Why program the 8051 in C?

- Compilers produce **hex files** that is downloaded into the ROM of the microcontroller

- The size of hex file is the main concern for two reasons
  - Microcontrollers have **limited on-chip ROM**.
  - The **code space for the 8051** is limited to **64KB**

Assembly Vs C

- **Assembly language Programming**
  - Produces a **hex file** that is much **smaller than C**
  - Programming in Assembly Language is **tedious and time consuming**.
- **C Programming**
  - **Less Time consuming**
  - **Easier to write**
  - Produces a **hex file that is much larger than ALP**

Why Write in C ?

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

What is Embedded C?

- Embedded C is a set of language extensions for the C programming language, enhanced for different embedded systems.
- The extensions are required to support exotic features such as fixed-point arithmetic, multiple distinct memory banks and basic I/O operations.
- Embedded C uses most of the syntax and semantics of standard C, e.g., main() function, variable definition, datatype declaration, conditional statements (if, switch case), loops (while, for), functions, arrays and strings,

  structures and union, bit operations, macros, etc.

Data Types in Embedded C

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

unsigned char

- Unsigned char is an 8-bit data type in the range of

  0 – 255 (00 – FFH)

  – The character data type is the most natural choice because 8051 is an 8-bit microcontroller
- Used for
  – Counter value
  – ASCII characters
- C compilers use the signed char as the default if we do not put the keyword unsigned

signed char

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

unsigned int

- The unsigned int is a 16-bit data type
- Takes a value in the range of 0 to 65535 (0000 – FFFFH)
- Used for:
  - 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

- 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

bit, sbit and sfr

- The bit data type allows access to single bits of bit-addressable memory spaces 20 – 2FH.
- To access bits of SFRs, we use sbit data type.
- To access the byte-size SFR registers, we use the sfr data type.

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

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

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.

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

- Note:

1. Pay careful attention to the size of the data

2. Try to use unsigned char instead of int if possible. Since 8051 has limited number of registers , using int will produce a large size hex file.

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

```
#include <reg51.h>
void main(void)
{
        While (1)
        {
                p1=0x55; 0x indicated data in hex(Binary)
                p1=0xAA;
        }
}
```

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

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

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

```
//Signed numbers
#include <reg51.h>
void main(void)
{
        signed char i;
        for (i=-4;i<=4;i++)
        P1=i;
}
```

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

```
#include <reg51.h>
sbit MYBIT=P1^0;  // sbit is defined before main
void main(void)
{
        unsigned int z;
        for (z=0;z<=50000;z++)
        {
                MYBIT=0;
                MYBIT=1;
        }
}
```

- Note: sbit keyword allows access to the single bits of the SFR registers

Time delay in C

- There are two ways to create a time delay in 8051 C
  - **Using the 8051 timer**
  - **Software delay using instructions**
- Note that accuracy of the software delay depends on Compiler choice
  - 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

```
//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++); //runs loop for 4000 times; without any
 operation
                p1=0xAA;
                for (x=0;x<40000;x++); //This will create some delay
        }
 }
```

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

```
#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 tim)
{
        unsigned int i,j;
        for (i=0;i<tim;i++)
        for (j=0;j<1275;j++);
}
```

I/O Port Programming : Byte Size I/O

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

```
#include <reg51.h>
#define LED P2;
void main(void)
{
        P1=00; //clear P1
        LED=0; //clear P2
        while(1)
        {
                P1++; //increment P1
                LED++; //increment P2
        }
}
```

Note : Ports P0 – P3 are byte - accessible and we can use the P0 – P3 labels as defined in the 8051 header file <reg51.h>

Write an 8051 C program to get a byte of data form P1, wait 1/2 second, and then send it to P2.

```
#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
        }
 }
void MSDelay(unsigned int tim)
{
        unsigned int i,j;
        for (i=0;i<tim;i++)
        for (j=0;j<1275;j++);
}
```

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.

```
#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
        }
}
```

I/O PROGRAMMING Bit-addressable I/O

- The I/O ports of P0-P3 are bit-addressable.
- Can access a single bit without disturbing the rest of the port.
- sbit data type is used to access a single pit of port P0-P3.
- P**x^y**
- **x – port 0,1,2,3**.
- **y- bits 0-7 of that port**.
- E.g. P**1^7** indicates P**1**.**7**

