6

8051 PARALLEL I/O PORTS

Learning Objectives

After you have completed this chapter, you should be able to

- Explain the data transfer synchronisation methods between the CPU and I/O interface
- Explain 8051 parallel I/O ports
- Input data from simple switches and output data to light emitting diodes (LED)
- Input data from matrix keyboard and output data to seven segment display
- Input data from matrix keyboard and output data to LCD display
- Use D/A converter to generate digital waveforms
- Interface multi channel 8 bit A/D converter to the 8051
- Interface serial A/D converter to the 8051
- Explain the operation and interfacing of stepper motor
- Explain the operation and interfacing of DC motor

6.1 BASIC I/O CONCEPTS

This chapter explores interfacing of peripheral devices, through which one interacts with a microcontroller. Examples of peripheral devices include switches, light emitting diodes, printers, keyboards, and liquid crystal displays. The speed and characteristics of these devices are different from that of the processor, so they cannot be connected to the processor directly. Hence, interface chips are needed to resolve the difference between processor and peripheral devices. The main function of an interface chip is to synchronize data transfer between the processor and an I/O device.

In case of input operation, the input devices place data in the data register of the interface chip and then, the processor reads the data. In the output operation, the processor writes data into the data register in the interface chip and the data register holds data until the output device fetches this data. An interface chip has data pins that are connected to the processor data bus, and I/O port pins that are connected to the I/O device. Data transfer between I/O devices takes place in serial (bits by bits) or in parallel (multiple bits). Data is generally transferred serially, in long distance communication and low speed devices. High speed I/O devices mainly use parallel data transfer. We will discuss only interfacing of parallel I/O in this chapter.

6.2 PORT STRUCTURES AND OPERATION

The 8051 has 32 I/O pins that are further divided into four ports—port 0, port 1, port 2 and port 3. These I/O pins allow the 8051 to monitor and control other devices. Port 0, port 2 and port 3 pins are multiplexed with an alternate function. Each port consists of a D latch, an output driver and an input buffer. To access external program and data memory the output driver of port 0 and port 2, and the input buffer of port 0 are used. In this application, low order address lines (A_7–A_0) and data bus (D_7–D_0) are multiplexed on port 0. Port 2 outputs the high-order address lines (A_{15}–A_8). Port 3 pins are used as external interrupt, for serial transfer, timer and external memory control signals as listed in Table 6.1. Port 1 pins have no dual functions. Port 3 is basically same as port 1 except that its, extra circuit allows dual functions. The output drivers of port 0 and port 2 are switchable to internal address/data (AD_0–AD_7) and address bus (A_8–A_{15}) respectively by an internal control signal during access of external memory. Ports 1, 2 and 3 have internal pull ups and port has open drain output. Open drain in Metal Oxide Semiconductors (MOS) means open collector in Transistor Transistor Logic (TTL).

