第6章 STC单片机C语言编程模型

何宾

2015.02

本章主要内容

- C语言发展历史
- C语言的优势
- 设计第一个C程序
- 常量和变量
- 数据类型
- 运算符
- 描述语句
- 数组

本章主要内容

- 指针
- 函数
- 预编译指令
- 复杂数据结构
- C程序中使用汇编语言
- C语言端口控制实现
- C语言中断程序实现

C语言发展历史

C语言之所以命名为C，因为 C语言源自Ken Thompson发明的B语言，而 B语言则源自BCPL语言。

- 1967年，剑桥大学的Martin Richards对CPL语言进行了简化，于是产生了BCPL（Basic Combined Programming Language）语言。
- 1970年，美国贝尔实验室的 Ken Thompson，以BCPL语言为基础，设计出很简单且很接近硬件的B语言（取BCPL的首字母）。并且他用B语言写了第一个UNIX操作系统。

C语言发展历史

- 1972年，美国贝尔实验室的 D.M.Ritchie 在B语言的基础上设计出了一种新的语言，他取BCPL的第二个字母作为这种语言的名字，这就是C语言。

C语言的优势

作为目前单片机开发中广泛采用的一种高级语言，C语言有着其独特的优势，主要体现在：

- 基础性
  - C语言语法简单精炼，概念少，效率高，包含了基本的编程元素。因此，通过学习C语言就可以掌握最基本的编程概念。此外，很多高级语言（C++、Java等）都参考了C语言的程序设计思想。
- 易学性
  - 由于C语言结构简单，对初学者来说，学习成本低，时间短，能够快速掌握高级语言的编程方法。

C语言的优势

- 广泛性
  - C语言不但执行效率高（比C++、Java都高），而且应用广泛。在桌面系统和嵌入式系统的程序设计中，C语言仍然被广泛使用。
- 通用性
  - 与汇编语言相比，由于封装了底层硬件细节，对于广大设计者来说，不必对底层硬件直接操作。取而代之的是，通过C语言所提供的对逻辑行为模型抽象的建模能力对单片机进行控制。
- 移植性
  - 如果开发人员需要将代码交给其它开发人员处理，他们无需掌握那些为发挥汇编语言的最大效率而需要的所有技巧便可立即开始修改代码。

设计第一个C程序
--建立新的设计工程

建立新设计工程的步骤主要包括：

- **打开μVision5集成开发环境。**
- **在μVision5集成开发环境主界面主菜单下，选择Project->New μVision Project...。**
- **出现Create New Project对话框界面。在文件名右侧的文本框中输入top。**
- **单击OK按钮。**
- **出现Select a CPU Data Base File对话框界面。在该界面中的下拉框中，选择STC MCU Database选项。**
- **单击OK按钮。**

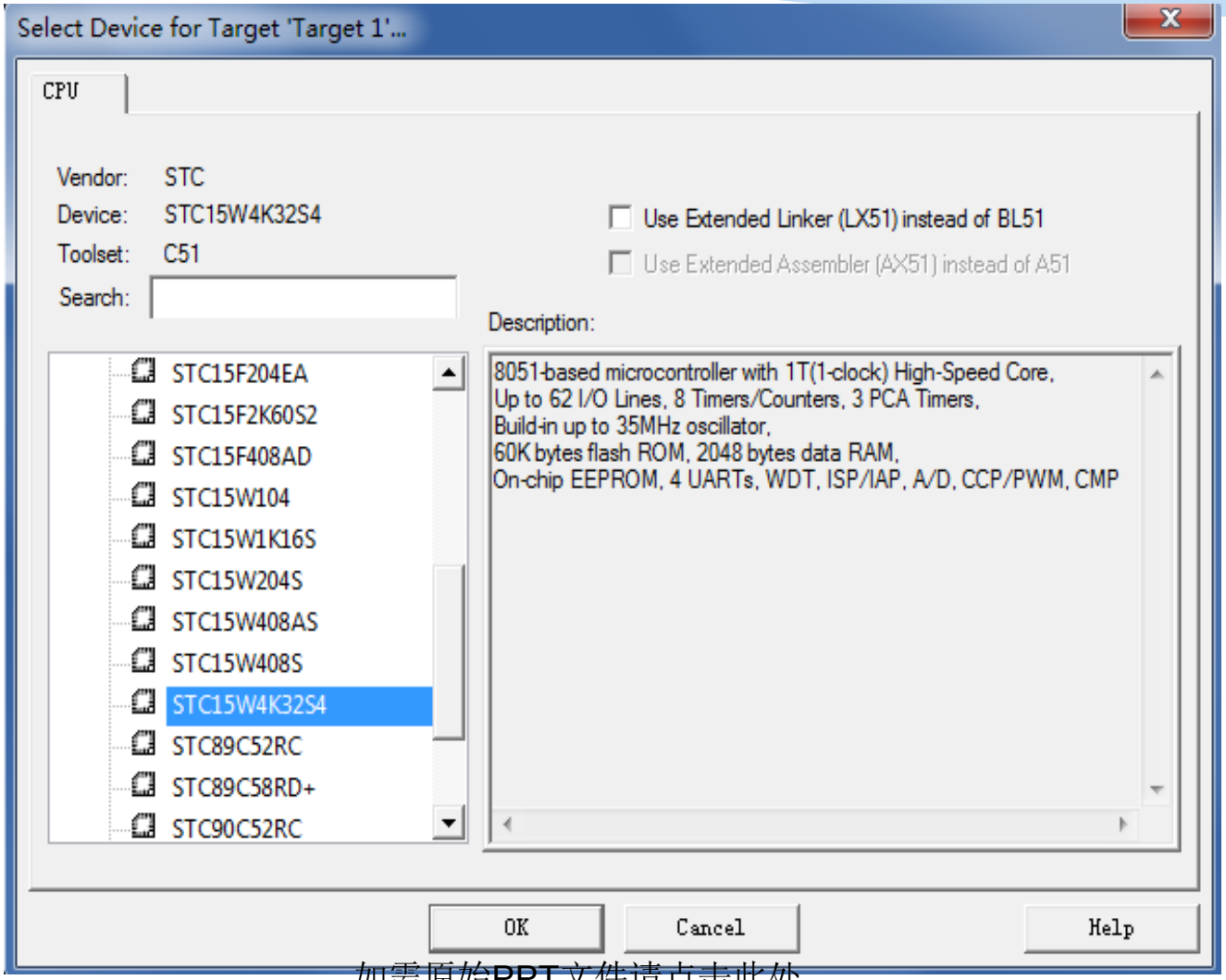
设计第一个C程序
--建立新的设计工程

- 出现Select Device for Target‘ Target 1’ ...对话框界面。在该界面中左侧的窗口中，找到并展开STC前面的‘+’。在展开项中，找到并选择STC15W4K32S4，如右图所示。

注：

全书设计使用STC的IAP15W4K58S4单片机。该单片机属于STC15W4K32S4系列。

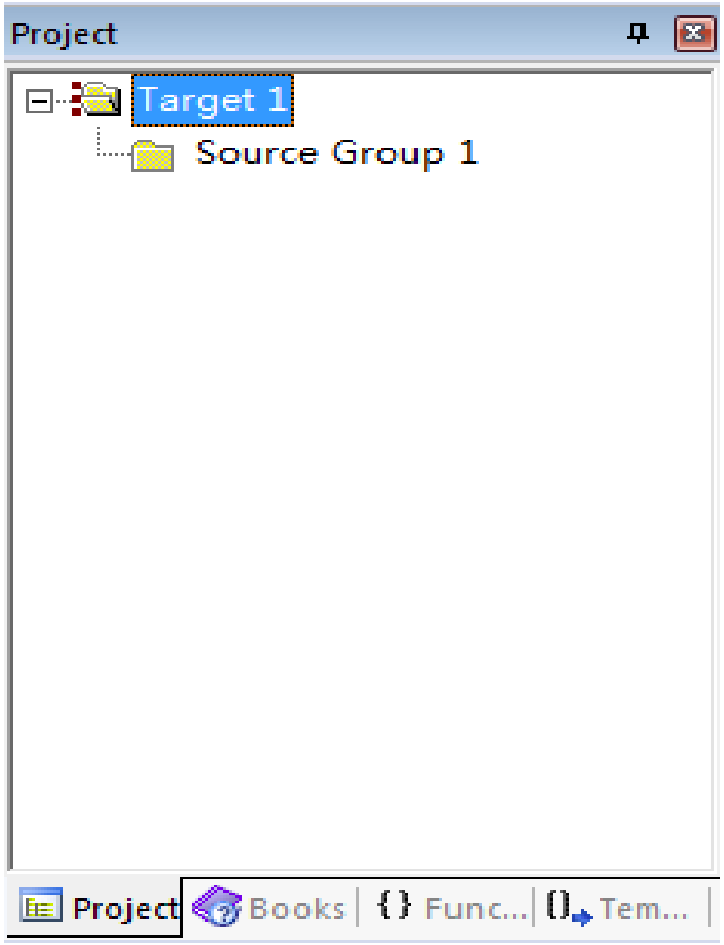
设计第一个C程序
--建立新的设计工程



设计第一个C程序
--建立新的设计工程

- 单击OK按钮。
- 出现Copy‘ STARTUP.A51’ to Project Folder and Add File to Project?对话框界面。该界面提示是不是在当前设计工程中添加STARTUP.A51文件。
- 单击“否（N）”按钮。
- 在主界面左侧窗口中，选择Project标签。在该标签窗口下，给出了工程信息，如下图所示。

设计第一个C程序
--建立新的设计工程



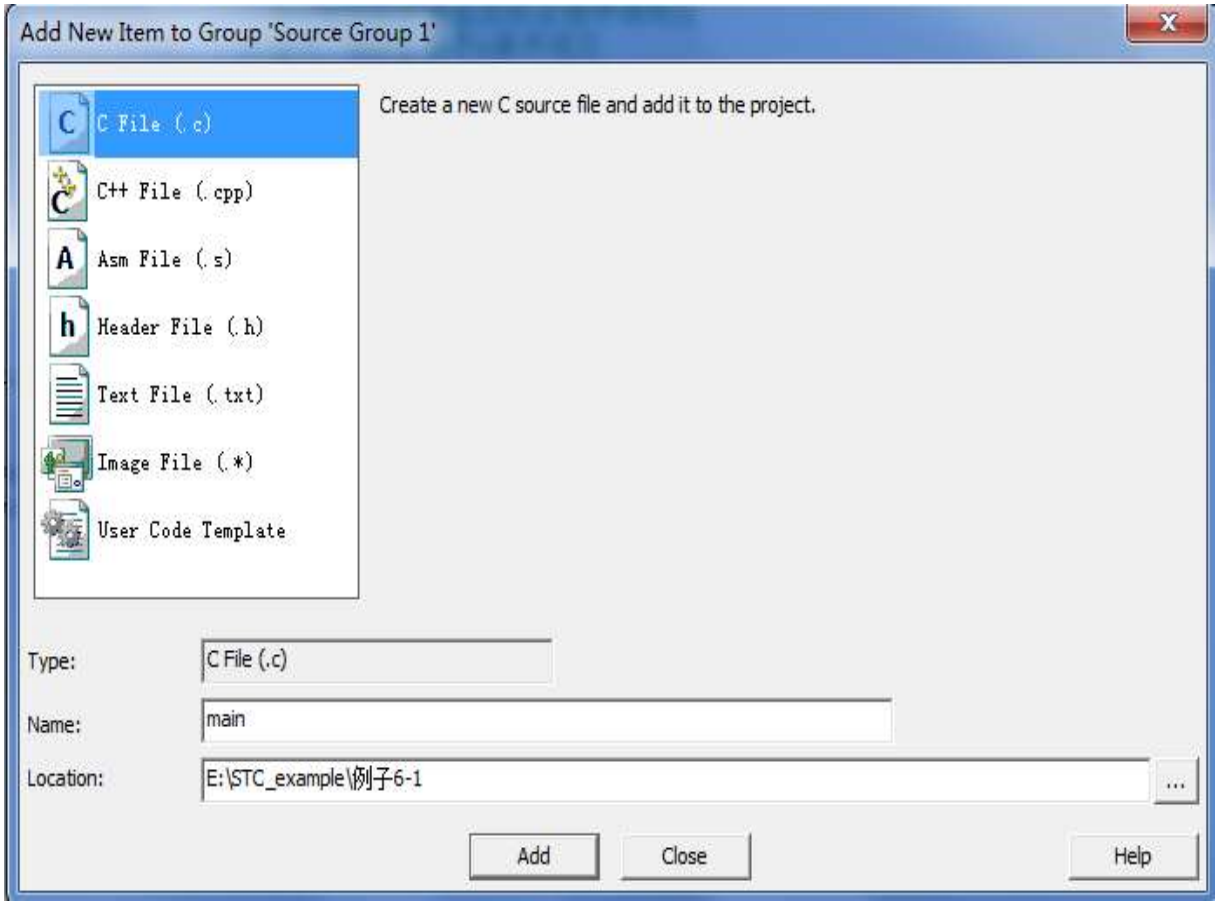
设计第一个C程序
--添加新的C语言文件

添加C语言文件的步骤主要包括：

- 在Project窗口界面下，选中Source Group 1，单击右键，出现浮动菜单。在浮动菜单内，选择Add New Item to Group ‘Source Group 1’ 选项。
- 出现Add New Item to Group’ Source Group 1’对话框界面，如右图所示，按下面设置参数：
  - 在该界面左侧窗口中，选中C File(.c)。
  - 在Name右侧的文本框中输入main。

注：该C语言的文件名字为main.c。

设计第一个C程序
--添加新的C语言文件



设计第一个C程序
--添加新的C语言文件

- 点击Add按钮。
- 在Project窗口中的Source Group 1子目录下，添加了名字为main.c的C语言文件。
- 在右侧窗口中，自动打开了main.c文件。

设计第一个C程序
--添加新的C语言文件

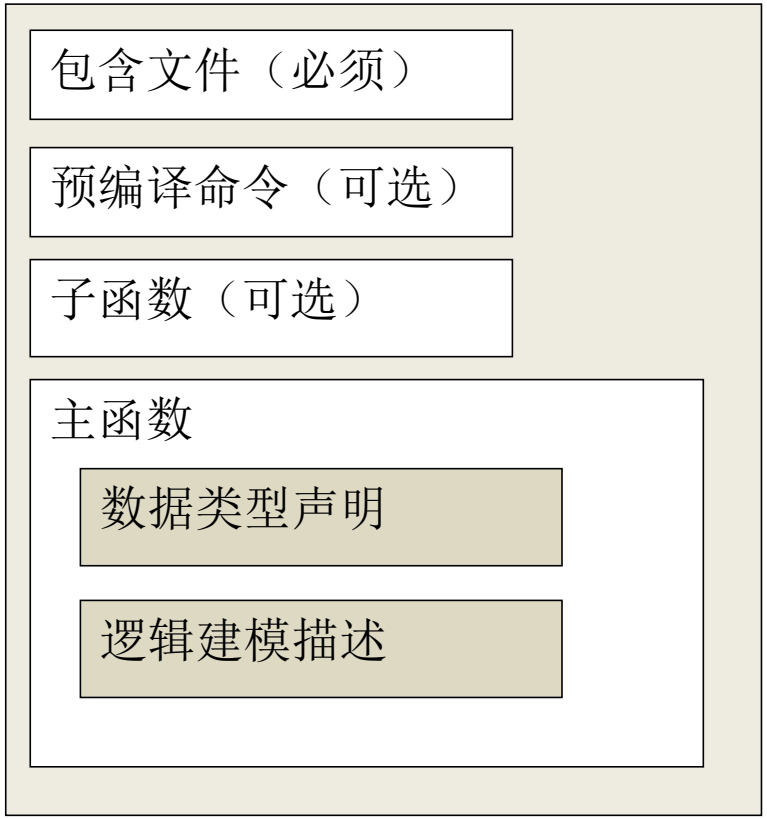
- 输入C语言代码，如下图所示。

main.c

```
1  #include "stdio.h"                    //预处理命令，调用库函数
2  #include "reg51.h"                    //预处理命令，调用库函数
3  int max(int a, int b)                 //定义max函数，a，b为形式参数
4  {                                     //{ }为函数的边界
5    if(a>b)                             //如果a大于b条件成立
6      return a;                         //将a作为返回值返回到主程序调用处
7    else                                //否则，a小于等于b条件成立
8      return b;                         //将b作为返回值返回到主程序调用处
9  }
10 main()                                //主函数
11 {
12   int i,j,k,l;                        //声明数据类型i,j,k
13   SCON= 0x52;                         /*8051单片机串口初始化*/
14   TMOD = 0x20;
15   TCON = 0x69;
16   TH1 = 0xF3;
17   scanf("%d %d",&i,&j);               //输入变量i和j的值
18   l=i+j;                              //变量i和变量j的值求和，送给l
19   k=max(i,j);                         //调用max函数，求取最大值k
20   printf("i + j =%d\n",l);            //输出l的值
21   printf("max value is %d\n",k);      //打印k的值
22 return 1;
23 }                                     //程序结束
```

- 保存设计代码。

设计第一个C程序
--C语言程序结构



设计第一个C程序
--C语言程序结构

本质上，一个C语言文件是由若干函数构成。

- 包含文件 ：指在当前的主程序中，将另一个文件的内容包含进去。
- 预编译命令：预编译命令可以在很大程度上为C语言本身提供许多功能和符号等方面的扩充，增强了C语言的灵活性和方便性。

设计第一个C程序
--C语言程序结构

- 子函数：子函数是一个模块的集合。在这个模块中，定义了所用到的变量类型、以及所要实现的逻辑模型功能。
- 主函数：主函数时整个C文件中最核心的部分，由它对整个C语言文件的其它部分进行调用和处理。在标准C语言中，主函数总是由main关键字标识，通过{ }符号确定主函数的边界。主函数主要包含数据类型声明语句和逻辑建模描述语句。

设计第一个C程序
--C语言程序结构

- 数据类型声明语句

在一个主函数中，总是需要对数据进行处理。在C语言中，通过数据类型声明语句，就可以区分所处理数据的对象。典型的，包括：整数、字符、浮点数。对于更复杂的数据来说，包括：数组、结构体和指针等。在设计中，声明了四个整数i、j、k、l。

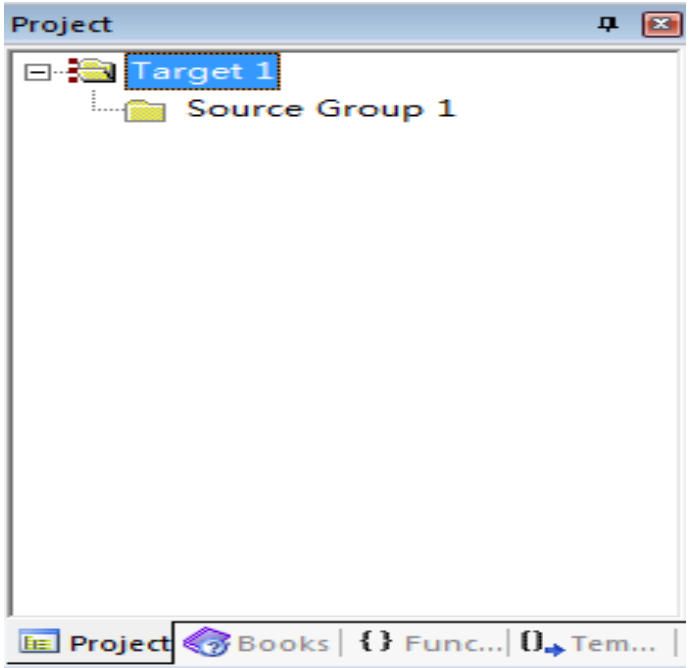
- 逻辑建模描述语句

主函数的目的就是要对所使用到的数据进行处理，处理的过程可以通过逻辑建模语句进行说明。典型地，包含：顺序描述语句、条件判断语句、子函数调用语句、输入scanf和输出函数printf语句等。

设计第一个C程序
--设计建立

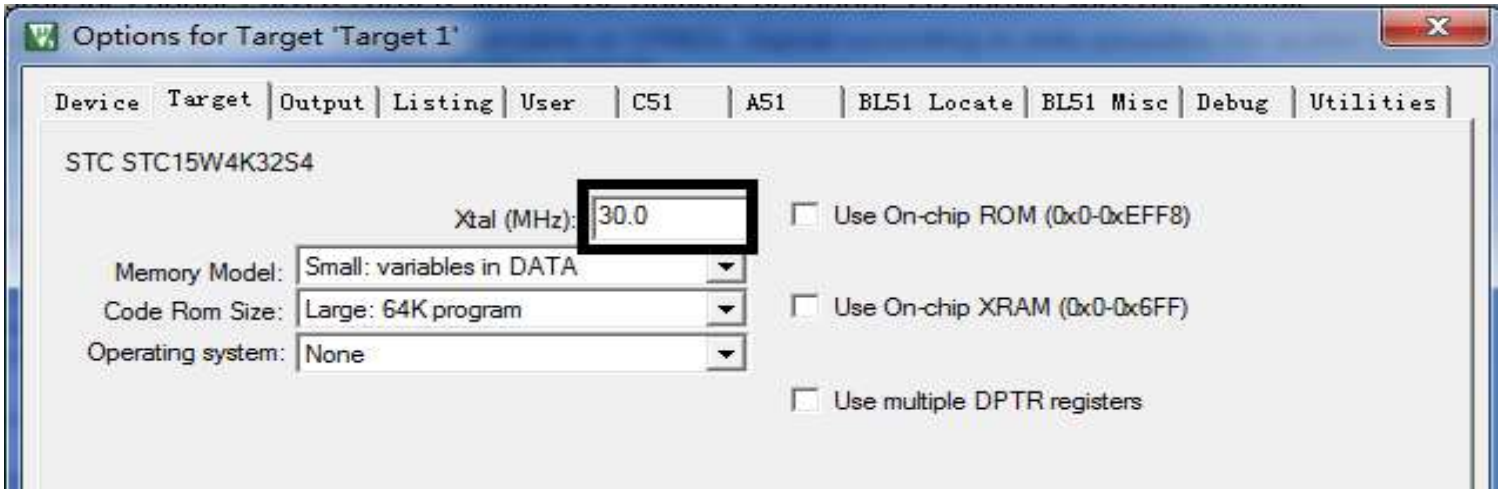
对设计建立（Build）参数进行设置，并实现对设计的建立过程，其步骤主要包括:

- 在如右图所示的窗口中，选中Target 1文件夹，并单击右键，出现浮动菜单。在浮动菜单内，选中Options for Target ‘Target 1’...选项。



设计第一个C程序
--设计建立

- 出现Options for Target ‘Target 1’对话框界面，在该界面中，点击Target标签。在该标签界面中，按下面设置参数:



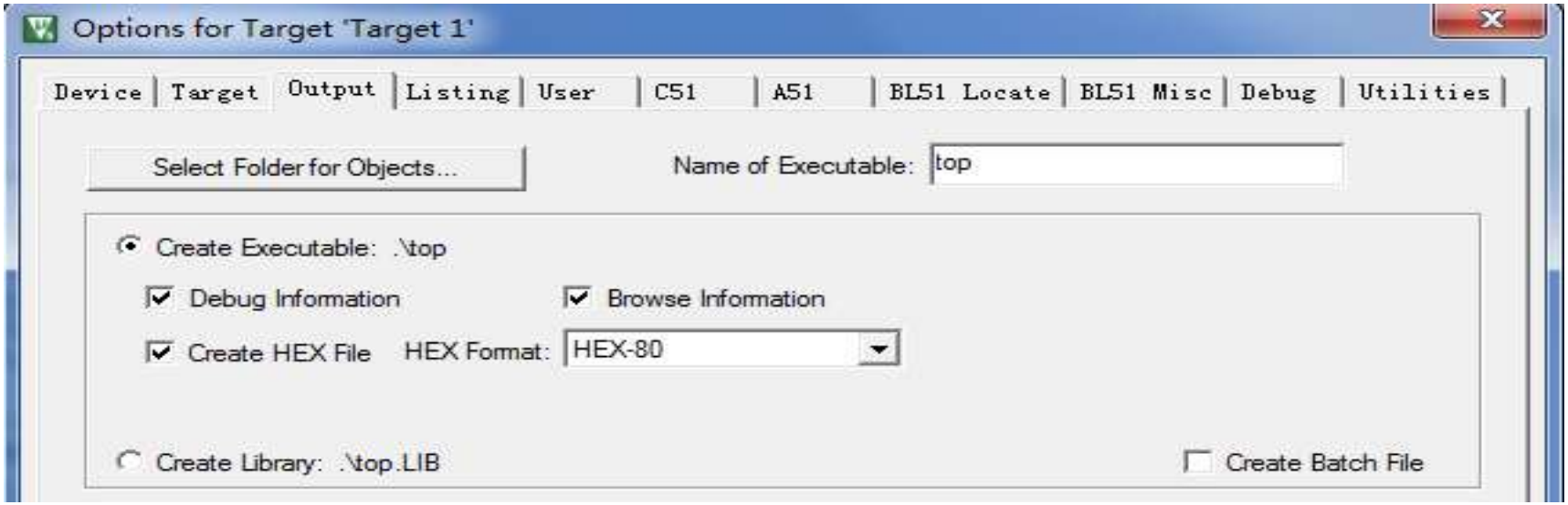
- 在Xtal（MHz）右侧文本框中，输入30.0。

注：该设置用于确定单片机晶体振荡器的工作频率。

- 其余按默认设置。

设计第一个C程序
--设计建立

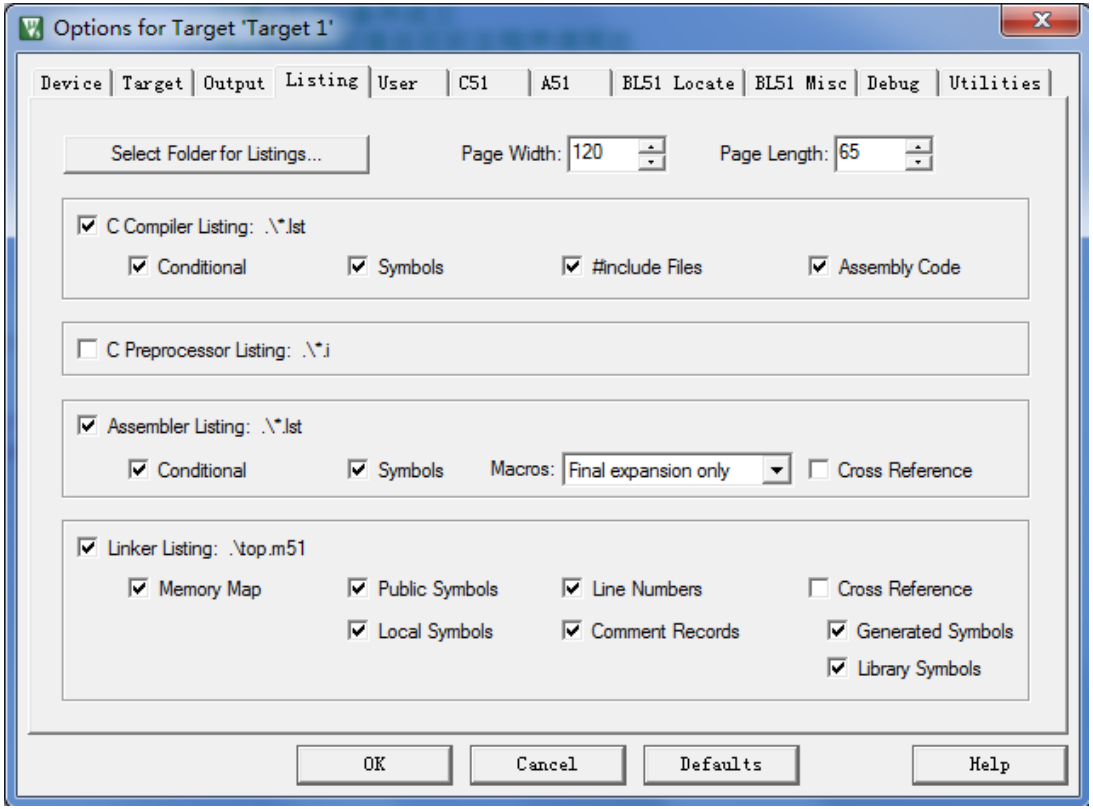
- 在该对话框界面下，再次选中Output标签。在该标签界面下，选中Create HEX File前面的复选框，如下图所示.



注：该设置用于说明在建立过程结束后，生成可用于编程STC单片机的十六进制HEX文件。

设计第一个C程序
--设计建立

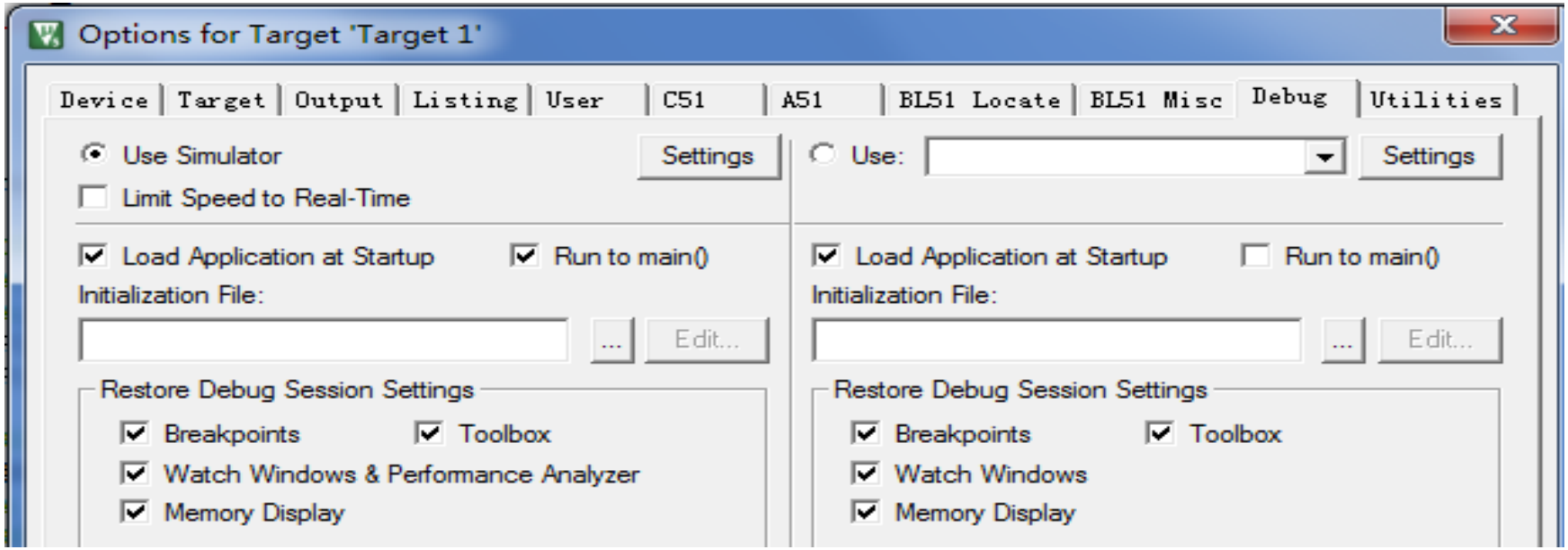
- 在该对话框界面下，再次选中Listing标签。在该标签界面下，选中Symbols、#include Files和Assembly Code前面的复选框。



注：该设置用于说明在.lst文件中，给出包含文件、符号和汇编代码的相关信息。

设计第一个C程序
--设计建立

- 在该对话框界面下，再次选中Debug标签。在该标签界面下，选中Use Simulator前面的单选框。



注：该设置用于说明在建立过程结束后，先进行脱离实际硬件环境的软件仿真。

设计第一个C程序
--设计建立

- 单击OK按钮，退出目标选项对话框界面。
- 在主界面主菜单下，选择Project->Build target。开始对设计进行建立的过程。

注：该过程对C文件，进行编译，最后生成可执行二进制文件和HEX文件。

设计第一个C程序
--设计运行和分析

设计运行和分析的步骤主要包括：

- **打开main.c文件，设置断点，如下图所示。**

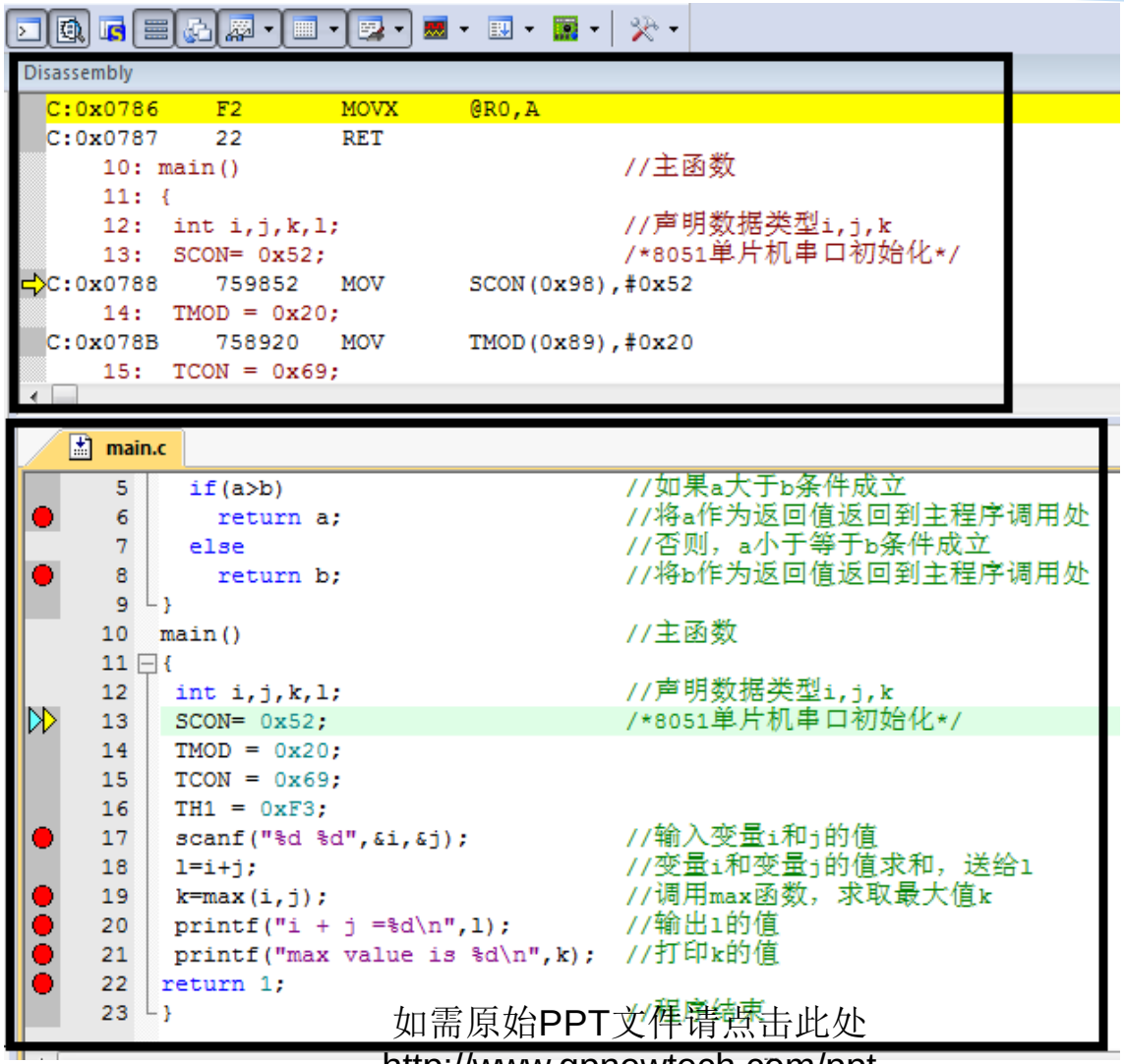
main.c

```
1  #include "stdio.h"
2  #include "reg51.h"
3  int max(int a, int b)
4  {
5    if(a>b)
6      return a;
7    else
8      return b;
9  }
10 main()
11 {
12  int i,j,k,l;
13  SCON= 0x52;
14  TMOD = 0x20;
15  TCON = 0x69;
16  TH1 = 0xF3;
17  scanf("%d %d",&i,&j);
18  l=i+j;
19  k=max(i,j);
20  printf("i + j =%d\n",l);
21  printf("max value is %d\n",k);
22 return 1;
23 }
```

设计第一个C程序
--设计运行和分析

- 在μVision主界面主菜单下，选择Debug->Start/Stop Debug Session，进入调试器模式。
- 如下图所示，在Disassembly窗口中，给出了每条C语言指令，所对应的汇编语言。程序停在main（ ）主程序的入口。表示该C语言主程序位于STC单片机的程序存储器Flash空间的0x0788的位置。

设计第一个C程序
--设计运行和分析

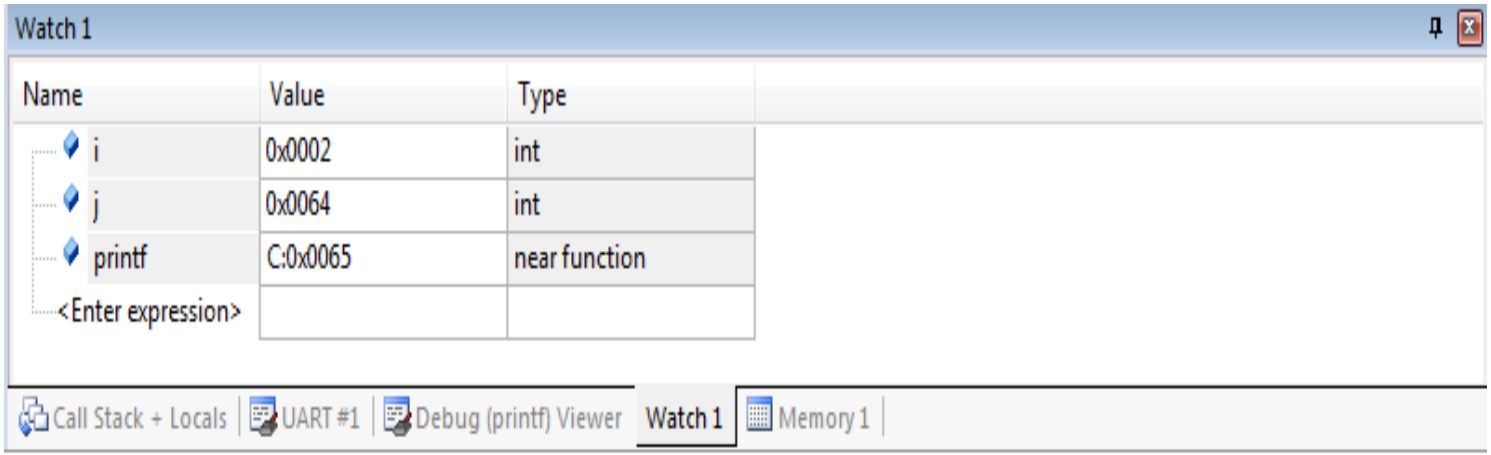


设计第一个C程序
--设计运行和分析

- 按F5键运行程序，运行完设置断点17行的代码。运行完scanf表示要输入变量i和j的值。
- 在当前调试模式主界面下，选择View->Serial Windows->UART #1。
- 在当前调试主界面右下侧的窗口内，输入两个数2和100，如下图所示。然后，按回车键。

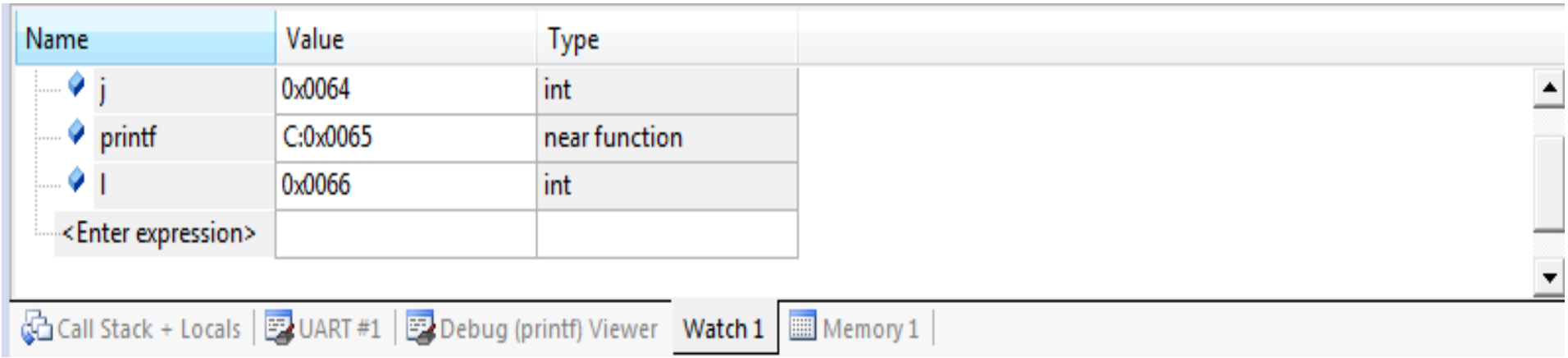
设计第一个C程序
--设计运行和分析

- 程序自动停到了第19行的断点处。从Disassembly窗口中可以看到，该子函数位于STC单片机程序存储器0x07BC的位置。
- 在当前调试主界面主菜单下，选择View->Watch Window->Watch 1。
- 出现Watch 1窗口界面，如下图所示。