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

```
//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
        }
}
```

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

```
#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;
        }
}
```

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

```
#include <reg51.h>
void MSDelay(unsigned int);
sbit Dsensor=P1^1;
sbit Buzzer=P1^7;
void main(void)
{
        Dsensor=1; //make P1.1 an input
        while (1)
        {
                while (Dsensor==1)//while it opens
                {
                        Buzzer=0;
                        MSDelay(200);
                        Buzzer=1;
                        MSDelay(200);
                }
        }
}
```

```
void MSDelay(unsigned int tim)
{
        unsigned int i,j;
        for (i=0;i<tim;i++)
        for (j=0;j<1275;j++);
}
```

The data pins of an LCD are connected to P1. The information is latched into the LCD whenever its Enable pin goes from high to low. Write an 8051 C program to send “KTU” to this LCD

```
#include <reg51.h>
#define LCDData P1 //LCDData declaration
sbit En=P2^0; //the enable pin
void main(void)
{
        unsigned char message[] =“KTU”;
        unsigned char z;
        for (z=0;z<3;z++) //send 3 characters
        {
                LCDData=message[z];
                En=1; //a high
                En=0; //-to-low pulse to latch data
        }
}
```

Write an 8051 C program to toggle all the bits of P0, P1, and P2 continuously with a 250 ms delay. Use the sfr keyword to declare the port addresses

```
#include <reg51.h>
sfr P0=0x80;
sfr P1=0x90;
sfr P2=0xA0;
void MSDelay(unsigned int);
void main(void)
{
        while (1)
        {
                P0=0x55;
                P1=0x55;
                P2=0x55;
                MSDelay(250);
                P0=0xAA;
                P1=0xAA;
                P2=0xAA;
                MSDelay(250);
        }
}
```

```
void MSDelay(unsigned int tim)
{
        unsigned int i,j;
        for (i=0;i<tim;i++)
        for (j=0;j<1275;j++);
}
```

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

```
#include <reg51.h>
sbit MYBIT=0x95;
void main(void)
{
        unsigned int z;
        for (z=0;z<50000;z++)
        {
                MYBIT=1;
                MYBIT=0;
        }
}
```

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

I/O PROGRAMMING: Using bit Data Type for Bit-addressable RAM

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

```
#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
        }
}
```

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

Operators in C

- **Logical operators**
  - AND (&&), OR (||), and NOT (!)
- **Bit-wise operators**
  - AND (&), OR (|), EX-OR (^), Inverter (~)
  - Shift Right (>>), and Shift Left (<<)

C logical Operators

- **0x35 & 0x0F - 0x05 /*ANDing*/**
  - **0011 0101 0x35**
  - **0000 1111 0x0F**
  - **0000 0101 = 0x05**
- **0x04 | 0x68 = 0x6C - /*ORing*/**
- **0x54 ^ 0x78 = 0x2C -/*XORing*/**
- **~0x55 = 0xAA - /*Inverting 55H*/**

Bit-wise shift Operation in C

- Data >> number of bits to be shifted right
- Data << number of bits to be shifted right

- **0x9A>>3 = 0x13 /*Shift right 3 times*/**
  - **1001 1010 >>1 = 0100 1101 = 0x4C**
  - **1001 1010 >>2 = 0010 0110 = 0x26**
  - **1001 1010 >>3 = 0001 0011 = 0x13**
- **0x77>>4 = 0x07 /*Shift right 4 times*/**
  - **0111 0111, 0011 1011, 0001 1101, 0000 1110, 0000 0111**
- **0x6<<4 =0x60 /*Shift left 4 times*/**
  - **0000 0110, 0000 1100, 0001 1000, 0011 0000, 0110 0000**

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.

```
#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.

