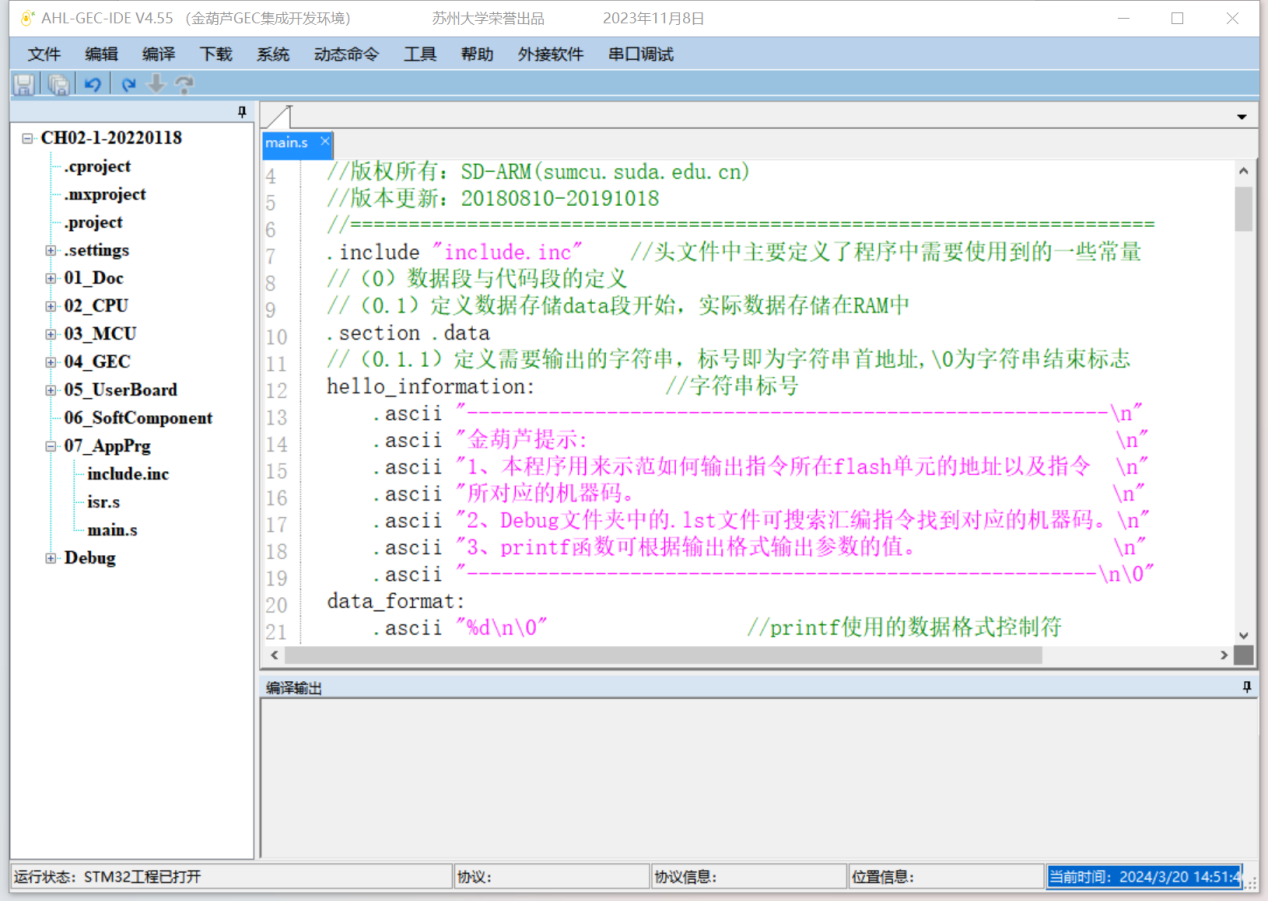
Part\_01:输出指定内容和累加

now照葫芦画瓢，开始新一个项目的流程。

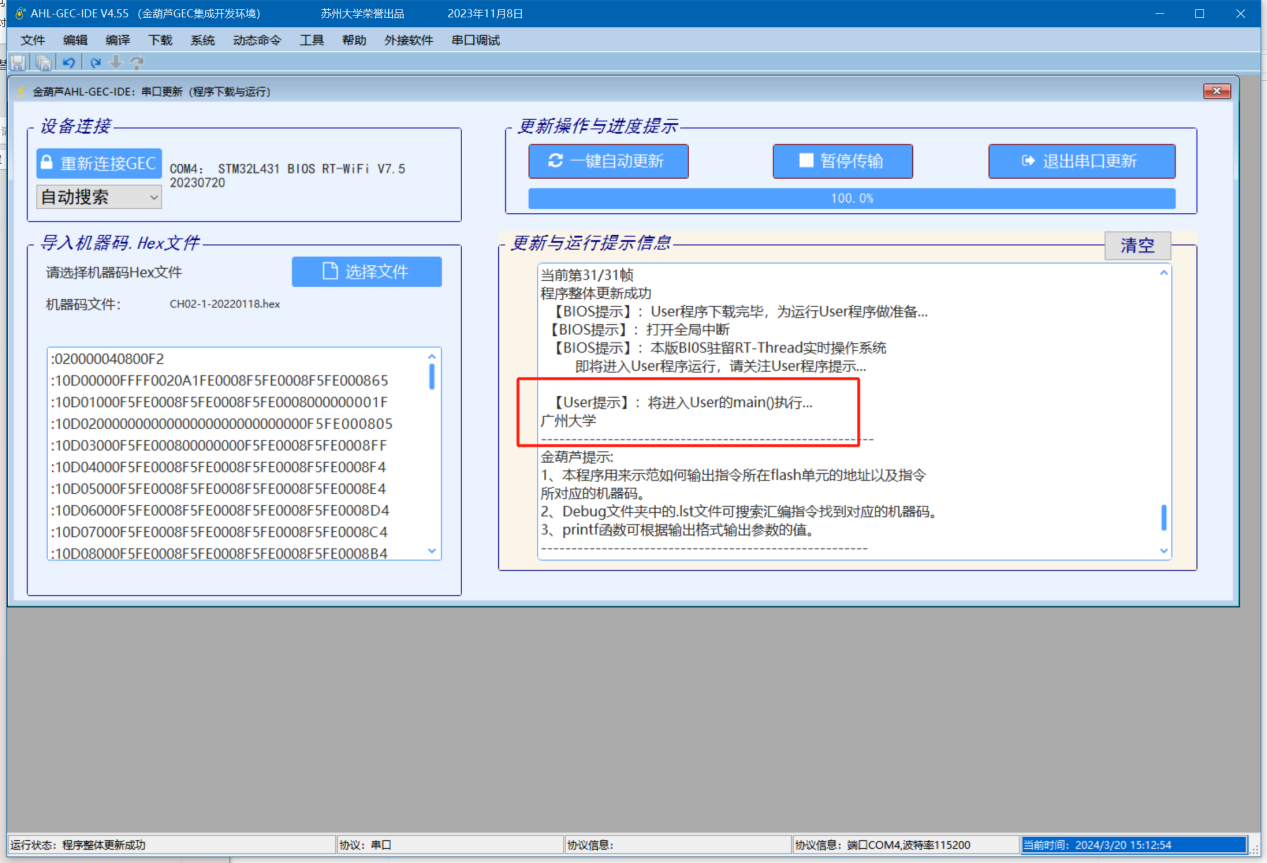
打开04-Software/ch02/CH02-1-20220118工程目录编译下载运行，单击菜单“文

件”→“导入工程”,出来的就是main.s



现在来看需求：修改main.s源文件，增加以下内容:

