



**《微机原理》实验报告**

姓名：曹建钬

学号：20375177

# 实验内容

**（一）题目1**

将两位 16 进制数 0x5c 中的每一位分别转换为 ASCII 码，并将结果存入 RAM 中。（0~9 转换为 ASCII 码时加 0x30，A~F 转换为 ASCII 码时加 0x37）

Implement code 如图1：



图 1：题目1的Implement Code

思路为用寄存器r1存16进制数0x5c的高位0x05，r2存低位0x0c，分别转换为ASCII码0x35和0x43，再将其合并为0x3543存入RAM中，调试过程如下：

* 进入复位过程：

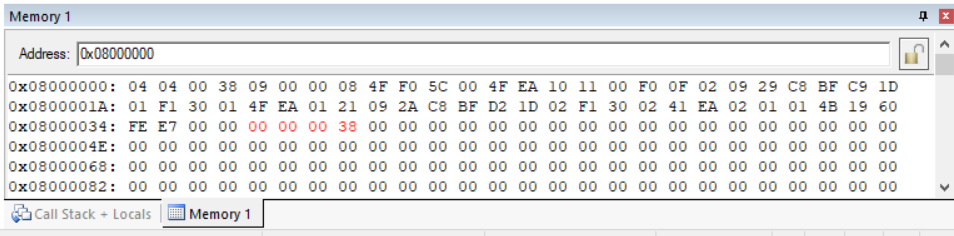


图 2：复位过程Memory窗口查看address

根据复位过程，从图2可以看出,处理器从0x00000000（0x80000000）处读取第一个字0x38000404赋给MSP，从0x00000004（0x80000004）处读取第二个字0x08000009赋给PC。查看Register窗口中的MSP寄存器的值（如图3）为0x38000404 ，说明主栈指针的初始值加载正确。而0x08000009的bit<0>为1标识该指令为Thumb指令，实际指令地址为偶地址0x08000008与Register窗口中的PC值一致，说明程序计数器加载正确。

