CODH - 汇编程序设计　实验报告

院系:　　姓名:　　学号:

2023 年 7 月 13 日

1　实验目的

- 理解 RISC-V 常用 32 位整数指令功能
- 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
- 熟悉 RISC-V 汇编程序仿真运行环境和调试基本方法

2　实验环境

- macOS 13.0
- Rars1_5.jar (Riscv Assembler and Runtime Simulator)

3　实验内容

3.1　设计汇编程序，自动测试以下指令功能

- add, addi, sub, auipc, lui
- and, or, xor
- slli, srli, srai
- lw, sw
- beq, blt, bltu, jal, jalr

3.2　设计汇编程序，实现可变长数组排序

- 数据结构：数组大小，数据元素，
- 数据类型：大小和元素均为 32 位无符号数
- 排序算法：算法不限，升序或降序排序

之后将其与串行调试单元模块（SDU_DM）整合后，下载至 FPGA 中测试。

4 逻辑设计 / 核心代码

4.1 设计汇编程序，自动测试以下指令功能

4.1.1 逻辑设计

这里完成了两种测试方式，一种是在测试某一条指令时默认其余指令都可用，通过载入寄存器、执行指令、比较结果并输出三步来完成指令的测试。这里针对 18 条指令都构建了对应的测试子程序。

另一种方式是从零开始，先检测不需要任何其他指令的 beq 指令，再以此为基础逐渐添加其他指令的测试，每条指令的测试只可使用之前已经测试完成的指令。对于测试结果的表示使用了将寄存器的相应位置为 1，并且陷入相应的死循环以保存结果显示，这里可能需要用到未经测试的指令（slli）。

4.1.2 核心代码 1

```
test_add:
    la a1, str_add
    lw a2, arg0
    lw a3, arg1
    lw a4, ans_add
    add a5, a2, a3
    la a0, str_pass
    beq a4, a5, add_pass
    la a0, str_fail
add_pass:
    ret
```

test.asm

4.1.3 核心代码 2

```
main:
    addi a6, zero, 1
jal_test:
    jal beq_test
    slli a6, a6, 1
    jal wrong

beq_test:
    beq zero, zero, lw_test
    slli a6, a6, 2
    jal wrong

lw_test:
    lw a0, arg0
    lw a1, arg1
    beq a0, a1, sw_test
    slli a6, a6, 3
    jal wrong

    ...

```

test1.asm

4.1.4 代码运行

使用 Rars1_5.jar 仿真运行，可以看到两份代码都测试结果正确。第一种测试:

```
Expected: 12 , Actual: 12.
Test of [ ADD   ] is passed.

Expected: -2 , Actual: -2.
Test of [ SUB   ] is passed.

Expected: 25 , Actual: 25.
Test of [ ADDI  ] is passed.

Expected: 268435796 , Actual: 268435796.
Test of [ AUIPC ] is passed.

Expected: 268435456 , Actual: 268435456.
Test of [ LUI   ] is passed.

Expected: 5 , Actual: 5.
Test of [ AND   ] is passed.
```

图 1: test.asm 运行结果

第二种测试：通过断点可以看到最终陷入表示正确的死循环，且寄存器 a6 说明运行结果正确。

a1	11	0xc0000000
a2	12	0x00000026
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000001
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000

图 2: test1.asm 运行结果

4.2 设计汇编程序，实现可变长数组排序

4.2.1 逻辑设计

依然采用冒泡排序，先用高级语言写出后参照结构更改为汇编语言:

```
1    for (int i = 1; i <= n - 1; i++)
2        for (int j = 1; j <= n - i + 1; j++)
3        if(M[j] > M[j+1])
```

```
4                 { swap(M[j], M[j+1]); }
```

sort.c

由此可以写出以下汇编代码，注意循环变量变化方向以及步长有所不同，这是为了匹配 RISC-V 的地址格式（按字存储）：

4.2.2 核心代码

```
1  sort:
2      mv t0, a1
3      slli t0, t0, 0x2
4      addi t6, zero, 0x8
5  outer_loop:
6      addi t0, t0, -0x4
7      addi t1, zero, 0
8  inner_loop:
9      add t2, a0, t1
10     addi t1, t1, 0x4
11     lw t3, 0(t2)
12     lw t4, 0x4(t2)
13     blt t3, t4, inner_loop_end
14     sw t4, 0(t2)
15     sw t3, 0x4(t2)
16 inner_loop_end:
17     blt t1, t0, inner_loop
18     bge t0, t6, outer_loop
19     ret
```

srt.asm

4.2.3 代码运行

使用.data 初始化数据段为 x10, xf, xe, xd, xc, xb, xa, x9, x8, x7, x6, x5, x4, x3, x2, x1, x0. 然后在主函数部分，先输出数组内容、进行排序、再输出排序后的数组内容，结果如下：

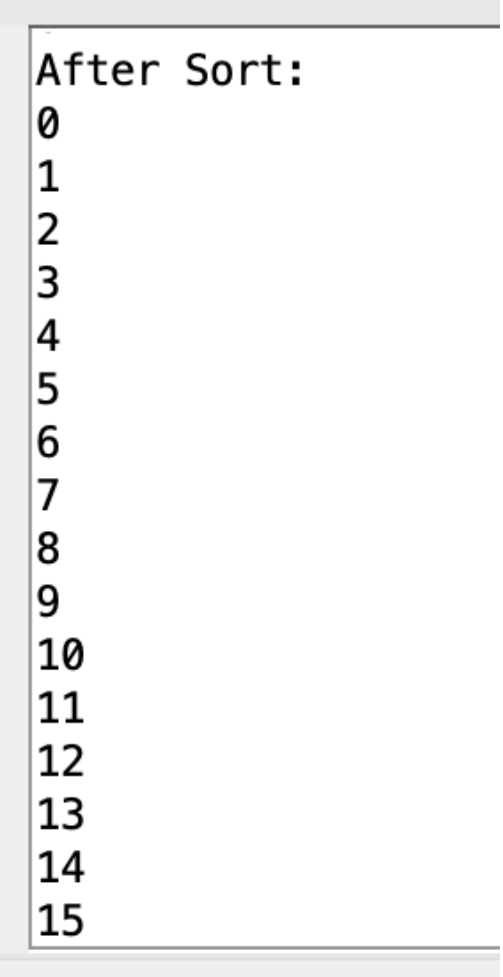

图 3: sort.asm 运行结果

5 生成 COE 文件

使用 File » Dump Memory，导出代码和数据。

并且将导出文本的开头添加以下两行

```
memory_initialization_radix  = 16;
memory_initialization_vector =
...
```

srt.asm

以生成 COE 文件。

6 实验总结

1. 通过本次实验，我学习了 RISC-v 指令集的基本指令以及尝试了使用其编写程序，对 RISC-v 指令集有了更深刻的认识。
2. 我理解了如何合理设计指令测试顺序，以及学习了指令测试间的依赖关系分析。
3. 还学习了如何使用 Rars 仿真器进行指令测试，以及通过设置断点，查看寄存器的方式进行调试。

7 意见/建议

无，这个实验设计很完美。