1. 在第一行显示“广州大学”字样。



理解一下：在汇编语言中，通常将程序的内存分为不同的段，其中最常见的是数据段（.data）和代码段（.text）。

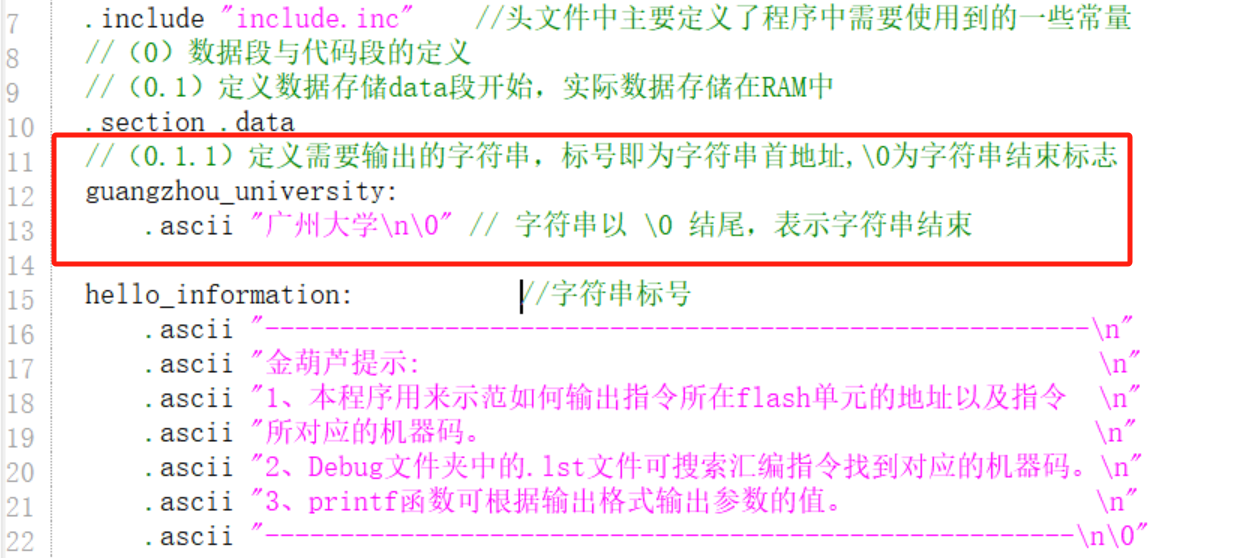
数据段（.data）：

* 数据段用于存储程序中使用的数据，这些数据通常存储在 RAM 中。
* 在数据段中，可以定义变量、常量以及字符串等数据。
* 在汇编语言中，用 .section .data 指令来定义数据段。

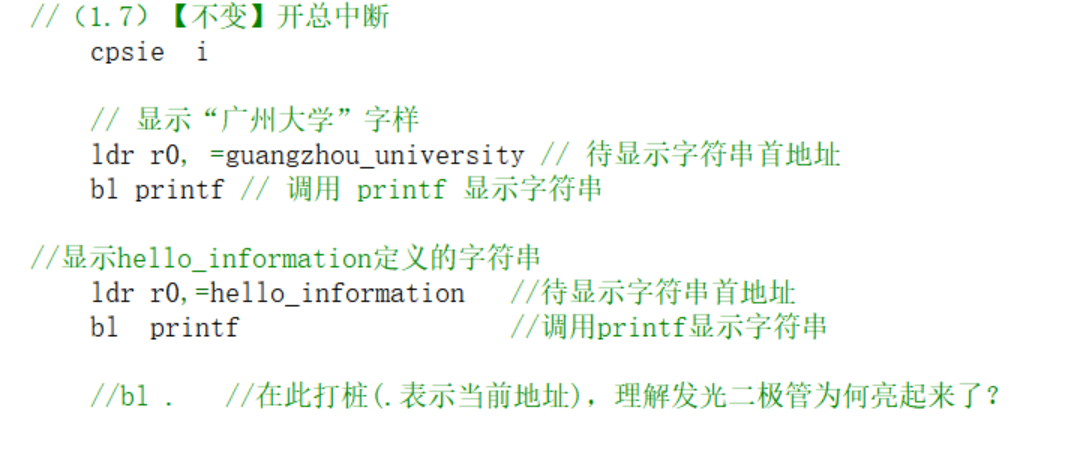
代码段（.text）：

* 代码段存储着程序的指令，通常存储在 Flash 或 ROM 中。
* 在代码段中，编写的汇编指令被存储和执行。
* 在汇编语言中，用 .section .text 指令来定义代码段。

