RISC-V 冒泡排序

李远航
PB20000137

【实验内容】:

基于 RV32 汇编，设计冒泡排序程序，并用 Ripes 工具调试执行。

【实验要求】:

1. 需要实现 Ripes 控制台 Console 输出
2. 待排序数据可以直接定义在数据段
3. 不少于 10 个正整数
4. 前两个数据为学号前两位和后两位。如 JL/PB12345678，前两个数据需为 12，78
5. 结果降序排列

【实现思路】:

1. 数据存储：
   直接使用 .word 字段，在内存中存储数组
2. 冒泡排序：
   使用两重循环，以此交换数组元素每次将最小的元素移动到数组末尾，算法的复杂度为 O(N^2)
3. 核心汇编部分思路

```
# 调用部分
andi t0, a0, 0
addi t2, a1, 0
jal sort
```

```
sort:
    andi t1, a0, 0
    addi t1, t1, 1
    sort_in:
        mul a0, t1, a3
        add a0, a2, a0
        lw a4, 0(a0)
        addi t1, t1, -1
        mul a0, t1, a3
        add a0, a2, a0
        lw a5, 0(a0)
        bge a5 ,a4, judge
        sw a4, 0(a0)
        addi t1, t1, 1
        mul a0, t1, a3
        add a0, a2, a0
        sw a5, 0(a0)
        addi t1, t1, -1
        judge:
        addi t1, t1, 2
        blt t1, t2, sort_in
    addi t0, t0, 1
    blt t0, t2, sort
    jr x1
```

- t2 寄存器存储数组的长度，使用 t0,t1 寄存器来标记循环的次数，t0 标记第几次遍历数组，t1 表示遍历时，遍历到的数组元素下标。

- 每次进入循环，先根据 t1 计算出偏移量，用 lw 指令取得连续两个数组元素的值，使用 bge 指令判断是否需要交换，如果需要交换，则使用 sw 指令对数据进行交换，将 t1 的值+1(由于内部是往前取一个数进行交换，所以汇编代码部分需要+2)

- 使用 blt 指令判断本次对数组的遍历是否结束，如果结束，则将 t0 的值+1，否则，继续循环遍历交换数组元素

- 使用 blt 指令判断是否完成遍历数组，如果尚未完成，则继续循环，否则，使用 jr 指令，回到 sort 调用的位置

具体思路还可以参照如下所示的 C++代码：

```cpp
#include <bits/stdc++.h>
int main()
{
    int a[10] = {20, 37, 1, 73, 2, 12, 43, 86, 57, 77};
    for (int i = 0; i < 10; i++)
        for (int j = 1; j < 10; j++)
        {
            if (a[j] > a[j - 1])
            {
                int temp = a[j];
                a[j] = a[j - 1];
                a[j - 1] = temp;
            }
        }
    for (int i = 0; i < 10; i++)
        std::cout << a[i] << " ";
    putchar('\n');
    return 0;
}
```

【实验过程】:

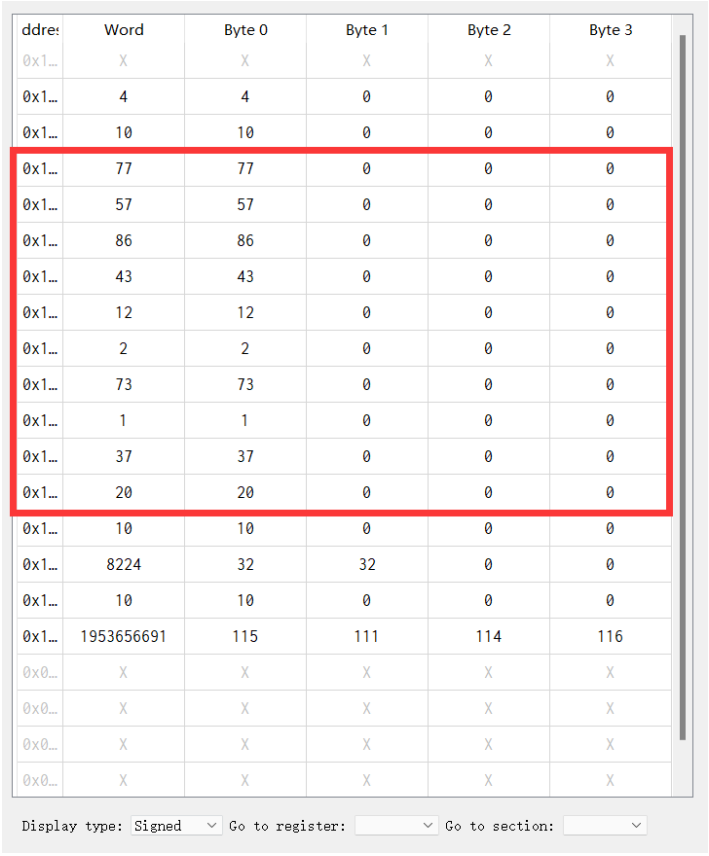
1. 程序运行时，console 控制台输出：

```
Console

sort
20  37  1  73  2  12  43  86  57  77
86  77  73  57  43  37  20  12  2  1
Program exited with code: 0
```

