链表排序

1、实验题目

Your Mission: 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

2、实验思路

首先构建链表，之后采用冒泡排序算法对其进行排序。

冒泡链表排序法的主要思想是，设置一个尾指针（假设为R1），从头结点开始扫描，如果当前结点（R2) 的value大于下一个value，则交换二者的value值，直到当前节点等于尾节点为止，当尾结点等于头结点时结束。

代码解释分析

首先初始化一些寄存器的值

```
.ORIG X3000
AND R0,R0,#0
AND R1,R1,#0
AND R2,R2,#0
AND R3,R3,#0;表示前一个的value
AND R4,R4,#0;表示后一个的value
AND R5,R5,#0
AND R6,R6,#0
AND R7,R7,#0
LD R0,START
LD R1,FINAL
LDR R0,R0,#0
```

在初始时刻开始部分位置是x3100，结束部分的位置在x0000。

```
JUDGE1:
       AND R5,R5,#0
       LDR R5,R0,#0
       NOT R2,R5
       ADD R2,R2,#1
       ADD R2,R2,R1
       BRz THEEND1
```

在这个JUDGE1块中，程序判断R0的下一个结点是否等于R1，如果相等，则跳出循环，进入THEEND1块，否则进入BUBBLE循环。这里同时考虑了负数的情况，如果链表中的数据存在负数的情况，如果R3的值对应为负数，继续查看R4的值，如果也是负数，则将其都翻转为正数后比较，如果R3的相反数小于R4的相反数，那么执行SWAP交换。如果R3的值为正数，对R4进行判断，如果R4是负数，那么执行SWAP交换，否则正常操作。

```
BUBBLE: ADD R6,R0,#1
        LDR R3,R6,#0
        BRn NEGA3
NEGABACK1:
        LDR R6,R0,#0
        LDR R4,R6,#1
        BRn SWAP
NEGABACK2:
        NOT R5,R3
        ADD R5,R5,#1
        ADD R5,R5,R4
        BRn SWAP
        BRnzp NEXT

NEGA3:
        LDR R6,R0,#0
        LDR R4,R6,#1
        BRn NEGA4
        BRnzp NEXT
NEGA4:
        NOT R4,R4
        ADD R4,R4,#1
        NOT R3,R3
        ADD R3,R3,#1
        NOT R5,R3
        ADD R5,R5,#1
        ADD R5,R5,R4
        BRp SWAPNEGA
        BRnzp NEXT

SWAPNEGA:
        NOT R4,R4
        ADD R4,R4,#1
        NOT R3,R3
        ADD R3,R3,#1
        BRnzp SWAP
SWAP: STR R4,R0,#1
      LDR R7,R0,#0
```

```
       STR R3,R7,#1
       BRnzp NEXT

NEXT: LDR R0,R0,#0
       BRnzp JUDGE1
```

在BUBBLE循环中，取出R0对应的value值放入R3中，并取出R0所存的地址对应的value放入R4中，判断R3和R4的大小，如果R3>R4，则交换这两个值的位置（跳入SWAP块），将R4的值存入R0的下一个地址位置，将R3存入R0中存的地址对应的value的位置。进入NEXT模块之后，将R0的值改变为R0所存的地址的值，即进入下一个结点，再次进入JUDHE1模块，开始循环。

```
THEEND1: ADD R1,R0,#0
         LD R0,START
         LDR R0,R0,#0
         BRnzp JUDGE2
```

如果R0的下一个结点是R1,则这层循环结束，将R1的值换成它的前一个结点，即R0的值，进入下一个循环。

```
JUDGE2:  AND R5,R5,#0
         AND R2,R2,#0
         LD R5,START
         LDR R5,R5,#0
         NOT R2,R1
         ADD R2,R2,#1
         ADD R2,R5,R2
         BRz THEEND2
         BRnzp JUDGE1
THEEND2: HALT
```

如果R1等于初始结点的下一个结点，循环结束，否则，继续JUDGE1开始的循环。

3、测试样例

【测试样例1】

数据初始化情况如下：

		Memory
x3100	x310D	12557
x3101	x0000	0
x3102	x3112	12562
x3103	x0143	323
x3104	x0000	0
x3105	x0000	0
x3106	x0000	0
x3107	x0000	0
x3108	x0000	0
x3109	x0003	3
x310A	x0000	0
x310B	x0000	0
x310C	x0000	0
x310D	x3102	12546
x310E	x0157	343
x310F	x0000	0
x3110	x0000	0
x3111	x0000	0
x3112	x3108	12552
x3113	x0039	57
x3114	x0000	0
x3115	x0000	0
x3116	x0000	0