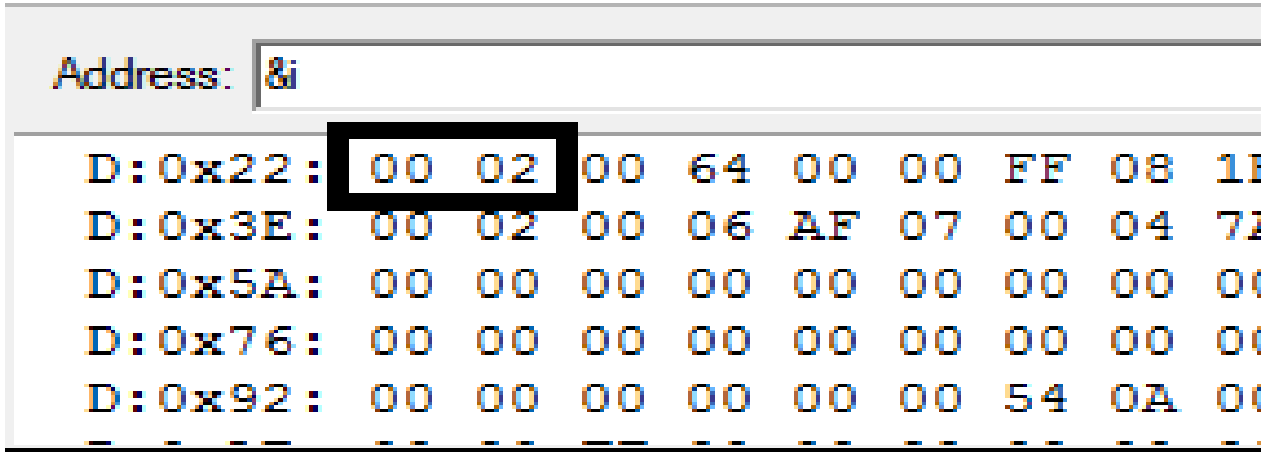
设计第一个C程序
--设计运行和分析

- 在该界面中，选中 <Enter expression>，并左键单击<Enter expression>。然后，输入I（变量I）可以看到在Value所对应的行显示0x0066，类型显示int，如下图所示。



设计第一个C程序
--设计运行和分析

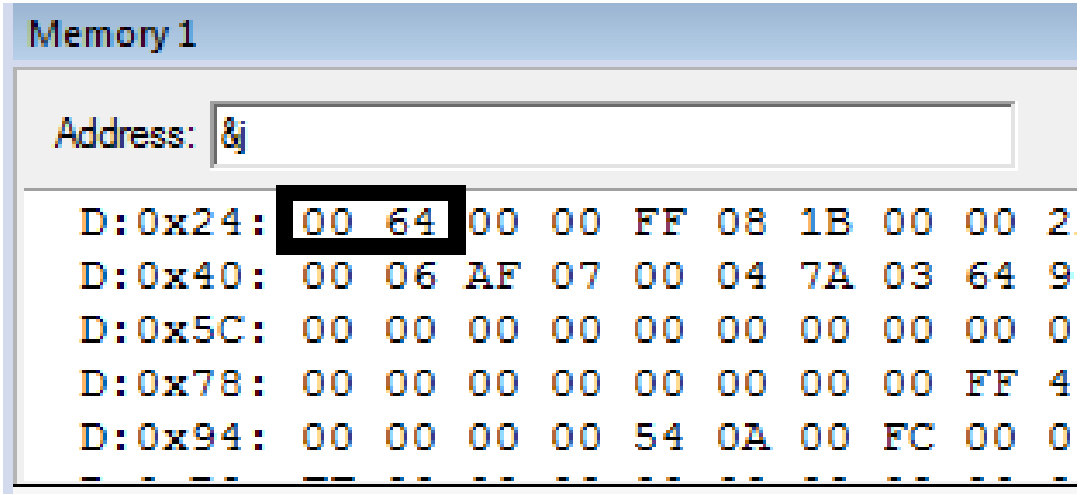
- 在当前调试模式主界面右下方的Memory窗口中，在Address：右侧的文本框中输入&i，可以看到下面显示D:0x22。然后，右侧显示00 02，如图所示。



注：表示所声明的整型变量i，在单片机的内部数据区的0x22的位置，其值占用了两个字节。

设计第一个C程序
--设计运行和分析

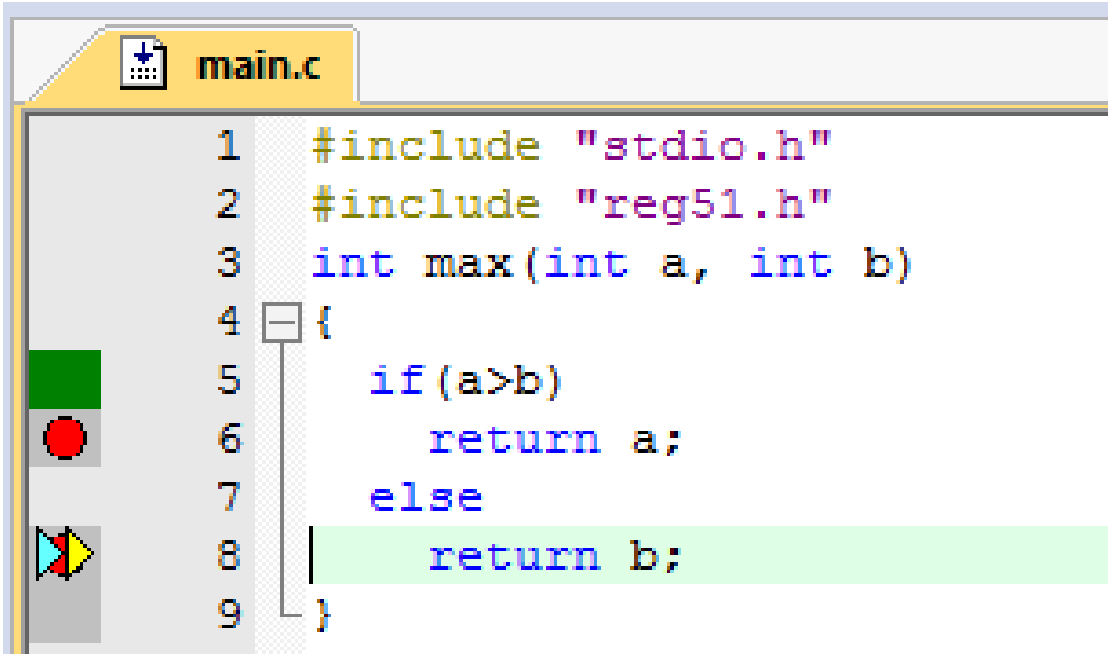
- 在当前调试模式主界面右下方的Memory窗口中，在Address：右侧的文本框中输入&j，可以看到下面显示D:0x24。然后，右侧显示00 64，如图所示。



注：表示所声明的整型变量j，在单片机的内部数据区的0x24的位置，其值占用了两个字节。

设计第一个C程序
--设计运行和分析

- 按F5按键，继续运行程序，断点停到了第8行。说明条件判断a>b不成立，即a≤b成立，将b的值作为函数max的返回值。



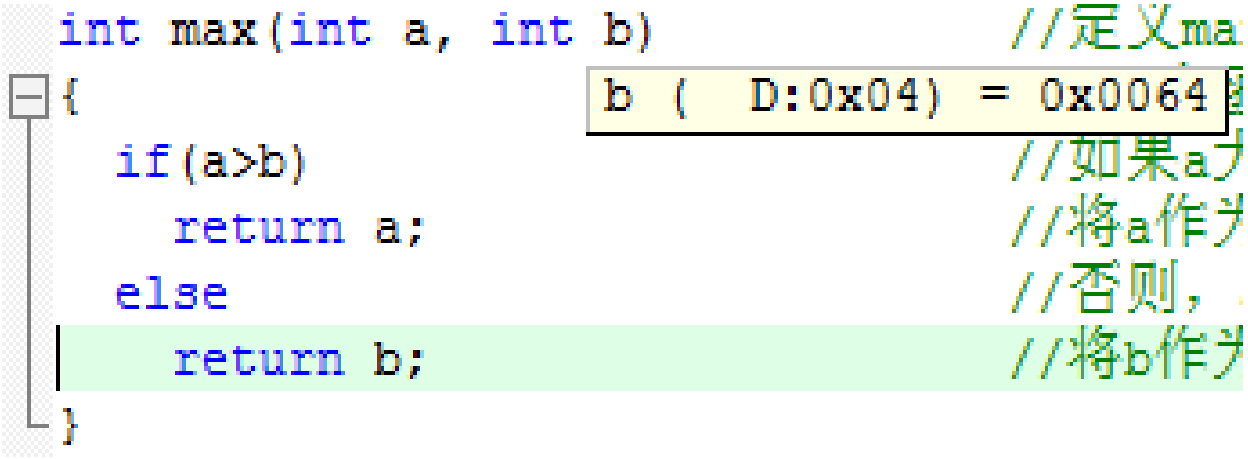
设计第一个C程序
--设计运行和分析

- 将鼠标光标放到第3行代码int a的字母a上，出现浮动菜单，表示形参a位于单片机片内RAM地址为0x06的位置，此时的值为0x0002，说明输入变量i的值传递给了函数的形式变量参数a。

```
3   int max(int a, int b)                /
4   {                a (  D:0x06) = 0x0002
5     if(a>b)                            /
6       return a;                        /
7     else                               /
8       return b;                        /
9   }
```

设计第一个C程序
--设计运行和分析

- 将鼠标光标放到第3行代码int b的字母b上，出现浮动菜单，表示形参b位于单片机片内RAM地址为0x04的位置，此时的值为0x0064，说明输入变量j的值传递给了函数的形式变量参数b。



- 按F5按键几次，继续运行程序，直到运行到第22行为止。

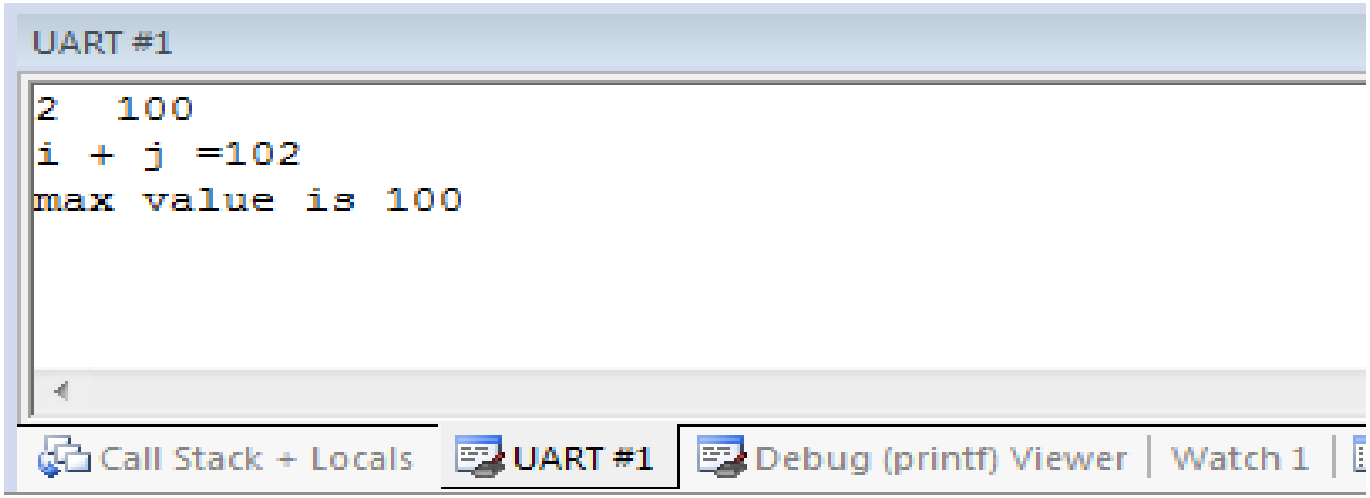
设计第一个C程序
--设计运行和分析

- 看到在当前调试主界面UART #1窗口中，打印如下的信息，如图所示：

  i + j =102

  max value is 100

这两条信息由程序代码中的printf语句控制。



常量和变量
--常量

在程序执行过程中，其值不发生改变的量称为常量

常量的分类

常量	说明
直接常量（字面量）	可以立即拿来用，无需任何说明的量，例如： （1）整型常量：12、0、-3 （2）实型常量：4.6、-1.23 （3）字符常量：‘a’、‘b’
符号常量	用标识符代表一个常量。在C语言中，可以用一个标识符来表示一个常量，称为符号常量。

常量和变量
--常量

表中：

- 符号常量在使用之前必须通过宏定义语句进行定义定义，其一般形式为：

  #define 标识符 常量

其中：

- #define也是一条预处理命令（预处理命令都以"#"开头），称为宏定义命令（在后面预处理程序中将进一步介绍），其功能是把该标识符定义为其后的常量值。一经定义，以后在程序中所有出现该标识符的地方均代之以该常量值。使用符号常量的好处就在于程序的可读性好，并且容易修改。

常量和变量
--变量

其值可以改变的量称为变量。一个变量应该有一个标识符，在内存中占据一定的存储单元。在使用变量前必须要事先定义变量。

- 变量定义一般放在函数体的开头部分。变量定义的一般形式为：

类型说明符 变量名, 变量名, ...;

在书写变量定义时，应注意以下几点：

- 允许在一个类型说明符后，定义多个相同类型的变量。各变量名之间用逗号间隔。类型说明符与变量名之间至少用一个空格分隔。
- 最后一个变量名之后必须以分号结尾。
- 变量定义必须放在变量使用之前。一般放在函数体的开头部分。

常量和变量
--变量

- **变量赋值**
  - 变量可以先定义再赋值，也可以在定义的同时进行赋值；在定义变量的同时赋初值称为初始化。
  - 在变量定义中赋初值的一般形式为：

    类型说明符 变量1= 值1, 变量2= 值2, ……;
  - 而在使用时赋值的一般形式为：

    变量1= 值1;

    变量2= 值2;

数据类型
--标准C语言所支持的类型

整数型

根据所占字节大小和表示的范围，整型数据可分为：

- 基本型：类型说明符为int，在内存中占2个字节。
- 短整型：类型说明符为short int或short。所占字节和取值范围同基本型。
- 长整型：类型说明符为long int或long，在内存中占4个字节。

注：（1）默认基本类型、短整型和长整型数都是有符号数。如果需要指明它们是无符号数，应该在类型说明符前面添加unsigned关键字。

（2）对于有符号数来说，在计算机中用补码表示。

数据类型
--标准C语言所支持的类型

C语言中各类整型数据所分配的内存字节数及表示范围

类型说明符	数的范围	字节数
int	-32768~32767，即 -2^{15}~$(2^{15}-1)$	2
short int	-32768~32767，即 -2^{15}~$(2^{15}-1)$	2
long int	-2147483648~2147483647，即 -2^{31}~$(2^{31}-1)$	4
unsigned int	0~65535，即 0~$(2^{16}-1)$	2
unsigned short int	0~65535，即 0~$(2^{16}-1)$	2
unsigned long	0~4294967295，即 0~$(2^{32}-1)$	4

- **上面提到的整数，都是十进制。在C语言中，常用的还有八进制和十六进制。在C语言程序中是根据前缀来区分各种进制数的。**

数据类型
--标准C语言所支持的类型

十进制数

- 十进制数没有前缀。其数字取值为0～9。
  - 比如：237、-568、65535、1627。

八进制数

- 八进制数必须以0开头，即以0作为八进制数的前缀。数字取值为0～7。八进制数通常是无符号数。
  - 比如：015、0101、0177777。

十六进制数

- 十六进制数的前缀为0X或0x。其数字取值为0~9，A~F或a~f。
  - 比如： 0X2A、0XA0、0xFFFF。

数据类型
--标准C语言所支持的类型

此外，可以用后缀“L”或“l”来表示长整型数。例如：

- 十进制长整型数：158L、358000L；
- 八进制长整型数：012L、077L、0200000L；
- 十六进制长整型数：0X15L、0XA5L、0X10000L。

注：对于长整型数158L和基本整型数158在数值上并无区别。但对158L，因为是长整型数，C编译系统将为它分配4个字节存储空间。而对158，因为是基本整型，只分配2个字节的存储空间。

注：无符号数也可用后缀表示，整型数的无符号数的后缀为“U”或“u”。

数据类型
--标准C语言所支持的类型

【例】整型数声明和使用的例子

```
void main()
{
    int i=32000,j=32000,h;
    unsigned int m=100,n=0x200;
    long int k=10000,l=-40000;
    h=i+j;
}
```

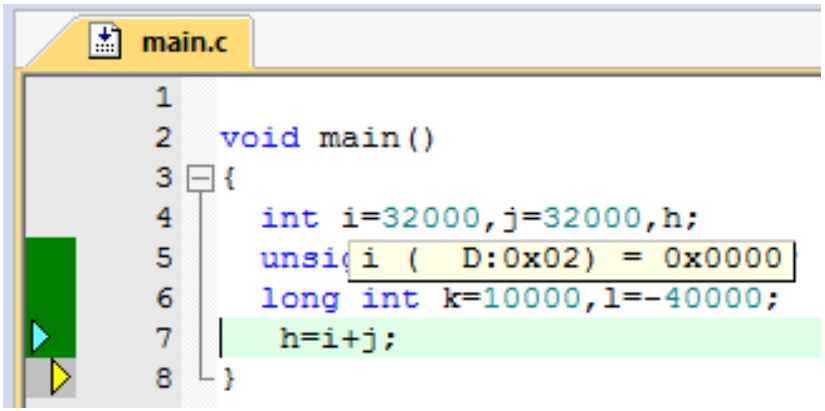
数据类型
--标准C语言所支持的类型

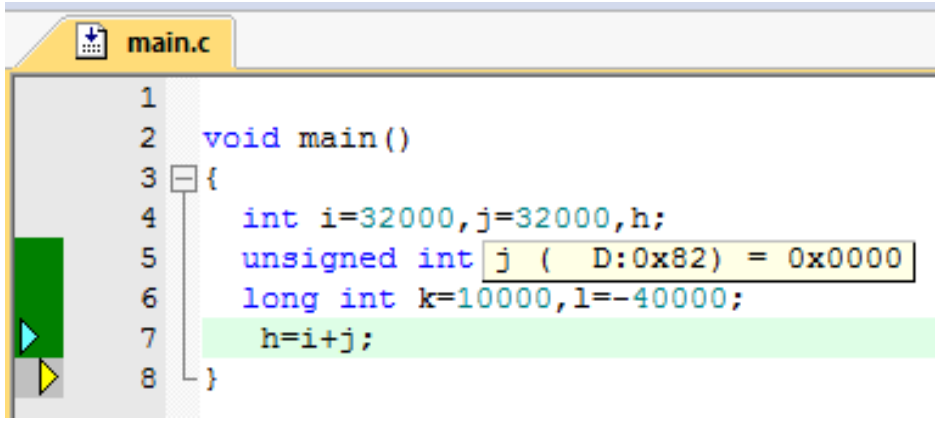
下面对该例子进行分析，分析步骤主要包括：

- **进入本书所提供资料的STC_example\例子6-2\目录下，在Keil μVision5集成开发环境下打开该设计。**
- **在集成开发环境主界面主菜单下，选择Debug->Start/Stop Debug Session。**
- **在调试器模式下，单步运行该程序，一直到程序的末尾。**

数据类型
--标准C语言所支持的类型

- 将鼠标光标分别放到变量i和j的名字上，读者会发现给出的信息是变量i在D:0x02的位置上，其保存的值为0x0000，如左图所示；类似的，给出的信息是变量j在D:0x82的位置上，其保存的值为0x0000，如右图所示。





读者自然会提出疑问，明明已经给i和j进行了赋值操作，为什么显示信息是0？

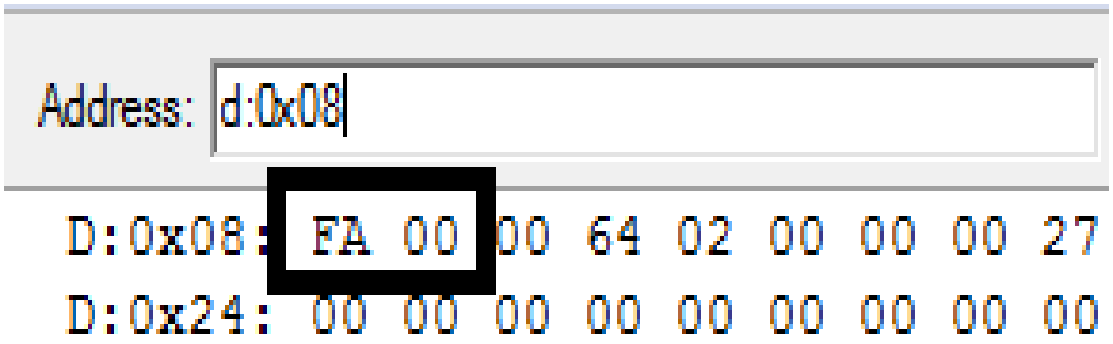
数据类型
--标准C语言所支持的类型

- 在Disassembly窗口中，找到h=i+j的反汇编程序，如下图所示。

```
        7:      h=i+j;
C:0x0026    2400        ADD         A,#0x00
C:0x0028    F509        MOV         0x09,A
C:0x002A    747D        MOV         A,#0x7D
C:0x002C    347D        ADDC        A,#0x7D
C:0x002E    F508        MOV         0x08,A
```

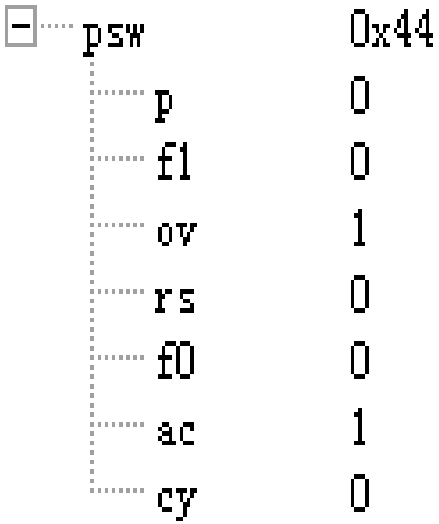
  - 在赋值操作的时候，是两个立即数0x7D直接相加，即：采用的是立即数寻址的方式。因此，在程序运行时，看不到i和j的内容。
  - 相加后结果（0xFA00）16保存在单片机内部数据区地址为0x08的位置，占用两个字节的存储空间，如图所示。

数据类型
--标准C语言所支持的类型



- （3200）10=（7D00）16，对于两个数（7D00）16相加来说，结果为（FA00）16，从计算的结果来看，（CY）=0，（OV）=1。很明显结果溢出，（FA00）16对应的有符号数为-1536，即：两个正数相加得到一个负数。
- 在调试界面的寄存器界面内，找到并展开psw，可以看到和前面一样的分析结果，如下图所示。

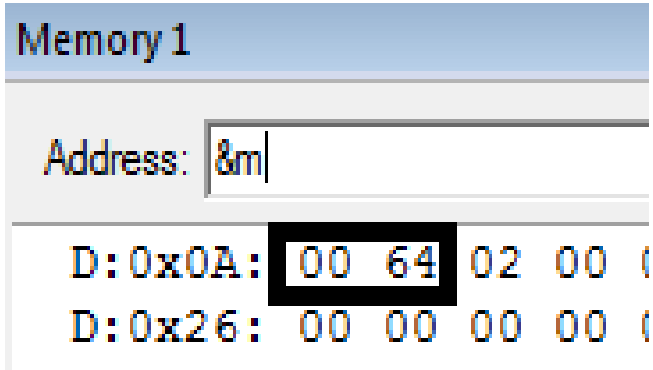
数据类型
--标准C语言所支持的类型

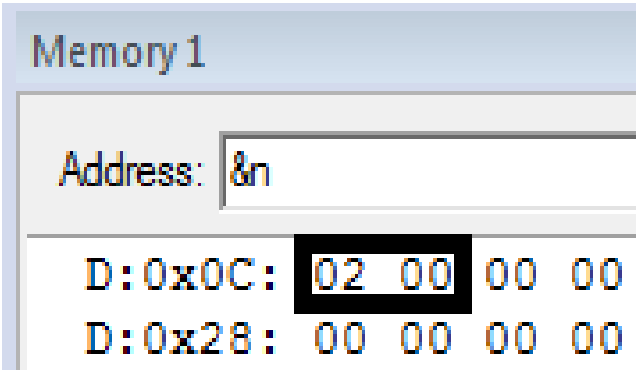


注：因此，提示读者在声明数据类型时，必须考虑所表示数据的范围，不然会由于不同整数类型数据位宽的不同而造成计算结果的溢出。

数据类型
--标准C语言所支持的类型

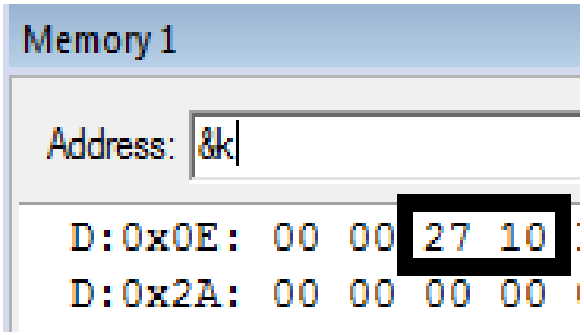
- 在Memory 1窗口界面内，分别重新输入&m和&n，可以看到将变量m的值0x64分配到单片机片内数据区地址为0x0A的位置，该变量占用两个存储字节空间；类似地，可以看到将变量n的值0x200分配到单片机内数据区地址为0x0C的位置，该变量占用两个存储字节空间。

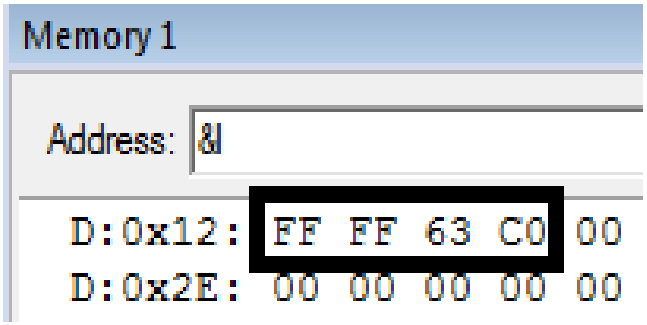




数据类型
--标准C语言所支持的类型

- 在Memory 1窗口界面内，分别重新输入&k和&l，可以看到将变量k的值（10000）10=（2710）16分配到单片机片内数据区地址为0x0E开始的位置，该变量占用四个存储字节空间；类似地，可以看到将变量l的值（-40000）10=（FFFF63C0）16分配到单片机内数据区地址为0x12开始的位置，该变量占用四个存储字节空间。





数据类型
--实数型

在C语言中，提供了两种实数的表示方法，包括：

- 十进制表示法
  - 由数字0~ 9以及小数点组成，例如：0.0、25.0、5.789、0.13、5.0、300.、-267.8230。

注：在使用十进制表示浮点数时，必须包含小数点。

- 指数形式表示法
  - 由十进制数字、阶码标志（小写字母“e”或大写字母“E”），以及阶码（只能为整数，可以带符号）组成。一般形式为：

    a E n

  - 其中：

    a和n均为十进制数；其表示的指数为：

    $a \times 10^n$

数据类型
--实数型

- 在标准的C语言中，将按照所能表示的数的动态范围和精度，将实数进一步的分成单精度实数、双精度实数和长双精度三种，分别用float、double和long double关键字声明这三种类型的实数。它们所分配的存储字长和表示数的范围不同。

注：

（1）对于单片机来说，double和浮点类型相同。

（2）可以在具体的数值后面加后缀字母“f”表示该数为单精度浮点数。

数据类型
--实数型

单精度、双精度和长双精度实数的字长及表示的范围

类型说明符	比特数（字节数）	有效数字	数的范围
float	32(4)	6~7	10^{-37}~10^{38}
double	64(8)	15~16	10^{-307}~10^{308}
long double	128(16)	18~19	10^{-4931}~10^{4932}

数据类型
--实数型

【例】浮点数声明的例子

```
void main()
{
    float i=100.00,j=245.6e30;
}
```

下面对该例子进行分析，分析步骤主要包括：

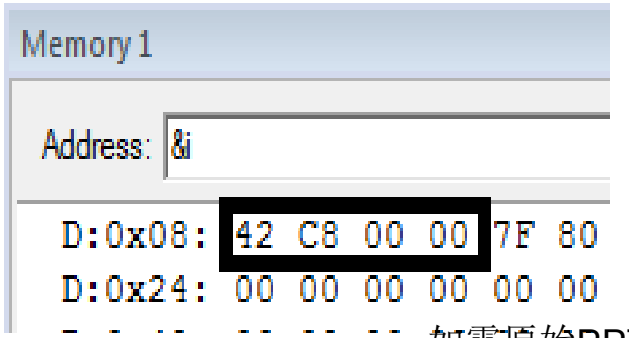
1）进入本书所提供资料的STC_example\例子6-3\目录下，在Keil μVision5集成开发环境下打开该设计。

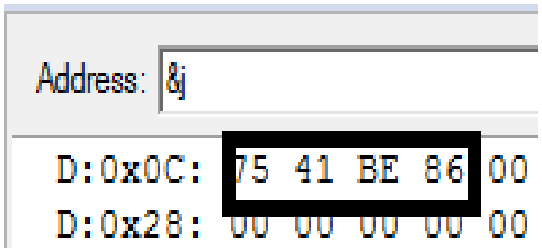
2）在集成开发环境主界面主菜单下，选择Debug->Start/Stop Debug Session。

3）在调试器模式下，单步运行该程序，一直到程序的末尾。

数据类型
--实数型

- 在Memory 1窗口界面内，分别输入&i和&j，可以看到将浮点变量i的值100.0分配到单片机片内数据区地址为0x08开始的位置，该变量占用四个存储字节空间，其值用十六进制数表示为0x42C80000；类似地，可以看到将浮点变量j的值245.6e30分配到单片机内数据区地址为0x0C开始的位置，该变量占用四个存储字节空间，其值用十六进制数表示为0x7541BE86。

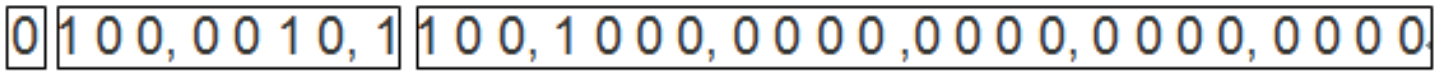




数据类型
--实数型

下面分析一下浮点数在计算机中存储的原理。

- **对于浮点数100.00来说，在计算机中存储的数0x42C80000。对应的二进制数表示为：**



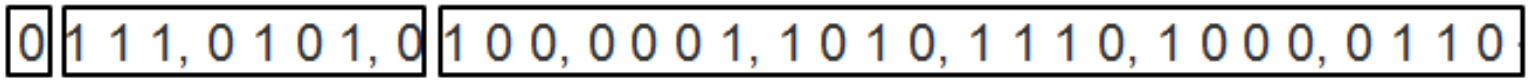
其中：

- **0：表示符号位，表示当前是正数；**
- **100，0010，1:表示阶数，对应的十进制数为133。在浮点标准中，这个值已经加上了偏移量127，所以实际的阶数为133-127=6，对应于2^6=64，即表示的是2的幂次方。**
- **100，1000，0000，0000，0000，0000：表示尾数，对应的十进制小数为0.5625。因为总是隐含1。所以，表示的小数实际值为1.5625。**

因此， $1.5625 \times 2^{133-127} = 1.5625 \times 64 = 100$

数据类型
--实数型

- 对于浮点数245.6e30来说，在计算机中存储的数0x7541BE86。对应的二进制数表示为：



其中：

- 0：表示符号位，表示当前是正数；
- 111，0101，0:表示阶数，对应的十进制数为234。在浮点标准中，这个值已经加上了偏移量127，所以实际的阶数为234-127=107，对应于2^107=1.6226×10^32，即表示的是2的幂次方。
- 100，0001，1010，1110，1000，0110：表示尾数，对应的十进制小数为0.51361083984375。因为总是隐含1，所以表示的小数实际值为1.51361083984375。

所以：

$$1.51361083984375 \times 2^{107} = 2.455974 \times 10^{32}$$

数据类型
--字符型

普通字符

- 在C语言中，普通字符型数据是用单引号括起来的一个字符。
  - 例如：'a'、'b'、'='、'+'、'?'。
- 在C语言中，字符型数据有以下特点：
  - 字符型数据只能用单引号括起来，不能用双引号或其它括号。
  - 字符型数据只能是单个字符，不能是字符串。
  - 字符可以是字符集中任意字符。但数字被定义为字符型之后就不能参与数值运算。如'5'和5是不同的。'5'是字符型数据，不能参与运算。

数据类型
--字符型

对于普通字符型数据来说，用关键字char进行声明。

- **实际上，就是八位的二进制数，或者说一个字节的存储宽度。**
- **更进一步的，还可以在char前面增加signed（有符号）和unsigned（无符号）声明。当：**
  - **signed char（有符号字符型）时，表示数的范围为-128~+127。**
  - **unsigned char（无符号字符型）时，表示数的范围为0~255。**

数据类型
--字符型

【例】字符型数据声明的例子

```
void main()
{
    char a='a',b='H';
    unsigned char c=250;
    signed char d=120;
}
```

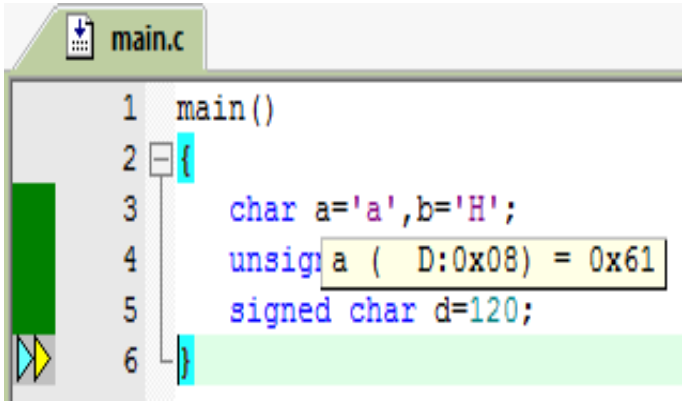
数据类型
--字符型

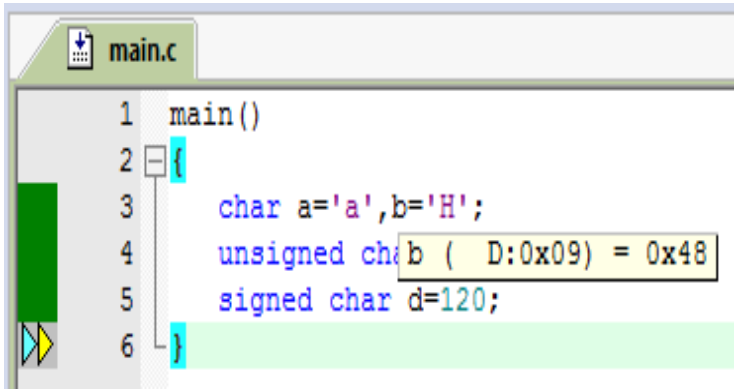
下面对该例子进行分析，分析步骤主要包括：

- 进入本书所提供资料的STC_example\例子6-4\目录下，在Keil μVision5集成开发环境下打开该设计。
- 在集成开发环境主界面主菜单下，选择Debug->Start/Stop Debug Session。
- 在调试器模式下，单步运行该程序，一直到程序的末尾。

数据类型
--字符型

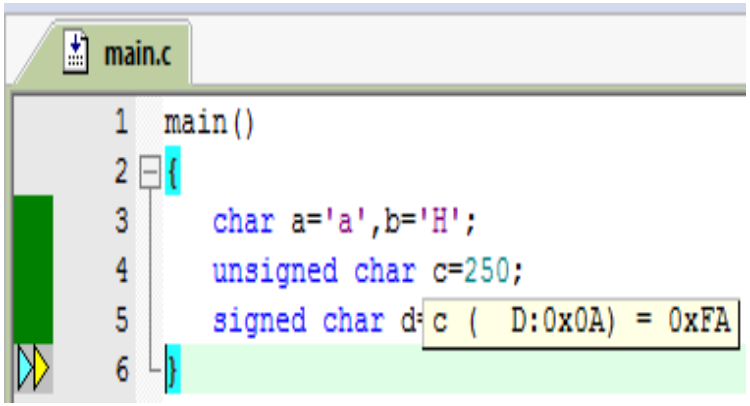
- 在main.c窗口中，将光标分别放在变量a、b、c、d上，就会看到字符变量的存储信息：
  - 显示字符变量a被分配到单片机片内数据区地址为0x08开始的位置，该变量占用一个字节存储空间，其值用十六进制数表示为0x61；
  - 显示字符变量b的值被分配到单片机内数据区地址为0x09开始的位置，该变量占用一个存储字节空间，其值用十六进制数表示为0x48。

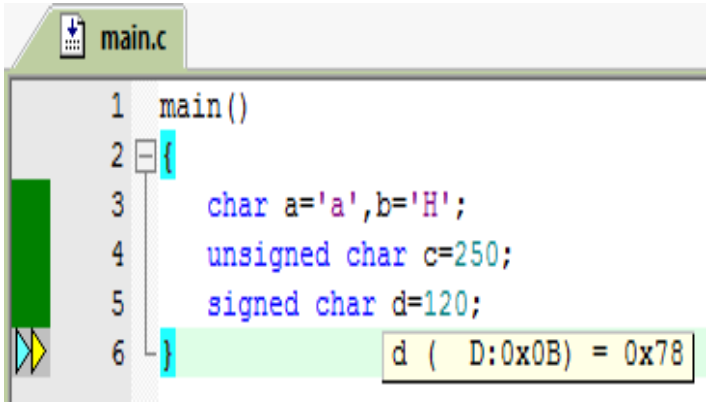




数据类型
--字符型

- 显示字符变量c的值被分配到单片机内数据区地址为0x0A开始的位置，该变量占用一个存储字节空间，其值用十六进制数表示为0xFA。
- 显示字符变量d的值被分配到单片机内数据区地址为0x0B开始的位置，该变量占用一个存储字节空间，其值用十六进制数表示为0x78。





- 退出调试器界面，并关闭该设计。

数据类型
--字符型

转义字符

- 转义字符是一种特殊的字符。
- 转义字符以反斜线"\"开头，后跟一个或几个字符。
- 转义字符具有特定的含义，不同于字符原有的意义，故称“转义”字符。
- 转义字符主要用来表示那些用一般字符不便于表示的控制代码。

数据类型
--字符型

常用转义字符及功能

转义字符	转义字符的意义	ASCII
\n	回车换行	10
\t	横向跳到下一制表位置	9
\b	退格	8
\r	回车	13
\f	走纸换页	12
\\	反斜线符"\"	92
\'	单引号符	39
\”	双引号符	34
\a	鸣铃	7
\ddd	1～3位八进制数所代表的字符	
\xhh	1～2位十六进制数所代表的字符	

数据类型
--字符型

【例】转义字符型数据的例子

```
void main()
{
    char a='n',b='\n';
    char c='t',d='\t';
    char e='\r',f='\\';
}
```

数据类型
--字符型

下面对该例子进行分析，分析步骤主要包括：

- 进入本书所提供资料的STC_example\例子6-5\目录下，在Keil μVision5集成开发环境下打开该设计。
- 在集成开发环境主界面主菜单下，选择Debug->Start/Stop Debug Session。
- 在调试器模式下，单步运行该程序，一直到程序的末尾。

