

# MIPS 汇编语言实现插入排序

高级体系结构实验报告

姓名：杨珺  
学号：1301213734

2013年 10月 22日

## 1 实验目的

1. 熟悉 MIPS 指令系统
2. 使用 MIPS 指令实现插入排序的递归和迭代算法
3. 统计不同输入情况下两种算法的指令条数

## 2 算法原理

### 2.1 迭代

```
1 INSERTION_SORT (A, n)
   for j = 2 to n do
3   key = A[j]
   { put A[j] into the sorted seqi ← j -1
5   while i > 0 and A[i] > key do
       A[i+1] = A[i]
7   i = i-1
   A[i+1] = key
```

### 2.2 递归

```
INSERTION_SORT (A, n)
2   INSERTION_SORT(A, n-1)
   { put A[n] into the sorted seqi ← n -1
4   while i > 0 and A[i] > A[n] do
       A[i+1] = A[i]
6   i = i-1
   A[i+1] = A[n]
```

## 3 算法内容

### 3.1 算法实现

#### 迭代

```
1 main:
3   .data                                # data field
5   str:.asciiz "Alikeworthpaybusqmdgfzjxcvn" # null-terminated string
7
9   .text                                # text field
11  la      $t0,    str                  # t0 = &str
13  addi    $t0,    $t0,    1
15  sort:
    addi    $t0,    $t0,    1            # t0 ++
    lb      $t1,    0($t0)              # t1 = *t0
```

```

17      beq    $t1,    $0,    print    # end
      sub    $t2,    $t0,    1        # t2 = t0 - 1

19  compare:
      lb     $t3,    0($t2)           # t3 = *t2
21      bgt   $t1,    $t3,    onedone  # t1 > t3
      sb     $t3,    1($t2)           # str[t2+1] = t3
23      sub   $t2,    $t2,    1        # t2 --
      j      compare

25
27  onedone:
      sb     $t1,    1($t2)           # str[t2+1] = t1
      j      sort

29
31  print:
      li     $v0,    4                # print_str
      la     $a0,    str
33      addi  $a0,    $a0,    1
      syscall
35      jr    $ra

```

insertion-sort-iterative.origin.s

## 递归

```

1  main:
3      .data                                # data field
5  str:  .asciiz "Alikeworthpaybusqmdgfzjxcvn" # null-terminated string
7
9      .text                                # text field
11 # sorting algorithm begins
11      la     $s0,    str                # s0 = &str
13      addi   $s0,    $s0,    1          # s0 ++
13      add    $a0,    $s0,    25         # a0 = s0 + 25
15
15      sub    $sp,    $sp,    4          # sp -= 4
17      sw     $ra,    0($sp)             # push ra
17      jal    sort                       # call sort
19
19      lw     $ra,    0($sp)             # pop ra
21      li     $v0,    4                  # print_str
21      addi   $a0,    $s0,    0
23      syscall
23      jr     $ra
25
27  sort:
27      beq    $a0,    $s0,    sortend    # return
29
29      sub    $sp,    $sp,    8          # sp -= 8
31      sw     $a0,    0($sp)             # push a0
31      sw     $ra,    4($sp)             # push ra
33
33      sub    $a0,    $a0,    1          # a0 = a0 - 1
35      jal    sort                       # call self
37
37      lw     $t0,    0($sp)             # pop a0
37      lw     $ra,    4($sp)             # pop ra

```

```

39      addi    $sp,    $sp,    8           # sp += 8
41      sub     $t1,    $t0,    1           # t1 = t0 - 1
41      lb      $t2,    0($t0)             # t2 = *t0
43 sortone:
45      lb      $t3,    0($t1)             # t3 = *t1
45      ble     $t3,    $t2,    onedone     # t3 <= t2
47      sb      $t3,    1($t1)             # str[t1+1] = t3
47      sub     $t1,    $t1,    1           # j --
49      j       sortone
51 onedone:
53      sb      $t2,    1($t1)             # str[t1+1] = t2
55 sortend:
55      jr      $ra

```

insertion-sort-recursive.origin.s

## 3.2 指令统计

### 统计方法

设置全局变量，在执行指令的同时使变量自加，最后输出该全局变量。

1. 设置全局变量 sum，并初始化为0
2. 找出所有 b、j 指令和 label 并标记
3. 代码分块：每两个标记间为一个代码块
4. 在每个块添加：sum+= 该块的指令数
5. 算法结束后输出 sum

### 代码实现

```

1  # algorithm: insertion-sort-iterative
2  # description: instruction number summing
3  # author: yangjvn
5  main:
6      .data                                     # data field
7
8      #str: .asciiz "Aabcdefghijklmnopqrstuvwxyz" # null-terminated string
9      #str: .asciiz "Azyxwvutsrqponmlkjihgfedcba"
10     str: .asciiz "Alikeworthpaybusqmdgfzjxcvn"
11
12     input: .asciiz "Input: "
13     output: .asciiz "\nOutput: "
14     sum: .asciiz "\nInstruction sum: "
15     cr: .asciiz "\n"
17
19     .text                                     # text field
21

