REPORT LAB03

PB19010450 和泳毅

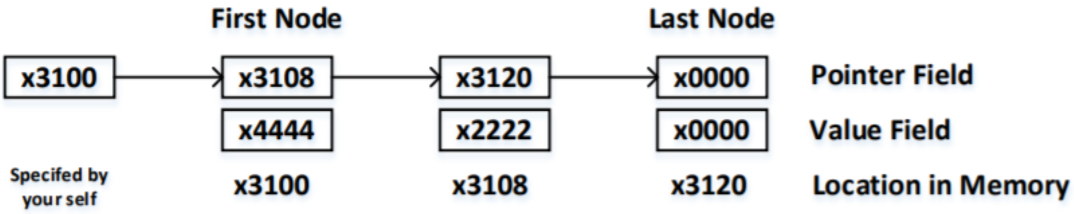
Requirements

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

- This program is going to make use of a common data structure, **the linked list**;
- 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;
- The addresses of pointer field and value field are **continuously**;
- 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`*;
- Your program should start at memory location x3000 and end with HALT;
- Test your program using linked lists of different length.

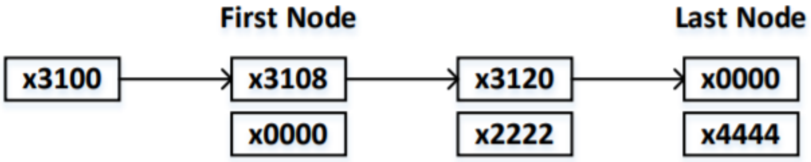
Example:

Original linked list:



- **x3100** is the address of the first node;
- **x3108** is the address of the second node and **x4444** is the value of the first node;
- **x3108** is stored in address **x3100** and **x4444** is stored in address **x3101**.

Sorted linked list:



Design

> **Selection Sort Algorithm:**
>
> Among the VALUEs to be sorted, select the smallest VALUE and exchange it with the VALUE of the first node. Among the remaining VALUEs, find the smallest VALUE and exchange it with the VALUE of the second node, i.e., put it at the end of the ordered series, and so on, until all the nodes are sorted.

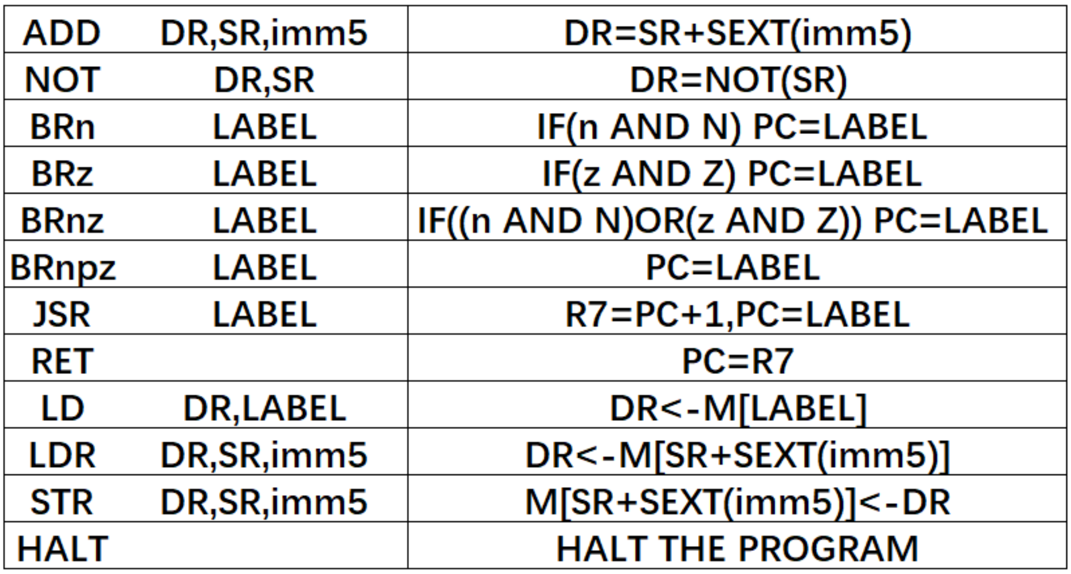
Note：We only swap the **Value Field** of the nodes, making the final node data in **ascending order**.

