北京航空航天大學
BEIHANG UNIVERSITY

《微机原理》实验报告

姓名：曹建钦

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：

```
Reset_Handler
;implement code here.
    mov     r0,     #0x5c
    mov     r1,r0,lsr#4
    and     r2,R0,#0x0F
    cmp     r1,#9
    addgt   r1,r1,#7
    add     r1,r1,#0x30
    lsl     r1,#8
    cmp     r2,#9
    addgt   r2,r2,#7
    add     r2,r2,#0x30
    orr     r1,r2
    ldr     r3,=result
    str     r1,[r3]
here
    b here
```

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：

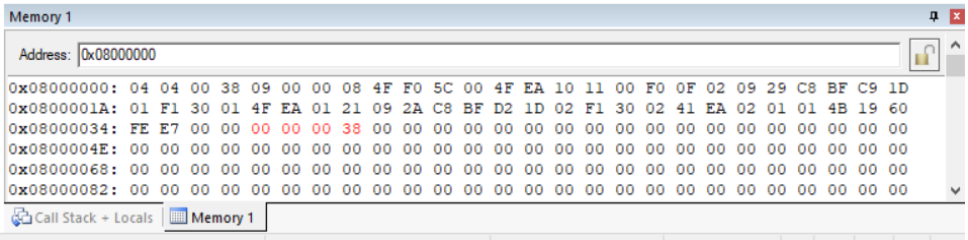


图 2：复位过程 Memory 窗口查看 address

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。

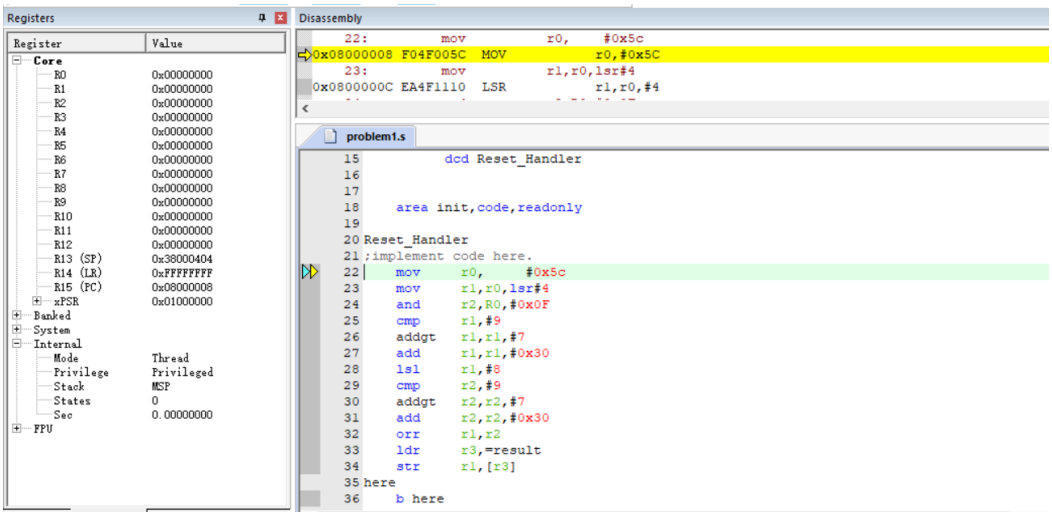


图 3：复位过程 Register 窗口

- 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：

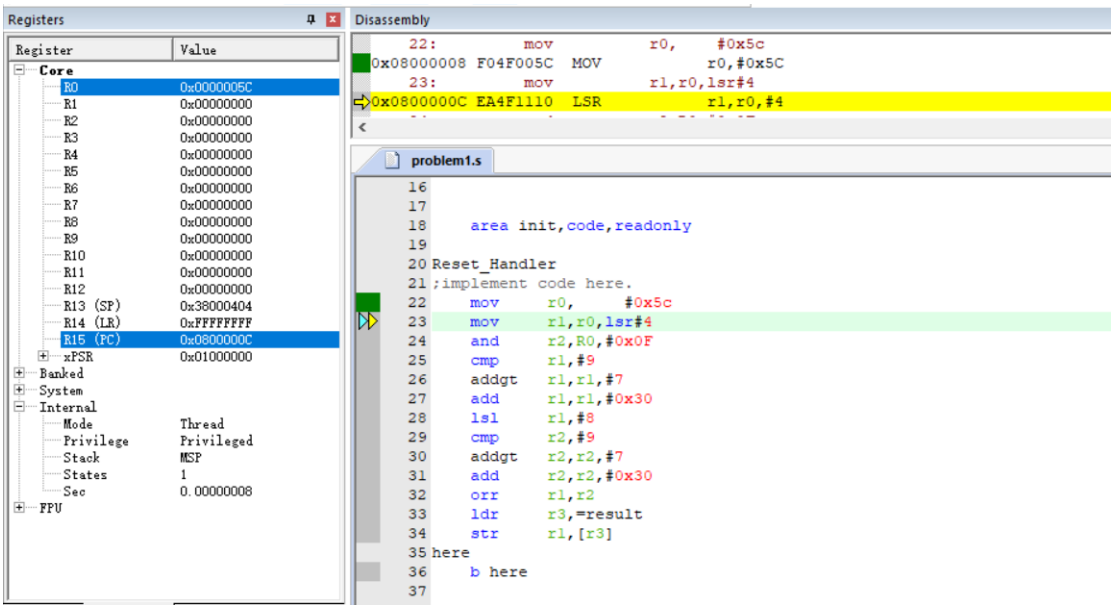


图 4：单步执行第 22 行指令

- 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：

