

Lab 3 report

PB21061281

刘卓

实验目的与内容

- 1.理解RISC-V常用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IDE:RARS，以及程序仿真运行环境和调试基本方法

- 设计汇编程序，实现自动测试前述18条的指令功能，并生成COE文件

-add, addi, sub, auipc, lui

-and, or, xor

-slli, srli, srai

-lw, sw

-beq, blt, bltu, jal, jalr

- 设计汇编程序，实现数组排序，并生成COE文件

-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

逻辑设计

part1

由于不能使用ecall输出错误信息，程序中使用寄存器t6作为信息记录寄存器，每次测试一条指令，如果错误则将编号记录在t6中，也就是说如果最终t6的值为0，说明全部正确，否则存储最后一个错误的指令

在测试文件中，首先利用x0测试beq，如果正常，跳转到后续测试代码，否则记录错误信息

对jal,jalr指令测试，不仅测试了是否正确跳转，也测试了ra寄存器是否更新为正确的值(PC+4)

注意指令的测试需要按照一定顺序，比如lui的实现实际用到了addi指令，那么需要先测试addi指令的正确性

需要注意的是，即使测试结果正确也不能保证正常工作，因为只测试了一组特定数据

另外实际上CPU不需要单独实现li指令，因为li指令实际上翻译为lui和addi组合或addi指令，这两条指令都是要实现的

part2

实现可变长数组排序，按照升序排序

运行程序前，地址0x2004存放数组大小，后面依次存放数组元素

使用选择排序，在原位排序，得到结果

```
.data
```

ADDR: .word 0x2004#起始地址，存放数组大小，后面存放数组数据，升序原位排序

```
.text
    lw t0, ADDR
    lw t0, 0(t0)#size
    lw t1, ADDR#pointer i
    lw t2, ADDR#pointer j
    li t3, 4
    mul t3, t3, t0
    add t3, t1, t3#end
    addi t1, t1, 4
    addi t2, t2, 4
LOOPI:
    bge t1, t3, END#i from 1 to n-1
    addi t2, t1, 4#j=i+1
LOOPJ:
    bgt t2, t3, ENDJ#j from i+1 to n
    lw t4, 0(t1)#a[i]
    lw t5, 0(t2)#a[j]
    bltu t4, t5, NEXT#Don't need to swap
    sw t5, 0(t1)#a'[i]=a[j]
    sw t4, 0(t2)#a'[j]=a[i]
NEXT:
    addi t2, t2, 4#j=j+1
    jal LOOPJ
ENDJ:
    addi t1, t1, 4
    jal LOOPI
END:
    li a7, 10
    ecall
```

测试结果与分析

part1

Assembly code and Data Segment view showing instructions and memory values.

Assembly Code:

Address	Instruction
0x00000000	beq x0,x0,0x0000001c
0x00000004	auipc x10,2
0x00000008	addi x10,x10,0xfffffff0
0x0000000c	addi x17,x0,4
0x00000010	ecall
0x00000014	addi x17,x0,10
0x00000018	ecall
0x0000001c	jal x1,0x0000001c
0x00000020	auipc x10,2
0x00000024	addi x10,x10,0xffffffe0
0x00000028	addi x17,x0,4
0x0000002c	ecall
0x00000030	addi x17,x0,10
0x00000034	ecall
0x00000038	addi x5,x0,0x00000020
0x0000003c	beq x1,x5,0x00000014

Data Segment:

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00002000	0x4f525245	0x49520052	0x00544847	0x00000020	0x000040a4	0x00008234	0xffff8234	0x00000000
0x00002020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000020a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000020c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000020e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000021a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Messages:

Run I/O

RIGHT

-- program is finished running (0) --

Clear

图1:测试代码的测试结果

如图输出正确信息，代表在测试代码的验证下功能正确

part2

Data Segment view showing memory values.

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00002000	0x00002004	0x0000000a	0x0000000a	0x00000009	0x00000008	0x00000007	0x00000006	0x00000005
0x00002020	0x00000004	0x00000003	0x00000002	0x00000001	0x00000000	0x00000000	0x00000000	0x00000000
0x00002040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000020a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000020c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000020e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000021a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

图2:排序的原始数据

如图，x2004处存放数组大小-0xa，后面依次降序存放数组元素

Data Segment view showing memory values after sorting.

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00002000	0x00002004	0x0000000a	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006
0x00002020	0x00000007	0x00000008	0x00000009	0x0000000a	0x0000000b	0x0000000c	0x0000000d	0x0000000e
0x00002040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000020a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000020c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000020e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00002180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x000021a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

图3:升序排序后结果

总结

这次实验主要考察了RISC-V指令集和汇编语言编程、调试，熟悉常用指令后做起来不难

- 出现bug:定义了标签ADDR的值为0x2000，一开始打算从0x2000存放第一个数据即数组大小，修改0x2000处内容为数组大小后运行，发现出错，报错信息为溢出。原因在于这两条语句

```
lw t0, ADDR  
lw t0, 0(t0)#size
```

这里我是打算将ADDR处的数据存下来(0x2000)，再从这个地址上读取到数组的大小(默认数组大小存放在0x2000)，然而内存中数据存放的起始位置就是0x2000，即ADDR代表的地址是0x2000，同时这里存放了起初定义的数据0x2000，也就是说默认m[0x2000]=0x2000，如果修改了0x2000处的值后执行，那么第一步lw t0, ADDR得到的t0不再是我们预想的数组起始地址0x2000，而是刚刚写入的数组大小0x0000a，后面再执行lw t0, 0(t0)，就访问了无法访问的内存空间