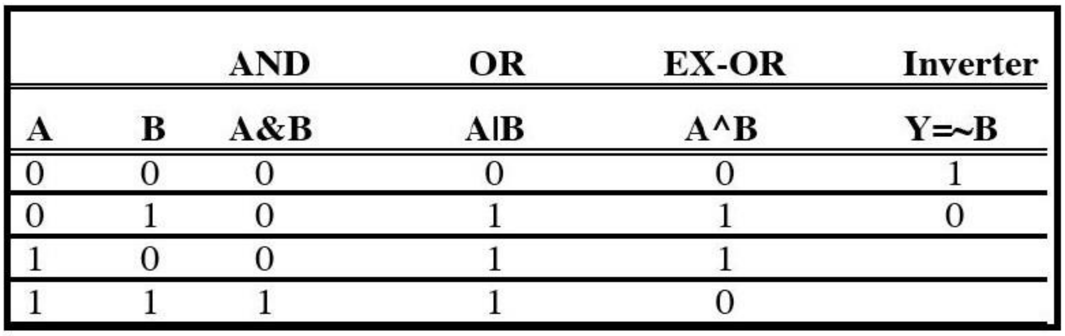
Basic Operation of Embedded C for 8051

By Frankie, K. F. Yip

In this lecture, we will discuss about the Logical operators in Embedded C. Data conversion programs in embedded C will also be written and discussed. At the end Data Serialization in 8051 using embedded C will be discussed.

1.1 Bitwise Operations in C

One of the most important and powerful features of the C language is its ability to perform bit wise manipulation. This section describes the action of bitwise logic operators. Figure1.1 shows the bitwise logical operators.



		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	

Figure 1.1: Bitwise Logical Operation

Using these bit wise operators Embedded c will perform the logical operations bit wise on binary numbers. Following are some examples which show how they are used.

- AND &

  `0x35 & 0x0F = 0x05`

- OR |

  `0x04 | 0x68 = 0x6C`

- Exclusive-OR ^

  `0x54 ^ 0x78 = 0x2C`

- Inverter ~

  `~0x55 = 0xAA`

```
NOT  0101  0101
      1010 1010
```

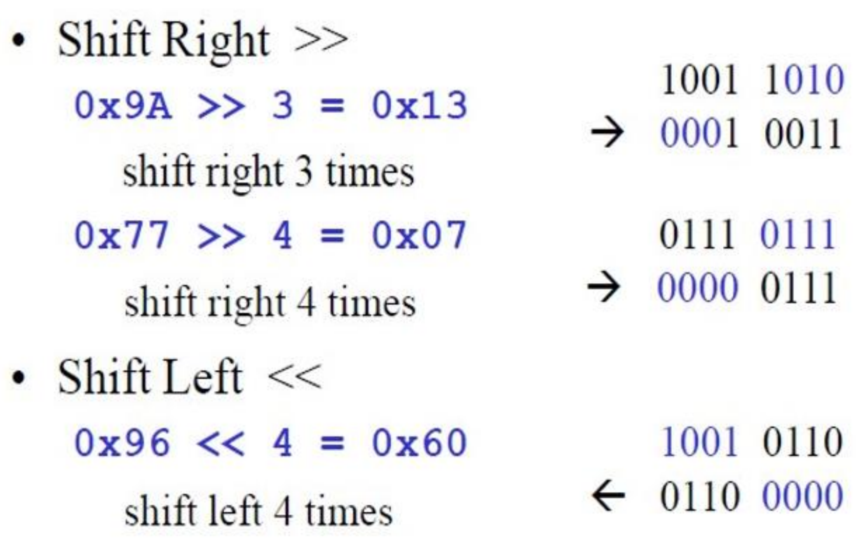
```
     0011 0101
AND  0000 1111
     0000 0101

     0000 0100
OR   0110 1000
     0110 1100

     0101 0100
XOR  0111 1000
     0010 1100
```

1.2 Bit-wise Shift Operation in C

There are two bit-wise shift operators in C: (1) shift right (»), and (2) shift left («).
Their format in C is as follows:

⇨ data » number of bits to be shifted right
⇨ data « number of bits to be shifted left

Some examples are shown below:



- Shift Right >>

  0x9A >> 3 = 0x13 $\quad$ 1001 1010

  shift right 3 times $\quad$ → 0001 0011

  0x77 >> 4 = 0x07 $\quad$ 0111 0111

  shift right 4 times $\quad$ → 0000 0111

- Shift Left <<

  0x96 << 4 = 0x60 $\quad$ 1001 0110

  shift left 4 times $\quad$ ← 0110 0000

Following program will show the demo of the logical operations. Run the following program on your simulator and examine the results.

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

Next program will show the operation on port pins to access data bit-wise and manipulate through logical operator.

