8051 Programming in C

Objectives:

At the end of this chapter we will be able to:

- List and define the C data types for 8051
- Comprehend and write 8051 C programs for
  - Time Delay and I/O Operations
  - I/O Bit Manipulation
  - Logic and Arithmetic Operations
  - ASCII & BCD Data Conversions
  - Binary (hex) to Decimal Conversion
  - To use the 8051 Code Space
  - Data Serialization

Note: All the C programs can be simulated using keil software.

Why Program 8051 in C?

Compilers produce hex file that is downloaded to ROM of microcontroller. The size of hex file generated by the compiler is the main concern, for two reasons:

- Microcontrollers have limited on-chip ROM
- Code space for 8051 is limited to 64K bytes

The language produces a hex file that is much smaller than C. But when both are compared, programming in assembly is tedious and time consuming while C programming is less time consuming, but has larger hex file size.

The reasons for writing programs in C instead of assembly are:

- It is easier and less time consuming to write in C than Assembly
- C is easier to modify and update
- You can use code available in function libraries
- C code is portable to other microcontroller with little or no modification

Data types and Time delay in 8051 C:

In this section we will first discuss the different data types used in C for 8051 programming and the provide codes for time delay function.

C data types for 8051:

A good understanding of C data types for 8051 can help programmers to create smaller hex files. The different data types are:

- unsigned char
- signed char
- unsigned int
- signed int
- sbit (single bit)
- bit and sfr

unsigned char:

As 8051 is an 8-bit microcontroller, the character data type is the most natural choice for many applications. The unsigned char is an 8-bit data type takes a value in the range of 0 – 255 (00H – FFH) and the most widely used data type for 8051. The unsigned char data

type can be used in situations such as setting a counter value and ASCII characters, instead of signed char.
It is important to specify the keyword unsigned infront of the char else compiler will use the signed char as the default. As 8051 has a limited number of registers and data RAM locations, using the int in place of char can lead to a larger size hex file. This is not a significant issue in high level language.

Writing a simple C program:

The first line in an 8051 C program is #include <reg51.h>. The library file reg51.h contains the definition of all the special function registers and their bits. Let us write and examine the usage of the unsigned char data type through some examples.

Example 1: Write an 8051 C program to send values 00H -0AAH to port P1
Program:

```
#include <reg51.h>              // contains all the port registers and internal
                               // RAM of 8051declared
 void main (void)
 {
         unsigned char z;       // z is allotted a space of 1 byte
         for (z=0; z<=170;z++)
         P1=z;                  // values 00H-0FFH sent to port 1
}
```

Note: On executing the program the values 00H to 0AAH will be displayed on port 1

Example 2: Write an 8051 C program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C, and D to port P1.
Solution:

```
#include <reg51.h>
void main(void)
{
        unsigned char mynum[]="012345ABCD";
        unsigned char z;
        for (z=0;z<=10;z++)
        P1=mynum[z];
}
```

Example 3: Write an 8051 C program to toggle all the bits of P1 continuously.
Solution:

```
//Toggle P1 forever
#include <reg51.h>
void main(void)
{         for (;;)
        {P1=0x55;
          P1=0xAA;
        }
}
```

***signed char*:**

The signed char is an 8-bit data type that uses the MSB D7 to represent – or +value. This implies there are only seven bits for the magnitude of a signed number, giving us values from –128 to +127. We should stick with the unsigned char unless the data needs to be represented as signed numbers. Example temperature.

Example 4: Write an 8051 C program to send values of –4 to +4 to port P1.

Solution:

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

Note: The negative values will be displayed in the 2's complement for as 1, FFH, 2, FEH, 3, FDH, 4, FCH.

unsigned int:

The unsigned int is a 16-bit data type that takes a value in the range of 0 to 65535 (0000 – FFFFH). This is used to define 16-bit variables such as memory addresses, set counter values of more than 256. Since registers and memory accesses are in 8-bit chunks, the misuse of int variables will result in a larger hex file. However, for 8051 programming, do not use unsigned int in place where unsigned char will do the job. Also in situations where there is no need for signed data we should use unsigned int instead of signed int. Remember, C compiler uses signed int as default if unsigned keyword is not used.

***signed int*:**

Signed int is a 16-bit data type that uses the MSB D15 to represent – or +value. We have 15 bits for the magnitude of the number from –32768 to +32767.