Detailed Design：

1. We use two levels of loops. The outer loop is the nodes that are exchanged after each comparison trip, and the inner loop is the nodes that need to be traversed during each comparison trip；
2. We first store the **Value Field** and **Pointer Field** of the first node, and in the other registers traverse from the **Value Field** of that node to the **Value Field** of the last node.
3. When traversing we compare the node with the **Value Field** of each node and execute a **SWAP** if we encounter a value smaller than ourselves. **SWAP** means writing the larger value to the memory where the data for the smaller value is stored and assigning the smaller value to the register where the larger value is stored.
4. The outer loop pointer points to the next node, repeating 2 to 3 steps until the end (The last node **Pointer Field** points to **x0000**).

Code Writing

1. Instructions to be used



ADD DR,SR,imm5	DR=SR+SEXT(imm5)
NOT DR,SR	DR=NOT(SR)
BRn LABEL	IF(n AND N) PC=LABEL
BRz LABEL	IF(z AND Z) PC=LABEL
BRnz LABEL	IF((n AND N)OR(z AND Z)) PC=LABEL
BRnpz LABEL	PC=LABEL
JSR LABEL	R7=PC+1,PC=LABEL
RET	PC=R7
LD DR,LABEL	DR<-M[LABEL]
LDR DR,SR,imm5	DR<-M[SR+SEXT(imm5)]
STR DR,SR,imm5	M[SR+SEXT(imm5)]<-DR
HALT	HALT THE PROGRAM

2. Start at memory location **x3000**

```
START   .ORIG   X3000
```

3. The outer loop

```
        LD      R0,START;START is the address of the first node
BEG     LDR     R2,R0,#1
        ADD     R3,R2,#0
        ADD     R1,R0,#0
```

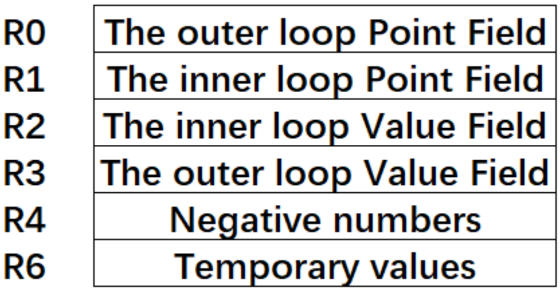
4. The inner loop

```
SORT    LDR     R1,R1,#0
        BRz    THEN2
        LDR     R2,R1,#1
        NOT     R4,R2
        ADD     R4,R4,#1
        ADD     R6,R3,R4
        BRn     THEN1
        JSR     SWAP
THEN1   BRnzp   SORT
THEN2   STR     R3,R0,#1
        LDR     R0,R0,#0
        BRnp    BEG
        HALT
```