```
#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
        }
}
```

Write an 8051 C program to read the P1.0 and P1.1 bits and issue an ASCII character to P0 according to the following table.

P1.1	P1.0	
0	0	send ‘0’ to P0
0	1	send ‘1’ to P0
1	0	send ‘2’ to P0
1	1	send ‘3’ to P0

```
#include <reg51.h>
void main(void)
{
        unsigned char z;
        z=P1;
        z=z&0x3;
        switch (z)
        {
                Case 0: P0='0';
                        break;
                Case 1: P0='1';
                        break;
                Case 2: P0='2';
                        break;
                Case 3: P0='3';
                        break;
        }
}
```

DATA CONVERSION PROGRAMS IN C

- **RTC (Real Time Clock)**
  - Date and Time are kept even when the **power is off**.
  - Provides Time and Date in **packed BCD**.
  - To display them they must be converted to **ASCII**

ASCII Numbers

- On ASCII keyboards when key "**0**" is activated "**0011 0000**"**(30H**) is provided to computer**.**

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

Packed BCD to ASCII conversion

The RTC provides the time of day (hour, minute, second) and the date (year, month, day) continuously, regardless of whether the power is on or off. However, this data is provided in packed BCD. To convert packed BCD to ASCII, it must first be converted to unpacked BCD. Then the unpacked BCD is tagged with 011 0000 (30H). The following demonstrates converting from packed BCD to ASCII. See also Example 7-24.

Packed BCD	Unpacked BCD	ASCII
0x29	0x02, 0x09	0x32, 0x39
00101001	00000010,00001001	00110010,00111001

DATA CONVERSION: Packed BCD to ASCII Conversion

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

```
#include <reg51.h>
void main(void)
{
        unsigned char x,y,z;
        unsigned char mybyte=0x29;
        x=mybyte&0x0F;          // mask lower 4 bits
        P1=x|0x30;              // make it to ASCII
        y=mybyte&0xF0;          // mask upper 4 bits
        y=y>>4;                 // shift it to lower 4 bits
        P2=y|0x30;              // make it to ASCII
}
```

ASCII to packed BCD conversion

To convert ASCII to packed BCD, it is first converted to unpacked BCD (to get rid of the 3), and then combined to make packed BCD. For example, 4 and 7 on the keyboard give 34H and 37H, respectively. The goal is to produce 47H or "0100 0111", which is packed BCD.

Key	ASCII	Unpacked BCD	Packed BCD
4	34	00000100	
7	37	00000111	01000111 or 47H

DATA CONVERSION: ASCII to Packed BCD Conversion

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(void)
{
        unsigned char bcdbyte;
        unsigned char w='4';
        unsigned char z='7';
        w=w&0x0F;               // mask 3
        w=w<<4;                 // shift left to make upper BCD digit
        z=z&0x0F;               // mask 3
        bcdbyte=w|z;            // combine to make packed BCD
        P1=bcdbyte;
}
```

Binary (hex) to decimal and ASCII conversion in 8051 C

The printf function is part of the standard I/O library in C and can do many things, including converting data from binary (hex) to decimal, or vice versa. But printf takes a lot of memory space and increases your hex file substantially. For this reason, in systems based on the 8051 microcontroller, it is better to write your own conversion function instead of using printf.

One of the most widely used conversions is the binary to decimal conversion. In devices such as ADC (Analog-to-Digital Conversion) chips, the data is provided to the microcontroller in binary. In some RTCs, data such as time and dates are also provided in binary. In order to display binary data we need to convert it to decimal and then to ASCII. Since the hexadecimal format is a convenient way of representing binary data we refer to the binary data as hex. The binary data 00 - FFH converted to decimal will give us 000 to 255. One way to do that is to divide it by 10 and keep the remainder, as was shown in Chapter 6. For example, 11111101 or FDH is 253 in decimal. The following is one version of an algorithm for conversion of hex (binary) to decimal:

	Quotient	Remainder
FD/0A	19	3 (low digit) LSD
19/0A	2	5 (middle digit)
		2 (high digit) (MSD)

Example 7-29 shows the C program for that algorithm.

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;          //binary(hex) byte
    x = binbyte / 10;        //divide by 10
    d1 = binbyte % 10;       //find remainder (LSD)
    d2 = x % 10;             //middle digit
    d3 = x / 10;             //most significant digit (MSD)
    P0 = d1;
    P1 = d2;
    P2 = d3;
  }
```