Principle and Interface Techniques of Microcontroller

--8051 Microcontroller and Embedded Systems Using Assembly and C

LI, Guang (李光) **Prof. PhD, DIC, MIET**

WANG, You (王酉) **PhD, MIET**

杭州 • 浙江大学 • 2022

Chapter 7

8051 Programming in C

C语言——最接近汇编的高级语言

- 数据类型的定义
- 程序三种基本结构
  - 顺序、选择、循环
- 赋值语句
- 数据输入输出
- 算术运算
- 逻辑运算
- 判断
- 循环控制
- 数组和指针
- 结构体
- 函数
- 位运算

51指令概述

一、MCS-51指令分类

MCS-51单片机共有111条指令。

1. 按指令所占的字节数分类

①单字节指令49条

②双字节指令46条

③三字节指令16条

指令分类（2）

2．按指令执行时间长短分

①单周期指令64条

②双周期指令45条

③四周期指令2条

指令分类（3）

3. 按功能分为以下五种:		C语言
①数据传送类指令(29条)	$\longleftrightarrow$	赋值语句
②数据运算类指令(24条)	$\longleftrightarrow$	算术运算
③逻辑操作类指令(24条)	$\longleftrightarrow$	逻辑运算
④控制控制类指令(17条)	$\longleftrightarrow$	选择和循环
⑤布尔操作类指令(17条)	$\longleftrightarrow$	位运算

Why Program 8051 in 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
- ✓ You can use code available in function libraries
- ✓ C code is portable to other microcontroller with little of no modification

DATA TYPES

- 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

Unsigned char

- The character data type is the most natural choice
  - 8051 is an 8-bit microcontroller
- Unsigned char is an 8-bit data type in the range of 0 – 255 (00 – FFH)
  - One of the most widely used data types for the 8051
    - Counter value
    - ASCII characters
- C compilers use the signed char as the default if we do not put the keyword *unsigned*

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

Solution:

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

1. Pay careful attention to the size of the data
2. Try to use unsigned *char* instead of *int if possible*

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];
 }
```

Unsigned char

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
  - ✓ 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
  - ✓ temperature

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];
 }
```

Unsigned and Signed int

- ➢ 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

Single Bit

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

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

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 DELAY

- There are two way s to create a time delay in 8051 C
  - ✓Using the 8051 timer (Chap. 9)
  - ✓Using a simple for loop

    be mindful of three factors that can affect the accuracy of the delay
    - The 8051 design
      - – The number of machine cycle
      - – The number of clock periods per machine cycle
    - The crystal frequency connected to the X1 – X2 input pins
    - 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.

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 time unknown
     P1=0xAA;
     for (x=0;x<40000;x++);
    }
  }
```

We must use the oscilloscope to measure the exact duration

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

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.

Solution:

```
#include <reg51.h>
#defind LED P2;
void main(void)
 {
   P1=00; //clear P1
   LED=0; //clear P2
   for (;;) //repeat forever
   {
     P1++; //increment P1
     LED++; //increment P2
   }
 }
```

Ports P0 – P3 are byte-accessable and we use the P0 – P3 labels as defined in the 8051/52 header file.

Byte Size I/O

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
   }
 }
```

Bit-addressable I/O

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, 1, 2, or 3 and y is the bit 0 – 7 of that port

Bit-addressable I/O

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;
     }
  }
```

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.

Solution:

```
#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);
        }
     }
  }
```

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 “The Earth is but One Country” to this LCD.

Solution:

```
#include <reg51.h>
#define LCDData P1 //LCDData declaration
sbit En=P2^0; //the enable pin
void main(void)
 {
   unsigned char message[]=“The Earth is but One Country”;
   unsigned char z;
   for (z=0;z<28;z++) //send 28 characters
    {
     LCDData=message[z];
     En=1; //a high-
     En=0; //-to-low pulse to latch data
    }
 }
```

Accessing SFR Addresses 80 - FFH

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.

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;
       P1=0x55;
       P2=0x55;
       MSDelay(250);
       P0=0xAA;
       P1=0xAA;
       P2=0xAA;
       MSDelay(250);
     }
  }
```

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

Accessing SFR Addresses 80 - FFH

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;
     }
  }
```

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

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 generation of 8051 microcontrollers.

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.

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 Operations

Bit-wise Operators 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

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	

Bit-wise Operators in C

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
 }
```

Bit-wise Operators in C

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);
    }
 }
```

Bit-wise Operators in C

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
     }
  }
```

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

Solution:

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

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.

