计算机组成原理·实验报告一

PB20111686　黄瑞轩

实验目标

1. 基于RISC-V汇编，设计一个冒泡排序程序，并用Debug工具调试执行。（Ripes仿真）
2. 测量冒泡排序程序的执行时间。

实验思路

冒泡排序是基于交换的排序算法，极端情况下，完全逆序的数据需要经过$O(n^2)$次交换。这里采用带标记的冒泡排序算法，一旦一轮检测中没有进行交换操作，说明排序已经结束。排序结束后，内存的低地址存较大的数据（有符号数）。

C风格伪代码思路如下：

```
#define ARR_LEN 10
bool re = 1;
while (re) {
    re = 0;
    for (int j = 0; j < ARR_LEN - 1; j++) {
        if (ARR[j] < ARR[j + 1]) {
            swap(ARR, j);
            re = 1;
        }
    }
}
```

根据C风格伪代码即可写出RISC-V程序。

实验代码

因为一开始内存里没有值，首先要向内存里写一些值，再执行排序。

为了显示排序的正确性，这里取开始地址为`0x100`，结束地址为`0x134`，初始数组内容为`0x800003a2`，增量为`-0xdd`，这样中间就有正数和负数的跨越，更能显示排序正确性。

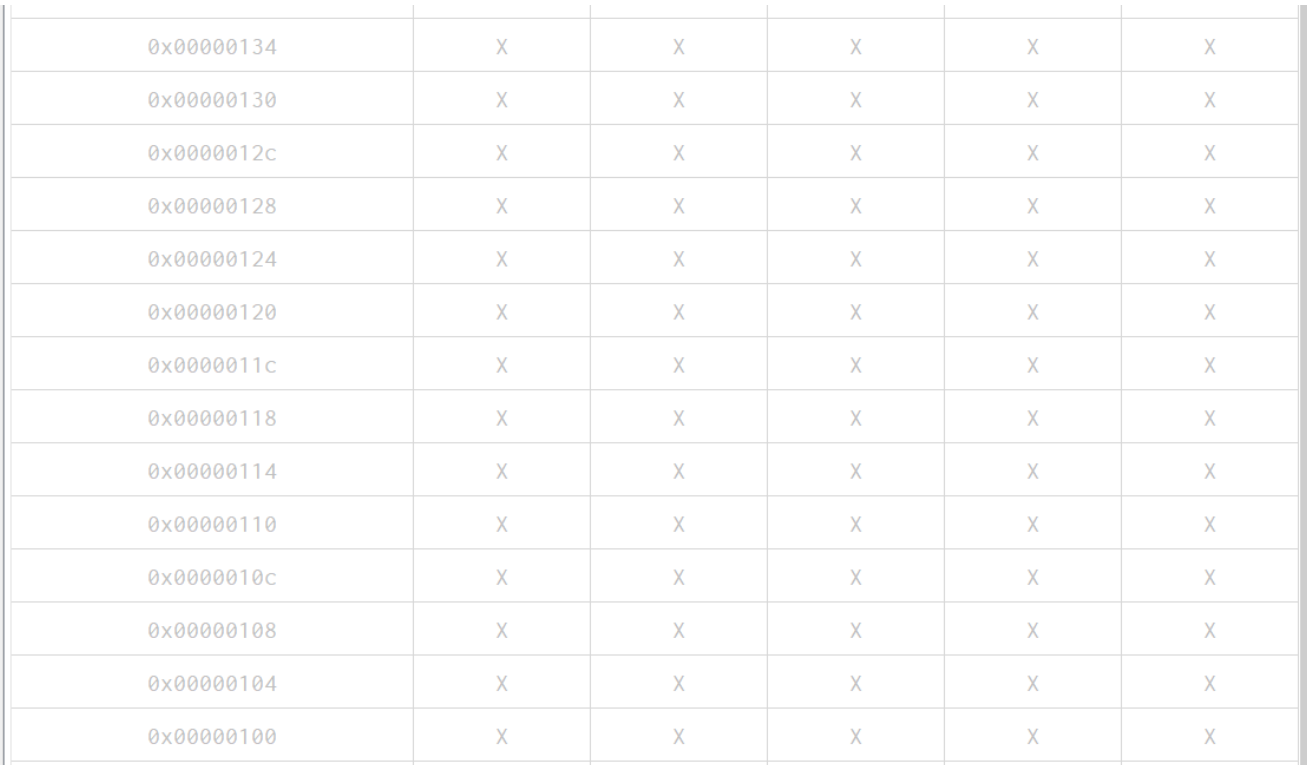
```
# A program to implement sort by Sprout.
# Using bubble sort.

.data
```

```
base: .word 0x100 # Base address of array
end: .word 0x134 # End of array
one: .word 0x1
AR0: .word 0x800003a2

.text
main:
    and a0, a0, zero      # relative offset
    lw a1, end            # load end address
    lw a2, base           # load array
    lw a4, AR0            # load init content of array
    lw s6, one            # load 1
saveLoop:
    add a3, a0, a2        # a3 is the next address to save content
    sw a4, 0(a3)          # save a4 to mem[a3]
    addi a0, a0, 4        # offset increment
    addi a4, a4, -0xdd    # set new content
    blt a3, a1, saveLoop  # if not end, branch to loop

    lw s2, one            # save ended, load re as one
    addi a1, a1, -4       # ignore the last element
whileRe:
    beq s2, zero, halt
    and s2, s2, zero      # set re as zero
    and a0, a0, zero      # set offset as zero
innerloop:
    add a3, a0, a2
    lw a6, 0(a3)
    lw a7, 4(a3)
    bge a6, a7, endloop

    mv s3, a7             # swap
    mv a7, a6
    mv a6, s3
    sw a6, 0(a3)
    sw a7, 4(a3)
    lw s2, one
endloop:
    addi a0, a0, 4
    blt a3, a1, innerloop
    beq a3, a1, whileRe
halt:
```