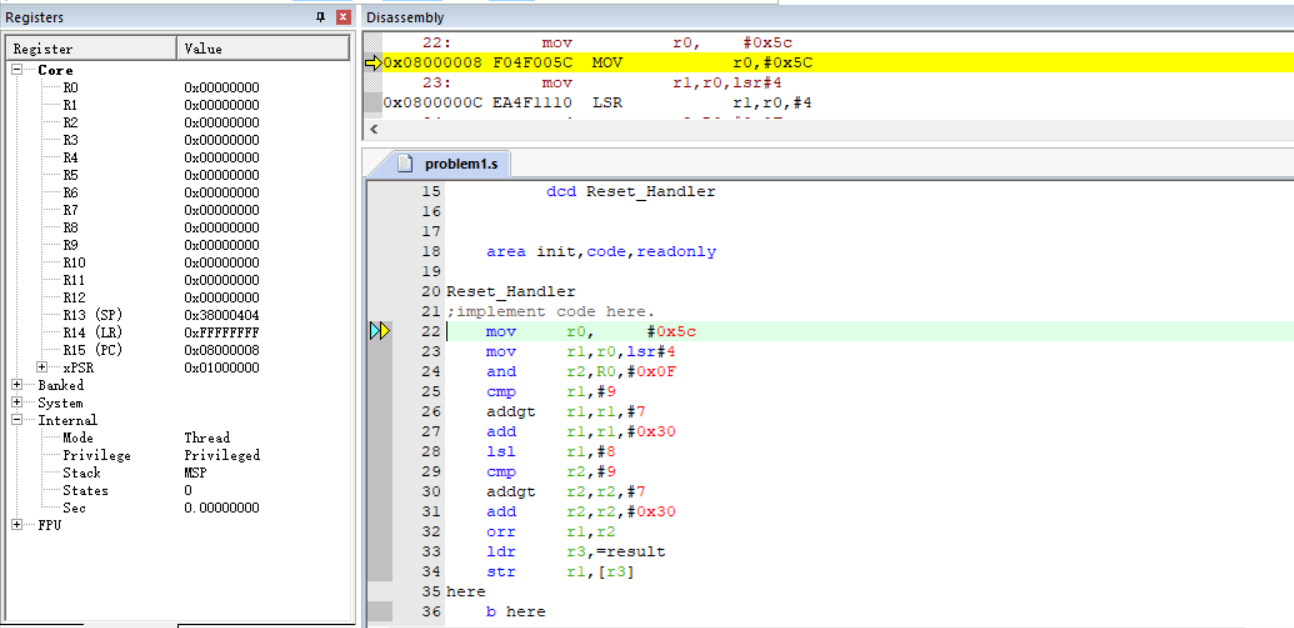


图 3：复位过程Register窗口

* 单步执行第22行数据传送指令，将r0赋值为两位16进制数0x5c，如图4所示：

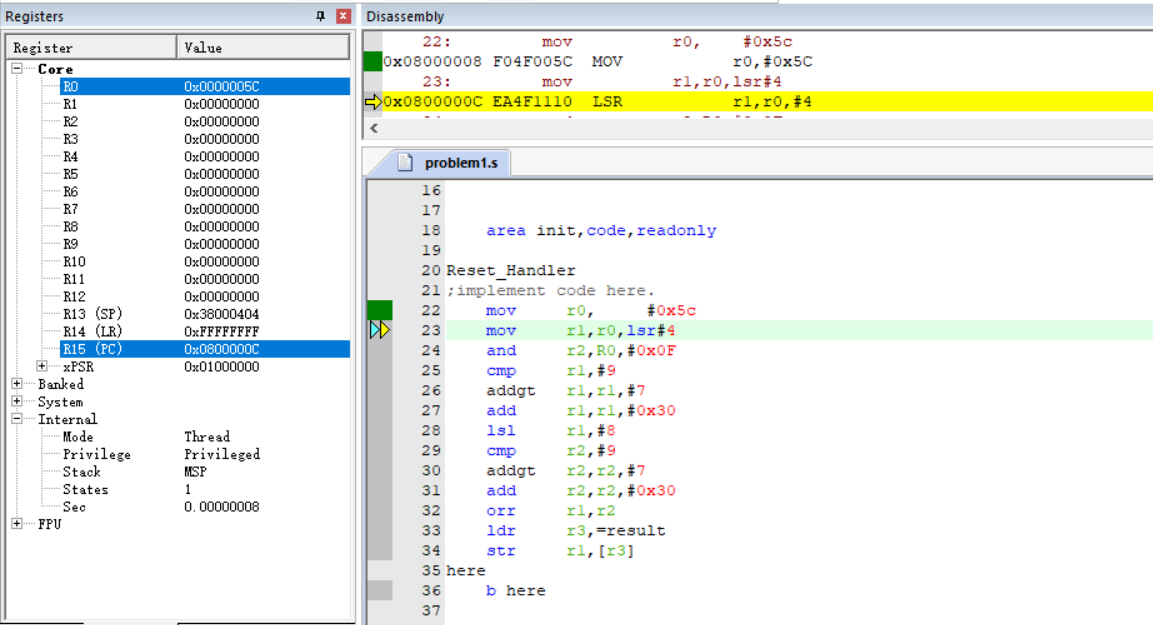


图 4：单步执行第22行指令

* 单步执行第23行数据传送指令，将r0中的数据0x5c逻辑右移4位赋给r1，r1存储0x5c高位：0x05，如图5所示：

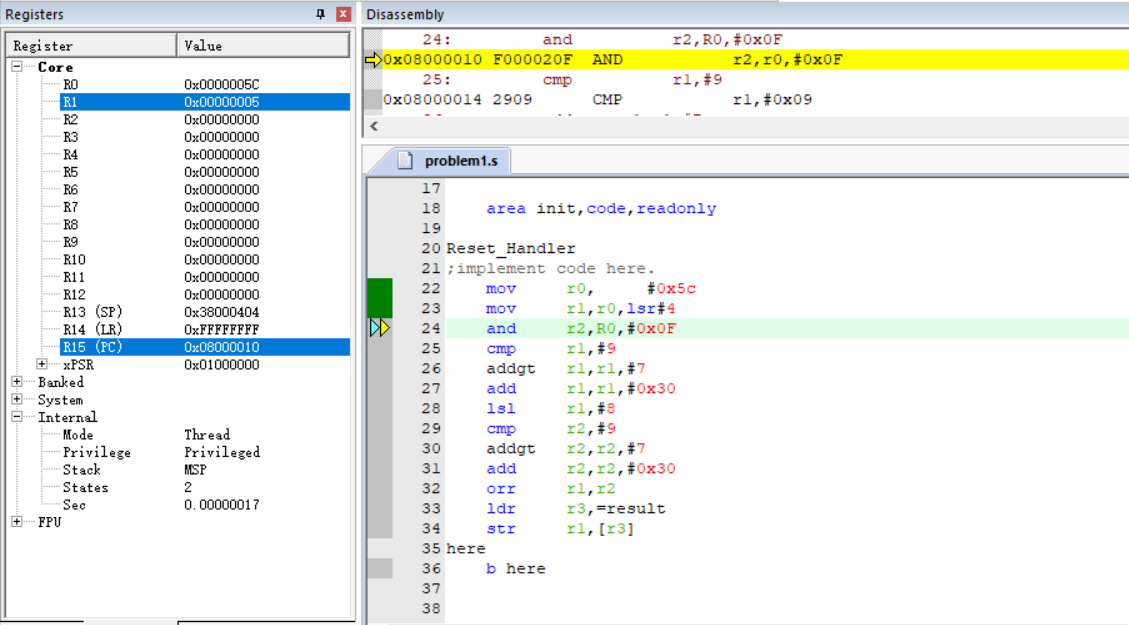


图 5：单步执行第23行指令

* 单步执行第24行逻辑与指令，将r0&0x0F赋给如r2，r2存储0x5c低位0x0c，如图6所示：

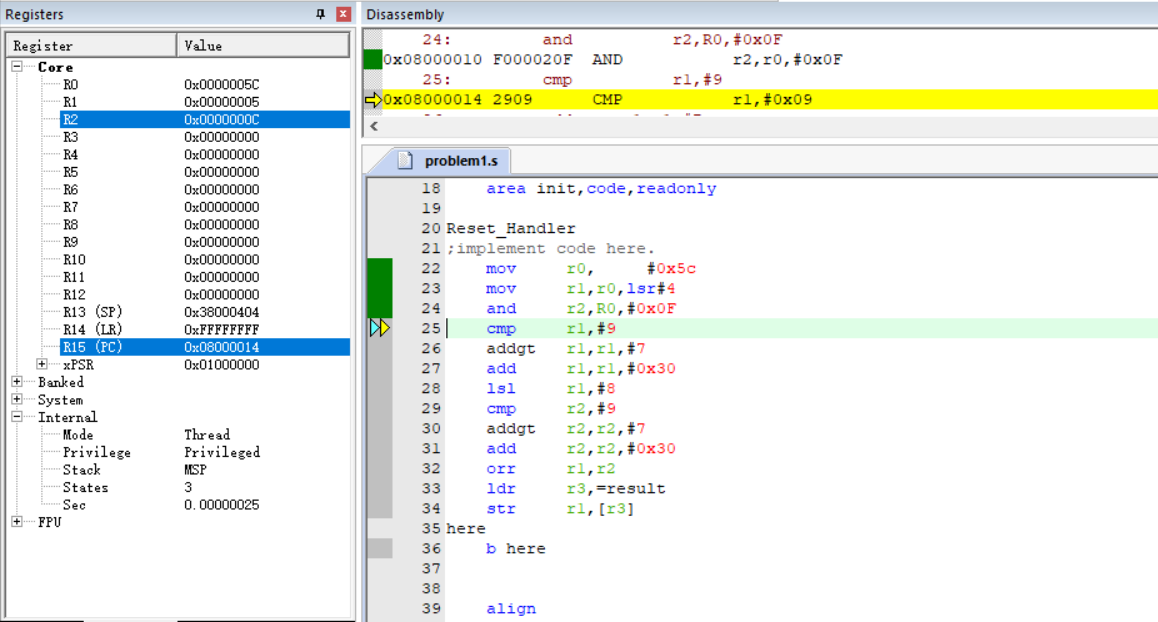


图 6：单步执行第24行指令

* 单步执行第25行比较指令，计算r1-9，APSR更新，N置0，但计算结果不保存，如图7所示：

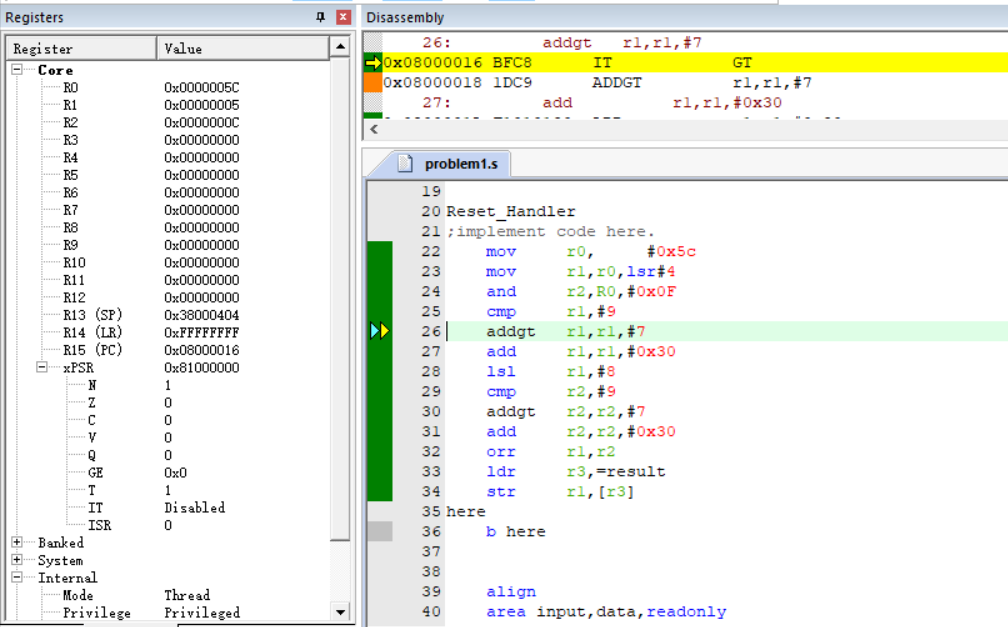


图 7：单步执行第25行指令

* 单步执行第26行比较指令，如果r1比9大则加7，在这里r1比9小则保持原数值，如图8所示：

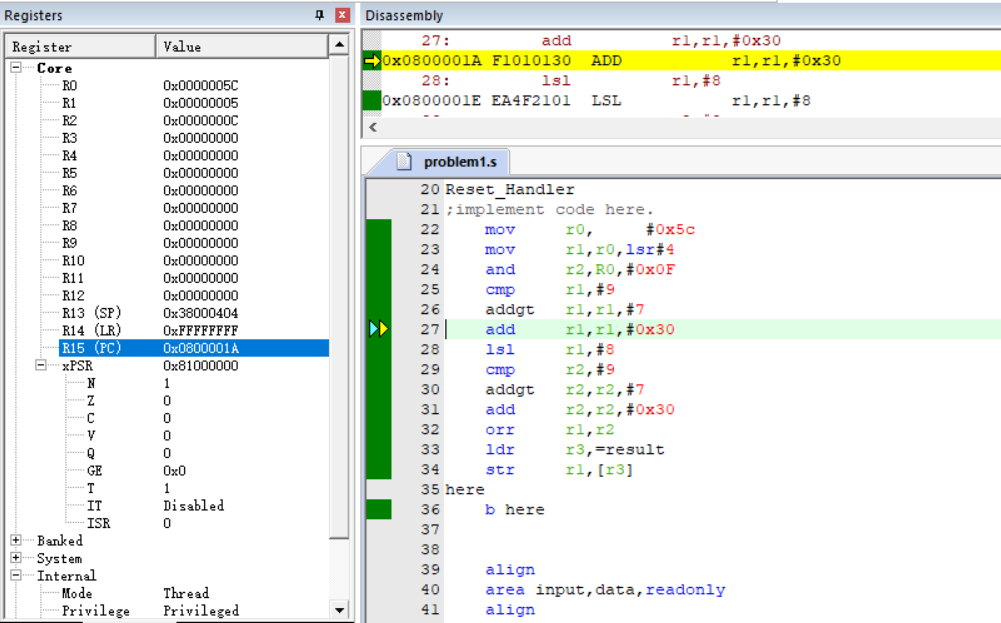


图 8：单步执行第26行指令

* 单步执行第27行指令，r1增加0x30得到0x35，如图9所示：

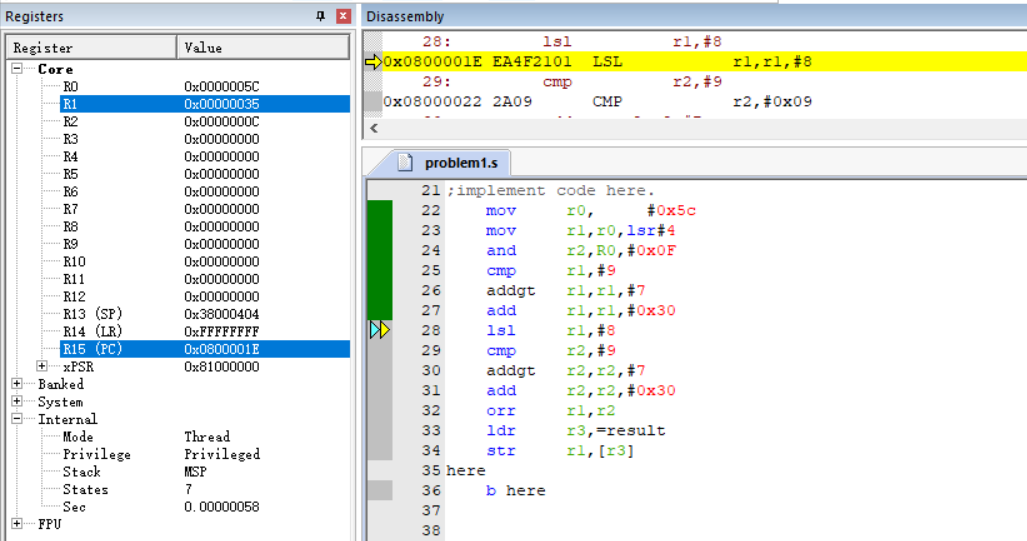


图 9：单步执行第27行指令

* 单步执行第28行逻辑左移指令，0x35左移8位得0x3500，如图10所示：

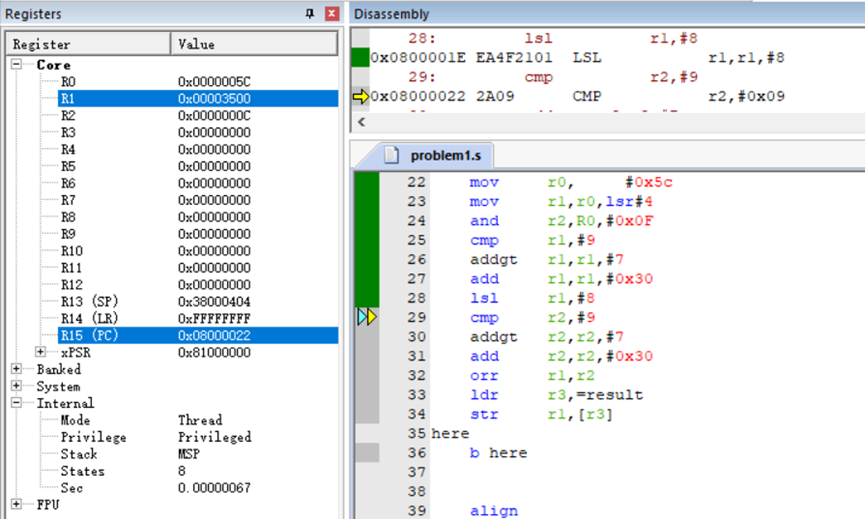


图 10：单步执行第28行指令

* 第29-31行指令与第25-27行实现的功能一致——0~9 加0x30转换为 ASCII 码，A~F 加7再加0x30转换为 ASCII 码，0x0c转为ASCII码为0x43，如图11所示：