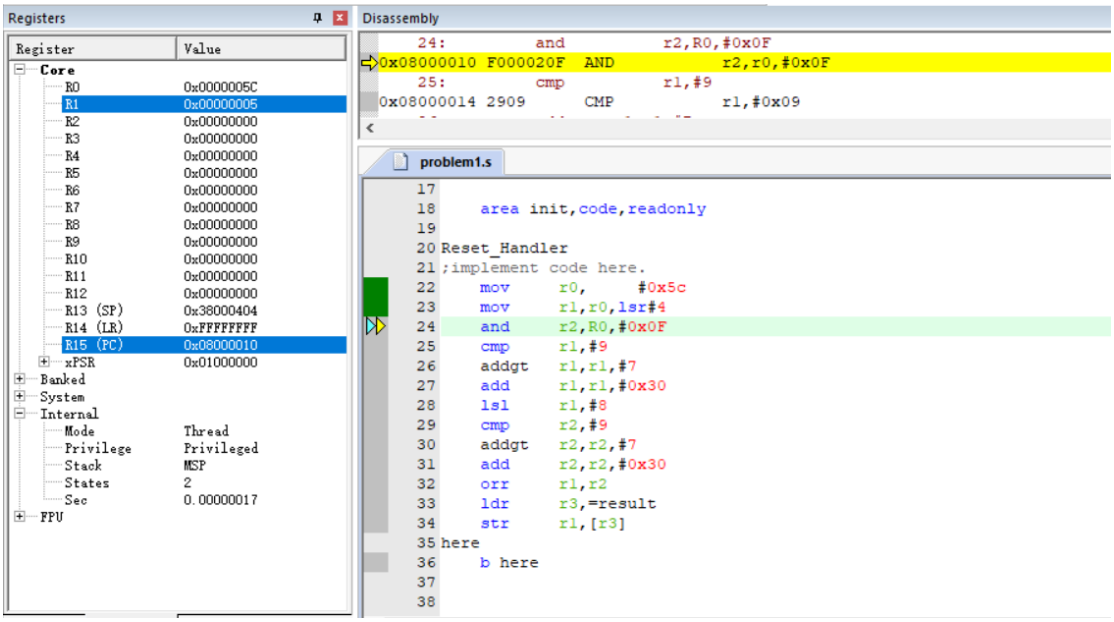


图 5：单步执行第 23 行指令

- 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：

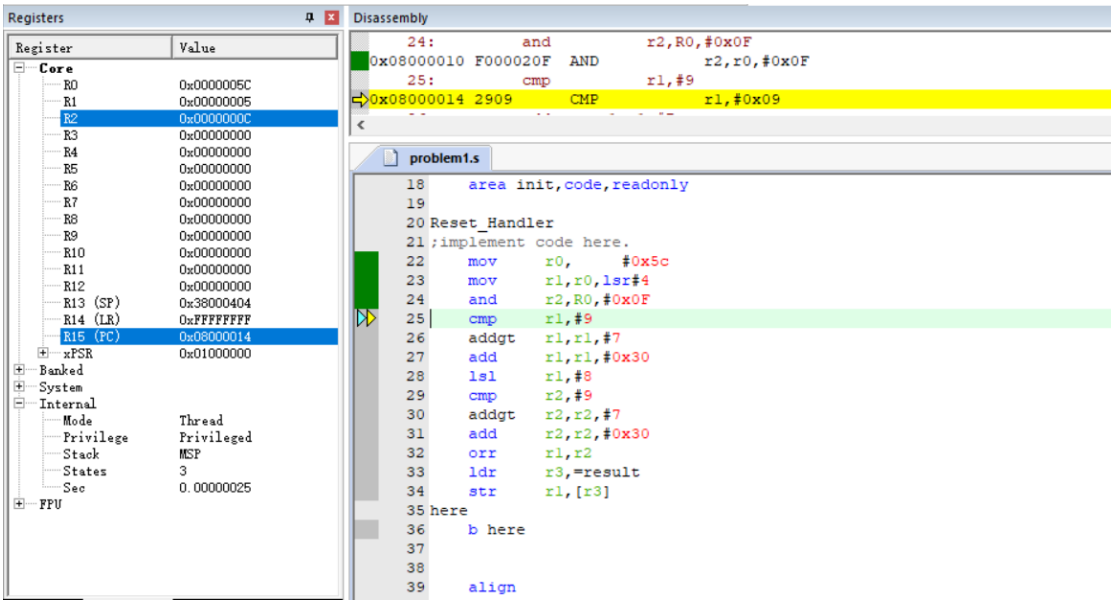


图 6：单步执行第 24 行指令

- 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：

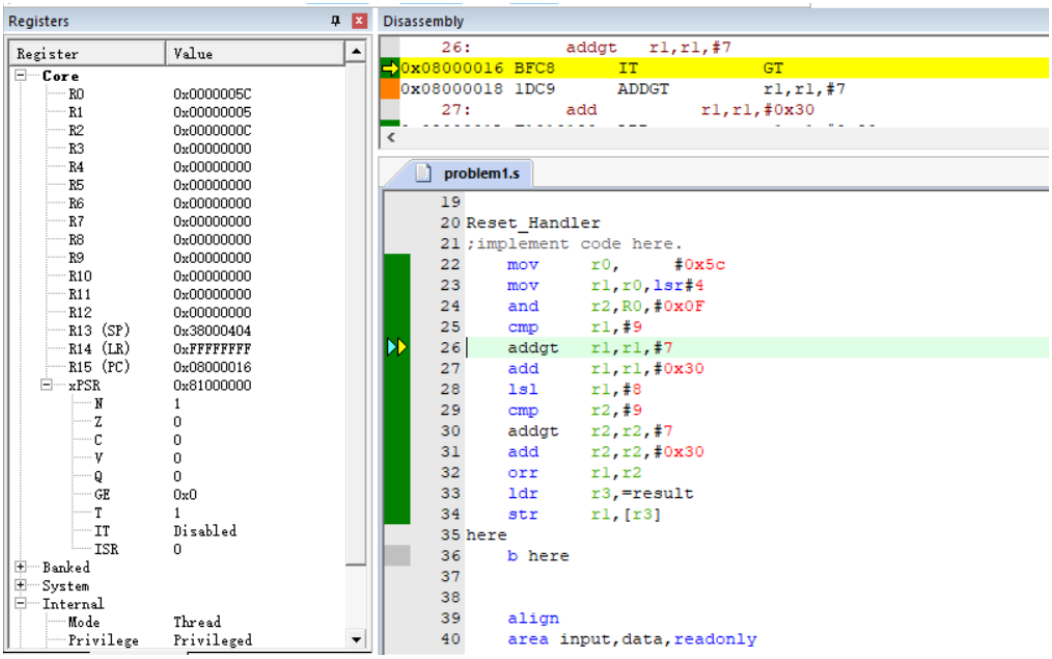


图 7：单步执行第 25 行指令

- 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：

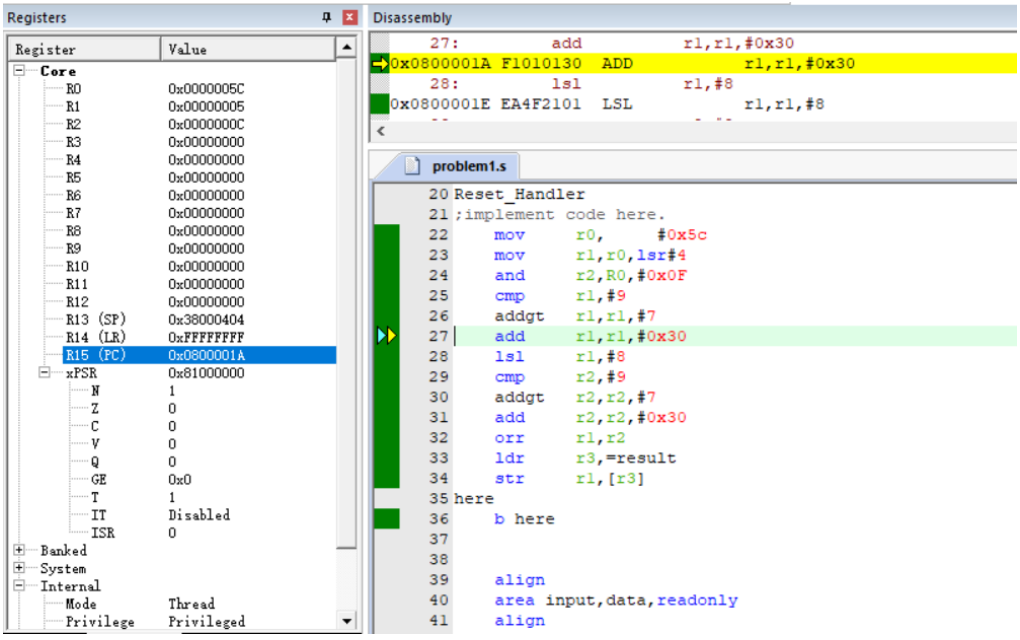


图 8：单步执行第 26 行指令

- 单步执行第 27 行指令，r1 增加 0x30 得到 0x35，如图 9 所示：

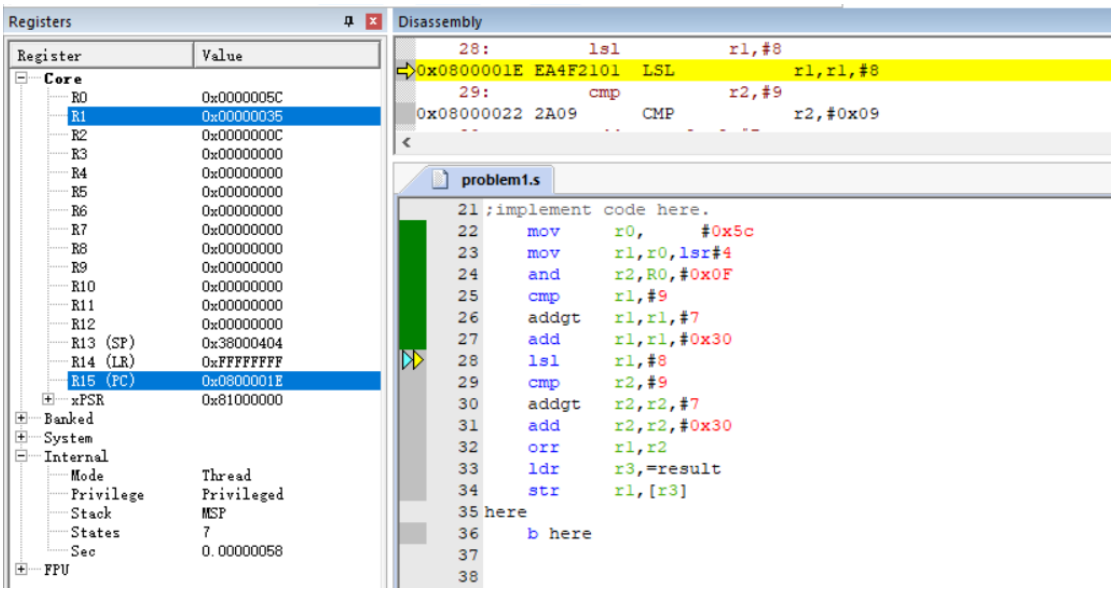


图 9：单步执行第 27 行指令

- 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：

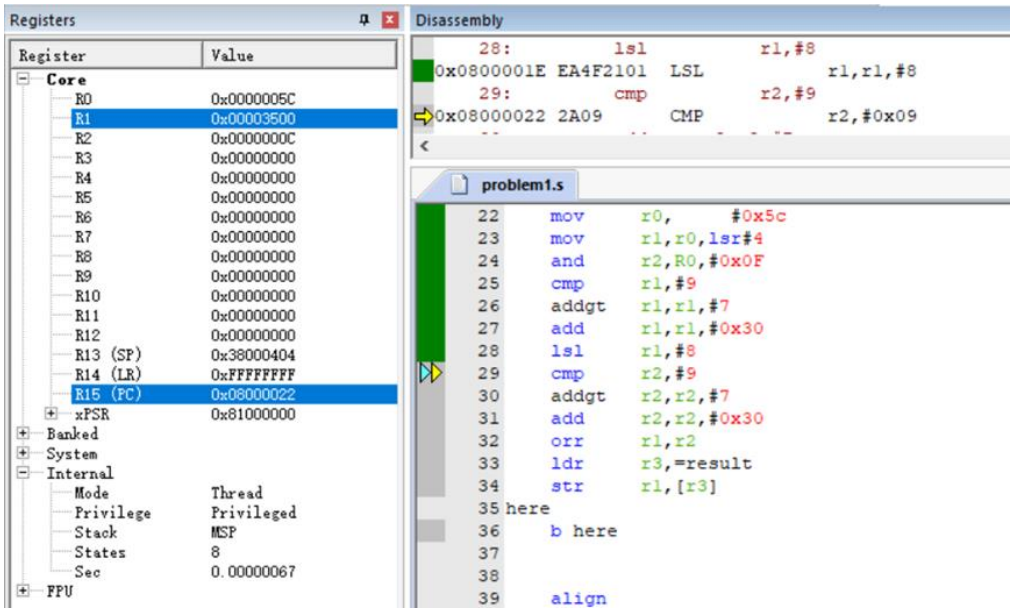


图 10：单步执行第 28 行指令

- 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：

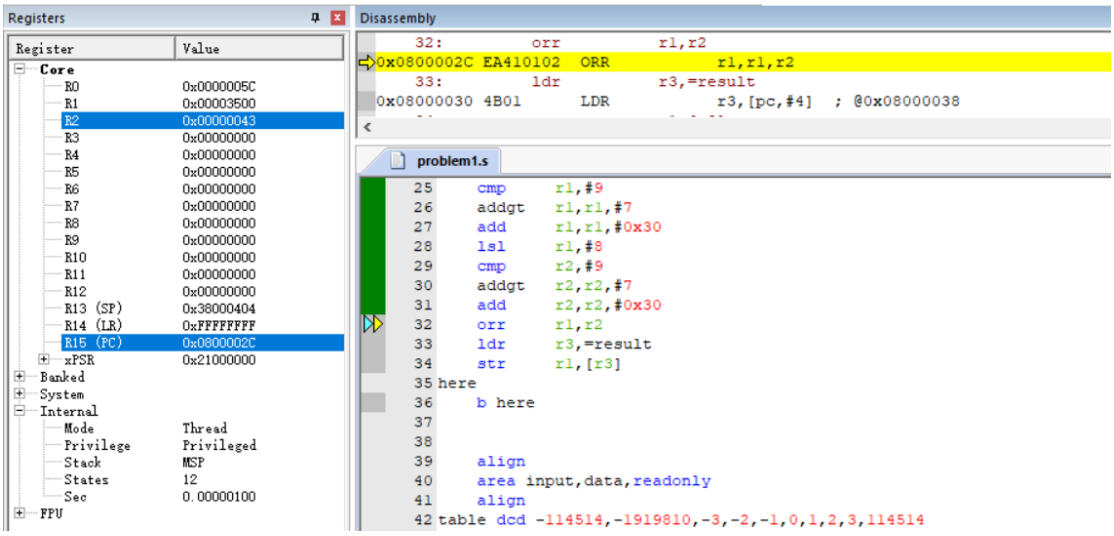


图 11：执行完第 31 行指令

- 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：

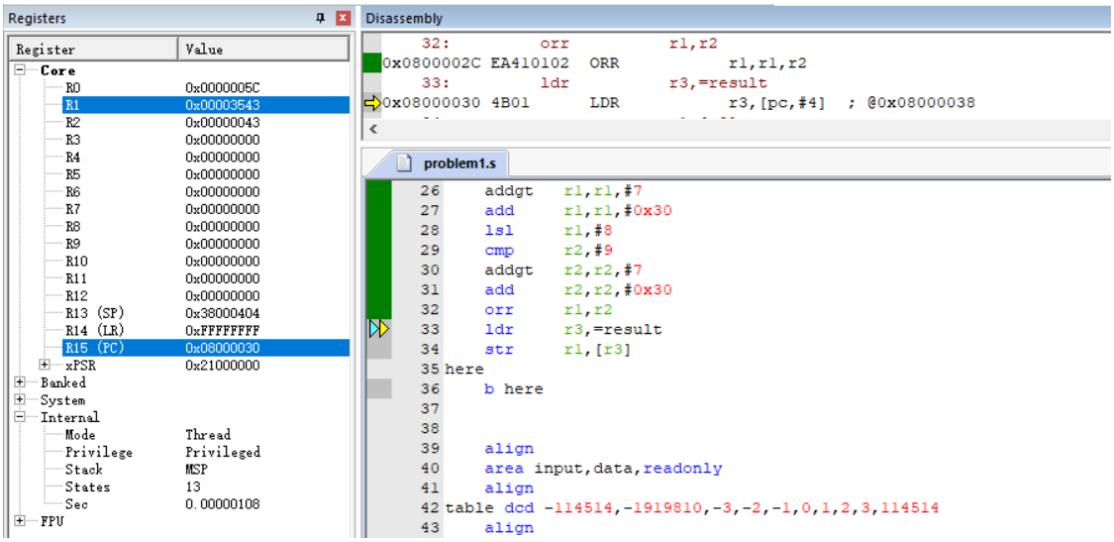


