# 嵌入式系统实验报告



|  |  |
| --- | --- |
| 实验名称： | Cortex-M 应用编程—C 与汇编 |
| 姓 名： | 王小龙 |
| 学 号： | 2020211502 |
| 学 院(系)： | 计算机学院 |
| 专 业： | 网络工程 |
| 指导教师： | 戴志涛、刘健培 |

2022年 12 月 1 日

# 实验目的

本实验主要是从 C 语言的角度分析 ARMv7 汇编的对应关系。

# 实验环境

* ST-Link 仿真器
* KeiluVision5 MDK集成开发软件
* Window10 (64bit)

# 实验要求

1.按照实验过程的说明阅读相关材料，观察、分析、实验 C 与 ARM 汇

编的关系。

2.参考步骤 2.3，将\_strcpy 函数翻译为 ARM 汇编代码。

# 实验原理

1.C 语言的各种结构分为数据流和控制流 2 个部分，数据流部分涉及数据的存储、寻址、加载、运算。控制流部分涉及处理数据的指令间的控制关系：跳转、条件分支、循环、函数调用与返回。

2.当把 C 语言编译为 ARM 机器指令后，所有的数据对象处于内存中（有些小

的立即数是存储在内存的指令里）。常量一般存储在 RO 段中，而全局变量则存

储在 RW 段中。

变量名字被翻译为数据的地址，变量的值是内存中存储的值。

局部变量的地址通过到堆栈指针的偏移量进行计算（相对寻址），全局变量

的地址在链接后已经设置为固定地址，通过在指令中将该地址加载到寄存器中进

行访问（寄存器间接寻址）。

变量的类型语义由 C 编译器生成相应的机器指令进行维护（例如，访问 char、

short、int 类型变量生成的 ld/st 指令是不一样的）。

C 的定义、赋值、运算符、表达式等语句一般翻译为 1 到多条机器指令。

如果是手写汇编，则要开发者自己管理数据的存储位置、访问方式，并根据

类型选择合适的运算指令。

除此之外，还需要决定如何分配寄存器。因为 ARM 处理器是"Load/Store”

体系结构的例子，意味着所有的计算（加法、与或非、移位等）都是使用寄存器

而不是内存中的数据来执行。换言之，在操作之前数据必须移入寄存器。数据传

输指令包括将常量装入寄存器、装载和存储内存数据、栈操作。开发者需要确定

数据所在的位置，使用合适的寻址方式访问数据，并决定使用哪些寄存器完成运

算。

3.C 定义语句——数据的定义

C 中定义了变量后，C 编译器会处理好将数据放在合适的位置，汇编则需要

开发者手动管理。

如果是 0-255 的常量，可以直接编码在指令里（立即数）。

如果超出指令立即数的表示范围，对最长 32bit 的常数，可以使用 ARM 汇

编的伪指令 LDR、STR 访问。汇编器会翻译该伪指令，在指令块的附近开辟内

存存储这些常数，然后生成指令，使用相对 PC（指令计数器）的方式对数据进

行寻址。

如果是数组、字符串、结构体之类的复杂多字节数据，可以使用 ARM 汇编

的伪指令 DCB、DCD、SPACE、FILL 等手动开辟一块内存，然后使用相对该内

存首地址的偏移量进行寻址访问。

如果是函数内的局部变量，可以相对 SP（堆栈指针）开辟一块内存，然后

使用相对 SP 的偏移量进行寻址访问。

4.C 赋值语句——数据的 load 与 store

C变量间的赋值是将数据从“右值”取出，存入“左值”，对应汇编的 LDR和 STR 两类指令，需根据赋值数据的大小选择合适的指令。如果是结构体赋值之类的复杂语句，需生成多条 LDR/STR 指令。

5.C 指针的寻址

C指针用于保存某个内存单元的地址。当通过指针访问该内存单元的值时（\*操作），即将数据从内存读入处理器或者将值从处理器写入内存，需要用到 LDR和 STR 指令，C 的指针\*操作对应 LDR 和 STR 指令的[]。