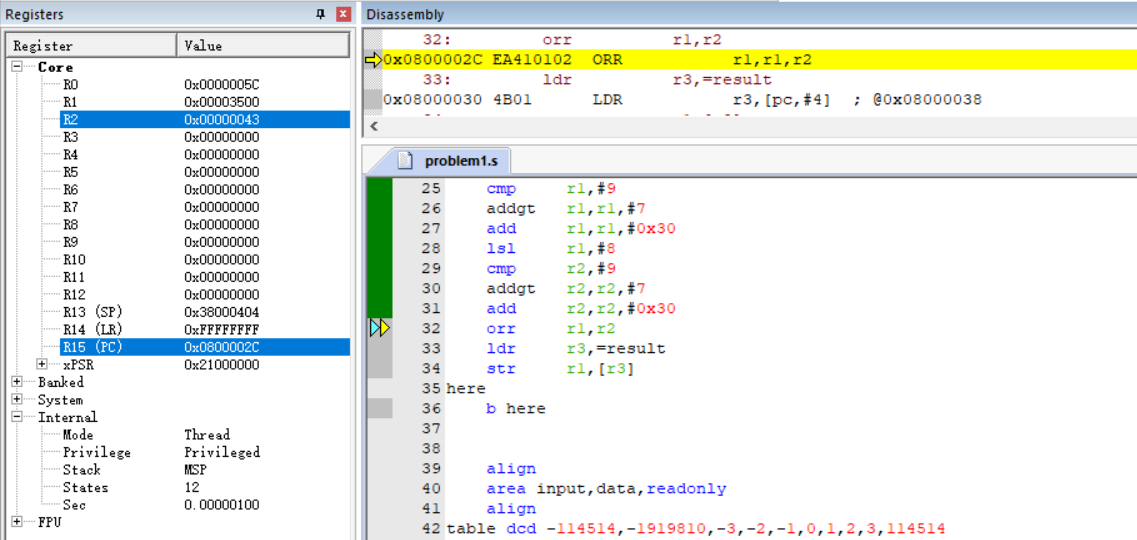


图 11：执行完第31行指令

* 第32行指令执行逻辑或运算，将0x5c高位和低位转成的8位ASCII码结合为0x3543并存储于r1，如图12所示：

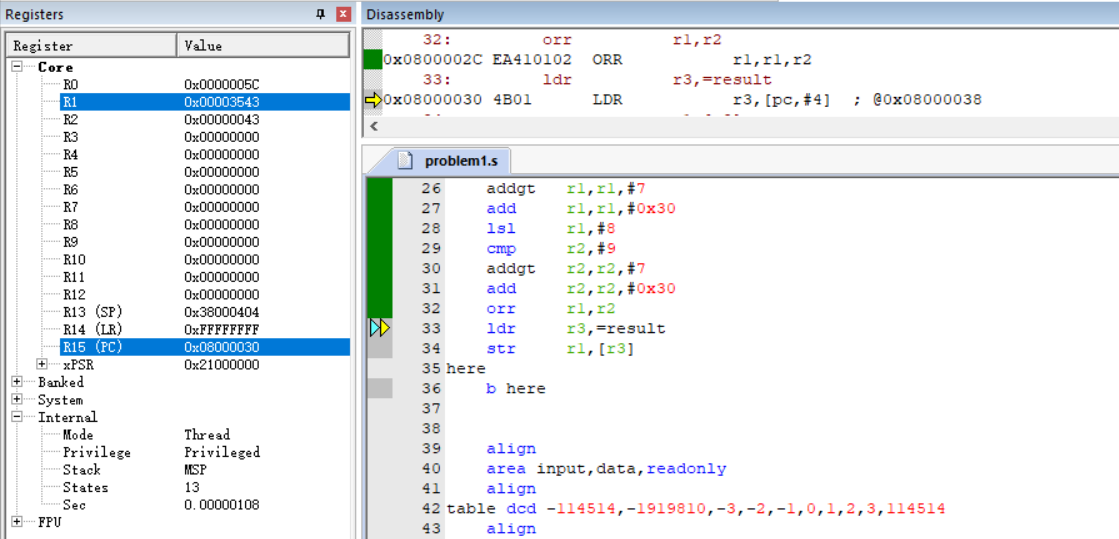


图 12：单步执行第32行指令

* 单步执行第33行指令，找到存放result（分配得一块连续的字存储单元）的首地址0x38000000并赋值给r3，如图13所示：

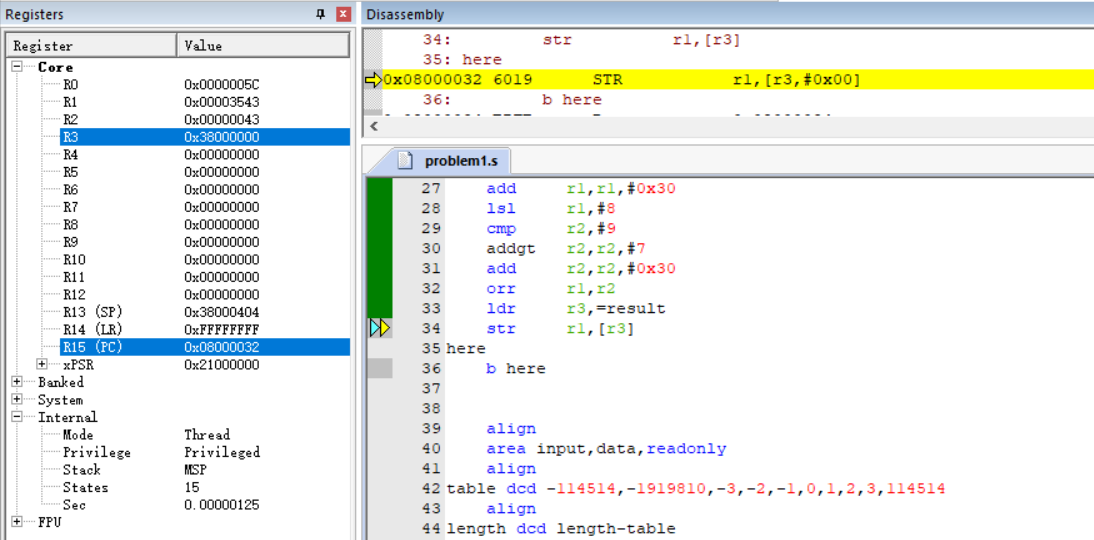


图 13：单步执行第33行指令

* 执行第35行指令，将最终结果存入result，即保存在RAM中。在Memory窗口搜索地址0x38000000，搜索结果如图14所示，为0x00003543，实现实验内容1的要求。

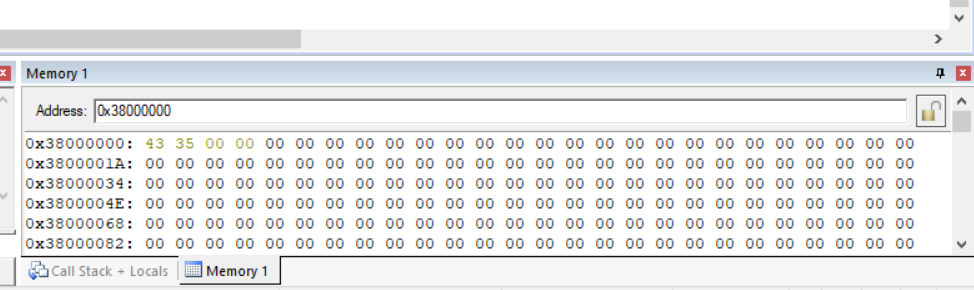


图 14：Memory窗口查看address

**（二）题目2**

求两个32位有符号数的最大值并将结果存入 RAM

Implement code 如图15：

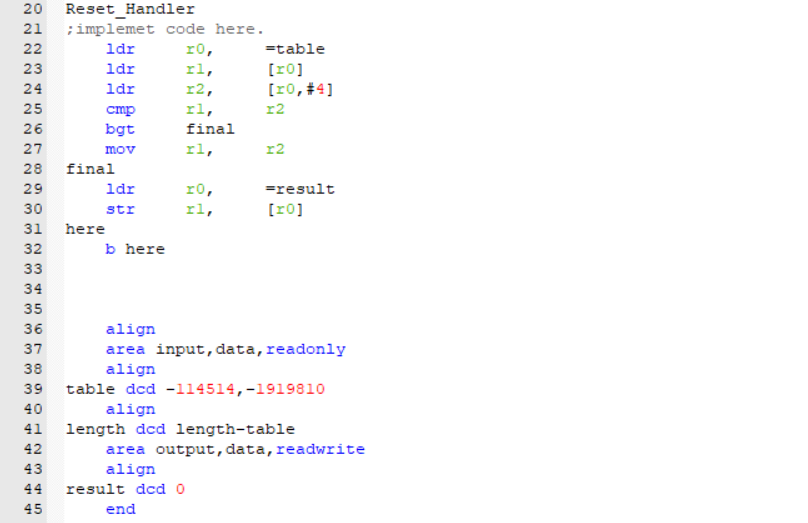


图 15：题目2的Implement Code

思路为分别用寄存器r1和r2存储这两个32位有符号数，如果r2中的值大于r1，则把r2的值赋给r1，否则保持原值，这样r1中的数就是两个数中的最大值，两个32位有符号数所在存储区域的首地址的标号为table，而两个32位有符号数的最大值所存放在的RAM区域的首地址的标号为result。调试过程如下：

* 复位后如图16所示

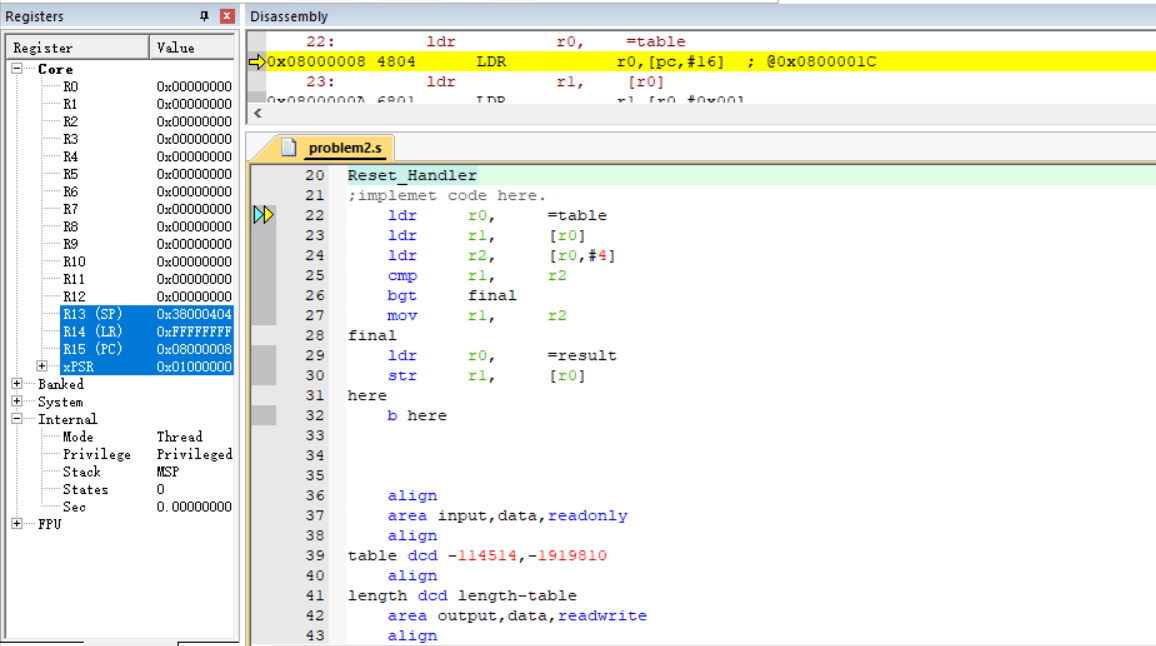


图 16：复位

* 第22行到第24指令将输入两个32位有符号数从IROM中取出来并把值分别赋给r1和r2，从图17和图18中可以看出r1和r2赋值成功，分别为0xFFFE40AE（-114514）和0xFFE2B4BE（-1919810）

