# 计算机组成原理(H) 实验

**PB20111699 吴骏东**

## 一、实验题目

**汇编程序设计**

## 二、实验目的

1. 了解汇编程序的基本结构，以及汇编程序仿真和调试的基本方法；
2. 熟悉RISC-V常用32位整数指令的功能，掌握简单汇编程序的设计，以及CPU下载测试方法和测试数据 (COE文件) 的生成方法。

## 三、实验平台

- **RARS** ——RISC-V Assembler & Runtime Simulator

## 四、实验过程

### 4.1 基本指令测试

**addi 指令与 add 指令**

```
.text
# This is the test program for instrution:
# [add]  [addi]

# First is the test for addi
addi t1, t1, 1      # t1 = t1 + 1 = x1
addi t1, t1, 15     # t1 = t1 + 15 = x10
addi t1, t1, -1     # t1 = t1 - 1 = xf
addi t1, t1, -16    # t1 = t1 - 16 = xffffffff

addi t2, t1, 2      # t2 = t1 + 2 = x1
addi t2, t2, -1     # t2 = t2 - 1 = 0
addi t1, t1, 1      # t1 = t1 + 1 = 0

# Now regt1 and t2 are zero.
#Then is the test for add
addi t1, t1, 5      # t1 = 5
add t2, t1, t1      # t2 = t1 + t1 = xa
add t1, t1, t2      # t1 = t1 + t2 = xf
```

```
addi t1, t1, -15    # t1 = 0
add t1, t1, t2      # t1 = t2 = xa

# Now regt1 and regt2 are xa
```

### sub 指令与 auipc 指令

```
.text
# This program will test instruction:
# [sub], [auipc]

# Frist is the test of sub
addi t1, t1, 255    # t1 = xff
addi t2, t2, 128    # t2 = x80
sub t2, t1, t2      # t2 = t1 - t2 = x7f
sub t2, t1, t2      # t2 = t1 - t2 = x80

# Then is the test for auipc
auipc t1, 1 # t1 = pc + x1000
auipc t1, 2 # t1 = pc + x2000
auipc t1, -1    # t1 = pc - x1000
```

### lw 指令与 sw 指令

```
# This is the test for lw and sw
# [lw],  [sw]

.data
1
2
3
-1
0
.text
addi t1, x0, 0
lw t2, 0(t1)
addi t2, t2, 1
sw t2, 0(t1)

addi t1, t1, 4
lw t2, 0(t1)
addi t2, t2, 1
sw t2, 0(t1)

addi t1, t1, 4
lw t2, 0(t1)
addi t2, t2, 1
sw t2, 0(t1)

addi t1, t1, 4
lw t2, 0(t1)
addi t2, t2, 1
sw t2, 0(t1)
```

```
addi t1, t1, 4
lw t2, 0(t1)
addi t2, t2, 1
sw t2, 0(t1)
```

**blt 指令与 beq 指令**

```
# This is the test for beq and blt
# [blt] , [beq]
.data
-9
.text
addi t0, x0, 0
lw t1, 0(t0)

blt x0, t1, Postive
beq x0, t1, Zero
Negtive:
addi s0, x0, 1
beq x0, x0, done
Zero:
addi s1, x0, 1
beq x0, x0, done
Postive:
addi s2, x0, 1
beq x0, x0, done
done:
```

**jal 指令与jalr 指令**

```
# This is the test fot jal and jalr
# [jal], [jalr]
.text
addi t0, x0, 1
jal s1, Here1
Here2:
jalr s3, 4(s1)
Here1:
jal s2, Here2
# Endless loop!
```

## 4.2 冒泡排序设计

### 4.2.1 排序核心单元

由于本实验要求的是无符号数排序，所以不能直接使用 blt 指令进行判断。为此本实验参考了 lab1 中 ALU 无符号数大小比较的方式进行了分类讨论。过程如下:

```
lw s2, 0(s1)        # load mem[s1]
lw s3, 4(s1)        # load mem[s2]
```

```
blt s2, x0, minus   # mem[s1] < 0
# mem[s1] >= 0
blt s3, x0, noswap  # mem[s2] < 0 ，无符号数 mem[s1] 严格小于无符号数 mem[s2]
# mem[s2] >= 0
# 此时两数均非负，可以正常比较大小
j compare
minus:
blt x0, s3, doswap  # mem[s2] >= 0 ，无符号数 mem[s1] 严格大于无符号数 mem[s2]
# now mem[s2] <= 0
beq s3, x0, doswap  # 此时 mem[s1] < 0, mem[s2] = 0， 应该交换
# 此时两数均为负，可以正常比较大小
compare:
blt s2, s3, noswap  # if mem[s1] < mem[s2] do not swap
doswap:
sw s2, 4(s1)
sw s3, 0(s1)
noswap:
# swap end
```

