

中国科学技术大学计算机学院
《嵌入式系统设计方法》



实验题目： LAB1 字符串排序

学生姓名： 钟书锐

学生学号： PB19000362

完成日期： 2021.11.8

嵌入式系统设计方法-LAB1-字符串排序

- PB19000362
- 钟书锐

一、实验要求

- 字符串排序，从键盘输入至少 5 个字符串，每个字符串的长度不小于 10；经过排序之后，从小到大输出排序的结果。
- 在 ADS 的 console 中显示输出结果
- 使用 C 语言完成字符串的输入及输出
- 使用 ARM 汇编语言完成排序操作

二、实验环境

- 处理器 Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz
- 操作系统 Windows 10
- ADS 1.2

三、实验思路

1. main.c

- main 函数中定义结构体 struct string_sort，结构体包含待排序的字符串数量，二维数组第二维大小和待排序的字符串

```
struct string_sort
{
    int num;
    int len;
    char s[MAXNUM][MAXSIZE];
};
```

- 首先读取用户输入的 num 信息，确定字符串数量。
- 然后读取字符串并存储。
- 调用 sort 进行排序。
- 输出排序后的字符串集合。

2. sort.s

- 采用冒泡排序的方式进行排序
- 针对传入的 string_sort 类型的结构体，用 r1 存储 num，r2 存储 len，r3 存储二维数组的地址

```
ldr    r1, [r0]           ;r1=num
ldr    r2, [r0,#4]        ;r2=len
add    r3, r0, #8         ;将r3指向s二维数组
```

- 令 r4, r5 分别存储 i, j 进行冒泡排序，对应的逻辑代码如下

```

for (i = 0; i < num - 1; i++)
    for (j = 0; j < num - 1 - i; j++)

```

- 在 2 个循环中调用 compare 进行比较和交换，使用 r0,r1 传入等待比较的 2 个字符串的地址，同时 r2 一直存储着 len 信息

```

STMFD    SP!, {R0-R6,lr}
mul      r0, r2, r5
add      r0, r0, r3          ;r0 指向第一个字符串
add      r1, r0, r2          ;r1 指向第二个字符串
bl       compare
ldmfd    SP!, {R0-R6,lr}

```

3. compare.s

- r0,r1 为传入的等待比较的 2 个字符串的地址。
- r7 初始设置为 1，为排序的计数器。
- 2 个字符串的每个字符依次进行比较。
- 若前者的字符小于后者，或者 r7 计数器大于 len(代表 2 个字符串相等)，此时 2 个字符串不需要交换，直接返回，若前者的字符大于后者，则需要交换 2 个字符串的内容

loop

```

cmp      r7, r2
bgt      end
ldrb     r5, [r3]
ldrb     r6, [r4]
cmp      r5, r6
blt      end          ;s1<s2
bgt      exchange     ;s1>s2
add      r3, r3, #1
add      r4, r4, #1
add      r7, r7, #1
b        loop

```

- 在本程序中采用交换内容，而非交换指针，符合冒泡排序一般性思路。
- 交换字符串时，r7 设置为 1，为交换所用的计数器；r5, r6 为 2 个要交换的字符；r3, r4 指向要交换的地址。

exchange

```

mov      r7, #1          ;i=1
mov      r3, r0
mov      r4, r1

```

loop_exchange

```

cmp      r7, r2
bgt      end
ldrb     r5, [r3]
ldrb     r6, [r4]
strb     r5, [r4]
strb     r6, [r3]
add      r3, r3, #1
add      r4, r4, #1
add      r7, r7, #1

