嵌入式系统课程设计

题目名称：汇编实验

姓名：王云鹏

学号：8213180228

专业：物联网工程

班级：1802

指导教师：贺建飚

编写日期：2021.6.2

目录

第 2 个实验

1. 问题描述

* 初步学会使用μVision5 IDE for ARM 开发环境及 ARM 软件模拟器；
* 通过实验掌握简单 ARM 汇编指令的使用方法。单步执行观察如 BL、STMFD、LDMFD 等指令的执行情况。
* 熟悉开发环境的使用并使用 ldr/str，mov 等指令访问寄存器或存储单元；使用 add/sub/lsl/lsr/and/orr 等指令，完成基本算术/逻辑运算。

2. 实验设备

系统：Windows10
IDE：μVision IDEfor ARM 集成开发环境

3. 系统设计

本实验思路很清晰，算法步骤如下

1	[sp] = R0 左移八位 + R1 右移一位
2	R2 = z 低八位 + y 右移一位
3	R2 = R2 右移一位 + [sp]或 0x01

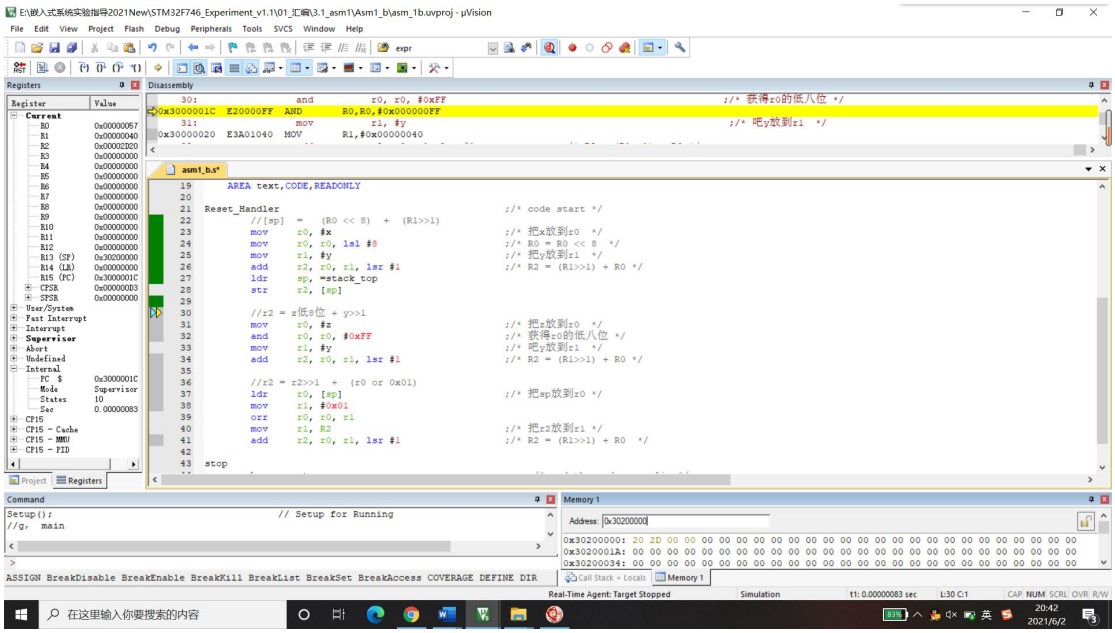
4. 源代码清单

```
;#*******************************************************************************************
;# NAME:        asm1_b.s
                              *
;# Author:  WUHAN R&D Center, Embest
                 *
;# Desc:        ARM instruction examples
                     *
;# History: TianYunFang 2007.05.12
                 *
;#*******************************************************************************************
;/*-------------------------------------------------------------------------------------------*
/
;/*                    constant define
                        */
;/*-------------------------------------------------------------------------------------------*
```