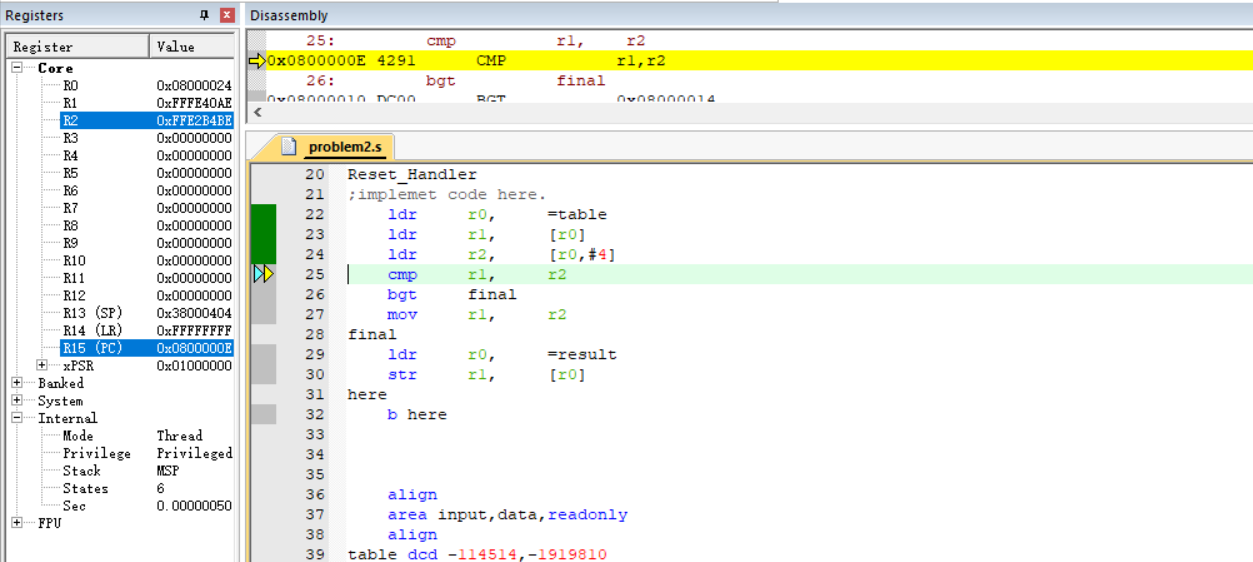


图 17：执行完第24行指令

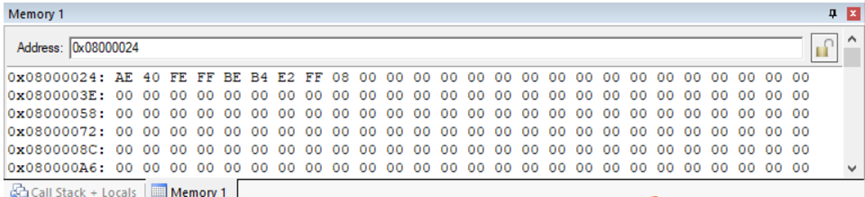


图 18：Memory查看table地址

* 第25行和第26行指令实现选择结构，比较r1和r2中的数值，如果r1比r2大则跳过第27行指令（将r2中的值赋给r1）转到final标号（第28行），这里-114514大于-1919810因此执行26行指令后直接挑战至final且r1保持不变，如图19所示：

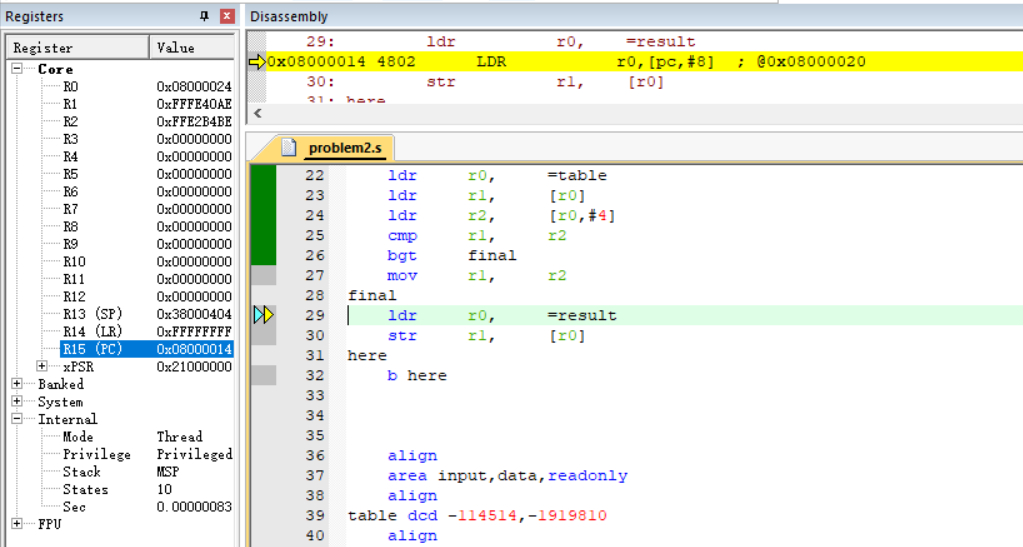


图 19：执行完25和26行指令

* 第28行至第30行指令实现将r1存入RAM中，查看result的首地址0x38000000，读字为FFFE40AE，说明更大的值-114514已经成功放入RAM中，如图20和图21所示：

