Lab 4 实验报告

- 姓名：谭旭
- 学号：PB20030775

实验内容

阅读并理解给出的代码，将其中的缺失的部分 (x) 补充完整，使得程序完成指定的功能。

Task 1

题目代码：

```
x3000;  1110010000001110;  LEA R2, x300F
x3001;  0101000000100000;  AND R0, R0, #0
x3002;  010010000000000x;  JSR ?
x3003;  1111000000100101;  HALT
x3004;  0111111010000000;  STR R7, R2, #0
x3005;  000101001010x001;  ADD R2, R2, #?
x3006;  0001000000100001;  ADD R0, R0, #1
x3007;  0010001000010001;  LD R1, x3019
x3008;  0001001x01111111;  ADD R1, ?, #-1
x3009;  0011001000001111;  ST R1, x3019
x300A;  0000010000000001;  BRz x300C
x300B;  0100111111111000;  JSR x3004
x300C;  0001010010111111;  ADD R2, R2, #-1
x300D;  01x0111010000000;  ?
x300E;  1100000111000000;  RET
x300F;  0000000000000000;
x3010;  0000000000000000;
x3011;  0000000000000000;
x3012;  0000000000000000;
x3013;  0000000000000000;
x3014;  0000000000000000;
x3015;  0000000000000000;
x3016;  0000000000000000;
x3017;  0000000000000000;
x3018;  0000000000000000;
x3019;  0000000000000101;
```

程序运行结束后，要求

```
R0 = 5, R1 = 0, R2 = 300F, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 3003
```

观察代码，可以看到在 `x300A` 和 `x300B` 处有两个跳转，

`x300A` 处的跳转相当于跳过了 `x300B` ，

`x300B` 处的跳转回到了 `x3004` ，并在之后的代码中将 R7 写入 R2 所指向的内存。

显然这是一个递归的结构， `x300A` 处的跳转则是递归的边界条件。

继而向上阅读递归部分，可以了解到 R0 是递归次数的计数器， R2 指向记录递归所用的栈。

递归部分负责每层令 R1 自减，直到 R1 减为 0 。

因此 `x3008` 的代码应当为 `x3008; 0001001001111111; ADD R1, R1, #-1`

`x3005` 的代码应当为 `x3005; 0001010010100001; ADD R2, R2, #1`

考虑到程序最终的 R7 为 `x3003` ，这就证明程序的第一层递归入口在 `x3002` 处。

所以 `x3002` 处的跳转应当与 `x300B` 处相同，都是跳转到 `x3004` ，

因此 `x3002` 的代码应当为 `x3002; 0100100000000001; JSR x3004`

最后考虑 `x300D` 处的代码，由于程序是递归的，每次执行 `RET` 指令时，应当将上一层的地址读入 R7 ，

因此 `x300D` 处应当是一个读入操作，故代码为 `x300D; 01x0111010000000; LDR R7, R2, #0`

最终程序为

```
x3000;  1110010000001110;   LEA R2, x300F
x3001;  0101000000100000;   AND R0, R0, #0
x3002;  0100100000000001;   JSR x3004
x3003;  1111000000100101;   HALT
x3004;  0111111010000000;   STR R7, R2, #0
x3005;  0001010010100001;   ADD R2, R2, #1
x3006;  0001000000100001;   ADD R0, R0, #1
x3007;  0010001000010001;   LD R1, x3019
x3008;  0001001001111111;   ADD R1, R1, #-1
x3009;  0011001000001111;   ST R1, x3019
x300A;  0000010000000001;   BRz x300C
x300B;  0100111111111000;   JSR x3004
x300C;  0001010010111111;   ADD R2, R2, #-1
x300D;  01x0111010000000;   LDR R7, R2, #0
x300E;  1100000111000000;   RET
x300F;  0000000000000000;
x3010;  0000000000000000;
x3011;  0000000000000000;
x3012;  0000000000000000;
x3013;  0000000000000000;
x3014;  0000000000000000;
x3015;  0000000000000000;
x3016;  0000000000000000;
x3017;  0000000000000000;
x3018;  0000000000000000;
x3019;  0000000000000101;
```

