Microprocessor
Ch.7 Programming In C

OUTLINE

- Data types and time delay
- I/O programming and Logic operations
- Data conversion programs
- Accessing code ROM space

DATA TYPE: C V.S. ASSEMBLY

- **C V.S. Assembly language**
  - Advantage of C
    - It's easier to program in C compared to assembly.
    - C code can be easily ported to other microcontroller, while assembly language can usually be used for 1 type of microcontroller
    - There are lots of function libraries written in C
  - Advantage of assembly language
    - The hex file generated by assembly is usually smaller
    - Code efficiency is higher (faster)
- **C programming**
  - C has become the standard for embedded system programming
    - Object oriented language (C++, Java, C#) are usually not as efficient as C
    - C is flexible
  - One of the main concerns in C programming for embedded system is to generate a hex file that is small in size
    - We need to pay attention to the size of variables during programming

DATA TYPE

- **Data types**
  - unsigned char, signed char, unsigned int, signed int, sbit, bit and sfr
- **unsigned char**
  - 8-bit. Most popular data type, matches most registers. Represent ASCII code or integers in the range of 0 ~ 255.
  - Example: write a C program to send the ASCII code of 0, 1, 2, A, B, C to port 1 (demo)

```
#include <reg51.h>              // the definition of registers
void main (void)
{
        unsigned char mynum[] = "012ABC";
        unsigned char z;
        for (z = 0; z <=5; z++)
                P1 = mynum[z];     //port 1 is denoted as P1
}
```

  - Example: write a C program to toggle all bits of P2 continuously

```
#include <reg51.h>
void main(void)
{
        for ( ; ; )                              // infinite loop
        {
                P1 = 0x55;                       // hex number: 0x55
                P1 = 0xAA;
        }
}
```

DATA TYPE

- **signed char**
  - an 8-bit number with the most significant bit representing sign (+ or -)
    - Range: -128 ~ 127 (range of unsigned char: 0 ~ 255)
  - Example: write a C program to send -3 to 3 to port 1

```
#include <reg51.h>
void main(void)
{
        char mynum[] = {-3,-2,-1,0,1,2,3};          // signed char
        unsigned char z;
        for (z = 0; z<=6; z++)
                P1 = mynum[z];
}
```

- **unsigned int**
  - 16-bit (needs to registers), range: 0 ~ 65535 (FFFFH)
  - E.g.: unsigned int a;
- **signed int**
  - 16-bit, MSB represents sign. Range: -32768 ~ 32767
  - E.g.: int a;
  - Since int requires twice as many memory space as char, use unsigned or signed int only when the number cannot be represented with unsigned or signed char.

DATA TYPE

- **sbit**
  - Single bit, used to access single-bit addressable registers
  - Example: write a C program to toggle bit D0 of P1 50,000 times

```
#include <reg51.h>
sbit MYBIT = P1^0;
void main(void)
{
        unsigned int z;; 50000 times, cannot use char
        for (z=0; z<=50000;z++)
        {
                MYBIT = 0;
                MYBIT = 1;
        }
}
```

- **bit**
  - Used to access single bit of bit-addressable RAM (20H – 2FH)
  - Example: bit mybit = 0; // the compiler will assign a RAM space to a
    //automatically
- **sfr**
  - Used to access special function registers
  - example: sfr ACC = 0xE0; // the address of reg. A

DATA TYPE: TIME DELAY

- **Time delay**
  - Two methods to achieve time delay:
    - Timer (Ch. 9)
    - Loop
  - When using loops in C, it's difficult to determine the exact time delay by means of calculation
    - For the same C code, different compilers usually will generate different assembly codes
    - Thus the same C codes with different compilers might generate different delays
    - The only way to know the exact delay is through oscilloscope.

```
#include <reg51.h>
void main(void)
{
        unsigned int x;
        while(1)
        {
                P1 = 0x55;
                for (x=0;x<40000;x++); //delay
                P1=0xAA;
                for (x=0;x<40000;x++); //delay
        }
}
```

OUTLINE

- Data types and time delay
- **I/O programming and Logic operations**
- Data conversion programs
- Accessing code ROM space

I/O PROGRAMMING

- **Byte size I/O**
  - Example: write a program to get a byte of data from P0. If it's 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 an input port
    while(1)
      {
        mybyte=P0;            //get a byte from P0
        if(mybyte<100)
          P1=mybyte;          //send it to P1 if less than 100
        else
          P2=mybyte;          //send it to P2 if more than 100
      }
  }
```

- **Self study: examples 7-9, 7-10**

I/O PROGRAMMING

- **Bit-addressable I/O Programming**
  - Example: write a 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;    //notice the way single bit is declared
void main(void)
  {
    mybit=1;          //make mybit an input
    while(1)
      {
        if(mybit==1)
          P0=0x55;
        else
          P2=0xAA;
      }
  }
```