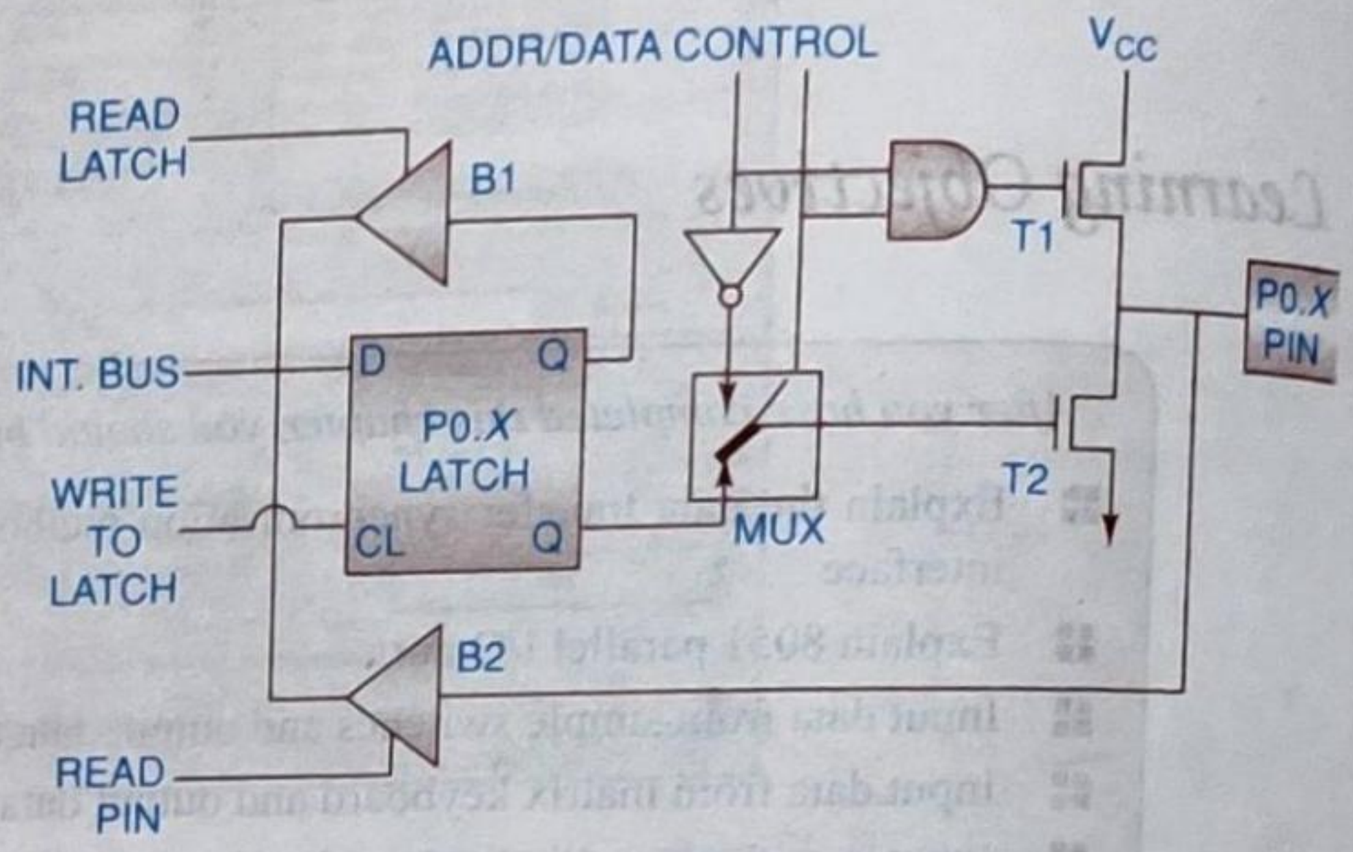

Figure 6.1 *PORT 0*

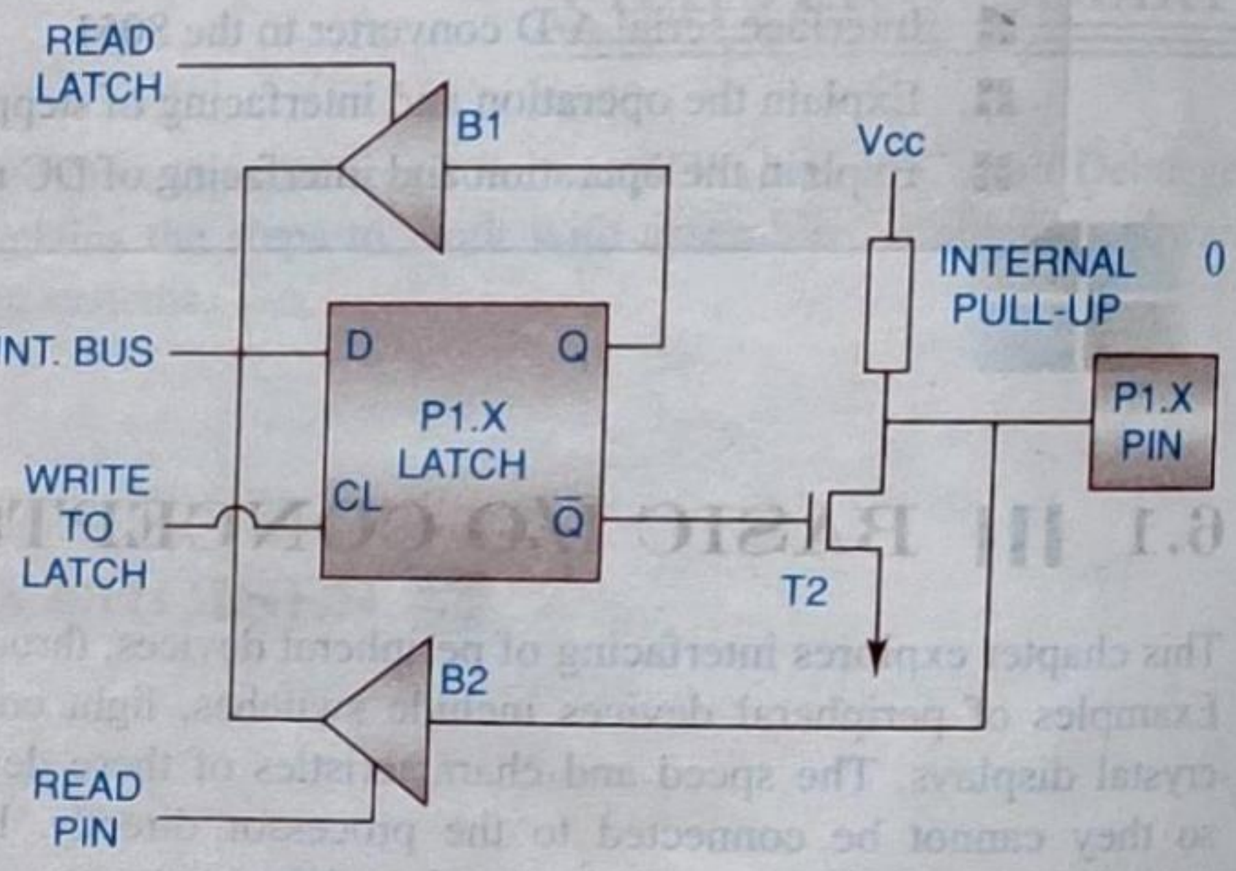

Figure 6.2 *PORT 1*

Input/Output Ports

All 8051 ports have both D-latch and buffer as shown in Fig. 6.1 to 6.4. All the ports can be defined as input or output port. Port 0 has no internal pull up resistors (as shown in Fig. 6.1), it is

simply an open drain. Now by writing a '1', to corresponding port 0 latch, both the transistors are OFF and that causes the pin to float in a high impedance state. When port 0 is used for simple data I/O, then external pull up resistor is required.

Output port All the bits of port register are physically connected with data bus. When the data is written to port register, the data bit is latched to the D flip flop in the port circuit. The processor generates $\overline{WR}$ signal, when an instruction moves the data to the port. $\overline{WR}$ signal is connected as active low clock of the D flip flop. The data from the data bus line is latched to the D flip-flop. If input is 0 (D = 0), then Q = 0 and $\overline{Q}$ = 1 and transistor T2 is on and the condition of the output pin is 0. If input is 1 (D = 1), then Q = 1 and $\overline{Q}$ = 0 and transistor T2 is off and the condition of the output pin is 1.

