# 实验6 实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 叶福伦 | 学号 | 21051331 |
| 专业 | 计算机专业 | 班级 | 21052313 |
| 指导教师 | 高志刚 | 课程名称 | 计算机组成原理课程设计（甲） |
| 实验序号 | 实验6 | 实验名称 | RISC-V汇编器与模拟器实验 |
| 实验时间 | 05.09 | 实验地点 | 1教223 |

# 成绩表

|  |  |  |
| --- | --- | --- |
| **考核项目** | **考查分项** | **得分** |
| **一、实验目的（5分）** | **完整性** |  |
| **二、实验要求及完成情况（75分）** | **实验要求1（5分）** |  |
| **实验要求2（20分）** |  |
| **实验要求3（20分）** |  |
| **实验要求4（20分）** |  |
| **实验要求5（10分）** |  |
| **三、思考与探索（15分）** | **问题与解决方案（5分）** |  |
| **拓展提升（10分）** |  |
| **格式（5分）** | **书写规范、图表清晰、表达清楚** |  |
| **合计** | |  |

**一、实验目的**

（5分）

学习RISC-V的RV32I指令集，熟悉其指令格式、汇编指令助记符，掌握机器指令编码方法；

学习RV32I汇编程序设计，学会使用RISC-V交叉编译器、汇编器，将高级语言程序翻译成汇编语言程序，进而翻译成二进制文件；同时，学会反汇编方法；

了解使用RISC-V的模拟器运行程序的方法。

**二、实验要求及完成情况**

实验要求1（5分）：

安装好带有RISC-V交叉编译器和模拟器的Linux虚拟机，并用hello.c程序验证，确保能使用。

完成情况：

[ashenye@meteor-04 step3]$ riscv32-unknown-elf-gcc hello.c -o hello

[ashenye@meteor-04 step3]$ riscv32-unknown-elf-objdump -d hello > dump32.txt

[ashenye@meteor-04 step3]$ cat dump32.txt | more

hello: file format elf32-littleriscv

Disassembly of section .text:

00010094 <exit>:

10094: ff010113 add sp,sp,-16

10098: 00000593 li a1,0

1009c: 00812423 sw s0,8(sp)

100a0: 00112623 sw ra,12(sp)

100a4: 00050413 mv s0,a0

100a8: 400000ef jal 104a8 <\_\_call\_ex

itprocs>

100ac: 0301a503 lw a0,48(gp) # 14ec

0 <\_global\_impure\_ptr>

100b0: 03c52783 lw a5,60(a0)

100b4: 00078463 beqz a5,100bc <exit+0

x28>

--More--

实验要求2（20分）：

将下面一段汇编语言程序acc.s，借助汇编器和反汇编器，将其翻译成机器语言程序；分析其功能；分析j Loop1是用什么指令实现的？

完成情况：

[ashenye@meteor-04 step4]$ riscv32-unknown-elf-gcc acc.s -s acc.o

[ashenye@meteor-04 step4]$ riscv32-unknown-elf-objdump -d acc.o > dump.txt

[ashenye@meteor-04 step4]$ cat dump.txt

t

acc.o: file format elf32-littleriscv

Disassembly of section .text:

00000000 <main>:

0: 000002b3 add t0,zero,zero

4: 00000333 add t1,zero,zero

8: 00a00393 li t2,10

0000000c <L1>:

c: 04032e03 lw t3,64(t1)

10: 01c282b3 add t0,t0,t3

14: 00430313 add t1,t1,4

18: fff38393 add t2,t2,-1

1c: 00038463 beqz t2,24 <L2>

20: fedff06f j c <L1>

00000024 <L2>:

24: 08502023 sw t0,128(zero) # 80 <L2+0x5c>

累加地址为0x40, 0x44, 0x48, 0x4C, 0x50, 0x54, 0x58, 0x5C, 0x60, 0x64, 0x68,

0x6C, 0x70, 0x74, 0x78, 0x7C的数据到t0寄存器并输出到0x80上的内存字

j 是伪命令, j <Label> 会被翻译成 jal x0, <Label>

实验要求3（20分）：

子程序BankMove：将主存一个地址连续的数据块（数组）复制到主存另一个区域，3个入口参数：

a0：源数据区域的首地址

a1：目标数据区域的首地址

a2：复制的数据个数（数组长度）

主程序main：调用BankMove，从内存区域30H复制10个数据到60H

(1)请对上述程序进行汇编与反汇编；

(2)说明子程序调用和返回的具体实现过程：解析子程序调用jal和返回指令jr的机器代码，计算它们的转移地址，写出指令执行的具体操作与结果，指出跳转目标地址的指令。

完成情况:

(2) jal bankmove 等价于jal 0 offset

offset = -34h, 转移地址为0x0，即为bankmove函数开头

jr ra是伪命令，会被展开为jalr x0, 0(ra)

转移地址跟ra寄存器相关，转移地址为0x38

a0寄存器设为0x30, a1寄存器设为0x60, a2寄存器设为10然后执行jal bankmove调用bankmove子程序

bankmove子程序先把a0-a2的值写到t0-t2寄存器 t3寄存器写入t0寄存器对应的内存单元的值, 再写入

t1寄存器对应的内存单元。然后t0, t1寄存器指向后一个内存单元，t2寄存器作为计数器减1，如果t2寄存器不等于零则跳转到L1, 直到0x30-0x58的内存单元的值都被复制到0x60-0x88的内存单元中。之后复制完成返回父程序main函数执行完成