```

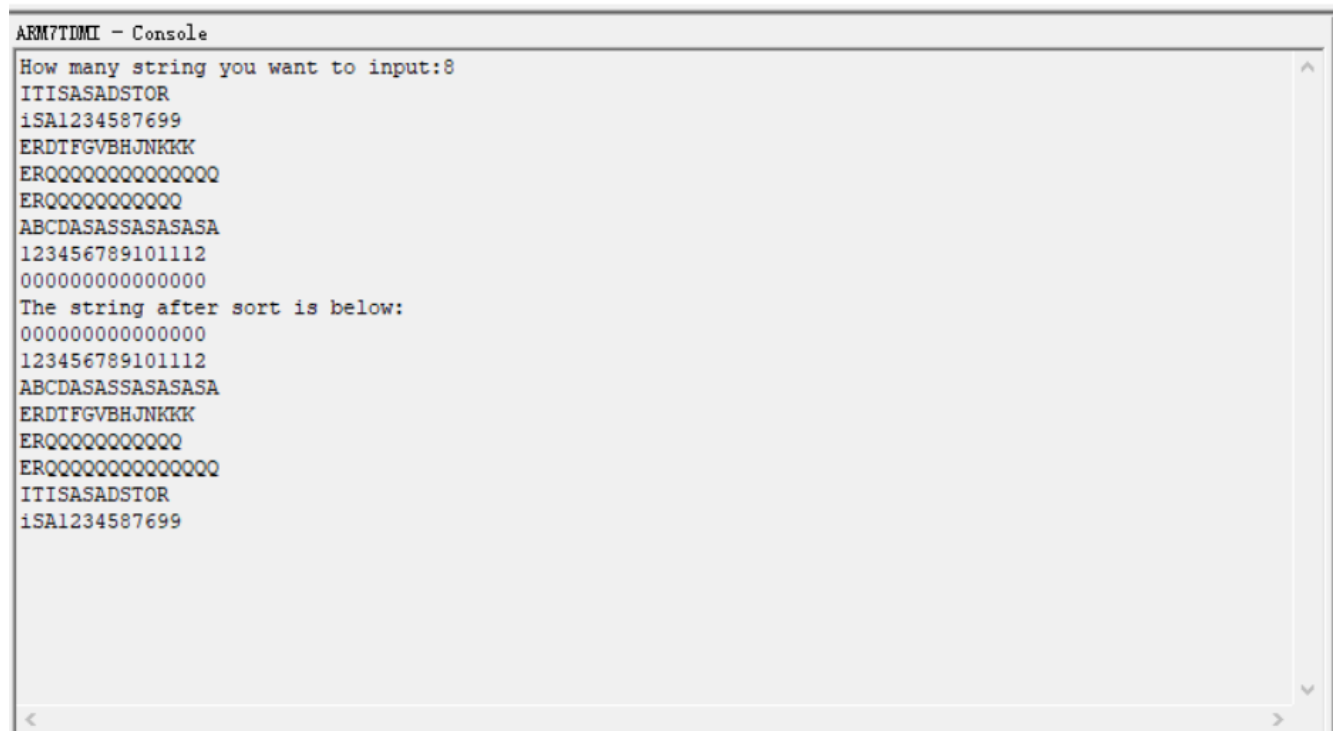
四、实验测试

- input:

```
8
ITISASADSTOR
iSA1234587699
ERDTFGVBHJNKKK
ERQQQQQQQQQQQQQQQ
ERQQQQQQQQQQQQQ
ABCDASASSASASASA
123456789101112
0000000000000000
```

- output:

```
The string after sort is below:
0000000000000000
123456789101112
ABCDASASSASASASA
ERDTFGVBHJNKKK
ERQQQQQQQQQQQQQ
ERQQQQQQQQQQQQQQ
ITISASADSTOR
iSA1234587699
```



```
ARM7TDMI - Console
How many string you want to input:8
ITISASADSTOR
iSA1234587699
ERDTFGVBHJNKKK
ERQQQQQQQQQQQQQQQ
ERQQQQQQQQQQQQQ
ABCDASASSASASASA
123456789101112
0000000000000000
The string after sort is below:
0000000000000000
123456789101112
ABCDASASSASASASA
ERDTFGVBHJNKKK
ERQQQQQQQQQQQQQ
ERQQQQQQQQQQQQQQ
ITISASADSTOR
iSA1234587699
```