数据类型
--字符型

- 在Disassembly窗口下，可以看到，经过转义后字符的取值明显不同。

```
      3:          char a='n',b='\n';
C:0x0003    75086E   MOV       0x08,#0x6E
C:0x0006    75090A   MOV       0x09,#0x0A
      4:          char c='t',d='\t';
C:0x0009    750A74   MOV       0x0A,#0x74
C:0x000C    750B09   MOV       0x0B,#0x09
      5:          char e='\r',f='\\';
C:0x000F    750C0D   MOV       0x0C,#0x0D
C:0x0012    750D5C   MOV       0x0D,#0x5C
```

数据类型
--字符串

字符串是由一对双引号括起的字符序列。字符串和字符不同，它们之间主要有以下区别：

- 字符由单引号括起来，字符串由双引号括起来。
- 字符只能是单个字符，字符串则可以含一个或多个字符。
- 可以把一个字符型数据赋给一个字符变量，但不能把一个字符串赋给一个字符变量。
- 在C语言中没有相应的字符串变量，将一个变量声明为字符串。但是可以用一个字符数组来存放一个字符串。
- 字符占用一个字节的存储空间，而字符串所占存储空间的字节数等于字符串的字节数加1。增加的一个字节用于存放字符 ‘\0’ (ASCII码为0)，它用于表示字符串结束。

数据类型
-单片机扩充的类型

除了支持标准的C所提供的数据类型外，Keil的编译器还提供了对单片机特定数据类型的支持。

注：在使用扩充数据类型时，必须添加头文件包含语句。

```
#include  <reg51.h>
```

数据类型
-单片机扩充的类型

bit类型

- 该数据类型可用于定义一个比特位
- 但是不能定义位指针，也不能定义位数组。

数据类型
-单片机扩充的类型

sfr类型

- 该数据类型可以用于定义8051单片机中的所有内部8位特殊功能寄存器SFR。
- sfr类型数据占用存储空间一个字节，取值范围为0~255。定义的格式：

sfr 标识符 =地址;

数据类型
-单片机扩充的类型

下面给出头文件reg51.h中已经定义的sfr类型。

```
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
sfr B    = 0xF0;
sfr SP   = 0x81;
sfr DPL  = 0x82;
sfr DPH  = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0  = 0x8A;
sfr TL1  = 0x8B;
sfr TH0  = 0x8C;
sfr TH1  = 0x8D;
sfr IE   = 0xA8;
sfr IP   = 0xB8;
sfr SCON = 0x98;
sfr SBUF = 0x99;
```

注：在C文件中，直接使用已经预定义的sfr类型，而无需重新进行定义。

数据类型
-单片机扩充的类型

sfr16类型

- 该数据类型可以用于定义8051单片机中的16位特殊功能寄存器。sfr16数据类型占用存储空间两个字节，取值范围为0~65535。

sbit类型

- 该数据类型可以用于定义8051单片机内的RAM中可寻址位或者特殊功能寄存器中的可寻址位。定义的格式为：

sbit 标识符 =地址;

数据类型
-单片机扩充的类型

下面给出头文件reg51.h中已经定义的sbit类型。

```
sbit CY   = 0xD7;
sbit AC   = 0xD6;
sbit F0   = 0xD5;
sbit RS1  = 0xD4;
sbit RS0  = 0xD3;
sbit OV   = 0xD2;
sbit P    = 0xD0;
```

```
/*  TCON  */
sbit TF1  = 0x8F;
sbit TR1  = 0x8E;
sbit TF0  = 0x8D;
sbit TR0  = 0x8C;
sbit IE1  = 0x8B;
sbit IT1  = 0x8A;
sbit IE0  = 0x89;
sbit IT0  = 0x88;
```

数据类型
-单片机扩充的类型

```
/*  IE   */
sbit EA   = 0xAF;
sbit ES   = 0xAC;
sbit ET1  = 0xAB;
sbit EX1  = 0xAA;
sbit ET0  = 0xA9;
sbit EX0  = 0xA8;
```

```
/*  IP   */
sbit PS   = 0xBC;
sbit PT1  = 0xBB;
sbit PX1  = 0xBA;
sbit PT0  = 0xB9;
sbit PX0  = 0xB8;
```

数据类型
-单片机扩充的类型

```
/*  P3  */
sbit RD   = 0xB7;
sbit WR   = 0xB6;
sbit T1   = 0xB5;
sbit T0   = 0xB4;
sbit INT1 = 0xB3;
sbit INT0 = 0xB2;
sbit TXD  = 0xB1;
sbit RXD  = 0xB0;
```

```
/*  SCON  */
sbit SM0  = 0x9F;
sbit SM1  = 0x9E;
sbit SM2  = 0x9D;
sbit REN  = 0x9C;
sbit TB8  = 0x9B;
sbit RB8  = 0x9A;
sbit TI   = 0x99;
sbit RI   = 0x98;
```

注：在C文件中，直接使用已经预定义的sfr类型，而无需重新进行定义。

数据类型
-单片机扩充的类型

此外，还可以通过下面的形式定义sbit：

sbit SFR位的名字=SFR的名字^i;

其中:

i=0,….,7

注:在本书后续章节中，会使用相关的定义。

数据类型
-单片机扩充的类型

定义和使用单片机扩展数据类型的步骤主要包括：

- **打开Keil μVision5集成开发环境。**
- **建立一个新的设计工程。**
- **按照前面添加新文件的方法，选择.h文件模板，将该文件命名为self_define.h。**
- **在该文件中，添加设计代码，如下图所示。**

数据类型
-单片机扩充的类型

self_define.h | reg51.h | main.c

```
1  #ifndef _STC15Fxxxx_H
2  #define _STC15Fxxxx_H
3
4
5  /*   BYTE Registers   */
6  sfr P4    = 0xC0;
7  /*   WORD Registers   */
8  sfr16 DP  = 0x82;
9
```

- 保存self_define.h文件。
- 按照前面添加新文件的方法，选择.c文件模板，将该文件命名为main.c。
- 在该文件中，添加设计代码，如下图所示。

数据类型
-单片机扩充的类型

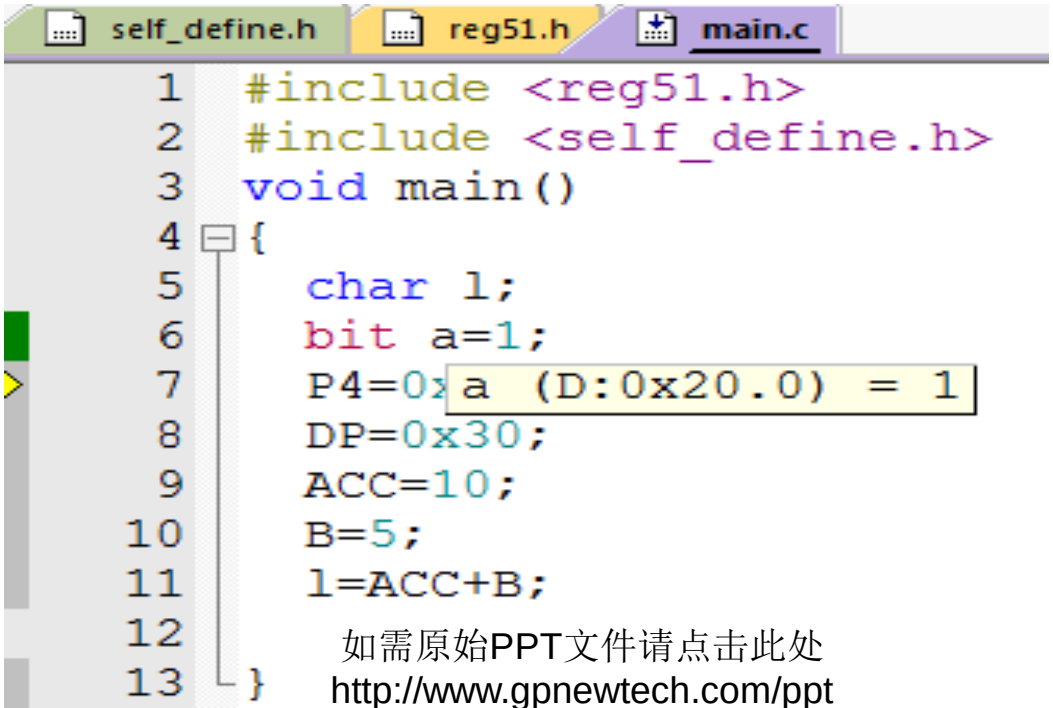
self_define.h | reg51.h | main.c

```
1  #include <reg51.h>
2  #include <self_define.h>
3  void main()
4  {
5     char l;
6     bit a=1;
7     P4=0xFF;
8     DP=0x30;
9     ACC=10;
10    B=5;
11    l=ACC+B;
12
13 }
```

- 保存main.c文件。
- 在集成开发环境主界面主菜单下，选择Debug->Start/Stop Debug Session。

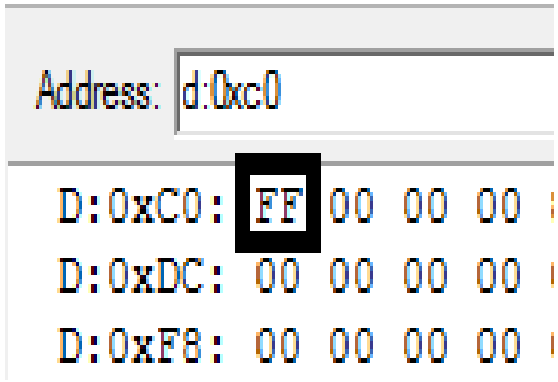
数据类型
-单片机扩充的类型

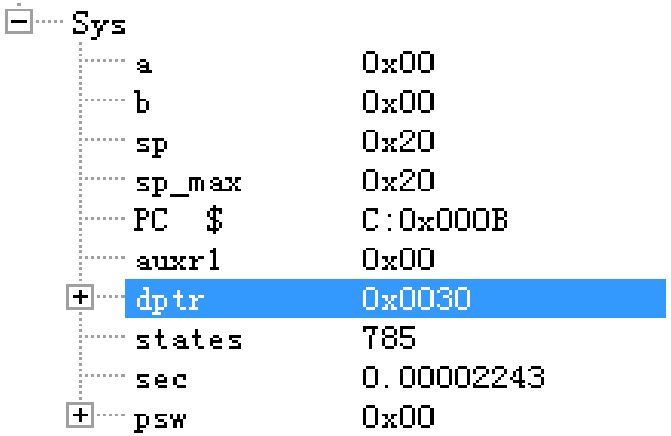
- 在调试器模式下，单步运行该程序，一直到第7行，如下图所示。将光标移到a上，弹出提示框。该提示信息说明比特类型变量a被分配到内部数据可位寻址区0x20的第0位（D：0x20.0）的位置，值为1。



数据类型
-单片机扩充的类型

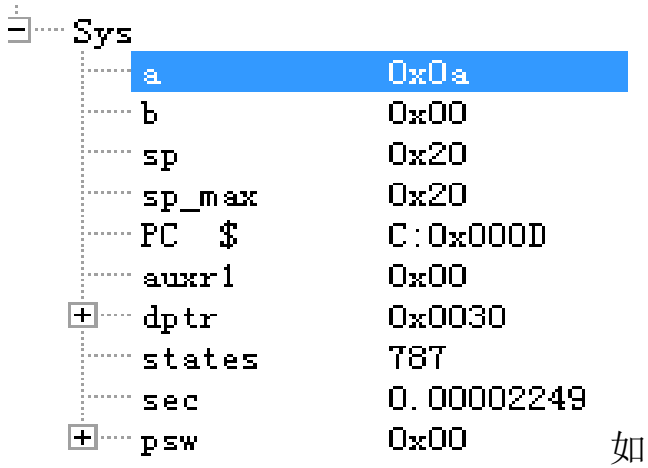
- 继续单步运行该程序，一直到第8行，该行指令用于给端口P4赋值为0xFF。在Memory 1窗口下，输入d:0xc0，如左图所示。
- 在调试器模式下，单步运行该程序，一直到第9行，该行程序给16位的数据指针DPTR赋值为0x30。在Register窗口内，可以看到数据指针寄存器的内容变为0x0030，如右图所示。

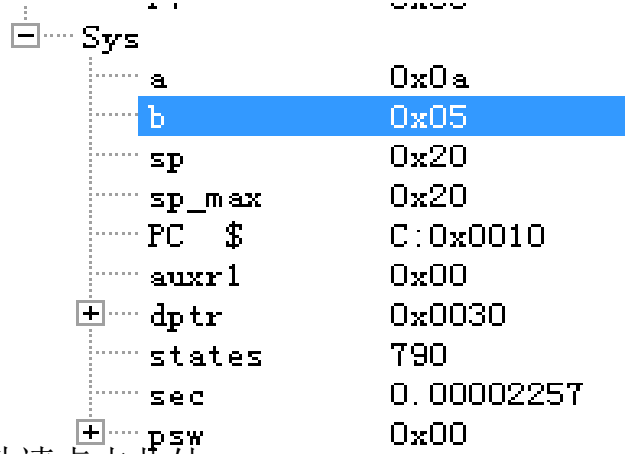




数据类型
-单片机扩充的类型

- 在调试器模式下，单步运行该程序，一直到第10行，该行程序给累加器ACC赋值为0x0A（十进制数10）。在Register窗口内，可以看到累加器ACC（显示为a）的内容变为0x0a。
- 在调试器模式下，单步运行该程序，一直到第11行，该行程序给寄存器B赋值为0x05（十进制数5）。在Register窗口内，可以看到寄存器B（显示为b）的内容变为0x05，如右图所示。





数据类型
-单片机扩充的类型

- 在调试器模式下，单步运行该程序，一直到结束，该行程序将累加器ACC的内容和寄存器B的内容相加后送给字符型变量l。将光标移到l上，弹出提示框。该提示信息说明字符类型变量l被分配到内部数据区地址为0x08的位置（D：0x08）的位置，值为0x0F。

self_define.h | reg51.h | main.c

```
1  #include <reg51.h>
2  #include <self_define.h>
3  void main()
4  {
5     char l;
6     bit a=1;
7     P4=0xFF;
8     DP=0x30;
9     ACC=10;
10    B=5;
11    l=ACC+B;
12    l (  D:0x08) = 0x0F
13 }
```

- 退出调试器模式，并关闭该设计。

数据类型
--自定义数据类型

自定义数据类型

- 在C语言中，除了可以使用上面所给出的数据类型外，设计者还可以根据自己的需要对数据类型进行重新定义。重新定义数据类型时需要用到关键字typedef，格式如下：

typedef 已有的数据类型 新的数据类型名

数据类型
--自定义数据类型

【例】对上面用浮点数声明的例子使用typedef重新定义数据类型并使用新定义的数据类型。

修改用浮点数声明的例子的步骤主要包括：

- 在STC_example目录下，新建一个名字为例子6-7的子目录。将STC_example\例子6-2目录下的所有文件复制到新建的子目录下。
- 按照前面的方法添加一个名字为type_define.h的头文件。

数据类型
--自定义数据类型

- 在该文件中添加代码，将unsigned int数据类型重新定义为u16，以及将long int数据类型重新定义为l32，如图所示。

type_define.h | main.c

```
1   typedef unsigned int  u16;
2   typedef long int  l32;
```

- 保存type_define.h头文件。

数据类型
--自定义数据类型

- 修改main.c文件。在该文件中，新添加包含头文件声明，以及使用新定义的头文件，如图所示。

type_define.h | main.c

```
1  #include "type_define.h"
2  void main()
3  {
4     int i=32000,j=32000,h;
5     u16 m=100,n=0x200;
6     l32 k=10000,l=-40000;
7      h=i+j;
8  }
```

- 保存main.c文件。
- 验证新定义的数据类型和原来的数据类型等价。
- 关闭并退出设计。

数据类型
--变量及存储模式

变量的值可以在程序执行过程中不断变化。

- **在使用变量之前，需要对变量进行定义，定义的内容包括：变量标识符、数据类型和存储模式。**
- **在标准C语言中，编译器会根据数据类型和硬件系统自动的确定存储模式。**

数据类型
--变量及存储模式

- 为了更好的利用所提供的存储空间，在单片机中提供了增强功能的变量存储模式定义功能，定义格式为：

[存储种类] 数据类型 [存储器类型] 变量名列表;

其中：

- 存储种类和存储器类型是可选项。
- 变量的存储种类有四种，包括：auto（自动）、extern（外部）、static（静态）和register（寄存器）。在没有明确说明变量的存储种类时，默认auto。

- Keil提供对8051系列单片机的硬件结构，以及不同存储器结构的支持。因此，可以在定义变量的时候，为每个定义的变量准确的指定其存储类型。这样，就可以准确定位变量所在的存储空间.

数据类型
--变量及存储模式

Keil所支持的单片机存储类型

存储器类型	说明
DATA	直接寻址片内数据存储器（128字节），访问速度最快
BDATA	可位寻址的片内数据存储器（16字节），允许位和字节混合访问
IDATA	间接访问的片内数据存储器（256字节），允许访问全部片内地址
PDATA	分页寻址的片外数据存储器（256字节），用MOV @Ri指令进行访问
XDATA	外部数据存储器（64KB），用MOVX @DPTR指令进行访问
CODE	程序存储器（64KB），用MOVC @A+DPTR指令进行访问

注：如果在定义变量时，没有指定存储器类型，则按编译时候使用的存储器模SMALL、COMPACT或者LARGE来确定默认的存储器类型。

数据类型
--变量及存储模式

在Keil C中，可以通过使用_at_定位变量的绝对地址。格式为：

[存储器类型] 数据类型 标识符 _at_ 地址常数

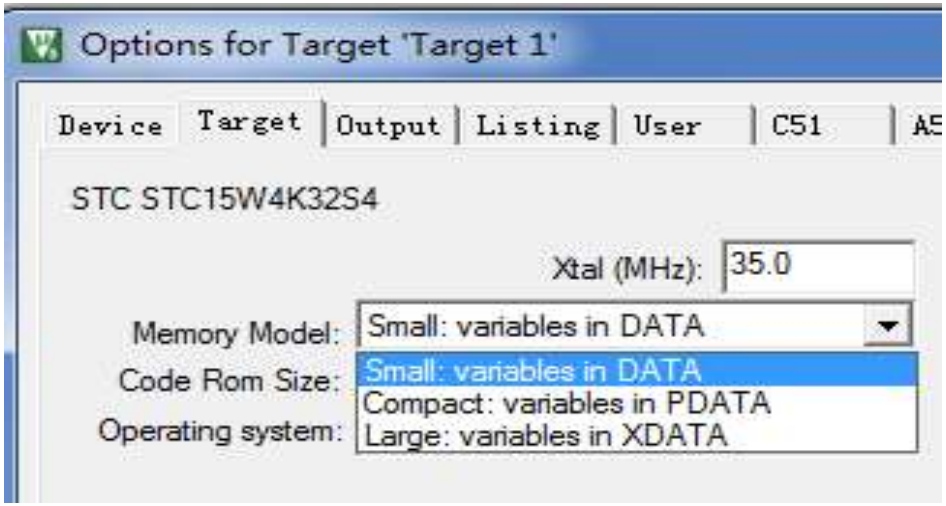
比如：

```
xdata int i1 _at_ 0x8000;
```

并且，在XDATA空间定义全局变量的绝对地址时，可以在变量前加一个关键字volatile，这样对该变量的访问就不会被Cx51编译器给优化掉。

数据类型
--变量及存储模式

- 在设置Target 1的对话框窗口界面中的Target标签窗口下，通过Memory Model右侧的下拉框可以选择存储器模式，如图所示。



注：从访问效率来看，SMALL存储器模式效率最高，而LARGE访问效率最低。

数据类型
--变量及存储模式

【例】带有存储器类型的变量定义例子

```
void main()
{
    xdata long int x=-1000,y=4000;
    xdata char m=90,n=70;
     pdata char k=10;
     bdata char l=0;
}
```

数据类型
--变量及存储模式

下面对该例子进行分析，分析步骤主要包括：

- 进入本书所提供资料的STC_example\例子6-8\目录下，在Keil μVision5集成开发环境下打开该设计。
- 在集成开发环境主界面主菜单下，选择Debug->Start/Stop Debug Session。
- 在调试器模式下，单步运行该程序，一直到程序的末尾。
- 在Memory 1窗口下，分别输入&x、&y、&m、&n、&k、&l，查看变量所在的存储器空间，以及所占用的字节数。
- 退出调试器模式，并关闭该设计。

运算符
--赋值运算符

在C语言中，赋值操作使用“=”号实现

- “=”称为赋值运算符，赋值语句的格式为：

  变量=表达式;

- 先计算由表达式所得到的值，然后将该值分配给等号左边的变量。

注：在进行赋值操作的时候，一定要注意变量和表达式值的数据类型。

运算符
--赋值运算符

【例】赋值操作的例子

```
void main()
{
    int a;
    char b;
    float c;
    a=10000;
    b=200;
    c=0.5;
}
```

注：读者可进入本书所提供资料的STC_example\例子6-9\目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，使用单步运行。

运算符
--算术运算符

在C语言中，所提供的算术运算符包括：

- \+ 加法运算或者取正数运算
- \- 减法运算或者取负数运算
- * 乘法运算
- / 除法运算
- % 取余运算

运算符
--算术运算符

- 在这些算数运算中，取正和取负运算是单目运算外，其它都是双目运算。
- 在求取表达式的值时，按照运算符的优先级进行。
  - 单目运算的优先级要高于双目运算。
  - 在双目运算中，优先级按照**、*、/、%、+、-从高到低排列。
  - 可以通过使用（ ）修改运算的优先级顺序。

运算符
--算术运算符

【例】算术运算操作的例子

```
void main()
{
    int a=1000,b=33,c,d,h,i;
    long int e,j;
    float f,g;
    c=a/b;
    d=a%b;
    e=a*b;
    f=(float)a/b;
    g=a+b-c;
    h=(a+b)*c;
    i=b-a*c;
    j=(long)a*b;
 }
```

运算符
--算术运算符

注：

（1）读者可以进入到本书所提供资料的STC_example\例子6-10\目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，使用单步运行。

（2）打开Watch 1窗口观察运算得到的结果。

运算符
--递增和递减运算符

在C语言中，还提供了递增运算符++和递减运算符--，它们的作用是为了对运算的数据进行加一和减一的操作。典型的,++i,i++,--i,i--。

- ++i和i++是不一样的。++i是先执行i+1操作，然后再使用i的值；而i++是先使用i的值，然后再执行i+1的操作。
- 类似的，--i和i--是不一样的。--i是先执行i-1操作，然后再使用i的值；而i--是先使用i的值，然后再执行i-1的操作。

运算符
--递增和递减运算符

【例】递增和递减运算符的例子

```
void main()
{
    int a=40,b,c,d,e,f,g;
    b=a--;
    c=a;
    d=--a;
    e=a++;
    f=a;
    g=++a;
}
```

注：

（1）读者可以进入到本书所提供资料的STC_example\例子6-11\目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，使用单步运行。

（2）打开Watch 1窗口观察运算得到的结果。

运算符
--关系运算符

在C语言中，提供了下面的关系运算符：

- > 大于
- < 小于
- >= 大于等于
- <= 小于等于
- == 等于
- != 不等于

注：（1）~（4）关系运算符的优先级相同，（5）和（6）关系运算符优先级相同。前一组关系运算符的优先级高于后一组的关系运算符。

运算符
--关系运算符

- 由这些运算符构成的关系表达式用于后面所介绍的条件判断语句中，用于确定判断的条件是否成立。关系表达式的格式为：

  表达式1 关系运算符 表达式2

- 关系运算的结果只有1和0两个值，即：当满足比较的条件时，关系运算的结果为1；否则为0。

运算符
--关系运算符

【例】关系运算符的例子

```
void main()
{
    int a=40,b=10;
    bit c,d,e,f,g,h;
    c=a<b;
    d=a<=b;
    e=a>b;
    f=a>=b;
    g=a!=b;
    h=a==b;
}
```

注:（1）读者可以进入到本书所提供资料的STC_example\例子6-12目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，使用单步运行。

（2）打开Watch 1窗口观察运算得到的结果。

运算符
--逻辑运算符

在C语言中，提供了逻辑运算符，包括：

- && 逻辑与
- || 逻辑或
- ! 逻辑非
- 逻辑运算符用在多个关系表达式的条件判断语句中，格式：
  - 条件表达式1 && 条件表达式2 && ……

用于确定这些条件表达式条件是否同时成立；如果都成立则返回1；否则返回0；

运算符
--逻辑运算符

□ 条件表达式1 || 条件表达式2|| …..

用于确定这些条件表达式中是否存在有条件表达式成立的情况。如果有一个条件表达式条件成立则返回1，否者返回0；

□ ！条件表达式

用于对条件表达式的返回值进行取反操作。如果条件表达式的值为1，则!操作将对条件表达式的值取反，变成0；如果条件表达式的值为0，否则!操作将对条件表达式的值取反，变成1。

■ 逻辑关系符的优先级按照：!、 &&、 ||依次降低。

运算符
--逻辑运算符

【例】逻辑运算符的例子

```
void main()
{
    int a=40,b=10;
    bit c,d,e,f,g,h,i;
    c=a<b && a>b;
    d=a<=b || a>b;
    e=!a>b;
    f=a!=b && a==b;
    g=a!=b || a==b;
    h=!(a==b);
    i=!a==b;
}
```

注：读者可以进入到本书所提供资料的STC_example\例子6-13目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，使用单步运行。打开Watch 1窗口观察运算得到的结果。

运算符
--位运算符

在C语言中，提供了对数据进行按位运算的位运算符，包括：

- ~ 按位取反
- << 左移
- >> 右移
- & 按位与
- | 按位或
- ^ 按位异或

运算符
--位运算符

- 对于按位取反、按位与、按位或和按位异或运算来说，格式为：

变量1 位运算符 变量2；

这些运算规则遵守逻辑代数运算规律，如下表所示。

逻辑变量x	逻辑变量y	~x	~y	x&y	x | y	x^y
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	1	0

注：按位逻辑运算的变量数据类型不能是浮点数。

运算符
--位运算符

- 对于左移和右移运算来说，格式为：

  变量 移位运算符 移位个数

  - 对于左移操作来说，在最右端（最低位）补0。
  - 对于右移操作来说，如果是无符号的数，则总是在最左端（最高位）补0；如果是有符号的数，如果符号位为1，则在最左端（最高位）补1；否则，在最左端（最高位）补0。

运算符
--位运算符

【例】位运算符的例子

```
void main()
{
char a=30, b=55;
char c,d,e,f,g;
int i=-50,j=60;
int k,h,l,m;
c=~a;
d=a & b;
e=a | b;
f=a ^ b;
g=~(a ^ b);
h=~(a & b);
k=i<<3;
h=i>>3;
l=j<<4;
m=j>>4;
}
```

注：读者可以进入到本书所提供资料的STC_example\例子6-14目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，使用单步运行。打开Watch 1窗口观察运算得到的结果。

运算符
--复合赋值运算符

在C语言中，提供了复合赋值运算符。复合赋值运算符是算术运算符、位运算符以及赋值运算符的组合。复合赋值运算符包括：

- +=　　加法赋值
- -=　　减法赋值
- *=　　乘法赋值
- /=　　除法赋值
- %=　　取模赋值
- <<=　　左移赋值

运算符
--复合赋值运算符

- >>=　　右移赋值
- &=　　逻辑与赋值
- |=　　逻辑或赋值
- ^=　　逻辑异或赋值
- ~=　　逻辑非赋值

复合赋值运算的格式如下：

变量　复合赋值运算符　表达式

注：在符合赋值表达式中，先进行表达式的运算操作，然后再执行赋值操作的过程。

运算符
--复合赋值运算符

【例】复合赋值运算符的例子

```
void main()
{
    int a=100,b=45;
    int c=900,d=140;
    int e=790,f=9900;
    int g=-90,h=890;
    int i=560,j=711;
    a+=b;
    c-=b;
    d*=b;
```

运算符
--复合赋值运算符

```
    c-=b;
    d*=b;
    e/=b;
    f%=b;
    g<<=2;
    h>>=3;
    i&=b;
    j|=b;
}
```

注：读者可以进入到本书所提供资料的STC_example\例子6-15目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，使用单步运行。打开Watch 1窗口观察运算得到的结果。

运算符
--逗号运算符

在C语言中，提供了“，”运算符，将两个表达式连接在一起，称为逗号表达式。

- 逗号表达式的格式为：

  表达式1,表达式2,表达式3,….,表达式n

- 程序运行时，对于表达式的处理是从左到右依次计算出各个表达式的值，而整个逗号表达式的值是最右边的表达式（即表达式n）的值。

运算符
--逗号运算符

【例】逗号运算符的例子

```
void main()
{
    int a,b,c,d,e,f,g;
    d=(a=10,b=100,c=1000);
    e=(a=100)*(b=30)+(c=750)-1000;
    f=(a=10)*(b=a*30)+(c=a+b);
}
```

注：读者可以进入到本书所提供资料的STC_example\例子6-16目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，使用单步运行。打开Watch 1窗口观察运算得到的结果。

运算符
--条件运算符

在C语言中，提供了条件运算符“？：”，该运算符是C语言中唯一的三目运算符，即该运算符要求有三个运算的对象，用于将三个表达式连接在一起构成一个表达式。

- 条件表达式的格式如下：

逻辑表达式 ？ 表达式1： 表达式2

首先计算逻辑表达式的值，当逻辑表达式的值为1时，将表达式1的值作为整个条件表达式的值；当逻辑表达式的值为0时，将表达式2的值作为整个条件表达式的值。

运算符
--条件运算符

【例】条件运算符的例子

```
void main()
{
    int a=10,b=20,d,e;
    d=(a==b) ? a:b;
    e=(a!=b) ? a:b;
}
```

注：读者可以进入到本书所提供资料的STC_example\例子6-17目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，使用单步运行。打开Watch 1窗口观察运算得到的结果。

运算符
--强制类型转换符

在C语言中，提供了强制类型转换符，用于将一个数据类型转换成另一个数据类型，格式为：

（类型关键字）

运算符
--强制类型转换符

【例】类型转换符的例子

```
void main()
{
    int a=1000,b=2000;
    long int c,e;
    float d;
    c=(long)a;
    d=(float)b;
    e=d/100;
}
```

注：读者可以进入到本书所提供资料的STC_example\例子6-18目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，使用单步运行。打开Watch 1窗口观察运算得到的结果。

运算符
--sizeof运算符

在C语言中，提供了求取数据类型、变量以及表达式字节个数的sizeof运算符。

- 其格式为：

sizeof（表达式）　或　sizeof（数据类型）

注：sizeof是一种特殊的运算符，不是函数。在编译程序的时候，就通过sizeof计算出字节数。

运算符
--sizeof运算符

【例】sizeof运算符的例子

```
void main()
{
    int a=10;
    float b=10.0;
    int c,d,e,f,g,h;
    c=sizeof(int);
    d=sizeof(char);
    e=sizeof(unsigned int);
    f=sizeof(long int);
    g=sizeof(float);
    h=sizeof(a+b);
}
```

描述语句
--输入输出语句

在C语言中，输入和输出操作是通过函数实现的，而不是通过C语言本身实现。

- **典型的，在C语言中，提供了scanf和printf函数用于实现输入和输出的人机交互。**

注：

（1）虽然在本节中声明和使用了这两个函数，但是在单片机中不建议使用，因为这两个函数所占用的存储资源比较多，建议设计者采用其他方式实现这个功能。

（2）在C语言中，调用输入和输出函数时，必须包含头文件**stdio.h**。并且在单片机初始化串口时，必须包含头文件**reg51.h**。

描述语句
--输入输出语句

putchar函数

- 当用在PC系统时，该函数向显示器输出一个字符；而当用在单片机系统时，该函数向串口终端输出一个字符。格式为：

putchar(字母)

描述语句
--输入输出语句

【例】putchar的例子

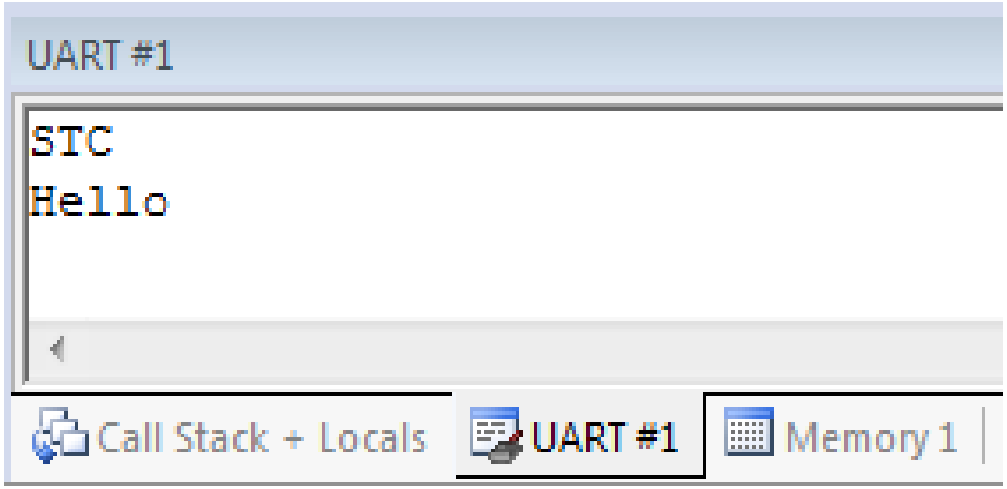
```
void main()
{      char a='S',b='T',c='C';
       char d='\n';
       char e='H',f='e',g='l',h='l',i='o';
       SCON= 0x52;      //初始化串口相关
       TMOD = 0x20;     //初始化串口相关
       TCON = 0x69;     //初始化串口相关
       TH1 = 0xF3;      // 初始化串口相关
putchar(a);
```

描述语句
--输入输出语句

```
putchar(b);
putchar(c);
putchar(d);
putchar(e);
putchar(f);
putchar(g);
putchar(h);
putchar(i);
putchar(d);
}
```

描述语句
--输入输出语句

注：进入到本书所提供资料的STC_example\例子6-20目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5按键运行程序。打开UART #1窗口观察运算得到的结果，如下图所示。



描述语句
--输入输出语句

getchar函数

- 当用在PC系统时，该函数从输入设备（通常是键盘）得到一个字符；而当用在单片机系统时，该函数从输入设备（通常是串口终端）得到一个字符。格式为：

  getchar()

描述语句
--输入输出语句

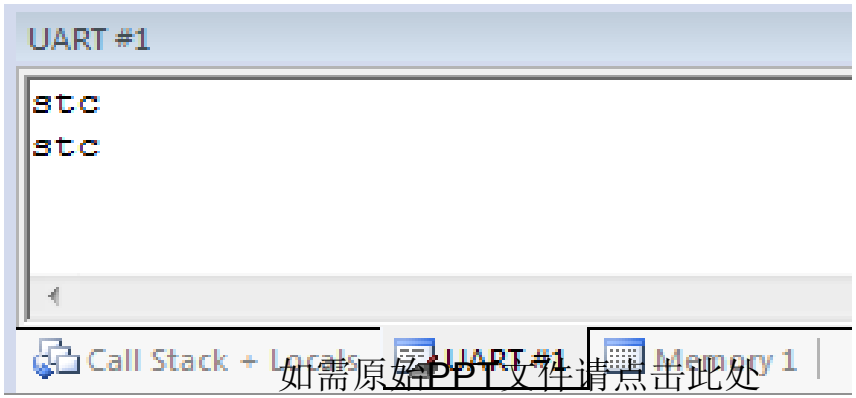
【例】getchar的例子

```
void main()
{
    char a,b,c;
    SCON= 0x52;
    TMOD = 0x20;
    TCON = 0x69;
    TH1 = 0xF3;
     a=getchar();
     b=getchar();
     c=getchar();
     putchar('\n');
     putchar(a);
```

描述语句
--输入输出语句

```
    putchar(b);
    putchar(c);
    putchar('\n');
}
```

注：进入到本书所提供资料的STC_example\例子6-21目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5按键运行程序。打开UART #1窗口，在窗口中输入三个字符，然后输出刚才所输入的三个字符，如下图所示。



描述语句
--输入输出语句

printf函数

- 在PC系统上，该函数向显示器终端输出指定个数任意类型的数据；而在单片机系统中，该函数向串口终端输出指定个数任意类型的数据。格式为：

printf(格式控制,输出列表)

比如：

printf(“%d,%c\n”,i,c);

描述语句
--输入输出语句

格式控制是双撇号括起来的一个字符串，称为“转换控制字符串”，简称格式字符串，包含：

- 格式声明
  - 由“%”和 格式字符组成，如%d、%f等。它的作用是将输出的数据转换为指定的格式输出。格式声明总是由“%”字符开始。

描述语句
--输入输出语句

- %d（%i），按照十进制整型数据输出。
- %o，按照八进制整型数据输出。
- %x，按照十六进制整型数据输出。
- %u，按照无符号十进制数据输出。
- %c，输出一个字符。
- %s，输出一个字符串。
- %f，以系统默认的格式输出实数。此外，%m.nf，用于控制输出m位整数和n位小数的浮点数。
- %e，以指数形式输出实数。

注：可以在格式字符d、o、x和u前面添加l字符，用于表示长整型数。

描述语句
--输入输出语句

- 普通字符
  - 是需要原样输出的字符。比如上面printf函数中双撇号内的逗号、空格和换行符。

输出表列

- 是需要输出的一些数据，可以是常量、变量或者是表达式。

注：由于printf语句消耗单片机资源很多，所以不建议大量使用。

描述语句
--输入输出语句

【例】printf的例子

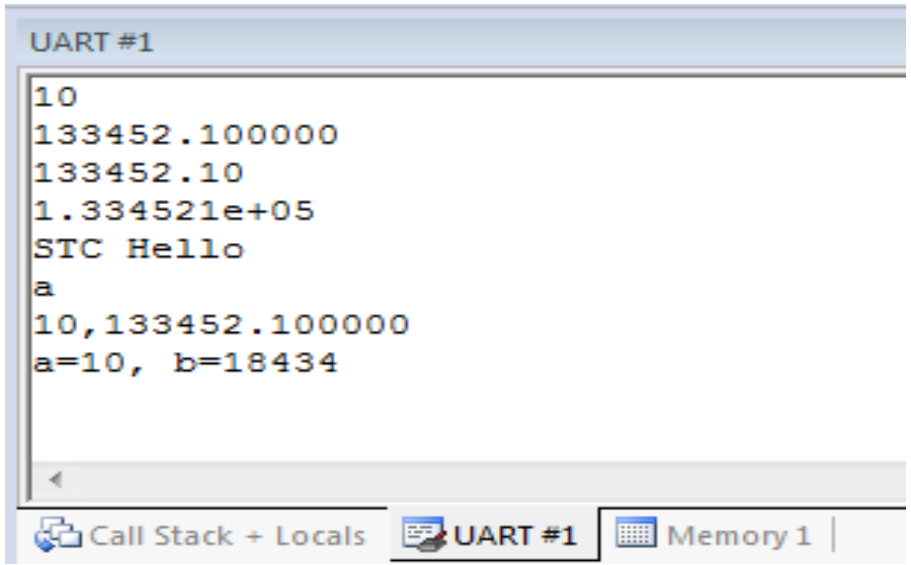
```
void main()
{
    int a=10;
    float b=133452.1243;
    char s1[20]={"STC Hello"};
    char c1='a';
    SCON= 0x52;
    TMOD = 0x20;
    TCON = 0x69;
    TH1 = 0xF3;
```

描述语句
--输入输出语句

```
    printf("%d\n",a);
    printf("%f\n",b);
    printf("%7.2f\n",b);
    printf("%e\n",b);
    printf("%s\n",s1);
    printf("%c\n",c1);
    printf("%d,%f\n",a,b);
    printf("a=%d, b=%d\n",a,b);
}
```

注：进入到本书所提供资料的STC_example\例子6-22目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5按键运行程序。打开UART #1窗口,可以看到显示的信息，如下图所示。

描述语句
--输入输出语句


UART #1
10
133452.100000
133452.10
1.334521e+05
STC Hello
a
10,133452.100000
a=10, b=18434
Call Stack + Locals
UART #1
Memory 1

描述语句
--输入输出语句

scanf函数

在PC系统上，该函数通过输入设备（例如：键盘），获取指定个数的任意类型数据；而在单片机系统中，该函数通过串口终端获取指定个数任意类型的数据。格式为：

scantf(格式控制,地址列表)