***sbit* (Single bit):**

The 8 bit keyword is a widely used 8051 C data types which is used to access single-bit addressable register. It allows access to the single bits of the SFR registers. As we know and have studied, some of the SFRs are bit addressable. Among the SFRs that are widely used are also bit addressable ports P0-P3.

Example 5: Write an 8051 C program to toggle bit D0 of the port P1 (P1.0) 50,000 times.

Solution:

```
#include <reg51.h>

sbit MYBIT=P1^0;
void main(void)
   {
```

sbit keyword allows access to the single bits of the SFR registers

bit and sfr:

The bit data type allows access to single bits of bit-addressable memory spaces 20 – 2FH. To access the byte-size SFR registers, we use the sfr data type. Table 1 summaries the different data types used for 8051 programming in C.

Table 1: Widely used 8051 C data types

Data Type	Size in Bits	Data Range/Usage
unsigned char	8-bit	0 to 255
(signed) char	8-bit	-128 to +127
unsigned int	16-bit	0 to 65535
(signed) int	16-bit	-32768 to +32767
sbit	1-bit	SFR bit-addressable only
bit	1-bit	RAM bit-addressable only
sfr	8-bit	RAM addresses 80 – FFH only

Time Delays

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

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

In either case, the delays can be observed either on the oscilloscope or using a simulator. Here only the discussion of time delay using simple for loop is carried out. In creating a time delay using for loop three factors need to be considered that can affect the accuracy of the delay.

1. Number of machine cycles and number of clocks period per machine cycle.
2. Crystal frequency connected between XTAL1 and XTAL2. Duration of the clock period for the machine cycle is the function of crystal frequency.
3. Compiler choice. The third factor that affects the accuracy of the time delay is the compiler used to compile the C program. In assembly language programming, we can know and control the delay generated, as, the number of instructions and the cycles per instruction is known to us. In case of C program, the C compiler will convert the C statements and functions to assembly language instructions. Hence, different compilers produce different code.

Example 6: Write an 8051 C program to toggle bits of P1 continuously forever with some delay.

Solution:

//Toggle P1 forever with some delay in between "on" and "off"
#include <reg51.h>
void main(void)
{
        unsigned int x;

I/O Programming in 8051 C

Byte size I/O: Ports P0-P3 is used which are byte accessible. Below are few examples that elaborate on this programming.

Example 8: LEDs are connected to bits P1 and P2. Write an 8051 C program that shows the count from 0 to FFH (0000 0000 to 1111 1111 in binary) on the LEDs.

Solution:

```
#include <reg51.h>

#define LED P2;
void main(void)
```

Ports P0 – P3 are byte-accessable and we use the P0 – P3 labels as defined in the 8051 header file.

Example 9: Write an 8051 C program to get a byte of data form P1, wait ½ second, and then send it to P2.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
        unsigned char mybyte;
        P1=0xFF; //make P1 input port
        while (1)
         {
            mybyte=P1; //get a byte from P1
            MSDelay(500);
            P2=mybyte; //send it to P2
         }
}
```

Example 10: Write an 8051 C program to get a byte of data form P0. If it is less than 100, send it to P1; otherwise, send it to P2.

Solution:

```
#include <reg51.h>
void main(void)
{       unsigned char mybyte;
        P0=0xFF;                //make P0 input port
        while (1)
         {  mybyte=P0;          //get a byte from P0
            if (mybyte<100)
            P1=mybyte;          //send it to P1
            else
            P2=mybyte;}}        //send it to P2
```

Bit-addressable I/O Programming

The I/O ports of P0-P3 are bit addressable. This means a single bit of the port can be accessed without disturbing the other bits. The bit can be accessed using the data type sbit. One way to do that is to use the format Px^y, where x is the port 0, 1, 2, 3 and y is the bit 0-7 of that port. For example P1^3 indicates P1.3. Study of few examples will help us become familiar with the syntax.

Example 11: Write an 8051 C program to toggle only bit P2.4 continuously without disturbing the rest of the bits of P2.

Solution:

```
//Toggling an individual bit
#include <reg51.h>
sbit mybit=P2^4;
void main(void)
{
          while (1)
           {
              mybit=1;              //turn on P2.4
              mybit=0;              //turn off P2.4
           }
}
```

Ports P0 – P3 are bit addressable and we use *sbit* data type to access a single bit of P0 - P3

Use the Px^y format, where x is the port 0, 2, or 3 and y is the bit 0 – 7 of that port

Example 12: Write an 8051 C program to monitor bit P1.5. If it is high, send 55H to P0; otherwise, send AAH to P2.

Solution:

