Homework 4

T1

1. `AND  R3, R2, #4` `// 0000 0100`

2. `AND  R3, R2, #12` `// 0000 1100`

3. `AND  R3, R2, #255` `// 1111 1111`

4. `AND  R3, R2, #128` `// 0100 0000`

   LC-3 最多只允许出现 imm5，所以如果只使用一条机器码指令则无法实现，
   可以使用多条指令或者用循环实现

T2

Address	Value	mean
x30FF	1110 001 000000001	LEA R1, x3101
x3100	0110 010 001 000010	LDR R2, R1, #2
x3101	1111 0000 0010 0101	HALT
x3102	0001 010 001 0 00 001	ADD R2, R1, R1
x3103	0001 010 010 0 00 010	ADD R2, R2, R2

使用 LC-3 Simulator 运行，结果如下

Status

Registers

R0: x7FFF **R1**: xFFFF **R2**: x1482 **R3**: x0000
R4: x0000 **R5**: x0000 **R6**: x0000 **R7**: xFD75
PC: xFD79 **IR**: xB02C **PSR**: x8001 **CC**: P

Clear R0–R7 | Reset all registers

T3

1. 在 LC-3 指令集中，我们可以使用以下的指令序列来模拟 MOVE R0,R1 操作：

   ```
   LDR R0, R1, #0
   ```

   这条指令将会从 R1 指向的内存地址读取数据，并将其加载到 R0 中。
2. 如果我们要在 LC-3 指令集中添加 `MOVE DR,SR` 指令，那么在解码操作之后，可以使用以下的微指令序列来模拟这个操作：

   ```
   MAR         <-   SR
   MDR         <-   Memory[MAR]
   MAR         <-   DR
   Memory[MAR] <-   MDR
   ```

   这个序列的操作如下：
   - 首先，将 SR 的值加载到 MAR 中，设置要读取的内存地址。
   - 然后，从 MAR 指向的内存地址读取数据，并将其加载到 MDR 中。
   - 接着，将 DR 的值加载到 MAR 中，设置要写入的内存地址。
   - 最后，将 MDR 中的数据写入到 MAR 指向的内存地址中。

T4

```
R1 XOR R2  =  ((NOT R1) AND R2) OR (R1 AND (NOT R2))
```

从而我们可以通过以下步骤实现 XOR 操作：

Adress	Value	Mean
x3000	1001 100 001 111111	NOT R4, R1

Adress	Value	Mean
x3001	0101 100 100 0 00 010	AND R4, R4, R2
x3002	1001 011 010 111111	NOT R3, R2
x3003	0101 011 001 0 00 011	AND R3, R1, R3
x3004	1101 011 011 0 00 100	OR R3, R3, R4

T5

LC-3 有五种寻址模式，分别是：

1. 立即数寻址（imm5）
2. 寄存器寻址
3. 基址偏移寻址（BaseR + offset6）
4. PC 相对寻址（PCoffset9/PCoffset11）
5. 间接寻址

对于给出的指令 ADD、NOT、LEA、LDR 和 JMP，我们可以将它们分类为操作指令、数据移动指令和控制指令：

- 操作指令：ADD、NOT
- 数据移动指令：LEA、LDR
- 控制指令：JMP

对于每个指令，可以使用的寻址模式如下：

- ADD：立即数寻址、寄存器寻址
- NOT：寄存器寻址
- LEA：PC 相对寻址
- LDR：基址偏移寻址
- JMP：基址偏移寻址

    参见附录A P525

T6

1. 将 R5 的内容复制到 R4 的单条 LC3 汇编指令是：

    ```
    ADD R4, R5, #0
    ```

2. 清除 R3 内容的单条 LC3 汇编指令是：

```
AND R3, R3, #0
```

3. 实现 R1 = R6-R7 的三条 LC3 汇编指令是：

```
NOT R7, R7
ADD R7, R7, #1    ; 取反加一
ADD R1, R6, R7
```

$$X_{补} - Y_{补} = X_{补} + \overline{Y_{补}} + 1$$

4. 将标签 DATA 处的值乘以 2 的三条 LC3 汇编指令是：

```
LDR R1, DATA
ADD R1, R1, R1
STR R1, DATA
```

这里我们首先从 DATA 处加载值到 R1，然后将 R1 的值加上自身，得到 2 倍的值，然后将结果存储回 DATA 处。

5. 使用一条 LC-3 指令根据 R1 的值设置条件码，不改变任何寄存器的值的指令是：

```
ADD R1, R1, #0
```

这条指令将 R1 的值加上 0，结果仍然是 R1 的值，但是会根据结果的值设置条件码。

T7

如果当前的程序计数器（PC）指向一个 JMP 指令的地址，那么 LC-3 处理这个指令需要进行 1 次内存访问。这是因为 JMP 指令是一条控制指令，它会直接改变 PC 的值，不需要访问内存。