比如：

scanf(“%d,%d”,&a,&b);

描述语句
--输入输出语句

格式控制是双撇号括起来的一个字符串，称为“转换控制字符串”，简称格式字符串，包含：

- 格式声明
  - 由“%”和 格式字符组成，如%d、%f等。它的作用是将输入的信息转换为指定的格式的数字。格式声明总是由“%”字符开始。

描述语句
--输入输出语句

- %d（%i），按照十进制整型数据输入。
- %o，按照八进制整型数据输入。
- %x，按照十六进制整型数据输入。
- %u，按照无符号十进制数据输入。
- %c，输入一个字符。
- %s，输入一个字符串。
- %f，用来以系统默认的格式输入实数。
- %e，以指数形式输入实数。

- **普通字符**
  - 需要原样输入的字符。如上面scanf函数中双撇号内的逗号、空格和换行符。

地址表列

- **由若干地址组成的列表，可以是变量的地址或者字符串的首地址。**

描述语句
--输入输出语句

【例】scanf的例子

```
void main()
{
    int i,j,l;
    float k;
    char c1;
    SCON= 0x52;
    TMOD = 0x20;
    TCON = 0x69;
    TH1 = 0xF3;
    scanf("%d,%d,%c",&i,&j,&c1);
    scanf("%f",&k);
    getchar();
```

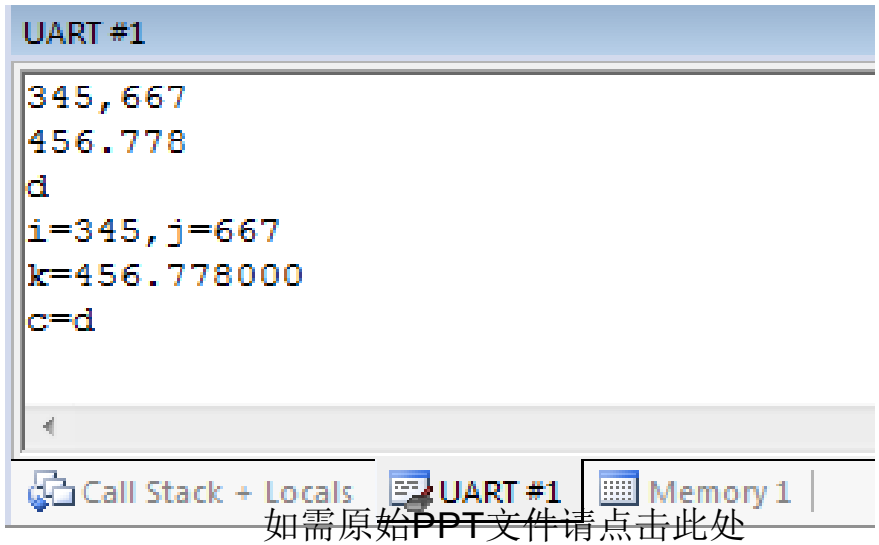
描述语句
--输入输出语句

```
    scanf("%c",&c1);
    printf("\ni=%d,j=%d\n",i,j);
    printf("k=%f\n",k);
    printf("c=%c\n",c1);
    return 1;
}
```

进入到本书所提供资料的STC_example\例子6-23目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5按键运行程序。打开UART #1窗口,按下面格式输入数据：

描述语句
--输入输出语句

- 先输入两个整数，两个整数之间必须用逗号分隔；
- 然后按回车键；
- 输入一个浮点数；
- 然后按回车键；
- 输入一个字符；
- 然后按回车键。
- 可以看到在UART#1窗口内显示了刚才输入数据的信息。



描述语句
--表达式语句

在C语言中，表达式语句是最基本的语句。

- 在C语言中，不同的表达式语句之间用“，”号分割。
- 此外，在表达式语句前面可以存在标号，标号后必须有“:”符号，用于标识每一行代码。格式为：

  标号: 表达式;

  - 这种表示方法可以用在goto等跳转语句中。
- 多条表达式语句，可以通过{}符号构成复合语句，{}符号可以理解成作用边界的分割符，用于确认条件作用的范围。
  - 典型的，main函数通过{}符号将大量的表达式关联到一个函数中，当然{}符号也用于标识main函数的作用范围。

描述语句
--条件语句

在C语言中，提供了条件判断语句，又称为分支语句。通过if关键字来标识条件语句。三种可能的条件语句包括：

- 格式1

```
if(条件表达式)
语句;
```

如果条件表达式成立，则执行语句；否则不执行该语句。

- 格式2

```
if(条件表达式)
    语句1;
else
    语句2;
```

如果条件表达式成立，则执行语句1；否则，执行语句2。

描述语句
--条件语句

- **格式3**

```
if(条件表达式1)
    语句1;
else if(条件表达式2)
    语句2;
else if(条件表达式3)
    语句3;
    .....
else
    语句N;
```

注：根据判断条件的复杂度，条件语句可以嵌套。

描述语句
--条件语句

【例】if-else语句的例子

- 在该例子中，根据三个输入变量a、b、c的值，判断是不是直角三角形。其判断逻辑是，先判断输入的三个变量a、b和c的值是否构成一个三角形；然后，才能判断这个三角形是不是直角三角形。
  - 根据数学知识，构成三角形的条件是同时满足：

    a+b>c,a+c>b且b+c>a

的条件。

  - 根据数学知识，构成直角三角形的条件是满足勾股定理。

描述语句
--条件语句

```
void main()
{    int a, b,c;
     SCON= 0x52;
     TMOD = 0x20;
     TCON = 0x69;
     TH1 = 0xF3;
     scanf("%d,%d,%d",&a,&b,&c);
     if(a+b>c && b+c>a && a+c>b)
        if((a*a+b*b)==c*c)
            printf("a=%d,b=%d,c=%d is a right angle triangle\n",a,b,c);
        else  printf("a=%d,b=%d,c=%d is not a right angle triangle\n",a,b,c);
     else   printf("a=%d,b=%d,c=%d is not a triangle\n",a,b,c);
 }
```

注：读者可以进入到本书所提供资料的STC_example\例子6-24目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5按键运行程序，并打开UART #1窗口。

描述语句
--开关语句

在C语言中，提供了开关语句switch，该语句也是判断语句的一种，用来实现不同的条件分支。

- 与条件语句相比，开关语句更简洁，程序结构更加清晰，使用便捷。开关语句的格式为：

```
switch(表达式)
{
    case 常数表达式1: 语句1;  break;
    case 常数表达式2: 语句2;  break;
          .....
    case 常数表达式n: 语句N;
}
```

注：（1）当一个常数表达式对应于多个语句时，用{}括起来。

（2）break为跳出开关语句，也可以用于跳出循环语句。

描述语句
--开关语句

【例】switch语句的例子

该例子输入月份的值1~12之间的一个数，然后打印出所对应的月份的英文单词。

```
void main()
{     int a;
      SCON= 0x52;
      TMOD = 0x20;
      TCON = 0x69;
      TH1 = 0xF3;
      scanf("%d",&a);
      switch(a)
      {      case 1:    puts("January\n"); break;
             case 2:    puts("February\n");break;
```

描述语句
--开关语句

```
        case 3:    puts("March\n");break;
        case 4:    puts("April\n");break;
        case 5:    puts("May\n"); break;
        case 6:    puts("June\n"); break;
        case 7:    puts("July\n");break;
        case 8:    puts("August\n"); break;
        case 9:    puts("September\n");break;
        case 10:   puts("October\n");break;
        case 11:   puts("November\n");break;
        case 12:   puts("December\n");break;
        default :  puts("input number should be in 1~12\n");
    }
}
```

描述语句
--循环语句

while语句

- 格式为：

```
while(条件表达式)
    语句;
```

或者：

```
while(条件表达式);
```

当条件表达式成立时，反复执行语句；如果条件表达式不成立，不执行语句。其中：

- 在语句中，必须有控制条件表达式的描述；
- 对于第二种while语句来说，当满足表达式时，一直进行while的判断，执行空操作。
- 该语句经常用于延迟或者轮询标志位的应用中。

描述语句
--循环语句

【例】while语句的例子

该例子计算1+2+…+100的和，并打印计算结果。

```
void main()
{     int s=0,i=1;
      SCON= 0x52;
      TMOD = 0x20;
      TCON = 0x69;
      TH1 = 0xF3;
      while(i<=100)
      {     s+=i;
            i++;
      }
      printf("1+2+3+...+100= %d\n",s);
}
```

描述语句
--循环语句

do-while语句

- 格式为：

  do 语句

  while(条件表达式);

【例】do-while语句的例子

该例子计算2的n次幂，n值由串口终端输入，并打印2的n次幂。

```
void main()
{
    int i=0,n;
    long int p=1;
    SCON= 0x52;
    TMOD = 0x20;
    TCON = 0x69;
```

描述语句
--循环语句

```
    TH1 = 0xF3;
    printf("the following will calculate 2**n\n");
    printf("please input n value(0~16)\n");
    do{
        p=1;
        i=0;
         scanf("%d",&n);
        do{
              p=2*p;
              i++;
          }
         while(i<n);
         printf("2**%d= %ld\n",n,p);
    }
while(1);
}
```

描述语句
--循环语句

for语句

- 格式为：

  for (表达式1;表达式2;表达式3) 语句

注：当有多条语句存在时，必须用{}括起来。

- 该循环结构的执行过程为：
  - 计算机先求出表达式1的值。
  - 然后求解表达式2，如果表达式2成立，则执行语句；否则，退出循环；
  - 求解表达式3的值。
  - 返回第2步继续执行。

描述语句
--循环语句

【例】for语句的例子

计算1+2+22+23+…..+263的和，并以10进制格式和指数格式打印计算的结果。

```
void main()
{
    int i=0,n=1;
    float p=1;
    float t=1.0;
    SCON= 0x52;
    TMOD = 0x20;
    TCON = 0x69;
    TH1 = 0xF3;
```

描述语句
--循环语句

```
    for(i=1;i<64;i++)
    {
      p=p*2;
      t+=p;
    }
    printf("sum = %f\n",t);
    printf("sum = %e\n",t);
}
```

注：读者可以进入到本书所提供资料的STC_example\例子6-28目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5按键运行程序，并打开UART #1窗口。

描述语句
--循环语句

goto语句

- 该语句是无条件跳转语句，格式为：

```
标号：
        ..............
        ..............
        goto 标号;
```

- 在C语言中，goto语句只能从内层循环跳到外层循环，而不允许从外层循环调到内层循环。

注：goto语句会破坏层次化设计结构，尽量少用。

描述语句
--循环语句

【例】goto语句的例子

当按键输入不是0时，一直提示输入0，直到按键输入为0时，提示退出程序。

```
void main()
{
    int a;
    SCON= 0x52;
    TMOD = 0x20;
    TCON = 0x69;
    TH1 = 0xF3;
```

描述语句
--循环语句

```
    loop:
      puts("please input 0 to end loop\n");
      scanf("%d",&a);
      if(a!=0)
          goto loop;
      else
          puts("end program\n");
}
```

描述语句
--break语句

前面在switch语句中，使用了break语句。

- 在循环语句中，break用于终止循环的继续执行，即退出循环。

描述语句
-- continue语句

continue语句和break语句一样，也可以打断循环。

- 但是，与break语句不同的是，contiune语句尽是不执行当前循环contiune后面的语句。但是，可以继续执行下一次的循环。

描述语句

【例】break和continue语句的例子

该例子执行从1到100的循环，每执行一次循环给出一个提示信息。

在循环中设置了break语句和continue语句。

```
void main()
{
    int i;
   SCON= 0x52;
   TMOD = 0x20;
   TCON = 0x69;
   TH1 = 0xF3;
```

描述语句

```
    for(i=0;i<100;i++)
    {
        if(i==50) continue;
        if(i==80) break;
        printf("i=%d is performed\n",i);
    }
}
```

注：读者可以进入到本书所提供资料的STC_example\例子6-30目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5按键运行程序，并打开UART #1窗口。

描述语句
--返回语句

返回语句用于终止程序的执行，并控制程序返回到调用函数时的位置。

- 返回语句格式：

```
return(表达式);
```

或者

```
return;
```

数组
--一维数组的表示方法

数组用数据类型、标识符和数组所含数据的个数进行标识。

- **对于一维数组，比如：**

  int A[10]

  - **该数组用标识符A标识，该数组共有10个元素。**
  - **每个元素的数据类型为int类型。**
  - **该数组中的每个元素通过索引号标识。**

数组
--一维数组的表示方法

- 对于A[10]这个数组
  - A[0]表示该数组的第一个数据元素；
  - A[1]表示该数组的第二个数据元素；
  - A[2]表示该数组中的第三个数据元素；
  - .....；以此类推；
  - A[9]表示该数组中的第十个数据元素。
- 对于数组中的每个数据元素来说，可以在声明数组的时候就给其赋值，也可以在后面动态地给其赋值。

数组
--一维数组的表示方法

【例】一维数组声明和赋值语句的例子

在该例子中，声明了三个数组a、b、c，其中：

- 数组a中，有10个int数据元素，其索引号从0~9；
- 数组b中，有4个char类型数据元素，其索引号从0~3；
- 数组c中，有40个char数据元素，其索引号从0~39。

注：数组c和b虽然都是char类型的，但是赋值方式并不相同，b数组是每个元素分别赋值；而c数组是整体赋值。

数组
--一维数组的表示方法

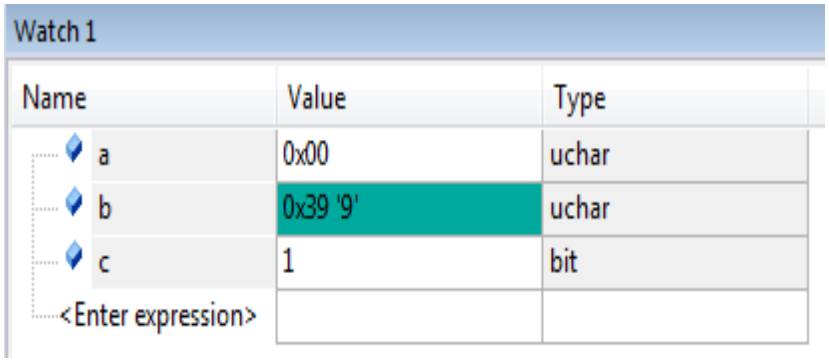
```
void main()
  {
     int a[10]={0,1,2,3,4,5,6,7,8,9};
     char b[4]={'a','b','c','d'};
     char c[40]={"hebin hello"};
     return 1;
}
```

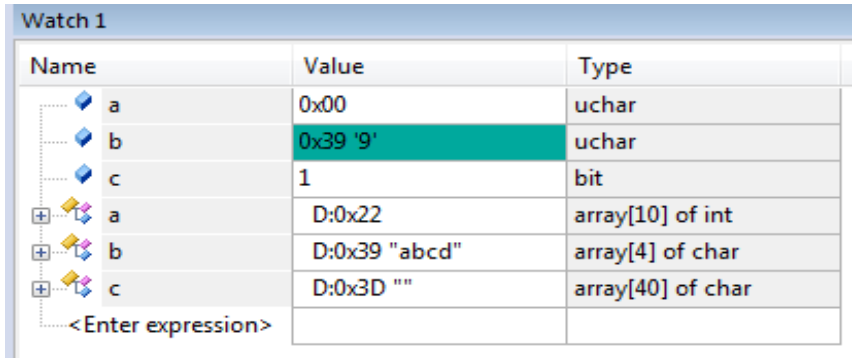
注：读者可以进入到本书所提供资料的STC_example\例子6-31目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，单步运行程序。使得单步运行到return 1的代码。

数组
--一维数组的表示方法

分析步骤如下：

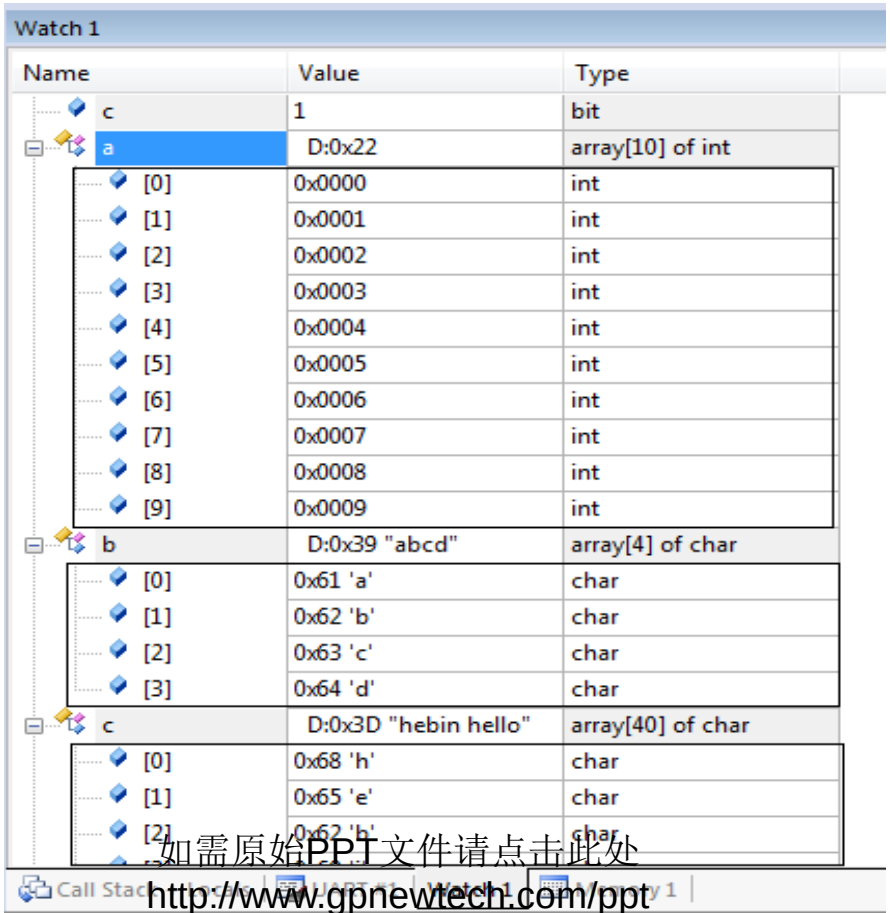
- 在当前调试主界面主菜单下，选择Watch Windows->Watch 1。
- 出现Watch 1界面。在该界面中，单击Enter expression，然后输入a；在下一行又出现Enter expression，然后输入b；按照类似的方法输入c。





数组
--一维数组的表示方法

- 在下图所示的界面中，分别单击a、b和c前面的+号，展开数组。可以看到各个数组的数据元素的值。

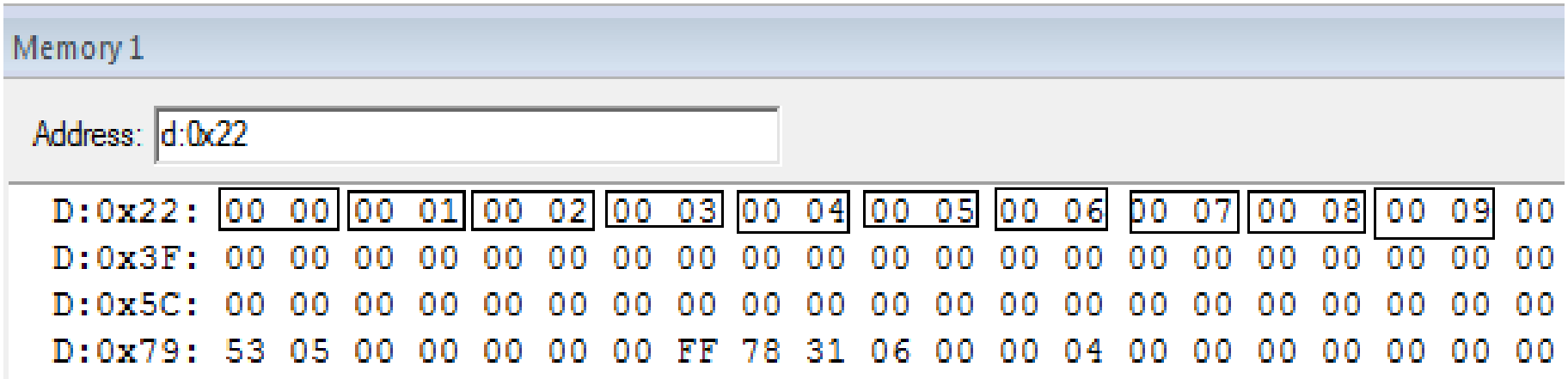


数组
--一维数组的表示方法

- 在图中可以看到下面的信息：
  - a右侧给出D:0x22信息，表示数组a存放在单片机内部数据区起始地址为0x22的区域；
  - b右侧给出D:0x39信息，表示数组b存放在单片机内部数据区起始地址为0x39的区域；
  - c右侧给出D:0x3D信息，表示数组c存放在单片机内部数据区起始地址为0x3D的区域。

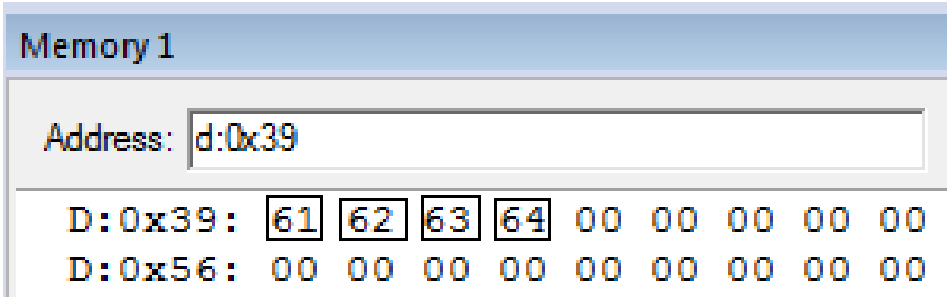
数组
--一维数组的表示方法

- 在Memory 1窗口内的Address：右侧文本框中输入0x22。可以看到从D：0x22开始的区域，每两个字节存放一个数据元素，即：00 00，00 01，00 02，00 03，00 04，00 05，00 06，00 07， 00 08，00 09。地址从d:0x22开始，一直到d:0x35结束，一共20个字节。

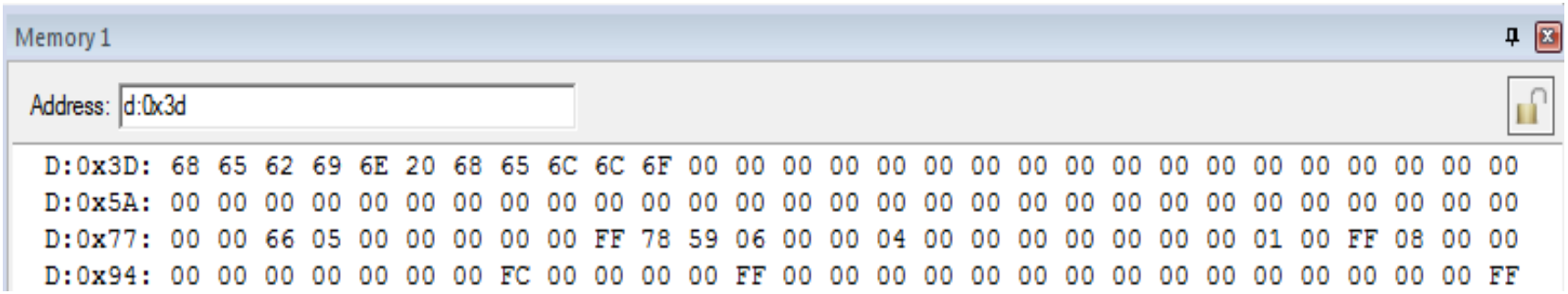


数组
--一维数组的表示方法

- 在Memory 1窗口内的Address：右侧文本框中输入0x39。



- 在Memory 1窗口内的Address：右侧文本框中输入0x3d。



数组
--多维数组的表示方法

- 多维数组的格式为：

数据类型 数组名[维数1][维数2]....[维数3]

比如：

char B[5][5]

- 表示一个字符型的二维数组，即：该数组一共有25个数据元素。

char C[5][5][5]

- 表示一个字符型的三维数组，即：该数组一共有5×5×5=125个数据元素。

- 对于多维数组来说，用于定位其中每个数据元素的格式：

数组名[索引i][索引j]

数组
--多维数组的表示方法

【例】多维数组声明和赋值语句的例子

在该例子中，声明了两个数组a、b，其中：

- 数组a[3][3]为二维数组，有9个int数据元素，按下面索引号顺序：[0][0]、[0][1]、[0][2]、[1][0]、[1][1]、[1][2]、[2][0]、[2][1]、[2][2]保存数据；
- 数组b[2][2][2]中，有8个char类型数据元素，按下面索引号顺序：[0][0][0]、[0][0][1]、[0][1][0]、[0][1][1]、[1][0][0]、[1][0][1]、[1][1][0]、[1][1][1]保存数据；

数组
--多维数组的表示方法

```
void main()
{
    int a[3][3]={1,2,3,4,5,6,7,8,9};
    char b[2][2][2]={11,12,13,14,15,16,17,18};
    return 1;
}
```

注：读者可以进入到本书所提供资料的STC_example\例子6-32目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，单步运行程序。 使得单步运行到return 1的代码。

数组
--多维数组的表示方法

分析步骤如下：

- **在当前调试主界面主菜单下，选择Watch Windows->Watch 1。**
- **出现Watch 1界面。在该界面中，单击Enter expression，然后输入a；在下一行又出现Enter expression，然后输入b，如下图所示。**

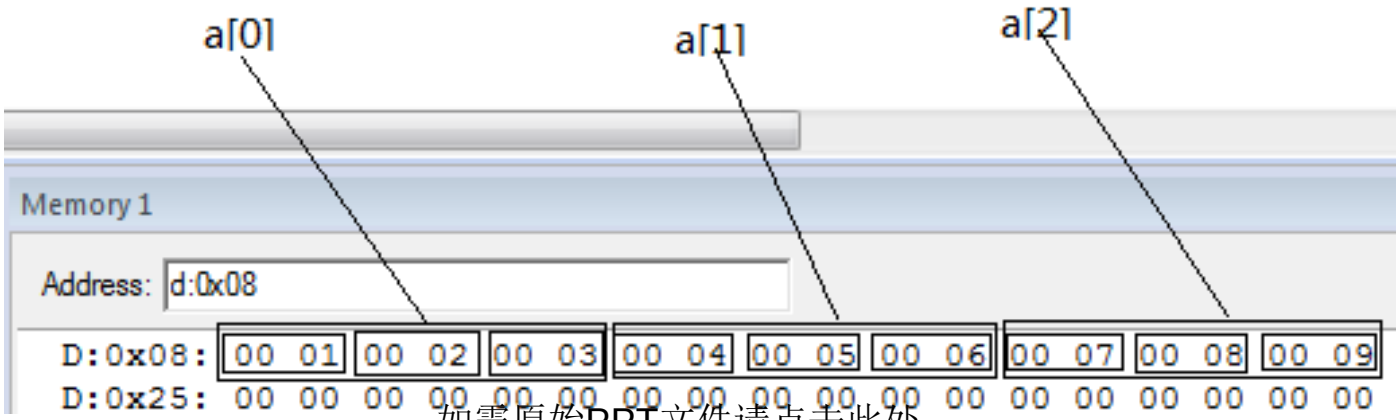
数组
--多维数组的表示方法

Watch 1

Name	Value	Type
a	D:0x08	array[3][3] of int
[0]	D:0x08	array[3] of int
[0]	0x0001	int
[1]	0x0002	int
[2]	0x0003	int
[1]	D:0x0E	array[3] of int
[0]	0x0004	int
[1]	0x0005	int
[2]	0x0006	int
[2]	D:0x14	array[3] of int
[0]	0x0007	int
[1]	0x0008	int
[2]	0x0009	int
b	D:0x1A	array[2][2][2] of char
[0]	D:0x1A	array[2][2] of char
[0]	D:0x1A "\v\f"	array[2] of char
[0]	0x0B	char
[1]	0x0C	char
[1]	D:0x1C "\r♫"	array[2] of char
[0]	0x0D	char
[1]	0x0E	char
[1]	D:0x1E	array[2][2] of char
[0]	D:0x1E "※╋"	array[2] of char
[0]	0x0F	char
[1]	0x10	char
[1]	D:0x20 "◀↕"	array[2] of char
[0]	0x11	char
[1]	0x12	char

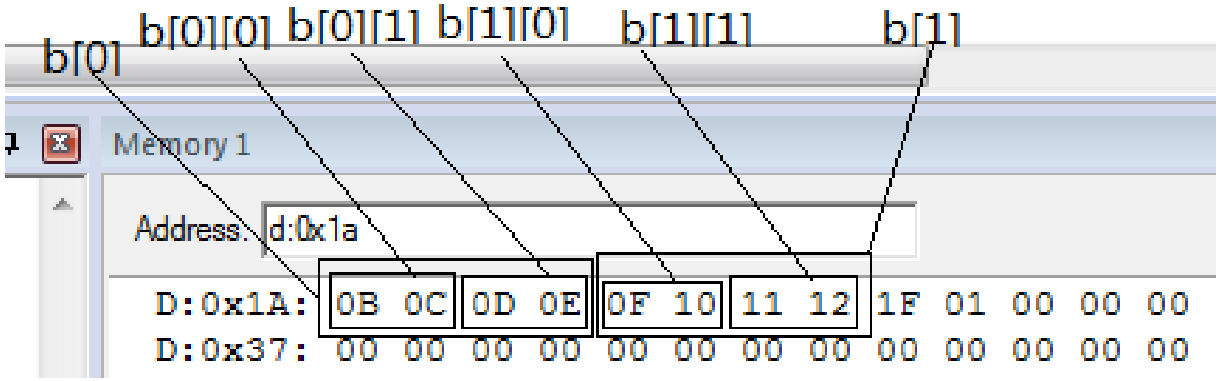
数组
--多维数组的表示方法

- 在Meomry 1界面内Address：右侧的文本框中，输入d:0x08。可以看到二维数组a的数据实际上是按照一维的形式保存在单片机的片内数据区地址为0x08的起始地址，并且是连续存放。所以，本质上，在物理存储器空间并不存在“多维的概念”，只是为了更清楚地划分数据而已。



数组
--多维数组的表示方法

- 在Meomry 1界面内Address：右侧的文本框中，输入d:0x1a。可以看到三维数组b的数据实际上是按照一维的形式保存在单片机的片内数据区地址为0x1a的起始地址，并且是连续存放。



数组
--索引数组元素的方法

【例】一维和多维数组索引元素的例子

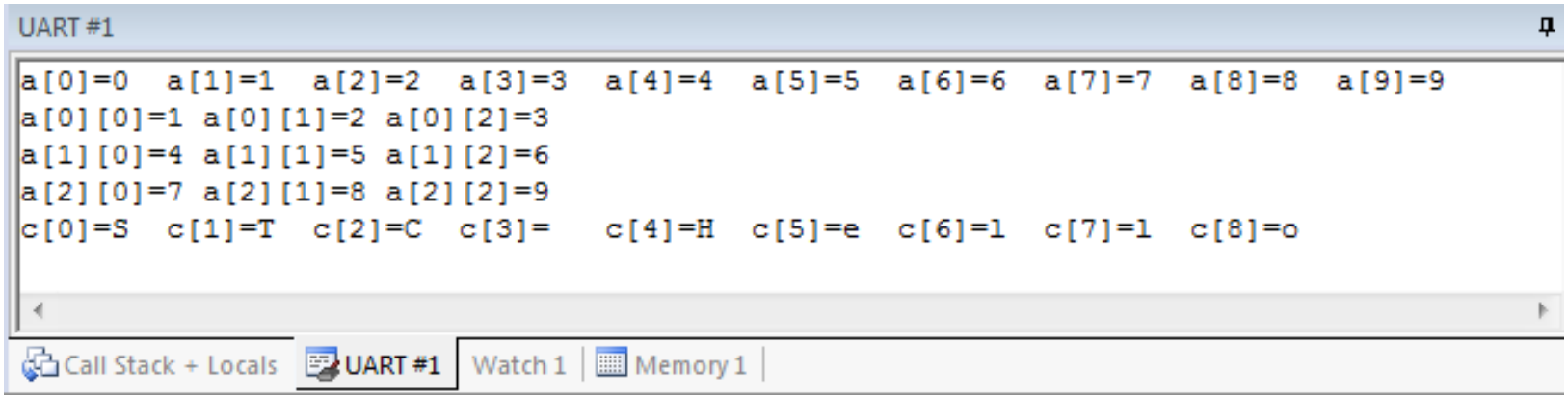
```
void main()
{
    int a[10]={0,1,2,3,4,5,6,7,8,9};
    int b[3][3]={{1,2,3},{4,5,6},{7,8,9}};
    char c[20]={"STC Hello"};
    int i=0,j=0,k=0;
    SCON= 0x52;
    TMOD = 0x20;
    TCON = 0x69;
    TH1 = 0xF3;
    for(i=0;i<10;i++)                          //循环打印一维int型数组a
    printf("a[%d]=%d  ",i,a[i]);               //索引数组a中的每个元素
```

数组
--索引数组元素的方法

```
 printf("\n");
for(i=0;i<3;i++)                                    //循环打印二维int型数组b
{
   for(j=0;j<3;j++)
      printf(“b[%d][%d]=%d ",i,j,b[i][j]);    //索引数组b中的每个元素
   printf("\n");
 }
for(i=0;i<20;i++)                                   //循环打印一维char类型数组c
{
   if(c[i]== ‘\0’ ) break;                          //判断字符是否结束，中断执行
   printf(“c[%d]=%c ” ,i,c[i]);                     //索引数组c中的每个元素
}
printf("\n");
}
```

数组
--索引数组元素的方法

注：可以进入到本书所提供资料的STC_example\例子6-33目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5运行程序。在UART #1窗口界面中，给出了数组a、b和c索引号和各个数据元素之间的关系。



数组
--索引数组元素的方法

下面对该段程序关键部分进行详细的说明。

- 在程序开始的地方，有一行代码

```
char c[20]={"STC Hello"};
```

  - 该行代码表示字符型数组c有20个元素，也就是单片机片内数据区存储空间为数组c分配20个char类型的数据元素。
  - 在实际中，赋值了一个字符串“STC Hello”，这个字符串包含9个字符，分别是'S'、'T'、'C'、' '、'H'、'e'、'l'、'l'、'o'。
  - 当然也可以采用下面的赋值方式，分别进行赋值：

```
char c[20]={'S','T','C',' ','H','e','l','l','o'};
```

  - 很明显，用字符串整体赋值比单个字符分别赋值要简单很多。但是，单个字符分别赋值，其索引号对应关系比字符串要清晰很多。

数组
--索引数组元素的方法

- 在C语言中，规定在最后一个赋值的字符后，插入‘\0’，表示字符的结束。因此虽然可以分配20个字节，但是只分配了9个字符。在下面的代码中：

```
for(i=0;i<20;i++)
 {
     if(c[i]=='\0') break;
     printf("c[%i]=%c  ",i,c[i]);
}
printf("\n");
```

数组
--索引数组元素的方法

- 为了不打印出没有分配的字符，因此在循环索引字符时，判断当前的字符c[i]是不是结束符'\0'，如果是结束符，则通过break语句停止继续打印下面的字符。

注：char类型其实不是什么特殊的数据类型，只不过在存储空间中，占用一个字节而已。由于通用的ASCII码范围在0~255之间，而char类型也在0~255之间，所以将其称为字符型而已。

数组
--索引数组元素的方法

- 在程序开始处代码：

```
b[3][3]={{1,2,3},{4,5,6},{7,8,9}};
```

其效果与

```
b[3][3]={1,2,3,4,5,6,7,8,9};
```

一样。

注：在{}里再使用{}括号，只是为了更好的表示数据存储的顺序而已。

数组
--动态输入数组元素的方法

本节将说明通过动态输入数据为数组赋值的方法。

【例】动态输入数组元素的例子

数组
--动态输入数组元素的方法

```
#include "stdio.h"
#include "reg51.h"
void main()
{
        int a[8];
        int b[3][3];
        xdata char str[40];          //将字符数组str定义在单片机xdata区
        int i,j;
        SCON= 0x52;
        TMOD = 0x20;
        TCON = 0x69;
        TH1 = 0xF3;
```

数组
--动态输入数组元素的方法

```
printf("please input data of a[8]\n");      //提示输入a[8]数组数据元素
for(i=0;i<8;i++)                            //循环语句
    scanf("%d,",&a[i]);                     //依次输入a[i]的值
    putchar(‘\n’);                          //换行
printf("please input data of b[3][3]\n");   //提示输入b[3][3]数组数据元素
for(i=0;i<3;i++)                            //外循环语句
        {
            for(j=0;j<3;j++)                //内循环语句
                scanf("%d,",&b[i][j]);
        }
putchar('\n');
```

数组
--动态输入数组元素的方法

```
printf("please input string of str[40]\n");
// scanf("%s,",str);                    //(可选的)使用该语句输入字符串
 // gets(str,40);                       //(可选的)使用该语句输入字符串
for(i=0;i<40;i++)
   {
        scanf("%c",&str[i]);            //根据索引号和scanf，得到字符数据
        if(str[i]==‘\n’) break;         //如果输入字符为换行，则停止输入
    }
putchar('\n');                          //换行
```

数组
--动态输入数组元素的方法

```
for(i=0;i<8;i++)
      printf(“a[%d]=%d,”,i,a[i]);                    //循环语句打印输入的a数组数据值
printf(“\n”);                                        //换行
for(i=0;i<3;i++)
          {
              for(j=0;j<3;j++)
              printf("b[%d][%d]=%d,",i,j,b[i][j]);
              putchar(‘\n’);                         //二维数组换行
          }
puts(str);                                           //输出字符串
while(1);
}                                                    //程序结束
```

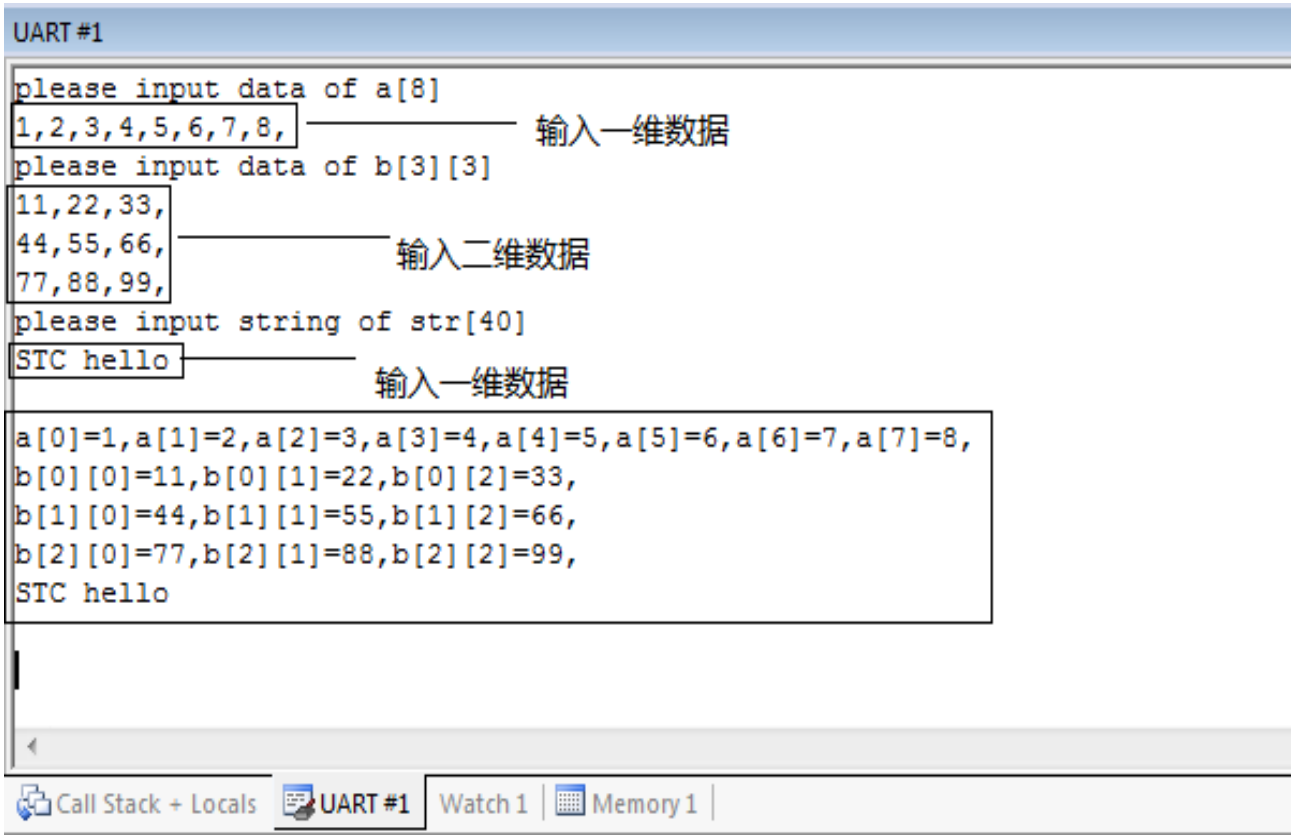
数组
--动态输入数组元素的方法

注：读者可以进入到本书所提供资料的STC_example\例子6-34目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5运行程序。