5. Swap function

```
SWAP    STR     R3,R1,#1
        ADD     R3,R2,#0
        RET
```

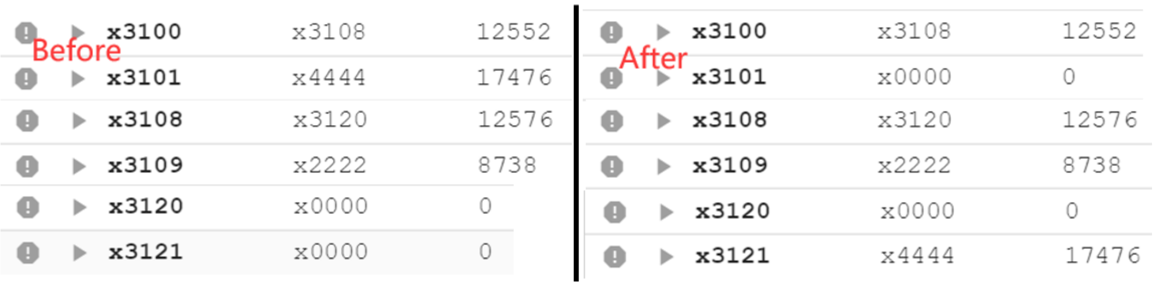
6. Register Usage



R0	The outer loop Point Field
R1	The inner loop Point Field
R2	The inner loop Value Field
R3	The outer loop Value Field
R4	Negative numbers
R6	Temporary values

Result Test

1.



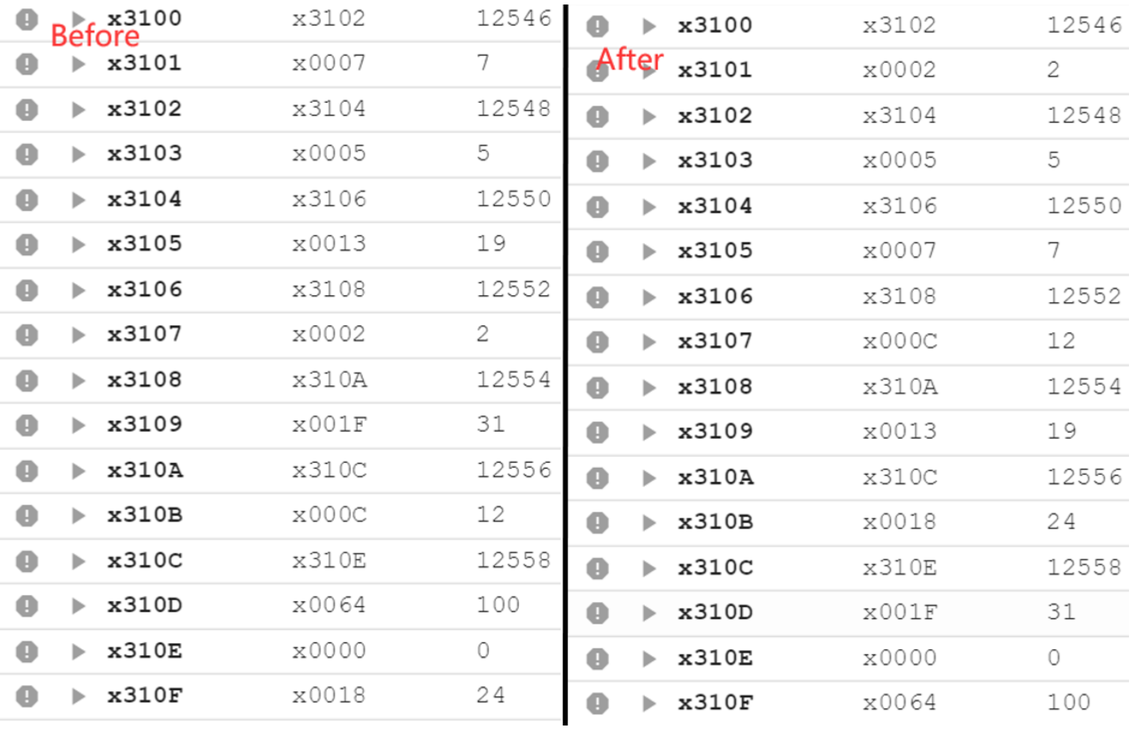
Before

```
x3100    x3108    12552
x3101    x4444    17476
x3108    x3120    12576
x3109    x2222    8738
x3120    x0000    0
x3121    x0000    0
```

After

```
x3100    x3108    12552
x3101    x0000    0
x3108    x3120    12576
x3109    x2222    8738
x3120    x0000    0
x3121    x4444    17476
```

2.