对于 ADD 指令，LC-3 不需要进行任何内存访问。ADD 指令是一条操作指令，它会直接在寄存器之间进行操作，不涉及到内存访问。

对于 LDI 指令，LC-3 需要进行 2 次内存访问。LDI 指令是一条数据移动指令，它首先会访问内存以获取间接地址，然后再访问该间接地址以获取最终的数据。

T8

Adress	Value	Mean/Value
x3010	1110 0110 0011 1110	LEA R3, x304F
x3011	0110 1000 1100 0001	LDR R4, R3, #1
x3012	0110 1111 0000 0001	LDR R7, R4, #1
x3013	0110 1101 1111 1111	LDR R6, R7, #-1

Adress	Value	Mean/Value
…	…	…
x304E	NAN	x70A4
x304F	NAN	x70A3
x3050	NAN	x70A2
x70A2	NAN	x70A4
x70A3	NAN	x70A3
x70A4	NAN	x70A2

1. 则由以上分析可知：

   - 先将 x304F 处的值加载到 R3　　R3 <- x70A3
   - 然后将 R3 处的地址 + 1 后加载到 R4　　R4 <- x70A2
   - R4 处的地址 + 1 后加载到 R7　　R7 <- x70A3
   - R7 处的地址 - 1 后加载到 R6　　R6 <- x70A4

   故运行后，R6 存储的值是 x70A4

2. 若仅为完成要求（仅使用一条 LEA 指令且仅考虑 R6），可以在地址 x3010 处使用以下指令

   ```
   1110 0110 0011 1101          ; LEA R6, x304E
   ```

   这样就会把地址x304E处的值加载到 R6 中，即 R6 <- x70A4
   从而在结果上与之前的处理一致

T9

After the execution of the following code, the value stored in R0 is 12. Please speculate what the value stored in R5 is like.

Address	Value	mean
x3000	0101 0000 0010 0000	AND R0, R0, #0
x3001	0101 1111 1110 0000	AND R7, R7, #0
x3002	0001 1100 0010 0001	ADD R6, R0, #1
x3003	0001 1101 1000 0110	ADD R6, R6, R6
x3004	0101 1001 0100 0110	AND R4, R5, R6

Address	Value	mean
x3005	0000 0100 0000 0001	BRz x3007
x3006	0001 0000 0010 0011	ADD R0, R0, #3
x3007	0001 1111 1110 0010	ADD R7, R7, #2
x3008	0001 0011 1111 0010	ADD R1, R7, #-14
x3009	0000 1001 1111 1001	BRn x3003
x300A	0101 1111 1110 0000	AND R7, R7, #0

首先，我们看到在地址 x3000 和 x3001 中，R0 和 R7 都被清零。然后，在地址 x3002 中，R6 被设置为 R0+1，也就是 1。接下来，在地址 x3003 中，R6 被设置为 R6+R6，也就是 2。

然后，在地址 x3004 中，R4 被设置为 R5 和 R6 的按位与结果。由于 R6 的值为 2（二进制表示为 10），因此当 R5 的次低位为 0 时， R4 的结果为 0。

接下来，在地址 x3005 中，如果 R4 为 0（也就是 R5 的最低位为 0），则跳转到地址 x3007，否则继续执行。由于我们知道最后 R0 的值为 12，而在地址 x3006 中，R0 的值被增加了 3，所以我们知道程序肯定执行且总共执行了 4 次地址 x3006 中的指令

然后，在地址 x3007 中，R7 的值被增加了 2，然后在地址 x3008 中，R1 被设置为 R7-14。接下来，在地址 x3009 中，如果 R1 为负，则跳转到地址 x3003，否则继续执行。

最后，在地址 x300A 中，R7 被清零。

由上述分析可知，R6 起到掩码的作用，用于检验 R5 的二进制表示中有多少个一，
由结果看，R5 中有 4 个“1”（R0 被操作了 12/3 = 4 次），而由于掩码 R6 没有检查到最低位，
从而最低位必为 0，4 个“1”可以出现在剩下的 7 个位置中的任意位置

T10

Address	Value	mean
x3000	1001 000 000 111111	NOT R0, R0
x3001	0001 000 000 1 00001	ADD R0, R0, #1
x3002	0101 010 010 1 00000	AND R2, R2, #0
x3003	0001 011 000 0 00 001	ADD R3, R0, R1
x3004		
x3005	0001 010 010 1 00001	ADD R2, R2, #1

Address	Value	mean
x3006		
x3007	0000 010 000000111	BRz x300F
x3008	0101 001 001 1 11111	AND R1, R1, #-1
x3009	0000 100 000000010	BRn x300C
x300A	0001 001 001 0 00 001	ADD R1, R1, R1
x300B	0000 111 111110111	BRnzp x3003
x300C		
x300D		
x300E	0000 111 111110100	BRnzp x3003
x300F	0101 010 010 1 00000	AND R2, R2, #0
x3010	0001 010 010 1 11111	ADD R2, R2, #-1
x3011		
x3012	1111000000100101	HALT

(时间原因，要复习电磁，先不写了，就翻译一下意思)