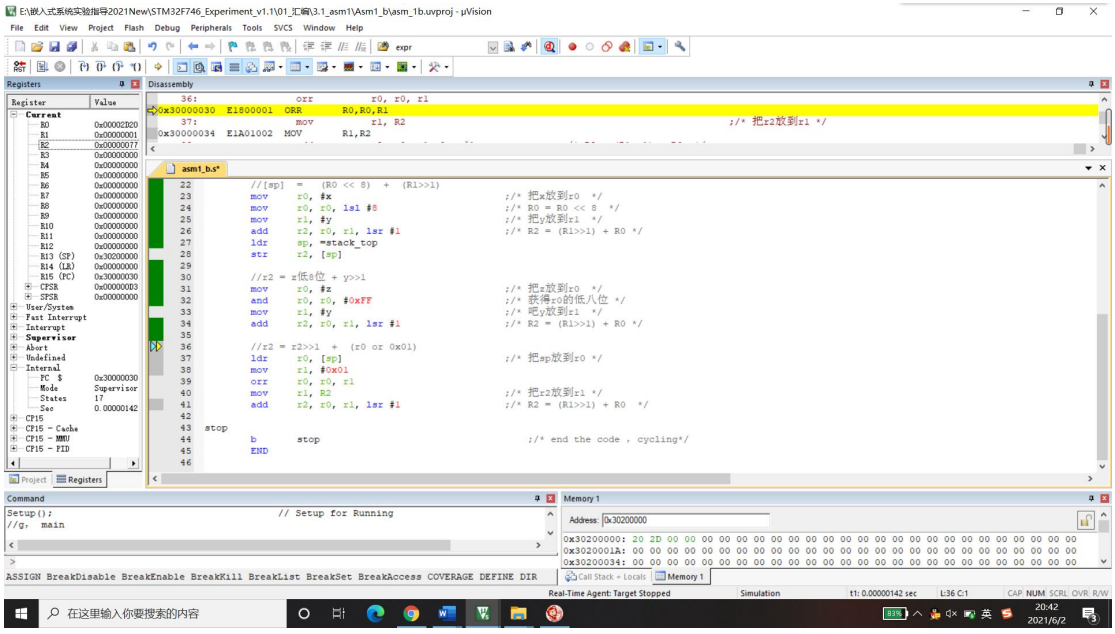
```
/
x           EQU 45                                  ;/* x=45 */
y           EQU 64                                  ;/* y=64 */
z           EQU 87                                    ;/* z=87 */
stack_top   EQU 0x30200000            ;/* define the top address for stacks*/
                              export Reset_Handler

;/*-------------------------------------------------------------------------------------------
*/
;/*                                                 code
                      */
;/*-------------------------------------------------------------------------------------------
*/
    AREA text,CODE,READONLY

Reset_Handler                                       ;/* code start */
        mov     r0, #x                              ;/* put x value into R0  */
        mov     r0, r0, lsl #8                      ;/* R0 = R0 << 8  */
        mov     r1, #y                              ;/* put y value into R1  */
        add     r2, r0, r1, lsr #1                  ;/* R2 = (R1>>1) + R0 */
        ldr     sp, =stack_top
        str     r2, [sp]

        mov     r0, #z                              ;/* put z value into R0  */
        and     r0, r0, #0xFF                       ;/* get low 8 bit from R0 */
        mov     r1, #y                              ;/* put y value into R1  */
        add     r2, r0, r1, lsr #1                  ;/* R2 = (R1>>1) + R0 */

        ldr     r0, [sp]                            ;/* put y value into R1 */
        mov     r1, #0x01
        orr     r0, r0, r1
        mov     r1, R2                              ;/* put y value into R1  */
        add     r2, r0, r1, lsr #1                  ;/* R2 = (R1>>1) + R0  */

stop
        b       stop                                    ;/* end the code ??cycling*/
        END
```

