# 中国科学技术大学计算机学院
# 《计算机系统概论》实验报告


## LAB 03: The Linked-List Sort

姓名:王晨 学号:PE20060014

完成日期:2020 年 12 月 21 日

# 一、实验要求：

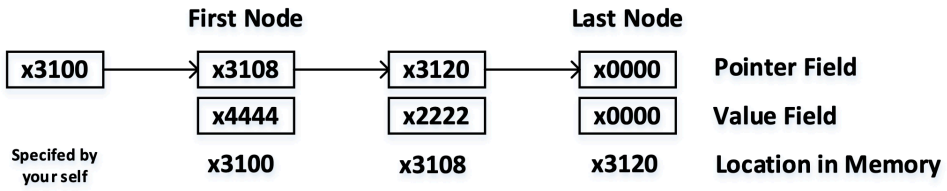
Write a program in LC-3 assembly language that sorts a linked list of 2's complement integers in ascending order.

## *Details:*

1. This program is going to make use of a common data structure, **the linked list**.
2. The linked list contains a value field and a pointer filed. The value filed stores the **VALUE** of the current node, and the pointer filed stores the **ADDRESS** of the next node.
3. The addresses of pointer field and value field are **continuously**.
4. The address of the first node of the linked list and **WHERE TO STORE IT** should be specified by yourself, and the pointer filed of the last node should be **x0000** to indicate the end of a linked list. Your program should identify the last node and finish the sort process.
5. Your program should start at memory location **x3000** and end with **HALT**.
6. Test your program using linked lists of different length.

## *Example:*

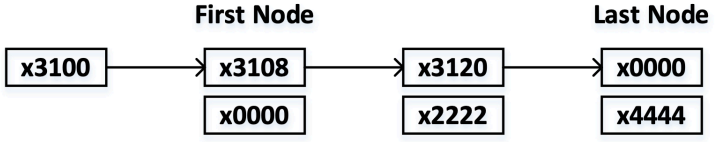
Original linked list:




Please note:

1. **x3100** is the address of the first node.
2. **x3108** is the address of the second node and **x4444** is the value of the first node.
3. **x3108** is stored in address **x3100** and **x4444** is stored in address **x3101**

Sorted linked list:




## *Notes and Suggestions:*

1. You can use any algorithms (bubble sort, select sort and so on) you want to sort the linked list.
2. Load the list into memory before your program begins to run.
3. You can swap two nodes in linked list or just swap the value field. Clarify it in your report.