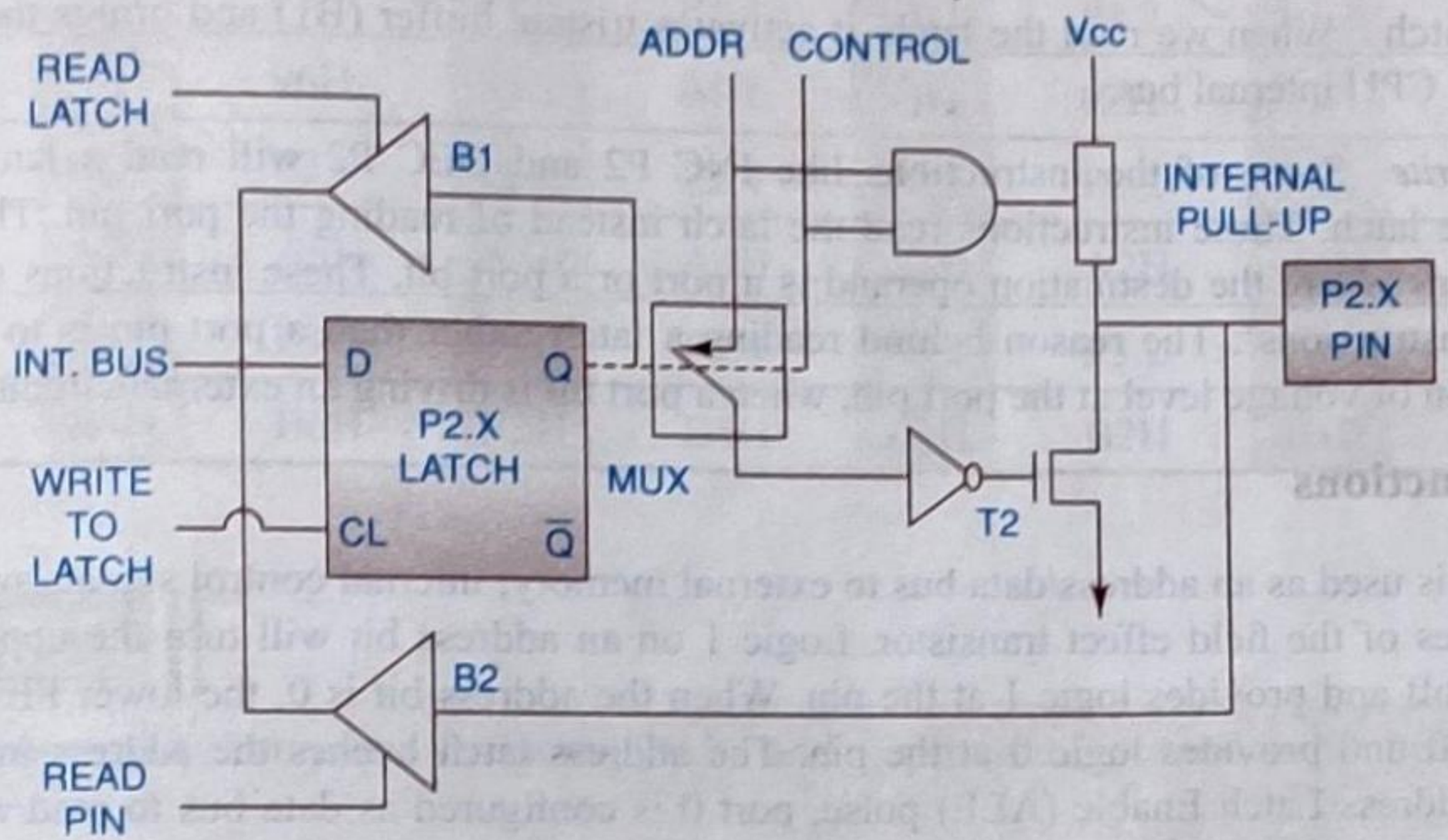

Figure 6.3 *PORT 2*

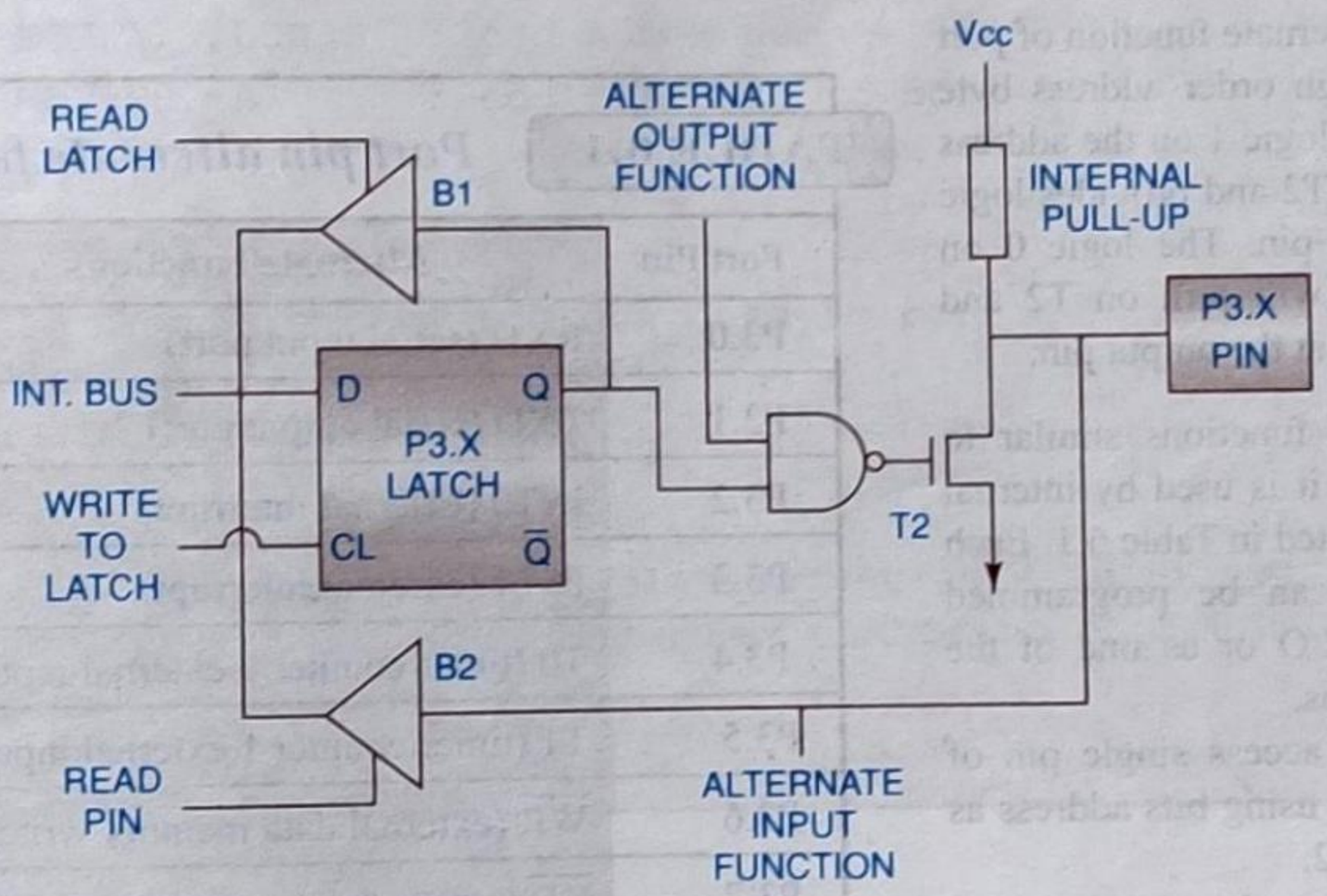

Figure 6.4 *PORT 3 (Courtesy Intel)*

Input port While reading the ports, there are two possibilities:

1. Reading the input pin
2. Reading the latch

Reading the input pin To define any port of the 8051 as input, we must first write '1' to that port bit. By writing '1' to that port bit, Q = 1 and $\overline{Q} = 0$. In port 0 and port 1, Q is connected to the transistor gate (T2); in port 2, Q is connected through not gate to T2; and in port 3, Q is connected through nand gate to T2. In all the cases, the transistor (T2) is turned off, the pin is simply pulled high by the pull up resistors, and connected to the input buffer (B2). When we read the input port, it activates B2 and brings the data from the pin into the CPU internal bus.

Reading the latch When we read the latch, it activates tristate buffer (B1) and brings the data from the Q latch into the CPU internal bus.

Read modify write Some of the instructions like INC P2 and DEC P2 will read a latch, modify and rewrite it to the latch. These instructions read the latch instead of reading the port pin. This happens for those instructions where the destination operand is a port or a port bit. These instructions are called 'read modify write instructions'. The reason behind reading a latch rather than a port pin is to avoid possible misinterpretation of voltage level at the port pin, when a port bit is driving an external circuit.

Alternate Functions