5. 运行结果测试与分析

1：[sp] = R0 左移八位 + R1 右移一位
即第一部分执行完时，右下角显示的内存中为 20 2D 00 00

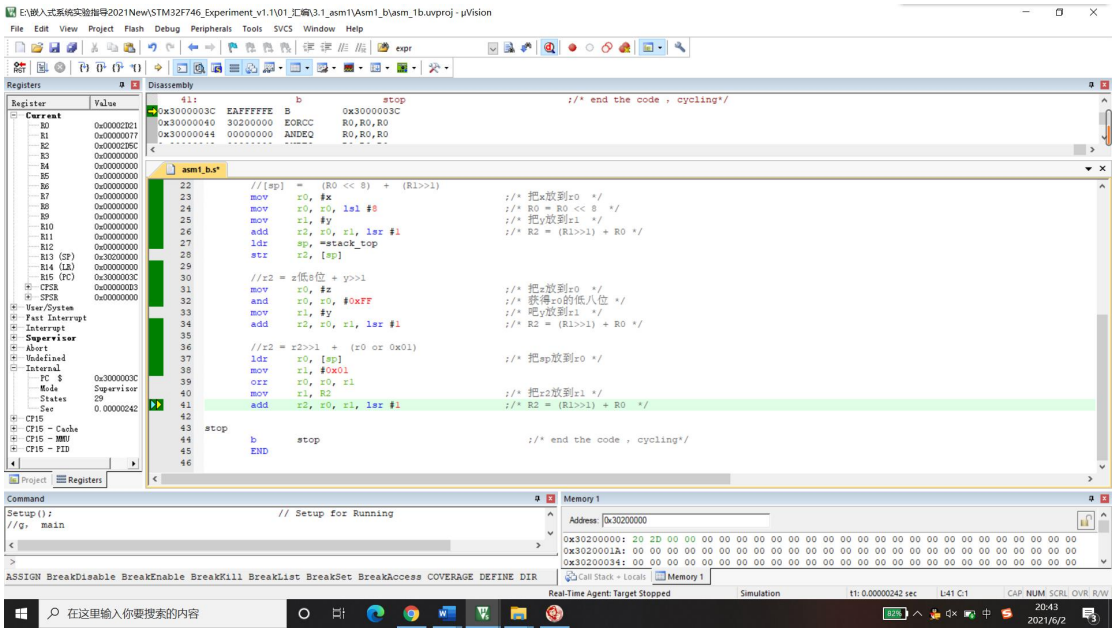


2：R2 = z 低八位 + y 右移一位
此时，左上角 R2 寄存器为 0x77



3：R2 = R2 右移一位 + [sp]或 0x01

此时，左上角 R2 寄存器中为 0x2D5C



6. 结论与心得

- 熟悉了开发环境的使用
- 并掌握了使用 ldr/str/mov 等指令访问寄存器或存储单元
- 能够使用 add/sub/1sl/lsr/and/orr 等指令，完成基本数学/逻辑运算。add/sub/1sl/lsr/and/orr 等指令，完成基本数学/逻辑运算