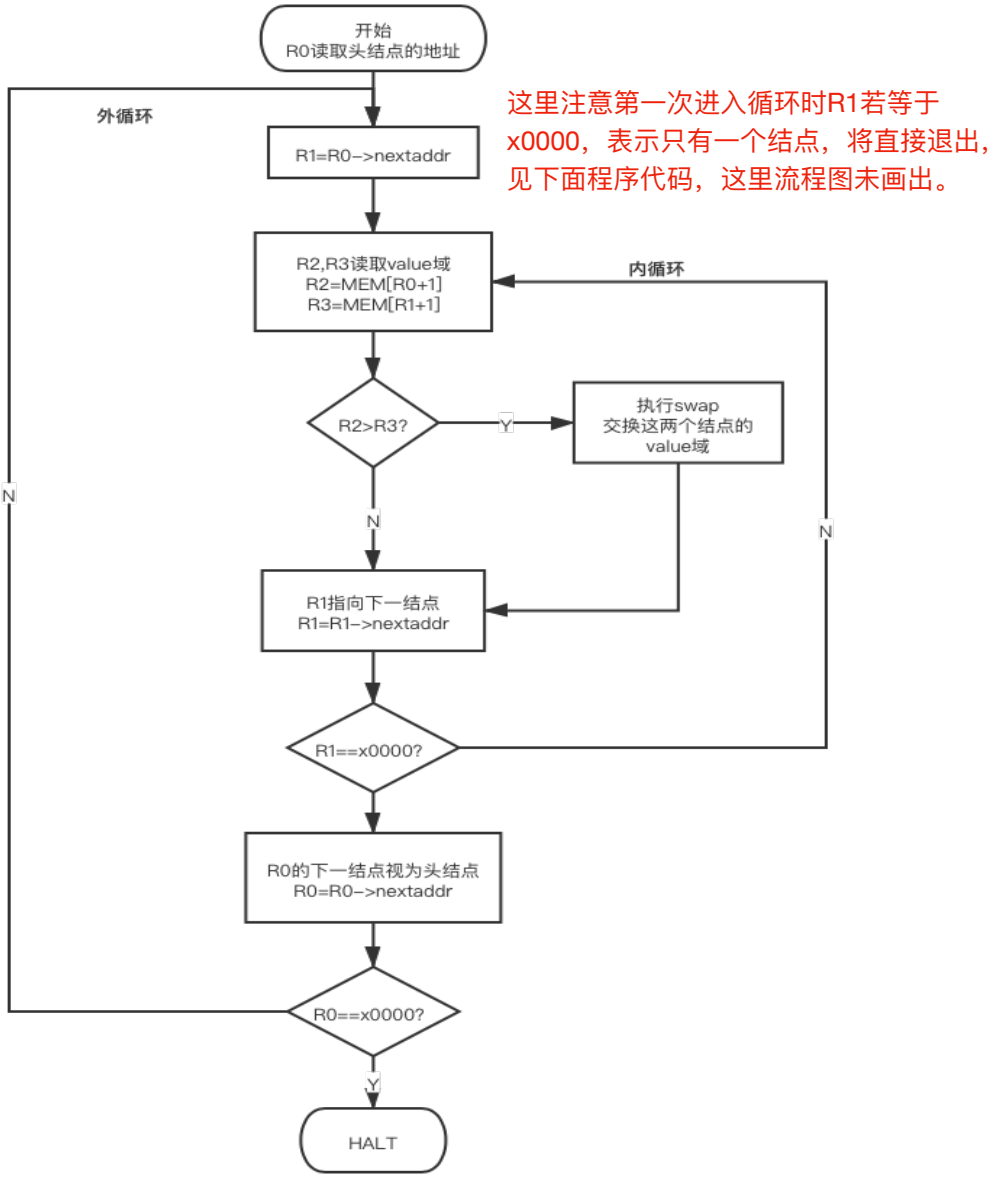
# 二、实验环境：

Mac OS Big Sur 11.01

LC-3 Simulator Mac Version

# 三、算法思路：

采用冒泡排序算法进行排序，两层循环，内循环执行比较和交换功能，外循环执行找到第 i 小的元素功能，外循环每执行一次可以使得第 i 个元素比它后面的元素都小，从而完成升序排序。本算法仅交换value域，首结点地址为x3100，尾结点的address域为x0000。下图是算法流程图：



寄存器说明： R0指向第 i 次要确认的最小元素的位置，也就是第 i 小元素的地址。

R1指向第 j 个（ j>i ）元素的地址。

R2，R3分别读取R0，R1的value域。

每一次比较和交换结束后，R1=R1->nextaddr，即R1指向下一个元素位置，再同R0位置的元素进行比较。当R1=x0000时，说明一轮循环结束，R0位置的元素已经是最小的值了，然后将R0=R0->nextaddr 继续依次比较后面的元素，直到最终R0=x0000完成排序。

# 四、程序代码及注释：

sort.asm

```
;冒泡排序，两层循环，第i次外循环结束，使得链表的第i个元素比它之后的元素都要小（只交换value域）。
            .ORIG x3000
            LD R0 FirstAddr
            LDR R1,R0,#0        ;检查R0指针域为x0000直接退出
            BRz Exit

IntLoop     LDR R1,R0,#0        ;内循环开始，R1=R0->nextaddr
Load        LDR R2,R0,#1        ;R2取数据1
            LDR R3,R1,#1        ;R3取数据2,下面开始比较这两个数据
            NOT R4,R3
            ADD R4,R4,#1
            ADD R4,R2,R4
            BRp SWAP            ;若R2大于R3,交换value field
Nextaddr    LDR R1,R1,#0        ;否则 R1=R1->nextaddr
            BRz ExtLoop         ;若R1=x0000，一次内循环结束，跳转到外循环
            BRnzp Load

SWAP        STR R3,R0,#1
            STR R2,R1,#1
            BRnzp Nextaddr

ExtLoop     LDR R0,R0,#0        ;R0=R0->nextaddr
            BRp IntLoop         ;若R0=x0000,外循环结束
Exit        HALT

FirstAddr   .FILL x3100
            .END
```

