Lab 3 汇编程序设计 report

PB21111681 朱炜荣

实验目的与内容

1. 了解并理解 RISC-V 常用的几个 32 位整数指令功能
2. 掌握 RISC-V 简单汇编程序设计，以及下载测试数据(COE 文件) 的生成方法
3. 熟悉 RISC-V 汇编程序仿真运行环境和调试基本方法

逻辑设计

1. 实验一的逻辑为：

利用寄存器数值中强制为 0 的 0 号寄存器，先测试 beq 指令，通过 beq 的正常跳转与否，来判断接下来的指令。如果 beq 正常跳转，则 t5 寄存器的值为 0，否则 t5 寄存器的值为 1（通过验证后续 addi 指令是否正确可以判断 beq 的跳转）。如果跳转成功，则可以利用 beq 来判断 jal 指令的正确性，即成功则继续顺序执行，失败则会利用 beq 的跳转至失败状态处。然后我们测试 lw 指令，利用 lw 读取数值并与 0 比较来判断 lw 是否成功。同样失败的话利用 jal 跳转。将 lw、jal、beq 指令验证完毕之后，其余的各种指令便通过设计容易判断是否正确。具体细节可以查看 test.asm 源代码。

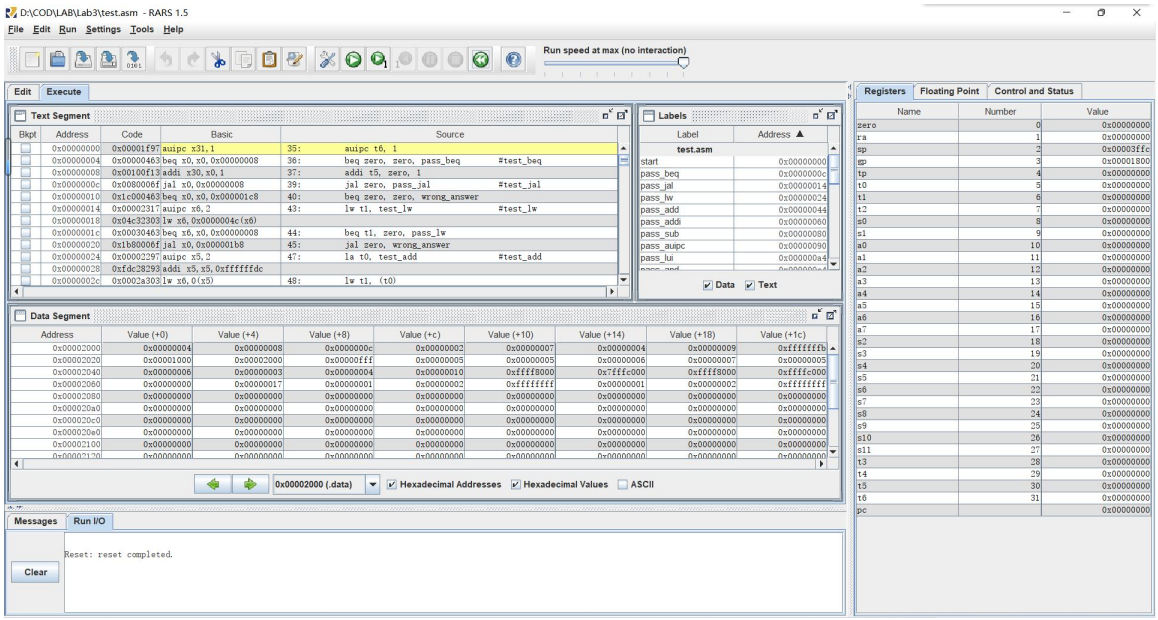
2. 实验二的逻辑为：

先读取要排序的数组长度。然后利用数组长度进行简单的冒泡排序，分别读取对应地址的两个数进行比较，如果更靠前的数更小，则将两个数交换位置，即分别将对方的数值输入到内存中。第一轮循环后将最大的数放到数组的第一位，然后移动“第一位”到第二位再进行循环，直到“第一位”到达最后一位。