- **在UART #1窗口界面中，给出数据输入的格式和输出信息的格式。**

  按照下面的格式输入信息：

数组
--动态输入数组元素的方法



数组
--动态输入数组元素的方法

下面对该代码的关键部分进行说明：

- 在代码开始部分，有下面一行代码：

  xdata char str[40];

  - 该行代码用于声明将字符型数组str放在单片机的xdata区，这样可以避免因为内部数据区空间有限，而导致无法为str数组分配存储空间的情况。

- 在代码中，可以看到在输入数据的时候，在数组标识符前面加‘&’符号前缀，用于表示取出当前存储器数据元素的地址，其中每个存储器数据元素通过数组名和索引号进行标识。

数组
--动态输入数组元素的方法

- 对于字符型数组的数据输入，该代码给出了以下可选择的方法：

```
for(i=0;i<40;i++)
  {
        scanf("%c",&str[i]);
        if(str[i]=='\n') break;
  }
```

  - 这种方法，就是分别输入字符串中的每个字符。如果不继续输入字符，则按回车键，跳出该循环。

数组
--动态输入数组元素的方法

scanf("%s,",str);

- 这种方法，就是直接输入字符串。但是，当使用这种方法时，不允许在字符之间存在空格，这是因为空格在此表示输入字符的结束。

gets(str,40);

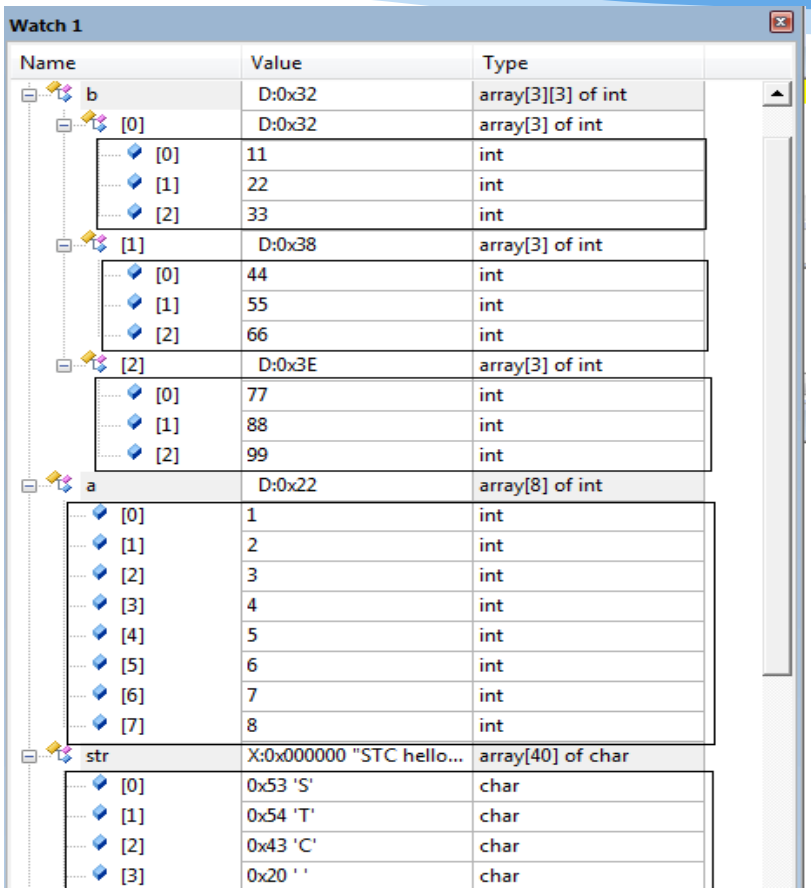
- 与第二种方法类似，注意在调用该函数的时候，需要说明输入字符的最大个数。采用这种方法输入字符串时，允许在字符之间存在空格。

数组
--动态输入数组元素的方法

下面查看一下Watch 1窗口内容和Memory 1窗口内容，其步骤主要包括：

- 在Watch 1窗口中，分别输入数组名字b、a和str。可以看到数组保存的数据信息，如下图所示。

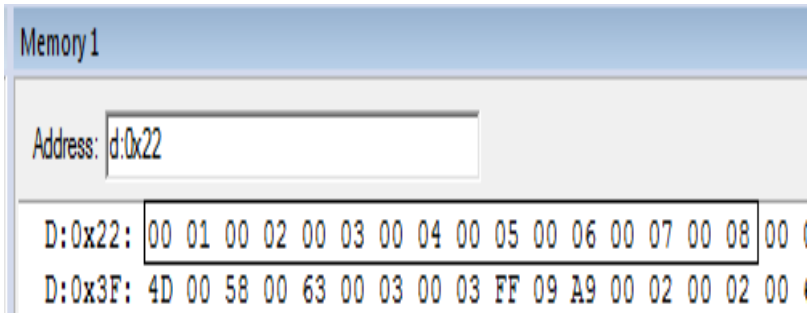
数组
--动态输入数组元素的方法

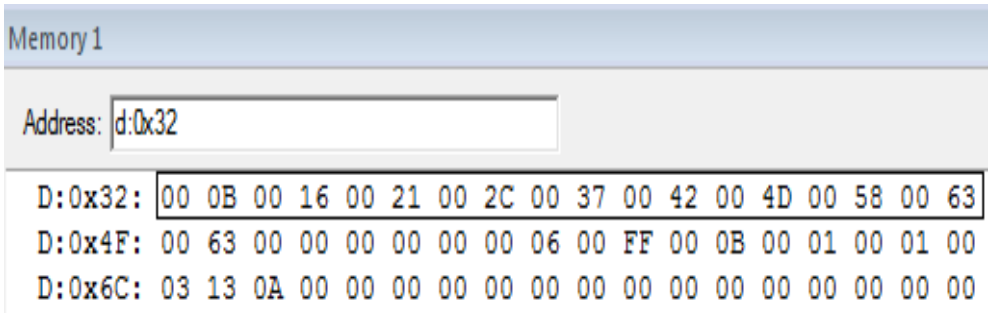


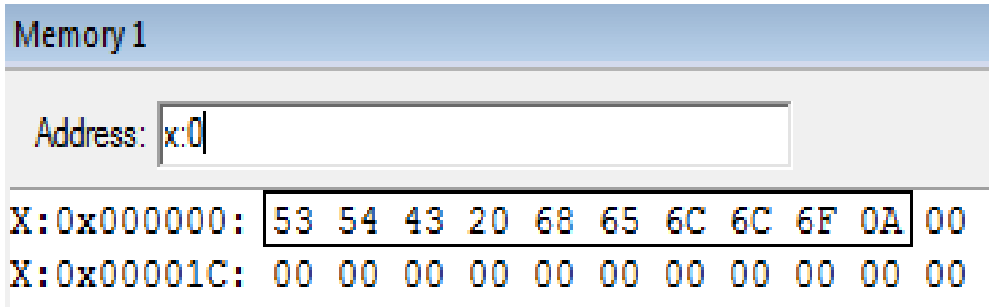
注：从图中可以看到数组b的起始地址为D:0x32，数组a的起始地址为D:0x22，数组c的起始地址为x:0x000000

数组
--动态输入数组元素的方法

- 在Memory 1窗口中，分别输入d:0x22、 d:0x32和x:0x000000，查看存储器的内容，如下图所示。







- 退出调试器模式，并关闭该设计。

数组
--数组运算算法

【例】矩阵乘法的例子

```
#include "stdio.h"
#include "reg51.h"
#define  row_a 4                  //宏定义矩阵A的行个数
#define  col_a 3                  //宏定义矩阵A的列个数
#define  row_b 3                  //宏定义矩阵B的行个数
#define  col_b 2                  //宏定义矩阵B的列个数
```

数组
--数组运算算法

```
void main()
{
        int a[row_a][col_a];
        int b[row_b][col_b];
        int c[row_a][col_b];
        int i,j,k;
        int m,n,o,p;
        SCON= 0x52;                    //初始化串口相关
        TMOD = 0x20;                   //初始化串口相关
        TCON = 0x69;                   //初始化串口相关
        TH1 = 0xF3;                    //初始化串口相关
        m=row_a;
        n=col_a;
        o=row_b;
        p=col_b;
```

数组
--数组运算算法

```
printf("please input data of a[%d][%d]\n",m,n);    //打印提示输入数组a信息
for(i=0;i<row_a;i++)                                //二重循环语句
        {    for(j=0;j<col_a;j++)
             scanf(“%d,”,&a[i][j]);                 //输入数组a的数据信息
        }
putchar('\n');
printf("please input data of b[%d][%d]\n",o,p);    //打印提示输入数组b信息
for(i=0;i<row_b;i++)                                //二重循环语句
        {    for(j=0;j<col_b;j++)                   //输入数组b的数据信息
             scanf("%d,",&b[i][j]);
        }
        putchar('\n');
```

数组
--数组运算算法

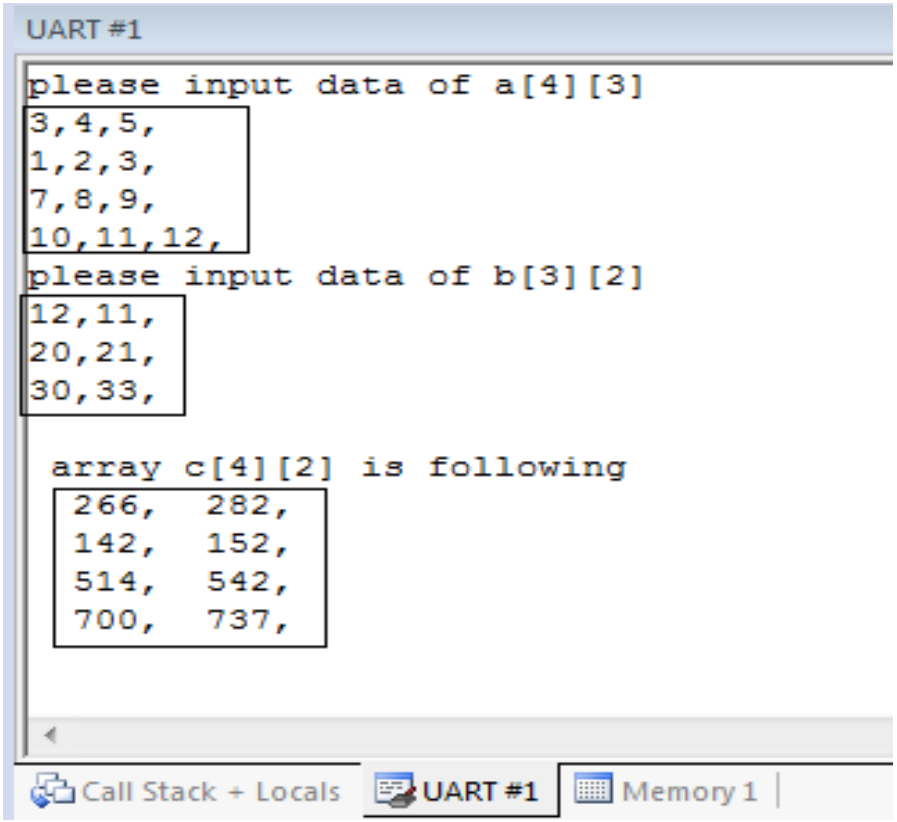
```
for (i=0;i<row_a;i++)                    //三重循环
 {
   for (j=0;j<col_b;j++)
    {
     c[i][j]=0;                          //数组c的每个元素初值为0
      for (k=0;k<col_a;k++)
     {
        c[i][j]+=a[i][k]*b[k][j];        //矩阵相应元素的乘和累加运算
     }
    }
 }
```

数组
--数组运算算法

```
printf("\n array c[%d][%d] is following\n",m,p);  //输出数组c的信息
for(i=0;i<row_a;i++)
        {
            for(j=0;j<col_b;j++)
                printf("%5d,",c[i][j]);              //输出数组c中每个数据的元素
            putchar('\n');
        }
while(1);
}
```

注：读者可以进入到本书所提供资料的STC_example\例子6-35目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5运行程序。在UART #1窗口界面中，给出数据输入的格式和输出信息的格式。

数组
--数组运算算法


UART #1
please input data of a[4][3]
3,4,5,
1,2,3,
7,8,9,
10,11,12,
please input data of b[3][2]
12,11,
20,21,
30,33,
array c[4][2] is following
266, 282,
142, 152,
514, 542,
700, 737,
Call Stack + Locals
UART #1
Memory 1

指针

C语言的一大特色就是提供了指针的功能，这就增加了C语言对单片机片内数据区和扩展数据区的管理。

- 实质上，指针的目的就是为了增加对单片机内存储空间的管理能力。
- 前面我们所定义的变量等，只能按照编译器给分配的空间地址进行存放，软件设计者没有任何能力对所分配空间的位置进行干预。

指针

指针的作用就是指向程序设计者所需要存放数据的具体的存储空间位置，然后对该存储空间位置进行取数和存数的操作。

- **更通俗的讲，就是指向一个具体的存储空间位置，然后对这个位置进行存取操作。**
- **在前面介绍了汇编语言指令时，提到存储器直接寻址和存储器的间接寻址模式。**
  - **C语言的指针就和这两种寻址模式有密切的关系。换句话说，C语言中的指针实际上就是对存储器直接寻址和间接寻址模式的抽象。**

指针
--指针的基本概念

指针的声明格式：

数据类型 *指针名字

- 前面提到过：

&变量/数组名字

  - 表示获取变量所在单片机存储空间的地址，或者是数组所在单片机存储空间的起始地址。

指针
--指针的基本概念

- 例如如下声明：

```
int *p1；
int a；
```

当进行下面操作：

```
p1=&a;
```

- 表示p1的值为变量a所在单片机存储空间的具体地址信息。该地址的内容就是变量a的值，用形式化的方式可以这样表示：

```
(p1)=a;
```

- *p1实际上就是获取指向地址的内容。所以，*p1的值就是变量a的值。

指针
--指针的基本概念

【例】指针基本概念的例子

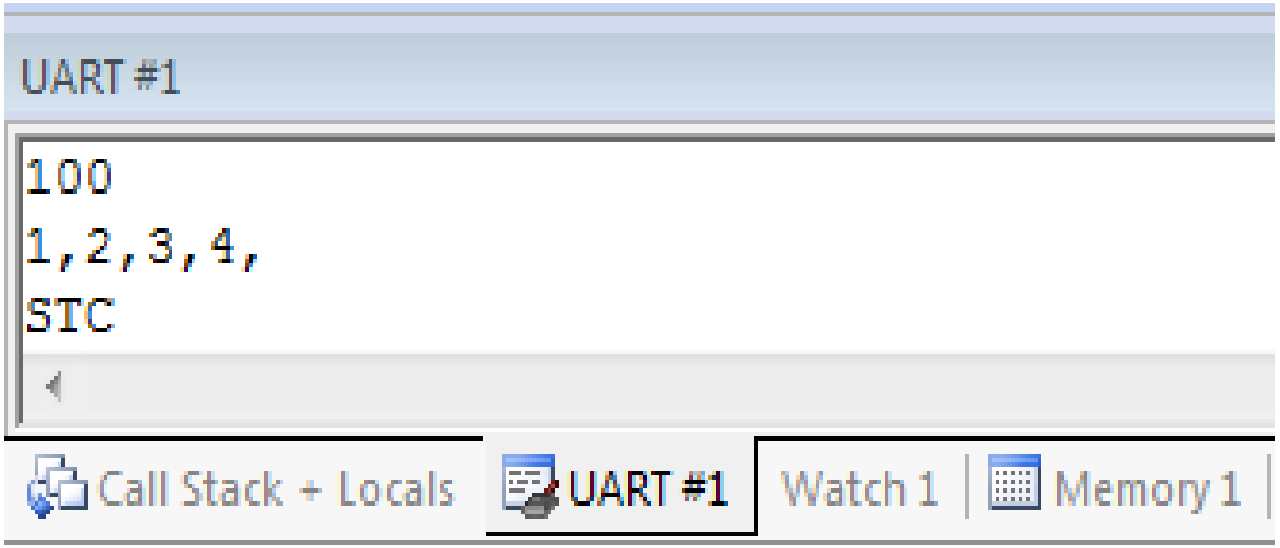
```
#include "stdio.h"
#include "reg51.h"
void main()
{
    int a=100;                  //定义整型变量
    int b[4]={1,2,3,4};         //定义整型数组
    char c[10]={“STC”};         //定义字符型的数组
    int *p1,*p2;                //定义指向整型数的指针
    char *p3;                   //定义指向字符型的指针
    SCON= 0x52;
    TMOD = 0x20;
    TCON = 0x69;
    TH1 = 0xF3;
```

指针
--指针的基本概念

```
p1=&a;                          //p1为变量a所在存储空间的地址
p2=&b;                          //p2为数组b所在存储空间的首地址
p3=&c;                          //p3为数组c所在存储空间的首地址
printf(“%d\n”,*p1);             //打印p1单元内存储空间的内容
printf(“%d\n”,*p2);             //打印p2单元内存储空间的内容
printf(“%d\n”,*(++p2));         //打印(p2+1)单元内存储空间的内容
printf(“%d\n”,*(++p2));         //打印(p2+2)单元内存储空间的内容
printf(“%d\n”,*(++p2));         //打印(p2+3)单元内存储空间的内容
printf(“%c”,*p3);               //打印p3单元内存储空间的内容
printf(“%c”,*(++p3));           //打印(p3+1)单元内存储空间的内容
printf(“%c\n”,*(++p3));         //打印(p3+2)单元内存储空间的内容
while(1);
}
```

指针
--指针的基本概念

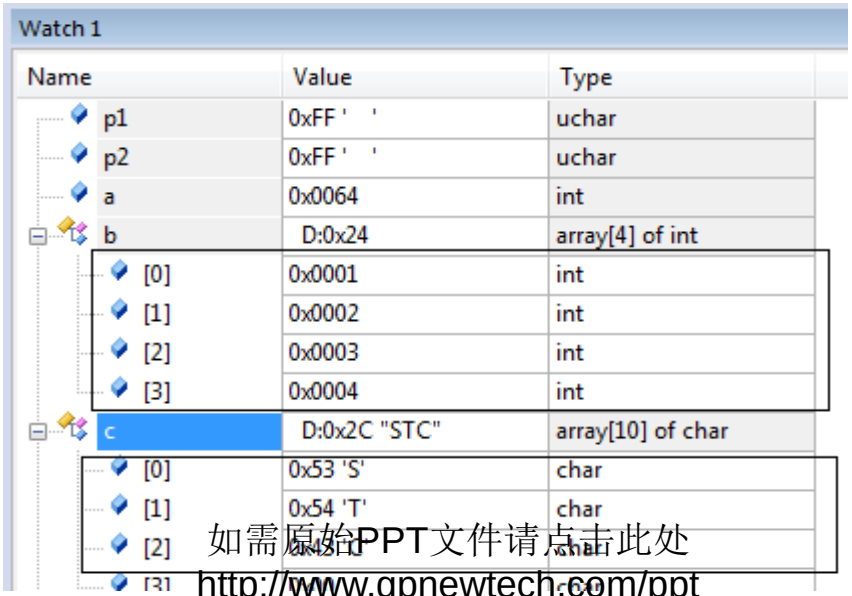
注：读者可以进入到本书所提供资料的STC_example\例子6-36目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5运行程序。在UART #1窗口界面中，给出输出信息的格式。



指针
--指针的基本概念

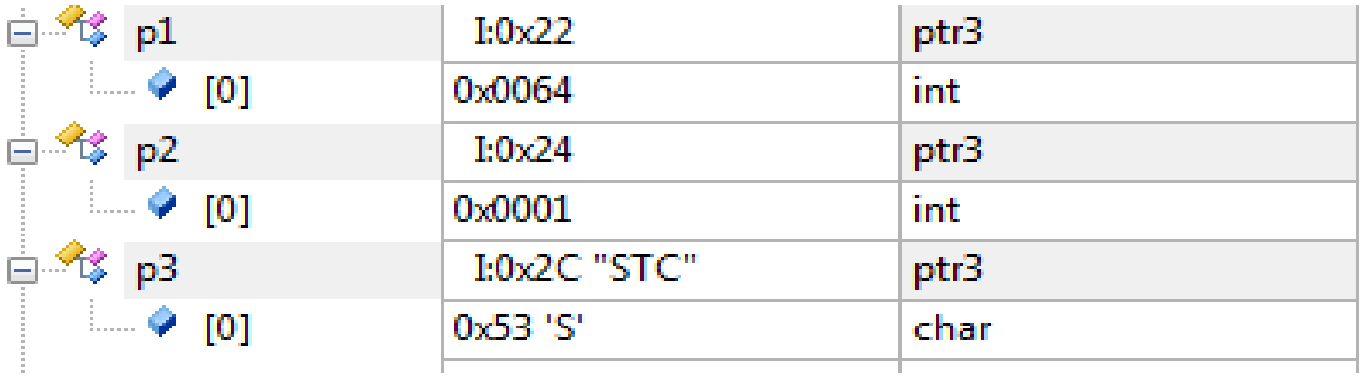
下面对该程序进行详细的分析：

- **按前面的方法，打开Watch 1窗口界面在该界面中，分别输入a，b和c，出现变量a的值，数组b的数据元素的值以及数组b的首地址，数组c的数据元素的值以及数组c的首地址。**



指针
--指针的基本概念

- 重新单步运行程序，程序执行完16行代码，在Watch 1窗口界面内输入p1,p2和p3的值，可以看到给出的信息。



p1	I:0x22	ptr3
[0]	0x0064	int
p2	I:0x24	ptr3
[0]	0x0001	int
p3	I:0x2C "STC"	ptr3
[0]	0x53 'S'	char

- 从图中可以看出，P1的内容为I:0x22，为地址信息，即指向单片机片内数据区位置为0x22的地址，该地址内容为0x0064。这是因为在程序代码中，设置将P1设置为变量a所在的地址。

指针
--指针的基本概念

- 类似的，P2的内容为I:0x26，为地址信息，即指向单片机片内数据区位置为0x26的地址，该地址内容为0x0002。在程序代码中，设置将P2设置为指向数组b的首地址，并且数组b的第一个数据元素的值为1。形式化表示为：

(p2)=1=b[0]=*p2

p1	I:0x22	ptr3
[0]	0x0064	int
p2	I:0x24	ptr3
[0]	0x0001	int
p3	I:0x2C "STC"	ptr3
[0]	0x53 'S'	char

- 也可以这样说，p2的内容就通过指针*p2表示。

指针
--指针的基本概念

- P3的内容为I:0x2c，为地址信息，即指向单片机片内数据区位置为0x2c的地址，该地址内容为0x53。在程序代码中，设置将P2设置为指向数组c的首地址，并且数组c的第一个数据元素的值为字符‘S’（该字符的ASCII值为0x53）。形式化表示为：

(p3)=0x53=c[0]=*p3

p1	I:0x22	ptr3
[0]	0x0064	int
p2	I:0x24	ptr3
[0]	0x0001	int
p3	I:0x2C "STC"	ptr3
[0]	0x53 'S'	char

- 也可以这样说，p3的内容就通过指针*p3表示。

指针
--指针的基本概念

- 继续单步执行完第18行代码，相继打印出*p1和*p2的值，为100和1。这个结果和前面分析的一致。
- 单步执行完第19行代码，即：

```
printf("%d,",*(++p2));
```

  - 该代码让p2在首地址基础上递增，由于p2指向的是int类型，所以实际地址在首地址基础上增加2。此时，p2的内容为I：0x26，指向单片机片内数据区位置为0x26的地址，该地址的内容为0x0002。形式化表示为：

(p2)=0x0002=b[1]=*p2

p1	I:0x22	ptr3
[0]	0x0064	int
p2	I:0x26	ptr3
[0]	0x0002	int
p3	I:0x2C "STC"	ptr3
[0]	0x53 'S'	char

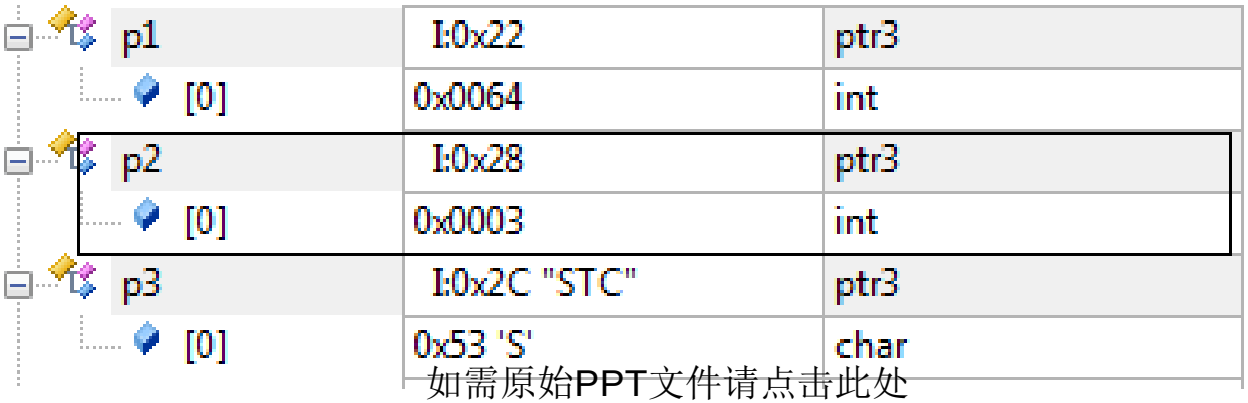
指针
--指针的基本概念

- 单步执行完第20行代码，即：

```
printf("%d,",*(++p2));
```

  - 该段代码让p2在前一个p2的基础上递增，由于p2指向的是int类型，所以实际地址在前一个地址基础上增加2。此时，p2的内容为I：0x28，为地址信息，即指向单片机片内数据区位置为0x28的地址，该地址的内容为0x0003。形式化表示为：

(p2)=0x0003=b[2]=*p2



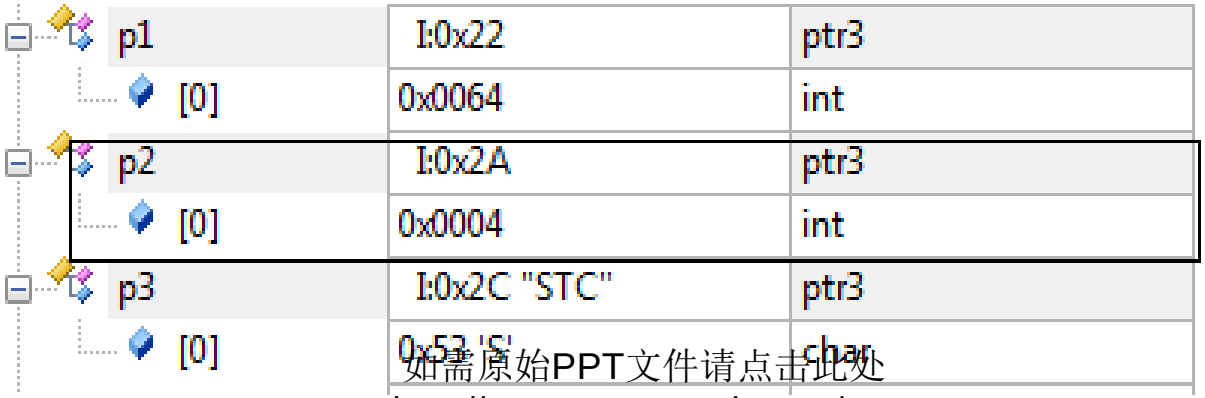
指针
--指针的基本概念

- 单步执行完第21行代码，即：

printf("%d,",*(++p2));

  - 该段代码让p2在前一个p2的基础上递增，由于p2指向的是int类型，所以实际地址在前一个地址基础上增加2。此时，p2的内容为I：0x2A，为地址信息，即指向单片机片内数据区位置为0x2A的地址，该地址的内容为0x0004。形式化表示为：

(p2)=0x0004=b[3]=*p2



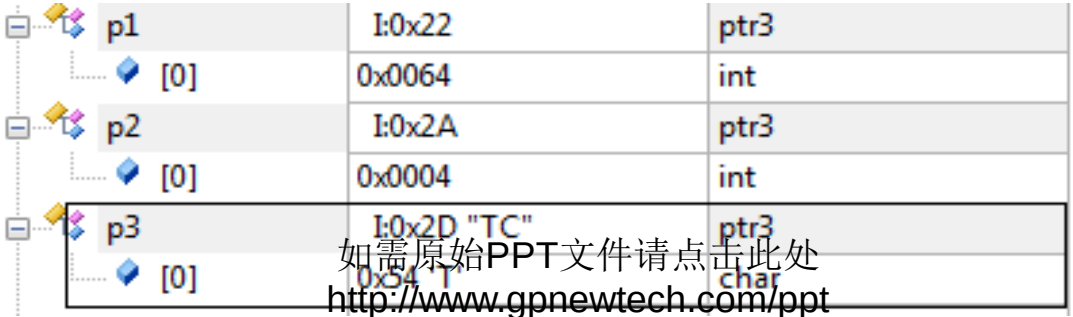
指针
--指针的基本概念

- 单步执行完第21行代码，打印出*p3的值，为字符‘S’。
- 单步执行完第22行代码，即：

```
printf("%c,",*(++p3));
```

  - 该段代码让p3在前一个p3基础上递增，由于p3指向的是char类型，所以实际地址在前一个地址基础上增加1。此时，p3的内容为I：0x2D，为地址信息，即指向单片机片内数据区位置为0x2D的地址，该地址的内容为字符‘T’（其ASCII码为0x54）。形式化表示为：

(p3)=0x54=c[1]=*p3



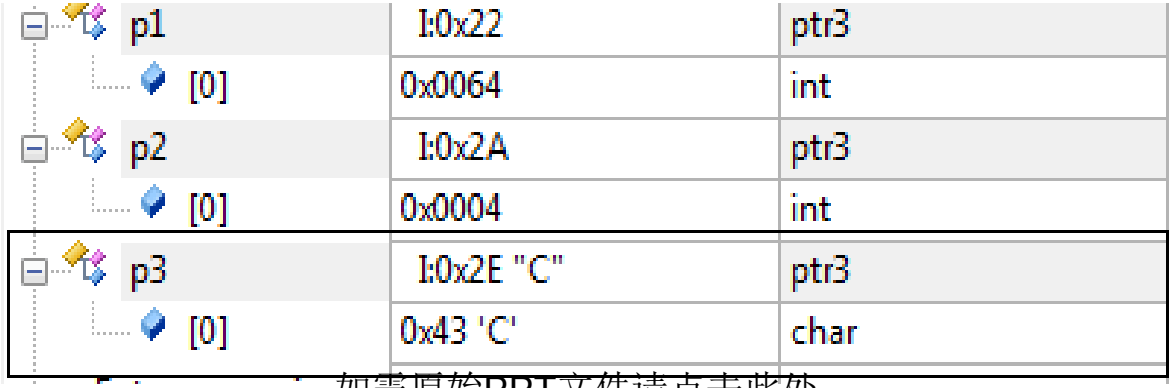
指针
--指针的基本概念

- 单步执行完第23行代码，即：

printf("%c,",*(++p3));

  - 该段代码让p3在前一个p3基础上递增，由于p3指向的是char类型，所以实际地址在前一个地址基础上增加1。此时，p3的内容为I：0x2E，为地址信息，即指向单片机片内数据区位置为0x2E的地址，该地址的内容为字符‘C’（其ASCII码为0x43）。形式化表示为：

(p3)=0x43=c[2]=*p3



p1	I:0x22	ptr3
[0]	0x0064	int
p2	I:0x2A	ptr3
[0]	0x0004	int
p3	I:0x2E "C"	ptr3
[0]	0x43 'C'	char

指针
--指针的基本概念

【例】使用指针的例子

```
#include "stdio.h"
#include "reg51.h"
void main()
{
  int a=100,b=10,t=0;                  //声明整型变量a、b和t
  int *p1,*p2,*p3;                     //声明整型指针*p1、*p2和*p3
  SCON= 0x52;
  TMOD = 0x20;
  TCON = 0x69;
  TH1 = 0xF3;
  printf("a=%d,b=%d\n",a,b);           //打印a和b的初值
  p1=&a;                               //p1指向变量a的地址
  p2=&b;                               //p2指向变量b的地址
```

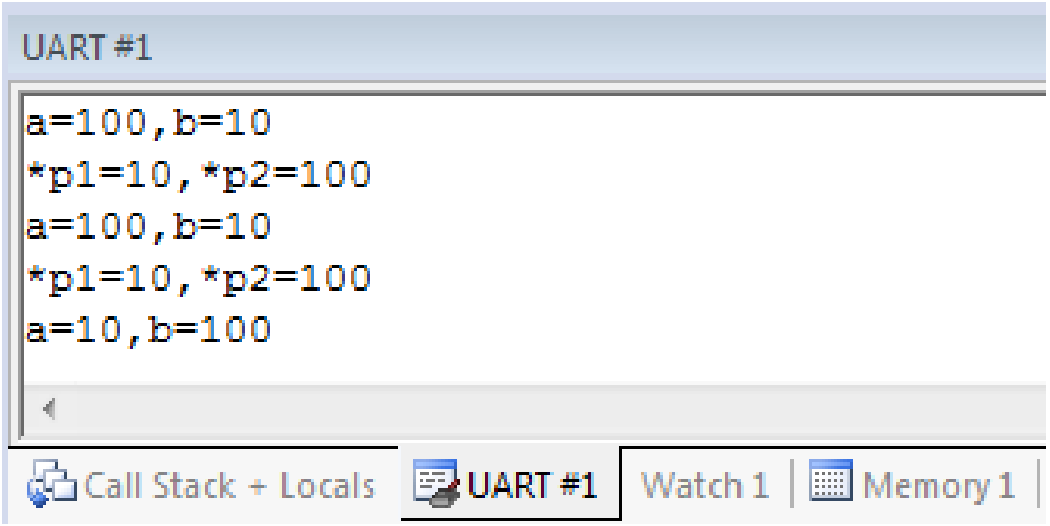
指针
--指针的基本概念

```
p3=p1;                                  //p3=p1，也就是p3指向变量a的地址
p1=p2;                                  //p1=p2，也就是p1指向变量b的地址
p2=p3;                                  //p2=p3，也就是p2指向变量a的地址
printf(“*p1=%d,*p2=%d\n”,*p1,*p2); //打印*p1和*p2的值
printf(“a=%d,b=%d\n”,a,b);              //打印a和b的值
p1=&a;                                  //p1指向变量a的地址
p2=&b;                                  //p2指向变量b的地址
t=*p1;                                  //将p1指向变量的内容赋值给t
*p1=*p2;                                //将p1指向变量内容赋给p1指向变量
*p2=t;                                  //将变量t的值赋给p2指向的变量
printf("*p1=%d,*p2=%d\n",*p1,*p2);  //打印*p1和*p2的值
printf("a=%d,b=%d\n",a,b);              //打印a和b的值
while(1);
}
```

指针
--指针的基本概念

注：读者可以进入到本书所提供资料的STC_example\例子6-37目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式.