所以找代码里面的数据段，定义一下广州大学，他怎么写我怎么写



在数据段定义了，然后就得在代码段编写相应的操作来执行，所以就去主函数的初始化部分之后调用 printf 函数来显示这个新的字符串，放在hello的前面



这段代码首先将字符串 "guangzhou\_university" 的首地址加载到寄存器 r0 中，然后通过 bl 指令调用 printf 函数，将寄存器 r0 中的地址作为参数传递给 printf 函数。

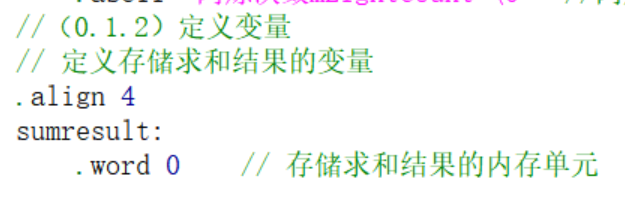
在 printf 函数内部，它会使用这个地址来访问存储在该地址处的字符串，然后将字符串打印到输出设备上

2、编写一个1+2+..+10的程序，将求和结果存入名为

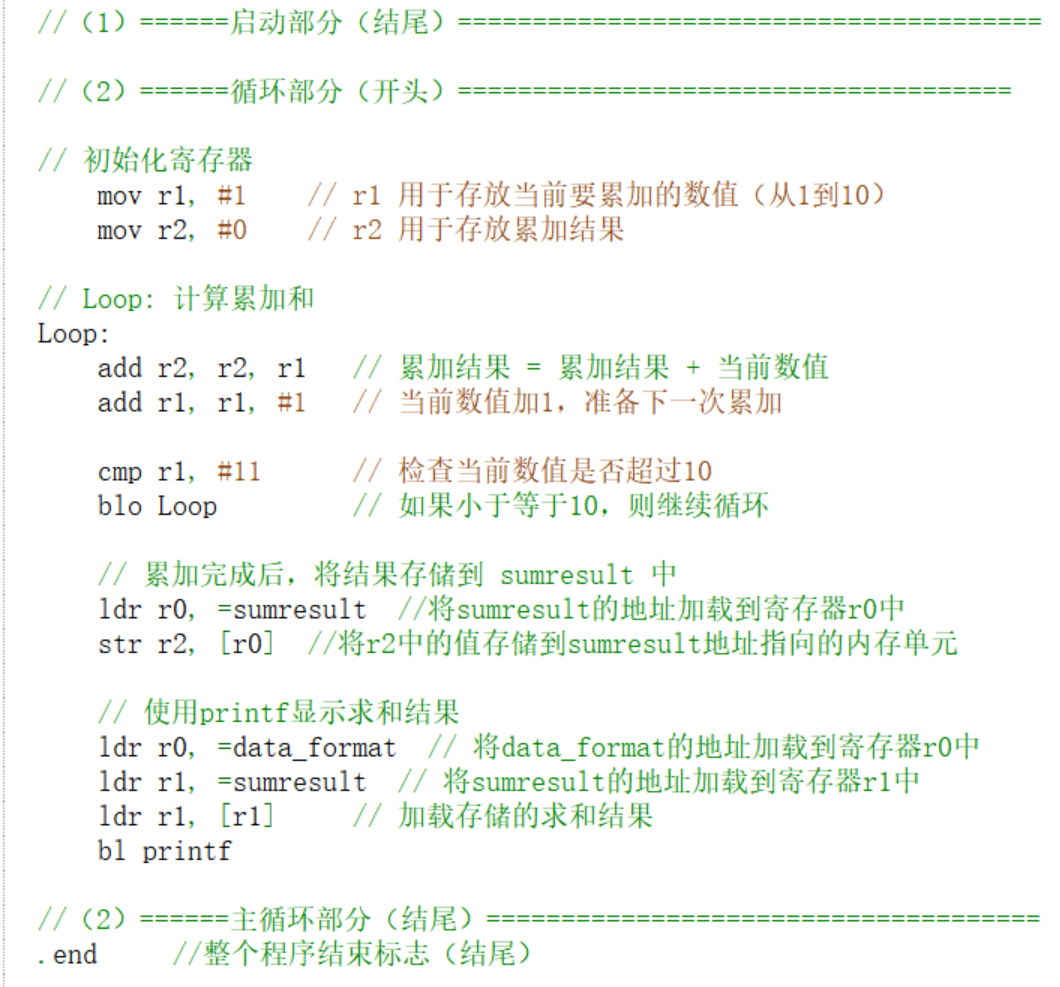
“sumresult”的内存单元中，并将求和结果用printf显示出来。



同样的操作，还是在数据段定义变量，在代码段实现操作：



初始内存单元的值为0是为了防止有其他的数据混杂导致结果错误



先初始化一下变量，然后再写操作（类似编写c语言的良好习惯）

然后是经典的累加操作，写一个循环，退出条件=11

累加完成之后就把得到的值存起来，再用printf输出

有的帅哥帅姐会问，为啥用这么怪异的格式输出，因为汇编就是这么的古板：

在ARM汇编中，函数的参数传递通常是按照一定的约定来进行的。在函数调用中，特定的寄存器用于传递参数给函数，这些约定由编译器和汇编器共同遵循。对于printf函数，通常的约定是：