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;
 }
```

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.

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;
 }
```

Checksum Byte in ROM

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’;
  }
```

Binary (hex) to Decimal and ASCII Conversion

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

RAM Data Space Usage by 8051 C Compiler

- 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

RAM Data Space Usage by 8051 C Compiler

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];
  }
```

RAM Data Space Usage by 8051 C Compiler

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;
     }
  }
```

8052 RAM Data 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
- We compile the C programs for the 8052 microcontroller
  - ✓ Use the reg52.h header file
  - ✓ Choose the8052 option when compiling the program

Compile and single-step the following program on your 8051 simulator. 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

Compare and contrast the following programs and discuss the advantages and disadvantages of each one.

(a)

```
#include <reg51.h>
void main(void)
 {
   P1='H';
   P1='E';
   P1='L';
   P1='L';
   P1='O';
 }
```

Short and simple, but the individual characters are embedded into the program and it mixes the code and data together

(b)

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

Use the RAM data space to store array elements, therefore the size of the array is limited

Use a separate area of the code space for data. This allows the size of the array to be as long as you want if you have the on-chip ROM.

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

However, the more code space you use for data, the less space is left for your program code

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 (Chap. 10)
  - ✓ 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

Data Serialization

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.

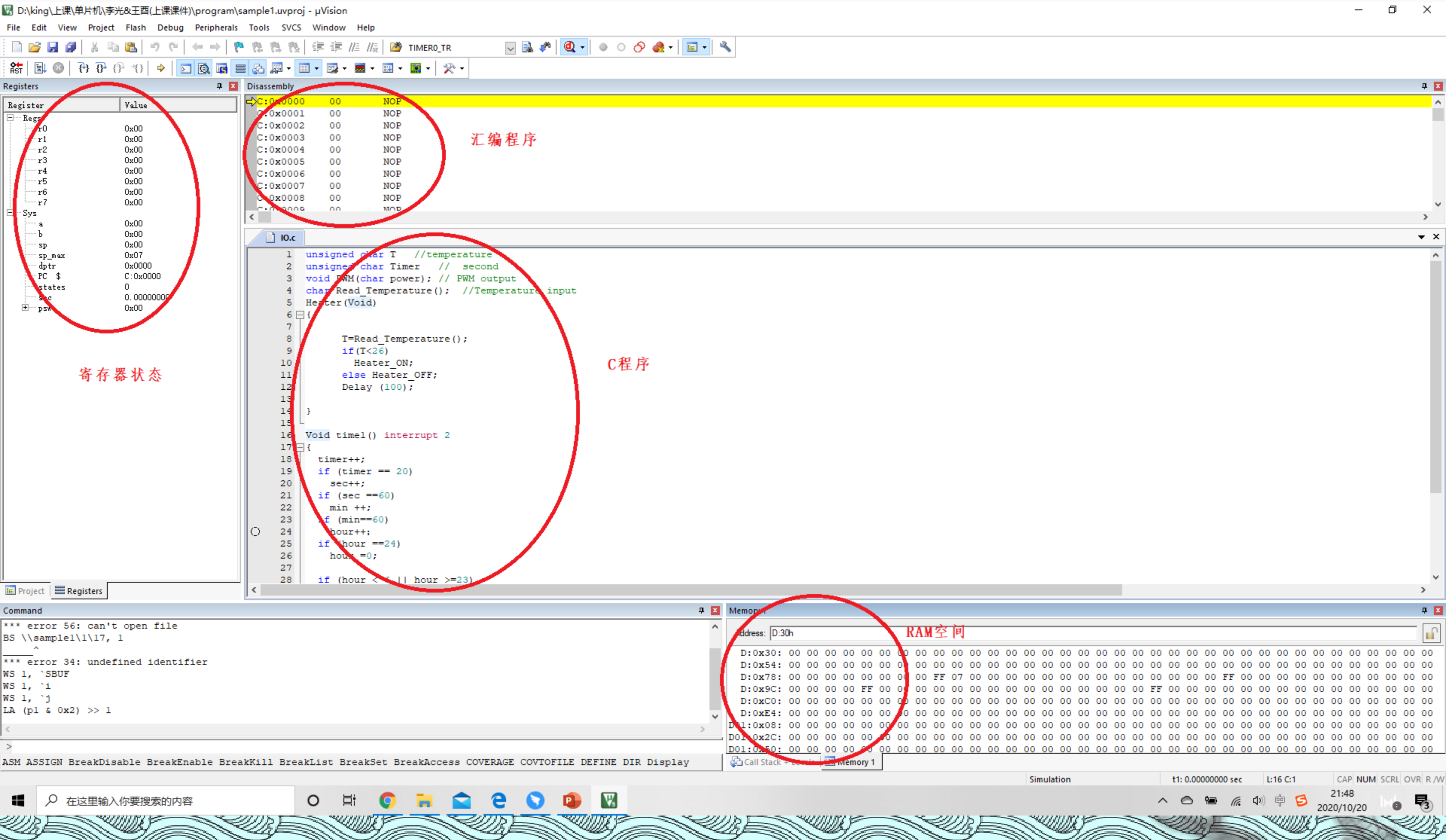
Solution:

```
#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
    }
 }
```

作用 排序

- 用C语言编程，实现对数组A[15]={27, 5,32,47,38,235,79,17,187, 58,23,35,211,104,9};进行从小到大排序，排序后的数组放在RAM。把调试结果和源程序附在实验报告中。
- 简要回答
  - 1：本例中采用什么排序算法最合适（时间/空间复杂度）
  - 2：和上一次作业相比，用C语言完成作业，和用汇编完成作业，有哪些方便之处
  - 3：试对比编译器完成的汇编代码，通过检查RAM空间，看看最后排序好的数组放在了什么地址上。

Keil程序界面



THANK YOU!!