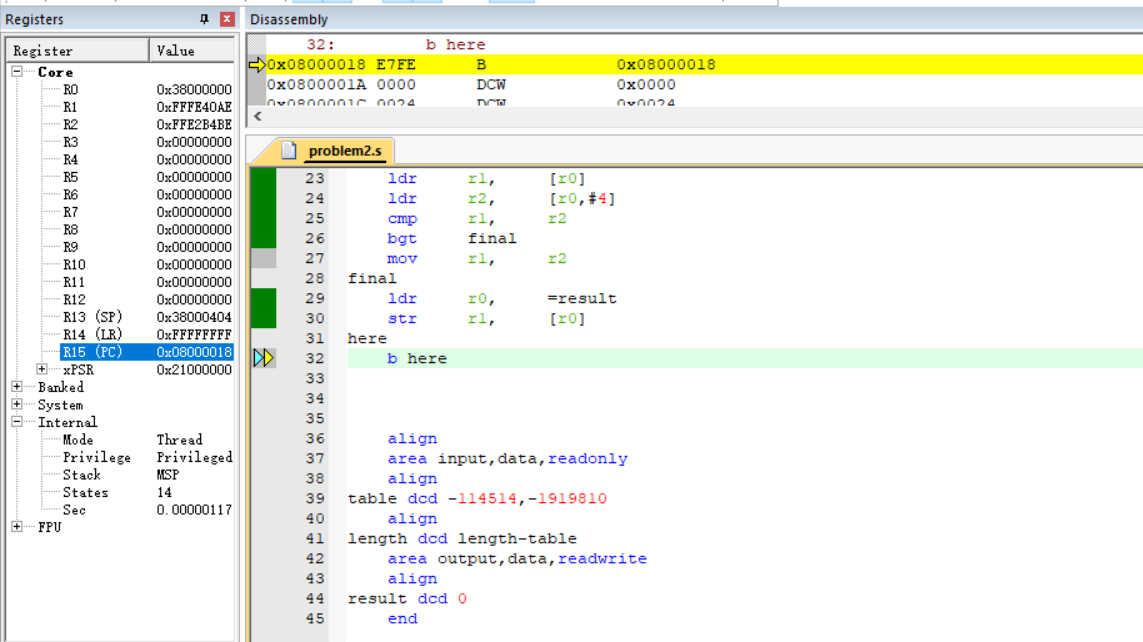


图 20：执行完第30行指令

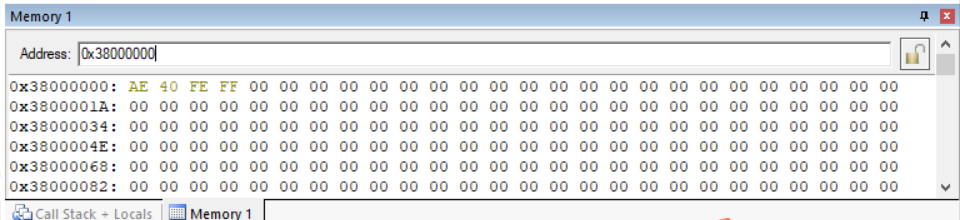


图 21：Memory查看result首地址

**（三）题目3**

任意给定一个 32 位有符号整数 x，实现以下表达式，将结果存入RAM：

x<-10 时，输出结果为-1

x> 10 时，输出结果为 1

-10<=x<=10 时，输出结果为 0

求两个 32 位有符号数的最大值并将结果存入 RAM

Implement code如图22：

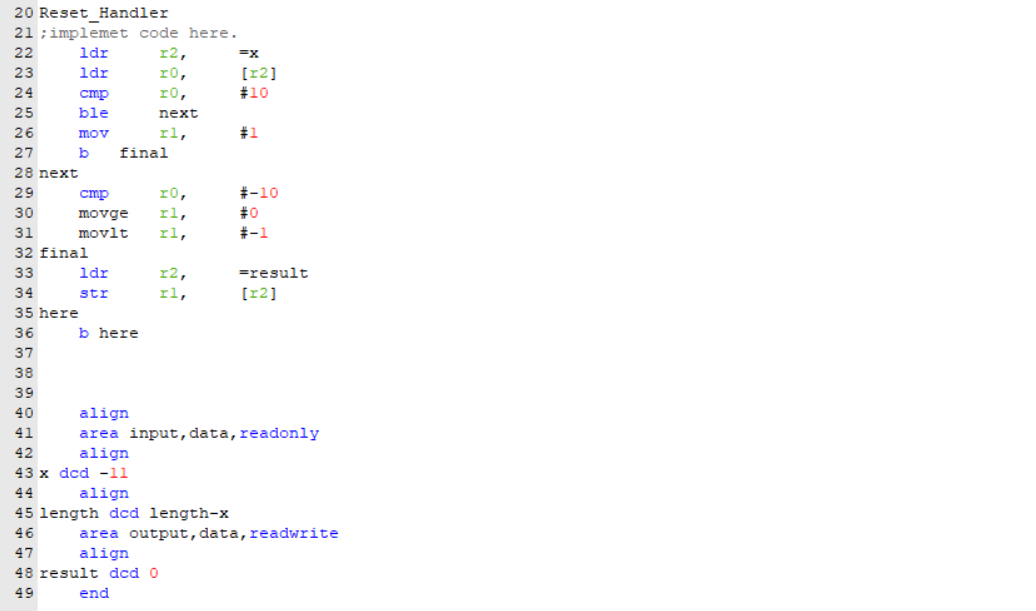


图 22：题目3的Implement code

思路为通过r2从flash中取出x标号下的有符号整数，将该整数存入r0，确定r0的范围并将输出结果记录在r1中，最后再通过r2将结果放入RAM中。

调试过程如下：

* 复位后如图23所示：

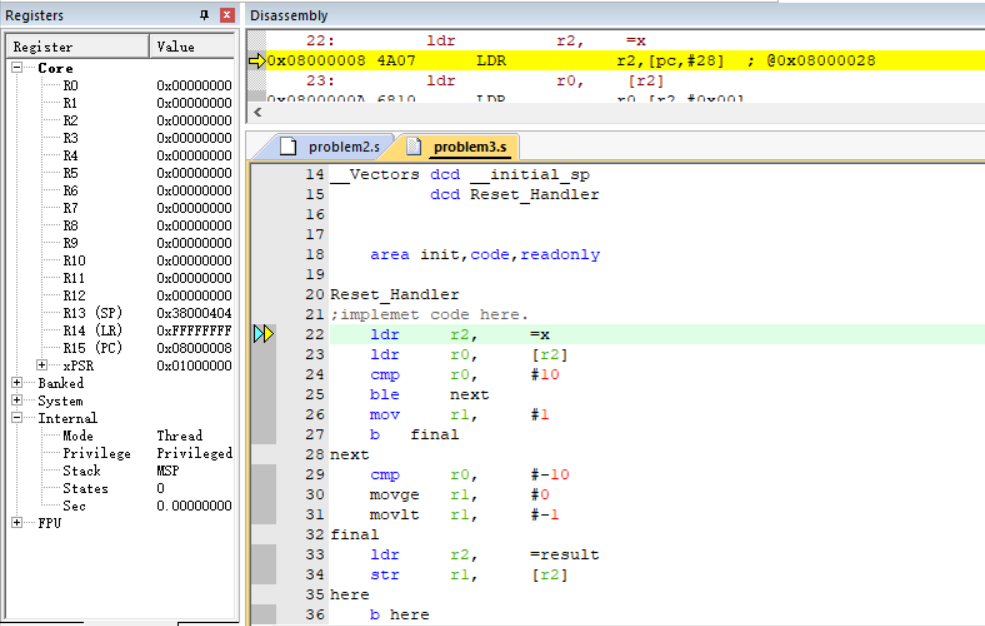


图 23：复位

* 执行第22和23行的指令后，r2存放x的首地址，并从0x08000030读取一个字到r0中，如图24和25所示，-11被成功赋给了r0

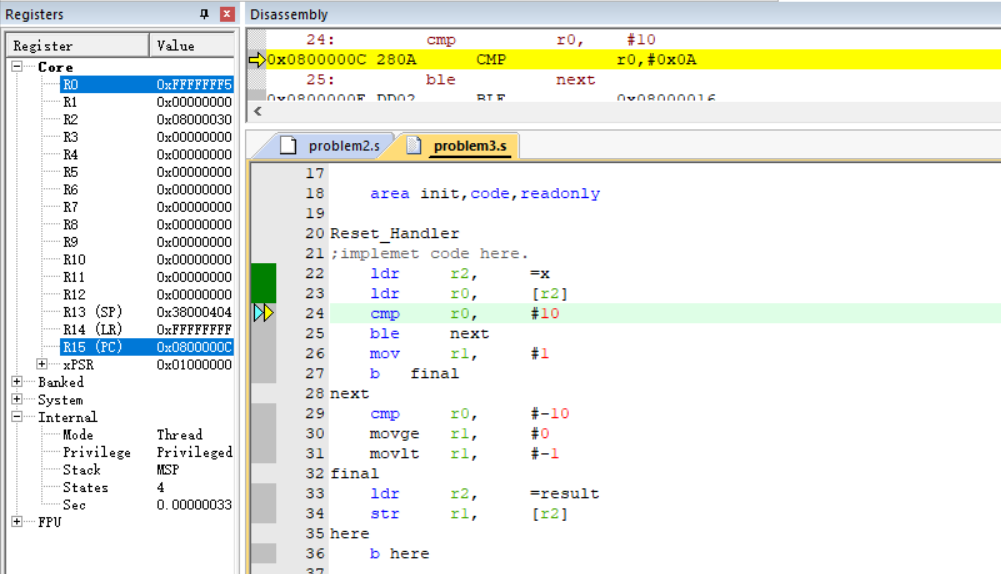


图 24：执行完第23行指令

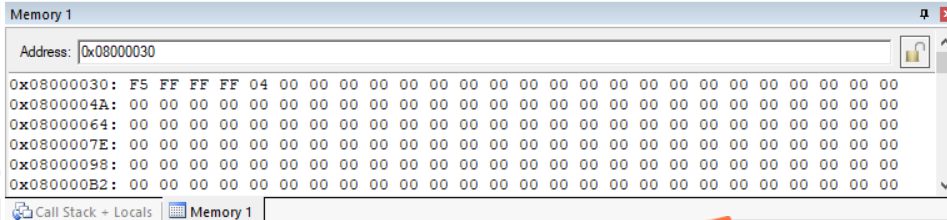


图 25：通过Memory查看地址

* 第24到第27行的指令为选择语句，比较r0和10，如果r0小于或等于10则跳转至标号next（第28行），如果r0大于10则执行26-27行的指令：将结果1赋给r1并跳转至标号final。这里r0为-11，小于10，则跳转至next标号，如图26所示：

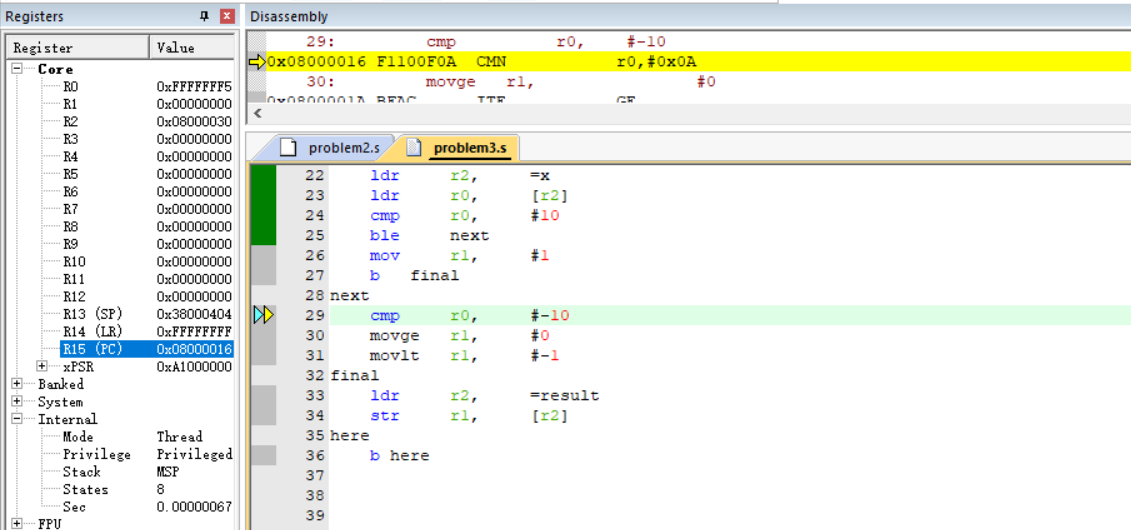


图 26：执行完第24-25行指令后跳转至next

* next标号下的第29-31行指令为比较r0和-10的大小，如果r0大于或等于-10则将结果0赋给r1，如果r0小于-10则把-1赋给r1。这里-11小于10，故r1中的值为0xFFFFFFFF（-1），如图27所示：

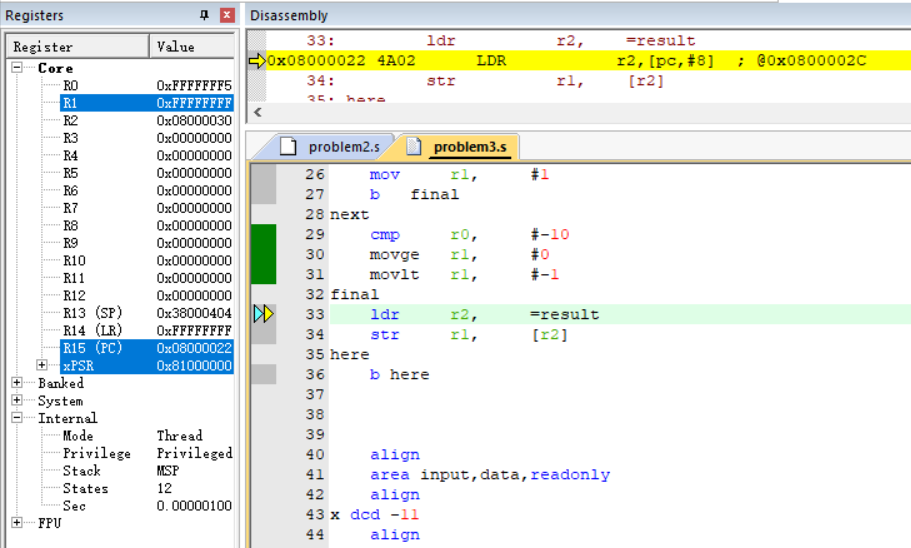


图 27：执行完第29-31行指令后

* final标号下的第33-34行指令为将result的首地址赋给r2并借助r2将r1存储的输出结果存入result，如图28所示。而从memory窗口中搜索result首地址后可看出最终结果（-1）已经顺利存入RAM中，如图29所示。