- 符合预期

五、性能分析

- 因为采用冒泡排序 时间复杂度为 $O(n^2)$
- 大多数时间复杂度为 $O(n \lg n)$ 的算法(如堆排序)，在汇编层次上编写过于复杂

六、反思与总结

- 汇编代码的编写难点主要还是集中在循环的跳转时候需要理清各寄存器存储的内容

附录1 main.c

```
1 #include <stdio.h>
2
3 #define MAXNUM 30
4 #define MAXSIZE 30
5
6 extern void compare(int a, int b);
7
8 struct string_sort
9 {
10     int num;
11     int len;
12     char s[MAXNUM][MAXSIZE];
13 };
14
15 extern void sort(struct string_sort *arg1);
16
17 int main(void)
18 {
19     int i;
20     struct string_sort my_strings;
21
22     my_strings.len = 30;
23
24     printf("How many string you want to input:");
25
26     scanf("%d", &my_strings.num);
27     for (i = 0; i < my_strings.num; i++)
28     {
29         scanf("%s", my_strings.s[i]);
30     }
31
32     sort(&my_strings);
33
34     printf("The string after sort is below:\n");
35     for (i = 0; i < my_strings.num; i++)
36         printf("%s\n", my_strings.s[i]);
37 }
```

附录二 sort.s

```
1  EXPORT sort
2  IMPORT compare
3  AREA SUMMING, CODE, READONLY
4
5  sort
6      STMFD    SP!, {R4-R5}
7      ldr      r1, [r0]           ;r1=num
8      ldr      r2, [r0,#4]       ;r2=len
9      add      r3, r0, #8        ;将r3指向s二维数组
10
11     mov      r4, #0
12     sub      r4, r4, #1        ;i=-1
13     sub      r1, r1, #1        ;r1=num-1
14
15  loop1                                ;冒泡排序
16     add      r4, r4, #1        ;i++
17     cmp      r1, r4
18     ble      back             ;if num-1<=i 返回
19     mov      r5, #0           ;j=0
20     sub      r6, r1, r4       ;r6=num-1-i
21
22  loop2                                ;if num-1-i<=j 跳出第一重循环
23     cmp      r6, r5
24     ble      loop1
25
26     STMFD    SP!, {R0-R6, lr}
27     mul      r0, r2, r5
28     add      r0, r0, r3       ;r0 指向第一个字符串
29     add      r1, r0, r2       ;r1 指向第二个字符串
30     bl       compare
31     ldmfdd   SP!, {R0-R6, lr}
32
33     add      r5, r5, #1        ;j++
34     b        loop2
35
36  back
37     ldmfdd   SP!, {R4-R5}
38     mov      pc, lr
39  END
```

附录三 compare.s

```
1      EXPORT compare
2      AREA SUMMING, CODE, READONLY
3
4  compare
5      mov     r3, r0
6      mov     r4, r1
7      mov     r7, #1          ;i=1
8  loop
9      cmp     r7, r2
10     bgt     end
11     ldrb     r5, [r3]
12     ldrb     r6, [r4]
13     cmp     r5, r6
14     blt     end              ;s1<s2
15     bgt     exchange        ;s1>s2
16     add     r3, r3, #1
17     add     r4, r4, #1
18     add     r7, r7, #1
19     b       loop
20
21
22
23  exchange
24     mov     r7, #1          ;i=1
25     mov     r3, r0
26     mov     r4, r1
27
28  loop_exchange
29     cmp     r7, r2
30     bgt     end
31     ldrb     r5, [r3]
32     ldrb     r6, [r4]
33     strb     r5, [r4]
34     strb     r6, [r3]
35     add     r3, r3, #1
36     add     r4, r4, #1
37     add     r7, r7, #1
38     b       loop_exchange
39
40  end
41     mov     pc, lr
42     END
```