第 4 个实验

1. 问题描述

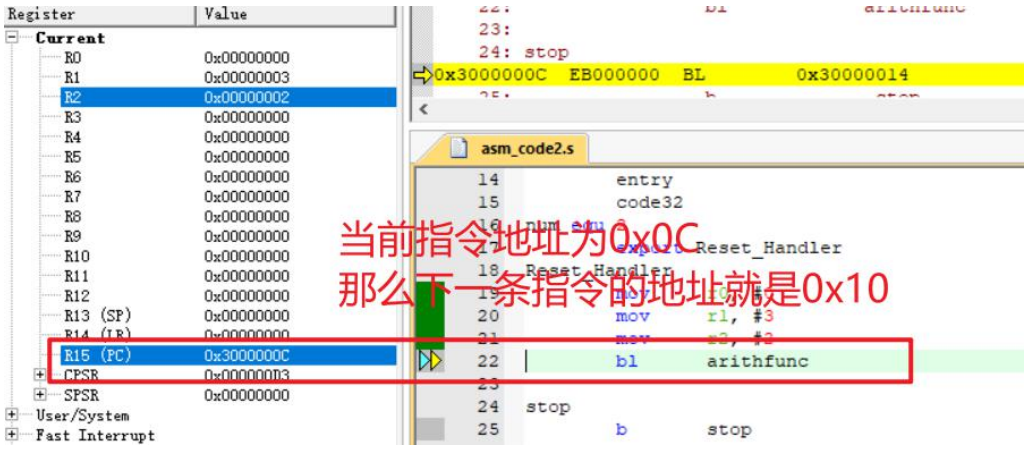
* 熟悉开发环境的使用并完成一块存储区的拷贝。完成分支程序设计，要求判断参数，根据不同参数，调用不同的子程序。
* 通过实验掌握使用 ldm/stm，b，bl 等指令完成较为复杂的存储区访问和程序分支；
* 学习使用条件码，加强对 CPSR 的认识；
* 在指令的执行过程中，通过观察可以对字的存储方式进行判断，了解大端模式和小端模式。

2. 实验设备

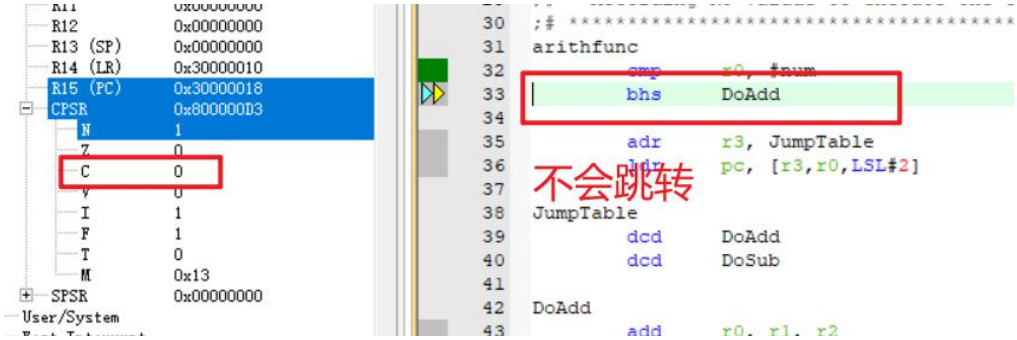
* PC 主机
* µVision IDEfor ARM 集成开发环境， Windows10
* 在指令的执行过程中，通过观察可以对字的存储方式进行判断，了解大端模式和小端模式。

3. 系统设计

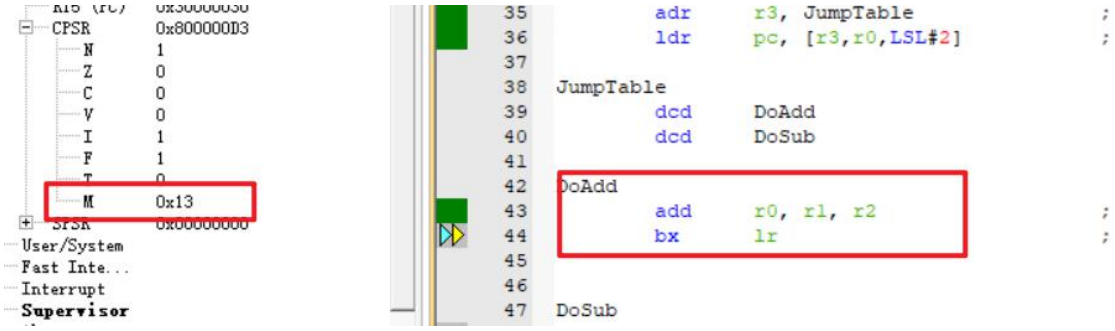
bl 指令：带链接的跳转，将 PC 修改为跳转地址，同时将当前 PC 的下一条指令地址放入 LR，其实就是子程序的调用



