

中国科学技术大学计算机学院
《计算机系统概论实验》报告



实验题目：成绩排序与统计

学生姓名：王章瀚

学生学号：PB18111697

完成日期：2019 年 11 月 10 日

1 实验要求

1.1 概述

对 60 个学生的成绩进行排序，其中成绩没有相同值。在排序的基础上，对每个学生的成绩给予 A/B/C/D 的评价，由此统计各等级学生数量。

评价标准如下：

- 排名为前 30%，并且分数在 85 分以上，获 A。
- 排名为前 50%，并且分数在 75 分以上，获 B。
- 分数 ≤ 95 ，获 D。
- 其他同学获 C。

基本要求：

用 LC-3 汇编语言编写程序。程序应能够对所有分数进行排序，并计数获得 A/B/C/D 等级的人数。

1.2 输入与输出

程序输入是 60 个未经排序的成绩。成绩取值范围为 0 到 100。放在从 x3200 开始的内存空间中，最后一个在 x323B。可以认为这些成绩没有相同的。

程序有两项输出：

- 排序后的 60 的成绩。放在从 x4000 开始的内存空间。最高分放在 x4000。
- 获得 A/B/C/D 各个等级的学生数，A 的放在 x4100；B 的放在 x4101；C 的放在 x4102；D 的放在 x4103。

2 设计思路

考虑到成绩的取值范围为 0 到 100，可以采用桶排序的思路，对每一个成绩的人数进行计数，最终能够以 $O(n)$ 的时间复杂度完成任务。

此外，本题题目要求中，有一个特殊点，即成绩没有重复。因此为了防止重新由内存地址算出成绩的繁琐，可以分以下两步：

- 将 x4000 到 x4064 赋值为 -1 后，遍历 x3200 到 x3238，直接将成绩按如下规则存储：假设当前处理的成绩为 g，则将它存在 x4064-g 的空间上。例如成绩 100，则存放在 x4000 处。
- 遍历 x4000 开始的 60 个成绩，将成绩前挪，消去空隙。

举个例子说明（学生数为 3）：

地址	第一步后	地址	第二步后
x4000	#100	x4000	#100
x4001	#-1	x4000	#97
x4002	#-1	x4000	#95
x4003	#97	x4000	#-1
x4004	#-1	x4000	#-1
x4005	#95	x4000	#-1

3 关键代码讲解

碍于篇幅原因，完整代码放在附录 A 中。其中代码主要分为四个部分：

- PART 1：将 x4000 到 x4064 的内存空间全部置-1。
- PART 2：遍历成绩，放到上述算法中描述的对应内存地址上。
- PART 3：对 x4000 到 x4064 中的成绩进行整理，同时统计各等级人数。
- PART 4：把剩余的空间（x403C 到 x4064）置 0

以下针对一些关键代码进行分析：

3.1 对于第一步的存放

存放过程中，由前述方法计算对应成绩应该存放的地址，用 STR 指令存储。

寄存器使用说明：

- R1：当前正在处理的成绩的内存地址
- R2：最后一个成绩的内存地址的相反数（便于作减法）
- R3：目标地址的最后一个地址（即 x4064，由 x4000+#100 得到）
- R4：当前处理的成绩
- R5：临时使用

```
1  ;;;; PART 2 ;;;;
2  ; search all the grade and put them into the according memory
3  ; R1    grade location
4  ; R2    nGRADE_END
5  ; R3    DEST_END100
6  ; R4    grade
```

```

7      ; R5      only for temp
8
9          LD      R1, GRADE    ; load the grades' address
10         LD      R2, nGRADE_END ; load the opposite number of grades' end
11         address
12         LD      R3, DEST_END100 ; load DEST_END100
13 SEARCH_LOOP LDR      R4, R1, #0 ; load the grade to R4 by R1
14             NOT      R5, R4
15             ADD      R5, R5, #1
16             ADD      R5, R5, R3 ; get the position the grade should be
17             STR      R4, R5, #0 ; store the grade into the position
18             ADD      R1, R1, #1 ; R1 ++
19             ADD      R5, R1, R2
20             BRnz     SEARCH_LOOP ; check whether finished.
21

```

3.2 对成绩对应等级的判断

经过排名后，可以通过存储该成绩的内存地址来判断排名，而成绩可以直接判断。有如下代码：

寄存器使用说明：

- R1：将要最终存放成绩的地址
- R2：要确认是否有成绩（即不是-1）的地址
- R3：最终存放成绩的地址的相反数（即 x4064, 由 x4000+#100 得到）
- R4：当前处理的成绩
- R5：临时使用

```

1      ;;;; PART 3 ;;;;
2      ; put the grades into the corret positions
3      ; R1      destination
4      ; R2      to find next filled memory
5      ; R3      nDEST_END
6      ; R4      grade
7      ; R5      only for temp
8
9      ; try A
10         LD      R5, A_GRADE
11         NOT      R5, R5
12         ADD      R5, R5, #1
13         ADD      R5, R4, R5 ; R5 <- (R4-A_GRADE)
14         BRn     NOTA      ; if (R4 < A_GRADE) goto NOTA
15         LD      R5, A_RANK
16         NOT      R5, R5
17         ADD      R5, R5, #1