6.C 数组的寻址

C 数组与 C 指针在寻址上几乎没有区别，以数组首地址为起始地址，根据下

标和数据单元的大小，计算偏移量，然后使用 LDR/STR 访问。

7.C 结构体的寻址

访问结构体的域时，首先需要获得结构体的首地址，然后根据域在结构体内

的偏移量，进行寻址。

8.C 函数局部变量的访问

局部变量是存储在函数的堆栈中的，通过相对堆栈SP的偏移量访问。ARMThumb-2 提供了 POP 和 PUSH 指令用于将一到多个字在处理器和内存间进行传输。

9.C 运算符与表达式

C 中的运算符与表达式一般会翻译为多条汇编指令，常见的模式是：LDR-OP-STR，即先用 LDR 指令将数据传入处理器，然后对数据进行操作，最后将结果传回内存。

10.. 控制结构

一旦处理器运行，会不断地重复取指—执行周期。因此处理器的默认操作是

执行从连续内存单元读取的线性指令序列。为了完成有用的工作，程序必须偶尔

从线性指令序列中离开并将控制转移到（有条件地或者无条件地）内存的另一部

分。在 C 语言中，控制转移由 break、continue、switch 和 goto 语句，函数调用和返回以及隐含的循环结尾产生。当代码编译时，上述每一项都被翻译为分支指令。C 的跳转目的地址是由编译器计算的，而汇编中则需要通过 label 显示指定。

11.判断条件

判定通常从对表达式值真假的判断开始。该表达式通常称为条件表达式，可以是两个值的简单比较。混合条件表达式包含了两个及两个以上值的比较，通过使用布尔操作符将这些基本条件表达式进行合并，来构成混合条件表达式。在汇编语言中，每个判定都是一个两步骤的过程：一条指令用于设置标识位，随后紧跟的一条指令利用这些标识位来做出实际的分支判定。

混合条件表达式的每个构成都需要这个两部分的序列 —— 一条比较指令及紧跟其后的一条条件分支指令。大多数情况下，第一条指令通过计算两个值的差来进行比较。差的实际值并不重要一一重要的是其特征。例如，比较指令（CMP）计算两个值的差，丢弃差值，而将差值特征保存在标识位中。然后通过检查标识

位（使用条件分支指令）对两个值的相对大小做出判定，并相应地对执行流进行

修改。

12.循环语句

循环通常由 4 部分构成：初始化、循环结束（或继续）的测试、循环控制变

量的更新、循环体（需要重复的代码）。有时循环变量的更新也是循环体的一部分。

循环可以通过如下方式来区分：循环结束的检测是在循环体之前还是之后。for 循环和 while 循环会在每次迭代之前进行循环结束的检测，然后重复 0 次或者更多次；do-while 循环在每次迭代之后进行检测，至少重复 1 次。

while 循环和 do-while 循环中循环底部到顶部具有隐含的分支，在翻译为汇

编代码时，该分支指令会变为显式分支。

计数循环（比如，for 的迭代次数循环）非常常见，因此 Thumb-2 指令集提

供了两条指令（CBZ 和 CBNZ）来将这些循环当中的比较和分支判定合并到一

条指令当中。

13.函数的实现

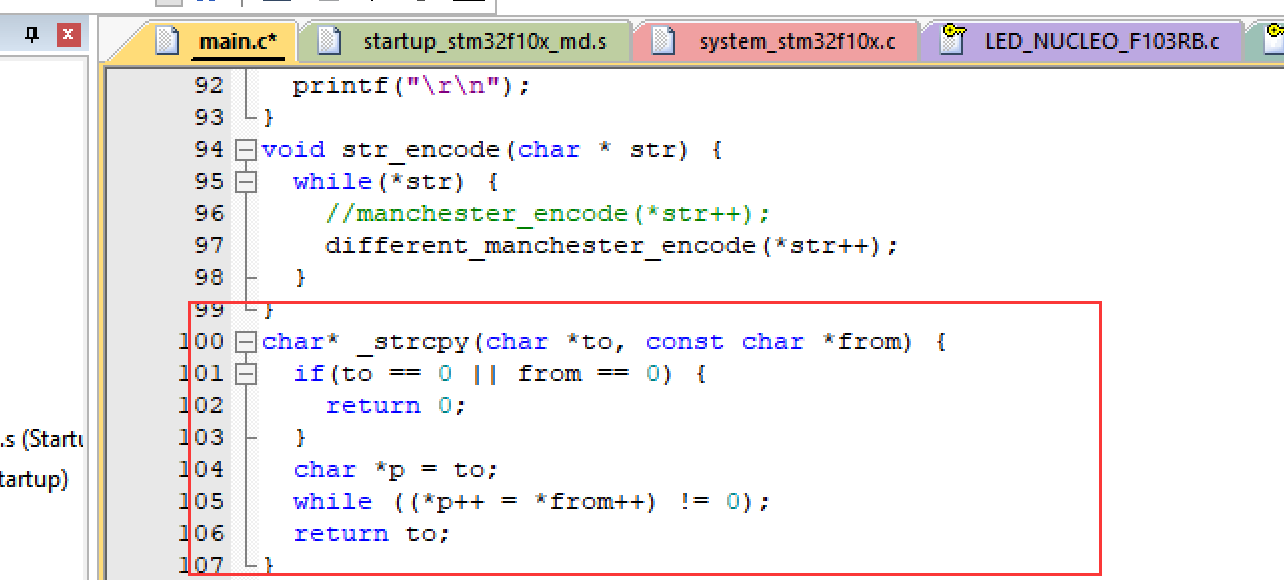
有两条指令最常用于从 C 函数中调用和返回，如下表所示。分支链接（BL）指令将返回地址（BL 指令之后紧跟的那条指令的地址）保存在 LR 寄存器中，而后跳转到函数的人口地址。接下来 BX 指令使用 LR 从被调用函数中返回，具