bhs 指令



bx：是带状态切换的跳转，最低位为 1 时，切换到 Thumb 指令执行，为 0 时，解释为 ARM 指令执行

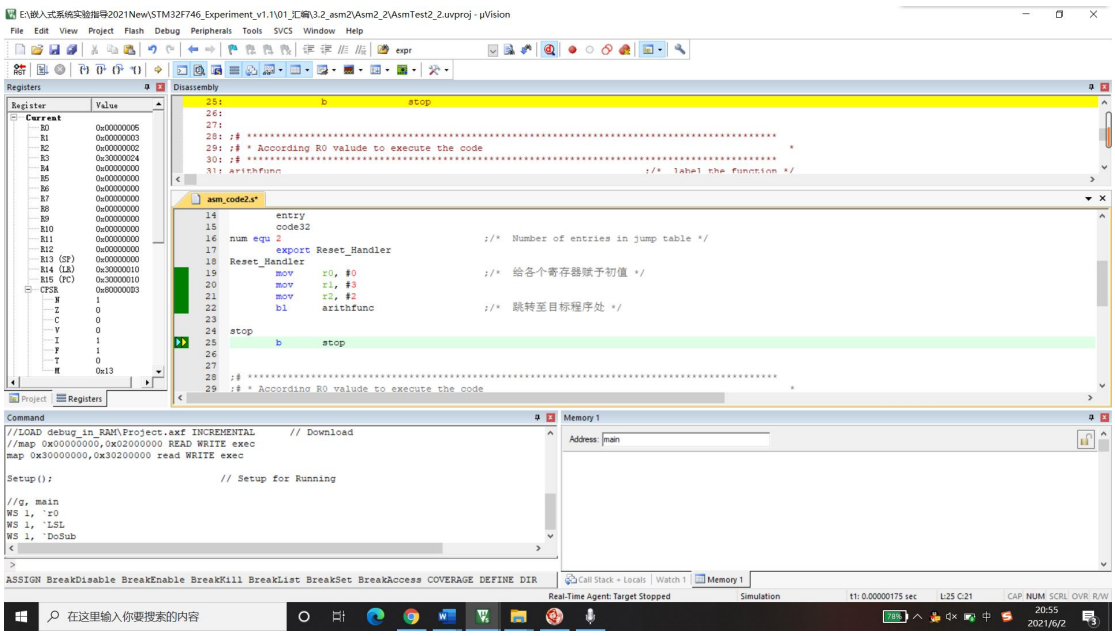


4. 源代码清单

```
;*******************************************************************************************************
*
;# NAME:    ARM_code2.s                                                                                  *
;# Author:  WUHAN R&D Center,Embest
       *
;# Desc:    ARM instruction examples                                                                     *
;#          Example for Condition Code                                                                   *
;# History: shw.He 2005.02.22                                                                            *
;*******************************************************************************************************
*


;/*-------------------------------------------------------------------------------------------------*
/
;/*                                  code                                                           */
;/*-------------------------------------------------------------------------------------------------*
/


        area start,code,readwrite
        entry
        code32
num equ 2                                   ;/*  Number of entries in jump table */
        export Reset_Handler
```

```
Reset_Handler
        mov     r0, #0                  ;/*  set up the three parameters */
        mov     r1, #3
        mov     r2, #2
        bl      arithfunc               ;/*  call the function */

stop
        b       stop


;# ***************************************************************************************
*
;# * According R0 valude to execute the code
   *
;# ***************************************************************************************
*
arithfunc                               ;/*  label the function */
        cmp     r0, #num                ;/*  Treat function code as unsigned integer */
        bhs     DoAdd                   ;/*  If code is >=2 then do operation 0. */

        adr     r3, JumpTable           ;/*  Load address of jump table */
        ldr     pc, [r3,r0,LSL#2]       ;/*  Jump to the appropriate routine */

JumpTable
        dcd     DoAdd
        dcd     DoSub

DoAdd
        add     r0, r1, r2              ;/*  Operation 0, >1 */
        bx      lr                      ;/*  Return */


DoSub
        sub     r0, r1, r2              ;/*  Operation 1 */
        bx      lr                      ;/*  Return */

        end                             ;/*  mark the end of this file */
```

