    lcdcmd (0x06);  
    lcdcmd (0x86);  
    lcddata ('H');  
    lcddata ('E');  
    lcddata ('L');  
    lcddata ('L');  
    lcddata ('O');  
}
```



Hardware schematic used to display HELLO on LCD.

```
void lcdcmd(unsigned char value)
```

```
{  
    ldata=value;  
    RS=0;  
    RW=0;  
    E=1;  
    delay (1);  
    E=0;  
    return;  
}
```

```
void lcddata(unsigned char value)
```

```
{  
    ldata=value;  
    RS=1;  
    RW=0;  
    E=1;  
    delay (1);  
    E=0;  
    return;
```

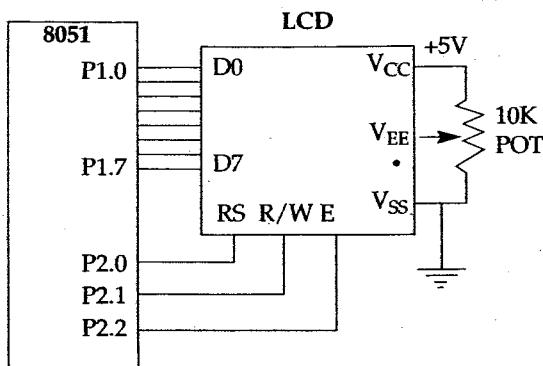


```
}
```

```
void delay(unsigned int count)
{
    unsigned int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<1275;j++);
}
```



3. Write An 8051 C Program to Send Letter 'E' & 'C' TO LCD using BUSY FLAG method.



```
#include<reg51xd2.h>
sfr ldata = 0x90;
sbit RS  = P2^0;      //Register select
sbit RW  = P2^1;      //Read (1) or Write (0)
sbit E   = P2^2;      //LCD Enable
sbit busy = P1^7;
void lcdcmd (unsigned char i);
void lcddata (unsigned char i);
void lcdready ();
void delay (unsigned char i);
void main ()
{
    lcdcmd (0x38);
    lcdcmd (0x0E);
    lcdcmd (0x01);
    lcdcmd (0x06);
    lcdcmd (0x86);
    lcddata ('E');
    lcddata ('C');
}
```



```
void lcdcmd(unsigned char value)
{
```

```
    lcdready();
    ldata=value;
    RS=0;
    RW=0;
    E=1;
    delay (1);
    E=0;
    return;
```

```
}
```

```
void lcddata(unsigned char value)
{
```

```
    lcdready();
    ldata=value;
    RS=1;
    RW=0;
    E=1;
    delay (1);
    E=0;
    return;
```

```
}
```

```
void lcdready()
```

```
{
```

```
    busy=1;
    RS=0;
    RW=1;
    while (busy==1)
    {
        E=0;
```



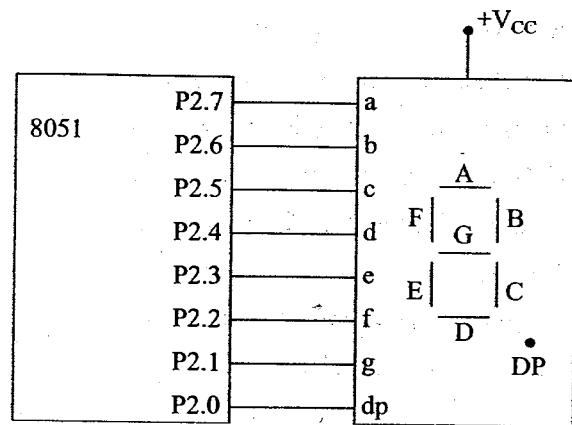
```
delay(1);  
E=1;  
}  
return;  
}  
  
void delay(unsigned int count)  
{  
    unsigned int i,j;  
    for (i=0;i<count;i++)  
        for (j=0;j<1275;j++);  
}
```



LED Display

- Display numbers 0-9 continuously on a 7 segment display connected to 8051 at port P2.

Number	Code	LED Status							
		A	B	C	D	E	F	G	DP
0	03	0	0	0	0	0	1	1	
1	9F	1	0	0	1	1	1	1	
2	25	0	0	1	0	0	1	0	1
3	0D	0	0	0	0	1	1	0	1
4	D9	1	1	0	1	1	0	0	1
5	49	0	1	0	0	1	0	0	1
6	41	0	1	0	0	0	0	0	1
7	1F	0	0	0	1	1	1	1	
8	01	0	0	0	0	0	0	0	1
9	19	0	0	0	1	1	0	0	1



8051 connected to a 7-segment display.

```
#include<reg51xd2.h>
void delay (unsigned char i);
void main ()
{
    Unsigned char displaycode [] = {03, 0x9F, 0x25, 0x0D,
                                    0xD9, 0x49, 0x41, 0x1F, 0x01, 0x19};
    unsigned char i;
    while (1)
    {
        for (i=0;i<10;i++)
        {
            P2= displaycode[i];
            delay (1);
        }
    }
}
```



```
void delay(unsigned int count)
{
    unsigned int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<1275;j++);
}
```