体方式是将 LR 的内容拷贝到 PC 程序计数器。

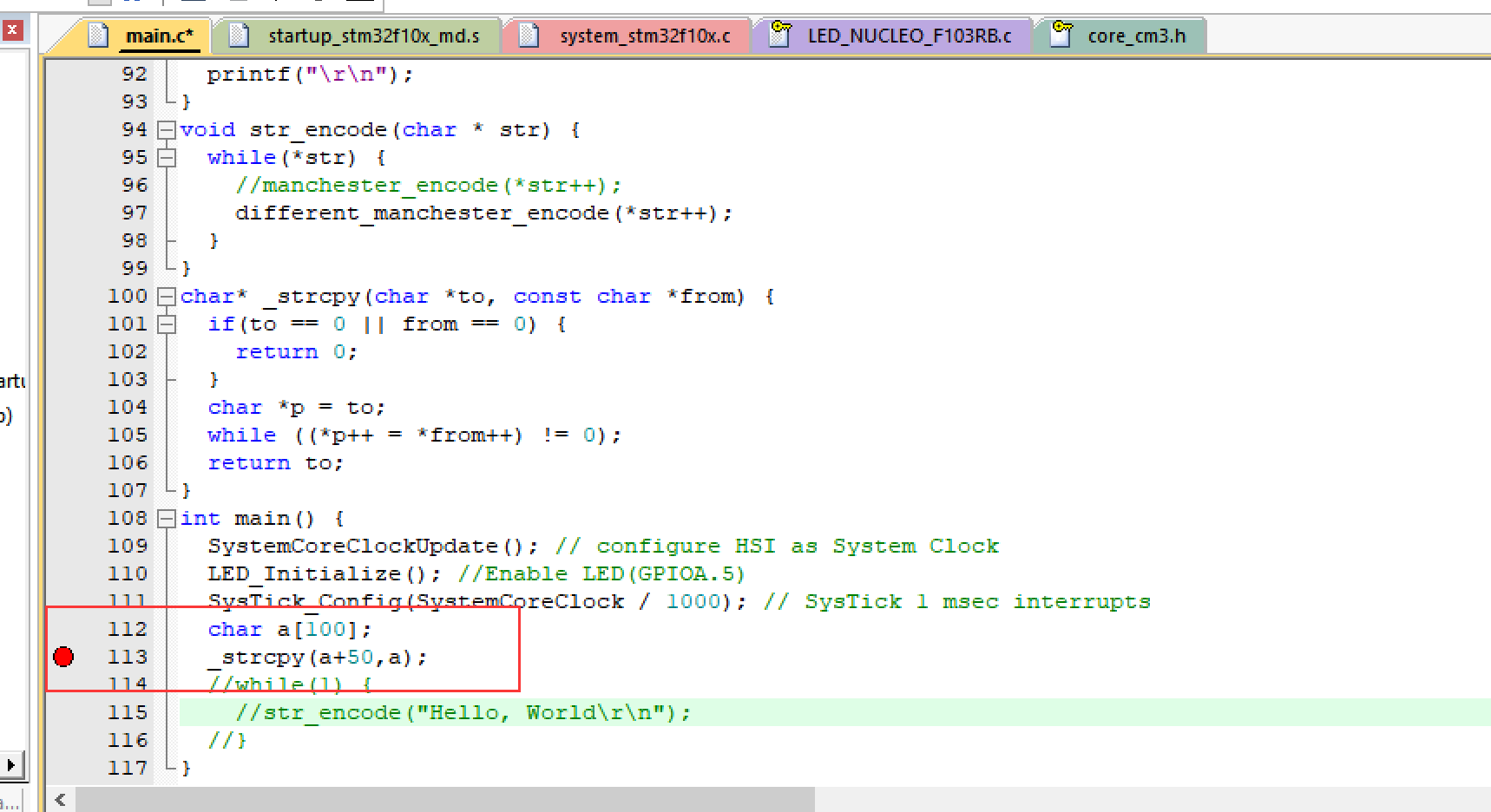
因为 LR 寄存器只能保存单个返回地址，因此调用其他函数的函数（非叶子函数）必须在函数入口点将 LR 保存起来，并在返回之前恢复原有内容。下图中 的代码通过使用PUSH和POP指令来达到这一目标。与使用POP指令来恢复 LR，并紧随其后使用 BX LR 来将 LR 拷贝到 PC 相比，更简单、常用的优化方法是使用单条 POP 指令将 LR 拷贝到 PC（POP {PC}）。