```
#include <reg51.h>
sbit mybit=P1^5;
void main(void)
{
          mybit=1;                  //make mybit an input
          while (1)
            {
               if (mybit==1)
               P0=0x55;
              else
              P2=0xAA;
}
}
```

Example 13: A door sensor is connected to the P1.1 pin, and a buzzer is connected to P1.7. 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. (Note: MSDelay is the same as written for previous examples).

Algorithm

1. Make P1.1 as input.

Accessing SFR Addresses 80 – FFH

Another way to access the SFR RAM space 80H-FFH is to use the sfr data type. When this data type is used no need of using the header file reg51.h. Table 4 shows the single bit addresses of ports.

Table 4: Single Bit Addresses of Ports

P0	87	86	85	84	83	82	81	80	**80**

P1	97	96	95	94	93	92	91	90	**90**
P2	A7	A6	A5	A4	A3	A2	A1	A0	**A0**
P3	B7	B6	B5	B4	B3	B2	B1	B0	**B0**
Ports/Port's bit	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	In Hex

Example 14: Write an 8051 C program to toggle all the bits of P0 and P2 continuously with a 250 ms delay. Use the sfr keyword to declare the port addresses.

Solution:

```
//Accessing Ports as SFRs using sfr data type
sfr P0=0x80;
sfr P1=0x90;
sfr P2=0xA0;
void MSDelay(unsigned int);
void main(void)
{
        while (1)
          {
                P0=0x55;
                P2=0x55;
                MSDelay(250);
                P0=0xAA;
                P2=0xAA;
                MSDelay(250);
           }
}
```

Another way to access the SFR RAM space 80 – FFH is to use the *sfr* data type

Note: MSDelay function is the same as discussed in above examples.

Example 15:Write an 8051 C program to turn bit P1.5 on and off 50,000 times.

Solution:

```
sbit MYBIT=0x95;
```

We can access a single bit of any SFR if we specify the bit address

```
void main(void)
{
        unsigned int z;
        for (z=0;z<50000;z++)
        {
           MYBIT=1;
           MYBIT=0;
        }
}
```

Notice that there is no #include <reg51.h>. This allows us to access any byte of the SFR RAM space 80 – FFH. This is widely used for the new

Example 16: Write an 8051 C program to get the status of bit P1.0, save it, and send it to P2.7 continuously.

Solution:

```
#include <reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit membit;     //use bit to declare
                //bit- addressable memory


void main(void)
{
     while (1)
     {
       membit=inbit;      //get a bit from P1.0
       outbit=membit;     //send it to P2.7
     }
}
```

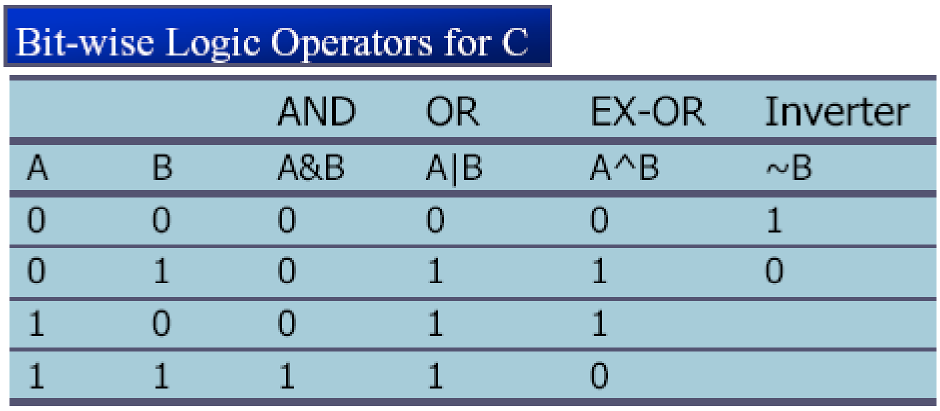
We use bit data type to access data in a bit-addressable section of the data RAM space 20 – 2FH

Logic Operation in 8051 C

One of the most important and powerful features of the C language is its ability to perform bit manipulation. This section describes the action of bitwise logic operators and provides some examples of how they are used.

The bitwise operators used in C are AND (&), OR (|), EX-OR (^), inverter (~), Shift right (>>) and Shift left (<<). These are widely used in software engineering for embedded systems and control; consequently, understanding and mastery of them are critical in microprocessor and microcontroller-based system design and interfacing. The operations are shown in table 5 below.

Table 5: Bit-wise logic operations



Bit-wise Logic Operators for C

		AND	OR	EX-OR	Inverter
A	B	A&B	A|B	A^B	~B
0	0	0	0	0	1
0	1	0	1	1	0
1	0	0	1	1	
1	1	1	1	0	

Below are some examples that illustrate the usage of these logic operators.

Example 17: The following program will explain the different logical operations .We can run on simulator and examine the results.

Solution:

```
#include <reg51.h>
void main(void)
{
        P0=0x35 & 0x0F;       //ANDing
        P1=0x04 | 0x68;       //ORing
        P2=0x54 ^ 0x78;       //XORing
        P0=~0x55;             //inversing
        P1=0x9A >> 3;         //shifting right 3
        P2=0x77 >> 4;         //shifting right 4
        P0=0x6 << 4;          //shifting left 4
}
```

Example 18: Write an 8051 C program to toggle all the bits of P0 and P2 continuously with a 250 ms delay. Using the inverting and Ex-OR operators, respectively.

Solution:

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
 {
        P0=0x55
```

Example 19: Write an 8051 C program to get bit P1.0 and send it to P2.7 after inverting it.

Solution:

```
#include <reg51.h>
sbit inbit=P1^0;
sbit outbit=P2^7;
bit membit;
void main(void)
{
        while (1)
        {
                membit=inbit;           //get a bit from P1.0
                outbit=~membit;         //invert it and send
                                        //it to P2.7
        }
}
```

Example 20: Write an 8051 C program to read the P1.0 and P1.1 bits and issue an ASCII character to P0 that is if P1.1 and P1.0 is 00 send ‘0’ if 01 send ‘1’, if 10 send ‘2’ and if 11 send ‘3’.

Algorithm

1. Make P1 as input port.
2. Read P1 value.
3. Mask all bits except D0 & D1 of P1 ad put the masked value in x.
4. If x=0; send ‘0’ to P0, else if x=1; send ‘1’ to P0, else if x=2; send ‘2’ to P0,

Data conversion programs in 8051 C

The basic data conversions we will be seeing into are:

- Packed BCD to ASCII conversion
- ASCII to Packed BCD conversion
- Binary (hex) to decimal and ASCII conversion

These conversions can be understood through the following examples:

Example 21: Write an 8051 C program to convert packed BCD number 0x45 to ASCII and display on P1 and P2.

Algorithm

1. Get the packed BCD number.
2. Convert it to unpacked BCD by masking the lower and upper digit.
3. Add 30H individually and send the ASCII number to P1 and P2

Example 22: Write an 8051 C program to convert ASCII digits to packed BCD and display them on P1.

Algorithm

1. Get the 1st ASCII number and mask the higher nibble.
2. Shift the number to get the higher nibble of BCD
3. Get the 2nd ASCII number and mask the higher nibble.
4. Add the result to the first number.
5. Display the BCD number on P1.

Solution:

```
#include <reg51.h>
void main(void)
{
        unsigned char bcdbyte;
        unsigned char w=‘4’;
        unsigned char z=‘7’;
        w=w&0x0F;
        w=w<<4;
        z=z&0x0F;
        bcdbyte=w|z;
        P1=bcdbyte;
```

Example 23: Write an 8051 C program to convert 11111101(FDH) to decimal and display the digits on P0, P1 and P2.

Algorithm:

1. Divide the binary number by 10 and save it in x.
2. Modulo divide the binary by 10 & save the result in units.
3. Divide x by 10 & save it in hundred.

Checksum byte in ROM

The checksum byte is used to detect any corruption of data and to ensure data integrity. The checksum process uses checksum byte. The calculation of checksum byte is done in two steps:

1. Add the bytes together and drop the carries.
2. Take 2's complement of total sum result is checksum byte.

The calculation of the checksum byte, checking of data integrity can be shown through the examples below:

Example 24: Assume there are 4 bytes of hex data: 25h, 62h, 3Fh and 52h.

(a) Find the checksum byte

(b) Perform checksum operation for data integrity

(c) If 2nd byte 62h is changed to 22h, show how checksum detects error.