Port 0 Port 0 is used as an address/data bus to external memory; internal control signals switch the AD0–AD7 to the gates of the field effect transistor. Logic 1 on an address bit will turn the upper FET on and the lower FET off and provides logic 1 at the pin. When the address bit is 0, the lower FET is on and the upper FET is off and provides logic 0 at the pin. The address latch latches the address into the external circuit. After Address Latch Enable (ALE) pulse, port 0 is configured as data bus to read/write data from the external memory. When Port 0 is configured as an input, logic 1 is automatically written by the internal control logic to all port 0 latches.

Port 2 The alternate function of port 2 is to send high order address byte (A8–A15). The logic 1 on the address bit will turn off T2 and provides logic 1 at the output pin. The logic 0 on the address bit will turn on T2 and provides logic 0 at the output pin.

Port 3 Port 3 functions similar to port 1 and also, it is used by internal peripherals as listed in Table 6.1. Each pin of port 3 can be programmed individually as I/O or as one of the alternate functions.

We can also access single pin of all the four ports using bits address as given in Table 6.2.

As discussed under Section 1.6, the drawback in assembly language

TABLE 6.1 *Port pin alternate functions*

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt)
P3.3	INT1 (external interrupt)
P3.4	T0 (timer/counter 0 external input)
P3.5	T1 (timer/counter 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

programming is that it is difficult and time consuming to write a program. Writing programs in a high level language such as C is easier. For interfacing I/O devices, programs are given both in assembly language and C. We will discuss few examples to become familiar with the syntax.