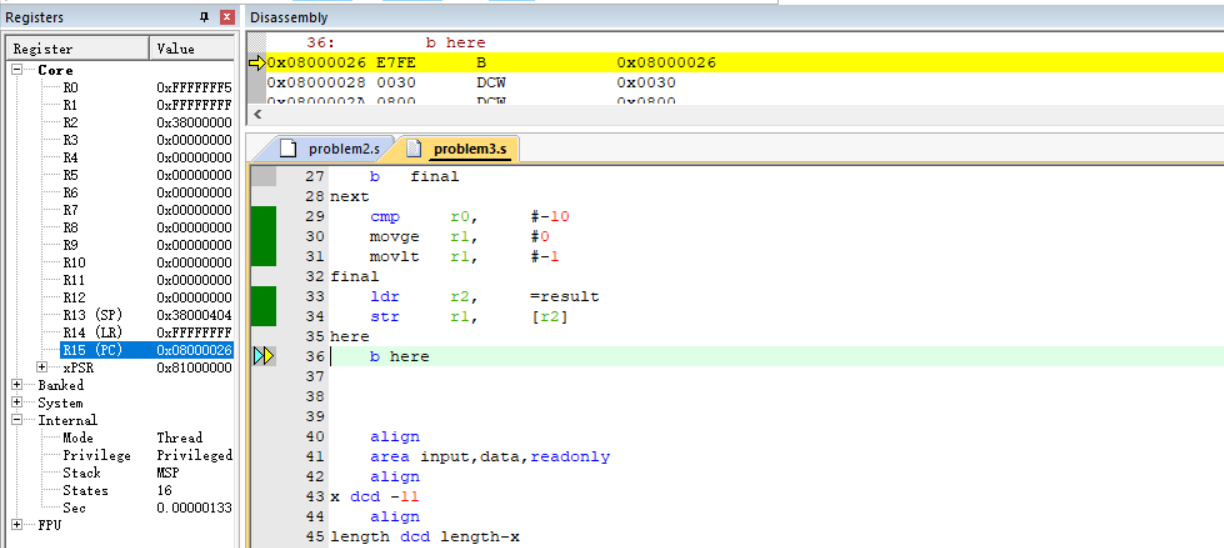


图 28：执行完第34行指令

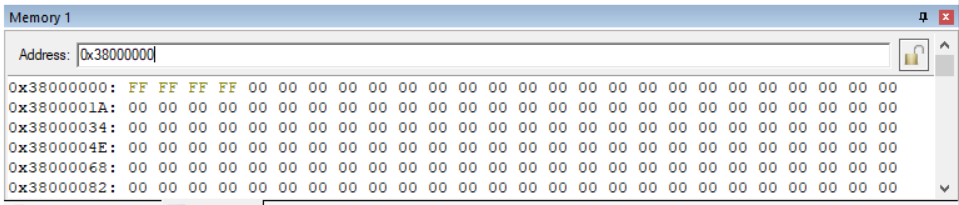


图 29：Memory窗口搜索地址

**（四）题目4**

计算字符串 String 的长度，String 以回车符结尾，回车符不计算在长度内（回车符”\r”，例如： DCB “Hello World!\r”）

Implement code如图30：

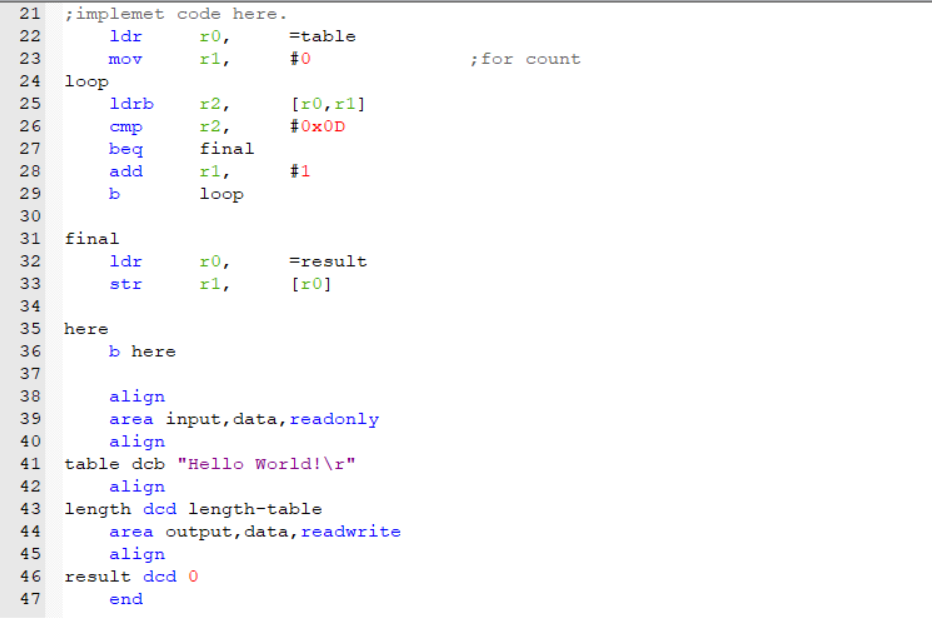


图 30：题目4的Implement Code

思路为：字符串存储在内存单元中时按字节存储每个字符，那么可以从第一个字符开始一个一个字符进行读取并记录共读取了多少了字符，直到读取到回车符——查ASCII码表可知其存储在字节单元中为0x0D。此时便可输出字符串的长度。

实现方法为：将要计算长度的字符串String存放在table标号下一片连续的字节存储单元中，最后计算得到的长度存放在result标号下的一片连续的字存储单元中，寄存器r0用于取table和result首地址，寄存器r1用于记录已经读取过的字符数，寄存器r2按字节遍历String，当r2遍历到0x0D时停止并将r1作为返回值存入RAM中。

调试过程如下：

* 复位后如图31所示：

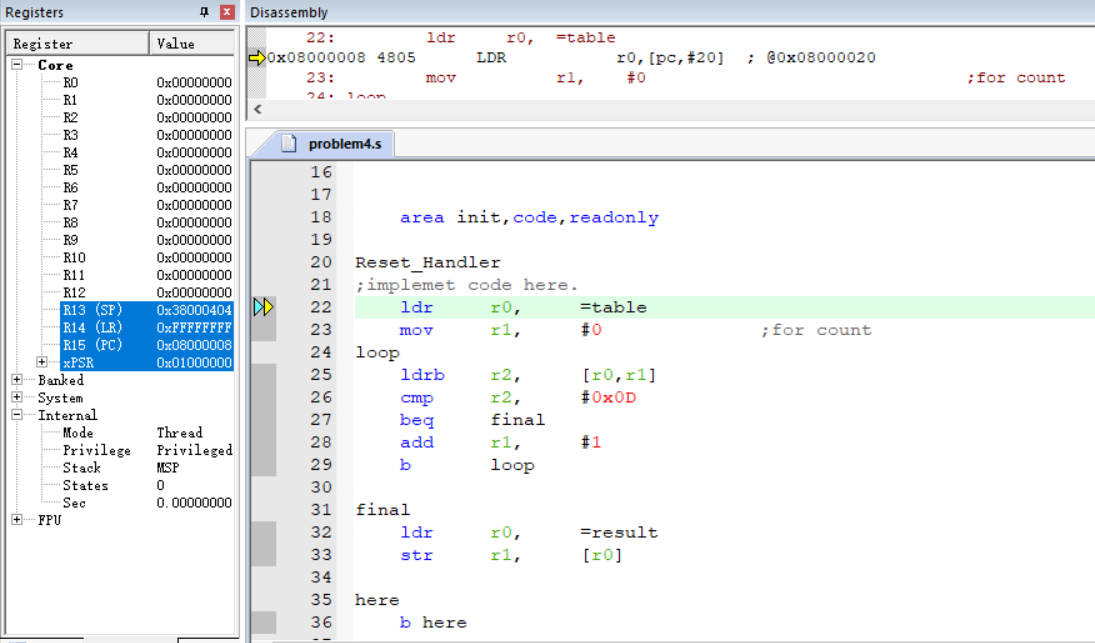


图 31：复位

* 如图32所示，单步执行第22行的指令将table首地址（0x08000028）存入r0。如图33所示，通过Memory窗口查看地址0x08000028，可发现String的第一个字符所在地址已经被成功存入r0。



图 32：单步执行第22行指令

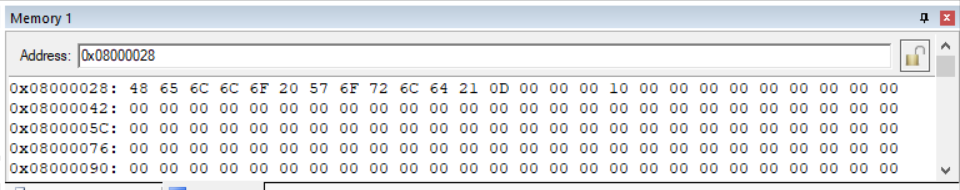


图 33：通过Memory窗口搜索地址

* 单步执行第23行指令，将r1赋初值0，如图34所示。

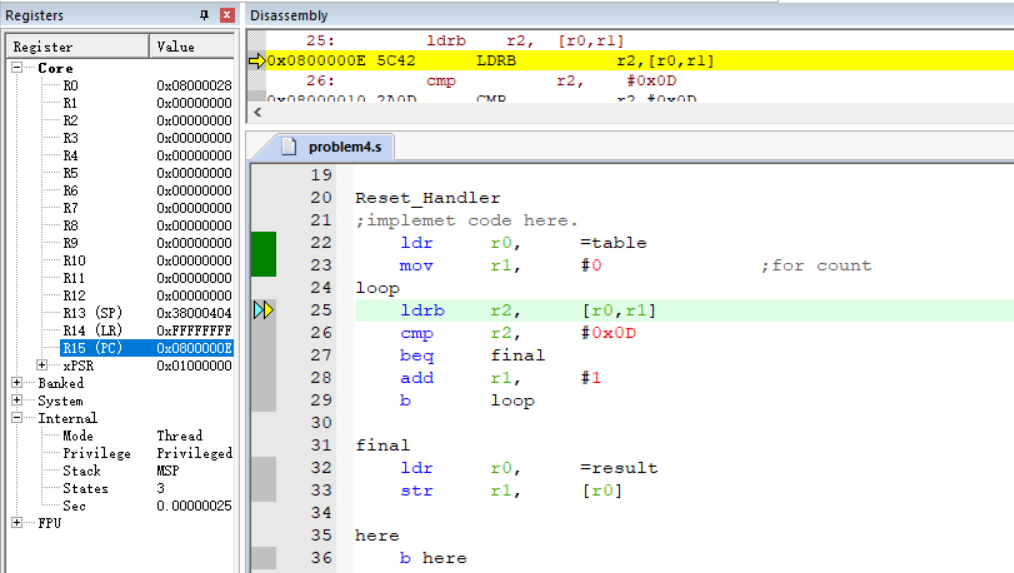


图 34：单步执行第23行指令

* 第24-29行指令实现遍历字节的过程，每次循环先判断当前读取到的字符是否为回车符（第26行指令），如果为回车符则结束循环进入final，否则r1自增1并再次执行循环体。在这里第一次执行循环体时遍历的第一个字符为H，因此r1自增1后变为0x00000001，如图35所示；结束循环时r1增加到12时，如图36所示，此时r2遍历到0x0000000D，符合预期。

