The experiment report for Lab 1

PB 18000227 艾语晨

The original problem:

> 1. 基于MIPS汇编，设计一个冒泡排序程序，并用Debug工具调试执行。
> 2. 测量冒泡排序程序的执行时间。
> 3. 提交实验报告，并在报告后附上源码。
> 4. 不强制提交，但写了会有额外加分，具体加分待定。

The original code

> Also provided as an independent .asm file in my GitHub repo

```
li $v0, 5                   # service 5 is read integer
syscall
add $s0, $v0, $zero         # load return value into register $s0: total
number

slti $t0, $v0, 1            # if the number of data < 1, terminate the
program
beq $t0, 1, Exit

add $t0, $zero, $zero       # the loop varible
add $s1, $sp, $zero         # store the initial address of the $sp, so
that we can get the boundary then

Init:   addi $t0, $t0, 1
    li $v0, 5
    syscall
    sw $v0, 0($sp)
    addi $sp, $sp, -4       # the stack moves to one lower store

    bne $t0, $s0, Init      # if the loop varible is not equal with the
total number, continue the loop

beq $s0, 1, Print               # if there's only one number, just print
it

addi $s2, $sp, 4                # the last statistic of the stack, stored
for promoting speed
addi $s3, $s0, -1               # number 'n-1'
addi $s4, $s1, 4                # aimed for a faster print
addi $sp, $sp, 4                # to point to the last statistic again

```

```
# start sorting (use $sp to point to the comparing bubble)
add $t0, $zero, $zero               # loop varible
li $v0, 30                  # 30 shows the system time
syscall
add $s7, $a0, $zero             # lower return value is in $a0

Cycle_i:addi $t0, $t0, 1            # loop varible increment
    Cycle_j:lw $t1, 0($sp)
        lw $t2, 4($sp)          # load one statistic and the one in higher
addr
        slt $t3, $t1, $t2       # if $t1 < $t2
        beq $t3, 1, NOSWAP      # put the larger one on the top
            # SWAP
            sw $t1, 4($sp)
            sw $t2, 0($sp)
        NOSWAP: addi $sp, $sp, 4    # move to the next level
        bne $sp, $s1, Cycle_j
    add $sp, $s2, $zero         # reinit the $sp
    addi $s1, $s1, -4           # update the surface (the largest is
already there!)
    bne $t0, $s3, Cycle_i

li $v0, 30                  # 30 shows the system time
syscall
sub $a0, $a0, $s7
li $v0, 1
syscall                     # print total time

li $v0, 11                  # print a character
addi $a0, $zero, 10             # 10 is '\n'
syscall

add $sp, $s2, $zero
add $t0, $zero, $zero               # loop varible
Print:  add $t0, $t0, 1
    li, $v0, 1              # print integer
    lw $a0, 0($sp)
    syscall
    # build table
    li, $v0, 11             # print a character
    addi $a0, $zero, 9              # 9 is '\t'
    syscall
    # loop ctrl
    addi $sp, $sp, 4            # move to the upper level
    bne $sp, $s4, Print

Exit:   li, $v0, 10             # 10 is exit
    syscall
```

Analyse these code

> Actually, the annotations *are* the analysis

结构

具体实现分以下几个步骤，我将结合 C++语言*伪代码*来解释：

> Cpp 程序中，使用一个堆分配的数组来代替stack，由于在MIPS中已经提供了 $sp，即栈指针
>
> 几个针对数据个数的特殊处理是为了应对输入的特殊情况，方法是对总个数进行判断跳转，`beq`，`bne` 指令
>
> 计时使用的方法为两次调用 syscall，分别记录系统时间，相减即得

Step one : 输入原始数据，对应汇编代码的前17行

cpp source code

```cpp
int NUM;
cin >> NUM;
int *array = new int [NUM];
int t0 = 0;                                 // count
for (; t0 < NUM; t0++)
{
    cin >> array[t0];
}
```