TABLE 6.2 *Bit address of the ports*

Port 0	P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	P0.0
Address	87H	86H	85H	84H	83H	82H	81H	80H
Port 1	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
Address	97H	96H	95H	94H	93H	92H	91H	90H
Port 2	P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0
Address	A7H	A6H	A5H	A4H	A3H	A2H	A1H	A0H
Port 3	P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0
Address	B7H	B6H	B5H	B4H	B3H	B2H	B1H	B0H

EXAMPLE 6.1

Write 8051 Assembly Language Program (ALP) and C program to send 8 bit hex number to port 0.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV A, #0CH     ;Load A hex number 0C
        MOV P0, A       ;Send the data to port 0
        END
```

C PROGRAM

```
// C program to send hex data to port 0
#include <Intel\8051.h>
void main (   )
     {
 unsigned char a;
     a=0x0c;    //Initialise a variable to hex value
     P0=a;           //Send hex number to port 0
     }
```

EXAMPLE 6.2

Write 8051 ALP and C program to send 8 bit binary numbers from 0 to 9 to port 2, repetitively.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
   S1:MOV A, #00H      ; Load A 00H
      MOV R1, #0AH     ; Load R1 0AH
START: MOV P2, A       ; Send the contents of accumulator to P2
      INC A            ; Increment A
      CALL DELAY       ; Call delay routine
      DJNZ R1, START   ; Decrement. R1,if it is not zero, branch to
                         START
      SJMP S1          ; Jump to S1
DELAY: MOV R2,#100     ; Delay routine for 100 ms
LOOP2: MOV R1,#125
LOOP1: PUSH ACC
  POP  ACC
  NOP
  NOP
  DNZ  R1, LOOP1
  DNZ  R2, LOOP2
  RET
  END
```

C PROGRAM

```
// C program to send 0-9 to port2
#include <Intel\8051.h> // Include header file of 8051
#include<standard.h>    // For delay routine
#define period 100      // 100 ms
void main  (   )
    {
unsigned char a;
while(1)     // Always perform
    {
for(a=0; a<10; a++) // Initialise variable a=0 and increment
        {
        P2=a;                  // Send 8-bit number to port 2
        delay_ms(period); // Delay routine for 100 ms
        }
}
}
```

EXAMPLE 6.3

Write 8051 ALP and C program to toggle 8 bit of port 1.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV A, #00H          ; Load A 00H
START:  MOV P1, A            ; Send the contents of A to P1
        CPL A                ; Complement contents of A
        CALL DELAY           ; Call delay routine
        AJMP START           ; Branch to start
DELAY:  MOV R2,#100          ; Delay routine for 100 ms
LOOP2:  MOV R1,#125
LOOP1:  PUSH ACC
        POP ACC
        NOP
        NOP
        DNZ R1, LOOP1
        DNZ R2, LOOP2
        RET
        END
```

C PROGRAM

```
// C program to toggle LED's at port 1
#include <Intel\8051.h>
#include<standard.h>        // For delay routine
#define period 100          // 100 ms

void main (   )
      {
      while(1)                          // Always perform
            {
            P1=0xFF;                    // To send 1s to port1
            delay_ms(period);
            P1=0x00;                    // To send 0s to port 1
            delay_ms(period);
            }
}
```

EXAMPLE 6.4

Write 8051 ALP and C program to toggle alternate bits at port 1.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV A, #0AAH        ;Load A AAH
START:  MOV P1, A           ;Send the contents of A to P1
        CPL A               ;Complement contents of A
        CALL DELAY          ;Call delay routine
        AJMP START          ;Branch to start
DELAY:  MOV R2,#100         ;Delay routine
LOOP2:  MOV R1,#125
LOOP1:  PUSH ACC
        POP ACC
        NOP
        NOP
        DNZ R1, LOOP1
        DNZ R2, LOOP2
        RET
        END
```

C PROGRAM

```
// C program to toggle alternate bits of port 1
#include <Intel\8051.h>
#include<standard.h>  // For delay routine
#define period 100    // 100 ms
void main ( )
    {
    while(1)          // Always perform
          {
          P1=0x55;          // To send 01010101 to port 1
          delay_ms(period);
          P1=0xAA;          // To send 10101010 to port 1
          delay_ms(period);
          }
}
```