I/O PROGRAMMING: SFR REGISTERS

- **Access SFR register**
  - Use sfr or sbit
  - Example: read P0, send the result to P1; read in the value of P2.6

```
sfr regA = 0xE0;
sfr P0 = 0x80;
sfr P1 = 0x90;
sbit inbit = 0xA6;          // bit address for P2^6
bit mybit;                  // compiler will automatically allocate memory to mybit
void main(void)
{
        unsigned int z;
        P0 = 0xff;    // input mode
        inbit = 1; //input mode
        for (z = 0; z<50000; z++)
        {
                regA = P0;
                P1 = regA;
                mybit = inbit;          //read P2^6 to mybit
        }
}
```

LOGIC

- **logic operators in C**
  - && (and), || (or), ! (not)
  - Example: if (var1 < 3 && var2 == 1)
    if (!(var > 5))
- **Bit-wise logic operators**
  - Bit-by-bit logical operations: & (and), | (or), ^ (xor), ~ (not)
  - Shifting: << (shift to left) >> (shift to right): NOT cyclic shift!
  - Example

```
#include <reg51.h>
void main(void)
{
        P0=0x35 & 0x0F;
        P1=0x04 | 0x68;
        P2=0x04 ^ 0x78;
        P3=~0x55;
        P0 = 0x35 << 3;
}
```

LOGIC: DATA SERIALIZATION

- **Data serialization**
  - 1. using serial port (Ch. 10)
  - 2. using shifting operators
  - Example: write a C program to bring in a byte of data serially one bit at a time via P1.0. LSB should come first

```
#include <reg51.h>
sbit P1b0 = P1^0;
sbin ACCMSB = ACC^7;
void main(void)
{
        unsigned char x;
        for (x=0; x<8; x++)
        {
                ACCMSB = P1B0;
                ACC = ACC >> 1;
        }
}
```

OUTLINE

- Data types and time delay
- I/O programming and Logic operations
- **Data conversion programs**
- Accessing code ROM space

DATA CONVERSION: PACKED BCD TO ASCII

- **Packed BCD to ASCII**
  - Recall: packed BCD → unpacked BCD → ASCII
  - Example: Write a C program to convert packed BCD 0x29 to ASCII, send the result to P1 and P2

```
#include <reg51.h>
void main(void)
{
        unsigned char x, y, z;
        unsigned char packedBCD = 0x29;
        x = packedBCD & 0x0F;       // extract low nibble
        P1 = x | 0x30;              // unpacked BCD→ASCII
        x = packedBCD & 0xF0;       //extract high nibble
        y = x>>4;                   // shift it to low nibble
        P2 = y | 0x30;              //unpacked BCD → ASCII
}
```

DATA CONVERSION: ASCII TO BCD

- **ASCII to packed BCD**
  - ASCII → unpacked BCD → packed BCD
  - Example: Write C program to convert “47” to packed BCD

DATA CONVERSION: BINARY TO ASCII

- **Binary (or decimal) to ASCII**
  - iteratively divided by 10 and keep the remainder
  - Example: convert 0xFD = 1111 1101 = 253D to ASCII: '2', '5', '3'

```
#include <reg51.h>
void main(void)
{
        unsigned char quotient, remainder, binbyte;
        binbyte = 0xFD;      //0xFD=253
        quotient = binbyte/10;          // quotient = 25
        remainder = binbyte % 10;       // remainder = 3
        P0 = remainder | 0x30;          // P0 = '3'
        binbyte = quotient;             // binbyte = 25;
        quotient = binbyte/10;          // quotient = 2
        remainder = binbyte%10;         // remainder = 5
        P1 = remainder | 0x30;          // P1 = '5'
        P2 = quotient | 0x30;           // P1 = '2'
}
```

OUTLINE

- Data types and time delay
- I/O programming and Logic operations
- Data conversion programs
- **Accessing code ROM space**

RAM AND ROM

- **In 8051, data can be stored in**
  - RAM (e.g. MOV)
  - ROM (e.g. MOVC)
  - External ROM (e.g. MOVX)
- **RAM**
  - Compiler will automatically allocate RAM space for declared variables (demo)
    - R0 – R7: bank 0 (00H – 07H)
    - Variables (including array): address 08 and beyond
    - stack: address right after variables.

RAM AND ROM

- **Accessing ROM in C**
  - To require the compiler store data in ROM, use the “code” keyword
    - Without code keyword, all the data will be stored in RAM
  - The compiler will automatically allocate ROM space for the variables.
  - Example

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