Lab4 实验报告

Task 1

通过阅读，可知代码执行的操作为：

> R2 = x300F
>
> R0 = 0
>
> R7 = x3003, PC = x3004 或 x3003
>
> HALT

x3003 为 HALT，所以上一条指令中 PC = x3004，第一个 x = 1

> MEM[R2] = R7
>
> R2 = R2 + 1 或 R2 = R2 + 9
>
> R0 = R0 + 1
>
> R1 = MEM[PC + 17]
>
> R1 = R1 - 1 或 R1 = R1 - 9
>
> MEM[PC+15] = R1
>
> BRz PC=PC+1
>
> R7 = PC, PC = PC - 8

可知上述代码依靠 R1 从 5 减到 0 执行循环，所以第三个 x = 0

> R2 = R2 -1
>
> R7 = MEM[R2], R7 = MEM[x300F]
>
> PC = R7

首先最后一步为 RET，所以在那之前需要将 R7 的值设置好，所以其上一条语句的指令只能是LDR，所以最后一个 x = 1

上述代码若想退回到 HALT（这一步是必须的），那么 R2 就需要最终变为 x300F，这样 R7 = MEM[R2] 后，经过 RET 语句，才能回到 HALT

而在最后一部分中，R2 是在减一，然后改变 R7 进行循环，R2 逐渐减一，最终实现 R2 = x300F

所以第二个 x = 0，R2 逐渐加 1，并修改MEM[R2]=R7，这样才能在最后一部分进行有效的循环，最终回到 HALT，达到题目需要的寄存器值

综上，补全后的代码如下：

```
1110010000001110
0101000000100000
0100100000000001
1111000000100101
0111111010000000
0001010010100001
0001000000100001
```

```
0010001000010001
0001001001111111
0011001000001111
0000010000000001
0100111111111000
0001010010111111
0110111010000000
1100000111000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000101
```

Task 2

根据提示，对一个数模 7 可以先模 8，即利用 8 = 7 + 1，2 * 8 = 2 * 7 +2 等一系列等式，将当前数字大于 7 的部分转化为 7 的某次方加一个余数，再将这个余数与原数的小于等于 7 的部分相加，再执行上述操作，直到最后求得该数模 7 的余数

阅读代码，可知代码执行的操作为：

> R1 = MEM[PC+21]
>
> R7 = x3002, PC = PC + 8

即在内存中取出要求余数的数字，然后进行跳转 JSR 指令，跳转到：

> R2 = 1, R3 = 8, R4 = 0

对寄存器置数

> R5 = R3 & R1
>
> PC = PC + 1
>
> R4 = R2 + R4
>
> R2 = R2 + R2
>
> XX = R3 + R3
>
> BRxxx PC = PC - 6

可知上述操作为模 8 操作，但是每一次减去 8 的倍数，都会记录减去的数模 7 的余数，并将其加在 R4 中，不断循环，最终实现模 7 操作，余数存储在 R4 中

所以上一部分操作中，第一处 XXX 是R3，第二处 XXX 是np，即只有当 R3 = 0 以后，退出循环

> RET

当上一个模 7 运算执行完后，回到原本 JSR 指令的下一条指令，即：

> R2 = R1 & 7
>
> R1 = R2 + R4

> R0 = RX -7
>
> PC = PC - XXX
>
> R0 = RX - 7
>
> PC = PC + 1
>
> R1 = R1 -7

根据前面的分析可知，在跳转到当前指令之前，已经完成了对原数大于 7 的部分的模 7 操作，并将余数存储在 R4 中，所以现在将原数小于 7 的部分与 R4 相加，如果此数大于七，则继续执行以上全部操作，如果次数小于 7 ，则为原数模 7 的余数

所以这一部分指令中，第一个 RX 为 R1，第二个 RX 也为 R1，而 PC = PC + XXX，XXX 为 -5

综上，补全后的代码如下：

```
0010001000010101
0100100000001000
0101010001100111
0001001010000100
0001000001111001
0000001111111011
0001000001111001
0000100000000001
0001001001111001
1111000000100101
0101010010100000
0101011011100000
0101100100100000
0001010010100001
0001011011101000
0101101011000001
0000010000000001
0001100010000100
0001010010000010
0001011011000011
0000101111111010
1100000111000000
0000000100100000
```