# 嵌入式系统实验报告



|  |  |
| --- | --- |
| 实验名称： | Lab4 Cortex-M应用编程——C与汇编 |
| 姓 名： | 江姝潼 |
| 学 号： | 2019211653 |
| 学 院(系)： | 计算机学院 |
| 专 业： | 网络工程 |
| 指导教师： | 戴志涛、刘健培 |

2021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，并通过手动操作来生成汇编代码，加强我们对C语言和arm汇编代码间的联系认识。

# 实验环境

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

# 实验要求

1. 阅读相关材料，观察、分析、实验 C 与 ARM 汇编的关系。

2. 将\_strcpy 函数翻译为 ARM 汇编代码，将答案写在作业答卷里。

# 背景知识

本实验旨在理解C语言和汇编指令的关系，C语言主要有定义、赋值、运算符、表达式等操作语句，一般翻译为 1 到多条机器指令。

* **定义：**

1. 如果是 0-255 的常量，可以直接编码在指令里（立即数）。
2. 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。
3. 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。

* **赋值：**

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。

* **运算和表达式：**

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。

* **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。

1. **判断语句**

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。

1. **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。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。

1. **函数**

·函数调用中的寄存器用法。

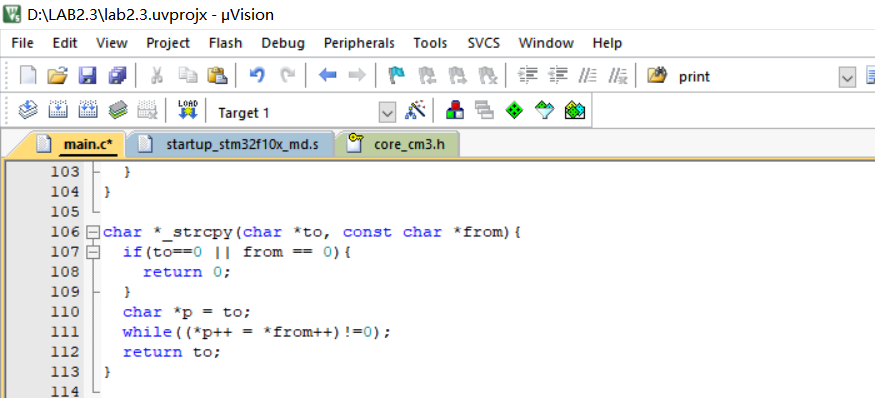
函数或子程序应该保持R4~R11、R13和R14（以及具有FPU的Cortex-M4中的S16~S31）的数值。若这些寄存器在函数或子程序执行期间被修改，则其数值应被保存在栈中并在返回调用代码前恢复。

·参数和返回值传递。对于简单的情况，输入参数可由R0（第1个参数）、R1（第2个参数）、R2（第3个参数）和R3（第4个参数）传递到函数中。对于具有FPU的Cortex-M4，根据所选择的ABI的类型，还可能会用到SO~S15（参见13.4.4节）。函数的返回值通常被存放在RO中，若传递给函数的参数超过4个，则需要使用栈。

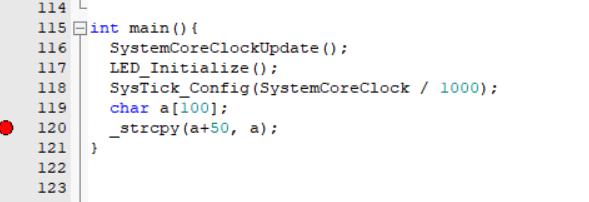
·栈对齐。若汇编函数需要调用C函数，就应该确保当前选择的栈指针指向了双字对齐的地址（如0x20002000、0×20002008和0x20002010等），这是EABI标准规定的。符合EABI的C编译器生成的程序代码会假定栈指针指向了双字对齐的位置，若汇编代码未调用任何C函数（直接或间接），则不必严格遵循这个要求。