5. 运行结果测试与分析



6. 结论与心得

- 为了理解 ldr 指令，汇编实验 1.1，1.2 中出现了 ldr sp,=stack-_top 这条汇编语句，其相当于执行 ldr R13，[pc,#offset],意思是将 sp 中存储的地址值存到 pc 加上 offset 中。而由 pc 加上 offset 后找到 pc 内存地址中存储的值并不是预先定义 stack_top 地址值 0x30200000，而是本条指令的 pc 值上加了 8，相当于移动了两条指令，这里变化涉及到了 ldr 指令在流水系统中会预取下一条指令。ARM9 每条指令执行是加 4。

自己设计的实验

1. 问题描述

用汇编实现快速排序

2. 实验设备

* PC 主机
* μVision IDEfor ARM 集成开发环境， Windows10

3. 系统设计

快速排序使用分治法（Divide and conquer）策略来把一个序列（list）分为较小和较大的 2 个子序列，然后递归地排序两个子序列。步骤为：

挑选基准值：从数列中挑出一个元素，称为“基准”（pivot），
分割：重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准后面（与基准值相等的数可以到任何一边）。在这个分割结束之后，对基准值的排序就已经完成，
递归排序子序列：递归地将小于基准值元素的子序列和大于基准值元素的子序列排序。
递归到最底部的判断条件是数列的大小是零或一，此时该数列显然已经有序。

