Homework 5

October 4, 2025

T1

What is the problem with the following LC-3 program? What may happen when it is executed? Give two possible way to fix the problem.

```
.ORIG x3000
        AND R1, R1, #0
PROGRAM ADD R1, R1, #1
        TRAP x23        ; IN
        LEA R0, MESSAGE
        TRAP x22        ; PUTS
MESSAGE .STRINGZ "Program reporting"
.END                    ; HALT
```

ANSWER

很显然，这个程序缺失了 HALT 指令。本程序执行完后 PC 不会停下，计算机将继续执行后续内存中的指令，直到遇到 HALT 指令或者发生异常为止。由于后续内存中的内容不可预知，可能会导致程序执行意料之外的指令，甚至引发系统崩溃等问题。

解决方法：1) 在.END 之前添加 HALT 指令

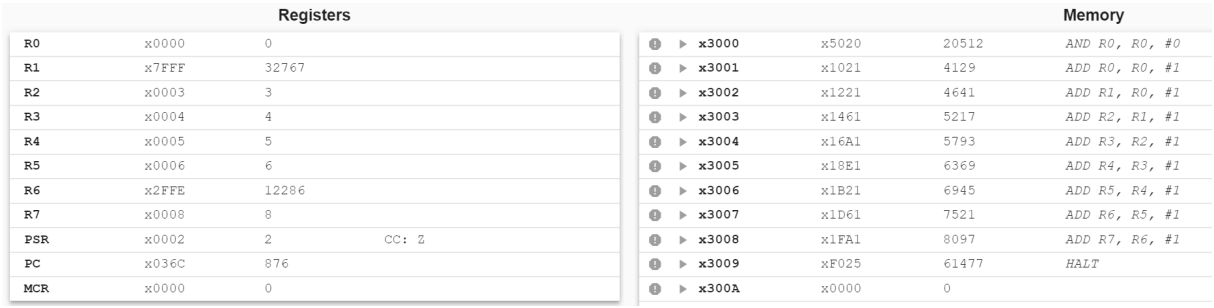
2) 加入一个无限循环（结尾处或者整个程序）来防止程序继续执行不可预知的内存内容。

T2

In LC-3 programming practice, when using LC3Tools for running and debugging programs, you may observe the following phenomenon: after the program executes the `HALT` instruction and stops, the stored values in registers are significantly different from your expectations. Traditionally, we solve this issue by setting breakpoints.

You may take a look at the following screenshot for reference.

1) Now, please explain why this situation occurs.



Registers

R0	x0000	0	
R1	x7FFF	32767	
R2	x0003	3	
R3	x0004	4	
R4	x0005	5	
R5	x0006	6	
R6	x2FFE	12286	
R7	x0008	8	
PSR	x0002	2	CC: Z
PC	x036C	876	
MCR	x0000	0	

Memory

x3000	x5020	20512	AND R0, R0, #0
x3001	x1021	4129	ADD R0, R0, #1
x3002	x1221	4641	ADD R1, R0, #1
x3003	x1461	5217	ADD R2, R1, #1
x3004	x16A1	5793	ADD R3, R2, #1
x3005	x18E1	6369	ADD R4, R3, #1
x3006	x1B21	6945	ADD R5, R4, #1
x3007	x1D61	7521	ADD R6, R5, #1
x3008	x1FA1	8097	ADD R7, R6, #1
x3009	xF025	61477	HALT
x300A	x0000	0	

ANSWER

在执行完 HALT 后，HALT 指令并不会立刻停机，而是会触发一个 Service Routine（服务例程）。这个例程会执行一系列操作，包括显示“HALT”消息、将 MCR（Machine Control Register）最高位置 0 等，最后停止计算机时钟。在这个过程中，寄存器的值可能会被修改。在 LC-3Tools 的实现中，HALT 的 SR 在停机前并不会保存之前寄存器的值。

2) The following shows a fragment of the HALT service routine. Under normal circumstances, will the instruction at address x036C be executed? If not, explain the purpose of this instruction.

