计算机组成原理MIPS实验

实验题目: MIPSS冒泡排序

学生姓名：王章瀚

学生学号：PB18111697

完成日期：2020 年 3 月 28 日

计算机实验教学中心制

2019年09月

1 题目要求

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。

2 代码结构分析

2.1 函数说明

2.1.1 main

用以调用随机数组生成函数和冒泡排序函数，并在此计时，最终输出时间。

2.1.2 array_generate

传入参数为:

- $a0: 数组起始地址
- $a1: 数组大小

最后会在数组起始地址开始，生成数组大小的个数的随机数，这里随机数小于等于10000。

2.1.3 bubble_sort

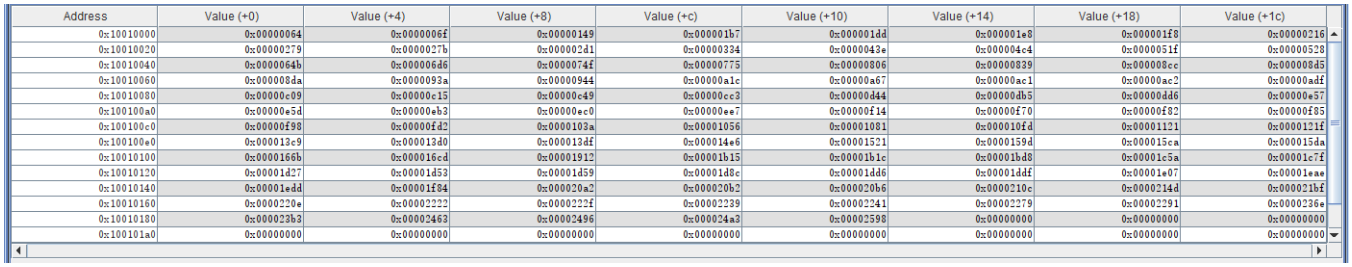
传入参数为:

- $a0: 数组起始地址
- $a1: 数组大小

这个函数实现了对数组起始地址开始数组大小个数的元素的冒泡排序（此处是从小到大）。冒泡排序算法比较出名，通过两个循环，每次把最大的数移到最末位，这里就不再赘述其原理了。

3 代码运行结果

这里放一张代码运行结果的截图，以证明确实完成了排序。



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000064	0x0000006f	0x00000149	0x000001b7	0x000001dd	0x000001e8	0x000001f8	0x00000216
0x10010020	0x00000279	0x0000027b	0x000002d1	0x00000334	0x0000043e	0x000004c4	0x0000051f	0x00000528
0x10010040	0x0000064b	0x000006d6	0x0000074f	0x00000775	0x00000806	0x00000839	0x000008cc	0x000008d5
0x10010060	0x000008da	0x0000093a	0x00000944	0x00000a1c	0x00000a67	0x00000ac1	0x00000ac2	0x00000adf
0x10010080	0x00000c09	0x00000c15	0x00000c49	0x00000cc3	0x00000d44	0x00000db5	0x00000dd6	0x00000e57
0x100100a0	0x00000e5d	0x00000eb3	0x00000ec0	0x00000ee7	0x00000f14	0x00000f70	0x00000f82	0x00000f85
0x100100c0	0x00000f98	0x00000fd2	0x0000103a	0x00001056	0x00001081	0x000010fd	0x00001121	0x0000121f
0x100100e0	0x000013c9	0x000013d0	0x000013df	0x000014e6	0x00001521	0x0000159d	0x000015ca	0x000015da
0x10010100	0x0000166b	0x000016cd	0x00001912	0x00001b15	0x00001b1c	0x00001bd8	0x00001c5a	0x00001c7f
0x10010120	0x00001d27	0x00001d53	0x00001d59	0x00001d8c	0x00001dd6	0x00001ddf	0x00001e07	0x00001eae
0x10010140	0x00001edd	0x00001f84	0x000020a2	0x000020b2	0x000020b6	0x0000210c	0x0000214d	0x000021bf
0x10010160	0x0000220e	0x00002222	0x0000222f	0x00002239	0x00002241	0x00002279	0x00002291	0x0000236e
0x10010180	0x000023b3	0x00002463	0x00002496	0x000024a3	0x00002598	0x00000000	0x00000000	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