```

```

23 # descriptions
    li      $v0,    4                # print_str
25    la      $a0,    input
    syscall
27    la      $a0,    str
    addi    $a0,    $a0,    1
29    syscall
    la      $a0,    output
31    syscall

33    li      $s1,    0                # instruction sum
35
37 # sorting algorithm begins
    la      $t0,    str                # t0 = &str
39    addi    $t0,    $t0,    1
41    addi    $s1,    $s1,    2
43 sort:
    addi    $t0,    $t0,    1          # t0 ++
45    lb      $t1,    0($t0)           # t1 = *t0
47    addi    $s1,    $s1,    3
    beq     $t1,    $0,    print      # end
49
    sub     $t2,    $t0,    1          # t2 = t0 - 1
51    addi    $s1,    $s1,    1
53 compare:
    lb      $t3,    0($t2)            # t3 = *t2
55    addi    $s1,    $s1,    2
    bgt     $t1,    $t3,    onedone   # t1 > t3
57
    sb      $t3,    1($t2)            # str[t2+1] = t3
59    sub     $t2,    $t2,    1          # t2 --
    addi    $s1,    $s1,    3
61    j      compare
63 onedone:
    sb      $t1,    1($t2)            # str[t2+1] = t1
65    addi    $s1,    $s1,    2
    j      sort
67
print:
    li      $v0,    4                # print_str
69    la      $a0,    str
71    addi    $a0,    $a0,    1
    syscall
73    addi    $s1,    $s1,    4
75
77 # descriptions
    li      $v0,    4                # print_str
    la      $a0,    sum
79    syscall
    li      $v0,    1                # print_int
81    add     $a0,    $s1,    0
    syscall
83    li      $v0,    4                # print_str
    la      $a0,    cr

```

```

85      syscall
87
      jr    $ra

```

## insertion-sort-iterative.s

```

# algorithm: insertion-sort-recursive
2 # description: instruction number summing
# author: yangjvn
4
main:
6
      .data                                # data field
8
#str: .ascii "Aabcdefghijklmnopqrstuvwxy" # null-terminated string
10 #str: .ascii "zyxwvutsrqponmlkjihgfedcba"
      str: .ascii "Alikeworthpaybusqmdgfzjxcvn"
12
input: .ascii "Input: "
14 output: .ascii "\nOutput: "
sum:   .ascii "\nInstruction sum: "
16 cr:   .ascii "\n"
18
20
      .text                                # text field
22
# descriptions
24      li      $v0, 4                      # print_str
      la      $a0, input
26      syscall
      la      $a0, str
28      add     $a0, $a0, 1
      syscall
30      la      $a0, output
      syscall
32
      li      $s1, 0                      # instruction sum
34
36 # sorting algorithm begins
      la      $s0, str                    # s0 = &str
38      add     $s0, $s0, 1
      add     $a0, $s0, 25                # a0 = s0 + 25
40
      sub     $sp, $sp, 4                  # sp -= 4
42      sw      $ra, 0($sp)                # push ra
44
      addi    $s1, $s1, 6
      jal     sort                        # call sort
46
      lw      $ra, 0($sp)                 # pop ra
48
      li      $v0, 4                      # print_str
50      add     $a0, $s0, 0
      syscall
52
      addi    $s1, $s1, 4
54
# descriptions
56      li      $v0, 4                      # print_str

```

```

58      la      $a0,    sum
59      syscall
60      li      $v0,    1          # print_int
61      add     $a0,    $s1,    0
62      syscall
63      li      $v0,    4          # print_str
64      la      $a0,    cr
65      syscall
66
67      jr      $ra
68
69  sort:
70      addi    $s1,    $s1,    1
71      beq     $a0,    $s0,    sortend    # return
72
73      sub     $sp,    $sp,    8          # sp -= 8
74      sw      $a0,    0($sp)            # push a0
75      sw      $ra,    4($sp)            # push ra
76
77      sub     $a0,    $a0,    1          # a0 = a0 - 1
78
79      addi    $s1,    $s1,    5
80      jal     sort                      # call self
81
82      lw      $t0,    0($sp)            # pop a0
83      lw      $ra,    4($sp)            # pop ra
84      addi    $sp,    $sp,    8          # sp += 8
85
86      sub     $t1,    $t0,    1          # t1 = t0 - 1
87      lb      $t2,    0($t0)            # t2 = *t0
88
89      addi    $s1,    $s1,    5
90
91  sortone:
92      lb      $t3,    0($t1)            # t3 = *t1
93      addi    $s1,    $s1,    2
94      ble     $t3,    $t2,    onedone    # t3 <= t2
95
96      sb      $t3,    1($t1)            # str[t1+1] = t3
97      sub     $t1,    $t1,    1          # j --
98      addi    $s1,    $s1,    3
99      j       sortone
100
101  onedone:
102      sb      $t2,    1($t1)            # str[t1+1] = t2
103      addi    $s1,    $s1,    1
104
105  sortend:
106      addi    $s1,    $s1,    1
107      jr      $ra                      # return