EXAMPLE 6.5

Write 8051 ALP and C program to toggle MSB bit of port 1.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        CLR P1.7            ;Clear the port 1.7
START:  CALL DELAY          ;Call delay routine
        CPL P1.7            ;Complement contents of port 1.7
        AJMP START          ;Branch to START
DELAY:  MOV R2,#100         ;Delay routine
LOOP2:  MOV R1,#125
LOOP1:  PUSH ACC
        POP ACC
        NOP
        NOP
        DNZ R1, LOOP1
        DNZ R2, LOOP2
        RET
        END
```

C PROGRAM

```
// C program to toggle MSB bit of port 1
#include <Intel\8051.h>
#include<standard.h>          // For delay routine
#define period 100            // 100 ms
BIT disp1 P1.7          // Identify port P1.7 as disp1
void main (   )
   {
   while(1)                        // Always perform
          {
          disp1=1;                 // Send 1 to port P1.7
          delay_ms(period);
          disp1=0;                 // Send 0 to port P1.7
          delay_ms(period);
          }
}
```

EXAMPLE 6.6

Write 8051 ALP and C program to toggle LSB bit of port 0.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        CLR P0.0            ;Clear the port 0.0
START:  CALL DELAY          ;Call delay routine
        CPL P0.0            ;Complement contents of port 0.0
        AJMP START          ;Branch to START
DELAY:  MOV R2,#100         ;Delay routine
LOOP2:  MOV R1,#125
LOOP1:  PUSH ACC
        POP ACC
        NOP
        NOP
        DNZ R1, LOOP1
        DNZ R2, LOOP2
        RET
        END
```

C PROGRAM

```
// C program to toggle LSB bit of port 0
#include <Intel\8051.h>
#include<standard.h>      // For delay routine
#define period 100        // 100 ms
BIT disp1 P0.0            // Declare port P0.0 as disp1
void main ( )
    {
  while(1)                // Always perform
        {
        disp1=1;          // Send 1 to port P0.0
        delay_ms(period);
        disp1=0;          // Send 0 to port P0.0
        delay_ms(period);
        }
}
```

EXAMPLE 6.7

Write 8051 ALP and C program to left shift data at port 1 repetitively.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV A, #01H          ;Load A 01H
        MOV R1, #07H         ;Load R1 07H
START:  MOV P1, A            ;Send the contents of A to P1
        RL A                 ;Rotate contents of A left by one position
        CALL DELAY           ;Call delay routine
        DJNZ R1, START       ;Branch to START
        AJMP S1              ;Branch to S1
DELAY:  MOV R2,#100          ;Delay routine
LOOP2:  MOV R1,#125
LOOP1:  PUSH ACC
        POP ACC
        NOP
        NOP
        DNZ R1, LOOP1
        DNZ R2, LOOP2
        RET
        END
```

C PROGRAM

```
// C program to left shift the data at port 1
#include <Intel\8051.h>
#include<standard.h>          // For delay routine
#define period 100            // 100 ms
void main ( )
{
unsigned char x;
P1=0x01;                      // Send a number to port 1
while(1)                      // Always perform
        {
for(x=0;x<8;x++)
        {
        P1=P1<<1;   // Shift data at port 1 to left by one bit.
        delay_ms(period);
        }
}
}
```

EXAMPLE 6.8

Write 8051 ALP and C program to receive 8 bits data from port P0 and P1. Perform AND operation of the received data and send the result to port P2.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV P0, #0FFH       ;Define port 0 as input
        MOV P1, #0FFH       ;Define port 1 as input
        MOV A, P1           ;Read data from port1
        MOV R1, A           ;Copy contents of A to R1
        MOV A, P0           ;Read data from port 0
        ANL A, R1           ;or cont. of A with R1
        MOV P2,A            ;send the contents of A to P2
        END
```

C PROGRAM

```
        // C program to AND 8 bits data of port 0 and port 1 and to
        send result to port 2
        #include <Intel\8051.h>
        unsigned char input1;
        unsigned char input2;
        unsigned char result;
        void main (   )
        {
        while(1)                        // Do always
        {

        input1=P1;                      //Read 8 bit data at port P1
        input2=P0;                      //Read 8 bit data at port P0
        result=input1 & input2;         //AND operation
        P2=result;                      //Send result to P2
        }

        }
```