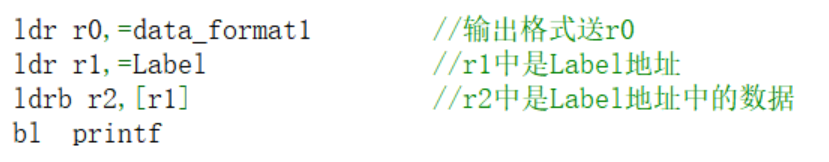
第一个参数（通常存储在 r0 寄存器中）是格式控制字符串，即包含格式化指示符的字符串，它告诉printf函数如何格式化后续参数。

后续的参数（如待打印的数值）通常存储在其他寄存器中，例如 r1、r2、r3 等。

那又有靓仔会问，累加得到的数值能不能用立即数存到sumresult 里，而不用先加载地址再加载结果呢？

那当然不行，因为因为地址通常是在运行时确定的，而立即数在汇编时就已经确定了。因此，在汇编语言中，通常需要先将地址加载到寄存器中，然后再将数据存储到该地址中。

按照我们照葫芦画瓢的规律，看下面的代码也能发现这个输出操作：



指定格式 地址 间接寻址 输出

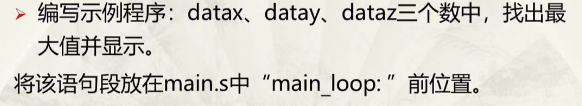
最后我用.end结束程序，下面的LED小灯显示的就不再运行了

这样我们就完成了这个简单的需求

窗体底端

Part\_02:比大小

要求：



思路：

* 定义变量：首先，我们需要定义datax、datay、dataz三个数的变量，以便能够在代码中引用。
* 比较大小：接下来，我们要比较这三个数的大小，找出其中的最大值。这里使用比较指令（cmp）来比较两个数的大小，并根据比较的结果来确定最大值。
* 显示最大值：一旦确定了最大值，我们就可以使用printf函数将其显示出来。在汇编中，我们需要加载printf函数所需的参数，并调用printf函数来进行输出。