1. ;冒泡排序，两层循环，第i次外循环结束，使得链表的第i个元素比它之后的元素都要小（只交换value域）。
2. .ORIG x3000
3. LD R0 FirstAddr
4. LDR R1,R0,#0 ;检查R0指针域为x0000直接退出
5. BRz Exit
6. 
7. IntLoop LDR R1,R0,#0 ;内循环开始，R1=R0->nextaddr
8. BRz Exit ;只有一个结点的链表将直接退出
9. Load LDR R2,R0,#1 ;R2取数据1
10. LDR R3,R1,#1 ;R3取数据2,下面开始比较这两个数据
11. NOT R4,R3
12. ADD R4,R4,#1

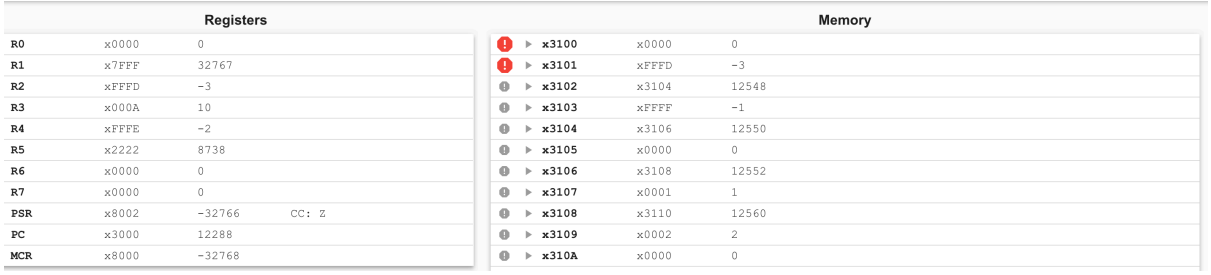
```
13.                ADD R4,R2,R4
14.                BRp SWAP            ;若R2大于R3,交换value field
15. Nextaddr   LDR R1,R1,#0        ;否则 R1=R1->nextaddr
16.                BRz ExtLoop         ;若R1=x0000，内循环结束，跳转到外循环
17.                BRnzp Load
18.
19. SWAP       STR R3,R0,#1
20.                STR R2,R1,#1
21.                BRnzp Nextaddr
22.
23. ExtLoop    LDR R0,R0,#0        ;R0=R0->nextaddr
24.                BRp IntLoop         ;若R0=x0000,外循环结束
25. Exit           HALT
26.
27. FirstAddr  .FILL     x3100
28.                .END
```

# 五、调试和测试：

测试不同长度且包含不同大小的正数、负数、零的链表。为了便于检查，链表结点都放在一页Memory中，红色❗表示链表结点位置。

① 链表长度为1，即x3100位置的指针域为x0000，只有一个结点，此时应该不执行任何操作直接退出。

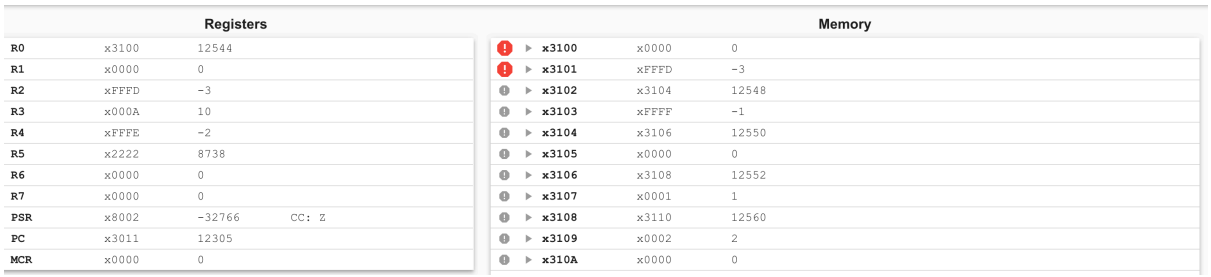
执行前：



| Registers | | | |
|---|---|---|---|
| R0 | x0000 | 0 | |
| R1 | x7FFF | 32767 | |
| R2 | xFFFD | -3 | |
| R3 | x000A | 10 | |
| R4 | xFFFE | -2 | |
| R5 | x2222 | 8738 | |
| R6 | x0000 | 0 | |
| R7 | x0000 | 0 | |
| PSR | x8002 | -32766 | CC: Z |
| PC | x3000 | 12288 | |
| MCR | x8000 | -32768 | |

| Memory | | | |
|---|---|---|---|
| x3100 | x0000 | 0 | |
| x3101 | xFFFD | -3 | |
| x3102 | x3104 | 12548 | |
| x3103 | xFFFF | -1 | |
| x3104 | x3106 | 12550 | |
| x3105 | x0000 | 0 | |
| x3106 | x3108 | 12552 | |
| x3107 | x0001 | 1 | |
| x3108 | x3110 | 12560 | |
| x3109 | x0002 | 2 | |
| x310A | x0000 | 0 | |

