Module No.3

8051 Programming using C

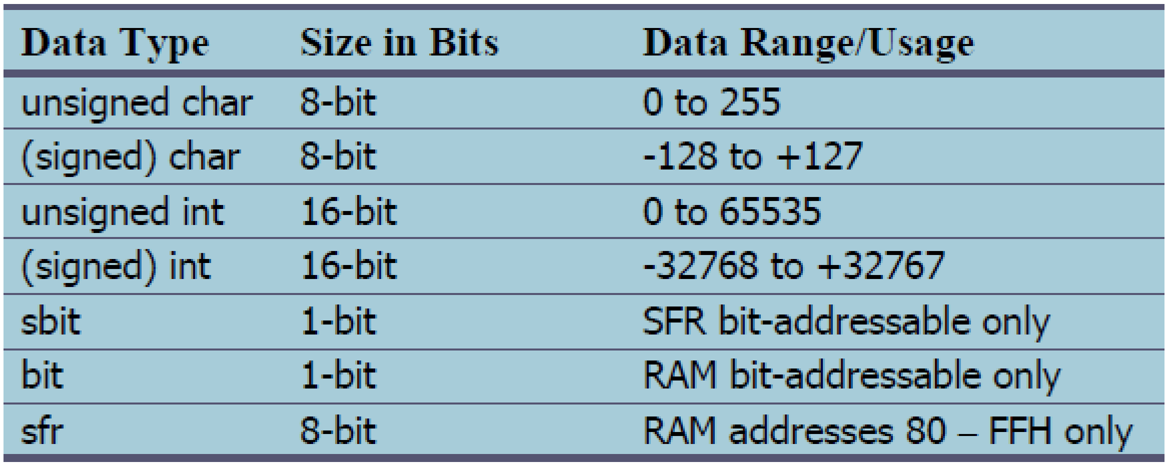
Compilers produce hex files that is downloaded to ROM of microcontroller. The size of hex file is the main concern. Microcontrollers have limited on-chip ROM. Code space for 8051 is limited to 64K bytes. C programming is less time consuming, but has larger hex file size. The reasons for writing programs in C. It is easier and less time consuming to write in C than Assembly. C is easier to modify and update. We can use code available in function libraries. C code is portable to other microcontroller with little no of modifications.

Data types in C

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

Unsigned char
Signed char
Unsigned int
Signed int
Sbit (single bit)
Bit and sfr

The character data type is the most natural choice. Unsigned char is an 8-bit data type in the range of 0 – 255 (00 – FFH). C compilers use the signed char as the default if we do not put the keyword unsigned.



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

Write an 8051 C program to send values 00 – FF to port P1.

```
#include <reg51.h>
void main(void)
{
unsigned char z;
for (z=0;z<=255;z++)
P1=z;
}
```

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];
}
```

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;
}
}
```

The signed char is an 8-bit data type. Use the MSB D7 to represent – or +. Give values from –128 to +127. We should stick with the unsigned char unless the data needs to be represented as signed numbers.

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

Solution:

```
//Singed numbers
#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];
}
```

The unsigned int is a 16-bit data type. Takes a value in the range of 0 to 65535 (0000 – FFFFH). 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. Signed int is a 16-bit data type. Use the MSB D15 to represent – or +.We have 15 bits for the magnitude of the number from –32768 to +32767. The bit data type allows to access single bits of bit-addressable memory spaces 20 – 2FH. To access the byte-size SFR registers, we use the sfr data types.

Time Delay using C

There are two way s to create a time delay in 8051 C.

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

Three factors that can affect the accuracy of the delay are.

The 8051 design

a) The number of machine cycle
b) The number of clock periods per machine cycle

The crystal frequency connected to the X1 – X2 input pins. C compiler converts the C statements and functions to Assembly language instructions. Different compilers produce different code.