```

insertion-sort-recursive.s

运行结果

```
[hurtle@hurtle-station mips-report]$ spim -file insertion-sort-iterative.s
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Input: abcdefghijklmnopqrstuvwxyz
Output: abcdefghijklmnopqrstuvwxyz
Instruction sum: 209
[hurtle@hurtle-station mips-report]$ spim -file insertion-sort-iterative.s
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Input: zyxwvutsrqponmlkjihgfedcba
Output: abcdefghijklmnopqrstuvwxyz
Instruction sum: 1834
[hurtle@hurtle-station mips-report]$ spim -file insertion-sort-iterative.s
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Input: likeworthpaybusqmdgfzjxcvn
Output: abcdefghijklmnopqrstuvwxyz
Instruction sum: 989
[hurtle@hurtle-station mips-report]$ |
```

图 3.1: 迭代算法结果

```
[hurtle@hurtle-station mips-report]$ spim -file insertion-sort-recursive.s
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Input: abcdefghijklmnopqrstuvwxyz
Output: abcdefghijklmnopqrstuvwxyz
Instruction sum: 387
[hurtle@hurtle-station mips-report]$ spim -file insertion-sort-recursive.s
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Input: zyxwvutsrqponmlkjihgfedcba
Output: abcdefghijklmnopqrstuvwxyz
Instruction sum: 2012
[hurtle@hurtle-station mips-report]$ spim -file insertion-sort-recursive.s
SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
Input: likeworthpaybusqmdgfzjxcvn
Output: abcdefghijklmnopqrstuvwxyz
Instruction sum: 1167
[hurtle@hurtle-station mips-report]$
```

图 3.2: 递归算法结果



表 3.1: 指令条数统计结果

算法	迭代	递归
正序	209	387
倒序	1834	2012
乱序	989	1167

## 4 讨论

对于插入算法的两种实现，时间复杂度均为  $O(n^2)$ ，平均情况与最坏情况有相同的量级。下面具体估计在最坏情况下的指令条数。递归和迭代两种算法中，一共包含三部分的指令：

1. 内层循环：迭代地试探前一个元素，以插入当前元素。该部分同时出现在两种算法中。
2. 递归调用：递归的调用过程，该部分只出现在递归算法中。
3. 外层循环：向后迭代地取出元素，并试图插入到排好序的部分。该部分只出现在迭代算法中。

### 4.1 内层循环

在这部分中，迭代地试探前一个元素，以插入当前元素。该部分同时出现在两种算法中。考虑有哨兵的情况（避免每次进行越界判断）而不计哨兵的开销，该部分包含5条指令：

1. 取值 (lb)：取出前一个值。
2. 判断 (bgt)：判断该值是否小于当前要排序的元素，如果判断成功将跳出该次循环。
3. 后移 (sb)：最坏情况中上述判断总是失败的，最终会后移该元素。
4. 自减 (sub)：内层循环下标前移。
5. 跳转 (j)：跳至下次循环。

内层循环执行次数为总的比较次数，即

$$n_{\text{comp}} = \sum_{i=2}^n i = \frac{n(n+1)}{2} - 1 = 350 \quad (4.1)$$

### 4.2 递归调用

递归的调用过程，该部分只出现在递归算法中。考虑参数最少的情况（1个参数），该部分包含8条指令：

1. 保存现场（3条）
  - (a) 参数压栈 (sw)。
  - (b) 返回地址压栈 (sw)。
  - (c) 降低栈指针 (sub)。

## 2. 调用子函数 (2条)

- (a) 计算参数 (sub) 。
- (b) 跳转 (jal) 。

## 3. 恢复现场 (3条)

- (a) 参数出栈 (sw) 。
- (b) 返回地址出栈 (sw) 。
- (c) 提高栈指针 (addi) 。

完整的递归调用次数为

$$n_{\text{call}} = n - 1 = 25 \quad (4.2)$$

## 4.3 外层循环

在该部分过程中，向后迭代地取出元素，并试图插入到排好序的部分。该部分只出现在迭代算法中。该部分包含6条指令：

1. 判断越界 (bge) 。
2. 取值 (lb)：取出下一个要插入的元素。
3. 初始化内层循环下标 (sub) 。
4. 插入 (sb)：在最坏情况中，子函数调用结束后一定会引起插入操作。
5. 自加 (addi)：外层循环下标后移。
6. 跳转 (j)：跳至下次循环。

外层循环执行次数为

$$n_{\text{out}} = n - 1 = 25 \quad (4.3)$$

## 4.4 总的指令条数

在最坏情况下，迭代算法的指令条数约为

$$n_{\text{iterative}} = 5n_{\text{comp}} + 6n_{\text{out}} = 1900 \quad (4.4)$$

在最坏情况下，递归算法的指令条数约为

$$n_{\text{recursive}} = 5n_{\text{comp}} + 8n_{\text{call}} = 1950 \quad (4.5)$$

无哨兵的情况，由于在内层循环需要添加一条越界判断，两种算法的条数各加 350 条。实际执行中，迭代为1834条，递归为2012条。

## 5 参考文献

1. SPIM 指令集文档：<http://vhouten.home.xs4all.nl/mipsel/r3000-isa.html>
2. SPIM 文档：<http://www.dsi.unive.it/~arcb/LAB/spim.htm>