图 12：单步执行第 32 行指令

- 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：

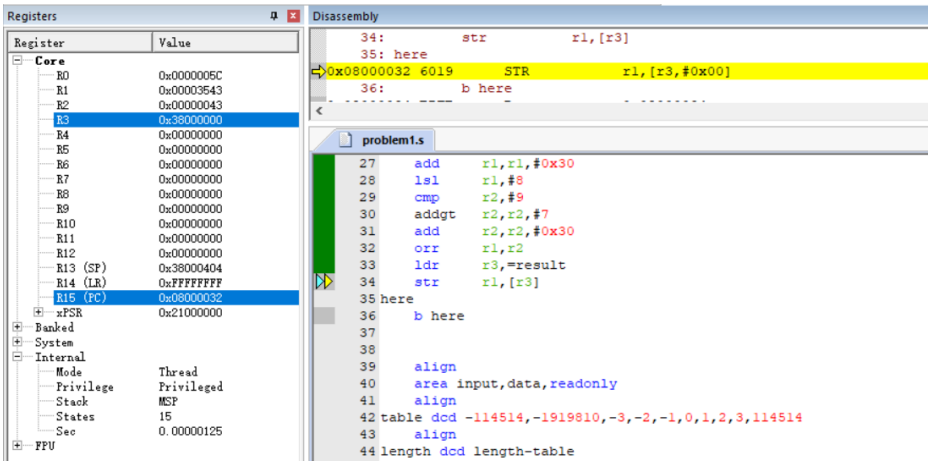


图 13：单步执行第 33 行指令

- 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。

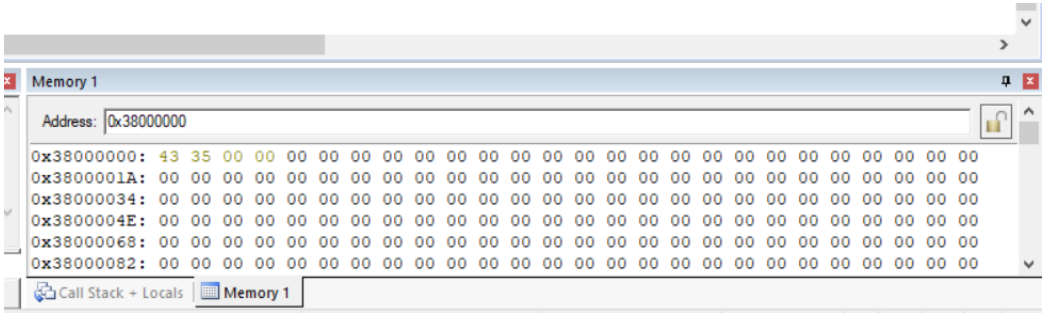


图 14：Memory 窗口查看 address

（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：

```
20  Reset_Handler
21  ;implemet code here.
22      ldr     r0,     =table
23      ldr     r1,     [r0]
24      ldr     r2,     [r0,#4]
25      cmp     r1,     r2
26      bgt     final
27      mov     r1,     r2
28  final
29      ldr     r0,     =result
30      str     r1,     [r0]
31  here
32      b here
33
34
35
36      align
37      area input,data,readonly
38      align
39  table dcd -114514,-1919810
40      align
41  length dcd length-table
42      area output,data,readwrite
43      align
44  result dcd 0
45      end
```