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

Solution:

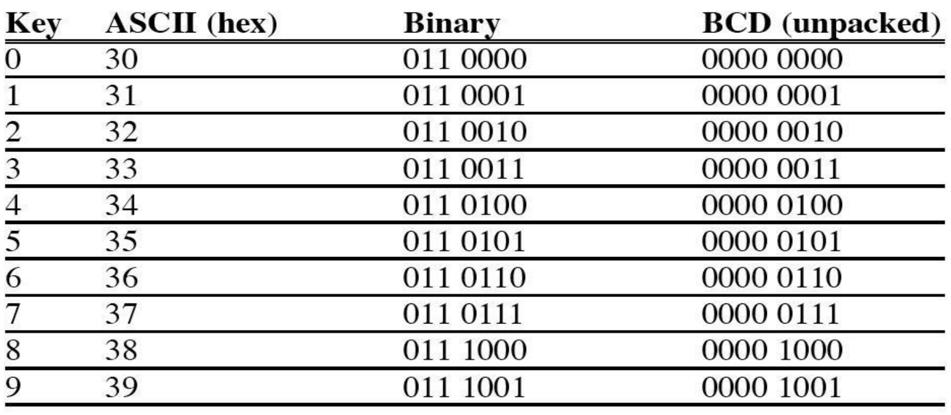
```
#include <reg51.h>

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
    }
}
```

1.3 Data Conversion

We have seen BCD numbers in previous modules. As stated there, many newer microcontrollers have a real-time clock (RTC) where the time and date are kept, even when the power is off. Very often the RTC provides the time and date in packed BCD. However, to display them they must be converted to ASCII. Like this example, we need to convert the data from one form to another form. In this section we show some example programs to demo how Embedded C helps for data conversions. Some data conversions needed are listed below.

- Packed BCD to ASCII conversion
- ASCII to packed BCD conversion
- Checksum byte in ROM
- Binary to decimal and ASCII conversion in C

Figure.2 shows the ASCII, binary and BCD codes for the digits 0 to 9.



Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

Figure 1.2: ASCII Code for Digits 0-9

Following program shows Packed BCD to ASCII conversion**.**

1.3.1 Example.1

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;
}
```

Following program shows the ASCII to packed BCD conversion.

1.3.2 Example.2

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;
}
```

Following program is for finding the Checksum byte in ROM.

1.3.3 Example.3

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

Steps to calculate Checksum byte in ROM are:

1. Add the bytes together and drop carries.
2. Take the 2’s complement (invert and then add 1) of the total sum. This is the Checksum byte, which becomes the last byte of the series.   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;  //logical operator used here
     P1 = chksumbyte;
}
```

Following program will demo Binary to Decimal and ASCII Conversion in 8051 C

1.3.4 Example.4

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;
}
```

1.4 Accessing Code ROM space in 8051 C

Using the code (program) space for predefined data is a widely used option in 8051. We saw how to use the Assembly language instruction MOVC to access the data stored in the 8051 code space. Here, we see the same concept with 8051 C.

In 8051, we have three spaces to store data:

1. The 128 bytes RAM space with address range 00-7FH

   ✓ If you declare variables (eg.: char) to store data, C compiler will allocate a RAM space for these variable.

2. User code space

   ✓ External code memory (64K) + on-chip ROM (64K)
   ✓ Data is embedded to code or is separated as a data section.

3. External data memory for data ✓ RAM or ROM is used.

2.1 RAM data space usage

The 8051 C compiler allocates RAM locations as follows:

1. Bank 0 – addresses 0 – 7

2. Individual variables – addresses 08 and beyond

3. 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.

4. Stack – addresses right after array elements.

Following is an example program to access data from memory.

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

```
#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;
   }
}
```

While running this program you can see how the data will be stored in an array and displayed in a port.

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

2.2 8052 RAM Space

One of the new features of the 8052 was an extra 128 bytes of RAM space.

- ✓ The extra 128 bytes of RAM helps the 8051/52 C compiler to manage its registers and resources much more effectively.
- ✓ Based on 8052 architecture, you should use the reg52.h header file. Choose the 8052 option when compiling the program.

2.2.1 Using ROM to Store Data

To make C compiler use the code space (on-chip ROM) instead of RAM space, we can put the keyword “code” in front of the variable declaration.

unsigned char mydata[ ] = “HELLO” ✓

HELLO is saved in RAM.

code unsigned char mydata[ ] = “HELLO” ✓

HELLO is saved in ROM.

This is discussed in the following examples.

2.2.2 Examples 5

Let us compare and contrast the following programs and discuss the advantages and disadvantages of each one.

Example 5(a):

```
#include <reg51.h>
void main(void)
 {
    P1=“H”;
    P1=“E”;
    P1=“L”;
    P1=“L”;
    P1=“O”;
}
```
Data is embedded into code. Simple, short, not flexible.

Example 5(b):

```
#include <reg51.h>
void main(void)
 {
     unsigned char mydata[ ]="HELLO";
     unsigned char z;
     for (z=0; z<5; z++)
         P1 = mydata[z];
}
```

Data is stored in RAM and does not occupy ROM.

Example 5(c):

```
#include <reg51.h>
void main(void)
 {
     Code unsigned char mydata[ ]="HELLO";
     unsigned char z;
     for (z=0; z<5; z++)
         P1 = mydata[z];
}
```

Data is stored in ROM. However, data and code are separate.

2.3 Data serialization using 8051 C

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 (discussed in Serial Communication module)
- ✓ 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.

The following are examples that demonstrate data serialization using embedded C.

Example. 6

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;
    }
}
```

2.3.1 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.

Example.7

```
#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;
}
```