```

```

18      ADD      R5, R1, R5   ; R5 ← (R1-A_RANK)
19      BRp      NOTA        ; else if(R5 > 0) goto NOTA
20      LDI      R5, As       ; load the address of statistics information.
21      ADD      R5, R5, #1   ; As++
22      STI      R5, As
23      BRnzp    LOOP_PRE
24

```

4 调试分析

由于设计的各种数值较多，导致编写代码时有些混乱。经过逐步调试，最终纠正过来。

5 代码测试

为测试代码，编写了生成测试数据的代码。（见附录 B）

5.1 测试数据 1

```

1      .ORIG      x3200
2      .FILL      #50
3      .FILL      #1
4      .FILL      #32
5      .FILL      #41
6      .FILL      #88
7      ;... here skipped some
8      .FILL      #24
9      .FILL      #52
10     .FILL      #54
11     .FILL      #40
12     .FILL      #14
13     .END
14     ; As:9  Bs:5  Cs:6  Ds40
15

```

R3	xBFC5	-16443	R7	xFD75	-651	CC	P
* x4000	0000000001100011	x0063				NOP	
* x4001	0000000001100001	x0061				NOP	
* x4002	0000000001011101	x005D				NOP	
* x4003	0000000001011100	x005C				NOP	
* x4004	0000000001011011	x005B				NOP	
* x4005	0000000001011010	x005A				NOP	
* x4006	0000000001011001	x0059				NOP	
* x4007	0000000001011000	x0058				NOP	
* x4008	0000000001010111	x0057				NOP	
* x4009	0000000001010011	x0053				NOP	
* x400A	0000000001010010	x0052				NOP	
* x400B	0000000001010001	x0051				NOP	
* x400C	0000000001010000	x0050				NOP	
* x400D	0000000001001100	x004C				NOP	
* x400E	0000000001001111	x0047				NOP	
* x400F	0000000001000110	x0046				NOP	
* x4010	0000000001000101	x0045				NOP	
* x4011	0000000001000011	x0043				NOP	
* x4012	0000000001000000	x0040				NOP	

图 1: 排序结果部分截图

R3	xBFC5	-16443	R7	xFD75	-651	CC	P
* x4100	0000000000000101	x0005				NOP	
* x4101	0000000000000101	x0005				NOP	
* x4102	0000000000000110	x0006				NOP	
* x4103	0000000000010100	x0028				NOP	
* x4104	0000000000000000	x0000				NOP	
* x4105	0000000000000000	x0000				NOP	

图 2: 统计结果截图

5.2 测试数据 2

1	.ORIG	x3200
2	.FILL	#41
3	.FILL	#67
4	.FILL	#58
5	.FILL	#14
6	.FILL	#26
7	;... here skipped some	
8	.FILL	#62
9	.FILL	#27
10	.FILL	#6
11	.FILL	#28
12	.FILL	#90
13	.END	
14	; As:10 Bs:6 Cs:11 Ds33	
15		
16		

R3	xBFC5	-16443	R7	xFD75	-651	CC	P
* x4000	0000000001100010	x0062				NOP	
* x4001	0000000001100000	x0060				NOP	
* x4002	0000000001011111	x005F				NOP	
* x4003	0000000001011110	x005E				NOP	
* x4004	0000000001011101	x005D				NOP	
* x4005	0000000001011011	x005B				NOP	
* x4006	0000000001011010	x005A				NOP	
* x4007	0000000001011001	x0059				NOP	
* x4008	0000000001011000	x0058				NOP	
* x4009	0000000001010110	x0056				NOP	
* x400A	0000000001010100	x0054				NOP	
* x400B	0000000001010011	x0053				NOP	
* x400C	0000000001010000	x0050				NOP	
* x400D	0000000001001111	x004F				NOP	
* x400E	0000000001001101	x004D				NOP	
* x400F	0000000001001100	x004C				NOP	
* x4010	0000000001001010	x004A				NOP	
* x4011	0000000001001001	x0049				NOP	
* x4012	0000000001001000	x0048				NOP	
* x4013	0000000001000110	x0046				NOP	
* x4014	0000000001000101	x0045				NOP	
* x4015	0000000001000011	x0043				NOP	