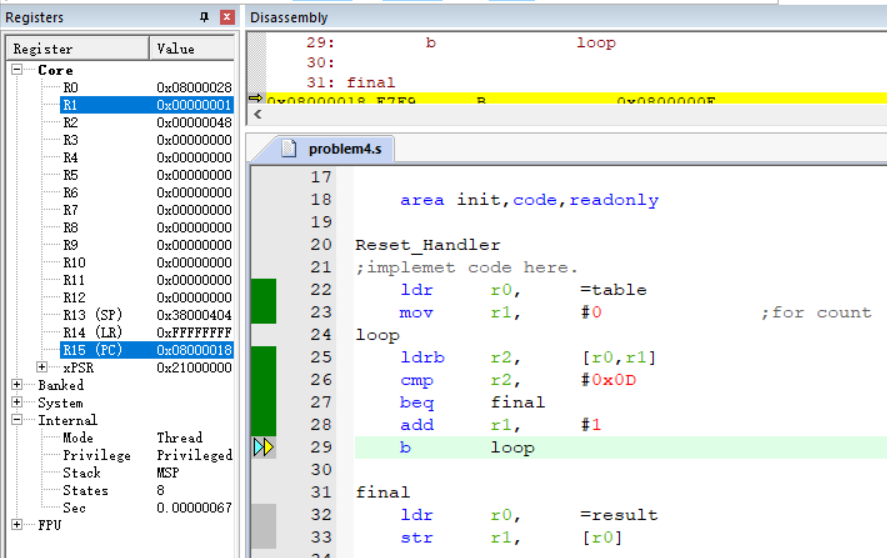


图 35：第一次循环结束

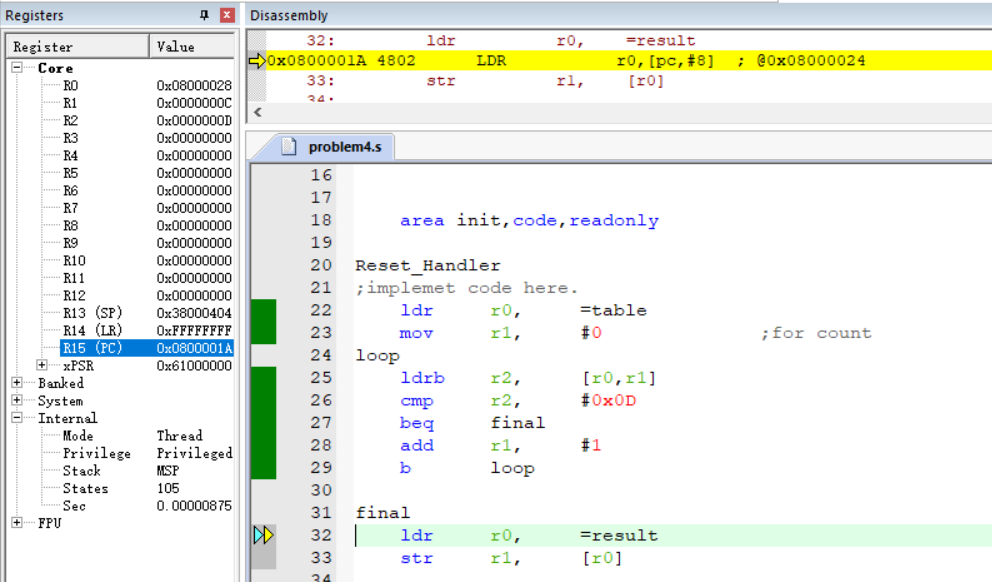


图 36：最后一次循环结束

* 执行第32-33行指令，将r1存入RAM中，如图37、38所示，最终字符串的长度12被成功放在了地址0x38000000处。

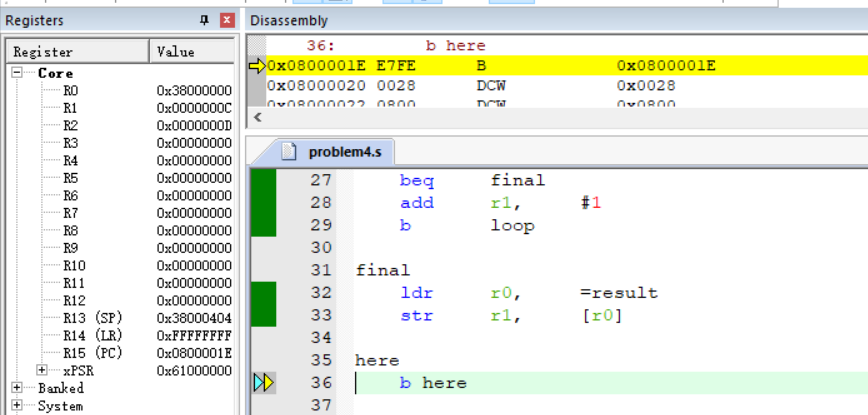


图 37：执行完32、33行指令

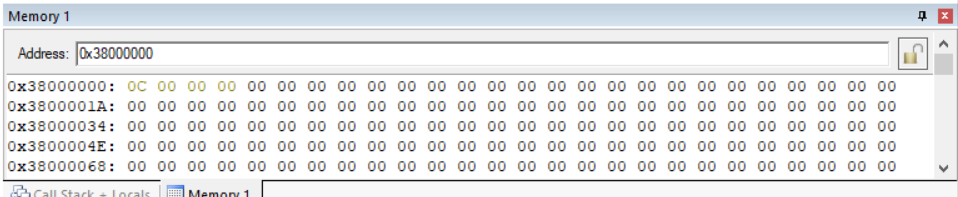


图 38：通过Memory窗口查看地址

**（五）题目5**

求一组 32 位有符号数的最大值，并将结果保存在RAM中

Implement Code如图39所示：

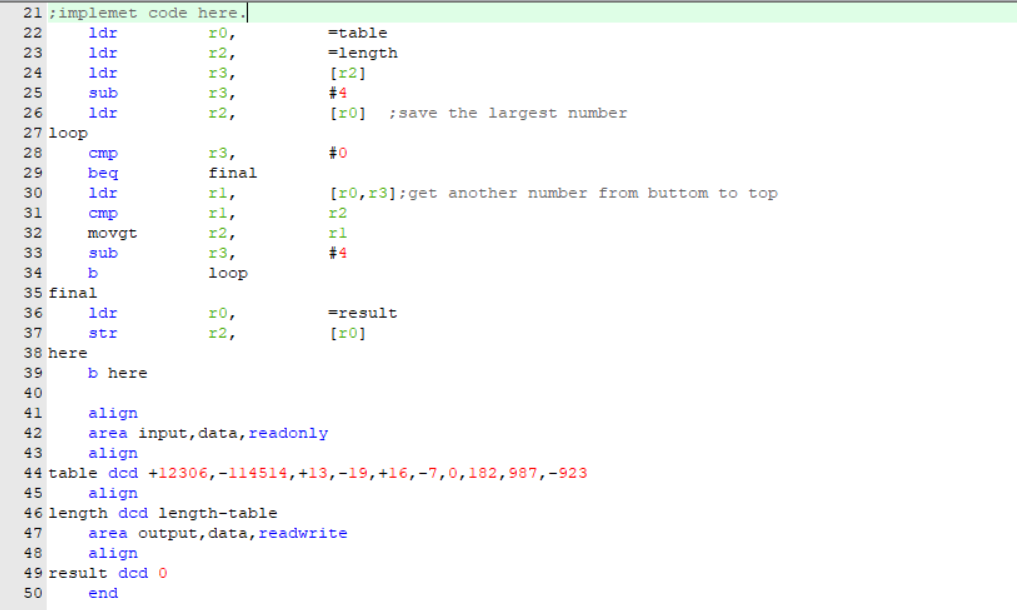


图 39：题目5的Implement Code

思路为：寻找一组32位有符号数的最大值只需要将所有的数遍历一遍即可——每次只取一个数与当前最大值比较，如果所取得的数更大，则更新当前最大值，当所有的数都取过一遍后，留下来的数便为该组数的最大值。

实现方法为：将一组32位有符号数存入标号为table的一片连续的字存储单元中，table所分配的字节数length-table存入标号length下的连续字存储单元，寄存器r0用于取table和result首地址；寄存器r2先辅助r3取得32有符号数组所占字节数后用第一个数进行初始化。使用循环结构实现两两比较大小的功能，r3用于计数、判断循环条件和充当寄存器移位寻址的偏移量offset，每次循环从后往前取值并赋给r1，r1中的数与r2中的数比较大小并将两数中的最大值赋给r2，当r1取到第2个数并比较后得到的r2中的值便为这组32位有符号数的最大值，将r2中的值存入result即可。

调试过程如下：

* 复位后如图40所示：

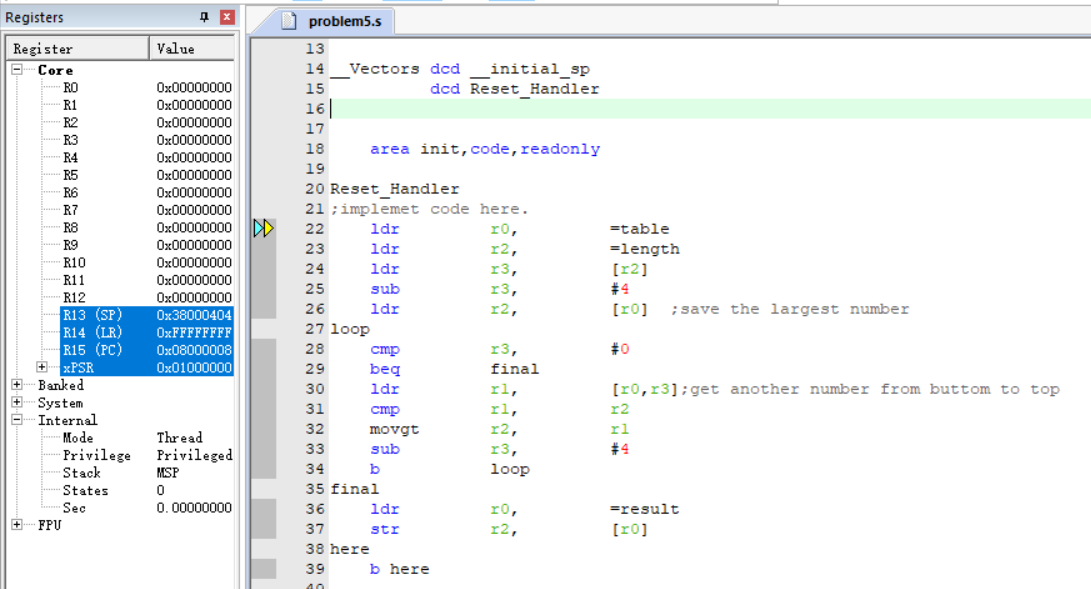


图 40：复位

* 如图41所示，执行第22行和第23行的指令后，将table首地址（0x08000038）存入r0，将length首地址（0x08000060）存入r2。



图 41：执行第22-23行指令

* 如图42所示，第24- 26行的指令将r2和r3初始化，r2保存第一个数，r3保存[r0,#0ffset]能取得最后一个数时的偏移量（因此需要在总字节数的基础上减去4）。如图43，查看Memory窗口，可验证第一个数0x00003012(+12306)被成功存入r2，00000028Byte=10字，为输入的数的个数。