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

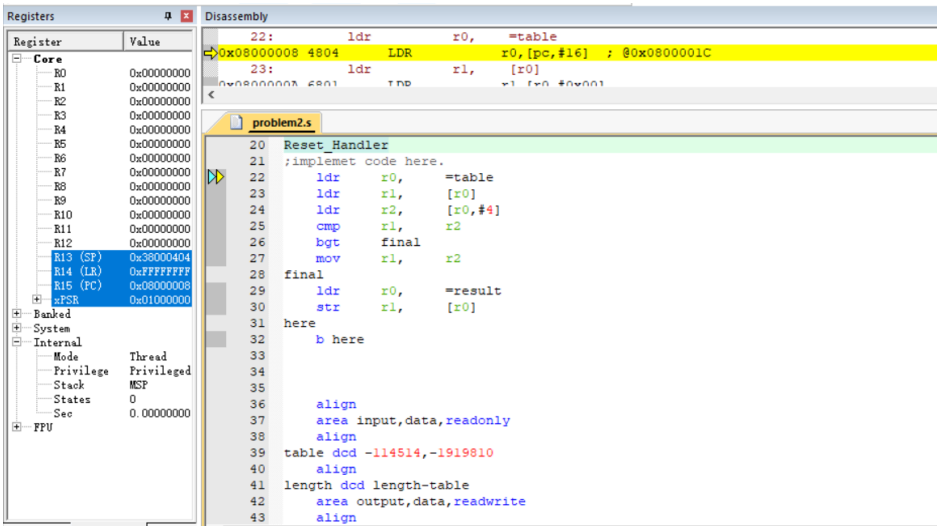


图 16：复位

- 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）

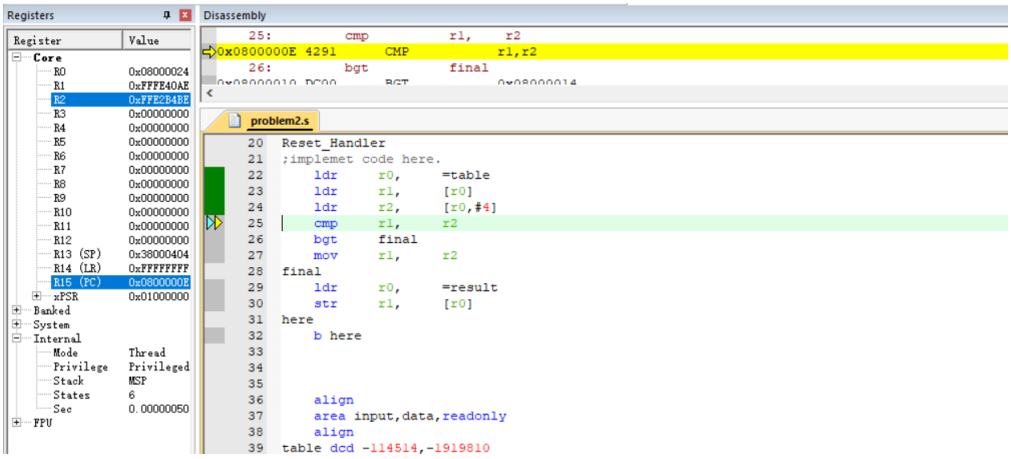


图 17：执行完第 24 行指令

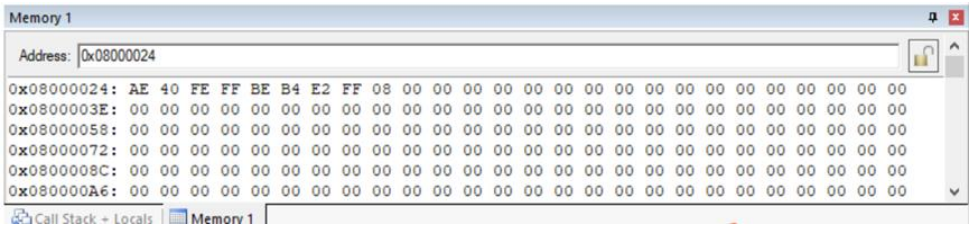


图 18：Memory 查看 table 地址

- 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：

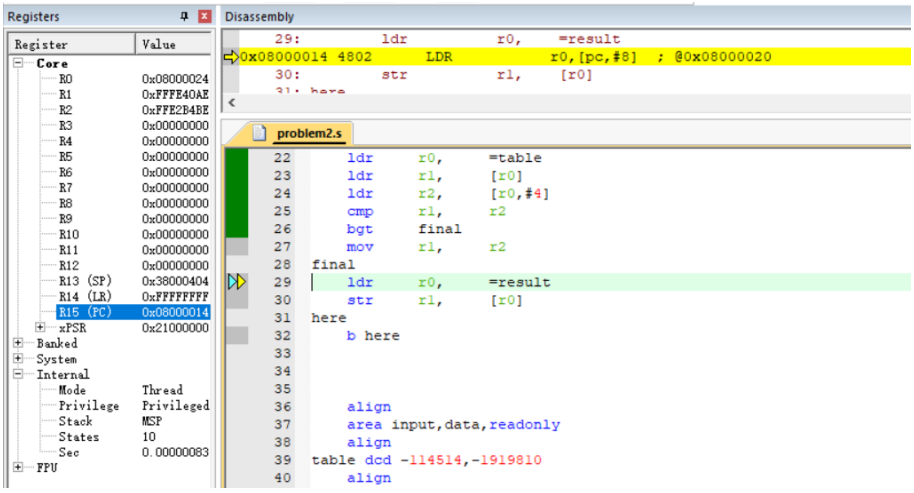


图 19：执行完 25 和 26 行指令

- 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：

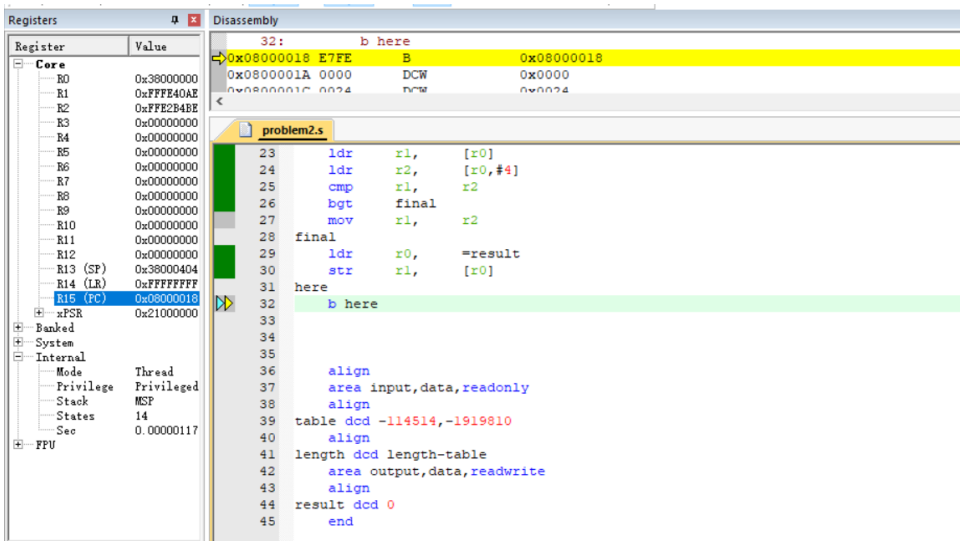


图 20：执行完第 30 行指令

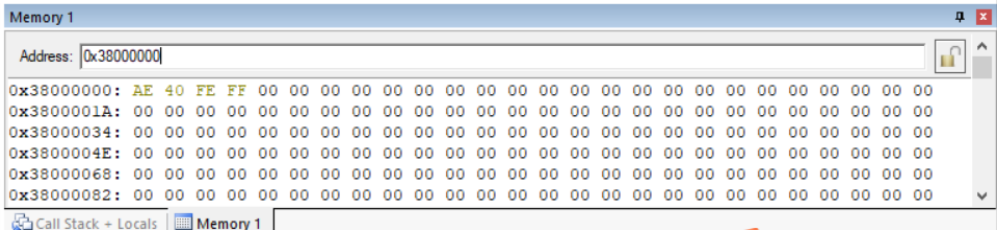


图 21：Memory 查看 result 首地址

（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：

```
Reset_Handler
;implemet code here.
    ldr     r2,     =x
    ldr     r0,     [r2]
    cmp     r0,     #10
    ble     next
    mov     r1,     #1
    b   final
next
    cmp     r0,     #-10
    movge   r1,     #0
    movlt   r1,     #-1
final
    ldr     r2,     =result
    str     r1,     [r2]
here
    b here



    align
    area input,data,readonly
    align
x dcd -11
    align
length dcd length-x
    area output,data,readwrite
    align
result dcd 0
    end
```