Address	Instruction
x0366	*LEA R0, TRAP_HALT_MSG*
x0367	*PUTS*
x0368	*LDI R0, OS_MCR*
x0369	*LD R1, MASK_HI*
x036A	*AND R0, R0, R1*
x036B	*STI R0, OS_MCR*
x036C	*BRnzp TRAP_HALT*

ANSWER

通常情况下，地址 x036C 处的指令不会被执行。因为在执行到 x036B 时，STI 指令已经将 MCR 的最高位置 0，这会停止计算机的时钟，从而使得程序无法继续执行下去。地址 x036C 处的 BRnzp 指令实际上是一个跳转到 TRAP SR 入口处的指令，它的目的是确保即使计算机出现异常没能成功停止时钟时也能防止意外行为。

T*3*

List at least three differences between TRAP and interrupt.

ANSWER

1. TRAP 是由程序显式调用的（程序内部引起），而中断是由外部事件（如 I/O 设备）触发的。

2. TRAP 通常用于请求操作系统服务，而中断用于处理异步事件、硬件请求。

3. TRAP 的触发是同步的，程序执行到 TRAP 指令时才会发生；而中断是异步的，可以在任何时间点发生。

4. TRAP 触发的时机和位置是可控可预测的，而中断的触发时机和位置是不可控不可预测的。

5. TRAP 执行完后会返回到下一条指令，中断处理完后会返回到中断发生前的指令。

6. TRAP 向量表地址在 x0000 到 x00FF 之间，而中断向量表地址在 x0100 到 x01FF 之间。

T4

1) What is system space?
2) What is user space?
3) Which memory addresses are inaccessible in user mode?
4) What is an ACV?
5) What determines the accessible memory range in the current state?
6) List all methods to transition from user mode to privilege mode.

ANSWER

1) 系统空间是操作系统代码和数据所在的内存区域，需要特权，包含系统栈，位于 x0000 到 x2FFF 之间。

2) 用户空间是用户程序代码和数据所在的内存区域，无需特权，包含用户栈，位于 x3000 到 xFFFF 之间。

3) x0000 到 x2FFF 的系统空间、xFE00 到 xFFFF 的内存映射 I/O 区域。

4) ACV（Access Control Violation，访问控制违规）是指程序试图访问其权限范围之外的内存地址时触发的异常。

5) 当前的特权级别（用户模式或特权模式）决定了可访问的内存范围，该级别存储在 PSR（Processor Status Register）中。

6) 通过执行 TRAP 指令、处理硬件中断可以从用户模式切换到特权模式。

T5

The following is an LC-3 program fragment. What is the value of register R0 after executing the code at label C.

Note: you need to show how you arrived at your answer.

```
A   LD R0, E
    LEA R7, E
B   .FILL x1021
    ADD R2, R0, #0
    LD R1, D
    LD R3, B
    ADD R3, R1, R3
```

```
8       ST R3, C
9   C   .BLKW 1
10      RET
11  D   .STRINGZ "!"
12  E   LD R1, A
13  F   ADD R0, R0, R1
```

ANSWER

1. 执行第一条指令 R0=*E* 处内存数据 =x23F3（考察：伪操作预留的内存空间，注意：伪操作本身并不占用内存，但是汇编器在处理伪操作后会预留对应空间，填充相应的内容。）

2. 执行第二条指令 R7=*E Address*

3. 执行第三条指令 x1021(ADD R0, R0, #1) R0=x23F4（考察：.FILL 伪操作填充的内容也算作是指令，计算机内存不对数据和指令进行区分。）

4. 执行第四条指令 R2=R0=x23F4

5. 执行第五条指令 R1=x0021

6. 执行第六条指令 R3=x1021

7. 执行第七条指令 R3=x1042

8. 执行第八条指令将 x1042 存储到标签 C 处的内存中

9. 执行第九条指令 x1042(ADD R0, R1, R2) R0=x2415

10. 执行第十条指令 RET 跳转到 *E Address*

11. 执行第十一条指令 R1=x200B

12. 执行第十二条指令 R0=x2415+x200B=x4420

得到 R0=x4420

T6

During the execution of an LC-3 program, an instruction in the program starts executing at clock cycle T and requires 15 cycles to complete. The table lists ALL five clock cycles during the processing of this instruction, which require use of the bus. The table shows for each of those clock cycles: which clock cycle, the state of the state machine, the value on the bus, and the important control signals that are active during that clock cycle.