- 将上述代码编辑后进入调试器模式，按F5运行程序。在UART #1窗口界面中，给出输出信息的格式，如下图所示。



指针
--指针的基本概念

下面对该程序进行详细的分析：

- **重新单步运行程序，运行完第11行代码，变量a的地址d:0x22；变量b的地址d:0x24。**

```
int a=100,b=10,t=0;
int *p1,*p2;
SCON= 0x52;
TMOD = 0x20;
TCON = 0x69;
TH1 = 0xF3;
```

a (D:0x22) = 0x0064

```
int a=100,b=10,t=0;
int *p1,*p2;
SCON= 0x52;
TMOD = 0x20;
TCON = 0x69;
TH1 = 0xF3;
```

b (D:0x24) = 0x000A

指针
--指针的基本概念

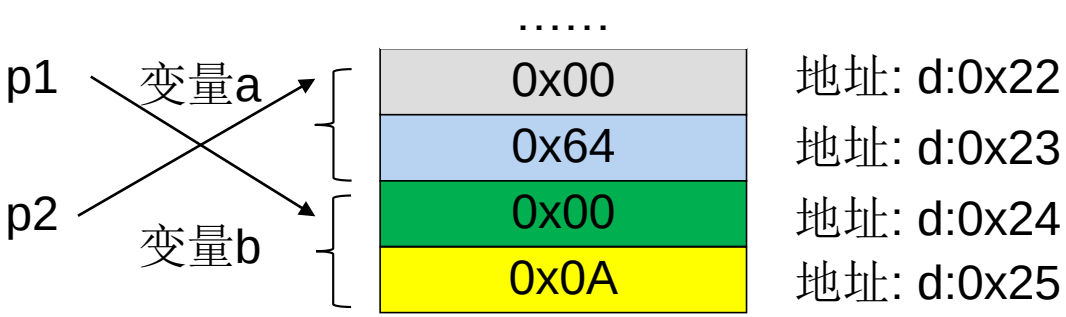
- 单步运行完第13行代码，指针p1指向变量a的地址，指针p2指向变量b的地址，如下图所示。

单片机片内数据区

p1和p2指针与存储空间的关系(1)

指针
--指针的基本概念

- **单步运行完第16行代码，指针p1指向变量b的地址，指针p2指向变量a的地址，如图所示。**

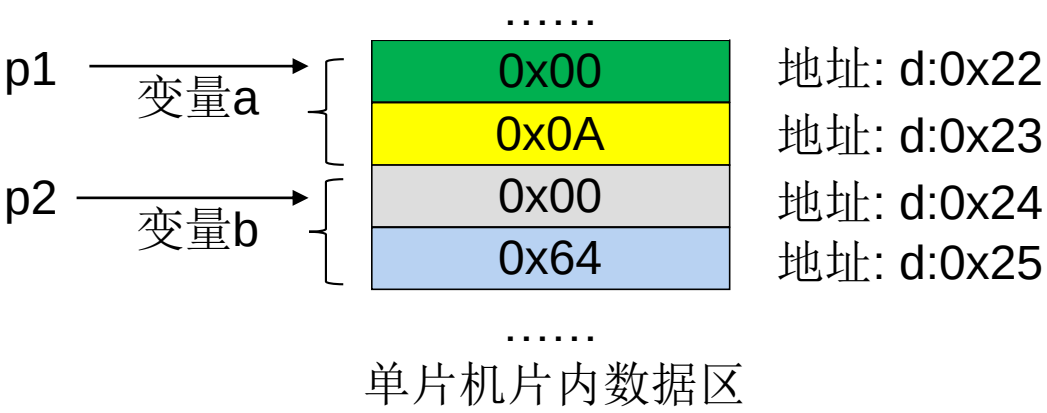


单片机片内数据区

p1和p2指针与存储空间的关系(2)

指针
--指针的基本概念

- 单步运行完第21行代码，指针p1重新指向变量a的地址，指针p2重新指向变量b的地址。
- 单步运行完第26行代码，指针p1指向变量a的地址，指针p2指向变量b的地址。



指针
--指向指针的指针

在C语言中，还提供了指向指针的指针的功能。

- 所谓的指向指针的指针实际上对应于单片机中的存储器间接寻址的概念，只是C语言将存储器间接寻址的概念抽象成指向指针的指针的概念而已。
- 声明格式为：

数据类型 **标识符；

指针
--指向指针的指针

例如如下声明：

```
int a;
int *p1;
int **p2;
```

当进行下面操作：

```
p1=&a;
p2=&p1;
```

可以表示为下面的关系，如图所示：

指针
--指向指针的指针

- 表示p1的值为变量a所在单片机存储空间的具体地址信息。该地址的内容(p1)就是变量a的值，用形式化的方式可以这样表示：

(p1)=a;

(p2)=p1;

((p2))=a;

在C语言中的，等效描述为：

*p1=a;

*p2=p1;

**p2=a;

指针
--指向指针的指针

【例】指向指针的指针基本概念的例子

```
#include "stdio.h"
#include "reg51.h"
void main()
{
    char data a=100;          //在单片机数据区内定义整型变量a
    char data *p1;            //定义指向char类型的指针
    char data **p2;           //定义指向char类型的指针的指针
    SCON= 0x52;
    TMOD = 0x20;
    TCON = 0x69;
    TH1 = 0xF3;
```

指针
--指向指针的指针

```
    p1=&a;                    //p1指向变量a的地址
    p2=&p1;                   //p2指向p1的地址
    printf(“%c\n”,**p2);      //打印指向指针的指针的内容
    while(1);
}
```

注：读者可以进入到本书所提供资料的STC_example\例子6-38目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5运行程序。

指针
--指向指针的指针

下面对该程序进行详细的分析：

- 按前面的方法打开Watch 1窗口界面。在该界面中，分别输入：a、p1、p2、*p2和**P2。此外，将光标放在a名字上，出现变量a的地址d:0x22。

Watch 1

Name	Value	Type
a	0x64 'd'	char
p1	D:0x22 "d""	data ptr
[0]	0x64 'd'	char
p2	I:0x23	ptr3
[0]	D:0x22 "d""	data ptr
*p2	D:0x22 "d""	data ptr
[0]	0x64 'd'	char
**p2	0x64 'd'	char
<Enter expression>		

Call Stack + Locals | UART #1 | Watch 1 | Memory 1

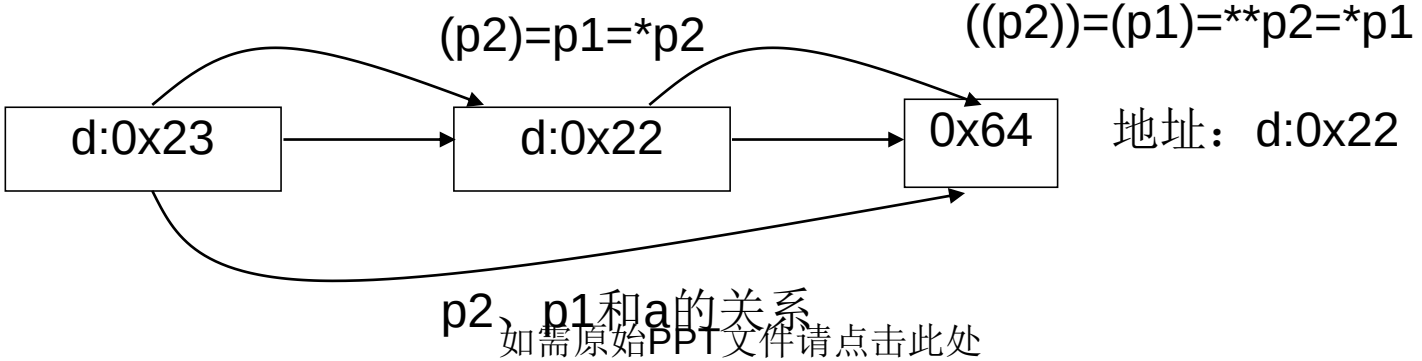
```
char data a=100;
char data a ( D:0x22) = 0x64
char data **p2;
```

指针
--指向指针的指针

- 由代码设计可知，p1指向a的地址，也就是p1的值为d:0x22。很明显*p1就是p1地址存储空间的内容(p1)=*p1=0x64，就是a的值。
- p2是p1的地址，即p1的地址在单片机内部数据区地址为I：0x23的位置。
- 很明显(p2)=p1=*p2=0x22，也就是p2的内容就是p1的值d:0x22。
- **p2，形式化表示为((p2))，即：p2单元内容的内容，(p2)= d:0x22，((p2))=(d:0x22)=0x64。

指针
--指向指针的指针

- p2、p1和a之间的访问关系。
  - 本来p2可以直接访问到a。但是，p2并没有直接访问a。而是借助了p1。p2的内容是单片机片内数据区的一个具体的存储地址，而不是数据。通过这个存储器的地址单元找到了a，这也就是说为什么指针是存储器直接寻址模式，而指针的指针是间接寻址模式的原因。也就是p2不象p1可以直接找到a，它需要借助“第三者”，也就是其它存储单元才能找到a。



p2、p1和a的关系

指针
--指向指针的指针

【例】指向数组的指针基本概念的例子

```
#include "stdio.h"
#include "reg51.h"
void main()
{
  int a[4]={0x01,0x10,0x100,0x1000};
  int *b[4]={&a[0],&a[1],&a[2],&a[3]};
  int **p2;
  int i;
 SCON= 0x52;
TMOD = 0x20;
```

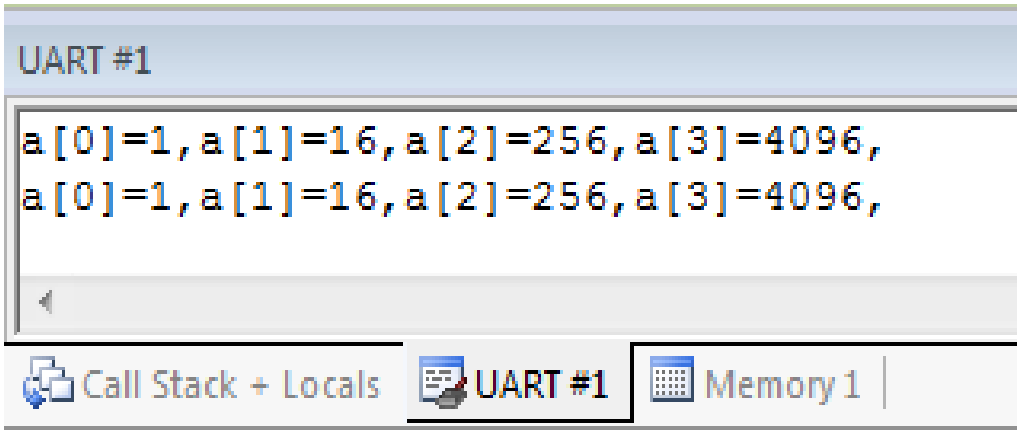
指针
--指向指针的指针

```
  TCON = 0x69;
  TH1 = 0xF3;
  p2=b;
  for(i=0;i<4;i++)
  printf("a[%d]=%d,",i,a[i]);
  putchar('\n');
  for(i=0;i<4;i++)
  printf("a[%d]=%d,",i,**(p2++));
  putchar('\n');
  while(1);
}
```

指针
--指向指针的指针

注：读者可以进入到本书所提供资料的STC_example\例子6-39目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5运行程序。

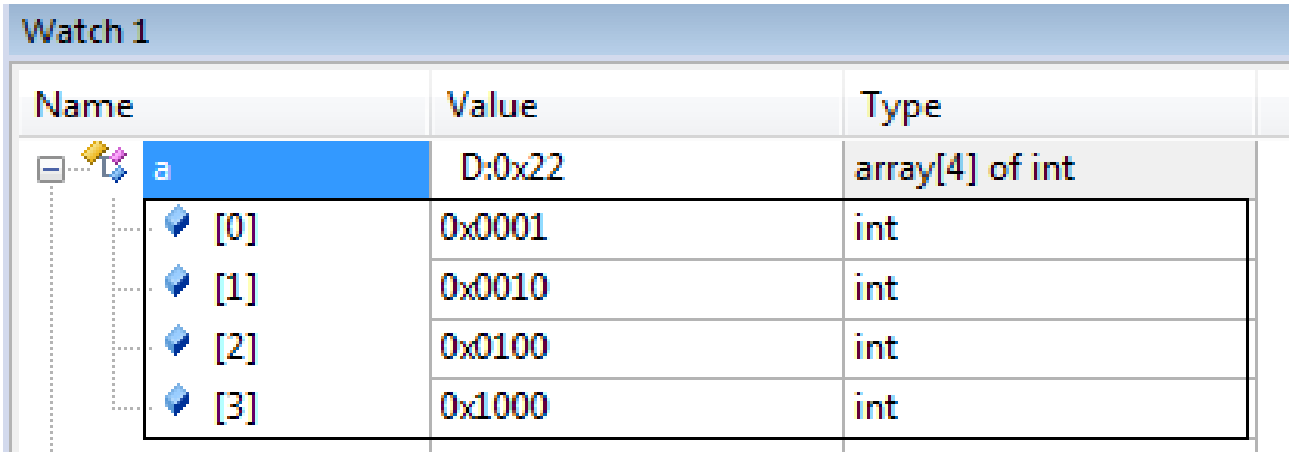
将上述代码编辑后进入调试器模式，按F5运行程序。在UART #1窗口界面中，给出输出信息的格式，如下图所示。



指针
--指向指针的指针

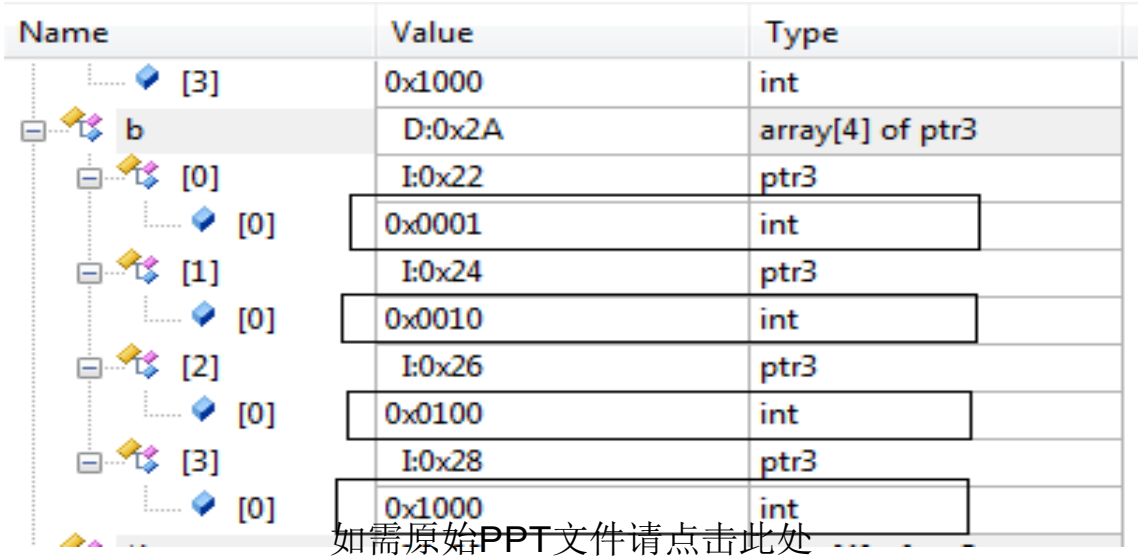
下面对该程序进行详细的分析：

- 打开Watch 1窗口界面，在该窗口中输入a，给出数组a的信息。数组a的首地址是0x22，依次为0x24、0x26、0x28。



指针
--指向指针的指针

- 类似的，在Watch 1窗口中输入b，给出b的信息。指针数组的地址在d:0x2A，该数组内保存着数组a中每个元素的地址信息I:0x22、I:0x24、I:0x26和I:0x28。在下一层，可以看到这些地址单元的内容为0x0001、0x0010、0x0100和0x1000。



Name	Value	Type
[3]	0x1000	int
b	D:0x2A	array[4] of ptr3
[0]	I:0x22	ptr3
[0]	0x0001	int
[1]	I:0x24	ptr3
[0]	0x0010	int
[2]	I:0x26	ptr3
[0]	0x0100	int
[3]	I:0x28	ptr3
[0]	0x1000	int

指针
--指向指针的指针

代码中：

p2=b;

- **表示将指针数组*b的地址给p2。因此，*p2=(p2)，就是指针数组第一个元素b[0]的值I:0x22。进一步，**p2=((p2))，就是p2内容的内容，p2的内容是I:0x22。而I:0x22的内容是0x0001。**

指针
--指针变量输入

本节将使用指针，为整数变量、字符数组、指针数组、整数数组赋值。

- 下面通过一个例子说明通过指针为变量和数组赋值的方法。

指针
--指针变量输入

【例】指针变量输入的例子

```
#include "stdio.h"
#include "reg51.h"
void main()
{
    int a=10,*p1;                          //声明整型变量a和整型指针*p1
    int i;                                 //声明整型变量i
    char b[40],*s;                         //声明字符数组b和字符型指针*s
    xdata char c[50],*s1="STC hello";  //在xdata定义数组c和字符指针s1
    xdata int d[4]={1,2,3,4},*p2;          //在xdata定义数组d和整型指针*p2
    SCON= 0x52;
    TMOD = 0x20;
```

指针
--指针变量输入

```
TCON = 0x69;
TH1 = 0xF3;
p1=&a;                                          //p1指向变量a的地址
s=&b;                                           //s指向字符数组b的首地址
s1=&c;                                          //s1指向字符数组c的首地址
p2=&d;                                          //p2指向整型数组d的首地址
printf("please input int value of pointer p1\n");
scanf(“%d”,p1);                                 //输入*p1的值
printf("please input string value of pointer s\n");
scanf(“%s”,s);                                  //输入s指向的字符串
printf("please input string value of pointer s1\n");
scanf(“%s”,s1);                                 //输入s1指向的字符串
```

指针
--指针变量输入

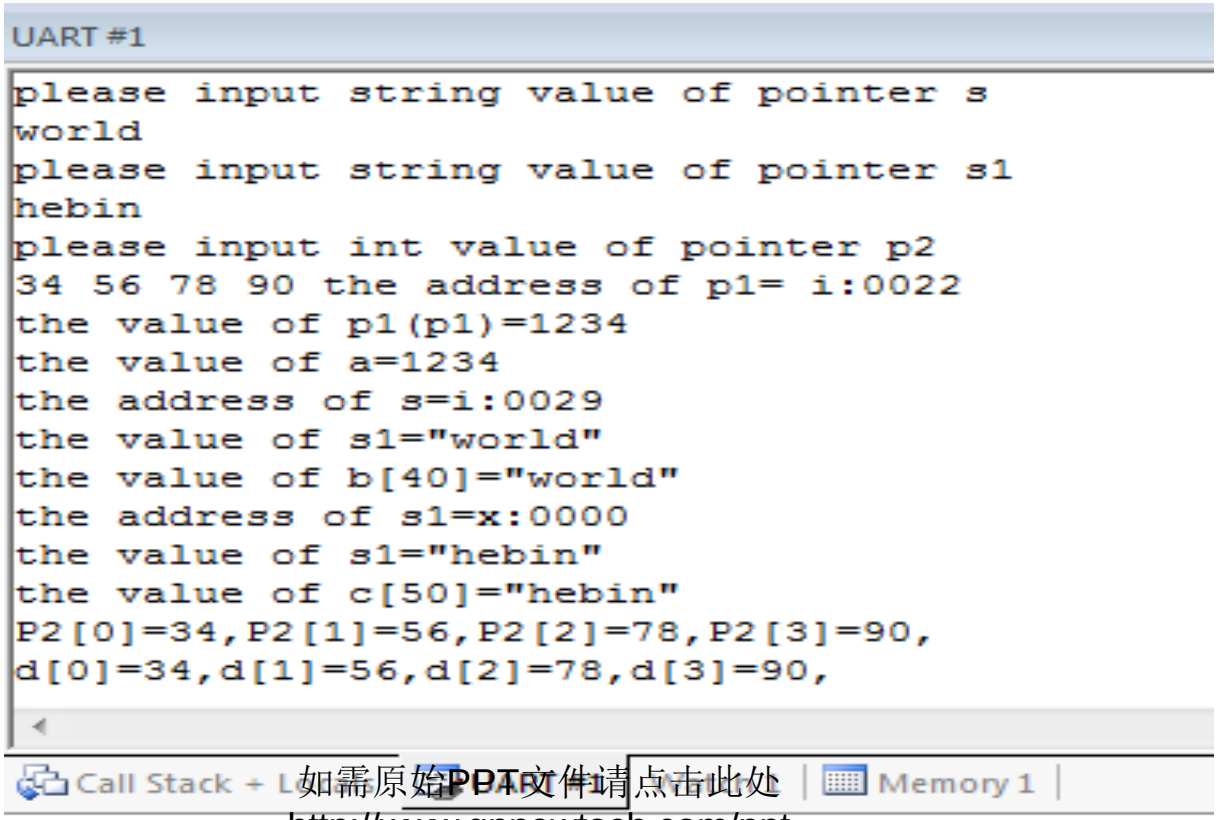
```
printf("please input int value of pointer p2\n");
for(i=0;i<4;i++)                                    //循环语句
  {   scanf(“%d”,p2);                               //输入p2指向的整数
    p2++;                                           //指针递增指向下一个地址
   }
printf(“the address of p1= %p\n”,p1);               //打印指针*p1的地址
printf(“the value of p1(p1)=%d\n”,*p1);             //打印指针*p1的内容
printf(“the value of a=%d\n”,a);                    //打印变量a的值
printf(“the address of s=%p\n”,s);                  //打印指针*s的首地址
printf(“the value of s1=\” %s\ “\n”,s);             //打印指针*s指向的字符串
printf(“the value of b[40]=\” %s\ “\n”,b);          //打印字符数组b的字符串
printf(“the address of s1=%p\n”,s1);                //打印指针*s1的地址
```

指针
--指针变量输入

```
printf("the value of s1=\"%s\"\n",s1);      //打印指针*s1指向的字符串的内容
printf("the value of c[50]=\"%s\"\n",c);    //打印字符数组c的字符串
p2=&d;                                      //指针*p2指向数组d的首地址
  for(i=0;i<4;i++)                          //循环语句
   {   printf(“p2[%d]=%d,”,i,*p2);          //打印当前*p2的内容
   p2++;                                    //p2递增，指向下一个地址
    }
putchar(‘\n’);                              //换行
for(i=0;i<4;i++)                            //循环语句
   {    printf(“d[%d]=%d,”,i,d[i]);         //打印数组d当前索引号所定应的值
     }
while(1);                                   //无限循环
}
```

指针
--指针变量输入

注：读者可以进入到本书所提供资料的STC_example\例子6-40目录下，在Keil μVision5集成开发环境下打开该设计。将上述代码编辑后进入调试器模式，按F5运行程序。在UART #1窗口界面中，给出输出信息的格式，如图所示。



函数

在C语言中，函数是构成C文件的最基本的功能。

- **前面已经说明了main()主程序的功能。**
- **本节将重点介绍子函数的声明、函数调用方法。**

函数
--函数声明

在C语言中，声明函数的格式如下：

```
函数类型  函数名字(数据类型 形参1,数据类型 形参2,...,数据类型 形参N)
  {
    局部变量定义;
    表达式语句;
  }
```

函数
--函数声明

【例】返回值的函数声明例子

```
int max(int x,int y)
{
    if(x>y) return x;
    else   return y;
}
```

在该例子中，x和y是函数max的两个形式化参数，其类型是int。该函数通过return返回值，其类型为函数max的函数类型int。

函数
--函数声明

【例】不返回值的函数声明例子

```
void max(int x,int y)
{
  if(x>y)
        printf("%d > %d\n",x,y);
 else
        printf("%d < %d\n",x,y);
}
```

在该例子中，x和y是函数max的两个形式化参数，其类型是int。该函数只打印信息，并不返回值。

函数
--函数调用

[变量][=]被调用的函数名字(实际参数1,实际参数2,....实际参数N)

【例】返回值的函数调用例子

```
d=max(a,b);
```

【例】不返回值的函数调用例子

```
max(a,b);
```

函数
--函数变量的存储方式

在C语言中，提供了多种变量的存储方式。

- 按照变量的作用范围分为局部变量和全局变量。
- 按照变量的存储方式，可以分为：
  - 自动变量(auto)
  - 外部变量(extern)
  - 静态变量(static)
  - 寄存器变量(register)

函数变量的存储方式
--自动变量

在声明变量时，如果没有指定其存储类型，则默认为auto。

在C语言中，这是一类用的最广泛的变量。

- 自动变量的作用范围在定义它的函数体或者复合语句的内部，只有在定义它的函数被调用，或者定义它的复合语句被执行时，编译器才为其在单片机内分配存储空间。
  - 当函数调用结束或者执行完复合语句后，将释放为变量所分配的存储空间变量的值因此也就不再存在。
  - 当再次调用函数或者执行复合语句时，会为变量重新分配单片机内的存储空间，但不会保留上次运行时的值，而且必须重新赋值。
- 因此，自动变量可以称为是局部变量。

函数变量的存储方式
--外部变量

使用存储类型说明符“extern”定义的变量称为外部变量。

- 默认地，凡是在所有函数之前，在函数外部定义的变量都是外部变量，定义时可以不写extern说明符。但是，如果在一个函数体内说明一个已经在函数体外或者别的程序模块文件中定义过的外部变量时，则必须使用extern说明符。
  - 一旦定义了外部变量，则就为该变量固定分配单片机内的存储空间。
  - 在程序运行的整个过程中，外部变量均有效，其值均被保存。

函数变量的存储方式
--外部变量

- 函数可以互相调用，因此函数都具有外部存储种类的属性。
  - 定义函数时如果用关键字extern，则将该函数明确定义为一个外部函数。
  - 如果定义函数时，省略了关键字extern，则隐含为外部函数。
  - 如果要调用当前程序模块文件以外的其它模块文件所定义的函数，则必须用关键字extern说明被调用的函数是一个外部函数。

函数变量的存储方式
--静态变量

使用存储类型说明符“static”定义的变量称为静态变量。

- **静态变量不像自动变量那样只有在函数调用它的时候才存在，退出函数之后它就消失，局部静态变量始终存在。**
- **但是只能在定义它的函数内部进行访问。**
- **退出函数值之后，静态变量以前的值仍然被保留，但是不能访问它。**

函数变量的存储方式
--静态变量

还有一种全局静态变量，它是在函数外部进行定义，作用范围从它的定义点开始，一直到程序结束。

- **当一个C语言文件由若干模块文件构成时，全局静态变量始终存在。**
  - **但它只能在被定义的文件中访问，其数据值可以为该文件内的所有函数共享。**
  - **退出该文件后，虽然变量值仍然保留，但是不能被其它模块文件访问。**
- **局部静态变量是一种在两次函数调用之间仍能保留其值的局部变量。**

函数变量的存储方式
--寄存器变量

为了提高程序执行的效率，在C语言中将一些使用频率很高的变量定义为能够直接使用硬件寄存器的所谓寄存器变量。

- 在定义变量时，在前面用register说明符，表示该变量是寄存器类型的。
- 寄存器变量是自动变量的一种，它的有效范围同自动变量一样。

【例】变量存储模式的例子

函数
--函数变量的存储方式

```
int i=0;
int cal(int x)
{       static int y=0;
         y=x-y-i;
        return y;
}
void main()
{
      int j=1000;
      int k;
      i=cal(j);
      i=cal(j);
      i=cal(j);
}
```

函数
--函数变量的存储方式

注：读者可以进入到本书所提供资料的STC_example\例子6-45目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按断点运行程序。

分析程序的执行过程：

- 当执行第14行代码时，j=1000，调用子函数y=0，i=0，x=j，因此，y=x-y-i=1000。由于y是静态变量，此时y的值变成1000。在执行完该子函数，返回值赋值给i后，由于i是全局变量所以i的值变成1000。

```
5    static int y=0;
6      y=x-y-i;
7    return y;
8  }
9
10 void main()
11 {
12   int j=1000;
13   int k;
14   i=cal(j);
15   i=cal(j);
16
17 }
```

函数
--函数变量的存储方式

- 当执行第15行代码时，j=1000，调用子函数y=1000，i=1000，x=j，因此，y=x-y-i=-1000。由于y是静态变量，此时y的值变成-1000。在执行完该子函数，返回值赋值给i后，由于i是全局变量，所以i的值变成-1000。
- 当执行第16行代码时，j=1000，调用子函数y=-1000，i=-1000，x=j，因此，y=x-y-i=3000。由于y是静态变量，此时y的值变成3000。在执行完该子函数，返回值赋值给i后，由于i是全局变量，所以i的值变成3000。

函数
--函数参数和局部变量的存储器模式

在Keil C51编译器中，允许采用三种存储模式：small、compact和large。

- 一个函数的存储器模式确定了函数参数和局部变量在内存中的地址空间。当：
  - 在small模式下，函数参数和局部变量位于单片机片内数据RAM中；
  - 在compact和large模式下，函数参数和局部变量位于单片机的扩展数据RAM中。
- 定义函数参数和局部变量的存储器模式格式为：

函数类型 函数名(形参列表) [存储器模式]

注：存储器模式的声明符:small、compact或large。

函数
--基本数据类型传递参数

本节将通过递归函数和递归调用说明基本数据类型传递参数的方法。

- 所谓的递归调用是指在调用递归函数的过程中间接或者直接地调用函数本身。

函数
--基本数据类型传递参数

典型的计算阶乘函数：

$$f(n)=n!$$

- 可以先计算f(n-1), f(n)=n×f(n-1)。
- 而f(n-1)又可以通过计算f(n-2)得到，即：f(n-1)=(n-1)×f(n-2)，以此类推直到得到f(1)。
- 这样通过f(1)，就可以得到f(2)，....，一直到f(n)。

函数
--基本数据类型传递参数

在单片机/计算机中，递归是通过入栈的过程和出栈的过程实现，这个过程将通过下面的例子进行说明。

- 在Keil Cx51编译器中，对于递归函数使用关键字reentrant标识，该关键字意思为“重入”，即表示在调用该函数的时候，可以自己调用自己。说明格式为：

  函数类型 函数名(形参列表) [reentrant]

函数
--基本数据类型传递参数

【例】递归函数计算阶乘n!的例子

```
#include "stdio.h"
#include "reg51.h"
int fac(int n) reentrant                          //reentrant声明为递归函数
{
        long int f;
        if(n<0)                                   //如果小于0，则打印错误信息
        printf("data must be larger than 0\n");
        else if(n<1)                              //如果n=0，则返回值为1
        f=1;
        else                                      //如果n>1，则递归调用自己
        f=fac(n-1)*n;
        return f;
}
```

函数
--基本数据类型传递参数

```
main()
{
        int n;
        long int y;
        SCON= 0x52;
        TMOD = 0x20;
        TCON = 0x69;
        TH1 = 0xF3;
        printf("please input an integer number\n");
        scanf("%d",&n);                          //输入n值
        y=fac(n);                               //mian函数调用fac函数
        printf("%d!=%ld\n",n,y);                 //打印结果
  while(1);
}
```

函数
--基本数据类型传递参数

注：读者可以进入到本书所提供资料的STC_example\例子6-46目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式。

按下面步骤运行和分析递归函数的调用过程：

- 在代码的第12行、第13行和第26行分别设置断点。
- 打开UART 1窗口界面。按F5按键，在UART 1窗口内出现提示信息“please input an integer number”，然后输入4，表示该程序将计算4！。

函数
--基本数据类型传递参数

- 按F5键，跳到断点26行代码。在当前调试模式主界面右下侧的Call Stack + Locals标签窗口下，可以看到主程序的信息，如图所示。该窗口说明，当前断点在Main主程序内，断点所调用程序的代码保存在单片机片内存储区的0x085B的位置。

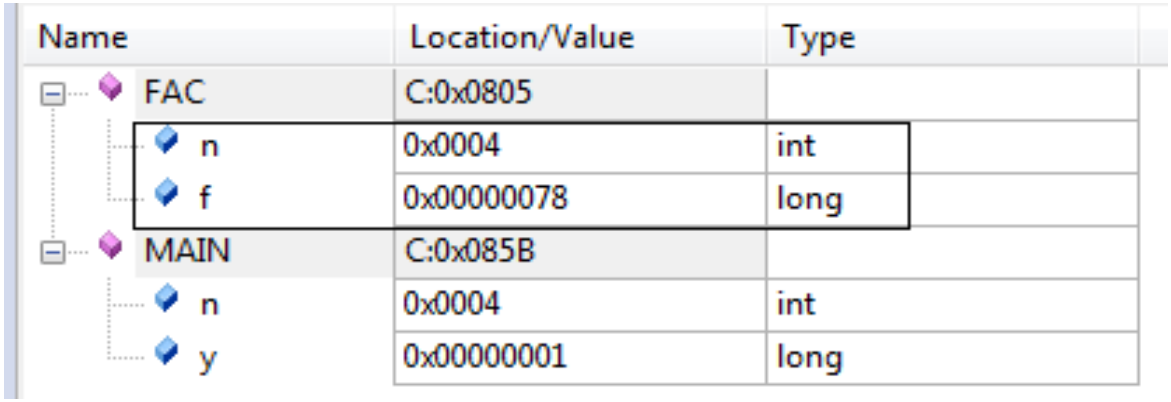
Call Stack + Locals

Name	Location/Value	Type
MAIN	C:0x085B	
n	0x0004	int
y	0x00000001	long

堆栈调用窗口信息(1)

函数
--基本数据类型传递参数

- 按F5键，跳到断点12行代码。
- 在当前调试模式主界面右下侧的Call Stack + Locals标签窗口下，可以看到调用函数FAC的信息，如图所示。在反汇编窗口可以看到fac函数的程序入口点在c:0x0805的位置。



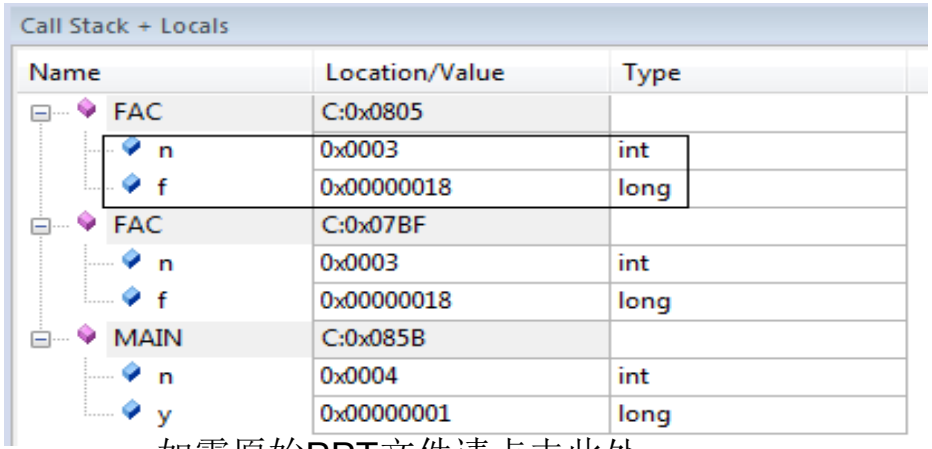
Name	Location/Value	Type
FAC	C:0x0805	
n	0x0004	int
f	0x00000078	long
MAIN	C:0x085B	
n	0x0004	int
y	0x00000001	long

堆栈调用窗口信息(2)

函数
--基本数据类型传递参数

- 当前断点程序代码的入口在单片机片内程序Flash空间的0x0805。表示继续调用fac函数。此时，n=4；由于fac要调用自己，所以fac函数当前运行的状态保存在堆栈中。

- **按F5键，再次跳到断点12行代码。**
  - 在Call Stack + Locals标签窗口下可以看到第二次被调用函数fac的信息，如图所示。



堆栈调用窗口信息(3)

函数
--基本数据类型传递参数

- 当前断点程序代码的入口在单片机片内程序Flash空间的0x0805。表示继续调用fac函数。此时，n=3；由于fac要调用自己，所以fac函数当前运行的状态保存在堆栈中，很明显上一次调用函数的程序入口点在单片机片内程序Flash区的0x07BF。

- **按F5键，再次跳到断点12行代码。**
  - 在Call Stack + Locals标签窗口下，可以看到第三次被调用函数fac的信息，如下图所示。

函数
--基本数据类型传递参数

Call Stack + Locals

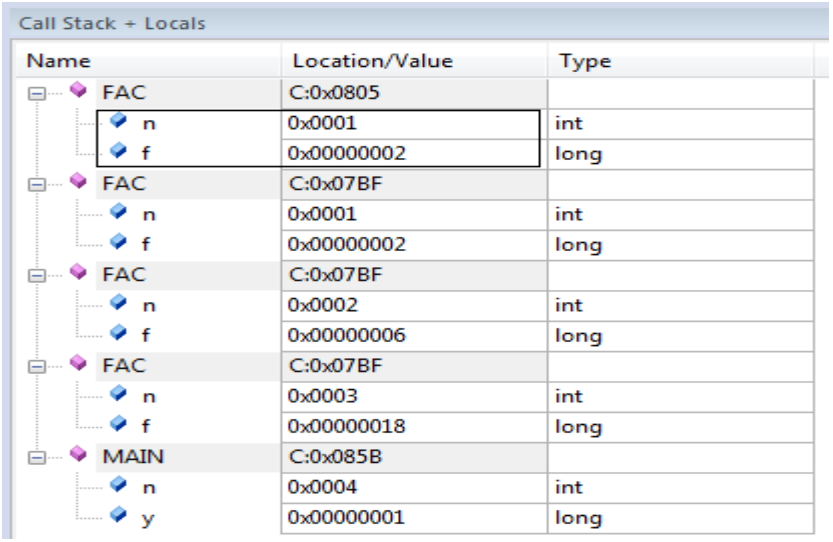
Name	Location/Value	Type
FAC	C:0x0805	
n	0x0002	int
f	0x00000006	long
FAC	C:0x07BF	
n	0x0002	int
f	0x00000006	long
FAC	C:0x07BF	
n	0x0003	int
f	0x00000018	long
MAIN	C:0x085B	
n	0x0004	int
y	0x00000001	long

堆栈调用窗口信息(4)

- 当前断点程序代码的入口在单片机片内程序Flash空间的0x0805。表示继续调用fac函数。此时，n=2,fac函数当前运行的状态保存在堆栈中，很明显上一次调用函数的程序入口点在单片机片内程序Flash区的0x07BF。

函数
--基本数据类型传递参数

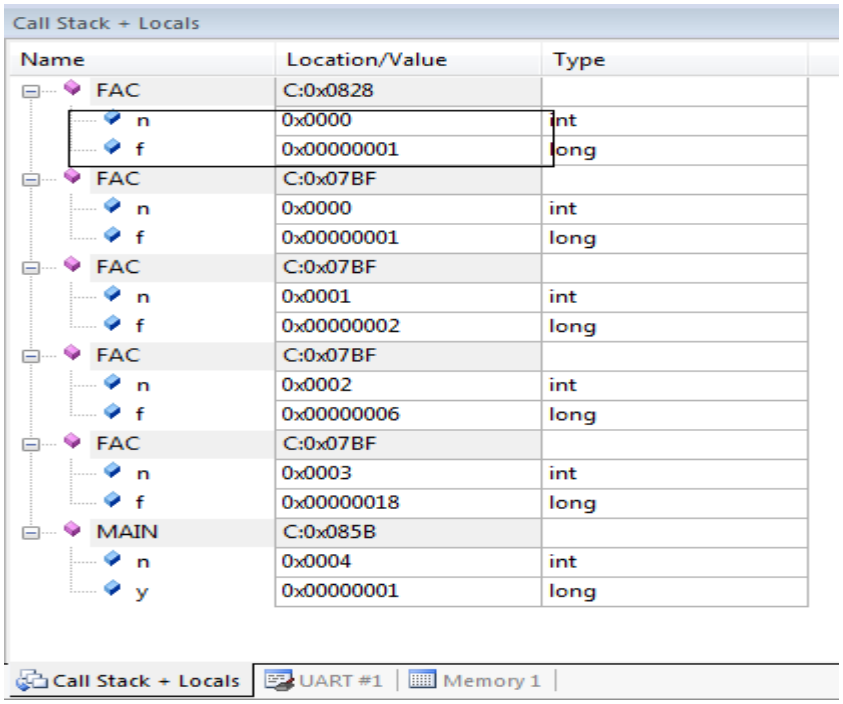
- 按F5键，再次跳到断点12行代码。
  - 在Call Stack + Locals标签窗口下，可以看到第四次被调用函数fac的信息，如图所示。



  - 当前断点程序代码的入口在单片机片内程序Flash空间的0x0805。表示继续调用fac函数。此时，n=1，所以fac函数当前运行的状态保存在堆栈中，很明显上一次调用函数的程序入口点在单片机片内程序Flash区的0x07BF。

函数
--基本数据类型传递参数

- 按F5键，跳到断点13行代码。
  - 在Call Stack + Locals标签窗口下，可以看到被调用函数fac的信息，如图所示。



函数
--基本数据类型传递参数

- 从图中可以看到当前断点程序代码的入口在单片机片内程序Flash空间的0x0828。前面保存的函数的信息从堆栈中消失，也就是出栈。表示递归的过程结束，将要陆续返回递归的结果。此时，n=1；

- **按F5键，跳到断点13行代码。**
  - 在当前调试模式主界面右下侧的Call Stack + Locals标签窗口下，可以看到被调用函数fac的信息。
  - 从图中可以看到当前断点程序代码的入口在单片机片内程序Flash空间的0x0828。前面保存的函数的信息从堆栈中消失，也就是出栈。表示递归的过程结束，将要陆续返回递归的结果。此时，n=2；

函数
--基本数据类型传递参数

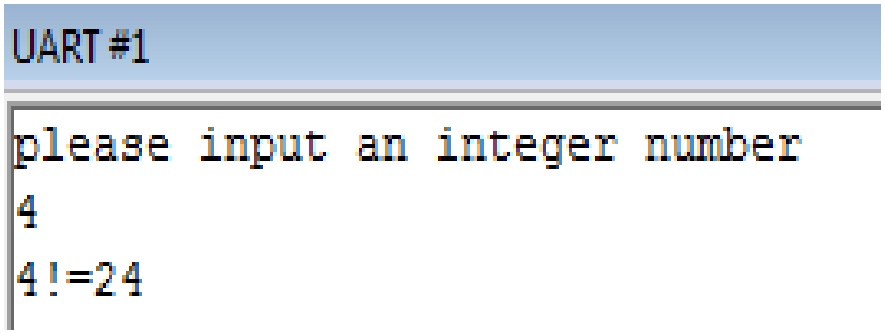
- 按F5键，跳到断点13行代码。
  - 在当前调试模式主界面右下侧的Call Stack + Locals标签窗口下，可以看到被调用函数fac的信息。
  - 从图中可以看到当前断点程序代码的入口在单片机片内程序Flash空间的0x0828。前面保存的函数的信息从堆栈中消失，也就是出栈。表示递归的过程结束，将要陆续返回递归的结果。此时，n=4，fac函数的调用结束，如图所示。

Call Stack + Locals

Name	Location/Value	Type
FAC	C:0x0828	
n	0x0004	int
f	0x00000018	long
MAIN	C:0x085B	
n	0x0004	int
y	0x00000001	long

函数
--基本数据类型传递参数

- 在UART #1窗口下，可以看到最后打印的结果，如图所示。



UART #1窗口打印信息

函数
--数组类型传递参数

- 在C语言中，数组元素可以作为函数实参传递参数，其用法和变量相同。
- 数组名也可以作为实参和形参，此时传递的是元素的首地址。

函数
--数组类型传递参数

【例】数组名字传递参数的例子

在该程序中，调用子函数实现对数组元素的升序排列。

```
#include "stdio.h"
#include "reg51.h"
void sort(int array[],int n)                    //声明排序子函数，不返回值
{       int i,j,k,t;
        for(i=0;i<n-1;i++)                      //二重循环排序
          {       k=i;
                  for(j=k+1;j<n;j++)
                  {       if(array[j]<array[k])
                                  k=j;
                          t=array[k];
                          array[k]=array[i];
                          array[i]=t;
                   }
          }
}
```

函数
--数组类型传递参数

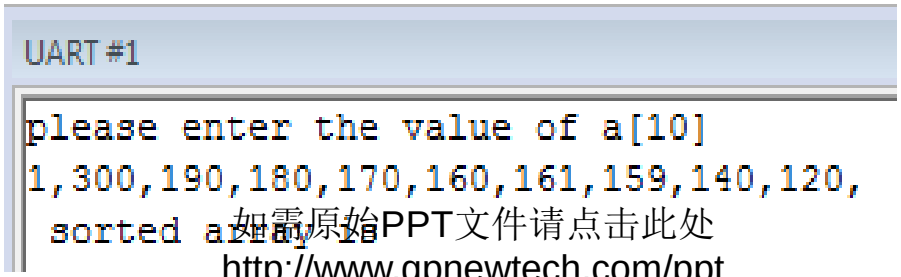
```
void main()
{
        int a[10],i;
        SCON= 0x52;
        TMOD = 0x20;
        TCON = 0x69;
        TH1 = 0xF3;
        printf("please enter the value of a[10]\n");
        for(i=0;i<10;i++)
            scanf("%d,",&a[i]);                    //输入数组元素
        printf("\n sorted array is\n");
        sort(a,10);                                //调用排序函数，数组名传递
        for(i=0;i<10;i++)
            printf("a[%d]=%d,",i,a[i]);            //打印排序后的结果
        while(1);
}
```