事不宜迟，开写：

//（0.1.2）定义变量

// 定义数据段变量

.data

datax: .word 10 // 假设初始值为10

datay: .word 20 // 假设初始值为20

dataz: .word 15 // 假设初始值为15

//（2）======主循环部分（开头）=====================================

// 比较datax、datay、dataz中的最大值并显示

ldr r0, =datax // 加载datax的地址到寄存器r0

ldr r1, [r0] // 将datax的值加载到寄存器r1

ldr r2, =datay // 加载datay的地址到寄存器r2

ldr r3, [r2] // 将datay的值加载到寄存器r3

cmp r1, r3 // 比较datax和datay的大小

movgt r1, r3 // 若datay更大，则将datay的值赋给r1

ldr r2, =dataz // 加载dataz的地址到寄存器r2

ldr r3, [r2] // 将dataz的值加载到寄存器r3

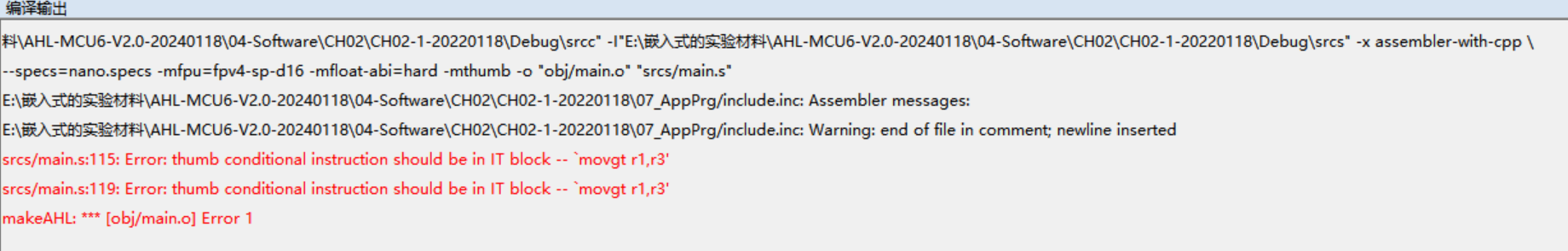
cmp r1, r3 // 再次比较r1（当前最大值）和dataz的大小

movgt r1, r3 // 若dataz更大，则将dataz的值赋给r1

ldr r0, =data\_format // 加载数据格式控制符的地址到r0

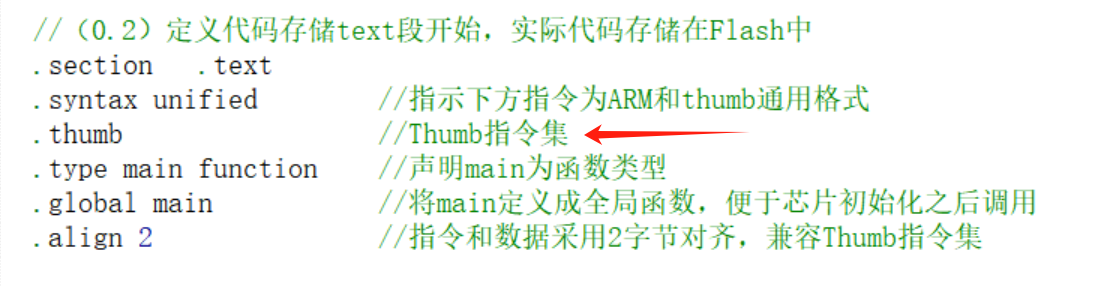
bl printf // 调用printf函数显示最大值

这里就用了r0 r1 r2 r3 三个常用的来比较，先比较前两个再跟第三个比，看样子不错，编译如下：



出了个小问题：IT block

这些错误是由于在Thumb指令集中的条件指令（例如movgt）未包含在IT块（If-Then指令）中引起的。在Thumb指令集中，条件指令必须位于IT块中。



指定了用thumb指令集，所以修改一下：

// 比较datax、datay、dataz中的最大值并显示

ldr r0, =datax // 加载datax的地址到寄存器r0

ldr r1, [r0] // 将datax的值加载到寄存器r1

ldr r2, =datay // 加载datay的地址到寄存器r2

ldr r3, [r2] // 将datay的值加载到寄存器r3

cmp r1, r3 // 比较datax和datay的大小

bgt datax\_is\_max // 若datax大于datay，跳转到datax\_is\_max标签

mov r1, r3 // 否则，将datay的值赋给r1

datax\_is\_max:

ldr r2, =dataz // 加载dataz的地址到寄存器r2

ldr r3, [r2] // 将dataz的值加载到寄存器r3

cmp r1, r3 // 比较r1（当前最大值）和dataz的大小

bgt max\_is\_r1 // 若r1大于dataz，跳转到max\_is\_r1标签

mov r1, r3 // 否则，将dataz的值赋给r1

max\_is\_r1:

ldr r0, =data\_format // 加载数据格式控制符的地址到r0

bl printf // 调用printf函数显示最大值

什么IT块？我不道啊？所以用一些投机取巧的方法，用了一个条件分支指令bgt，根据比较的结果跳转到相应的标签处，这样就绕过了IT块的限制。

**这里的出来的结果是正确的，20。我忘记截图了**

但是我想为了美观，加一行输出语句，提示一下输出了啥，

修改如下：

max\_is\_r1:

ldr r0, =output\_max\_msg // 加载输出语句地址到r0

bl printf // 调用printf函数显示“这里的最大值是：”

ldr r0, =data\_format // 加载数据格式控制符的地址到r0

bl printf // 调用printf函数显示最大值

// 数据段定义

output\_max\_msg:

.ascii "这里的最大值是：\n\0" // 最大值提示语句

想当然，各位是不是觉得一定是正确的，当然，事情没那么简单，这里的结果又变成了10 （不信可以试试看）

为啥呢？左思右想，我觉得是添加输出语句之后，结果变得不正确。原因是在显示输出语句后，重新加载了数据格式控制符的地址到r0，然后调用printf函数来显示最大值。但是，r0中已经加载了输出语句的地址，而且printf函数在显示输出语句后，会将r0寄存器的内容移动到内部寄存器，以准备下一个printf调用。因此应该在显示输出语句后，重新加载最大值到r1中，然后调用printf函数来显示最大值:

max\_is\_r1:

cmp r1, r3 // 比较r1（当前最大值）和dataz的大小

bgt exit\_max\_is\_r1 // 若r1大于dataz，跳转到exit\_max\_is\_r1标签

mov r1, r3 // 否则，将dataz的值赋给r1

exit\_max\_is\_r1:

ldr r0, =output\_max\_msg // 加载输出语句地址到r0

bl printf // 调用printf函数显示“这里的最大值是：”

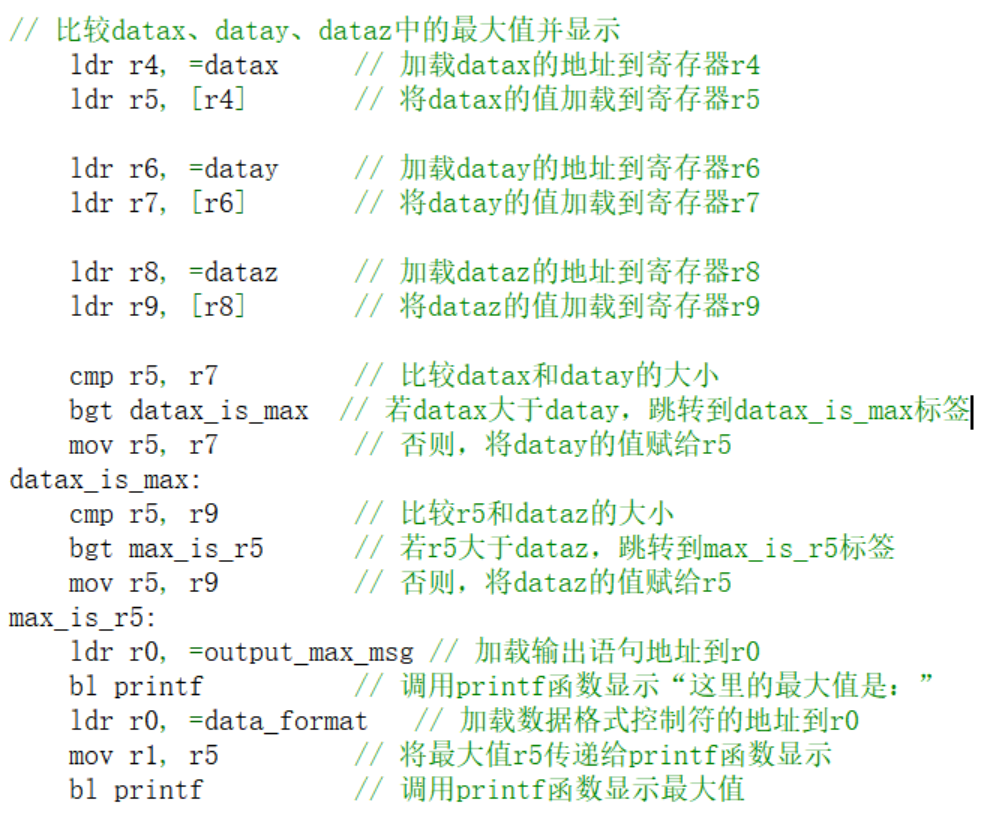
ldr r0, =data\_format // 加载数据格式控制符的地址到r0