1) Fill in the missing entries in the table.

2) What is the instruction being processed?

3) Where in memory is that instruction?

4) How many clock cycles does it take memory to read or write?

5) There is enough information above for you to know the contents of three memory locations. What are they, and what are their contents?

Cycle	State	Bus	ImportantControl Signals for This Cycle
T	18	x3010	LD.MAR=1,LD.PC=1,PCMux=PC+1,GatePC=1
T+4			
T+6		x3013	
T+10		x4567	
T+14		x0000	LD.REG=1,LD.CC=1,GateMDR=1,DR=001

ANSWER

1) 见下表
2) LDI R1, #2
3) x3010
4) 2 个时钟周期
5)

a. x3010: xA202 (LDI R1, #2)

b. x3013: x4567

c. x4567: x0000

从最后一个状态可以很容易看出这是一个内存读取操作，总线访问次数为 5 次，说明是 LDI 指令，剩下的也不难了。掌握 LC-3 的微操作和状态机很重要，务必熟悉。

Cycle	State	Bus	ImportantControl Signals for This Cycle
T	18	x3010	LD.MAR=1,LD.PC=1,PCMux=PC+1,GatePC=1
T+4	30	xA202	Gate.MDR = 1, LD.IR = 1
T+6	10	x3013	ADDR1MUX = PC, ADDR2MUX = SEXT(IR[8:0]), MARMUX = ADDR.ADD, Gate.MARMUX = 1
T+10	26	x4567	
T+14	27	x0000	LD.REG=1,LD.CC=1,GateMDR=1,DR=001

T7

Consider the following program:

```
        .ORIG x3000
        LD R0, A
        LD R1, B
        BRz DONE
        --------- (a)
        --------- (b)
        BRnzp AGAIN
DONE    ST R0, A
        HALT
```

```
A       .FILL x0--- (c)
B       .FILL x0001
        .END
```

The program uses only R0 and R1. Note lines (a) and (b) indicate two missing instructions. Complete line (c). Note also that one of the instructions in the program must be labeled AGAIN, and that label is missing.

After execution of the program, the contents of A is x1800.

Cycle Number	State Number	Control Signals
1	18	LD.MAR: ☐ LD.REG: ☐ GateMDR: ☐ LD.PC: ☐ PCMUX: ☐ GatePC: ☐
☐	0	LD.MAR: ☐ LD.REG: ☐ BEN ☐ LD.PC: ☐ LD.CC: ☐
☐	☐	LD.REG: 1 DR: 000 GateMDR: ☐ GateALU: ☐ GateMARMUX: ☐
54 57	1	LD.MAR: ☐ ALUK: ☐ GateALU: ☐ LD.REG: ☐ DR: ☐ GatePC: ☐
86 77	22	ADDR1MUX: ☐ ADDR2MUX: ☐ LD.PC: ☐ LD.MAR ☐ PCMUX: ☐
117 101	15	

During execution, we examined the computer during each clock cycle and recorded some information for certain clock cycles, producing the table shown below. The table is ordered by the cycle number in which the information was collected. Note that each memory access takes five clock cycles.

Fill in the missing instructions in the program, and complete the program by labeling the appropriate instruction AGAIN. Also, fill in the missing information in the table.

Given values for A and B, what does the program do?

ANSWER

> 破局的关键在于思考如何跳出循环进入 DONE 环节，是增加一条 BR 指令还是跳转到 BRz DONE 以及之前？考虑到只有两个指令空位，我们应该重点考虑后者，（前者的可能性很容易就能排除，因为这么干这个程序就干不了啥了）
>
> 结合 BRz 的条件，我们不难猜出 R1 是一个计数器，计数用来干啥呢？干别的都不现实，只能让 R0 不断自己加自己实现左移了。
>
> 题目有意图忽悠我们往内存写入上想，但这种可能性其实很好排除。
>
> 程序和表格见下文，最后一问的答案为把 A 左移无符号 B 位（或者说乘以 2 的 B 次方）。

```
        .ORIG x3000
        LD R0, A
        LD R1, B
AGAIN   BRz DONE
        ADD R0, R0, R0
        ADD R1, R1, #-1
        BRnzp AGAIN
DONE    ST R0, A
```

```
9           HALT
10  A       .FILL x0C00
11  B       .FILL x0001
12          .END
```