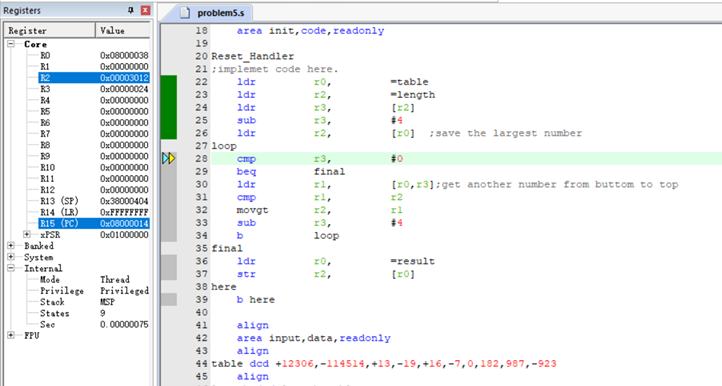


图 42：执行至第28行指令

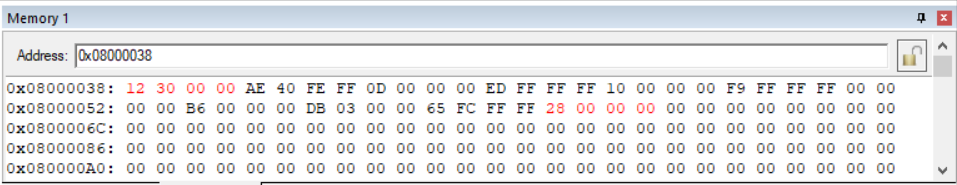


图 43：通过Memory窗口查看地址

* 第27-34行指令为循环结构，第28行指令：比较r3和0；第29行指令：如果r3等于0（此时第二个数已经取过了，同时第一个数在初始化r2时也被遍历过，此时则不用再进行取数比较）则跳转至final；第30行指令：寄存器移位寻址，取数赋给r1；第31行指令：比较r1和r2；第32行指令：如果r1大于或等于r2，则把r1赋值给r2，这样r2中的数便为r1和r2中的最大值；第33行指令：r3自减4，可保证下一次循换取前一个数；第34行指令：再次执行循环体。执行第一次循环后如图44所示，r1（-923）小于r2（+12306），因此r2保持不变，依旧为0x00003012。

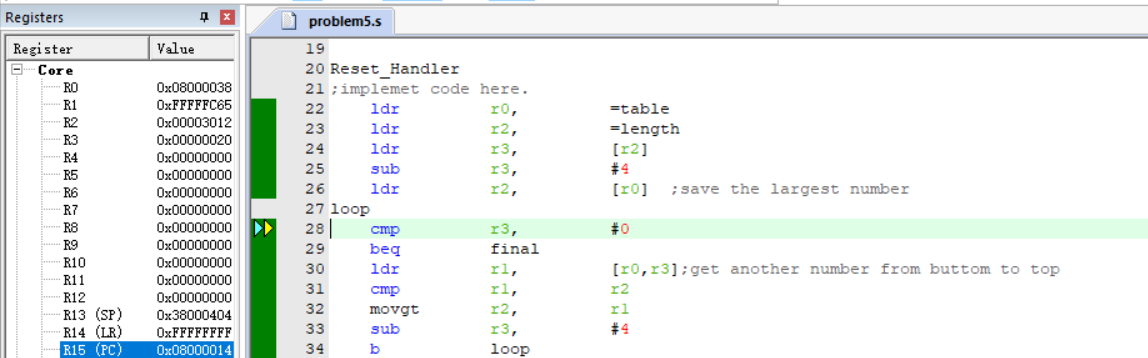
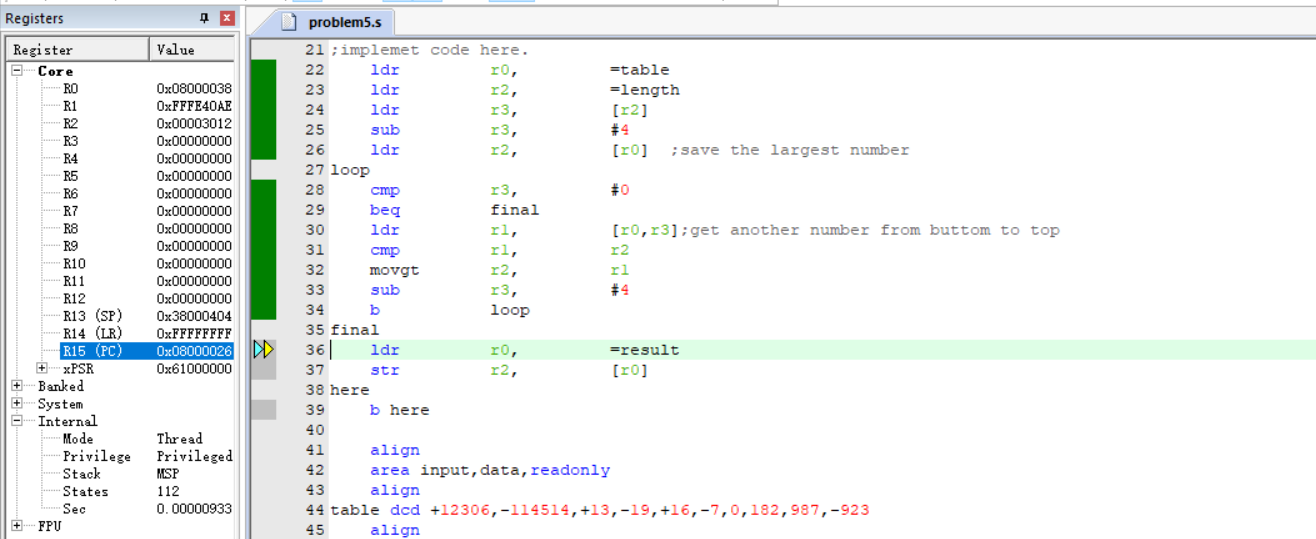


图 44：执行一次循环后

* 如图45所示，循环结束后，r2保存最大值0x00003012，最后一个参与比较的数r1（0xFFFE40AE）的确为第二个数，从Memory窗口（图46）也可看出所有数均已遍历。



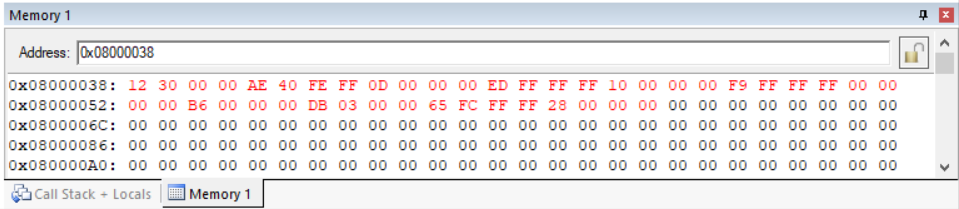


图 45：验证所有的数均已遍历

* 执行第36-37行指令，将r2存入RAM中，如图46、47所示，最大值0x00003012（+12306）被成功放在了地址0x38000000处。

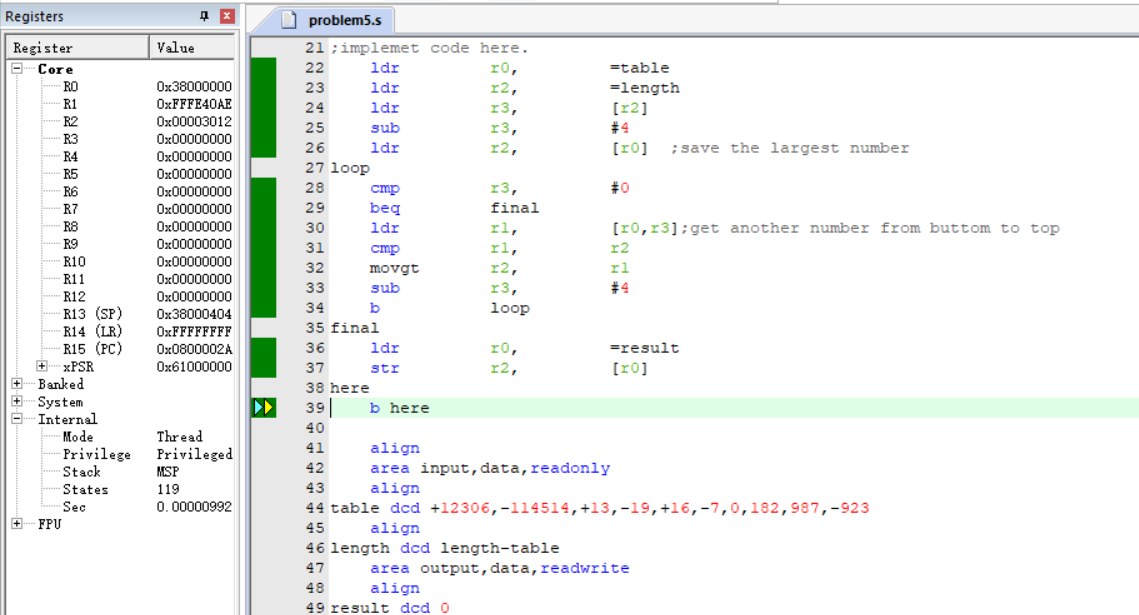


图 46：执行完第36-37行指令

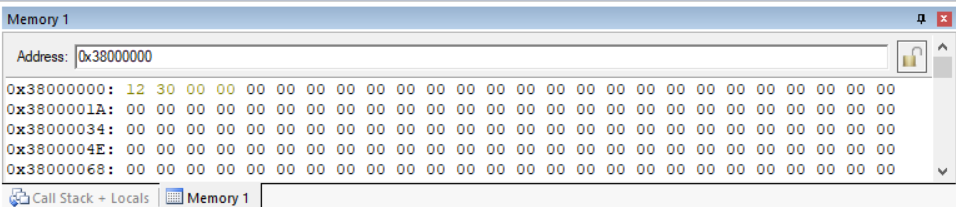


图 47：通过Memory窗口查看地址