4. 源代码清单

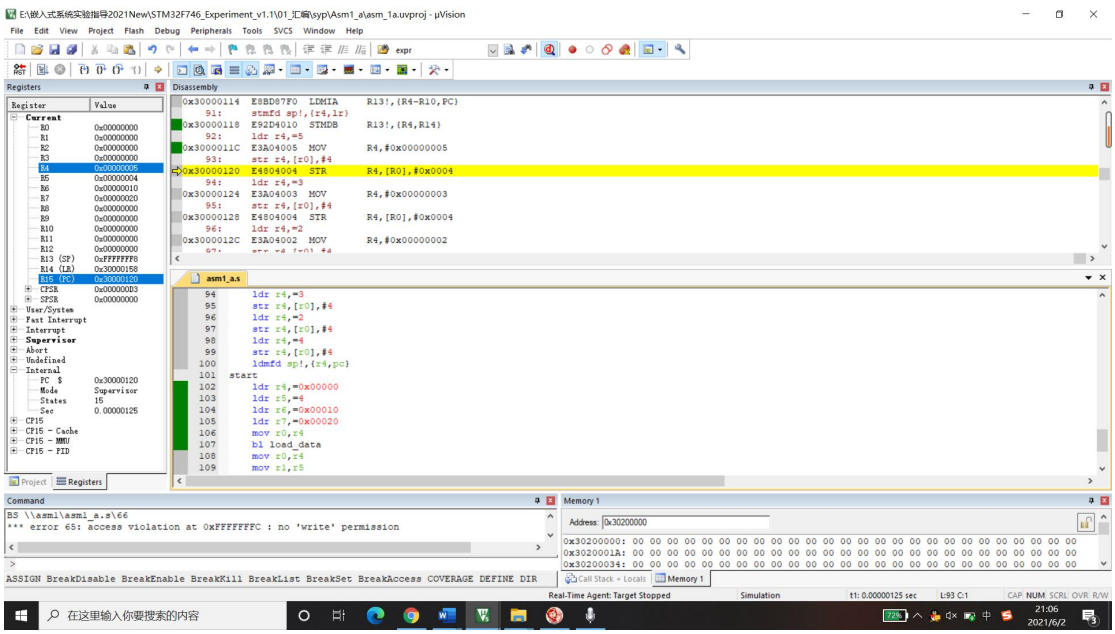
```
        area word,code,readonly
        entry
        b start
move
        stmfd       sp!, {r4-r5,lr}
        ldr r4,=0
        cmp r1,r4
        ble _move_return
        add r4,r4,#1
_move_next
                ldr r5,[r0],#4
                str r5,[r2],#4
                add r4,r4,#1
```

```
            cmp r4,r1
            ble _move_next
_move_return
      ldmfd      sp!, {r4,r5,pc}
append
      stmfd      sp!, {r4,lr}
      mov r4,r2,lsl#2
      str r0,[r1,r4]
      ldmfd      sp!, {r4,pc}
sort
      stmfd      sp!, {r4-r10,lr}
      ldr r4,=1
      cmp r1,r4
      ble _sort_return
      mov r4,r0
      mov r5,r1
      mov r6,r2
      mov r7,r3
      ldr r8,=2
      ldr r9,=0
      ldr r10,=0
      ldr r3,[r4],#4
_sort_next
            ldr r0,[r4],#4
            cmp r0,r3
            ble _sort_small_append
_sort_big_append
                  add r10,r10,#1
                  mov r1,r7
                  mov r2,r10
                  b _sort_next_append
_sort_small_append
                  add r9,r9,#1
                  mov r1,r6
                  mov r2,r9

_sort_next_append
                  bl append
            add r8,r8,#1
            cmp r8,r5
            ble _sort_next
      mov r0,r3
                  add r9,r9,#1
                  mov r1,r6
```

```
                    mov r2,r9
                    bl append
        mov r0,r5,lsl#2
        sub r4,r4,r0
         mov r0,r6
         mov r1,r9
         mov r2,r4
         bl move


         mov r0,r9,lsl#2
         add r0,r0,#4
         add r2,r4,r0
         mov r0,r7
         mov r1,r10

         bl move
         mov r0,r4
         mov r1,r9
         mov r2,r6
         mov r3,r7
         bl sort
         add r0,r4,r9
         mov r1,r10
         mov r2,r6
         mov r3,r7
         bl sort
_sort_return
        ldmfd sp!,{r4-r10,pc}

load_data
        stmfd sp!,{r4,lr}
        ldr r4,=5
        str r4,[r0],#4
        ldr r4,=3
        str r4,[r0],#4
        ldr r4,=2
        str r4,[r0],#4
        ldr r4,=4
        str r4,[r0],#4
        ldmfd sp!,{r4,pc}
start
        ldr r4,=0x00000
        ldr r5,=4
```

```
        ldr r6,=0x00010
        ldr r7,=0x00020
        mov r0,r4
        bl load_data
        mov r0,r4
        mov r1,r5
        mov r2,r6
        mov r3,r7
        bl sort
stop
        mov r0,#0x18
        ldr r1,=0x20026
        swi 0x123456
end
```

5. 运行结果测试与分析



6. 结论与心得

stmfd 指令,stmfd sp!,{r4-r11}为堆栈寻址方式，组合方式为空向下生长型；堆栈指针 sp 在加上！符号后，sp 减 4 后将 r4 值存入，sp 相应更新直到寄存器表{r4-r11}入栈完毕。实验 1.3、1.4 中出现的 ARM 转移类指令包括 B、BX、和 BL。以 BX 指令为例，BX{cond} Rm，其中：ARM 是含转移地址的寄存器，通过把 Rm 的内容拷贝到 PC 实现转移。若 Rm 的位 0 为 0，这 CPSR 中的标志 T 置位，目标地址代码解释为 Thumb 代码；若 Rm 的位为 0 为 0，则位 1 就不能为 1。