图 3: 排序结果部分截图

R3	xBFC5	-16443	R7	xFD75	-651	CC	P
* x4100	0000000000000101	x000A				NOP	
* x4101	0000000000000110	x0006				NOP	
* x4102	0000000000000101	x000B				NOP	
* x4103	0000000000010001	x0021				NOP	
* x4104	0000000000000000	x0000				NOP	
* x4105	0000000000000000	x0000				NOP	

图 4: 统计结果截图

6 实验总结

本次实验中，经过一系列的编程练习，我对于 LC-3 的汇编代码更加熟悉，同时也能够熟练使用对应 Simulator。

此外，初见题目便匆忙开始写代码，没有过多的分析，最终使用了选择排序这样的代码来完成题目，虽然也能完成，但执行指令数（平均在 40000 左右）相对于用现在的算法的指令数（2800 左右）较多。经过思考，修改方案为桶排序，并进行一定程度的改进，最终有了上述结果。因此本次实验的一个较大收获是，它让我明白了分析的重要性，而不应拿到一个任务就急忙上手。

7 附录

7.1 附录 A

```
1      .ORIG      x3000      ; start from x3000
2
3      ;;;; PART 1 ;;;;
4      ; make the destination memory into 0, from DEST to DEST_END
5      ; R1  DEST
6      ; R2  nDEST_END100
7      ; R3  -1
8
9      LD      R1, DEST
10     LD      R2, nDEST_END100
11     AND     R3, R3, #0
12     ADD     R3, R3, #-1
13 SETn1_LOOP STR     R3, R1, #0 ; store 0 into the position
14     ADD     R1, R1, #1 ; R1 ++
15     ADD     R5, R1, R2
16     BRnz   SETn1_LOOP ; check whether finished.
17
18     ;;;; PART 2 ;;;;
19     ; search all the grade and put them into the according memory
20     ; R1  grade location
21     ; R2  nGRADE_END
22     ; R3  DEST_END100
23     ; R4  grade
24     ; R5  only for temp
25
26     LD      R1, GRADE ; load the grades' address
27     LD      R2, nGRADE_END ; load the opposite number of grades' end
28 address    LD      R3, DEST_END100 ; load DEST_END100
29
```

```

30 SEARCH_LOOP LDR    R4, R1, #0 ; load the grade to R4 by R1
31             NOT     R5, R4
32             ADD     R5, R5, #1
33             ADD     R5, R5, R3 ; get the position the grade should be
34             STR     R4, R5, #0 ; store the grade into the position
35             ADD     R1, R1, #1 ; R1 ++
36             ADD     R5, R1, R2
37             BRnz    SEARCH_LOOP ; check whether finished.
38
39 ;;;; PART 3 ;;;;
40 ; put the grades into the corret positions
41 ; R1 destination
42 ; R2 to find next filled memory
43 ; R3 nDEST_END
44 ; R4 grade
45 ; R5 only for temp
46         LD     R1, DEST
47         LD     R2, DEST
48         LD     R3, nDEST_END
49
50 SORT_LOOP ADD     R2, R2, #1
51         LDR     R4, R2, #-1 ; load the grade to R4 by R2
52         BRn     SORT_LOOP ; if the grade is negative(-1), take next
53
54 ; else find the A/B/C/D for this grade.
55 ; try A
56         LD     R5, A_GRADE
57         NOT     R5, R5
58         ADD     R5, R5, #1
59         ADD     R5, R4, R5 ; R5 <- (R4-A_GRADE)
60         BRn     NOTA ; if (R4 < A_GRADE) goto NOTA
61         LD     R5, A_RANK
62         NOT     R5, R5
63         ADD     R5, R5, #1
64         ADD     R5, R1, R5 ; R5 <- (R1-A_RANK)
65         BRp     NOTA ; else if (R5 > 0) goto NOTA
66         LDI     R5, As ; load the address of statistics information.
67         ADD     R5, R5, #1 ; As++
68         STI     R5, As
69         BRnzp   LOOP_PRE
70
71 ; else try B
72 NOTA     LD     R5, B_GRADE
73         NOT     R5, R5
74         ADD     R5, R5, #1
75         ADD     R5, R4, R5 ; R5 <- (R4-B_GRADE)
76         BRn     NOTB ; if (R4 < B_GRADE) goto NOTB
77         LD     R5, B_RANK
78         NOT     R5, R5
79         ADD     R5, R5, #1
80         ADD     R5, R1, R5 ; R5 <- (R1-B_RANK)
81         BRp     NOTB ; else if (R5 > 0) goto NOTB
82         LDI     R5, Bs ; load the address of statistics information.
83         ADD     R5, R5, #1 ; Bs++
84         STI     R5, Bs ; store the new Bs