EXAMPLE 6.9

Write 8051 ALP and C program to receive 1 bit data from port P0.0 and P1.3. Perform AND operation of the received bits and send the result to port P2.0.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV P0, #0FFH       ;Define port 0 as input
```

```
MOV P1, #0FFH          ;Define port 1 as input
MOV A, P1              ;Read data from port1
RR A                   ;Rotate contents of A, 3 times right
RR A
RR A
MOV R1, A              ;Copy contents of A in R1
MOV A, P0              ;Read data from port 0
ANL A, R1              ;Perform AND operation
MOV P2, A              ;Send the result to P2.0
END
```

C PROGRAM

```
//C program AND P0.0 and P1.3 send result at P2.0
#include <Intel\8051.h>
BIT input1 P0.0        //Declare port P0.0 as input1
BIT input2 P1.3        //Declare port P1.3 as input2
BIT output P2.0        //Declare port P2.0 as output
void main ( )

{
while(1)               // Always perform
      {
      output=input1 & input2;  // AND operation
      }
}
```

EXAMPLE 6.10

Write 8051 ALP and C program to receive 1 bit data from port P0.0 and P1.2. Perform OR operation of the received bit and send the result to port P2.7.

ASSEMBLY LANGUAGE PROGRAM

```
ORG 0000H
MOV P0, #0FFH      ;Define port 0 as input
MOV P1, #0FFH      ;Define port 1 as input
MOV A, P1          ;Read data from port 1
RR A               ;Rotate contents of A right by two positions
RR A
MOV R1, A          ;Copy contents of A to R1
MOV A, P0          ;Read data from port 0
```

```
        ORL A, R1               ;Perform OR operation
        RR A                    ;Rotate right to get LSB in MSB
        MOV P2, A               ;Send the result to P2.7
        END
```

C PROGRAM

```
// C program OR P0.0 and P1.2 send result at P2.7
#include <Intel\8051.h>
BIT  input1 P0.0        //Declare port P0.0 as input1
BIT input2 P1.2         //Declare port P1.2 as input2
BIT output P2.7         //Declare port P2.7 as output

void main ( )
        {
        while(1)
                {
                output=input1|input2;
                }
        }
```

EXAMPLE 6.11

Write 8051 ALP and C program to send ASCII character data ('A', '@', '!', '*') through port P_0, P_1, P_2 and P_3 respectively.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV A, #41H         ;Load ASCII code for A  in hex 41H to A
        MOV P0, A           ;Send the data to port 0
        MOV A, #40H         ;Load ASCII code for @  in hex 40H to A
        MOV P1, A           ;Send the data to port 1
        MOV A, #21H         ;Load ASCII code for ! in hex 21H to A
        MOV P2, A           ;Send the data to port 2
        MOV A, #2AH         ;Load ASCII code for * in hex 2AH to A
        MOV P3, A           ;Send the data to port 3
        END
```

C PROGRAM

```
// C program to send ASCII values of the characters
#include <Intel\8051.h>
const unsigned char input[4]={ 'A', '@', '!', '*'}; //
declare ASCII data array
```

```
void main (  )

{
P0=input [0];          //Send ASCII value to P0
P1=input [1];          //Send ASCII value to P1
P2=input [2];          //Send ASCII value to P2
P3=input [3];          //Send ASCII value to P3
}
```

EXAMPLE 6.12

Write 8051 ALP and C program to read port 1. If the received data is equal to 20H, send FFH to port 2, otherwise send 00H to port 3.

ASSEMBLY LANGUAGE PROGRAM

```
        ORG 0000H
        MOV P1, #0FFH       ;Define port 1 as input
        MOV A, P1           ;Read data from port 1
        CLR C               ;Clear carry flag
        SUBB A, #20H        ;Compare (A) with 20H using sub operation
        JZ LOOP1            ;If data is equal, branch to Loop1
        MOV A, #00H         ;If data is not equal, sends 00 to P3
        MOV P3, A
        SJMP LOOP2          ;Branch to Loop2
LOOP1:  MOV A, #0FFH        ;Send FF to port2
        MOV P2, A
LOOP2:  NOP
        END
```

C PROGRAM

```
// C program to read port 1 and send data to port 2 and port 3
#include <Intel\8051.h>
unsigned char a;
unsigned char b=0x00;          //Declare variable b=00H
unsigned char c=0xFF;          //Declare variable c=FFH

void main (  )
```