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：

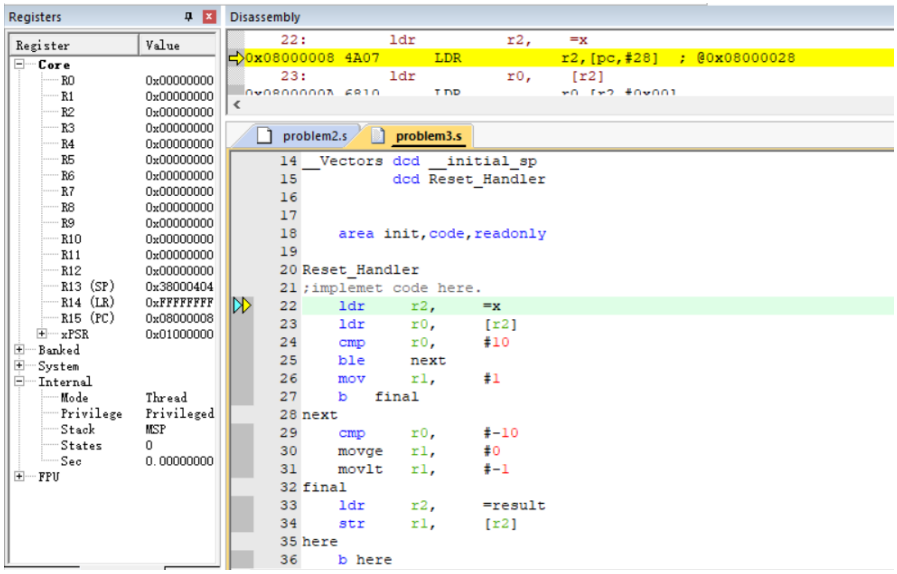


图 23：复位

- 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

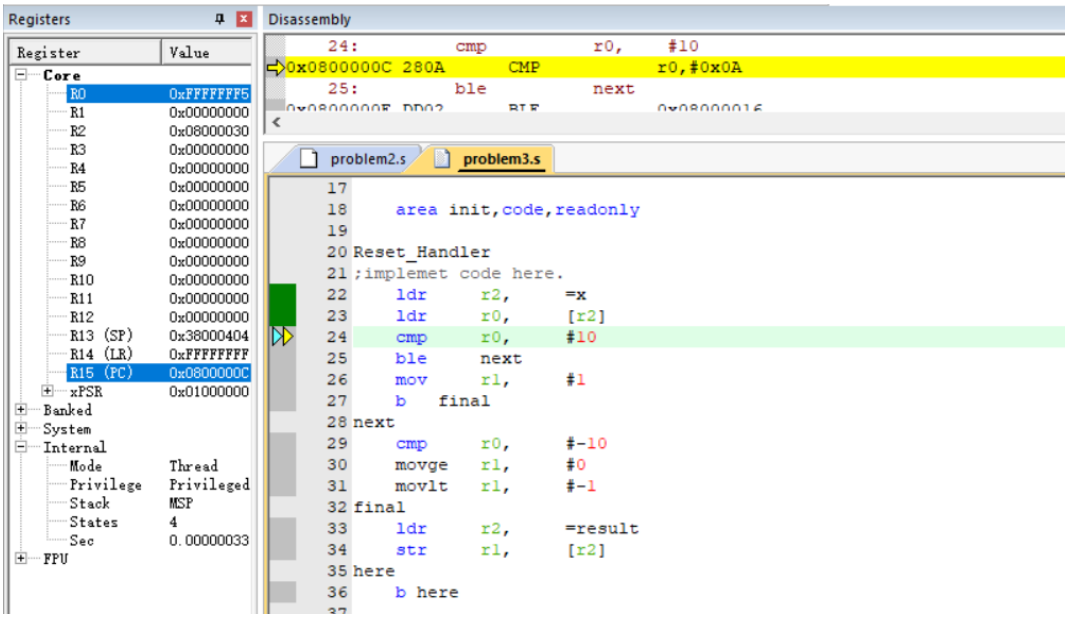


图 24：执行完第 23 行指令

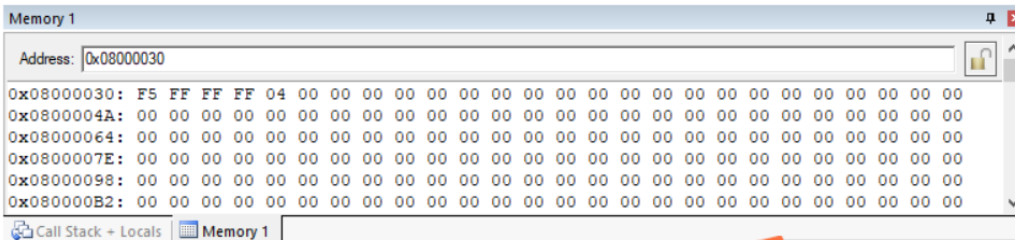


图 25：通过 Memory 查看地址

- 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：

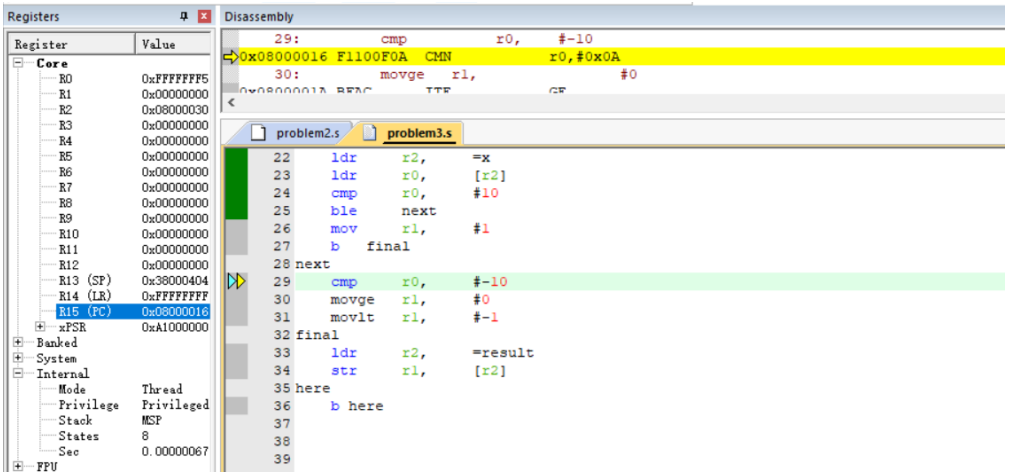


图 26：执行完第 24-25 行指令后跳转至 next

- 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：

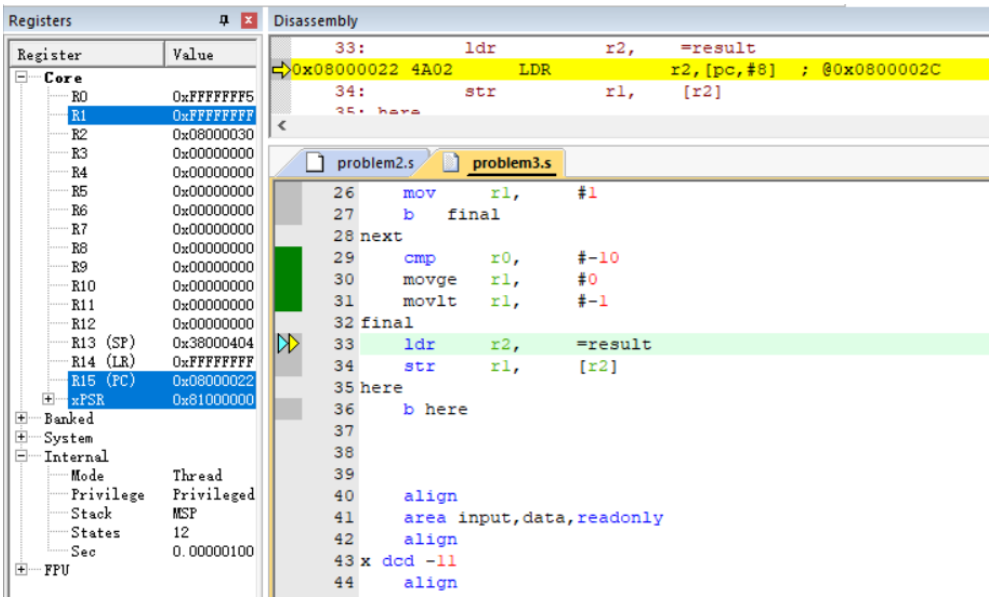


图 27：执行完第 29-31 行指令后

- 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。

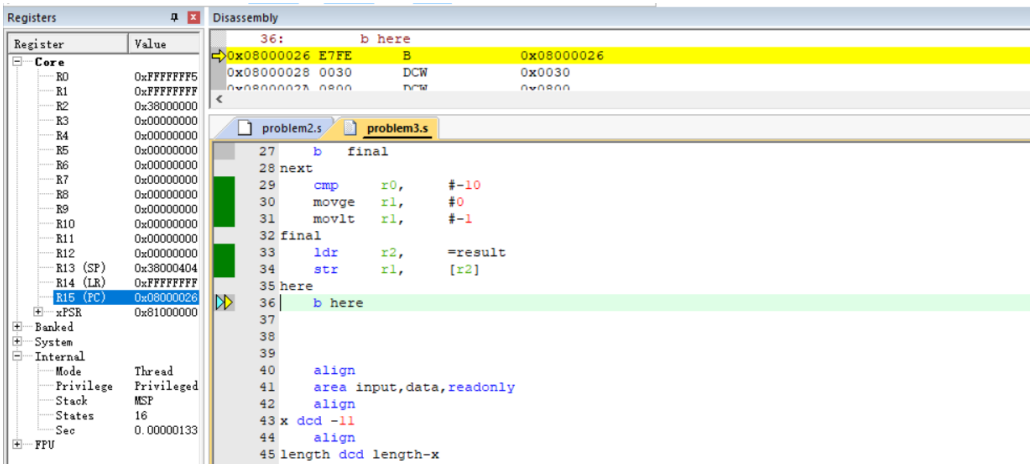


图 28：执行完第 34 行指令

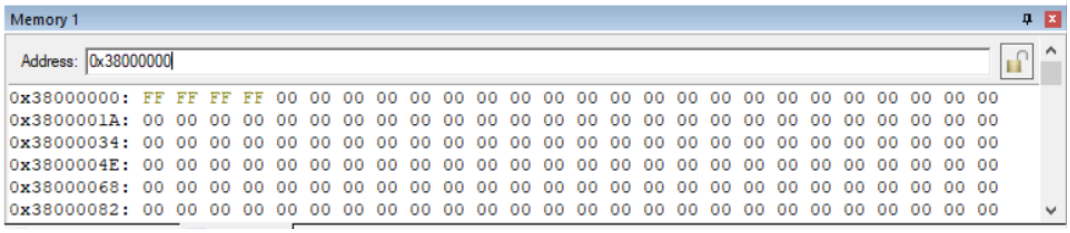


图 29：Memory 窗口搜索地址

（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：

```
;implemet code here.
    ldr     r0,     =table
    mov     r1,     #0              ;for count
loop
    ldrb    r2,     [r0,r1]
    cmp     r2,     #0x0D
    beq     final
    add     r1,     #1
    b       loop

final
    ldr     r0,     =result
    str     r1,     [r0]

here
    b here

    align
    area input,data,readonly
    align
table dcb "Hello World!\r"
    align
length dcd length-table
    area output,data,readwrite
    align
result dcd 0
    end
```