24: 00008067 ret

34: fcdff0ef jal 0 <bankmove>

20: fe0396e3 bnez t2,c <L1>

实验要求4（20分）：

对于下面一段C语言程序，请动手编写对应的RV32I汇编程序sum.s，并进行汇编和反汇编。

完成情况：

[ashenye@meteor-04 step7]$ riscv32-unknown-elf-gcc mysum.s -c mysum.o

[ashenye@meteor-04 step7]$ riscv32-unknown-elf-objdump -d mysum.o > dump.txt

[ashenye@meteor-04 step7]$ cat dump.txt

mysum.o: file format elf32-littleriscv

Disassembly of section .text:

00000000 <sum>:

0: 000502b3 add t0,a0,zero

4: 00100313 li t1,1

8: 00000393 li t2,0

0000000c <.l1>:

c: 0062c863 blt t0,t1,1c <.l2>

10: 006383b3 add t2,t2,t1

14: 00130313 add t1,t1,1

18: ff5ff06f j c <.l1>

0000001c <.l2>:

1c: 00038533 add a0,t2,zero

20: 00008067 ret

00000024 <main>:

24: 00a00513 li a0,10

28: fd9ff0ef jal 0 <sum>

实验要求5（10分）：

[ashenye@meteor-04 step8]$ riscv32-unknown-elf-gcc sum.c -c sum.o

[ashenye@meteor-04 step8]$ riscv32-unknown-elf-objdump -d sum.o > dump.txt

[ashenye@meteor-04 step8]$ cat dump.txt

sum.o: file format elf32-littleriscv

Disassembly of section .text:

00000000 <sum>:

0: fd010113 add sp,sp,-48

4: 02812623 sw s0,44(sp)

8: 03010413 add s0,sp,48

c: fca42e23 sw a0,-36(s0)

10: fe042623 sw zero,-20(s0)

14: fe042423 sw zero,-24(s0)

18: 00100793 li a5,1

1c: fef42623 sw a5,-20(s0)

20: 0200006f j 40 <.L2>

00000024 <.L3>:

24: fe842703 lw a4,-24(s0)

28: fec42783 lw a5,-20(s0)

2c: 00f707b3 add a5,a4,a5

30: fef42423 sw a5,-24(s0)

34: fec42783 lw a5,-20(s0)

38: 00178793 add a5,a5,1

3c: fef42623 sw a5,-20(s0)

00000040 <.L2>:

40: fec42703 lw a4,-20(s0)

44: fdc42783 lw a5,-36(s0)

48: fce7dee3 bge a5,a4,24 <.L3>

4c: fe842783 lw a5,-24(s0)

50: 00078513 mv a0,a5

54: 02c12403 lw s0,44(sp)

58: 03010113 add sp,sp,48

5c: 00008067 ret

00000060 <main>:

60: fe010113 add sp,sp,-32

64: 00112e23 sw ra,28(sp)

68: 00812c23 sw s0,24(sp)

6c: 02010413 add s0,sp,32

70: 06400793 li a5,100

74: fef42623 sw a5,-20(s0)

78: fec42503 lw a0,-20(s0)

7c: 00000097 auipc ra,0x0

80: 000080e7 jalr ra # 7c <main+0x1c>

84: fea42423 sw a0,-24(s0)

88: 00000793 li a5,0

8c: 00078513 mv a0,a5

90: 01c12083 lw ra,28(sp)

94: 01812403 lw s0,24(sp)

98: 02010113 add sp,sp,32

9c: 00008067 ret

完成情况：

1. 在我的代码中，变量我都放在了寄存器上，而gcc变量放在了栈上, 因此gcc代码比我的代码多了很多sw和lw指令，而我直接操作寄存器，不需要sw和lw指令，因此我的代码比gcc代码少了很多指令而且效率更高。

2. gcc的代码中有栈的生长和收缩，我的代码中没有

3. gcc的是完整的函数调用，而我的调用函数只有jal和jr指令，没有保存寄存器的指令,不是一个标准的函数调用

**三、思考与探索**

1. 问题与解决方案（5分）

（整个实验过程中发生了什么问题？你是如何解决的。）

发现反编译出来的指令字长不为4字节,发现编译的选项开了指令压缩。解决：把指令压缩选项关闭解决。

1. 思考题（15分）

（力所能及，尝试实践或回答教材上的思考与探索题目，至少完成2道）

1. 多了a1寄存器入参指令

lui a5,%hi(.LC0)

addi a1,a5,%lo(.LC0)

lui a5,%hi(.LC1)

addi a0,a5,%lo(.LC1)

2.

main:

addi t0, x0, x0;

addi t1, x0, 0x40;

addi t3, x0, 0x72;

L1:

lw t2 , 0(t1);

add t0, t0, t2;

addi t1, t1, 4;

bne t1, t3, L1;

j L1;

L1:

sw t0, 0x80(x0) #好吧只少了一条指令

3.全字, sw而不是sh或者sb。a0-a2是传入的参数不改变传入的参数更语义化

4. 主要考虑到大数不能直接用一条指令实现，要多条指令配合

lui a0,0x10000;

lui a1,0x1000;

addi a1,a1,256;

addi a2, zero, 20;

jal BankMove;

5. 把bgt放在l1开头而不是l2前面