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;
for (;;) //repeat forever
{
P1=0x55;
for (x=0;x<40000;x++); //delay size
//unknown
P1=0xAA;
for (x=0;x<40000;x++);
}
}
```

Write an 8051 C program to toggle bits of P1 ports continuously with a 250 ms.

Solution:

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

```
while (1) //repeat forever
{
P1=0x55;
MSDelay(250);
P1=0xAA;
MSDelay(250);
}
}
void MSDelay(unsigned int itime)
{
unsigned int i,j;
for (i=0;i<itime;i++)
for (j=0;j<1275;j++);
}
```

I/O Programming in C

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)
{
P1=00; //clear P1
LED=0; //clear P2
for (;;) //repeat forever
{
P1++; //increment P1
LED++; //increment P2
}
}
```

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
}
}
```

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
}
}
```

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;
}
}
```

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

Solution:

```
sbit MYBIT=0x95;
void main(void)
{
```

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

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
}
}
```

Logic Operations in C

Logical operators

AND (&&), OR (||), and NOT (!)

Bit-wise operators

AND (&), OR (|), EX-OR (^), Inverter (~),
Shift Right (>>), and Shift Left (<<)

These operators are widely used in software
engineering for embedded systems and control

Run the following program on your 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
}
```

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;
P2=0x55;
while (1)
{
P0=~P0;
P2=P2^0xFF;
MSDelay(250);
}
}
```

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
}
}
```

Data Conversion using C

Write an 8051 C program to convert packed BCD 0x29 to ASCII and display the bytes on P1 and P2.

Solution:

```
#include <reg51.h>
void main(void)
{
unsigned char x,y,z;
unsigned char mybyte=0x29;
x=mybyte&0x0F;
P1=x|0x30;
y=mybyte&0xF0;
y=y>>4;
P2=y|0x30;
}
```

Write an 8051 C program to convert ASCII digits of ‘4’ and ‘7’ to packed BCD and display them 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;
}
```

Write an 8051 C program to calculate the checksum byte for the data 25H, 62H, 3FH, and 52H.

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;
}
chksumbyte=~sum+1;
P1=chksumbyte;
}
```

Write an 8051 C program to perform the checksum operation to ensure data integrity. If data is good, send ASCII character ‘G’ to P0.Otherwise send ‘B’ to P0.

Solution:

```
#include <reg51.h>
void main(void)
{
unsigned char mydata[]
={0x25,0x62,0x3F,0x52,0xE8};
unsigned char shksum=0;
unsigned char x;
for (x=0;x<5;x++)
chksum=chksum+mydata[x];
if (chksum==0)
P0=‘G’;
else
P0=‘B’;
}
```

Write an 8051 C program to convert 11111101 (FD hex) to decimal and display the digits on P0, P1 and P2.

Solution:

```
#include <reg51.h>
void main(void)
{
unsigned char x,binbyte,d1,d2,d3;
binbyte=0xFD;
x=binbyte/10;
d1=binbyte%10;
d2=x%10;
d3=x/10;
P0=d1;
P1=d2;
P2=d3;
}
```

Accessing Code ROM Space

The 8051 C compiler allocates RAM locations. 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.

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

Solution:

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

Write, compile and single-step the following program on your 8051 simulator. 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;
}
}
```

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 is known as serialization. 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.

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;
ACC=conbyte;
for (x=0;x<8;x++)
{
P1b0=regALSB;
ACC=ACC>>1;
}
}
```

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

Solution:

```
#include <reg51.h>
sbit P1b0=P1^0;
sbit regAMSB=ACC^7;
void main(void)
{
unsigned char conbyte=0x44;
unsigned char x;
ACC=conbyte;
for (x=0;x<8;x++)
{
P1b0=regAMSB;
ACC=ACC<<1;
}
}
```

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

Solution:

```
#include <reg51.h>
sbit P1b0=P1^0;
sbit ACCMSB=ACC^7;
bit membit;
void main(void)
```

```
{
unsigned char x;
for (x=0;x<8;x++)
{
membit=P1b0;
ACC=ACC>>1;
ACCMSB=membit;
}
P2=ACC;
}
```

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;
}
```