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：

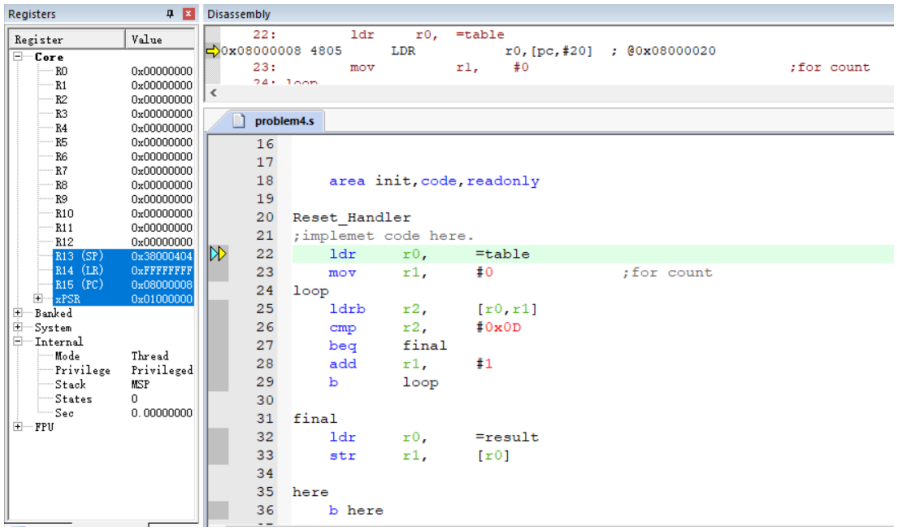


图 31：复位

- 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。

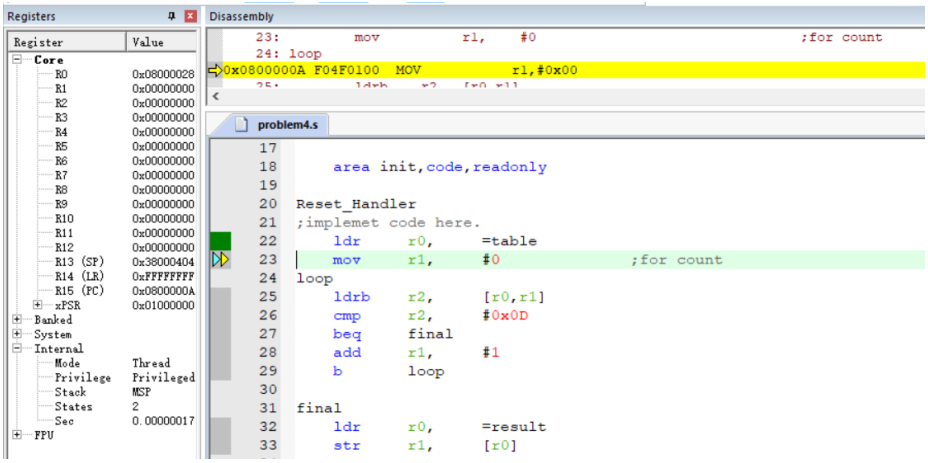


图 32：单步执行第 22 行指令

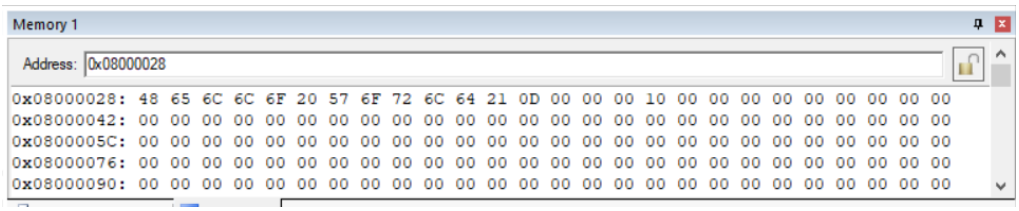


图 33：通过 Memory 窗口搜索地址

- 单步执行第 23 行指令，将 r1 赋初值 0，如图 34 所示。

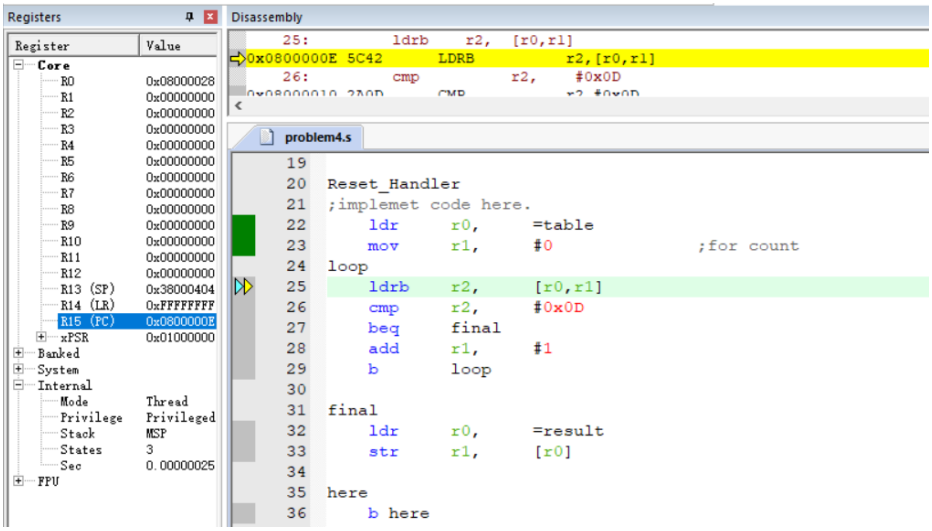


图 34：单步执行第 23 行指令

- 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。

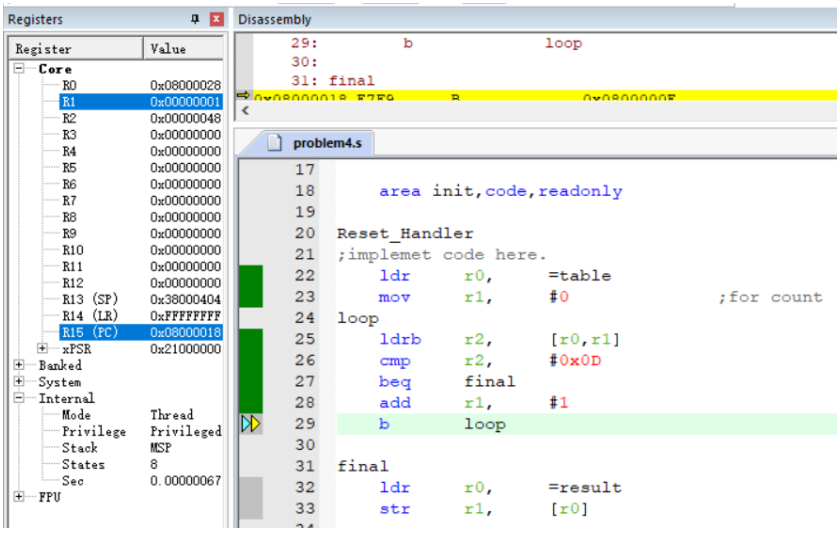


图 35：第一次循环结束

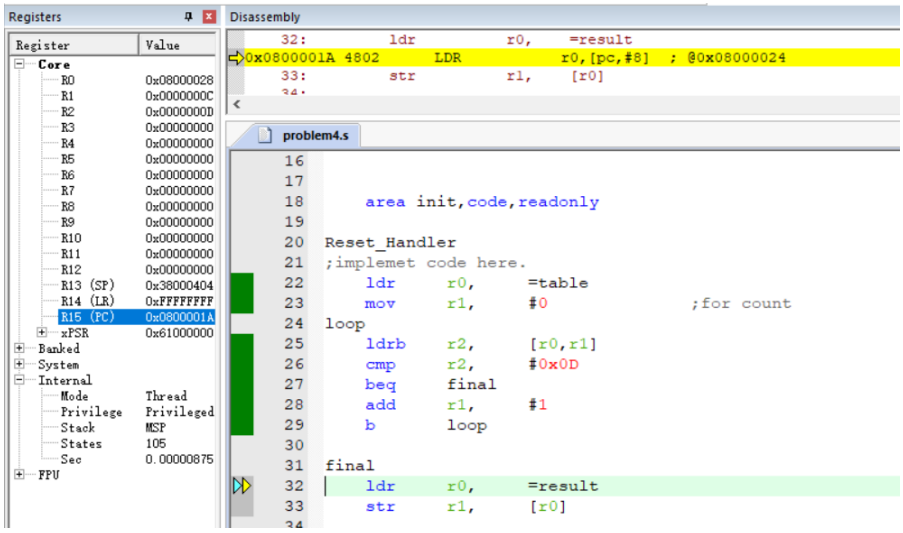


图 36：最后一次循环结束

- 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。

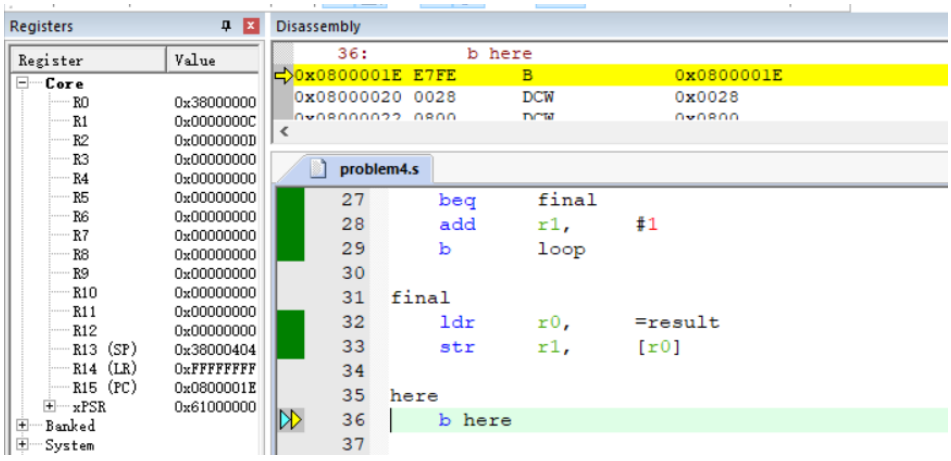


图 37：执行完 32、33 行指令

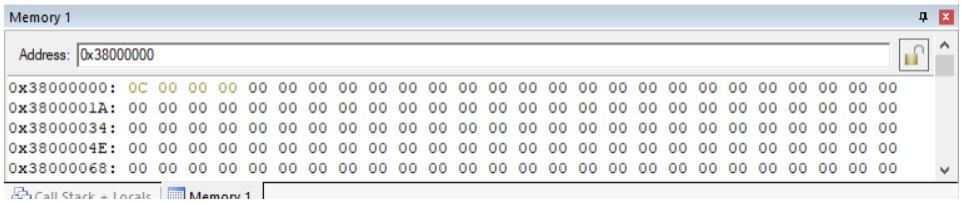


图 38：通过 Memory 窗口查看地址

（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：

```
;implemet code here.
    ldr         r0,         =table
    ldr         r2,         =length
    ldr         r3,         [r2]
    sub         r3,         #4
    ldr         r2,         [r0]  ;save the largest number
loop
    cmp         r3,         #0
    beq         final
    ldr         r1,         [r0,r3];get another number from buttom to top
    cmp         r1,         r2
    movgt       r2,         r1
    sub         r3,         #4
    b           loop
final
    ldr         r0,         =result
    str         r2,         [r0]
here
    b here

    align
    area input,data,readonly
    align
table dcd +12306,-114514,+13,-19,+16,-7,0,182,987,-923
    align
length dcd length-table
    area output,data,readwrite
    align
result dcd 0
    end
```