Step two : 冒泡排序，即通过逐个比较相邻的两个数据，把最大的放到栈顶

cpp source code

```cpp
for (int i = 0; i < NUM - 1; ++i)
{
    for (int j = NUM - 1; j >= i; --j)
    {
        if (array[j] > array[j - 1])
        {
            swap(array[j], array[j - 1]);
        }
        // else
            // pass
    }
}
```

Step three : 输出时间和排序结果，依然是利用栈指针来遍历

cpp source code

```cpp
cout << time2 - time1;
cout << "\t";
for (int t0 = NUM - 1; t0 > -1; --t0)
{
    cout << array[t0];
    cout << "\t";
}
```

Step four : 结束程序运行

cpp source code

```cpp
return 0;
```

一些细节

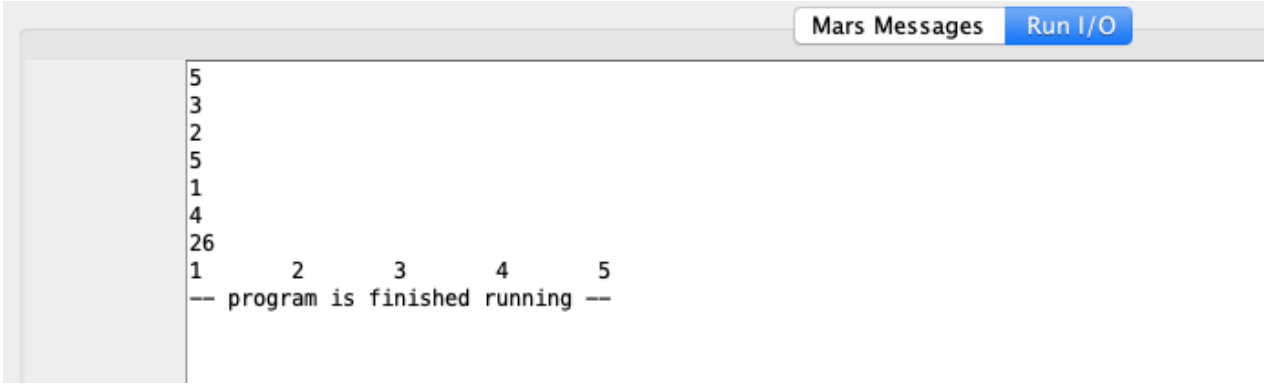
1. 寄存器的使用
    - 在基本的存储使用（系统调用使用、函数调用存储、循环变量, etc）以外，还使用了`$s1`, `$s2`, `$s3`, `$s4` 来存储一些量/信息，以优化数据采集速度/提供必需的边界条件（具体寄存器的含义请移步注释）
    - 减少一些不必要的寄存器开销：如在一开始写的程序里面，两次记录时间存储到 保留寄存器 `$s7`, `$s6` 中，实际上第二次不需要复制一遍，直接用返回值寄存器调用 `sub` 指令即可
2. 栈指针的使用
    - 一种方式是每一次都清零，从栈顶开始循环，不过如果知道栈指针的当前位置，逆向循环可以节约指令
    - 栈指针（或者保留下来的地址）也可以作为循环的边界条件，避免越界

要点

1. 在操纵内存/寄存器的数据都时候，最好在纸上画出来堆栈，标出来用到的寄存器中数据的含义
2. 冒泡排序的理论实现~~（这个不需要多说了吧）~~是：
    1. 外层循环控制水面（就是每一次比较的终点），每循环一次找到剩余数据里面最大的那个，冒到水面（栈顶）
    2. 内层循环是比较的两个数据（j 和 j-1）每一次比较都是把大的挑出来
3. 在汇编实现排序的时候，一定一定要注意当前栈指针的位置，比如在顺着输入完数据之后栈指针需要回调一格（`addi $sp, $sp, 4`），而且切记栈是**从高向低**

> 以上就是我在做这次实验时的思考点

实验结果



> 结果解释：

第一行的 5 是表示总共 5 个数

后面依次乱序输入1～5

26 是运行总时间（26 ms）

后面是打表输出排序结果（由于是栈底到栈顶，故为从小到大）