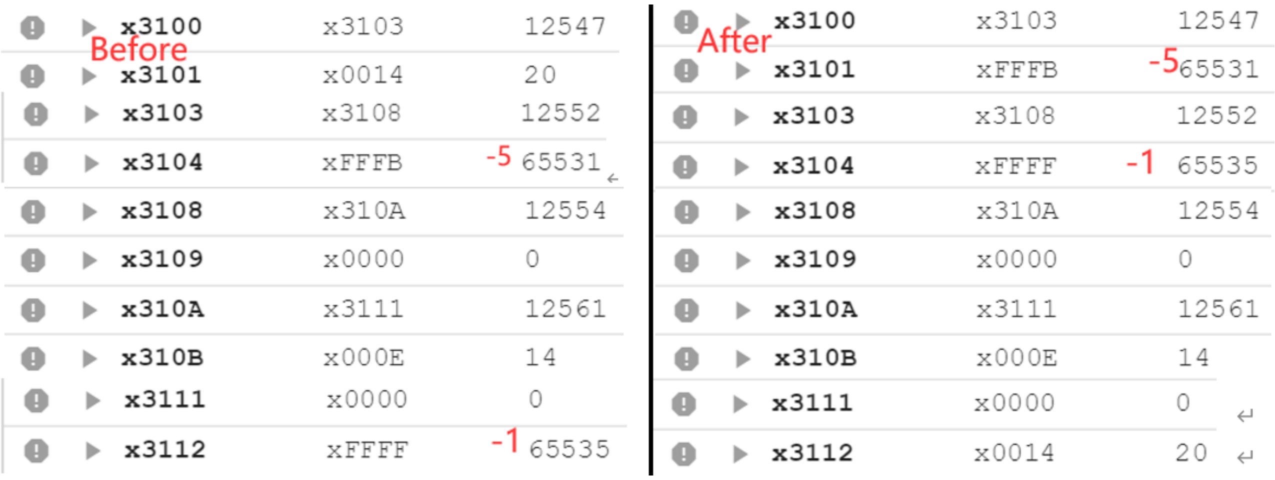
Before

x3100	x3102	12546
x3101	x0007	7
x3102	x3104	12548
x3103	x0005	5
x3104	x3106	12550
x3105	x0013	19
x3106	x3108	12552
x3107	x0002	2
x3108	x310A	12554
x3109	x001F	31
x310A	x310C	12556
x310B	x000C	12
x310C	x310E	12558
x310D	x0064	100
x310E	x0000	0
x310F	x0018	24

After

x3100	x3102	12546
x3101	x0002	2
x3102	x3104	12548
x3103	x0005	5
x3104	x3106	12550
x3105	x0007	7
x3106	x3108	12552
x3107	x000C	12
x3108	x310A	12554
x3109	x0013	19
x310A	x310C	12556
x310B	x0018	24
x310C	x310E	12558
x310D	x001F	31
x310E	x0000	0
x310F	x0064	100

3.



Before

x3100	x3103	12547
x3101	x0014	20
x3103	x3108	12552
x3104	xFFFB	-5 65531
x3108	x310A	12554
x3109	x0000	0
x310A	x3111	12561
x310B	x000E	14
x3111	x0000	0
x3112	xFFFF	-1 65535

After

x3100	x3103	12547
x3101	xFFFB	-5 65531
x3103	x3108	12552
x3104	xFFFF	-1 65535
x3108	x310A	12554
x3109	x0000	0
x310A	x3111	12561
x310B	x000E	14
x3111	x0000	0
x3112	x0014	20

Thinking

1. HALT operation will make changes to R0 and R1, so test with breakpoints before HALT；
2. Functions that are used repeatedly can be written as sub-code for invocation；
3. We should be very careful with the Pointer Field of the linked list nodes. Be careful not to break the linked list or lose nodes.

Summary

Writing LC-3 programs in assembly language is much simpler than machine language. The functions of assembly language and the objects to be manipulated are clear at a glance. The use of LABLE also makes the program to be versatile, allowing jumps to be made at any location without having to calculate addresses when writing code.And some useful data structure can Simplify lots of problems.