函数
--数组类型传递参数

注：读者可以进入到本书所提供资料的STC_example\例子6-47目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式。

按下面步骤运行分析数组类型传递参数的过程：

- 在代码的第4行、第7行、第17行和第30行分别设置断点。
- 按F5按键，运行程序。
- 打开UART #1窗口。在该窗口界面内，出现提示信息”please enter the value of a[10]”,然后，在该窗口内输入10个数据，每个数据之间用“,”隔开，如图所示。



函数
--数组类型传递参数

- 打开Watch 1窗口，输入a。更进一步的，在Memory 1窗口中，输入d:0x22，可以看到所保存数组a的数据元素信息。

Watch 1

Name	Value	Type
a	D:0x22	array[10] of int
[0]	1	int
[1]	300	int
[2]	190	int
[3]	180	int
[4]	170	int
[5]	160	int
[6]	161	int
[7]	159	int
[8]	140	int
[9]	120	int

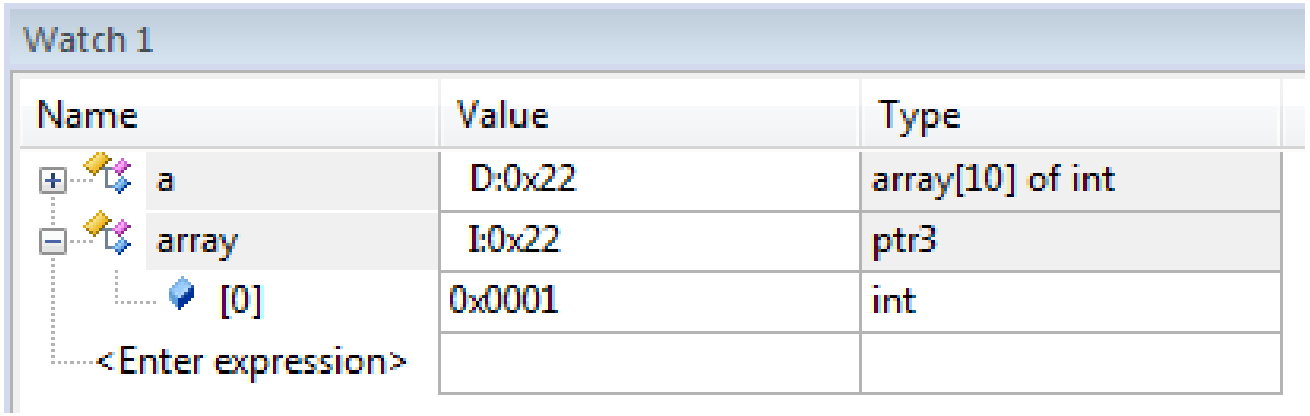
Memory 1

Address: d:0x22

```
D:0x22: 00 01 01 2C 00 BE 00 B4 00 AA 00 A0 00 A1 00 9F 00 8C 00 78 00
D:0x3F: 00 00 00 00 00 00 00 00 00 00 FF 00 12 00 01 00 01 00 04 28 09
```

函数
--数组类型传递参数

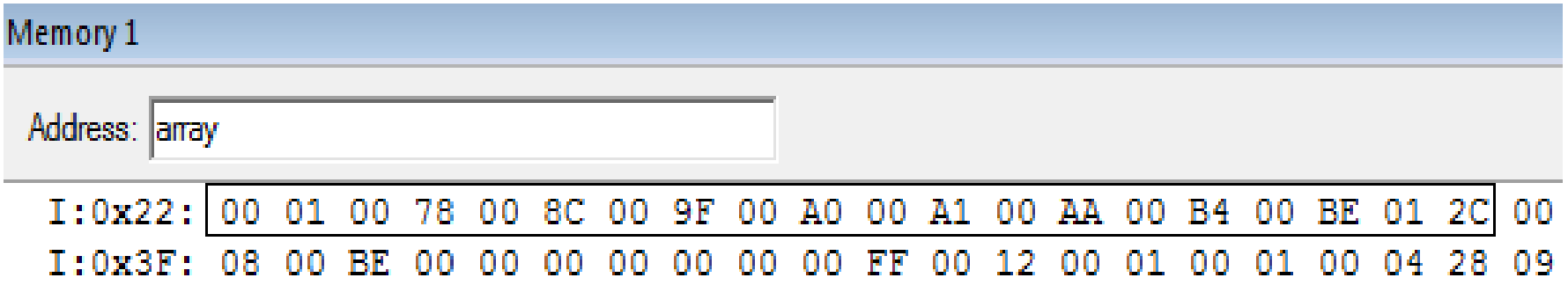
- 按F5按键，继续运行程序，程序停到了第7行代码的位置。在Watch 1窗口中，输入array。从图中可以看到形参数组array在单片机片内数据区0x22的起始地址。



Wach 1窗口的array信息

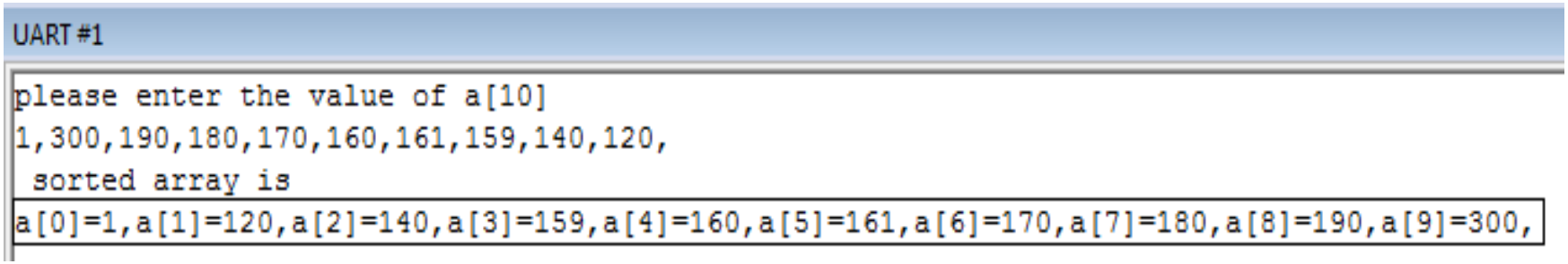
函数
--数组类型传递参数

- 按F5按键，运行程序到第17行代码。此时，再观察Memory 1窗口内的内容，如图所示。读者会发现这个地址空间的内容发生了变化。



函数
--数组类型传递参数

- 按F5按键，运行程序。由于数组a一直指向单片机片内地址为0x22的起始位置，而由于子函数操作修改了单片机片内地址为0x22开始连续20个字节的内容。因此，打印出来的数据，就是修改后0x22地址开始的存储空间的内容，如图所示。



函数
-指针类型传递参数

当函数的参数是指针类型的变量时，主调函数将实际参数的地址作为被调函数中形式参数的地址。

- 因此，指针类型传递参数也是通过地址传递。
- 下面将通过一个例子来说明指针类型传递参数的过程。

函数
-指针类型传递参数

【例】两个字符数组首尾连接的例子

比如：字符串a=”STC”，字符串b=“Hello”，将两个字符串连接后的字符串c=“STC Hello”。

```
#include "stdio.h"
#include "reg51.h"
void con_string(char *s1, char *s2 ) //声明子函数，有两个字符指针参数
{       while(*s1!= ‘\0’ )          //如果指针*s1指向的字符不是结束，则继续
                s1++;               //指针递增
        while(*s2!= ‘\0’ )          //如果指针*s2指向的字符不是结束，则继续
                *s1++=*s2++;        //将指针*s2指向的内容赋值到*s1的末尾
        *s1= ‘\0’ ;                 //在*s1当前指向的内容后添加结束标志
}
```

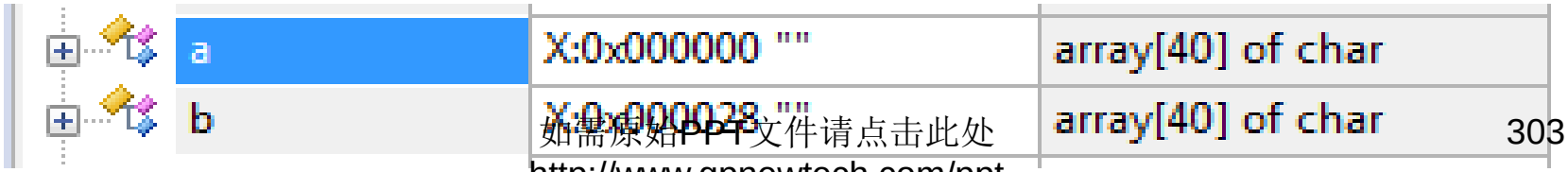
函数
-指针类型传递参数

```
void main()
{    xdata char  a[40],b[40];                    //在单片机xdata区域内声明字符数组a和b
     SCON= 0x52;
     TMOD = 0x20;
     TCON = 0x69;
     TH1 = 0xF3;
     printf("please enter the string of a[40]\n");   //提示输入字符串a
     gets(a,40);                                     //输入字符串a
     printf("please enter the string of b[40]\n");  //提示输入字符串b
     gets(b,40);                                     //输入字符串b
     printf(“\n connected the string is\n”);         //提示连接后的字符串信息
     con_string(a,b);                                //调用连接字符串函数
     puts(a);                                        //打印字符串a
     while(1);
}
```

函数
-指针类型传递参数

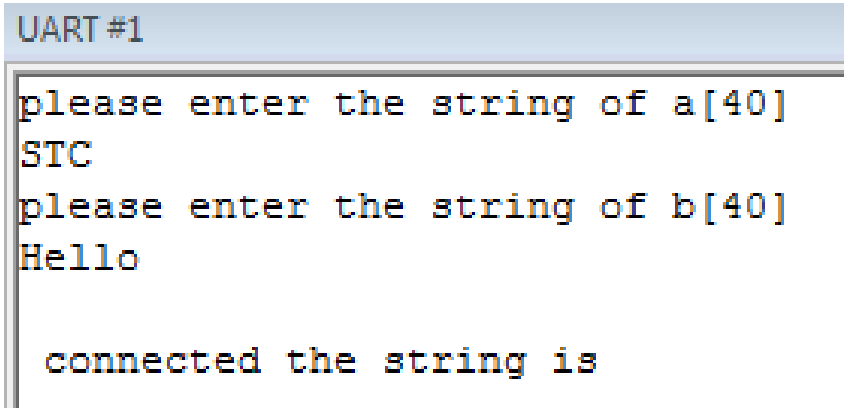
将上述代码编辑后进入调试器模式，按下面步骤运行分析指针传递参数调用过程：

- 在代码的第4行、第10行、第21行和第26行分别设置断点。
- 按F5按键，运行程序。
- 运行到第21行代码。在Watch 1窗口界面内，输入a和b，看到相关信息。数组a的起始地址位于单片机扩展数据RAM地址为0x000000的位置，数组b的起始地址位于单片机扩展数据RAM地址为0x000028的位置。展开后其内部用二进制数0填充。



函数
-指针类型传递参数

- 打开UART #1窗口。在该窗口界面内，出现提示信息”please enter the string of a[40]”。
- 在下面一行输入字符串“STC”，然后按回车键。
- 出现提示信息“please enter the string of b[40]。
- 在下面一行输入字符串“Hello”，按回车键。

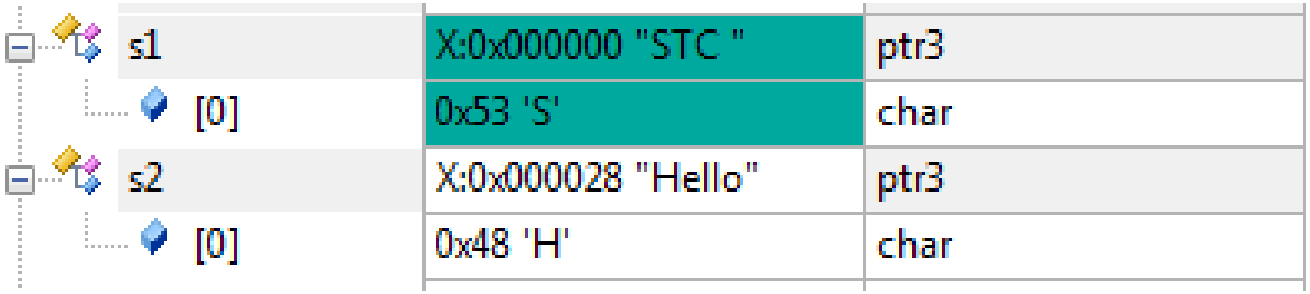


函数
-指针类型传递参数

- 按F5按键，运行程序。程序运行到第26行。
  - 从反汇编程序窗口可以看到该行代码在单片机片内程序Flash存储空间地址为0x0551的位置。在该行汇编结束的时候有LCALL调用con_string(c:056E)，可以看到被调用函数的起始地址在单片机片内程序Flash存储空间地址为0x0560的位置。
- 按F5按键，运行程序。断点停在第4行代码。
  - 从Disassembly窗口中可以看到con_string子函数的起始地址位于单片机片内程序Flash存储空间地址为0x56E的位置。

函数
-指针类型传递参数

- 单步运行一条代码。在Watch 1窗口界面内输入s1和s2，显示相关信息。s1地址为x:0x000000，s2地址为x:0x000028。也就是s1指向数组a，s2指向数组b。

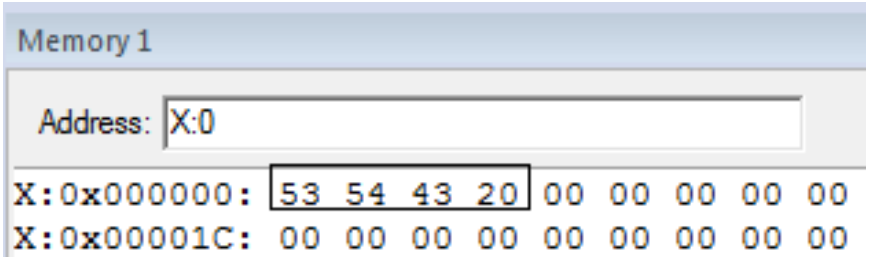


s1	X:0x000000 "STC "	ptr3
[0]	0x53 'S'	char
s2	X:0x000028 "Hello"	ptr3
[0]	0x48 'H'	char

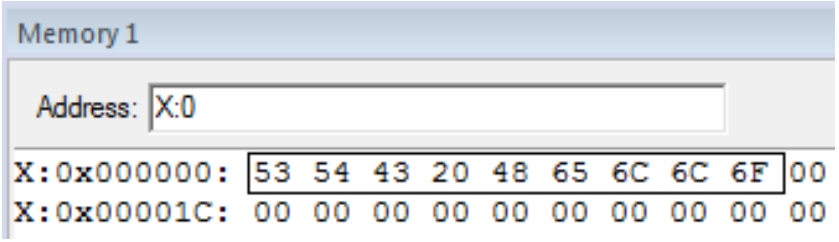
Watch 1窗口界面

函数
-指针类型传递参数

- 单步运行程序代码到第10行。注意观察地址为x:0x000000起始位置的内容改变，原来的内容如图(a)所示，修改后的内容如图(b)所示。



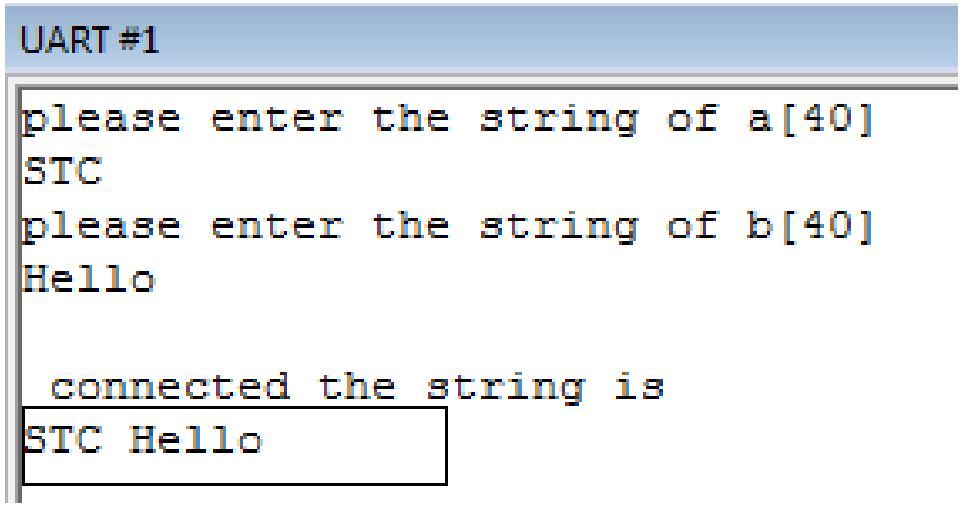
（a）修改前存储器空间的内容



（b）修改后存储器空间的内容

函数
-指针类型传递参数

- 按F5按键运行程序。在UART #1窗口内打印出拼接后的字符串“STC Hello”信息，如图所示。



UART #1窗口内打印的信息

预编译指令

在C语言中，提供了对程序的编译预处理功能。

- 通过一些预处理命令，为C语言本身提供许多功能和符号等方面的扩展。因此，增加了C语言的灵活性和方便性。
- 在编写C语言程序时，可以将预处理命令添加到需要的位置，但它只在编译程序时起作用，且通常是按行进行处理，因此又称为编译控制行。

预编译指令
--宏定义

宏定义的命令为#define

- **它的作用是用一个字符串进行替换，这个字符串既可以是常数，也可以是其他任何字符串，甚至还可以是带参数的宏。**
- **宏定义的简单形式是符号常量定义，复杂形式是带参数的宏定义。**

预编译指令
--宏定义

不带参数的宏定义

- 不带参数的空定义格式为：

```
#define 标识符 常量表达式
```

【例】符号常量宏定义的例子

```
#define  PI 3.1415926
#define  R 3.0
#define  L 2*PI*R
#define  S PI*R*R
```

预编译指令
--宏定义

带参数的宏定义

- 带参数的宏定义与符号常量定义的不同之处在于，对于源程序中出现的宏符号名不仅进行字符串替换，而且还能进行参数替换。
- 带参数的宏定义格式为：

#define 宏符号名(参数表) 表达式

其中：

参数表中的参数是形参，在程序中用实际参数进行替换。

预编译指令
--宏定义

【例】带参数的宏定义例子

```
#define MAX(x,y)   (((x)>(y)) ? (x) : (y))
#define SQ(x) (x*x)
#define S(r) PI*r*r
```

【例】宏定义例子

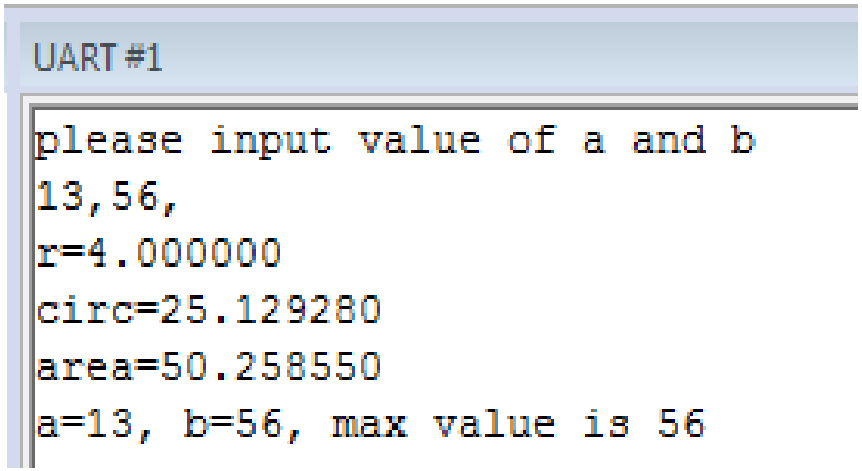
```
#include "stdio.h"
#include "reg51.h"
#define PI 3.14115926
#define CIRCLE(R,L,S) L=2*PI*R; S=PI*(R)*(R)
#define MAX(x,y)   (((x)>(y)) ? (x) : (y))
```

预编译指令
--宏定义

```
void main()
{
        float r,l,s;
        int a,b;
        SCON= 0x52;
        TMOD = 0x20;
        TCON = 0x69;
        TH1 = 0xF3;
        printf("please input r:\n");
        scanf("%f",&r);
        printf("please input value of a and b\n");
        scanf("%d,%d",&a,&b);
        CIRCLE(r,l,s);
        printf("\nr=%f\ncirc=%f\narea=%f\n",r,l,s);
        printf("a=%d, b=%d, max value is %d\n",a,b,MAX(a,b));
        while(1);
}
```

预编译指令
--宏定义

注：读者可以进入到本书所提供资料的STC_example\例子6-51目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5按键。打开UART #1窗口界面，输入并显示信息，如图所示。



UART #1窗口内打印的信息

预编译指令
--文件包含

文件包含是指一个程序文件将另一个指定的文件的全部内容包含进来。

- 在前面已经多次使用了#include “stdio.h”和#include “reg51.h”包含头文件命令。文件包含命令的格式为：

  #include 文件名

预编译指令
--文件包含

【例】文件包含的例子

在该例子中，定义了max.c文件和max.h文件，通过文件包含将这两个文件整合到设计中，包含文件的步骤包括：

- 在当前工程下，建立一个名字为top.uvproj的工程；
- 新建一个名字为max.c的文件，并在该文件中添加如下设计代码。

预编译指令
--文件包含

```
int max(int x,int y)              //两个数中，求取最大值
{
        if (x>y) return x;
        else return y;
}
int min(int x,int y)              //两个数中，求取最小值
{
        if (x<y) return x;
        else return y;
}
float avg(int x,int y)            //求取两个数的平均值
{
        return((x+y)/2.0);
}
```

预编译指令
--文件包含

- 保存该文件
- 新建一个名字为max.h的文件，并在该文件中添加如下设计代码。

```
int i=11,j=100;                 //定义两个全局整型变量
int max(int x,int y);           //声明max函数类型
int min(int x,int y);           //声明min函数类型
float avg(int x,int y);         //声明avg函数类型
```

- 保存该文件。
- 新建一个名字为main.c的文件，并在该文件中添加如下设计代码。

预编译指令
--文件包含

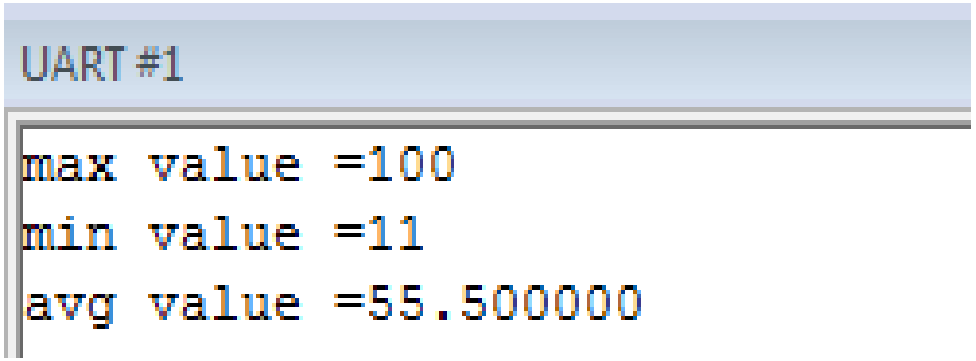
```
#include "stdio.h"
#include "reg51.h"
#include "max.h"                    //包含自定义头文件
void main()
{
        int k,l;
        float m;
        SCON= 0x52;
        TMOD = 0x20;
        TCON = 0x69;
        TH1 = 0xF3;
```

预编译指令
--文件包含

```
    k=max(i,j);                          //调用max函数
    l=min(i,j);                          //调用min函数
    m=avg(i,j);                          //调用avg函数
    printf("max value =%d\n",k);   //打印值k
    printf("min value =%d\n",l);    //打印值l
    printf("avg value =%f\n",m);   //打印值m
}
```

预编译指令
--文件包含

- 保存该文件，并进入调试器模式，按F5按键。打开UART #1窗口界面，显示如下信息，如下图所示。



UART #1窗口内打印的信息

预编译指令
--条件编译

一般情况下，希望对所有的程序行进行编译。但是有时希望只对其中的一部分内容在满足一定的条件时才进行编译，这就是条件编译。

- Keil Cx51编译器的预处理器提供了三种条件编译格式：

预编译指令
--条件编译

- 条件编译命令格式一

```
#ifdef 标识符
    程序段1
#else
    程序段2
#endif
```

预编译指令
--条件编译

条件编译命令格式二

```
#ifndef 标识符
    程序段1
#else
    程序段2
#endif
```

预编译指令
--条件编译

条件编译命令格式三

```
#if 常量表达式1
   程序段1
#elif 常量表达式2
   程序段2
.......
#else
    程序段n
#endif
```

预编译指令
--条件编译

【例】条件编译的例子

```
#include "stdio.h"
#include "reg51.h"
void main()
{
    SCON= 0x52;
    TMOD = 0x20;
    TCON = 0x69;
    TH1 = 0xF3;
    #ifdef SYMBOL
    printf("define SYMBOL in file\n");
    #else
      printf("not define SYMBOL in file\n");
    #endif
    while(1);
}
```

预编译指令
--条件编译

注：读者可以进入到本书所提供资料的STC_example\例子6-53目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5按键，打开UART #1窗口界面，显示信息“not define SYMBOL in file”。

下面给出定义SYMBOL的两种方法：

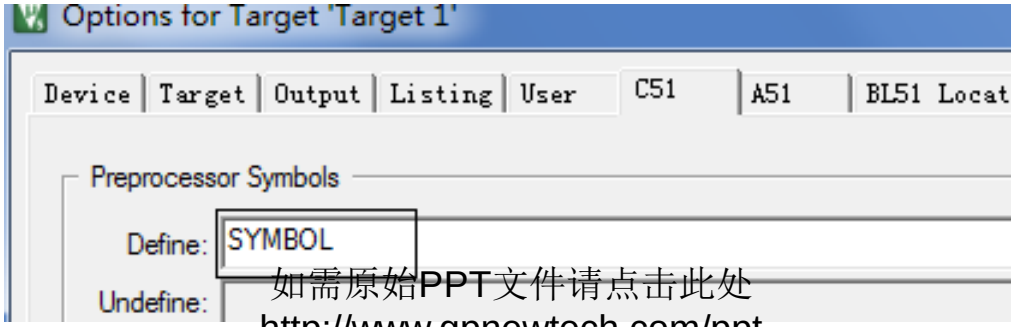
- 在main.c文件中，添加下面一行代码。
  - 保存并编译文件。然后，进入调试器模式，按F5按键。打开UART #1窗口界面，显示信息“define SYMBOL in file”。

main.c

```
1  #include "stdio.h"
2  #include "reg51.h"
3  #define SYMBOL
4  void main()
5  {
```

预编译指令
--条件编译

- 在当前工程主界面左侧的Project窗口内，选择Target 1，单击右键出现浮动菜单，选择Options for Target ‘Target 1’….。出现Options for Target ‘Target 1’对话框界面。
  - 在该界面内选择C51标签。在该标签窗口下，找到Preprocessor Symbols（预处理器符号）标题栏。在该标题栏下的Define：右侧输入SYMBOL。
  - 单击OK按钮。



预编译指令
--条件编译

重新编译设计。然后，进入调试器模式，按F5按键。打开UART #1窗口界面，显示信息“define SYMBOL in file”。

预编译指令
--其它预处理命令

Keil Cx51编译器还支持#error、#pragma和#line预处理命令。

本节介绍#error和#pragma命令。

预编译指令
--其它预处理命令

#error

- 该命令通常用于条件编译中，用于捕获一些不可预知的编译条件。
- 正常情况下该条件的值为假，如果条件为真，则输出一条由#error命令后面字符串给出的错误信息并停止编译。

预编译指令
--其它预处理命令

- 比如：在前面的代码中插入：

```
#else
  printf("not define SYMBOL in file\n");
#error stop!!!
#endif
```

表示当没有定义符号时，报错并停止编译。

预编译指令
--其它预处理命令

#pragma

该命令用于在源程序中向编译器传送各种编译控制命令，格式为：

#pragma 编译命令序列

该命令可以控制编译器对程序处理的方法。

复杂数据结构

在C语言中，除了提供基本数据类型、数组和指针外，还提供了复杂的数据结构，用于将不同类型的数据放在一起。复杂数据结构包括：结构、联合和枚举。

复杂数据结构
--结构

结构是将不同数据类型有序组合在一起而构成的一种数据的集合体。

- 结构中的每个数据类型分别占用所声明类型的存储空间。

复杂数据结构
--结构

结构类型的定义

```
struct  结构名
{
    结构元素列表
 }
```

其中：

- 结构元素列表为不同数据类型的列表。

复杂数据结构
--结构

【例】结构体的声明例子

```
struct student
{
    char name[30];
    char gender;
    char age;
    long int num;
};
```

复杂数据结构
--结构

结构变量的定义

- 在声明的时候定义

【例】结构体的声明例子

```
struct student
{
    char name[30];
    char gender;
    char age;
    long int;
 }stu1,stu2;
```

复杂数据结构
--结构

- 在声明后单独定义

  格式为：

  struct 结构名 结构变量1,结构变量2,...,结构变量N

注：在实际使用的时候，如果变量很多，可以将这些变量整合到一个数组内，这样更加方便操作。

【例】结构体的声明例子2

```
struct student stu1,stu2;
```

复杂数据结构
--结构

结构变量内元素的引用

- 当定义完结构变量后，就可以引用结构变量内的元素。格式：

  结构变量名.结构元素

复杂数据结构
--结构

【例】结构体使用的例子

```
#include "stdio.h"
#include "reg51.h"
struct student{                          //定义结构体
        char name[30];                   //字符类型数组
        char gender;                     //字符类型数据
        int age;                         //整型数据
        long int num;                    //长整型数据
      };
xdata struct student stu[2];             //xdata区定义结构数组变量
```

复杂数据结构
--结构

```
void main()
{
 int i;
SCON= 0x52;
  TMOD = 0x20;
  TCON = 0x69;
  TH1 = 0xF3;
  for(i=0;i<2;i++)                    //循环输入结构数组变量元素
 {
 printf("please input stu[%d].name\n",i);
 scanf(“%s”,stu[i].name);            //输入结构中的name元素
 getchar();
```

复杂数据结构
--结构

```
printf("please input stu[%d].gender\n",i);
scanf(“%c”,&stu[i].gender);                    //输入结构中的gender元素
putchar('\n');
printf("please input stu[%d].age\n",i);
 scanf(“%d”,&stu[i].age);                      //输入结构中的age元素
printf("please input stu[%d].num\n",i);
scanf(“%ld”,&stu[i].num);                      //输入结构中的num元素
}
putchar('\n');
```

复杂数据结构
--结构

```
for(i=0;i<2;i++)
 {
    printf("the following students information is:\n");
    printf(“stu[%d].name=%s,” ,i,stu[i].name);    //输出结构中的name元素
    printf("stu[%d].gender=%c, ",i,stu[i].gender); //输出结构中的gender元素
    printf(“std[%d].age=%d,” ,i,stu[i].age);        //输出结构中的age元素
    printf(“std[%d].num=%ld,” ,i,stu[i].num);     //输出结构中的num元素
    putchar('\n');
    }
while(1);
}
```

复杂数据结构
--结构

注：读者可以进入到本书所提供资料的STC_example\例子6-57目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5按F5按键。打开UART #1窗口，在该界面下按照提示信息输入结构元素的值，最后打印输入的信息。

UART #1

```
please input stu[0].name
hebin
please input stu[0].gender
m
please input stu[0].age
39
please input stu[0].num
20145
please input stu[1].name
libaolong
please input stu[1].gender
m
please input stu[1].age
24
please input stu[1].num
20147

the following students information is:
stu[0].name=hebin, stu[0].gender=m, std[0].age=39, std[0].num=20145,
the following students information is:
stu[1].name=libaolong, stu[1].gender=m, std[1].age=24, std[1].num=20147,
```

复杂数据结构
--结构

为了进一步的理解结构体的概念，给出Watch 1窗口内的信息。

- 图中可以看出，没有为student结构本身分配存储空间；
- 但是为结构变量stu[0]和stu[1]分配了空间。其中：
  - 将结构变量stu[0]分配到单片机扩展数据区起始地址为0x000000的地方；
  - 将结构变量stu[1]分配到了单片机扩展数据区起始地址为0x000025的地方。

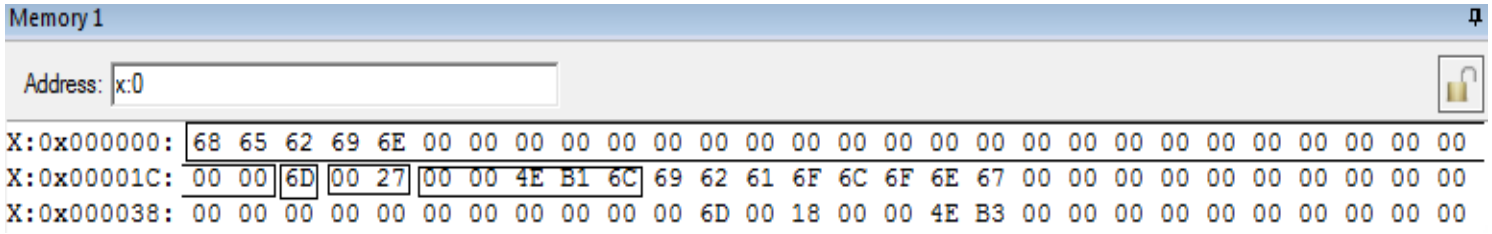
复杂数据结构
--结构

Watch 1

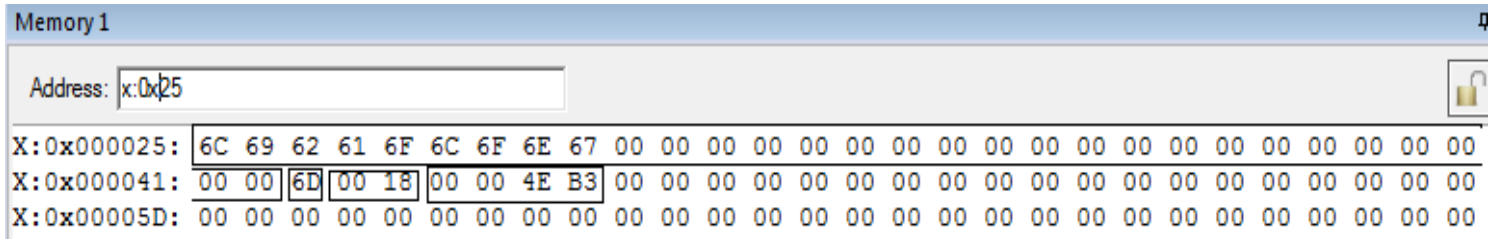
Name	Value	Type
stu	X:0x000000 stu	array[2] of struct stud...
[0]	X:0x000000 &stu	struct student
name	X:0x000000 stu[] "hebin"	array[30] of char
gender	0x6D 'm'	char
age	0x0027	int
num	0x00004EB1	long
[1]	X:0x000025	struct student
name	X:0x000025 "libaolong"	array[30] of char
gender	0x6D 'm'	char
age	0x0018	int
num	0x00004EB3	long
student	<cannot evaluate>	uchar
<Enter expression>		

复杂数据结构
--结构

- 通过stu[0]和stu[1]在单片机扩展数据区的内容可以更好的理解该数据类型。



Memory 1窗口中结构变量stu[0]的内容



Memory 1窗口中结构变量stu[1]的内容

结构
--指向结构的指针

在C语言中，一个指向结构类型变量的指针称为结构型指针，该指针变量的值是它所指向的结构变量的起始地址。

- 结构型指针也可以用来指向结构数组，或者指向结构数组中的元素。定义结构型指针的一般格式为：

  struct 结构类型标识符 *结构指针标识符

- 通过结构型指针引用结构元素的格式为：

  结构指针标识符->结构中的元素

结构
--指向结构的指针

【例】结构体使用的例子

```
#include "stdio.h"
#include "reg51.h"
struct student{                          //定义结构体
        char name[30];                   //字符类型数组
        char gender;                     //字符类型数据
        int age;                         //整型数据
        long int num;                    //长整型数据
      };
xdata struct student stu[2],*p;          //声明结构数组和指针
```

结构
--指向结构的指针

```
void main()
{
   int i;
  SCON= 0x52;
 TMOD = 0x20;
 TCON = 0x69;
 TH1 = 0xF3;
 for(i=0;i<2;i++)
{
      p=&stu[i];                                  //结构指针指向当前数组首地址
      printf("please input stu[%d].name\n",i);
      scanf(“%s”,&p->name);                       //输入p->name指向字符串的信息
      getchar();
```

结构
--指向结构的指针

```
printf("please input stu[%d].gender\n",i);
scanf("%c",&p->gender);                    //输入p->gender指向字符的信息
putchar('\n');
printf("please input stu[%d].age\n",i);
scanf("%d",&p->age);                       //输入p->age指向整数的信息
printf("please input stu[%d].num\n",i);
scanf("%ld",&p->num);                      //输入p->num指向长整数的信息
}
putchar('\n');
```

结构
--指向结构的指针

```
for(i=0;i<2;i++)
 {
    p=&stu[i];                          //结构指针指向当前数组首地址
    printf("the following students information is:\n");
    printf("stu[%d].name=%s, ",i,p->name);    //打印p->name指向字符串信息
    printf("stu[%d].gender=%c, ",i,p->gender); //打印p->gender指向字符信息
    printf("std[%d].age=%d, ",i,p->age);         //打印p->age指向整数的信息
    printf("std[%d].num=%ld, ",i,p->num);      //打印p->num指向长整数的信息
    putchar('\n');
    }
while(1);
}
```

结构
--指向结构的指针

注：读者可以进入到本书所提供资料的STC_example\例子6-58目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5按键。打开UART #1窗口界面，在该界面下按照提示信息输入结构元素的值，最后打印输入的信息。

复杂数据结构
--联合

在C语言中，提供了联合类型的数据结构。

- 在一个联合的数据结构中，可以包含多个数据类型。
  - 但是，不像结构类型那样，所有的数据单独分配存储空间，而联合数据类型是共用存储空间。
  - 这种方法分时使用同一个存储空间，因此提高了单片机片内数据存储空间的使用效率。
- 联合类型变量的定义格式：

```
union 联合变量的名字
    {成员列表
} 变量列表
```

复杂数据结构
--联合

【例】联合数据结构的例子

```
#include "stdio.h"
#include "reg51.h"
union {                          //定义联合体
      char data_str[8];          //定义字符数组
      struct {                   //定义结构体
            int  a;
            int  b;
            long int c;
           }data_var;
}shared_information;             //联合体的名字
```

复杂数据结构
--联合

```
void main()
{
   int i;
   SCON= 0x52;
   TMOD = 0x20;
   TCON = 0x69;
   TH1 = 0xF3;
   shared_information.data_var.a=100;            //结构体整型数a赋值
    shared_information.data_var.b=1000;          //结构体整型数b赋值
    shared_information.data_var.c=100000000; //结构体长整型c赋值
    for(i=0;i<8;i++)                                    //打印联合体内data_str
    {
       printf("data[%d]=%c,\n",i,shared_information.data_str[i]);
     }
while(1);}
```

复杂数据结构
--联合

注：读者可以进入到本书所提供资料的STC_example\例子6-59目录下，在Keil μVision5集成开发环境下打开该设计。

- 并进入调试器模式，按F5按键。打开Watch 1窗口界面，在该界面下输入data_str和data_var。data_str和data_var都位于单片机片内数据区起始地址为0x22的位置，共享8个字节的片内数据区。

Watch 1

Name	Value	Type
data_var	<cannot evaluate>	uchar
shared_inform...	D:0x22 &shared_infor...	union <untagged>
data_str	D:0x22 &shared_infor...	array[8] of char
[0]	0x00	char
[1]	0x64 'd'	char
[2]	0x03	char
[3]	0xE8 '?'	char
[4]	0x05	char
[5]	0xF5 '?'	char
[6]	0xE1 '?'	char
[7]	0x00	char
data_var	D:0x22 &shared_infor...	struct <untagged>
a	0x0064	int
b	0x03E8	int
c	0x05F5E100	long
<Enter expression>		