运行结果为为：

▶	x3100	x310D	12557
▶	x3101	x0000	0
▶	x3102	x3112	12562
▶	x3103	x0039	57
▶	x3104	x0000	0
▶	x3105	x0000	0
▶	x3106	x0000	0
▶	x3107	x0000	0
▶	x3108	x0000	0
▶	x3109	x0157	343
▶	x310A	x0000	0
▶	x310B	x0000	0
▶	x310C	x0000	0
▶	x310D	x3102	12546
▶	x310E	x0003	3
▶	x310F	x0000	0
▶	x3110	x0000	0
▶	x3111	x0000	0
▶	x3112	x3108	12552
▶	x3113	x0143	323
▶	x3114	x0000	0
▶	x3115	x0000	0
▶	x3116	x0000	0

可以看到测试成功，该链表的值升序排列。

【测试用例2】如果存在负数的情况

x3100	x310D	12557
x3101	x0000	0
x3102	x3112	12562
x3103	xFFF9	65529
x3104	x0000	0
x3105	x0000	0
x3106	x0000	0
x3107	x0000	0
x3108	x0000	0
x3109	x0039	57
x310A	x0000	0
x310B	x0000	0
x310C	x0000	0
x310D	x3102	12546
x310E	x0024	36
x310F	x0000	0
x3110	x0000	0
x3111	x0000	0
x3112	x3108	12552
x3113	x0159	345
x3114	x0000	0
x3115	x0000	0
x3116	x0000	0

测试结果：

▶	**x3100**	x310D	12557
▶	**x3101**	x0000	0
▶	**x3102**	x3112	12562
▶	**x3103**	x0024	36
▶	**x3104**	x0000	0
▶	**x3105**	x0000	0
▶	**x3106**	x0000	0
▶	**x3107**	x0000	0
▶	**x3108**	x0000	0
▶	**x3109**	x0159	345
▶	**x310A**	x0000	0
▶	**x310B**	x0000	0
▶	**x310C**	x0000	0
▶	**x310D**	x3102	12546
▶	**x310E**	xFFF9	65529
▶	**x310F**	x0000	0
▶	**x3110**	x0000	0
▶	**x3111**	x0000	0
▶	**x3112**	x3108	12552
▶	**x3113**	x0039	57
▶	**x3114**	x0000	0
▶	**x3115**	x0000	0
▶	**x3116**	x0000	0

由测试结果可以看到，测试成功。

4、源代码

```
.ORIG X3000
AND R0,R0,#0
AND R1,R1,#0
AND R2,R2,#0
AND R3,R3,#0;表示前一个的value
AND R4,R4,#0;表示后一个的value
AND R5,R5,#0
```

```
AND R6,R6,#0
AND R7,R7,#0
LD R0,START
LD R1,FINAL
LDR R0,R0,#0
;判断头结点是不是等于尾结点，否则调到BUBBLE
JUDGE1:
       AND R5,R5,#0
       LDR R5,R0,#0
       NOT R2,R5
       ADD R2,R2,#1
       ADD R2,R2,R1
       BRz THEEND1

BUBBLE: ADD R6,R0,#1
        LDR R3,R6,#0
        BRn NEGA3
NEGABACK1:
        LDR R6,R0,#0
        LDR R4,R6,#1
        BRn SWAP
NEGABACK2:
        NOT R5,R3
        ADD R5,R5,#1
        ADD R5,R5,R4
        BRn SWAP
        BRnzp NEXT

NEGA3:
        LDR R6,R0,#0
        LDR R4,R6,#1
        BRn NEGA4
        BRnzp NEXT
NEGA4:
        NOT R4,R4
        ADD R4,R4,#1
        NOT R3,R3
        ADD R3,R3,#1
        NOT R5,R3
        ADD R5,R5,#1
        ADD R5,R5,R4
        BRp SWAPNEGA
        BRnzp NEXT

SWAPNEGA:
        NOT R4,R4
        ADD R4,R4,#1
        NOT R3,R3
        ADD R3,R3,#1
        BRnzp SWAP
SWAP: STR R4,R0,#1
      LDR R7,R0,#0
      STR R3,R7,#1
```

```
      BRnzp NEXT

NEXT: LDR R0,R0,#0
      BRnzp JUDGE1
;内层循环结束，重新配置R1的值，将R0的值赋值给R1
THEEND1: ADD R1,R0,#0
         LD R0,START
         LDR R0,R0,#0
         BRnzp JUDGE2
JUDGE2:  AND R5,R5,#0
         AND R2,R2,#0
         LD R5,START
         LDR R5,R5,#0
         NOT R2,R1
         ADD R2,R2,#1
         ADD R2,R5,R2
         BRz THEEND2
         BRnzp JUDGE1
THEEND2: HALT

START .FILL x3100
FINAL .FILL x0000

.END
```