Rewriting in RISC-V Assembly Language

The idea remains the same, using the Euclidean Algorithm. Use **Jupiter**, an open source and education-oriented RISC-V assembler and runtime simulator.

1. Instructions to be used

   - **MV RD RS**

     i.e.**ADD RD RS1 0**

     Achieve copy function by addition operation， x[RD] = x[RS1] + 0

   - **BEQ RS1,RS2,LABLE**

     Branch, if (RS1 == RS2) pc = LABEL

   - **BGE RS1,RS2,LABLE**

     Branch, if (RS1 >= RS2) pc = LABEL

   - **LW RD imm RS**

     RD = M[RS + sext(imm)]

   - **SW RD imm RS**

     M[RS + sext(imm)] = RD

   - **J LABEL**

     Jump to the LABEL

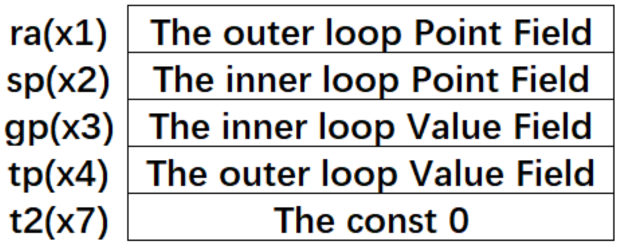
2. Core Code

```

    LI   x1,168     #start
BEG:
    LW  x3,4(x1)
    MV  x4,x3
    MV  x2,x1
SORT:
    LW  x2,0(x2)
    BEQ x2,x7,THEN2
    LW  x3,4(x2)
    BGE x3,x4,THEN1
SWAP:
    SW  x4,4(x2)
    MV  x4,x3
THEN1:
    J   SORT
THEN2:
    SW  x4,4(x1)
```

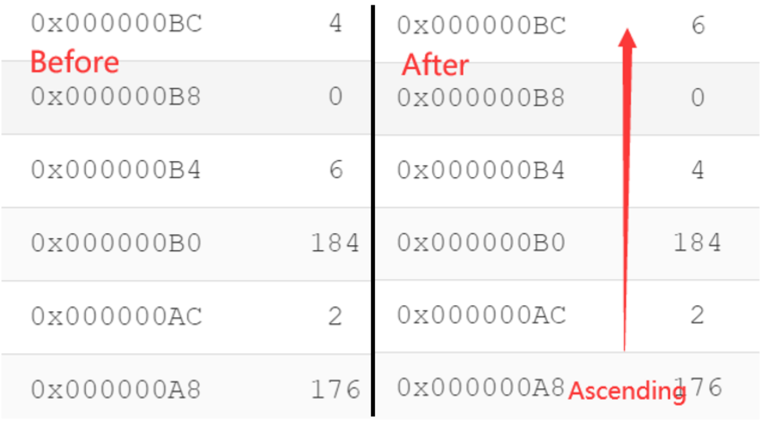
```
LW  x1,0(x1)
BNE x1,x7,BEG
```

3. Register usage

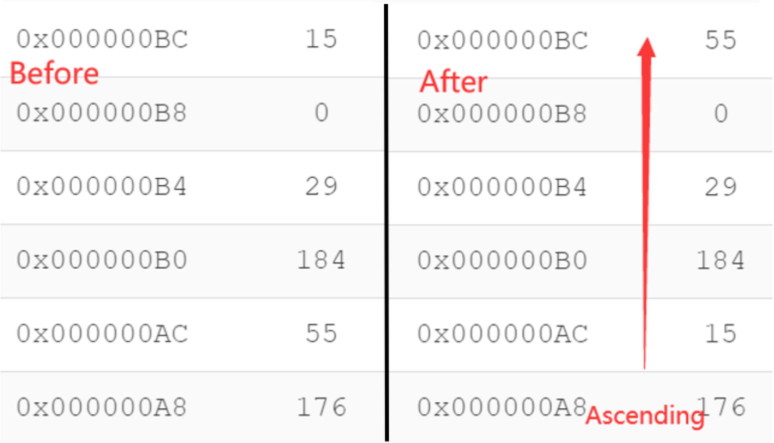


ra(x1)	The outer loop Point Field
sp(x2)	The inner loop Point Field
gp(x3)	The inner loop Value Field
tp(x4)	The outer loop Value Field
t2(x7)	The const 0