```



```

84          BRnzp  LOOP_PRE
85      ; else try D
86      NOTB      LD      R5, D_GRADE
87              NOT      R5, R5
88              ADD      R5, R5, #1
89              ADD      R5, R4, R5 ; R5 <- (R4-D_GRADE)
90              BRp      NOTD      ; if (R4 < D_GRADE) goto NOTD
91              LDI      R5, Ds      ; load the address of statistics information.
92              ADD      R5, R5, #1 ; Ds++
93              STI      R5, Ds      ; store the new Ds
94              BRnzp  LOOP_PRE
95      ; else, C
96      NOTD      LDI      R5, Cs      ; load the address of statistics information.
97              ADD      R5, R5, #1 ; Cs++
98              STI      R5, Cs      ; store the new Cs
99
100      ; above finished get the A/B/C/D
101      LOOP_PRE  STR      R4, R1, #0 ; store the grade to the correct position
102              ADD      R1, R1, #1 ; R1++
103              ADD      R5, R3, R1
104              BRnz  SORT_LOOP ;
105
106      ;;;; PART 4 ;;;;
107      ; clear the following into 0
108      ; R1 destination
109      ; R2 nDEST_END100
110      ; R4 0
111
112          LD      R2, nDEST_END100
113          AND      R4, R4, #0
114      SET0_LOOP STR      R4, R1, #0 ; store 0 into the position
115          ADD      R1, R1, #1 ; R1 ++
116          ADD      R5, R1, R2
117          BRnz  SET0_LOOP ; check whether finished.
118
119          HALT
120
121      A_GRADE  .FILL      85      ; the least grade requirement for A
122      B_GRADE  .FILL      75      ; the least grade requirement for B
123      D_GRADE  .FILL      59      ; the grade requirement for D
124      A_RANK   .FILL      x4011 ; the rank requirement for A.
125      B_RANK   .FILL      x401C ; the rank requirement for B.
126
127      GRADE     .FILL      x3200 ; the start address of grades
128      nGRADE_END .FILL      x-323B ; the opposite number of end address of grades
129      DEST      .FILL      x4000 ; the start address of grades
130      DEST_END100 .FILL      x4064 ; the end address of destination.
131      nDEST_END100 .FILL      x-4064 ; the opposite number of end address of destination.
132      nDEST_END .FILL      x-403B ; the opposite number of end address of destination.
133      As        .FILL      x4100 ; the address of number of As.
134      Bs        .FILL      x4101 ; the address of number of Bs.
135      Cs        .FILL      x4102 ; the address of number of Cs.
136      Ds        .FILL      x4103 ; the address of number of Ds.
137

```

138		.END
139		

7.2 附录 B

```

1  #include <iostream>
2  #include <algorithm>
3  #include <cstdlib>
4  #include <ctime>
5  using namespace std;
6
7  int cmp(int a, int b){
8      return b < a;
9  }
10
11 int main() {
12     FILE* f = fopen("data.asm", "w");
13     srand((unsigned)time(NULL)); //将系统时间作为随机种子
14     int n, temp;
15     cin >> n;
16     int* scores = new int(n);
17     fprintf(stdout, "\t.ORIG\t\ttx3200\n");
18     fprintf(f, "\t.ORIG\t\ttx3200\n");
19     for(int i = 0; i < n; i++){
20         temp = rand()%100;
21         bool flag = 1;
22         while(flag) {
23             int j;
24             for(j = 0; j < i; j++) {
25                 if(scores[j] == temp){
26                     temp = rand()%100;
27                     break;
28                 }
29             }
30             if(j == i)
31                 flag = 0;
32         }
33         scores[i] = temp;
34         fprintf(stdout, "\t.FILL\t\t##\d\n", temp);
35         fprintf(f, "\t.FILL\t\t##\d\n", temp);
36     }
37     fprintf(f, "\t.END\n");
38     fprintf(stdout, "\t.END\n");
39
40     int grades[4] = {0};
41     sort(scores, scores+n, cmp);
42     for(int i = 0; i < n; i++){
43         if(scores[i] >= 75) {
44             if(scores[i] >= 85 && i < n*0.3)

```

```

45         grades[0]++;
46         else if(i < n*0.5)
47             grades[1]++;
48     }
49     else if(scores[i] < 60)
50         grades[3]++;
51     else
52         grades[2]++;
53     printf("%x\n", scores[i]);
54 }
55     fprintf(f, " ; As:%d\tBs:%d\tCs:%d\tDs%d\n", grades[0], grades[1], grades[2],
56     grades[3]);
57     fprintf(stdout, " ; As:%d\tBs:%d\tCs:%d\tDs%d\n", grades[0], grades[1], grades
58     [2], grades[3]);
59     fclose(f);

```