执行后，可以看到程序在读到R1=x0000后直接HALT，链表不发生改变：



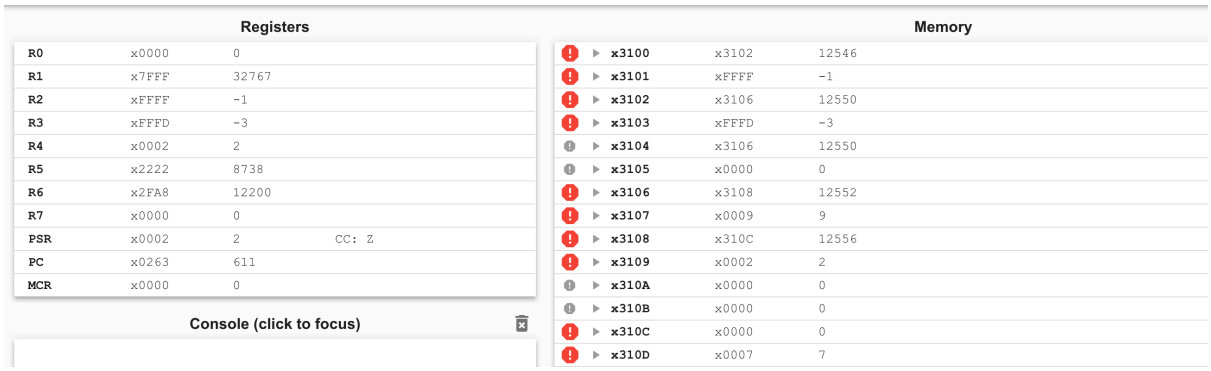
Registers

| | | | |
|---|---|---|---|
| R0 | x3100 | 12544 | |
| R1 | x0000 | 0 | |
| R2 | xFFFD | -3 | |
| R3 | x000A | 10 | |
| R4 | xFFFE | -2 | |
| R5 | x2222 | 8738 | |
| R6 | x0000 | 0 | |
| R7 | x0000 | 0 | |
| PSR | x8002 | -32766 | CC: Z |
| PC | x3011 | 12305 | |
| MCR | x0000 | 0 | |

Memory

| | | |
|---|---|---|
| x3100 | x0000 | 0 |
| x3101 | xFFFD | -3 |
| x3102 | x3104 | 12548 |
| x3103 | xFFFF | -1 |
| x3104 | x3106 | 12550 |
| x3105 | x0000 | 0 |
| x3106 | x3108 | 12552 |
| x3107 | x0001 | 1 |
| x3108 | x3110 | 12560 |
| x3109 | x0002 | 2 |
| x310A | x0000 | 0 |

② 链表长度为5，执行前value值分别为-1，-3，9，2，7（decimal）：



Registers

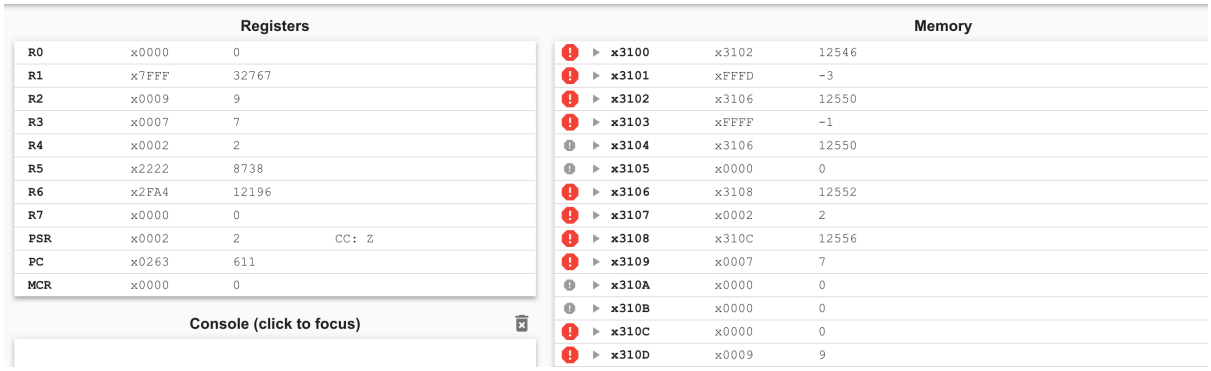
| | | | |
|---|---|---|---|
| R0 | x0000 | 0 | |
| R1 | x7FFF | 32767 | |
| R2 | xFFFF | -1 | |
| R3 | xFFFD | -3 | |
| R4 | x0002 | 2 | |
| R5 | x2222 | 8738 | |
| R6 | x2FA8 | 12200 | |
| R7 | x0000 | 0 | |
| PSR | x0002 | 2 | CC: Z |
| PC | x0263 | 611 | |
| MCR | x0000 | 0 | |