实验结果

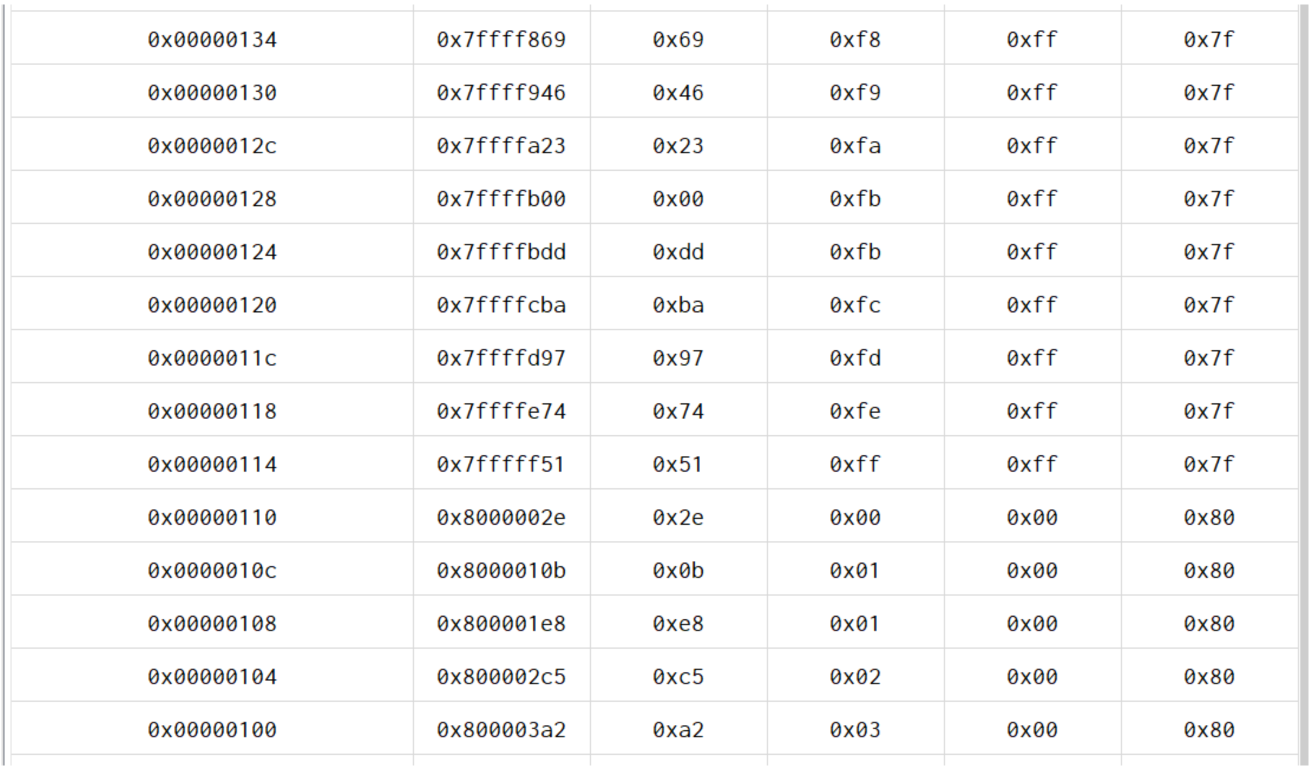
利用Ripes工具进行仿真调试，因为数据都存在内存里，所以我们关心内存的变化。

设置内容前：



0x00000134	X	X	X	X	X
0x00000130	X	X	X	X	X
0x0000012c	X	X	X	X	X
0x00000128	X	X	X	X	X
0x00000124	X	X	X	X	X
0x00000120	X	X	X	X	X
0x0000011c	X	X	X	X	X
0x00000118	X	X	X	X	X
0x00000114	X	X	X	X	X
0x00000110	X	X	X	X	X
0x0000010c	X	X	X	X	X
0x00000108	X	X	X	X	X
0x00000104	X	X	X	X	X
0x00000100	X	X	X	X	X

设置内容后：



0x00000134	0x7ffff869	0x69	0xf8	0xff	0x7f
0x00000130	0x7ffff946	0x46	0xf9	0xff	0x7f
0x0000012c	0x7ffffa23	0x23	0xfa	0xff	0x7f
0x00000128	0x7ffffb00	0x00	0xfb	0xff	0x7f
0x00000124	0x7ffffbdd	0xdd	0xfb	0xff	0x7f
0x00000120	0x7ffffcba	0xba	0xfc	0xff	0x7f
0x0000011c	0x7ffffd97	0x97	0xfd	0xff	0x7f
0x00000118	0x7ffffe74	0x74	0xfe	0xff	0x7f
0x00000114	0x7fffff51	0x51	0xff	0xff	0x7f
0x00000110	0x8000002e	0x2e	0x00	0x00	0x80
0x0000010c	0x8000010b	0x0b	0x01	0x00	0x80
0x00000108	0x800001e8	0xe8	0x01	0x00	0x80