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：

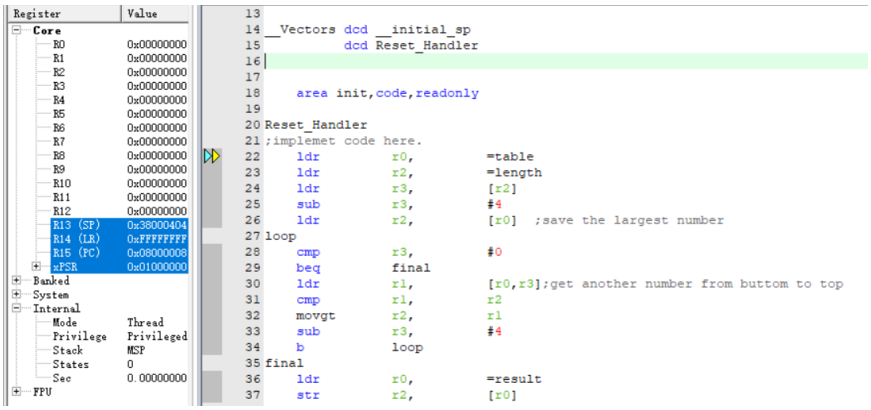


图 40：复位

- 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。

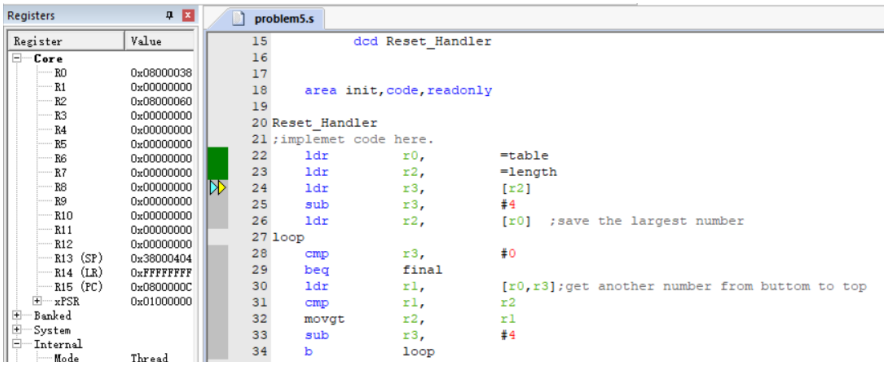


图 41：执行第 22-23 行指令

- 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。

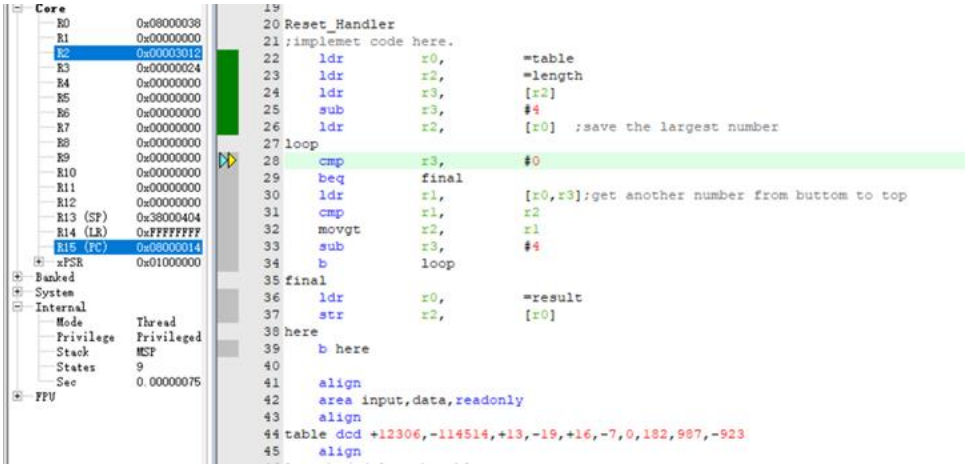


图 42：执行至第 28 行指令

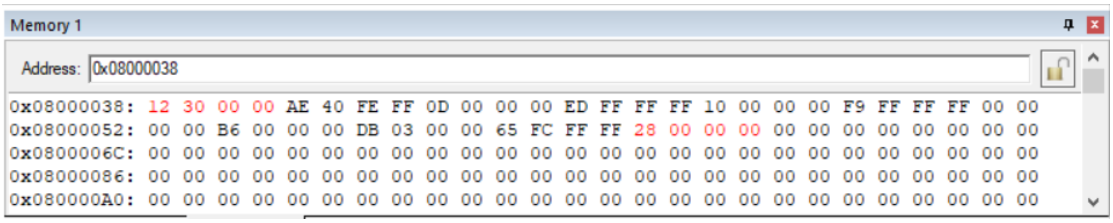


图 43：通过 Memory 窗口查看地址

- 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。

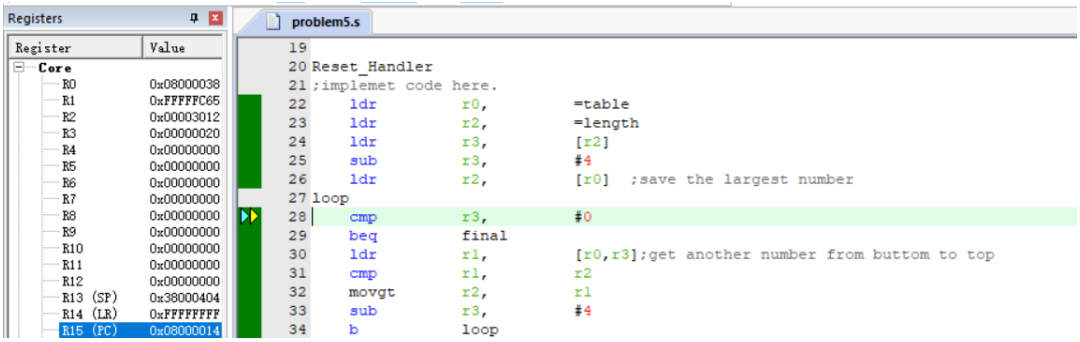
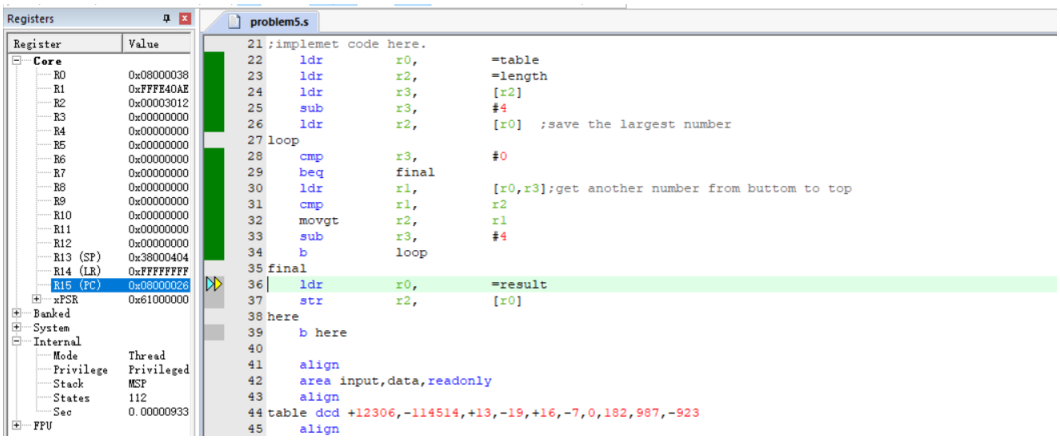


图 44：执行一次循环后

- 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。



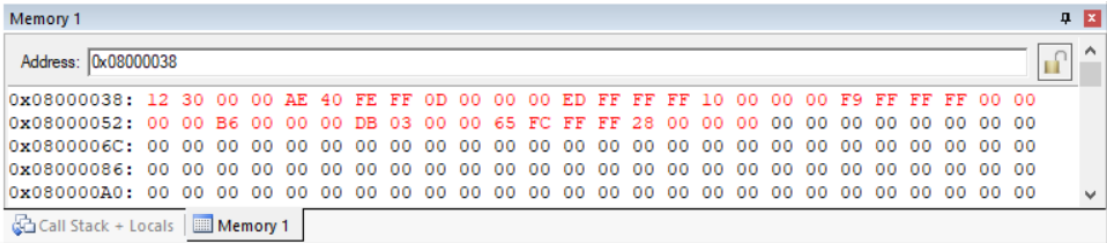


图 45：验证所有的数均已遍历

- 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。

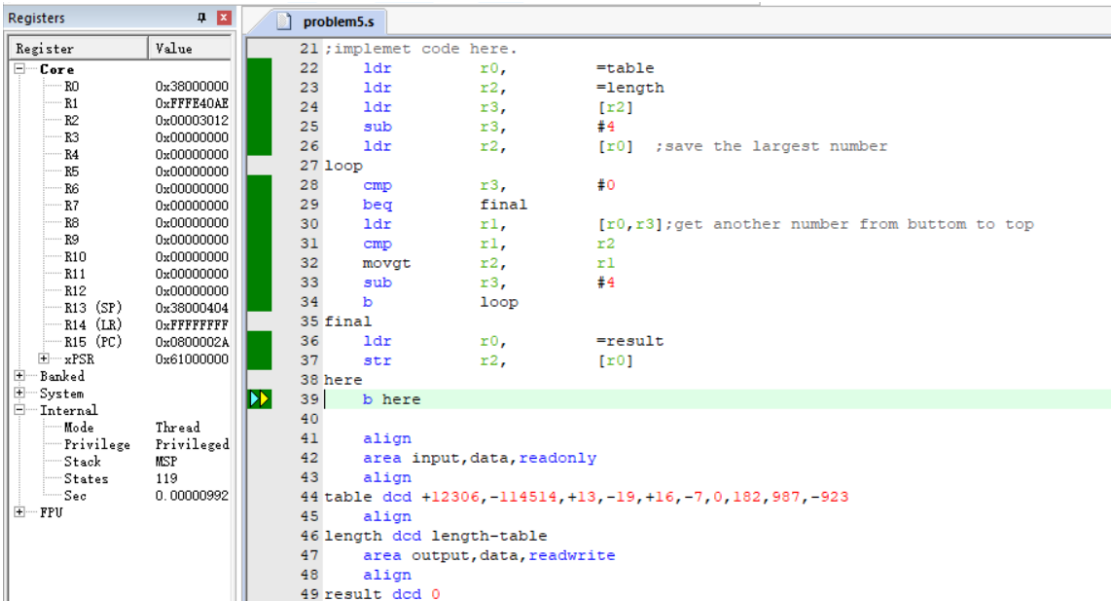


图 46：执行完第 36-37 行指令

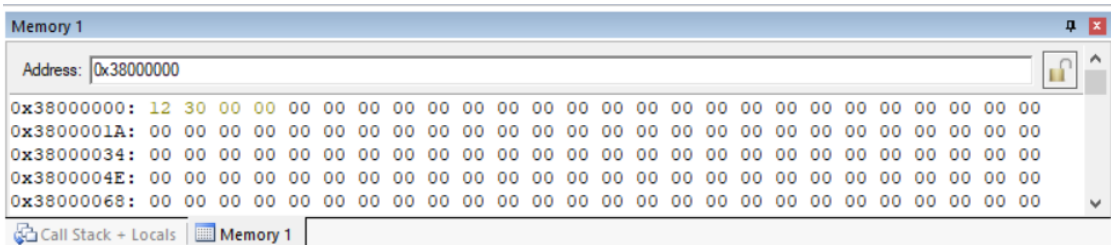


图 47：通过 Memory 窗口查看地址