Memory

| | | |
|---|---|---|
| x3100 | x3102 | 12546 |
| x3101 | xFFFF | -1 |
| x3102 | x3106 | 12550 |
| x3103 | xFFFD | -3 |
| x3104 | x3106 | 12550 |
| x3105 | x0000 | 0 |
| x3106 | x3108 | 12552 |
| x3107 | x0009 | 9 |
| x3108 | x310C | 12556 |
| x3109 | x0002 | 2 |
| x310A | x0000 | 0 |
| x310B | x0000 | 0 |
| x310C | x0000 | 0 |
| x310D | x0007 | 7 |

Console (click to focus)

执行后链表状态如下，可以看到value值分别为-3，-1，2，7，9，完成了升序排列：



Registers

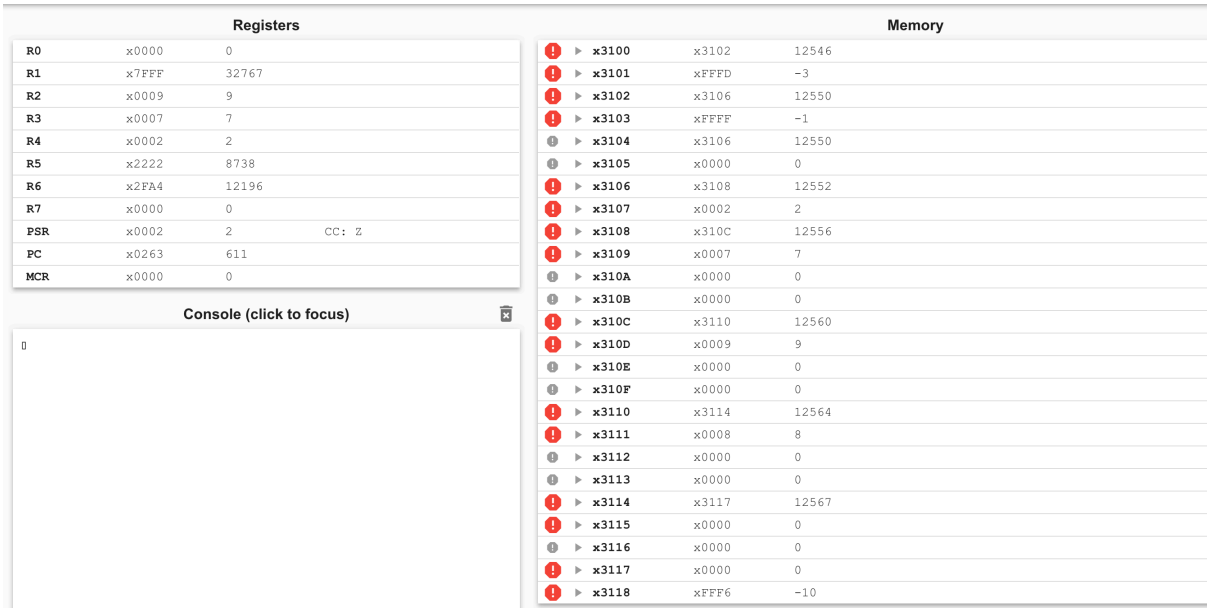
| | | | |
|---|---|---|---|
| R0 | x0000 | 0 | |
| R1 | x7FFF | 32767 | |
| R2 | x0009 | 9 | |
| R3 | x0007 | 7 | |
| R4 | x0002 | 2 | |
| R5 | x2222 | 8738 | |
| R6 | x2FA4 | 12196 | |
| R7 | x0000 | 0 | |
| PSR | x0002 | 2 | CC: Z |
| PC | x0263 | 611 | |
| MCR | x0000 | 0 | |