Solution:

(a) Addition of data is: 25h+62h+3Fh+52h = 118H, discarding the carry and taking 2's complement the result is E8H

The above calculation can be done through the examples below:

Example 25: Write an 8051 C program to calculate the checksum byte for the data given in example 24.

Solution:

```
#include <reg51.h>
void main(void)
 {
   unsigned char mydata[]={0x25,0x62,0x3F,0X52};
   unsigned char sum=0;
   unsigned char x;
   unsigned char chksumbyte;
   for(x=0;x<4;x++)
          {
            P2=mydata[x];
            sum=sum+mydata[x];
            P1=sum;                    //send sum to port 1
          }
          chksumbyte=~sum+1;           // take 2's complement
          P1=chksumbyte;
  }
```

Example 26: Write an 8051 C program to perform step (b) of example 24. If data is error free send ASCII 'G' else send ASCII 'B'.

Solution:

```
#include <reg51.h>
void main(void)
 {
```

Accessing Code ROM Space in 8051 C

RAM data space vs. code data space
Three spaces to store data:

1. The 128 bytes of RAM space with address range 00-7FH
2. The 64KB of code space with address 0000-FFFFH
   - used for storing program & under control of PC
   - data accessed using MOVC A,@A+DPTR

The problems using this code space for data: can burn predefined data and tables but can't write during the execution of program. Secondly, more of code space is used for data and less for program code: Eg- DS89C420 and hence Intel created another memory space called *external memory* especially for data.

3. The 64KB of external memory – RAM & ROM

The 8051 C compiler allocates RAM locations as

- Bank 0 – addresses 0 – 7
- Individual variables – addresses 08 and beyond
- Array elements – addresses right after variables

Array elements need contiguous RAM locations and that limits the size of the array due to the fact that we have only 128 bytes of RAM for everything

- Stack – addresses right after array elements

Example 27: Compile and single-step the following program on your 8051simulator. Examine the contents of the 128-byte RAM space to locate the ASCII values.

Solution:

```
#include <reg51.h>
void main(void)
{
```

Example 28: Write, compile and single-step the following program on your 8051simulator. Examine the contents of the code space to locate the values.

Solution:

```
#include <reg51.h>
void main (void)
{
            unsigned char mydata[100]; //RAM space
            unsigned char x, z=0;
            for (x=0;x<100;x++)
                {  z--;
                   mydata [x]=z;
                   P1=z;
                }
}
```

Example 29: Compile and single-step the following program on your 8051simulator. Examine the contents of the code space to locate the ASCII values.

Solution:

```
#include <reg51.h>
void main(void)
{
            code unsigned char mynum[]="ABCDEF";
            unsigned char z;
            for (z=0;z<=6;z++)
            P1=mynum[z];
}
```

To make the C compiler use the code space instead of the RAM space, we need to put the keyword code in front of the variable declaration

Data Serialization

Serializing data is a way of sending a byte of data one bit at a time through a single pin of microcontroller

- ❑ Using the serial port
- ❑ Transfer data one bit a time and control the sequence of data and spaces in between them

In many new generations of devices such as LCD, ADC, and ROM the serial versions are becoming popular since they take less space on a PCB.

Example 30: Write a C program to send out the value 44H serially one bit at a time via P1.0. The LSB should go out first.

Solution:

```
#include <reg51.h>
sbit P1b0=P1^0;
sbit regALSB=ACC^0;
void main(void)
{
        unsigned char conbyte=0x44;
        unsigned char x;
        for (x=0;x<8;x++)
        {
                ACC = conbyte;
                P1b0 = regALSB;
                ACC = ACC>>1;
                conbyte = ACC;
        }
}
```

Example 31: Write a C program to bring in a byte of data serially one bit at a time via P1.0. The MSB should come in first.

Solution:

```
#include <reg51.h>
sbit P1b0=P1^0;
sbit regALSB=ACC^0;
bit membit;
void main(void)
{
        unsigned char x;
        for (x=0;x<8;x++)
        {
                membit=P1b0;
                ACC=ACC<<1;
                regALSB=membit;
        }
        P2=ACC;}
```

Summary

This chapter dealt with 8051 C programming, especially I/O programming and time delays in 8051 C. We also showed some logical operations such as AND, OR, XOR and complement. In addition, some applications for these were discussed. This chapter also described data conversions such as BCD, ASCII and binary (HEX). We also compared and contrasted the use of code space and RAM data space in 8051 C. The widely used technique of data serialization was also discussed.