Task 2

题目代码：

```
x3000;   0010001000010101;   LD R1 x3016
x3001;   0100100000001000;   JSR x300A
x3002;   0101010001100111;   AND R2, R1, #7
x3003;   0001001010000100;   ADD R1, R2, R4
x3004;   00010000xxx11001;   ADD R0, ?, ?
x3005;   00000011xxx11011;   BRp ?
x3006;   00010000xxx11001;   ADD R0, ?, ?
x3007;   0000100000000001;   BRn x3009
x3008;   0001001001111001;   ADD R1, R1, #-7
x3009;   1111000000100101;   HALT
x300A;   0101010010100000;   AND R2, R2, #0
x300B;   0101011011100000;   AND R3, R3, #0
x300C;   0101100100100000;   AND R4, R4, #0
x300D;   0001010010100001;   ADD R2, R2, #1
x300E;   0001011011101000;   ADD R3, R3, #8
x300F;   0101101011000001;   AND R5, R3, R1
x3010;   0000010000000001;   BRz x3012
x3011;   0001100010000100;   ADD R4, R2, R4
x3012;   0001010010000010;   ADD R2, R2, R2
x3013;   0001xxx011000011;   ADD ?, R3, R3
x3014;   0000xxx111111010;   BR? x300F
x3015;   1100000111000000;   RET
x3016;   0000000100100000;   x0120
```

该程序完成的功能为对一个数进行模 7 操作。

首先，程序将 `x3016` 的值读入 R1 ，程序将通过一系列操作将其模 7 的结果存入 R1 。

观察 `x3004` 和 `x3005` 行，这两行均实现了一个加操作，并且其第 3 、 4 为均为 1 ，

所以这两条加指令一定是一个寄存器加一个五位立即数，故这两条指令为

```
x3004;  00010000xx111001;   ADD R0, ?, #-7
x3006;  00010000xx111001;   ADD R0, ?, #-7
```

观察 `x300F` 到 `x3014` ，可以看出，这个循环的结束依赖于 R3 + R3 的状态码。

而在 `x300F` 行，进行了 R3 与 R1 的操作，想到 lab 1 中的 p 版本乘法，可以猜测这里 R3 的功能为取出 R1 中的每一位。

因此 `x3013` 行应该是 `x3013; 0001011011000011; ADD R3, R3, R3`

`x3014` 行应该是 `x3014; 0000001111111010; BRp x300F`

补全该部分后，可以知道 `x300F` 到 `x3014` 的循环统计了 R1 整除 8 的结果，将结果保存在 R4 。

考虑到有以下性质

$$x = 8k + b = k + b \mod 7$$

因此最终保存在 R1 中的数应该是 R1 模 8 的结果加 R4 。

由 `x3002` `x3003` 可以验证这一猜想。

最后，由于通过上述方法得到的 R1 并不一定小于 7 ，需要通过多次迭代，将新的 R1 放入程序中计算。

故 `x3004` 到 `x3006` 行为

```
x3004;   0001000001111001;  ADD R0, R1, #-7
x3005;   0000001111111011;  BRp x3001
x3006;   0001000001111001;  ADD R0, R1, #-7
```

最终程序为

```
x3000;   0010001000010101;  LD R1 x3016
x3001;   0100100000001000;  JSR x300A
x3002;   0101010001100111;  AND R2, R1, #7
x3003;   0001001010000100;  ADD R1, R2, R4
x3004;   0001000001111001;  ADD R0, R1, #-7
x3005;   0000001111111011;  BRp x3001
x3006;   0001000001111001;  ADD R0, R1, #-7
x3007;   0000100000000001;  BRn x3009
x3008;   0001001001111001;  ADD R1, R1, #-7
x3009;   1111000000100101;  HALT
x300A;   0101010010100000;  AND R2, R2, #0
x300B;   0101011011100000;  AND R3, R3, #0
x300C;   0101100100100000;  AND R4, R4, #0
x300D;   0001010010100001;  ADD R2, R2, #1
x300E;   0001011011101000;  ADD R2, R2, #1
x300F;   0101101011000001;  AND R5, R3, R1
x3010;   0000010000000001;  BRz x3012
x3011;   0001100010000100;  ADD R4, R2, R4
x3012;   0001010010000010;  ADD R2, R2, R2
x3013;   0001011011000011;  ADD R3, R3, R3
x3014;   0000001111111010;  BRp X300F
x3015;   1100000111000000;  RET
x3016;   0000000100100000;  x0120
```