# 实验步骤

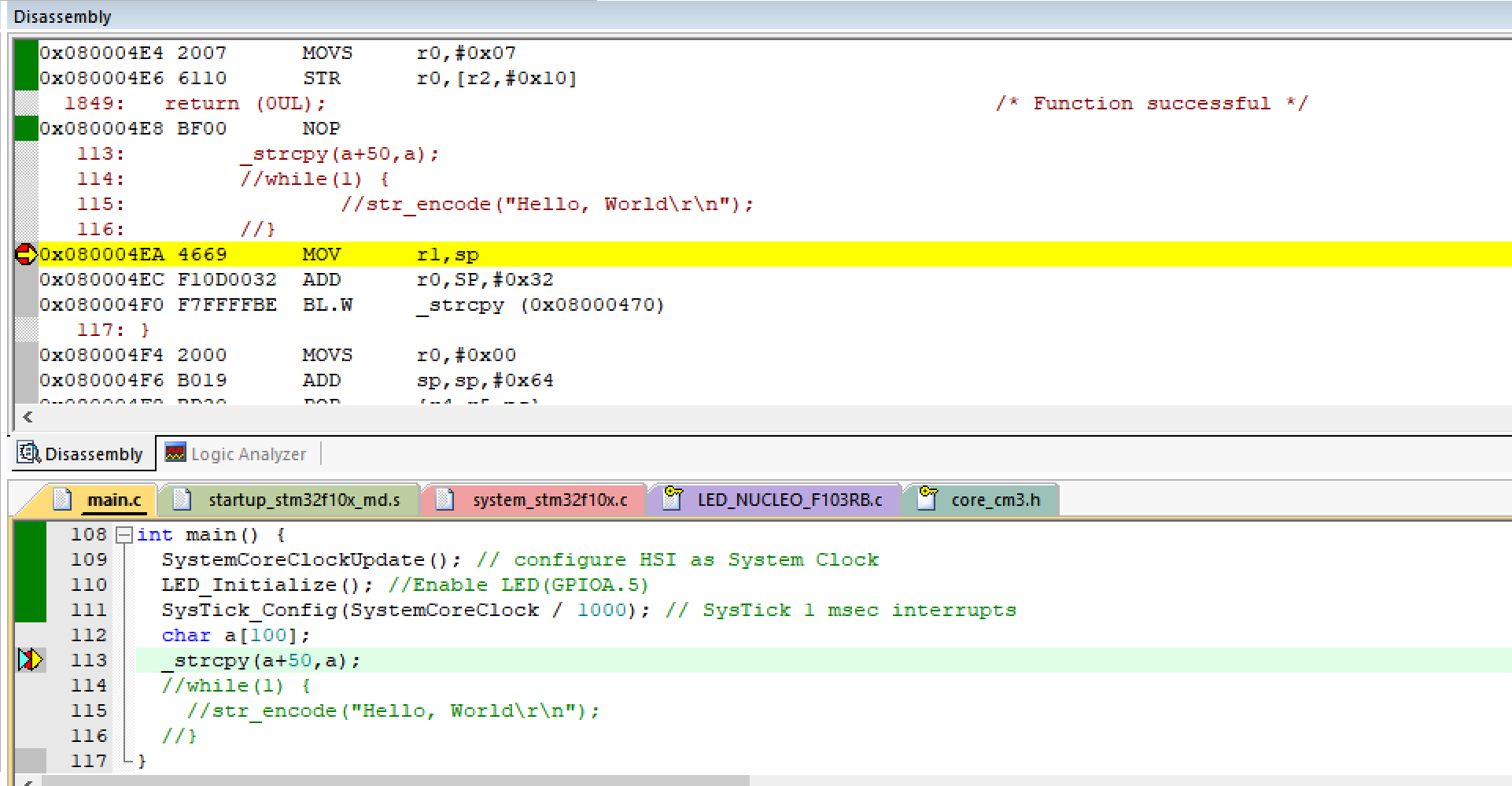
1.在lab3的代码上加上\_strcpy函数，如下图：



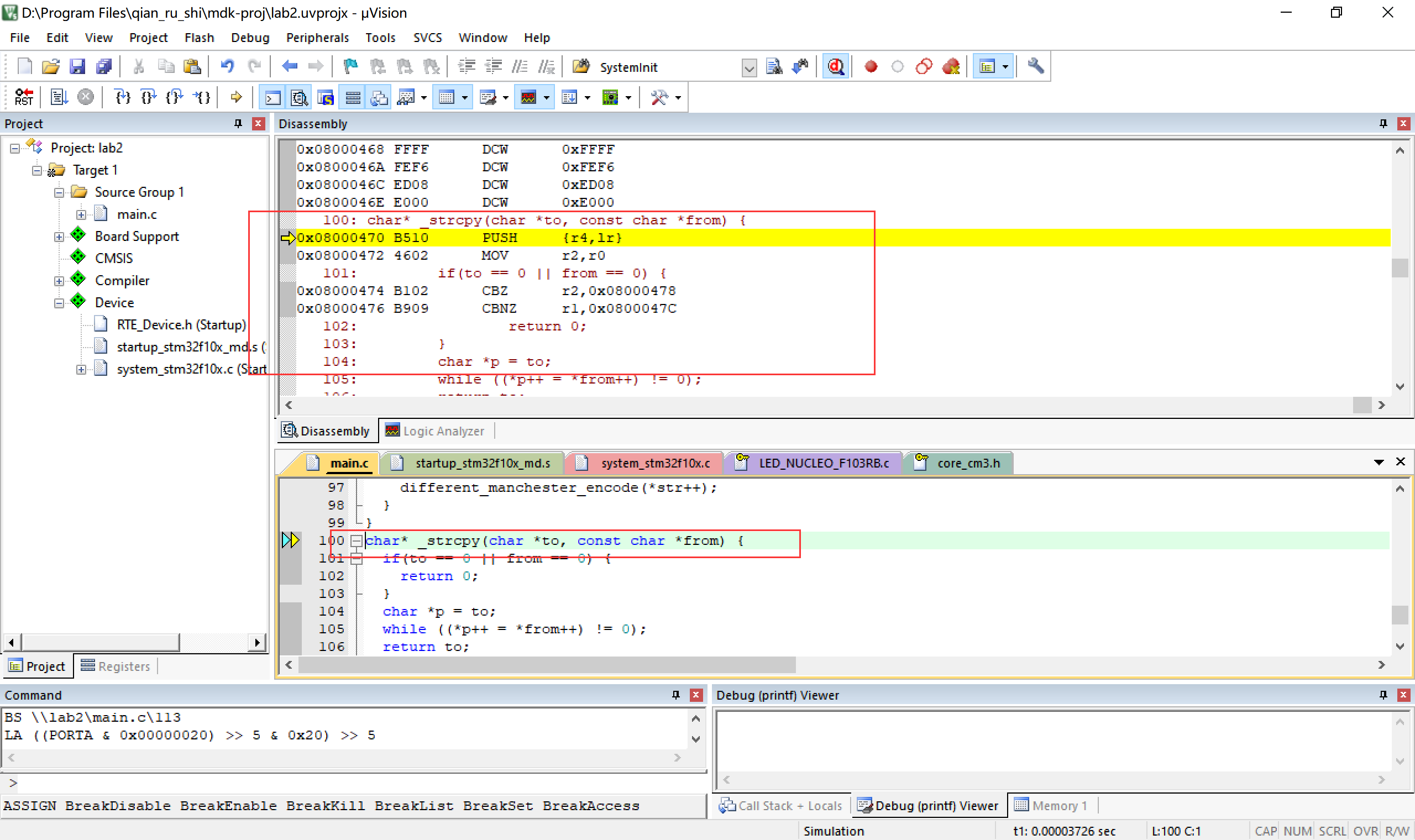
2.然后在main函数里调用该函数，并在该函数入口加上断点，如下图：



3.开始调试，程序运行到函数入口处：



4.单步执行，运行到函数内部：



5.可以从Disassembly窗口得到函数的反汇编代码：

100: char\* \_strcpy(char \*to, const char \*from) {

0x08000470 B510 PUSH {r4,lr}

0x08000472 4602 MOV r2,r0