随后为排序子程序补充寄存器保存与恢复功能：

```
# 程序调用
jal x1, Swap
# ......

Swap:   # will check the number in mem[s1] and mem[s1+4]
# if mem[s1] > mem[s2] then swap them
# reg s2 contains mem[s1]
# reg s3 contains mem[s1+4]
addi sp, sp, -8     #change the stack pointer
sw s2, 4(sp)
sw s3, 0(sp)

# ......

lw s3, 0(sp)
lw s2, 4(sp)
addi sp, sp, 8

# 程序返回
jalr x0, 0(x1)
```

### 4.2.2 双重循环

冒泡排序的双重循环实现如下：

```
addi t0, x0, 0          # i = 0
addi s11, s11, -1       # s11 = n - 1
    For1:
    beq t0, s11, Done       # if (i == n - 1) done
    addi t1, x0, 0          # j = 0
    sub s2, s11, t0         # s2 = n - i - 1
        For2:
        beq t1, s2, For1done    # if (j == n - i) break;
        addi s1, t1, 0          # s1 = j
        add s1, s1, s1
        add s1, s1, s1          # s1 = s1 * 4  修正地址

        jal x1, Swap

        addi t1, t1, 1          # j++
        j For2

    For1done:
    addi t0, t0, 1          # i++
    j For1

Done:
#......
```

## 4.3 仿真输出设计

RARS 提供了专用的输入输出仿真工具 MMIO：Memory-Mapped Input and Output。例如，输出显示 (Display) 外设的相关地址为：

- 0x7f08: 状态地址
- 0x7f0c: 数据地址

程序通过轮询状态地址的数值进行判断，通过向数据地址中写入 ASCII 码进行模拟输出。

**16进制 - 8421BCD 转换**

事实上这是程序中最为恼人的部分。如何在不使用移位指令与逻辑运算的前提下将 32-bit 二进制数转化成 8位数码。本实验中采取了从高到低每四位进行转换的方式。具体设计如下：

```
numberstart:
add t1, x0, x0      # t1 存储前四位转换结果
# t0 存储目标32bit数据

place1:
lw s0, 4(t3)    # s0 = x8000_0000
addi s1, x0, 8
sub s2, t0, s0
blt s2, x0, place2
add t1, t1, s1
sub t0, t0, s0

place2:
lw s0, 8(t3)    #s0 = x4000_0000
addi s1, x0, 4
sub s2, t0, s0
blt s2, x0, place3
add t1, t1, s1
sub t0, t0, s0

place3:
lw s0, 12(t3)  #s0 = x2000_0000
addi s1, x0, 2
sub s2, t0, s0
blt s2, x0, place4
add t1, t1, s1
sub t0, t0, s0
```

```
place4:
lw s0, 16(t3)  #s0 = x1000_0000
addi s1, x0, 1
sub s2, t0, s0
blt s2, x0, place5
add t1, t1, s1
sub t0, t0, s0
place5:        # nothing

add t0, t0, t0
add t0, t0, t0
add t0, t0, t0
add t0, t0, t0
# 将32bit数字左移四位
# 此时 s1 中存储着前四位对应的数字

# output
wait1:
lw t4, 0(t5)
beq t4, x0, wait1

addi t6, x0, 9
blt t6, t1, notnumber
# 为 0~9
addi t1, t1, 48
sw t1, 4(t5)
addi s0, x0, 7
beq s9, s0, numberdone

addi s9, s9, 1  # 计数器，每输出8个数字进行换行
j numberstart

notnumber:
# 为 A~F
addi t1, t1, 55
sw t1, 4(t5)
addi s0, x0, 7
beq s9, s0, numberdone

addi s9, s9, 1  # 计数器，每输出8个数字进行换行
j numberstart
```

## 五、实验结果

设置排序的数字数目为 16。在 .data 部分中写入：

```
.data
-1
-1
-1
0
0
0
1
```

```
1
1
114
113
112
111
-110
109
108
```

排序后仿真输出结果为：

```
00000000
00000000
00000000
00000001
00000001
00000001
0000006C
0000006D
0000006F
00000070
00000071
00000072
FFFFFF92
FFFFFFFF
FFFFFFFF
FFFFFFFF
```

针对256个数的排序由于耗时较长，此处没有列出实现结果。

## 六、心得体会

这次实验难度适中，算是为了后续单周期设计做好了铺垫。