0x00000104	0x800002c5	0xc5	0x02	0x00	0x80
0x00000100	0x800003a2	0xa2	0x03	0x00	0x80

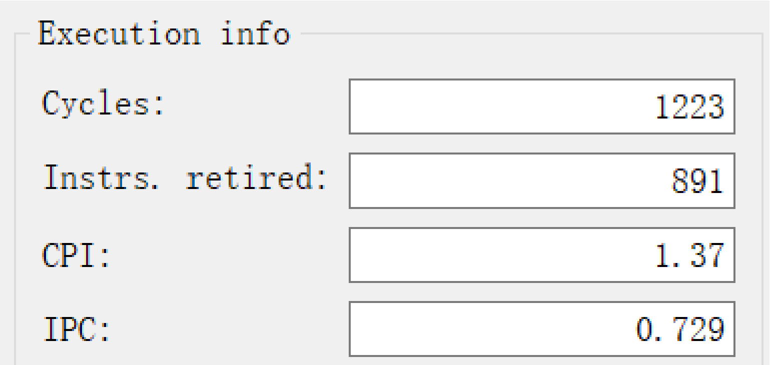
排序后：

0x00000134	0x8000002e	0x2e	0x00	0x00	0x80
0x00000130	0x8000010b	0x0b	0x01	0x00	0x80
0x0000012c	0x800001e8	0xe8	0x01	0x00	0x80
0x00000128	0x800002c5	0xc5	0x02	0x00	0x80
0x00000124	0x800003a2	0xa2	0x03	0x00	0x80
0x00000120	0x7ffff869	0x69	0xf8	0xff	0x7f
0x0000011c	0x7ffff946	0x46	0xf9	0xff	0x7f
0x00000118	0x7ffffa23	0x23	0xfa	0xff	0x7f
0x00000114	0x7ffffb00	0x00	0xfb	0xff	0x7f
0x00000110	0x7ffffbdd	0xdd	0xfb	0xff	0x7f
0x0000010c	0x7ffffcba	0xba	0xfc	0xff	0x7f
0x00000108	0x7ffffd97	0x97	0xfd	0xff	0x7f
0x00000104	0x7ffffe74	0x74	0xfe	0xff	0x7f
0x00000100	0x7fffff51	0x51	0xff	0xff	0x7f

实验结果符合预期。

程序用时

根据仿真程序记录，整个程序的执行情况如下：



Execution info

Cycles: 1223

Instrs. retired: 891

CPI: 1.37

IPC: 0.729

程序的`Auto clock interval`设置为1 ms，则理论消耗时间为

$$1223 \times 1\text{ms} = 1.223\text{s}$$