101: if(to == 0 || from == 0) {

0x08000474 B102 CBZ r2,0x08000478

0x08000476 B909 CBNZ r1,0x0800047C

102: return 0;

103: }

104: char \*p = to;

105: while ((\*p++ = \*from++) != 0);

106: return to;

0x08000478 2000 MOVS r0,#0x00

107: }

0x0800047A BD10 POP {r4,pc}

104: char \*p = to;

0x0800047C 4613 MOV r3,r2

105: while ((\*p++ = \*from++) != 0);

0x0800047E BF00 NOP

0x08000480 F8114B01 LDRB r4,[r1],#0x01

0x08000484 F8034B01 STRB r4,[r3],#0x01

0x08000488 2C00 CMP r4,#0x00

0x0800048A D1F9 BNE 0x08000480

106: return to;

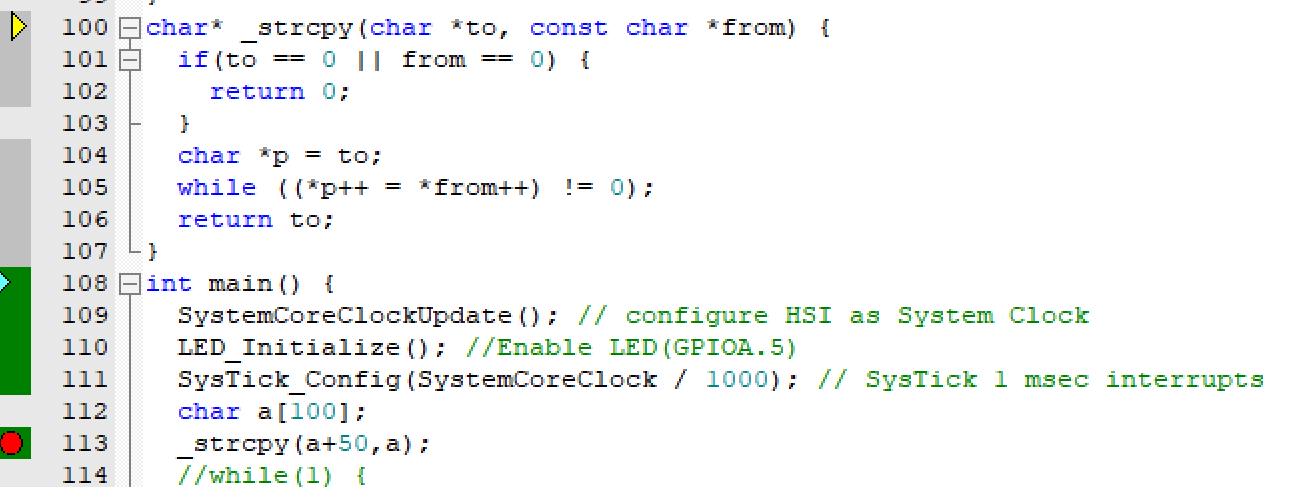
107: }

0x0800048C 4610 MOV r0,r2

0x0800048E E7F4 B 0x0800047A

# 实验方案与实现

## 源代码



# 实验总结

通过本次实验，学习理解了C 与 ARM 汇编的关系，对代码底层有了一个更好的认识，收获颇多。