Homework 3

T1

- 计算已占用 bits 数
  - opcode 有 64
  - register 有 56

    两者各自需要 6 bits 才能表示完整，
- 计算 IMM 可用位数

$$32 - 6 - 6 - 6 = 14$$

即 IMM 可用 14 bits
其表示范围为

$$-2^{13} \leq IMM \leq 2^{13} - 1$$

T2

```
- x3000: 0101 000 000 1 00000 (AND R0, R0, #0)      - 将R0寄存器清零
- x3001: 0001 000 000 1 00010 (ADD R0, R0, #2)      - 将R0寄存器加2
- x3002: 0000 011 X   - (BRp X) 如果上一条指令的结果为正，则跳转到PC + X
- x3003: 0001 000 000 1 00011 (ADD R0, R0, #3)      - 将R0寄存器加3
- x3004: 0001 000 000 1 00001 (ADD R0, R0, #1)      - 将R0寄存器加1
- x3005: 1111 0000 0010 0101 (HALT)                 - 停止程序执行
```

- X = 000000010：
  - 程序会跳转到 x3002 + X = x300A，
  - 这是一个不存在的地址，此时视作程序停止。R0 中存储的值是"2"。
- X = 000000001：
  - 程序会跳转到 x3002 + X = x3003，
  - 然后会继续执行，最后在 x3005 处停止。R0 中存储的值是 6。
- X = 000000000：
  - 程序会继续执行，最后在 x3005 处停止。
  - 在 x3005 处的指令执行完毕后，R0 中存储的值是 6。
- X = 111111111：
  - 程序会跳转到 x3002 - 1 = x3001，
  - 然后继续执行到 x3002 再次跳转，程序不会停止
- X = 111111110：
  - 程序会跳转到 x3002 - 2 = x3000，
  - 然后继续执行到 x3002 再次跳转，程序不会停止

X	程序是否停止?	R0 中存储的值
000000010	是	2
000000001	是	6
000000000	是	6

X	程序是否停止?	R0 中存储的值
111111111	否	~
111111110	否	~

T3

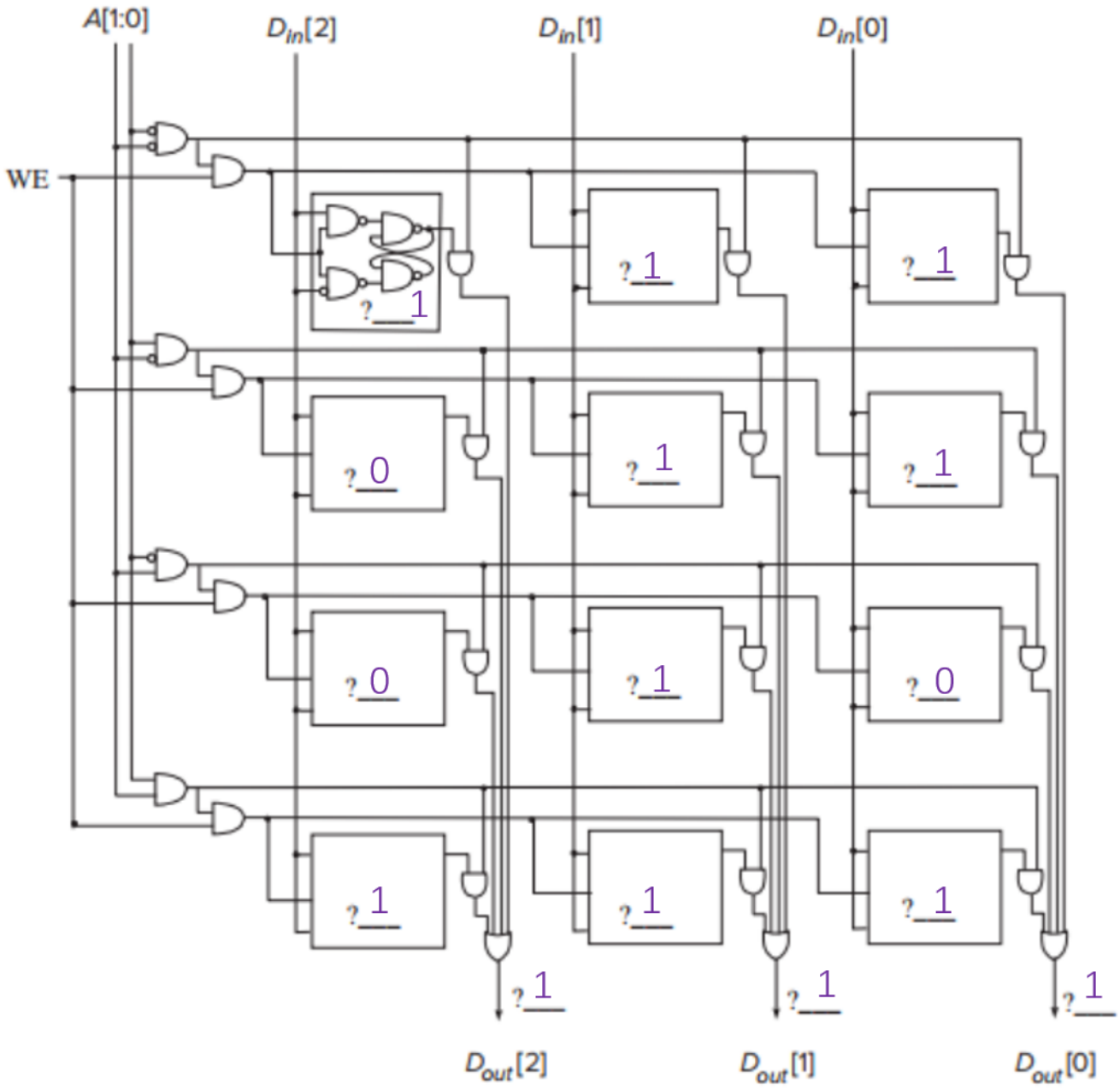
- `x3000` 处的 `0001 000 001 0 00 010` 指令意为：
  - 将寄存器R_2与R_1相加
  - 所得结果存储进R_0
- `x3001` 处的 `0000 011 000000111` 的指令意为：
  - 若上一条指令的结果为正，将跳转至 `x3001 + 7` 的位置继续执行