# 实验步骤

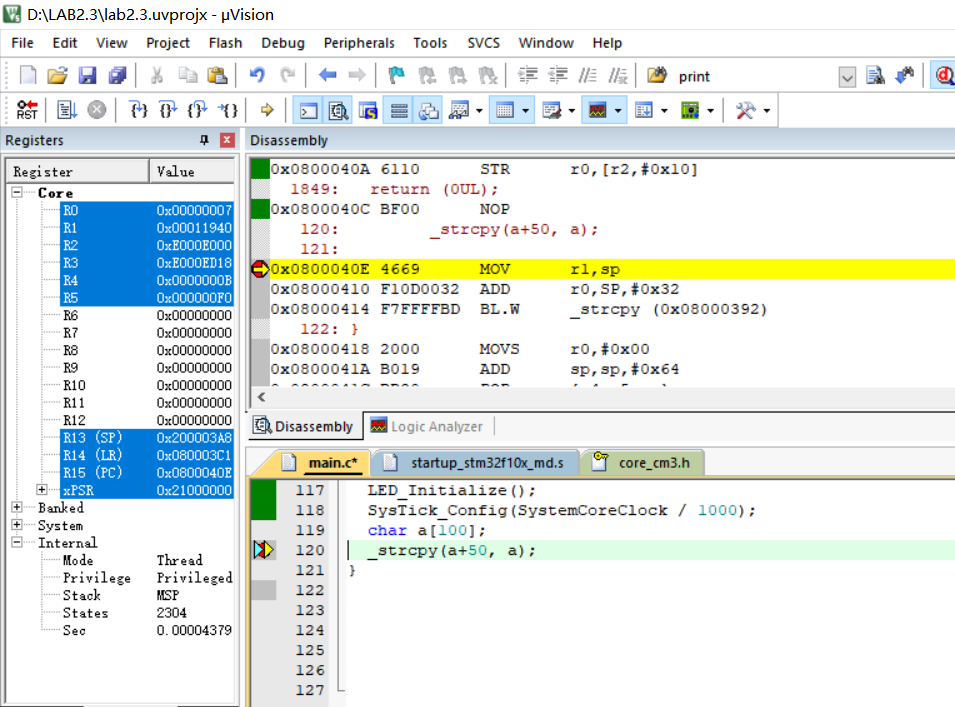
* 在 lab3 的程序基础上加上\_strcpy的函数代码如图所示：



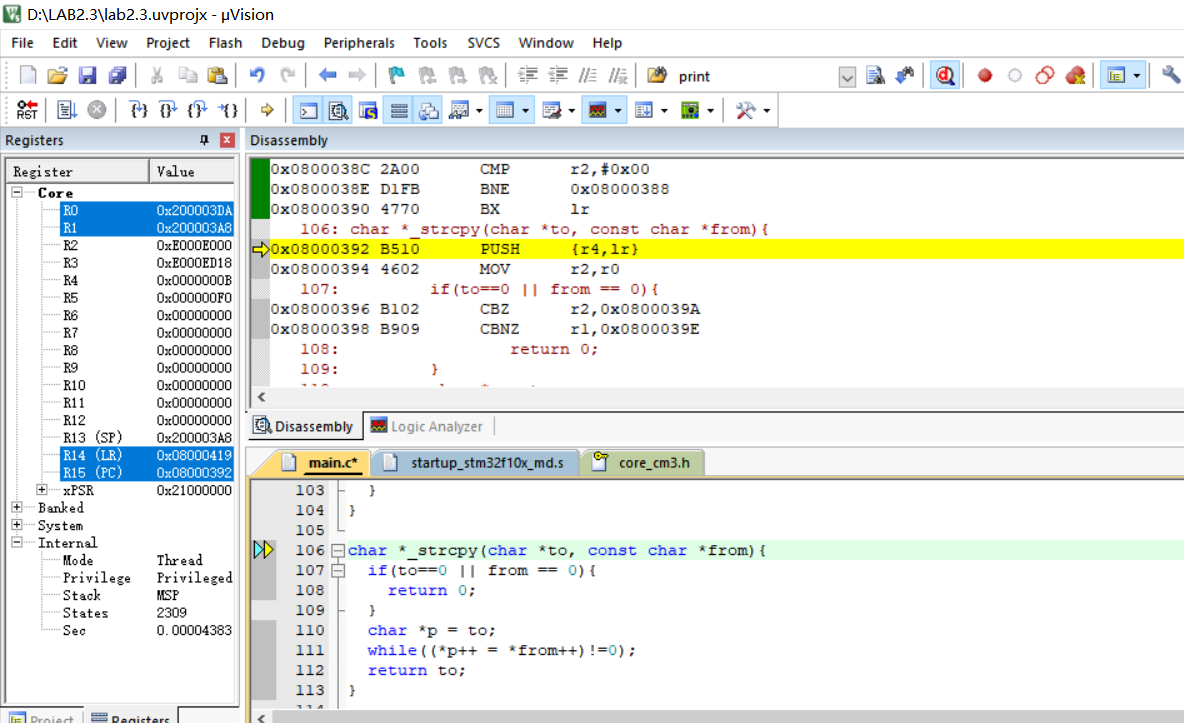
* 在main函数里调用该子函数，并在该函数入口处加上断点：



* 启动start Debug Session，开始执行到该函数的入口处；



* 继续单步向下执行，进入函数内部：



* 从Disassembly窗口获得的函数的反汇编代码如下：

1. 106: **char** \*\_strcpy(**char** \*to, **const** **char** \*from){
2. 0x08000392 B510      PUSH     {r4,lr}
3. 0x08000394 4602      MOV      r2,r0
4. 107:         **if**(to==0 || from == 0){
5. 0x08000396 B102      CBZ      r2,0x0800039A
6. 0x08000398 B909      CBNZ     r1,0x0800039E
7. 108:                 **return** 0;
8. 109:         }
9. 110:         **char** \*p = to;
10. 111:         **while**((\*p++ = \*from++)!=0);
11. 112:         **return** to;
12. 0x0800039A 2000      MOVS     r0,#0x00
13. 113: }
14. 114:
15. 0x0800039C BD10      POP      {r4,pc}
16. 110:         **char** \*p = to;
17. 0x0800039E 4613      MOV      r3,r2
18. 111:         **while**((\*p++ = \*from++)!=0);
19. 0x080003A0 BF00      NOP
20. 0x080003A2 F8114B01  LDRB     r4,[r1],#0x01
21. 0x080003A6 F8034B01  STRB     r4,[r3],#0x01
22. 0x080003AA 2C00      CMP      r4,#0x00
23. 0x080003AC D1F9      BNE      0x080003A2
24. 112:         **return** to;
25. 113: }
26. 114:
27. 0x080003AE 4610      MOV      r0,r2
28. 0x080003B0 E7F4      B        0x0800039C
29. 0x080003B2 0000      MOVS     r0,r0

# 实验总结

本次实验主要聚焦于C语言和汇编语言之间的关系，了解了汇编语言的主要内容和C语言间如何相互转化。实验中通过手动将C语言代码转换为汇编语言，并单步执行观察寄存器的数值变化，了解了C语言代码是如何通过汇编语言这一渠道和硬件执行关联起来的。同时也认识到了，语言越靠近底层，可读写性越差，但也越方便硬件执行这一性质。