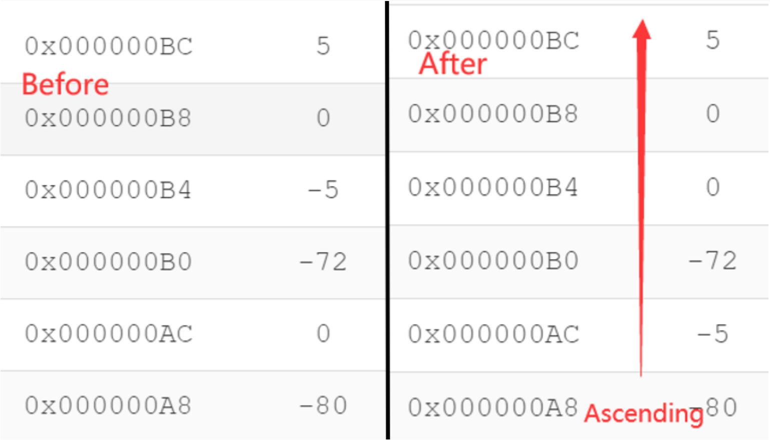
4. Result test

- 

0x000000BC	4	0x000000BC	6
0x000000B8	0	0x000000B8	0
0x000000B4	6	0x000000B4	4
0x000000B0	184	0x000000B0	184
0x000000AC	2	0x000000AC	2
0x000000A8	176	0x000000A8	176

- 

0x000000BC	15	0x000000BC	55
0x000000B8	0	0x000000B8	0
0x000000B4	29	0x000000B4	29
0x000000B0	184	0x000000B0	184
0x000000AC	55	0x000000AC	15
0x000000A8	176	0x000000A8	176

- 

0x000000BC	5	0x000000BC	5
0x000000B8	0	0x000000B8	0
0x000000B4	-5	0x000000B4	0
0x000000B0	-72	0x000000B0	-72
0x000000AC	0	0x000000AC	-5
0x000000A8	-80	0x000000A8	-80

Appendix

Complete code:

LC-3：

```
        .ORIG   x3000
        LD      R0,START
BEG     LDR     R2,R0,#1
        ADD     R3,R2,#0
        ADD     R1,R0,#0
;
SORT    LDR     R1,R1,#0
        BRz    THEN2
        LDR     R2,R1,#1
        NOT     R4,R2
        ADD     R4,R4,#1
        ADD     R6,R3,R4
        BRn     THEN1
        JSR     SWAP
THEN1   BRnzp   SORT
THEN2   STR     R3,R0,#1
        LDR     R0,R0,#0
        BRnp    BEG
        HALT
;
SWAP    STR     R3,R1,#1
        ADD     R3,R2,#0
        RET
START   .FILL   x3100
        .END
```

RISC-V：

```
.globl main
.text
main:
    LI      x1,168     #start
BEG:
    LW  x3,4(x1)
    MV  x4,x3
    MV  x2,x1
SORT:
    LW  x2,0(x2)
    BEQ x2,x7,THEN2
    LW  x3,4(x2)
    BGE x3,x4,THEN1
SWAP:
    SW  x4,4(x2)
    MV  x4,x3
THEN1:
    J   SORT
THEN2:
    SW  x4,4(x1)
    LW  x1,0(x1)
    BNE x1,x7,BEG
    LI      a0, 10
    ECALL
```