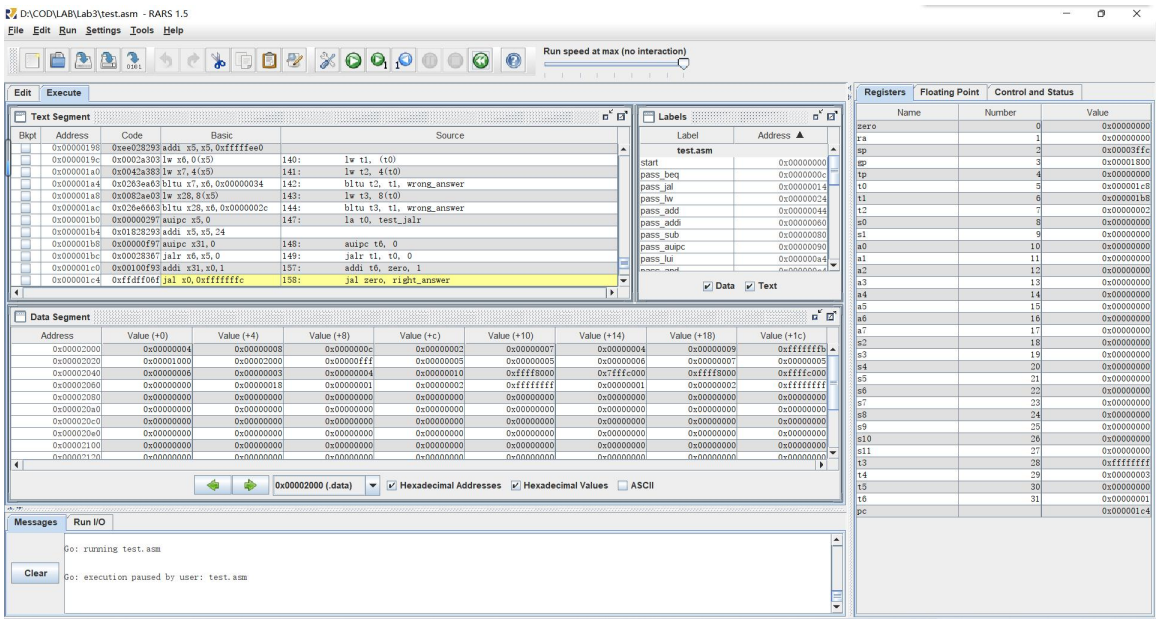
测试结果与分析

1. 实验一：

在运行之前整体的情况为：



在运行之后得到的结果为：

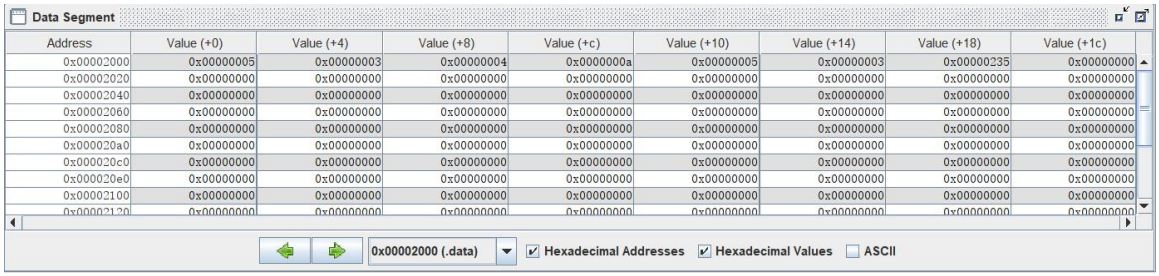


其中 t5 寄存器代表了 beq 指令是否正常，t5 为 0 则正常，t5 为 1 则异常。

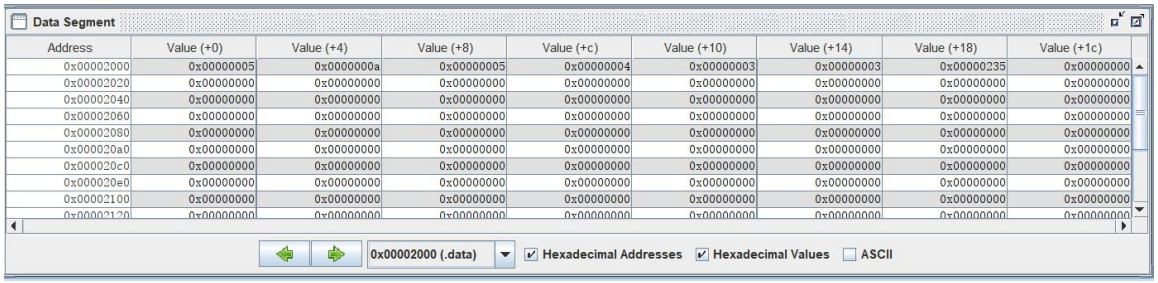
t6 寄存器代表了当前状态是什么状态，t6 为 1 则为正常状态，t6 为 0 则为异常状态。

2. 实验二：

排序前的内存数值为：（其中数值的第一位为要排序的数组长度）



进行排序之后得到的结果为：



总结

本次实验总体而言难度较为正常，同时以实验督促学习，在实验中我进一步学习了有关 RISC-V 的有关语法和相关指令知识。同时借助 RARS 软件学会了如何将编写的汇编程序转变成机器码，也即生成 COE 文件的过程。同时学习如何在 RARS 中调试 RISC-V 汇编程序。本次实验的最大难度在于，如何设计合适的测试指令顺序使得，在后序测试需要的指令在该指令测试之前就要有所测试。最开始我想先测试 jal 指令，但在助教的提醒之下发现程序内存在逻辑错误，程序过程有点想自举，就不在此展示了。然后是写了多次的冒泡排序，我们先在 C 语言里学习、然后实现。再然后是 python、然后是 LC-3、接着是上次的 Verilog，这次是 RISC-v，不知道是不是冒泡排序在排序中的地位与 hello，world 在编程中的地位一样，写了很多次。不过好在都有惊无险地写出来了，也还算得上是一个基本的计科人吧。