Memory

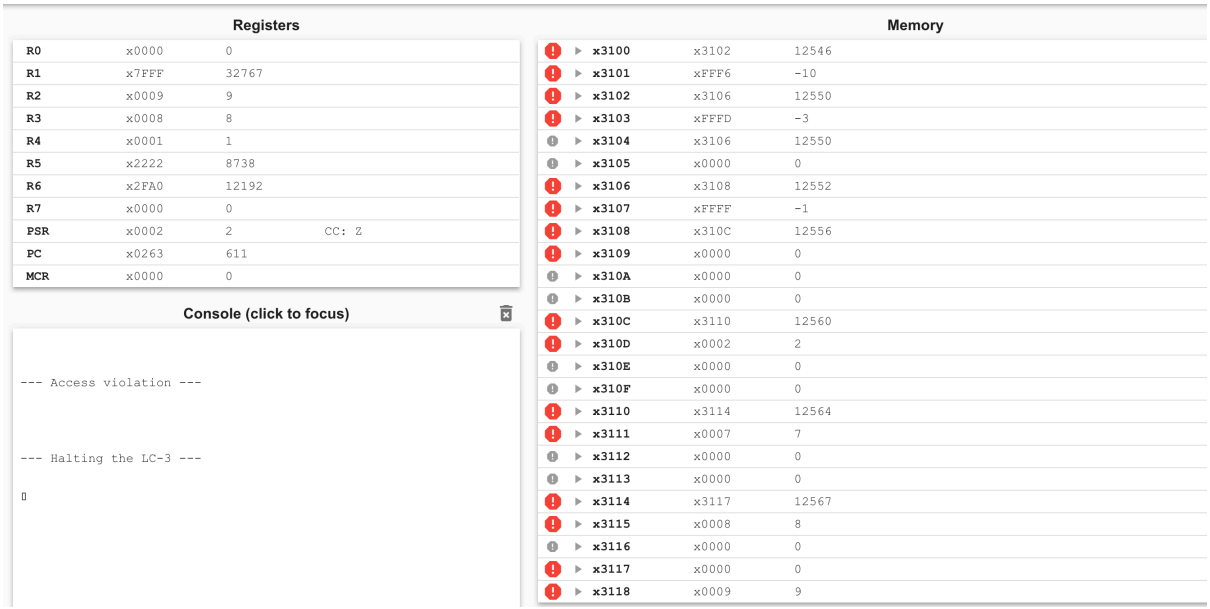
| | | |
|---|---|---|
| x3100 | x3102 | 12546 |
| x3101 | xFFFD | -3 |
| x3102 | x3106 | 12550 |
| x3103 | xFFFF | -1 |
| x3104 | x3106 | 12550 |
| x3105 | x0000 | 0 |
| x3106 | x3108 | 12552 |
| x3107 | x0002 | 2 |
| x3108 | x310C | 12556 |
| x3109 | x0007 | 7 |
| x310A | x0000 | 0 |
| x310B | x0000 | 0 |
| x310C | x0000 | 0 |
| x310D | x0009 | 9 |

Console (click to focus)

③ 链表长度为8，执行前value值分别为-3，-1，2，7，9，8，0，-10：




执行后链表状态如下，可以看到value值分别为-10，-3，-1，0，2，7，9，完成了升序排列：




④ 链表长度为15，执行前value值分别为-3，-1，2，7，9，8，0，-10，-20，21，2，4，88，16，-9。

执行后得到value序列-20，-10，-9，-3，-1，0，2，2，4，7，8，9，16，21，88，完成了升序排列。由于链表在一页中表示不下，就不截图展示了。

# 六、分析和改进：

本方法采用冒泡排序，对于N个结点的链表来说，时间复杂度为O(N)，空间复杂度为O(1)。这种方法的优点是程序思路非常简单，对于不太长的链表（指数级以下）可以很好的完成排序，且基本不需要额外空间暂存排序过程中的变量。该排序算法是稳定的。

主循环体指令条数为15条，整个程序使用寄存器R0～R4，可以说程序是非常简洁的。

采用其他排序算法，可以获得更佳的时间复杂度，但是程序写起来会相对更加复杂些，尤其是在寄存器的使用上，这是由于其他更快的算法的空间复杂度可能会更高些。

# 七、实验心得：

本实验完成了对链表的排序功能，实验成功。在完成本实验时，建议将整个程序分为几个功能模块，例如：读取地址和值，比较，SWAP，排序。排序若采用冒泡排序又可以分为两个循环体，之后画出流程图后，就可以非常快速的写完程序。由于本实验算法难度不大，写完后调试一下，关注各个寄存器的运行状态后基本就不会出问题。