Cycle Number	State Number	Control Signals
1	18	LD.MAR: 1 LD.REG: 0 GateMDR: 0 LD.PC: 1 PCMUX: PC+1 GatePC: 1
44	0	LD.MAR: 0 LD.REG: 0 BEN 0 LD.PC: 0 LD.CC: 0
54	1	LD.REG: 1 DR: 000 GateMDR: 0 GateALU: 1 GateMARMUX: 0
54 57	1	LD.MAR: 0 ALUK: ADD GateALU: 1 LD.REG: 1 DR: 001 GatePC: 0
86 77	22	ADDR1MUX: PC ADDR2MUX: 0x09 LD.PC: 1 LD.MAR 0 PCMUX: +
117 101	15	

T8

There are two parts of an element in a **singly linked list**: the data and a pointer to the next element. The data and the pointer are in adjacent memory locations, with the data preceding the next element pointer. They are both 16-bit integers. The address of a list element is the starting address of such element. Example below:

List with 4 elements

x4000	x0041
x4001	x4100
x4100	x0042
x4101	x5000
x5000	x0043
x5001	x5090
x5090	x0044
x5091	x0000

You need to implement the function below, which takes R0 as its only parameter. R0 is the address of an existing list element, and you need to delete such element from the list (the element is not the last element). You also need to output the element's data to the monitor (for convenience, we consider the data an 8-bit character and ignore other bits). When the function returns, R0 should carry the address of the next element of the deleted element. All other registers are caller saved.

> Note: the addresses of the elements are allowed to change.

```
____ ________#0
____ ----------
TRAP x21
____ __, R2, __
____ R1, R0, __
____ R2, ------
____ ________#0
____ ________#1
____ R2, ------
RET
```

ANSWER

> 注意观察，第三行指令为 TRAP x21, 该指令将 R0 的低 8 位作为字符输出到监视器。说明在这之前 R0 已经加载了 R0 初值所存地址处的数据，而前方又只有两个指令，说明前面两个指令分别做到了复制 R0 的值（否则 R0 变为 DATA 数据后就没有指向当前节点的指针了）以及 LDR R0, R0, #0（加载数据）。又观察到第四行的 SR 中出现 R2, 说明要么 R2 之前存储了数据要么 AND R2, R2, #0 对 R2 进行清零，考虑到该函数长度有限，且后面 R2 又作为 DR，此处 R2 应该是存储了数据，那想必是 R0 的数据了，结合第一行最后的立即数是 #0，说明只能是通过 ADD 进行复制，而非 AND。
>
> 看到第五行的 SR 又用到了 R0，说明此时 R0 已经不是数据了，而是地址，那么第四行要么是个 STR 获取下个节点的地址，要么是再把 R2 的值拷贝回来。
>
> 现在就要考虑删除节点的原理了（说实话，稍微有点蠢）。单链表正常来说是没法删除当前节点的，因为没有前驱节点指针，所以只能把当前节点的数据覆盖为下一个节点的数据，然后把当前节点的 next 指针改为下下个节点的地址。在这种算法下 R0 仅需保持最初的地址不变即可。我们这样就可以知道第四行是 ADD R0, R2, #0 或 AND R0, R2, #-1。
>
> 容易看出第五行是 LDR R1, R0, #1。否则在只剩五行可供发挥的指令中，不先在搬运数据和地址前获取指向下个地点的指针而是干些别的，实在是有点太悠闲了。
>
> 剩下的四条指令就是数据和指针的搬运了，答案显而易见。完整的指令序列如下：

```
ADD R2, R0, #0
LDR R0, R0, #0
TRAP x21
ADD R0, R2, #0;OR AND R0, R2, #-1
LDR R1, R0, #1
LDR R2, R1, #0
STR R2, R0, #0
LDR R2, R1, #1
STR R2, R0, #1
RET
```