ICS_Lab4_Report

Xiaoma

2022 年 12 月 27 日

实验目的

使用 LC-3 汇编命令实现对 16 个人的成绩的升序排列，并求出这 16 个人中获得评级 A，B 的数量。
已知当该同学的成绩为 85 分及以上，且其排名为 25% 时，评级为 A。
当该同学的成绩低于 85 分但大于等于 75 分，且其排名为 50% 时，评级为 B。

- 16 个人的成绩存储在以`x4000`开始的连续内存空间中
- 每个人的成绩都在 0-100 之间，且每个人的成绩都不相同
- 排序后的成绩存储在以`x5000`开始的连续内存空间中
- 评级 A,B 的数量存储在内存位置`x5100`,`x5101`中

实验原理

为了降低代码复杂度，我们使用选择排序的方式，既要对 16 个人的程序进行升序排列，又要统计评级 A，B 的数量，已知 A，B 既有分数限制又有排名限制，所以如果在升序排序的同时判断评级，则会因无法判断其排名而得到正确结果。所以我们采用降序排序的同时判断评级，最后逆序存储序列。
根据该思想可以得到伪代码：

Algorithm 1: mySort

Input: The scores of students: score[16];

Output: The number of A: num_a; The number of B: num_b;

```
num_a = 0;
num_b = 0;
for i = 16; i >= 0; − − i do
    max = i;
    j = i - 1;
    for ; j >= 0; − − j do
        if score[max] < score[j] then
            max = j;
    swap(score[max], score[i]);
    if score[i] >= 85&&num_a < 4 then
        num_a += 1;
    else if score[i] >= 85&&num_a + num_b < 8 then
        num_b += 1;
    else if score[i] >= 75&&num_a + num_b < 8 then
        num_b += 1;
return num_a, num_b;
```

实验步骤

选择排序的实现

对于通常情况下的选择排序，第二次循环起始位置应该为初始 max 的下一个位置，但若果这样做，会导致两个循环的判断终止条件不同，为了降低代码复杂度，设定第二次循环的起始位置为 max。

判断评级

设定 num_a, num_b 为此时评级为 A，B 的数量。

$$\begin{cases} num\_a+=1 & score \geq 85, num\_a < 4 \\ num\_b+=1 & score \geq 85, num\_a + num\_b < 8, num\_a \geq 4 \\ num\_b+=1 & 75 \leq score < 85, num\_a + num\_b < 8 \end{cases}$$

倒序存储

对降序排序的序列进行倒序存储，得到的即为正向的升序排列。

代码讲解

初始化变量

将 $R0$ 作为 16 个成绩的指针，$R1$ 为循环的最后一个位置，$R6, R7$ 存储评级 A，B 的数量，即

$$R0 \leftarrow \&score, R1 \leftarrow \&score[15]$$

```
1    LD R0, SCORE
2    ADD R1, R0, #15
3    AND R6, R6, #0
4    AND R7, R7, #0
```

外部循环

选择排序的外部循环，初始化 max, j 后进入内部循环，从内部循环返回后，进入 $JUDGE$ 判断评级，然后将局部最大值写入相应位置，逆序存储结果，循环结束时，进入 $STORE$ 存储评级结果，即

$$swap(score[max], score[i])$$

$$sortScore[16 - i] \leftarrow score[i]$$

```
5     LOOPA
6     AND R2, R2, #0
7     ADD R2, R0, #0
8     AND R3, R3, #0
9     ADD R3, R0, #0
10    BRnzp LOOPB
11    RETB
12    LDR R4, R2, #0
13    LDR R5, R0, #0
14    STR R4, R0, #0
15    STR R5, R2, #0
16    LD R3, RESULTS
17    LD R2, SCORE
18    ADD R3, R2, R3
19    NOT R2, R0
20    ADD R2, R2, #1
21    ADD R2, R2, R1
22    ADD R3, R3, R2
23    STR R4, R3, #0
24    BRnzp JUDGE
25    RETJ
26    ADD R0, R0, #1
27    NOT R4, R0
28    ADD R4, R4, #1
29    ADD R4, R1, R4
30    BRn STORE
31    BRnzp LOOPA
```