附加代码

test.asm

```
1.  .data
2.  test_add:
3.   .word 4, 8, 12
```

```
test_addi:
 .word 2, 7
test_sub:
 .word 4, 9, -5
test_auipc:
 .word 4096
test_lui:
 .word 8192
test_and:
 .word 4095, 5
test_or:
 .word 5, 6, 7
test_xor:
 .word 5, 6, 3
test_slli:
 .word 4, 16
test_srli:
 .word -32768, 2147467264
test_srai:
 .word -32768, -16384
test_lw:
 .word 0
test_sw:
 .word 23
test_blt:
 .word 1, 2, -1
test_bltu:
 .word 1, 2, -1

.text
start:
 auipc t6, 1
 beq zero, zero, pass_beq #test_beq
 addi t5, zero, 1
pass_beq:
 jal zero, pass_jal  #test_jal
 beq zero, zero, wrong_answer
pass_jal:

 lw t1, test_lw   #test_lw
 beq t1, zero, pass_lw
 jal zero, wrong_answer
pass_lw:
 la t0, test_add   #test_add
 lw t1, (t0)
 lw t2, 4(t0)
 lw t3, 8(t0)
 add t4, t1, t2
```

```
  beq t3, t4, pass_add
  jal zero, wrong_answer
pass_add:
  la t0, test_addi  #test_addi
  lw t1, (t0)
  lw t2, 4(t0)
  addi t1 , t1, 5
  beq t1, t2, pass_addi
  jal zero, wrong_answer
pass_addi:
  la t0, test_sub   #test_sub
  lw t1, (t0)
  lw t2, 4(t0)
  lw t3, 8(t0)
  sub t4, t1, t2
  beq t3, t4, pass_sub
  jal zero, wrong_answer
pass_sub:
  lw t1, test_auipc  #test_auipc
  beq t6, t1, pass_auipc
  jal zero, wrong_answer
pass_auipc:
  lw t1, test_lui   #test_lui
  lui t2, 2
  beq t1, t2, pass_lui
  jal zero,wrong_answer
pass_lui:
  la t0, test_and   #test_and
  lw t1, (t0)
  lw t2, 4(t0)
  and t3, t1, t2
  addi t4, zero, 5
  beq t3, t4, pass_and
  jal zero, wrong_answer
pass_and:
  la t0, test_or    #test_or
  lw t1, (t0)
  lw t2, 4(t0)
  lw t3, 8(t0)
  or t4, t1, t2
  beq t3, t4, pass_or
  jal zero, wrong_answer
pass_or:
  la t0, test_xor   #test_xor
  lw t1, (t0)
  lw t2, 4(t0)
  lw t3, 8(t0)
  xor t4, t1, t2
```

```
 beq t3, t4, pass_xor
 jal zero, wrong_answer
pass_xor:
 la t0, test_slli  #test_slli
 lw t1, (t0)
 lw t2, 4(t0)
 slli t1 , t1, 2
 beq t1, t2, pass_slli
 jal zero, wrong_answer
pass_slli:
 la t0, test_srli  #test_srli
 lw t1, (t0)
 lw t2, 4(t0)
 srli t1 , t1, 1
 beq t1, t2, pass_srli
 jal zero, wrong_answer
pass_srli:
 la t0, test_srai  #test_srai
 lw t1, (t0)
 lw t2, 4(t0)
 srai t1 , t1, 1
 beq t1, t2, pass_srai
 jal zero, wrong_answer
pass_srai:
 la t0, test_sw   #test_sw
 lw t1, (t0)
 addi t2, t1,1
 sw t2, (t0)
 lw t3, (t0)
 beq t1, t3, wrong_answer

 la t0, test_blt   #test_blt
 lw t1, (t0)
 lw t2, 4(t0)
 blt t2, t1, wrong_answer
 lw t3, 8(t0)
 blt t3, t1, pass_blt
 jal zero, wrong_answer
pass_blt:
 la t0, test_bltu  #test_bltu
 lw t1, (t0)
 lw t2, 4(t0)
 bltu t2, t1, wrong_answer
 lw t3, 8(t0)
 bltu t3, t1, wrong_answer


 la t0, test_jalr
```

```
 auipc t6, 0
 jalr t1, t0, 0

right_answer:
# la a0, str2
#        li a7, 4
#        ecall
# li a7, 10
#        ecall
  addi t6, zero, 1
        jal zero, right_answer


test_jalr:
 addi t1, t1, -8
 beq t1, t6, pass_jalr
 jal zero, wrong_answer
pass_jalr:
 jal zero, right_answer

wrong_answer:
# la a0, str1
#       li a7, 4
#        ecall
# li a7, 10
#        ecall
        and t6, t6, zero
        jal zero, wrong_answer
        beq zero, zero, wrong_answer

```

sort.asm

```
    .text
    start:
     la a2, array
     la a3, n
     lw t1, (a3)
     addi t1, t1, -1 # t1 = n - 1
    loop_i:
     blt t1, zero, end # t1 >= 0?
     and t2, t2, zero # t2 = 0
     and t3, t3, zero # t3 = 0
    loop_j:
     beq t2, t1, exit_j # t2 == t1
     slli t3, t1, 2  # t3 = t1 * 4
```

```
 add t3, a2, t3
 lw t5, (t3)
 slli t4, t2, 2  # t4 = t2 * 4
 add t4, a2, t4
 lw t6, (t4)
 blt t5, t6, smaller
 sw t6, (t3)
 sw t5, (t4)
smaller:
 addi t2, t2, 1
 jal zero, loop_j
exit_j:
 addi t1, t1, -1
 jal zero, loop_i
end:
 jal zero, end


.data
n:
 .word 5
array:
 .word 3,4,10,5,3,565

```