可以看到从0x10010004开始的100个数都按从小到大的顺序排好了。

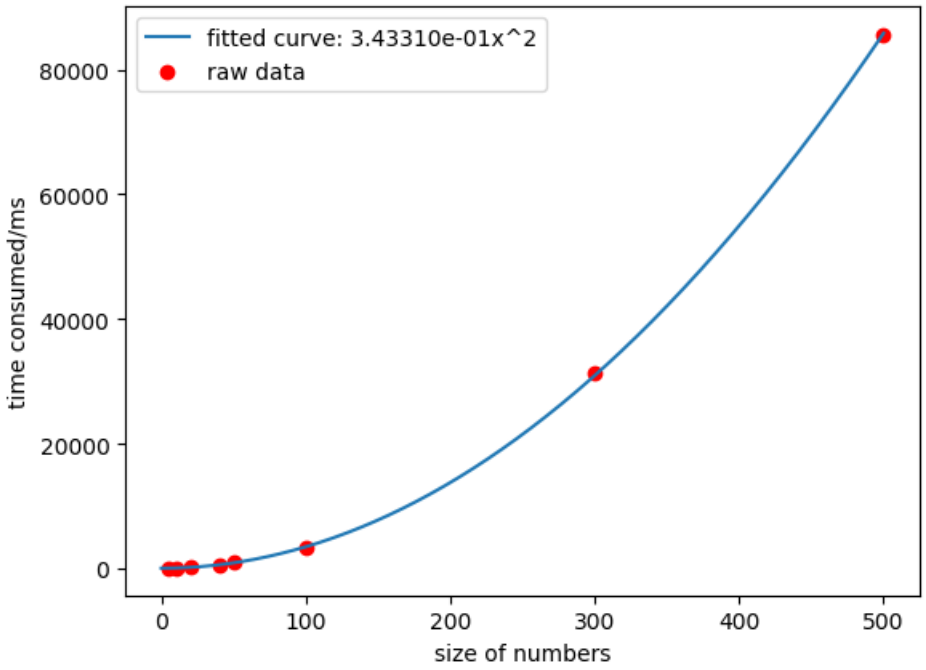
4 代码时间测量

4.1 测量结果

时间测量部分以数组大小为变量测量了多组数据，如下表所示：

# \ size	10	50	100	300	500
1	37	879	3389	31878	85790
2	36	885	3362	30819	85483
3	35	884	3402	31596	85639
average	36	882.6667	3384.333	31431	85637.33

然后用Python程序对数据进行了二次函数拟合，结果如下图



4.2 结果说明

可以看到，冒泡排序的时间随数组大小成二次函数的关系，这是符合冒泡排序的$o(n^2)$的时间复杂度的。

5 总结

本次实验用MIPS汇编实现了冒泡排序，并验证了冒泡排序$o(n^2)$的时间复杂度。

6 附代码

```
1
2       .text
3       .globl main
4   main:
5       # Generate the random numbers
6       la   $a0, array
```

```
    lw  $a1, N
    jal array_generate
    # Start time
    li  $v0, 30
    syscall
    move    $s4, $a0
    move    $s5, $a1
    # Sort
    la  $a0, array
    lw  $a1, N
    jal bubble_sort
    # End time
    li  $v0, 30
    syscall
    move    $s6, $a0
    move    $s7, $a1
    sub $s6, $s6, $s4
    sub $s7, $s7, $s5
    # Print time consumed(only considering about the lower bits of time)
    li  $v0, 1
    move    $a0, $s6
    syscall
    # Halt
    li  $v0, 10
    syscall


# Bubble Sort from small to large
bubble_sort:
    # Register assignments
    # $s1 = &array[i], i is from N to 1
    # $s2 = &array[j], j is from 1 to i-1
    # $s3 = &array[0]
    # $t0, temp for swap
    # $t1 = array[j-1]
    # $t2 = array[j]

    # Save registers
    addi $sp, $sp, -4       # Adjust stack pointer
    sw $s1, 0($sp)          # Save $s1
    addi $sp, $sp, -4       # Adjust stack pointer
    sw $s2, 0($sp)          # Save $s2
    addi $sp, $sp, -4       # Adjust stack pointer
    sw $s3, 0($sp)          # Save $s3

    # Start the sort loops
    move    $s1, $a1        # get the size of the array
```

```
54      sll $s1, $s1, 2
55      move    $s3, $a0
56      add $s1, $s3, $s1       # let $s3 store the last element's address in the array
57      add $s3, $s3, 4
58  loop1:
59      ble $s1, $s3, exit1
60      add $s2, $s3, 0
61  loop2:
62      beq $s1, $s2, exit2
63      lw  $t1, -4($s2)
64      lw  $t2, 0($s2)
65      ble $t1, $t2, noswap
66      sw  $t1, 0($s2)     # swap
67      sw  $t2, -4($s2)        # swap
68  noswap:
69      add $s2, $s2, 4
70      j   loop2
71  exit2:
72      add $s1, $s1, -4
73      j   loop1
74  exit1:
75      # Sorting finished, restore the registers
76      lw $s3, 0($sp)          # Restore $s3
77      addi $sp, $sp, 4        # Adjust stack pointer
78      lw $s2, 0($sp)          # Restore $s2
79      addi $sp, $sp, 4        # Adjust stack pointer
80      lw $s1, 0($sp)          # Restore $s1
81      addi $sp, $sp, 4        # Adjust stack pointer
82      # return
83      jr  $ra
84
85  array_generate:
86      move    $s1, $a0
87      move    $s2, $a1
88  loop_gen:
89      beq $s2, 0, exit
90      li  $v0, 42
91      li  $a1, 10000
92      syscall
93      sw  $a0, 0($s1)
94      add $s2, $s2, -1
95      add $s1, $s1, 4
96      j   loop_gen
97  exit:
98      jr  $ra
99      .data
100 N:   .word   40          # amount of the numbers
```

```
101 array:   .space  4000
```

bubble_sort.asm