内部循环

选择排序的内部循环，若当前指针指向的值大于 max 指向的值，则更新 max，即

$$max \leftarrow argmax_{x \in \{max, j\}}(score[x])$$

```
32      LOOPB
33      LDR R4, R2, #0
34      LDR R5, R3, #0
35      NOT R4, R4
36      ADD R4, R4, #1
37      ADD R4, R5, R4
38      BRnz #2
39      AND R2, R2, #0
40      ADD R2, R3, #0
41      ADD R3, R3, #1
42      NOT R4, R3
43      ADD R4, R4, #1
44      ADD R4, R1, R4
45      BRzp LOOPB
46      BRnzp RETB
```

判断评级

若成绩不低于 85 且排名不低于 25%，则评级 A；若成绩不低于 85 且排名低于 25% 高于 50%，则评级 B; 若成绩低于 85 不低于 75 且排名低于 25% 高于 50%，则评级 B，即

$$\begin{cases} num\_a+ = 1 & score \geq 85, num\_a < 4 \\ num\_b+ = 1 & score \geq 85, num\_a + num\_b < 8, num\_a \geq 4 \\ num\_b+ = 1 & 75 \leq score < 85, num\_a + num\_b < 8 \end{cases}$$

```
47      JUDGE
48      LDR R4, R0, #0
49      LD R5, SCOREMARKA
50      ADD R4, R4, R5
```

```
51      BRn #4
52      ADD R5, R6, #-4
53      BRz #4
54      ADD R6, R6, #1
55      BRnzp RETJ
56      ADD R4, R4, #10
57      BRn RETJ
58      ADD R5, R6, R7
59      ADD R5, R5, #-8
60      BRz RETJ
61      ADD R7, R7, #1
62      BRnzp RETJ
```

存储结果

分别将评级 A，B 的数量存储到对应地址，即

$$mem[\texttt{x5100}] \leftarrow num\_a$$

$$mem[\texttt{x5101}] \leftarrow num\_b$$

```
63      STORE
64      STI R6, RESULTA
65      STI R7, RESULTB
```

实验结果

依次对实验文档给出的例子进行测试，结果如下：

汇编评测

3 / 3 个通过测试用例

- 平均指令数: 2036.3333333333333
- 通过 100:95:90:85:80:60:55:50:45:40:35:30:25:20:10:0, 指令数: 2010, 输出: 0,10,20,25,30,35,40,45,50,55,60,80,85,90,95,100,4,1
- 通过 95:100:0:50:45:40:80:65:70:75:35:20:25:15:10:90, 指令数: 2043, 输出: 0,10,15,20,25,35,40,45,50,65,70,75,80,90,95,100,3,2
- 通过 88:77:66:55:99:33:44:22:11:10:9:98:97:53:57:21, 指令数: 2056, 输出: 9,10,11,21,22,33,44,53,55,57,66,77,88,97,98,99,4,1

自行编写了部分测试例子，分别考虑了评级的几种不同情况，结果如下：

汇编评测

4 / 4 个通过测试用例

- 平均指令数: 2136.5
- 通过 0:1:2:3:4:5:6:7:8:10:11:12:13:14:15:16, 指令数: 2125, 输出: 0,1,2,3,4,5,6,7,8,10,11,12,13,14,15,16,0,0
- 通过 0:1:2:3:4:5:6:7:8:10:11:12:13:14:15:95, 指令数: 2127, 输出: 0,1,2,3,4,5,6,7,8,10,11,12,13,14,15,95,1,0
- 通过 0:1:2:3:4:5:6:7:8:9:10:91:92:93:94:95, 指令数: 2138, 输出: 0,1,2,3,4,5,6,7,8,9,10,91,92,93,94,95,4,1
- 通过 0:1:2:3:4:5:6:81:82:83:84:91:92:93:94:95, 指令数: 2156, 输出: 0,1,2,3,4,5,6,81,82,83,84,91,92,93,94,95,4,4