故若要使得执行完这两条指令之后，下一条为 `x3009` 处的，应该使寄存器R_2与R_1至少有一个不为零

T4

```
(a)
ADD(0001) 和 AND(0101) 有了更大的立即数范围。
NOT(1001) 没有获益。
(b)
load 和 store 能有更多的一位去寻址。
(c)
branch 中没有寄存器，因此没有变化。
```

T5



T6

MAR（Memory Address Register）保存了要访问的内存地址，
MDR（Memory Data Register）保存了要写入或从内存中读取的数据。

根据题目中的信息和约束条件，我们可以推断出以下的答案：

操作序号	读/写	MAR	MDR
1	W	x4000	11110
2	R	x4003	10110
3	W	x4001	10110
4	R	x4002	01101
5	W	x4003	01101

地址	访问 1 前	访问 3 后	访问 5 后
x4000	01101	11110	11110
x4001	11010	10110	10110
x4002	01101	01101	01101
x4003	10110	10110	01101
x4004	11110	11110	11110

- x4004 从始至终不需要动，操作 1 为写入，观察第二个表格的“访问 3 后”，x4000 的值末尾正好有个 0，符合要求，所以第一个 MAR 填入 x4000
- 操作 3 为写入，且 MDR 的值发生了改变，故操作 2 的 MAR 必然为读取，观察第二个表格的“访问 1 前”只有地址 x4003 的值符合，故第二个 MAR 填入 x4003
- 根据上一条推断，此处的 MDR 填入 10110，再观察第二个表格的“访问 3 后”，地址 x4001 的值格式很符合，故第三个 MAR 填入 x4001
- 此时再看到地址 x4003，它在整个过程中共经历了三个值，故最后两次操作必然为从某处读取值再写入 x4003
- 观察表格后很容易发现是读取 x4002 的值，由此最后两个操作的 MDR 与 MAR 也得以确定
- 最后第二个表格剩下的空照抄即可（值不改变）

T7

1. 每秒发生2×10^8个机器周期
2. 计算机在 1 秒内可以处理$\frac{2 \times 10^8}{8} = 2.5 \times 10^7$条指令
3. 此时每条指令执行时间相当于只需要 1 个 cycle，所以每秒可以执行约 2×10^8条指令

T8

```
A XOR B = (A AND (NOT B)) OR ((NOT A) AND B)
```

下面是推导过程

```
x3000 R1 取反存入 R7        NOT R1
x3001 R2 取反存入 R6        NOT R2
x3002 R2 与 R7 存入 R5      R2 AND (NOT R1)
x3003 R1 与 R6 存入 R4      R1 AND (NOT R2)
下面完成 OR 步骤
x3004 R5 取反存入 R1        (NOT R2) AND R1
x3005 R4 取反存入 R2        (NOT R1) AND R2
x3006 R1 与 R2 存入 R0      ((NOT A) AND (NOT B)) OR (A AND B)
x3007 R0 取反存入 R3        A XOR B
```

从而最终的结果为：

地址	指令
x3000	1001 111 001 111111
x3001	1001 110 010 111111
x3002	0101 101 010 000 111
x3003	0101 100 001 000 110

地址	指令
x3004	1001 001 101 111111
x3005	1001 010 100 111111
x3006	0101 000 010 000 001
x3007	1001 011 000 111111

T9

- `0001 010 001 1 00010` ADD指令，将数2（`00010`）加到寄存器R1中，并将结果存储在寄存器R2中
- `0000 111 000000000` BR指令，无条件地跳转到PC+0的位置。
- `0000 101 000000100` BR指令，若当前状态为正或零，则跳转到PC+4的位置
- `1001 010 111 111111` NOT指令，将 R7 取反并存入 R2
- `1111 0000 00100011` TRAP指令，执行服务例程，服务例程的向量是x23（`00100011`）

所以指令 `0000 111 000000000` 可以被用来作为 NOP 指令

T10

BR和JMP都是跳转指令，但它们在使用上有所不同。
BR指令在一定范围内进行条件跳转，而JMP指令则可以无条件地跳转到任何位置。

在LC-3指令集中，BR指令是条件跳转指令。
它根据条件寄存器（N - 负数，Z - 零，P - 正数）的状态来决定是否跳转。
然而，BR指令的一个限制是它只能在当前程序计数器（PC）的一定范围内跳转。

相比之下，JMP指令（跳转指令）可以解决这个问题。
JMP指令允许无条件地跳转到任何位置。
在执行JMP指令时，会将某个寄存器的内容加载到程序计数器（PC）中。
因此，通过改变该寄存器的值，可以实现对任意地址的跳转，从而克服了BR指令的限制。