复杂数据结构
--联合

- data_var.a=0x0064，存在下面的关系：
  - data_var.a高8位=>data_str[0]；
  - data_var.a低8位=>data_str[1]；
- data_var.b=0x03E8，存在下面的关系：
  - data_var.b高8位=>data_str[2]；
  - data_var.b低8位=>data_str[3]；
- data_var.c=0x05F5E100，存在下面的关系：
  - data_var.c第31位~第24位=>data_str[4]；
  - data_var.c第23位~第16位=>data_str[5]；
  - data_var.c第15位~第8位=>data_str[6]；
  - data_var.c第7位~第0位=>data_str[7]；

复杂数据结构
--枚举

在C语言中，提供了枚举数据类型。

- 如果一个变量只有有限个取值，则可以将变量定义为枚举类型。
  - 例如：对于星期来说，只有星期一~星期日这7个可能的取值情况；
  - 对于颜色，只有红色、蓝色和绿色三个基本颜色。所以，星期和颜色都可以定义为枚举类型
- 枚举类型的格式为：

  enum 枚举名字{枚举值列表} 变量列表;

复杂数据结构
--枚举

- 在枚举值列表中，每一项代表一个整数值。默认：
  - 第一项为0；
  - 第二项为1；
  - 第三项为2；
  - 以此类推。
- 此外，也可以通过初始化指定某些项的符号值。

复杂数据结构
--枚举

【例】枚举数据结构的例子

该例子将若干红、绿、黄三种颜色的小球全排列组合，输出每种组合的三种颜色。

```
#include "stdio.h"
#include "reg51.h"
enum color{red,green,blue};
enum color i,j,k,st;
void main()
{
    int n=0,m;
    SCON= 0x52;
    TMOD = 0x20;
    TCON = 0x69;
    TH1 = 0xF3;
```

复杂数据结构
--枚举

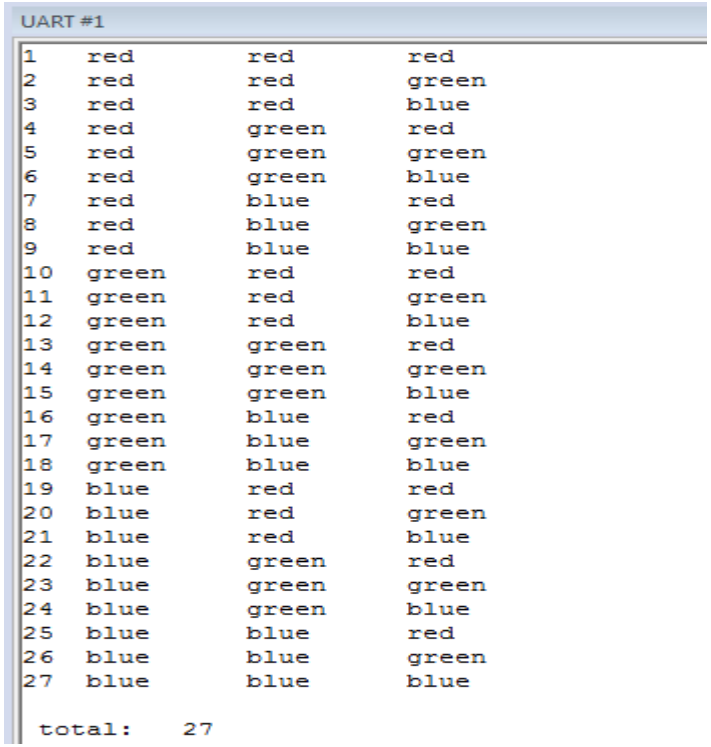
```
for(i=red;i<=blue;i++)
 for(j=red;j<=blue;j++)
  for(k=red;k<=blue;k++)
  {
    n=n+1;
     printf("%-4d",n);
     for(m=1;m<=3;m++)
    {
       switch(m)
      {
        case 1 : st=i;break;
        case 2:  st=j;break;
        case 3:  st=k;break;
        default: break;
  }
```

复杂数据结构
--枚举

```
    switch(st)
    {
        case red  :  printf("%-10s","red");break;
        case green:  printf("%-10s","green");break;
        case blue :  printf("%-10s","blue");break;
       default: break;
    }
 }
printf("\n");
}
printf("\n total:%5d\n",n);
while(1);
}
```

复杂数据结构
--枚举

注：读者可以进入到本书所提供资料的STC_example\例子6-60目录下，在Keil μVision5集成开发环境下打开该设计，并进入调试器模式，按F5按键。打开UART #1窗口界面，在该界面下看到输出数据的信息。



C程序中使用汇编语言

有时候需要在使用C语言编写程序代码的过程中使用汇编语言。

- 有些汇编程序在整个软件设计工程中是必须的。
  - 比如：启动引导代码。
- 而其它地方使用汇编语言是为了提高整个软件设计工程的运行效率。
- 在C语言中使用汇编语言的方法包括两种：
  - 在C语言程序代码中内嵌汇编语言；
  - C语言代码程序，调用外部汇编语言编写的程序。

C程序中使用汇编语言
--内嵌汇编语言

在C源文件中，将汇编代码写在指令：

```
#pragma asm
.....
#pragma endasm
```

中间。

- 本节将通过一个具体的例子说明在C语言中内嵌汇编语言的方法。

C程序中使用汇编语言
--内嵌汇编语言

在C程序中内嵌汇编语言的步骤主要包括：

- 建立新的设计工程。
- 新建并添加一个名字为main.c的源文件。按下面输入代码，保存该文件。

```
#include "stdio.h"
#include "reg51.h"
idata unsigned char C1 _at_ 0x22;     //_at_声明变量C1在idata区0x22位置
idata unsigned char B1 _at_ 0x24;     //_at_声明变量B1在idata区0x24位置
idata unsigned char D1 _at_ 0x26;     //_at_声明变量D1在idata区0x26位置
idata unsigned int e  _at_ 0x28;        //_at_声明int类型变量e在idata区的0x28位置
```

C程序中使用汇编语言
--内嵌汇编语言

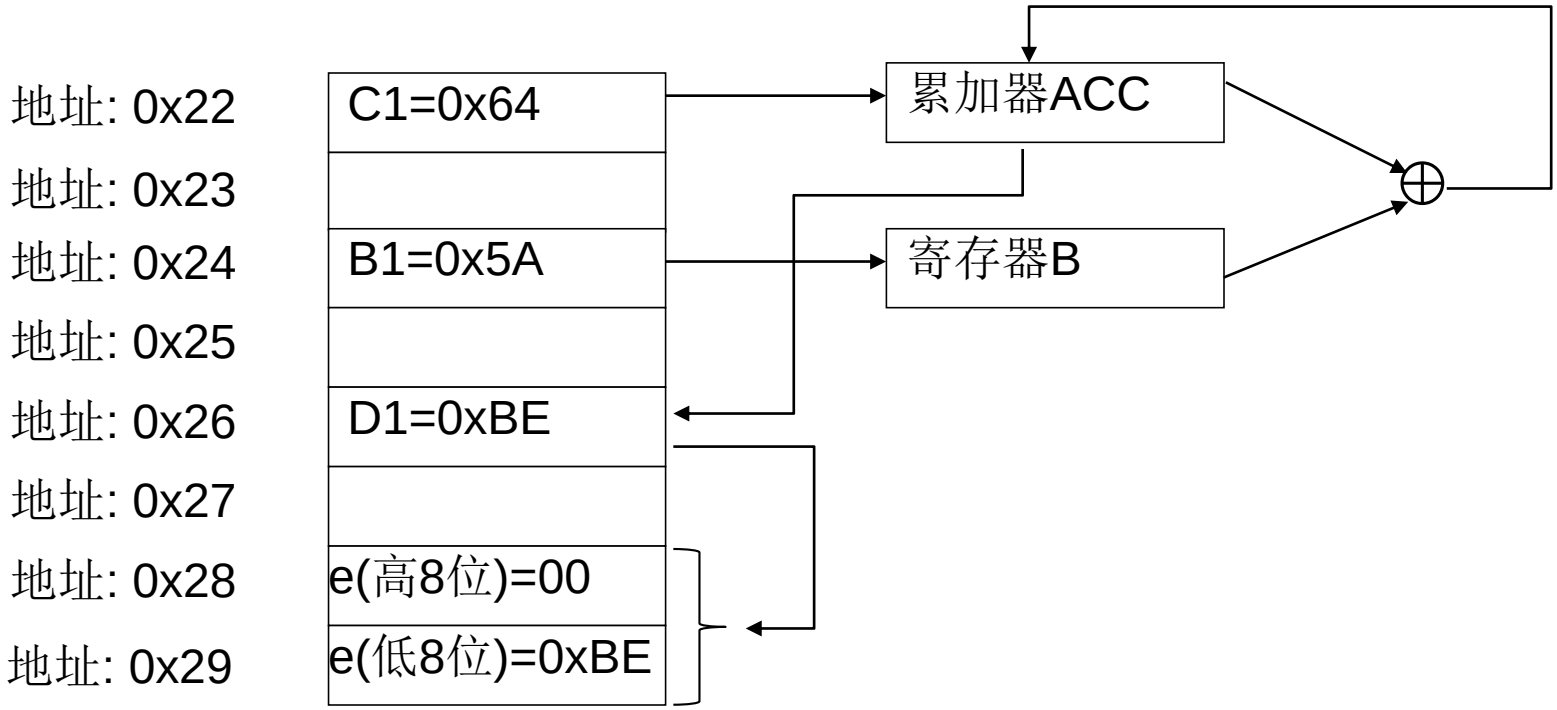
```
void main()
{
    C1=100;                 //变量C1赋值为100
    B1=90;                  //变量B1赋值为90
    SCON= 0x52;
    TMOD = 0x20;
    TCON = 0x69;
    TH1 = 0xF3;
```

C程序中使用汇编语言
--内嵌汇编语言

```
#pragma asm            //内嵌汇编命令，表示开始
   MOV A,0x22          //单片机片内数据区0x22单元内容送给累加器ACC
   MOV B,0x24          //单片机片内数据区0x24单元内容送给寄存器B
   ADD A,B             //累加器ACC和寄存器B的内容相加，结果保存ACC
   MOV 0x26,A          //累加器ACC内容送到单片机片内数据区0x26单元。
 #pragma endasm        //内嵌汇编命令，表示结束
 e=D1;                 //将D1的值送给e
printf(“%d\n”,e);      //打印e的值
while(1);
}
```

C程序中使用汇编语言
--内嵌汇编语言

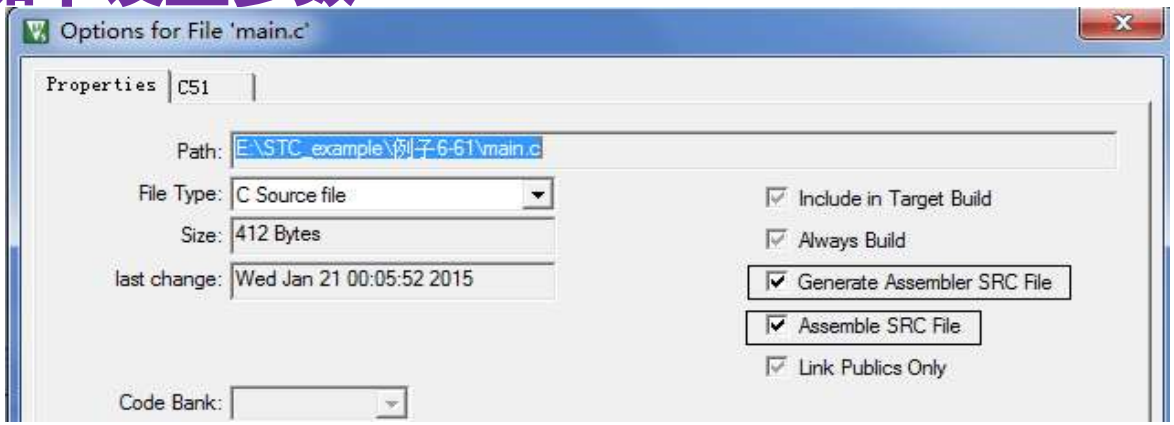
该程序代码的设计思路，如图所示。



内嵌汇编设计原理

C程序中使用汇编语言
--内嵌汇编语言

- 在当前工程主界面左侧的Project窗口中，找到并选择mian.c，单击右键出现浮动菜单。在浮动菜单内，选择Options for File ‘main.c’...选项。出现Options for File ‘main.c’...对话框界面，按如下设置参数：

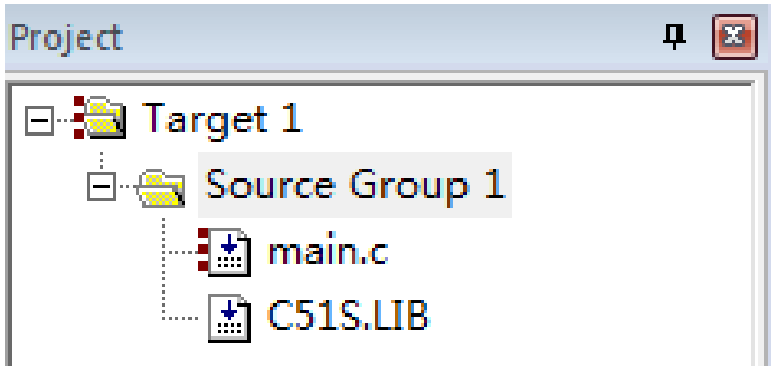


  - 在该界面内，单击“Generate Assembler SRC文件”前面的复选框。
  - 在该界面内，单击“Assemble SRC File”文件前面的复选框。

- 单击OK按钮。

C程序中使用汇编语言
--内嵌汇编语言

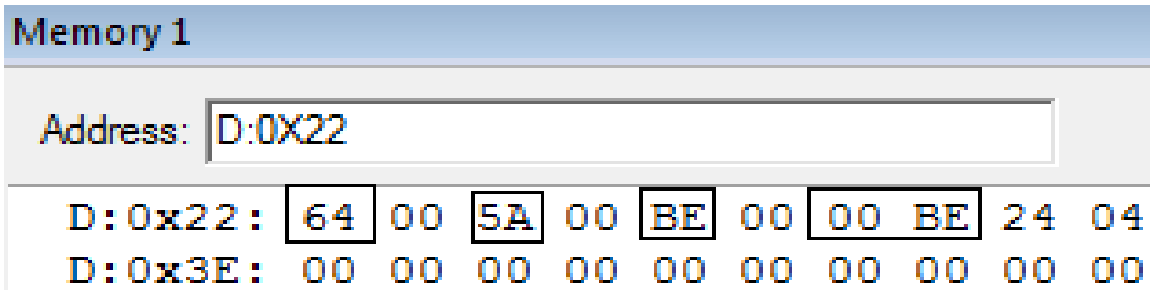
- 在当前工程主界面左侧的Project窗口中，看到新添加的库文件。



- 编译设计。
- 进入调试器模式。
- 打开Watch 1窗口界面。在该界面中，分别输入B1、C1和D1。
- 单步运行程序，观察变量的变化情况。

C程序中使用汇编语言
--内嵌汇编语言

- 打开Memory窗口界面。在该界面中，输入d:0x22，观察单片机片内数据区的存储内容。



- 打开UART #1窗口。在该窗口下，打印e的值为190。
- 退出调试器模式，并关闭该设计。

C程序中使用汇编语言
--调用汇编程序

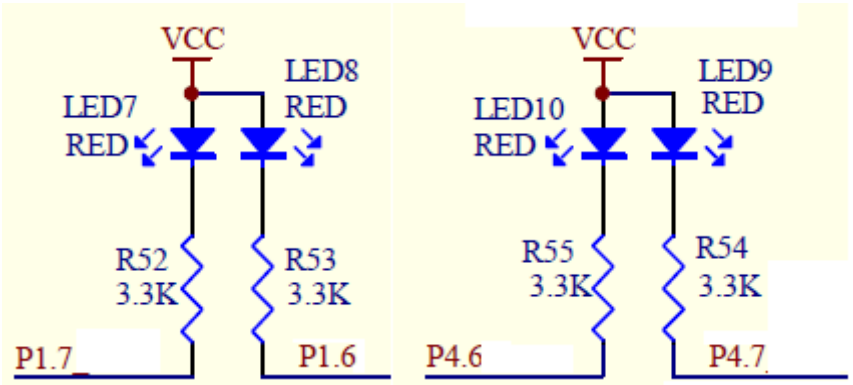
本节将在外部调用汇编语言程序对端口进行控制，并通过STC提供的学习板对设计进行硬件仿真和调试。

- 在主程序中，编写调用汇编语言编写的子函数代码。
- 在汇编语言中，对累加器ACC进行递增操作，并且通过STC学习板上的四个LED灯，显示ACC中的第3位~第0位值。显示的范围从0~15。

C程序中使用汇编语言
--调用汇编程序

硬件设计原理

在该设计中，使用了STC提供的学习板。在该学习板上提供了四个LED灯、分别用LED7、LED8、LED9和LED10表示。



- 这四个LE灯的阳极共同接到+5V，另一端通过限流电阻R52、R53、R54、R55与STC的 ISP15W4K58S4单片机P1.7、P1.6、P4.7和P4.6引脚连接。

C程序中使用汇编语言
--调用汇编程序

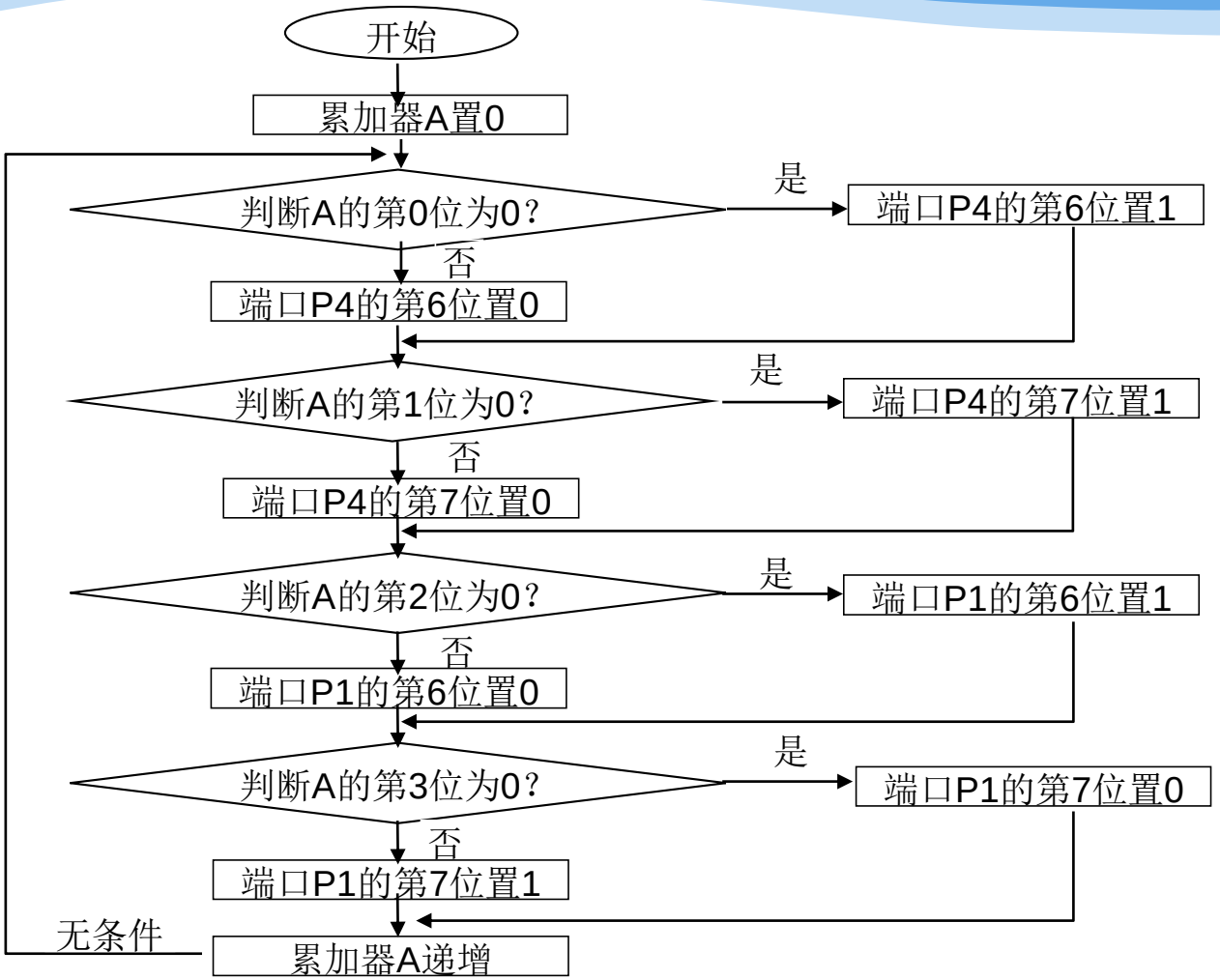
- 当：
  - 单片机对应的引脚置位为低时，所连接的LED亮；

单片机对应的引脚置位为高时，所连接的LED灭。

软件设计原理

- 下面给出软件设计流程图，如后图所示。

C程序中使用汇编语言
--调用汇编程序



C程序中使用汇编语言
--调用汇编程序

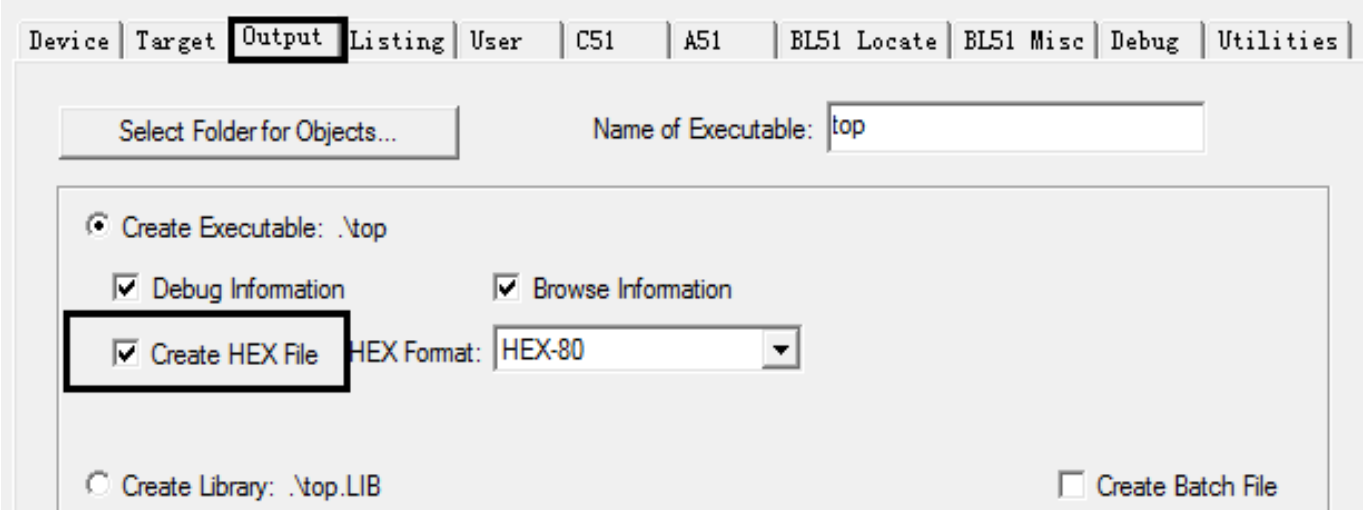
设计实现过程

通过C语言，调用汇编语言的步骤主要包括：

- 建立新的设计工程。
- 在当前工程主界面中，选中Target 1，单击右键，出现浮动菜单。在浮动菜单内，选择Options for Target ‘Target 1’选项。

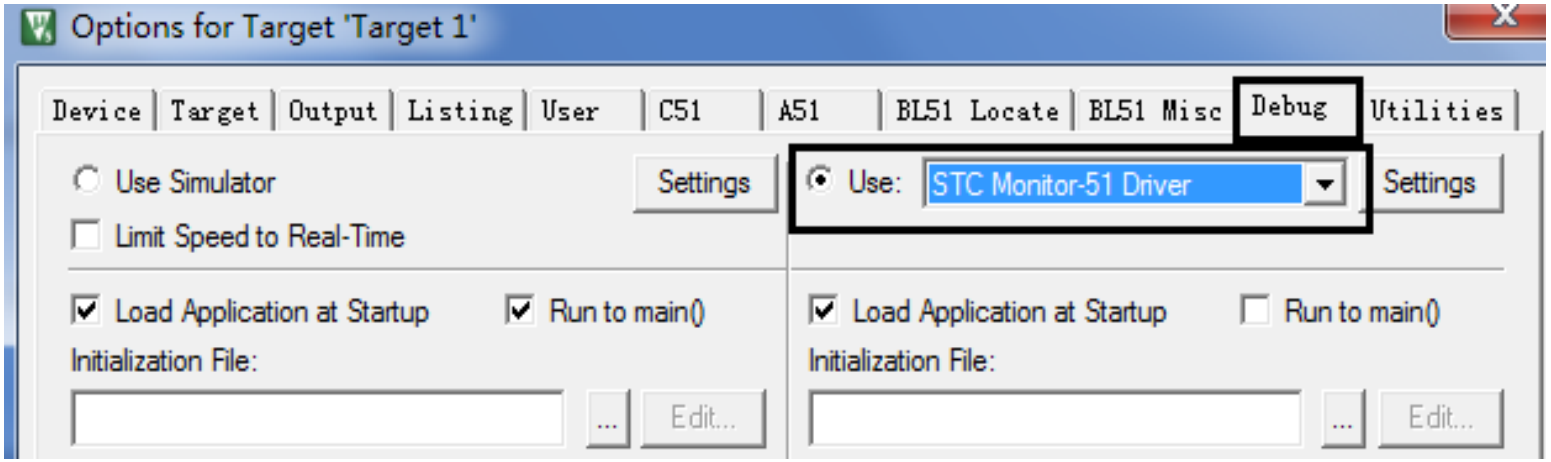
C程序中使用汇编语言
--调用汇编程序

- 出现Options for Target ‘Target 1’对话框界面。在该界面窗口下，按如下设置参数：
  - 选择Output标签。在该标签窗口下，选中Create HEX File前面的复选框，如图所示。



C程序中使用汇编语言
--调用汇编程序

- 选择Debug标签。在该标签窗口下，选中右侧Use前的复选框，表示要使用硬件仿真。在右侧的下拉框中，选择STC Monitor-51 Driver选项。单击右侧的Settings按钮，出现Target Setup选项。选择COM Port，单击OK按钮，返回到设置界面。



C程序中使用汇编语言
--调用汇编程序

- 单击OK按钮，退出Options for Target ‘Target 1’对话框界面。
- 建立新的名字为main.c的C语言源文件，输入下面的代码，并保存该设计文件。

```
extern void CONTROL_GPIO(void);      //声明函数为外部函数
void main()
{
CONTROL_GPIO();                      //调用外部函数
}
```

C程序中使用汇编语言
--调用汇编程序

- 建立新的名字为gpio.a51文件，输入下面的代码，并保存该设计文件。

```
$NOMOD51                    //使A51不识别预定义的SFR符号
$INCLUDE (reg51.inc)        //包含reg51.inc文件，该文件预定了符号
NAME CONTROL_GPIO           //模块名字CONTROL_GPIO
P4      DATA 0C0H           //定义端口4的地址
segcode segment code        //声明代码段segcode
public CONTROL_GPIO         //声明public，外部模块可以调用
rseg segcode                //引用代码段segcode
CONTROL_GPIO:               //程序入口
    MOV  A,#0               //初始化累加器ACC为0
```

C程序中使用汇编语言
--调用汇编程序

```
Loop2:   JB     ACC.0, SETP41    //判断ACC的第0位是否为1，是则跳转
         SETB   P4.6             //否则将P4.6设置为1（灯灭）。
         JMP    CON              //无条件跳转到CON
SETP41:  CLR    P4.6             //将P4.6设置为0（灯亮）
CON:     JB     ACC.1,SETP42     //判断ACC的第1位是否为1，是则跳转
         SETB   P4.7             //否则将P4.7设置为1（灯灭）。
         JMP    CON1             //无条件跳转到CON1
SETP42:  CLR    P4.7             //将P4.7设置为0（灯亮）
CON1:    JB     ACC.2, SETP43    //判断ACC的第2位是否为1，是则跳转
         SETB   P1.6             //否则将P1.6设置为1（灯灭）。
         JMP    CON2             //无条件跳转到CON2
```

C程序中使用汇编语言
--调用汇编程序

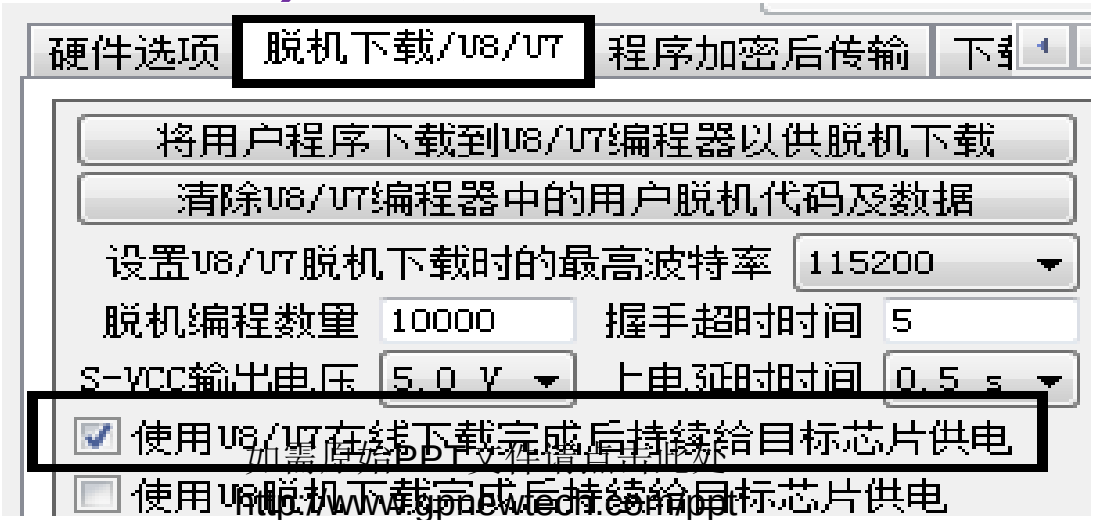
```
SETP43:  CLR    P1.6              //否则将P1.6设置为0（灯亮）。
  CON2:  JB     ACC.3,SETP44      //判断ACC的第3位是否为1，是则跳转
         SETB   P1.7              //否则将P1.7设置为1（灯灭）。
         JMP    CON3              //无条件跳转到CON3
SETP44:  CLR    P1.7              //否则将P1.7设置为0（灯亮）。
CON3:    INC    A                 //累加器A递增
         JMP    Loop2             //无条件跳转到Loop2
         RET
END
```

C程序中使用汇编语言
--调用汇编程序

- 在gpio.a51的第28行设置断点。
- 将STC提供的学习板通过USB电缆和主机进行连接。
- 打开STC提供的STC-ISP(v6.82)软件。在该界面中按如下设置参数：
  - 在该界面串口号右侧下拉框中，选择所识别出来的STC USB-UART的窗口号。
  - 在该界面中，找到单击打开程序按钮，定位到当前设计工程目录下，找到并打开top.HEX文件。

C程序中使用汇编语言
--调用汇编程序

- 在该软件主界面下，选择脱机下载/U8/U7标签界面。在该标签界面下，选中“使用U8/U7在线下载完成后持续给目标芯片供电”前面的复选框。
- 在STC-ISP软件主界面右侧窗口内，选择Keil仿真设置标签。在该标签窗口下，单击“将IAP15W4K58S4设置为仿真芯片（宽压系统，支持USB下载）”按钮。



C程序中使用汇编语言
--调用汇编程序

- 按一下STC学习板上的SW19按键。
- 开始下载程序。当程序下载成功后，在STC-ISP软件的右下角窗口中出现提示信息。
- 在Keil μVision5主界面主菜单下，选择Debug->Start/Stop Debug Session。
- 进入调试器模式，连续按F5按键，断点运行程序。
- 退出调试器模式，并关闭该设计。

C语言端口控制实现

本节将全部使用C语言实现上面的例子。实现C语言控制端口的步骤主要包括：

- 建立新的设计工程。
- 在当前工程主界面中，选中Target 1，单击右键，出现浮动菜单。在浮动菜单内，选择Options for Target ‘Target 1’选项。
- 出现Options for Target ‘Target 1’对话框界面。在该界面窗口下，选择Output标签。在该标签窗口下，选中Create HEX File前面的复选框。
- 单击OK按钮，退出Options for Target ‘Target 1’对话框界面。

C语言端口控制实现

- 建立新的名字为main.c的C语言源文件，输入下面的代码，并保存该设计文件。

```
#include "reg51.h"
void main()
{
}
```

- 保存设计代码。

C语言端口控制实现

- 对设计进行编译。成功编译后，在主界面左侧的Project窗口中出现reg51.h文件。双击打开reg51.h文件，下面对该文件进行修改：
  - 在第16行添加一行代码，该行代码声明P4端口的地址，如图1所示。
  - 在第92行添加代码，该行代码声明P4端口每一位的地址，如图2所示。
  - 保存该文件。

```
12  sfr P0   = 0x80;
13  sfr P1   = 0x90;
14  sfr P2   = 0xA0;
15  sfr P3   = 0xB0;
16  sfr P4   = 0xC0;
17  sfr PSW  = 0xD0;
```

```
91  /* P4 */
92  sbit P40=0xC0;
93  sbit P41=0xC1;
94  sbit P42=0xC2;
95  sbit P43=0xC3;
96  sbit P44=0xC4;
97  sbit P45=0xC5;
98  sbit P46=0xC6;
99  sbit P47=0xC7;
```

- 打开main.c文件，添加剩余的代码，并保存文件。

C语言端口控制实现

```
void main()
{
   int i=0;
   long  int j=0;
   while(1)                          //无限循环
   {
      for(i=0;i<4;i++)               //for循环从0~3
       {
          switch(i)
           {
                case 0:               //当计数值为0，
                   {
                      P46=1;          //P4.6置位为1（灯灭）
                      P47=1;          //P4.7置位为1（灯灭）
                    }
                break;
```

C语言端口控制实现

```
case 1:                          //当计数值为1
{
   P46=0;                        //P4.6置位为0（灯亮）
   P47=1;                        //P4.7置位为1（灯灭）
}
 break;
case 2:                          //当计数值为2
{
    P46=1;                       //P4.6置位为1（灯灭）
    P47=0;                       //P4.7置位为0（灯亮）
}
break;
```

C语言端口控制实现

```
case 3:                                  //当计数值为3
{
    P46=0;                                 //P4.6置位为0（灯亮）
    P47=0;                               //P4.7置位为0（灯亮）
}
 break;
default :  ;
}
for(j=0;j<222222;j++){;}          //软件延迟代码
 }
 }
 }
```

C语言端口控制实现

- 对该文件进行编译，生成HEX文件。
- 打开STC-ISP软件。按照前面选择正确的器件型号、设置串口参数。
- 在该软件中，单击“打开程序文件”按钮。定位到当前工程路径，并打开top.HEX文件。
- 在STC-ISP软件中，单击“下载/编程”按钮。
- 在STC提供学习板上，找到并按一下SW19按键。开始编程STC单片机。
- 关闭该设计。

C语言中断程序实现
--C语言中断程序实现原理

Keil Cx51编译器支持在C语言源程序中直接编写8051单片机的中断服务程序，从而降低了采用汇编语言编写中断服务程序的复杂度。

- 在Keil Cx51编译器中，对函数定义进行了扩展，增加了关键字interrupt。定义中断服务程序的格式为：

  函数类型 函数名(形参列表) [inperrupt n][using n]

  - interrupt后面的n为中断号，范围为0~31。编译器中8n+3处产生中断向量。
  - using后面的n用于选择使用的工作寄存器组。

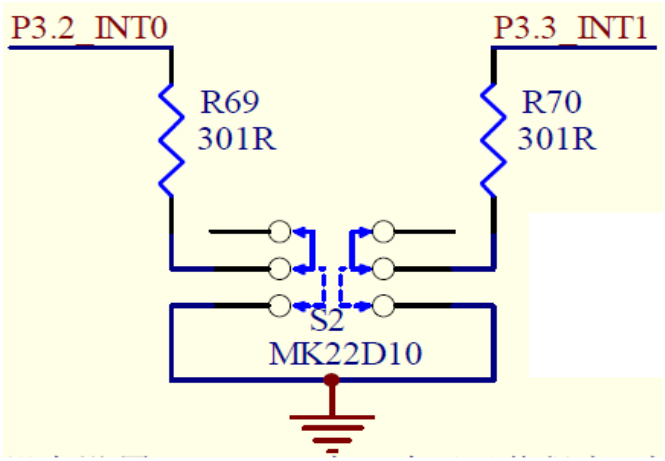
C语言中断程序实现
--外部中断电路原理

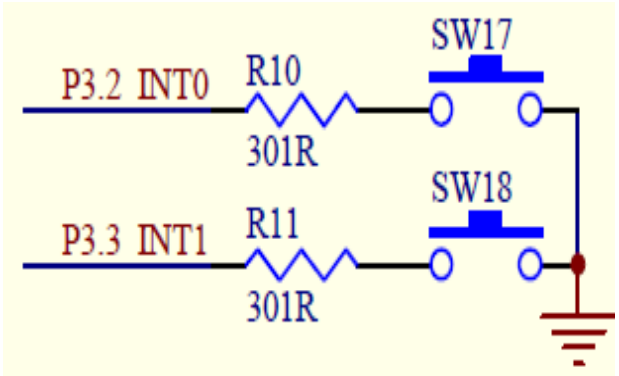
在该设计中，将设计一个在0~3之间计数（4进制）的计数器。通过STC学习板上的P4.6和P4.7端口上的LED，显示计数的值。

- 和前面例子不一样的是，计数是通过外部中断INT0触发的，即：每次当INT0引脚下拉到地时，触发一次中断，计数器递增一次。
- 该设计的硬件电路的触发由开关控制，如后图所示。

C语言中断程序实现
--外部中断电路原理

- STC学习板上没有焊接开关。但是，提供了SW17和SW18两个按键。当：
  - 按下SW17时，P3.2引脚接地，也就是产生一个INT0下降沿低脉冲信号。
  - 按下SW18时，P3.3引脚接地，也就是产生一个INT1下降沿低脉冲信号。





C语言中断程序实现
--C语言中断具体实现过程

在该设计中，当通过外部中断触发中断事件，并进行相应的处理。实现C语言中断程序的步骤主要包括：

- 建立新的设计工程。
- 在当前工程主界面中，选中Target 1，单击右键，出现浮动菜单。在浮动菜单内，选择Options for Target ‘Target 1’选项。
- 出现Options for Target ‘Target 1’对话框界面。在该界面窗口下，选择Output标签。在该标签窗口下，选中Create HEX File前面的复选框。
- 单击OK按钮，退出Options for Target ‘Target 1’对话框界面。

C语言中断程序实现
--C语言中断具体实现过程

- 建立名为main.c的C语言源文件，输入下面的代码，并保存该设计文件。

```
#include "reg51.h"
int i=0;                          //声明全局变量i，初值为0
servivce_int0() interrupt 0       //声明该函数是中断函数，对应于中断号0
{
    if(i==4) i=0;                 //如果i计数到4，则复位到0
    if(i==0)                      //当计数值为0时
   {
      P46=1;                      //设置P4.6为1（灯灭）
     P47=1;                       //设置P4.7为1（灯灭）
   }
```

C语言中断程序实现
--C语言中断具体实现过程

```
else if(i==1)                    //当计数值为1时
{
    P46=0;                       //设置P4.6为0（灯亮）
    P47=1;                       //设置P4.7为1（灯灭）
 }
else if(i==2)                    //当计数值为2时
{
    P46=1;                       //设置P4.6为1（灯灭）
    P47=0;                       //设置P4.7为0（灯亮）
}
```

C语言中断程序实现
--C语言中断具体实现过程

```
    else if(i==3)                    //当计数值为3时
    {
      P46=0;                         //设置P4.6为0（灯亮）
       P47=0;                        //设置P4.7为0（灯亮）
    }
    i=i+1;                           //i递增
}
```

C语言中断程序实现
--C语言中断具体实现过程

```
void main()
{
    IT0=1;                  //只允许下降沿触发
    EX0=1;                  //允许外部中断0
    EA=1;                   //CPU允许响应中断
    while(1);               //无限循环
}
```

- 按照前面的方法，编译并下载设计到STC单片机中。