mov r1, r1 // 将最大值r1传递给printf函数显示

bl printf // 调用printf函数显示最大值

嘿嘿，还是错的，我终于发现问题了，我把代码放loop里了，意思就是说，跟上面的累加的代码一起了，所以在输出了那个提示语句之后再加载寄存器里的值再输出，就不知道混到了哪个值塞到了r1里面，一直找不到正确值（我没找到这个ide的值跟踪调试在哪），所以我用了一个最简单粗暴的办法：用别的没污染过的寄存器4 5 6 7 8 9来装这仨的值

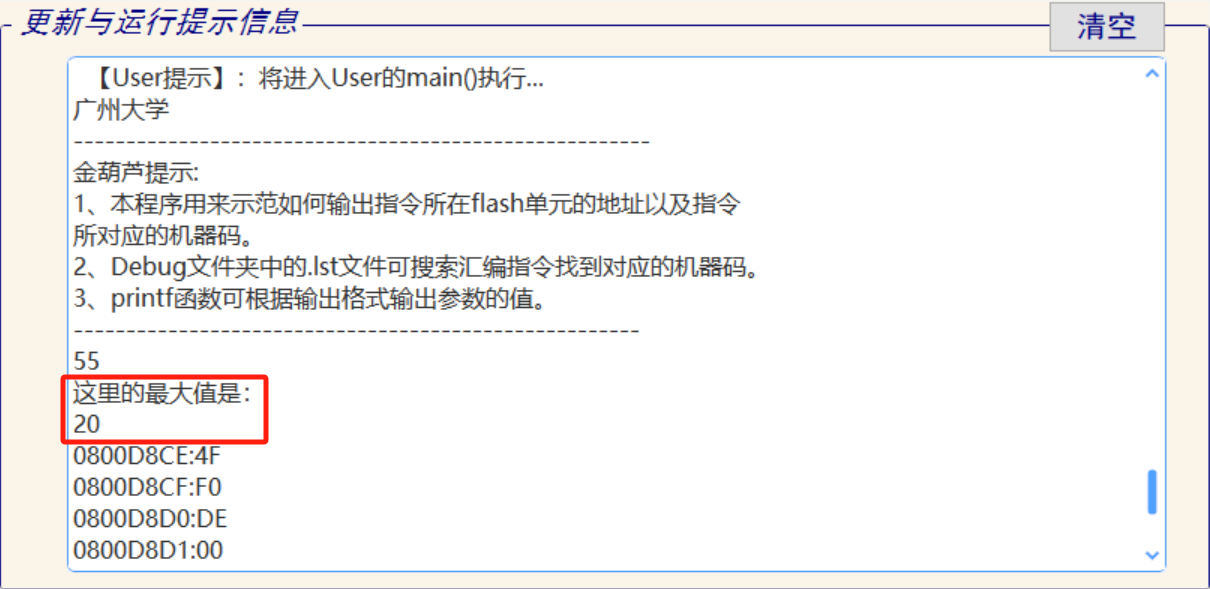
贴一段代码:



这个时候再编译，串口更新，运行一看，行了！

这样就简单明了了，xy比较一下，大的就存r5，再拿r5和z比较一下，r5大就输出，否则把z给r5，再输出，这里就没有别的寄存器干扰

正确的结果：



完成任务!