T1

假设当前 `PC` 指向一个 `LDR` 指令所在的地址。如果让 LC-3 处理该指令，则需要多少次内存访问？ `STI` 和 `TRAP` 呢？

T2

我们希望设计一个“什么也不做”的指令。许多 ISA 都有一个什么都不做的操作码，通常称之为 `NOP` 指令，即 `NO OPERATION`。该指令仍然经历指令周期的六个节拍，只是在执行节拍它什么也不做。请你分别用 `ADD`、 `AND`、 `BR` 指令给出一条可以扮演 `NOP` 功能的机器指令，不影响程序的正常执行。

T3

列举 LC-3 数据搬移指令的 4 种寻址模式，并指出每种模式下操作数存在的位置。

T4

下列两条 LC-3 指令有何异同？

A: `0000 111 101010101` B: `0100 1 11101010101`

T5

阐述 PC 相对寻址的寻址范围，将当前指令的地址视为原点。

T6

程序从 `x3001` 处开始执行，停止后， `R0`, `R1`, `R2` 和 `R3` 的内容是什么？

Adress	Data
x3001	1110 001 111111101
x3002	0001 011 001 100011
x3003	0101 010 001 000011
x3004	0011 010 000000001
x3005	1001 001 001 11111
x3006	1001 011 010 11111
x3007	1111 0000 0010 0101

T7

当求解一个复杂问题时，常常需要将其分解为若干子任务，先实现这些子任务，再将它们基于特定结构有机结合，完成任务目标。最常见的基本构建方法是 **顺序、条件和循环** 这三种结构。

1. 使用 **顺序** 结构求解斐波那契数列第 5 项。
2. 使用 **循环** 结构求解斐波那契数列第 10 项。

T8

LC-3 没有提供对应于逻辑 `OR` 的操作码。换句话说，不存在能够执行 `OR` 操作的 LC-3 指令。但我们可以通过一组指令来完成等价的 `OR` 功能。以下 4 条指令的任务是将 `R1` 和 `R2` 的内容相或，并将结果保存在 `R3` 中。试填写下面空缺的指令：

(1): `1001 100 001 111111`

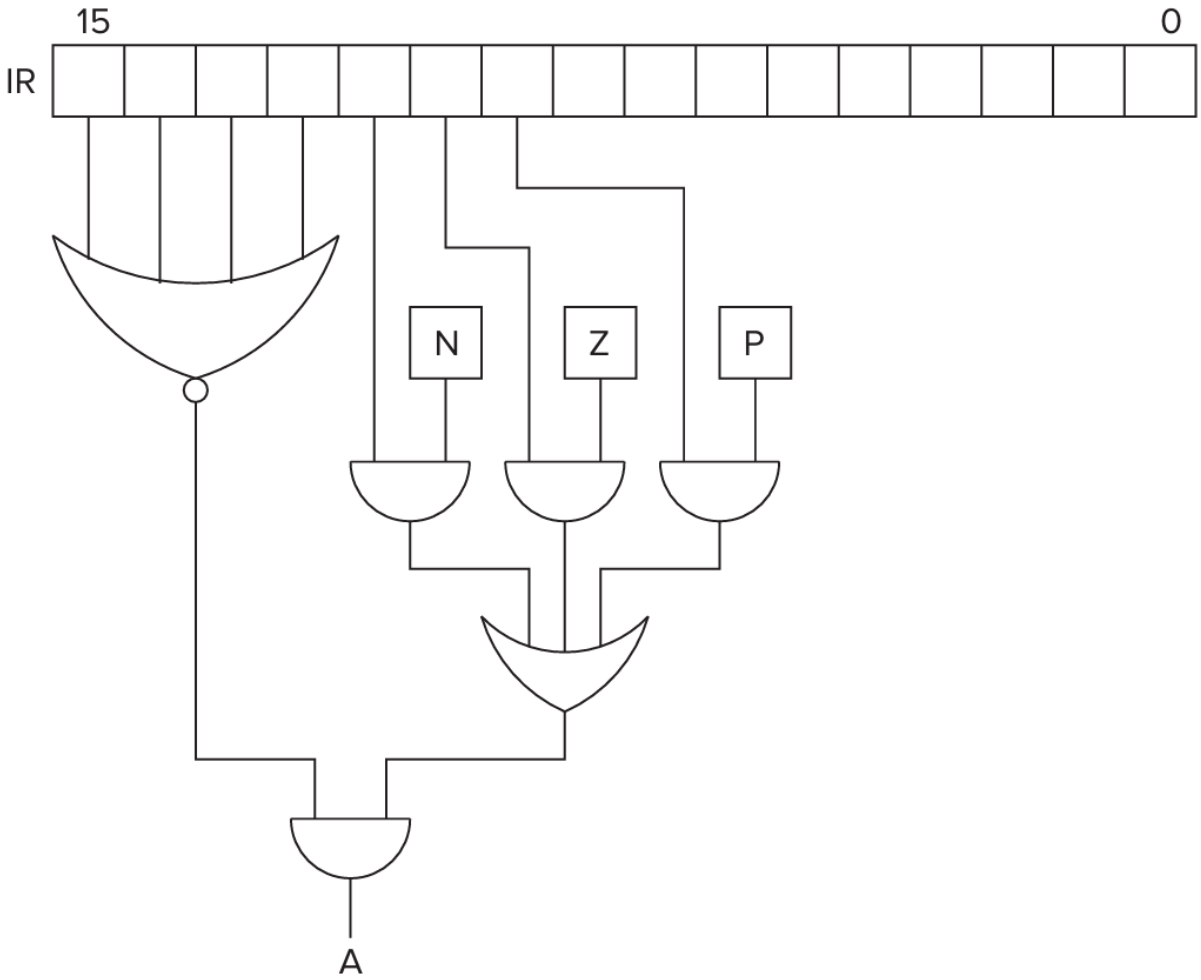
(2):

(3): `0101 110 100 000 101`

(4):

T9

下图所示为 LC-3 的局部图，试回答标识为 A 的信号作用是什么？



T10

下列代码是一个实现乘法的程序，它将 `R1` 和 `R2` 中的整数相乘，结果储存在 `R0` 中。

Adress	Data	Operation
x3000	0101 0000 0010 0000	`AND R0 <- R0, #0`
x3001	0001 0010 0111 1111	`ADD R1 <- R1, #-1`
x3002	0000 1000 0000 0010	`BRn x3005`
x3003	0001 0000 1000 0000	`ADD R0 <- R2, R0`
x3004	0000 1111 1111 1100	`BRnp x3001`
x3005	1111 0000 0010 0101	`HALT`

如果我们用 `ADD R0 <- R0, R1` 取代 x3003 处的指令，`R0` 最终会是什么？结果用 `R1` 或 `R2` 表示。