2. 内存数据段

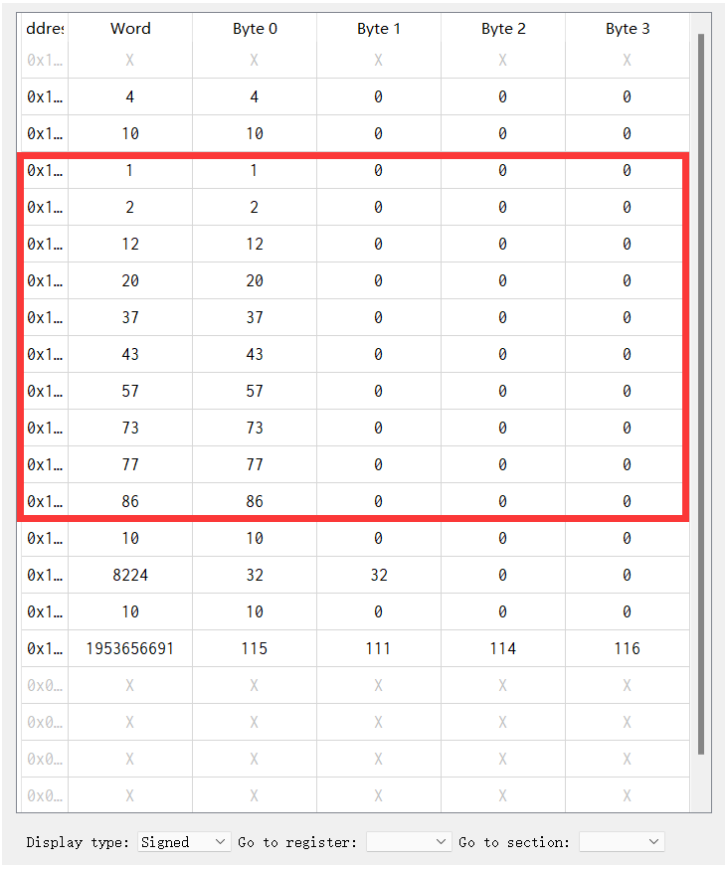
a) 运行程序前内存数据段



ddres	Word	Byte 0	Byte 1	Byte 2	Byte 3
0x1…	X	X	X	X	X
0x1…	4	4	0	0	0
0x1…	10	10	0	0	0
0x1…	77	77	0	0	0
0x1…	57	57	0	0	0
0x1…	86	86	0	0	0
0x1…	43	43	0	0	0
0x1…	12	12	0	0	0
0x1…	2	2	0	0	0
0x1…	73	73	0	0	0
0x1…	1	1	0	0	0
0x1…	37	37	0	0	0
0x1…	20	20	0	0	0
0x1…	10	10	0	0	0
0x1…	8224	32	32	0	0
0x1…	10	10	0	0	0
0x1…	1953656691	115	111	114	116
0x0…	X	X	X	X	X
0x0…	X	X	X	X	X
0x0…	X	X	X	X	X
0x0…	X	X	X	X	X

Display type: Signed Go to register: Go to section:

b) 冒泡程序运行后内存数据段



ddres	Word	Byte 0	Byte 1	Byte 2	Byte 3
0x1…	X	X	X	X	X
0x1…	4	4	0	0	0
0x1…	10	10	0	0	0
0x1…	1	1	0	0	0
0x1…	2	2	0	0	0
0x1…	12	12	0	0	0
0x1…	20	20	0	0	0
0x1…	37	37	0	0	0
0x1…	43	43	0	0	0
0x1…	57	57	0	0	0
0x1…	73	73	0	0	0
0x1…	77	77	0	0	0
0x1…	86	86	0	0	0
0x1…	10	10	0	0	0
0x1…	8224	32	32	0	0
0x1…	10	10	0	0	0
0x1…	1953656691	115	111	114	116
0x0…	X	X	X	X	X
0x0…	X	X	X	X	X
0x0…	X	X	X	X	X
0x0…	X	X	X	X	X

Display type: Signed Go to register: Go to section:

3. 程序执行的时间
   选择 50ms 作为时钟周期，完成程序的运行需要 1410 个时钟周期，因此，需要 70.5s
4. 实验中遇到的问题
   a) 数组的存储：
      可以直接使用.word 字段写在内存里
   b) 数组元素的访问
      使用基址偏移寻址的方式，对数组元素进行访问和存储

【实验收获】:

1. 熟悉了 RV32 汇编，掌握了 Ripes 工具基本的使用
2. 对内存有了进一步的认识

【具体代码】:

```
.data
str:        .string     "sort\n"
delimiter:  .string     "  "
newline:    .string     "\n"
array:      .word       20, 37, 1, 73, 2, 12, 43, 86, 57, 77
length:     .word       10
size:       .word       4


.text
main:
    la a0, str
    li a7, 4
    ecall

    jal init
    jal loopPrint

    jal init

    li a7, 4
    la a0, newline
    ecall

    # 调用部分
    andi t0, a0, 0
    addi t2, a1, 0
    jal sort

    jal init
```

```
    jal loopPrint
    li a7, 10
    ecall

init:
    la a1, length
    lw a1, 0(a1)
    la a2, array
    la a3, size
    lw a3, 0(a3)
    jr x1

loopPrint:
    andi t0, a0, 0
    addi t1, a1, 0
    loop:
        mul t2, a3, t0
        add t2, t2, a2
        lw a0, 0(t2)
        li a7, 1
        ecall
        # Print a delimiter between the numbers
        li a7, 4
        la a0, delimiter
        ecall
        # Increment
        addi t0, t0, 1
        blt  t0, t1, loop
    jr x1

sort:
    andi t1, a0, 0
    addi t1, t1, 1
    sort_in:
        mul a0, t1, a3
        add a0, a2, a0
        lw a4, 0(a0)
        addi t1, t1, -1
        mul a0, t1, a3
        add a0, a2, a0
        lw a5, 0(a0)
        bge a5 ,a4, judge
        sw a4, 0(a0)
        addi t1, t1, 1
```

```
        mul a0, t1, a3
        add a0, a2, a0
        sw a5, 0(a0)
        addi t1, t1, -1
        judge:
        addi t1, t1, 2
        blt t1, t2, sort_in

    addi t0, t0, 1
    blt t0, t2, sort
    jr x1
```