第四次实验

姓名: 傅申 学号: PB20000051

1. Task 1
2. Task 2

1 Task 1

`task1.txt` 文件中的机器码,翻译为汇编语言后为

```
        .ORIG   x3000
        LEA     R2, #14     ; x300F → R2
        AND     R0, R0, #0  ; x0000 → R1
        JSR     #x          ; x3003 → R7, x3003 + x → PC
        HALT
FUNC    STR     R7, R2, #0  ; R7 → mem[R2]
        ADD     R2, R2, #y  ; R2 + y → R2
        ADD     R0, R0, #1  ; R0 + 1 → R0
        LD      R1, #17     ; mem[x3019] → R1
        ADD     R1, Rz, #-1 ; Rz - 1 → R1
        ST      R1, #15     ; R1 → mem[x3019]
        BRz     #1
        JSR     #-8         ; x3013 → R7, x3005 → PC // Call FUNC
        ADD     R2, R2, #-1 ; R2 - 1 → R2
        INS                 ; Unknown instruction
        RET
        .BLKW   #10         ; x300F ~ x3018
        .FILL   #5          ; x3019
        .END
```

其中 `x = 0 or 1, y = 1 or 9, z = 1 or 5, INS = JSR #-384 or LDR R7, R2, #0` 不难看出, 两个 `JSR` 指令都是调用 `FUNC` 子程序, 则 `x = 1`, 而且 `FUNC` 程序是一个递归函数, 所以在其开头要入函数栈, 在其 `RET` 之前要出函数栈, 因此有 `y = 1`, `INS = LDR R7, R2, #0`. 这样 `FUNC` 子程序的伪代码如下:

```
FUNC():
    R0 = R0 + 1
    R1 = mem[x3019]
    R1 = Rz - 1
    mem[x3019] = R1
    if R1 != 0:
        FUNC()
    Return
```

在程序结束时的寄存器状态中，发现 R0 = 5，说明 FUNC 被调用的次数为 5 次，显然 z = 5 是说不通的，因此 z = 1. 这样 FUNC 子程序的伪代码如下：

```
FUNC():
    R0 = R0 + 1
    mem[x3019] = mem[x3019] - 1
    if mem[x3019] != 0:
        FUNC()
    Return
```

所以 FUNC 的功能是将内存中 x3019 位置的值清零，并将原来的值赋给 R1.

最终填完空后的机器码为

```
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
```

使用 LabS 的模拟器运行程序, 输出为:

```
> lc3simulator.exe -f rec.txt
R0 = 5, R1 = 0, R2 = 300f, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 3003
COND[NZP] = 001
PC = 0

cycle = 3a
```

与要求一致.

2 Task 2

`task2.txt` 文件中的机器码,翻译为汇编语言后为

```
        .ORIG   x3000
        LD      R1, #21     ; x3016 → R1
LOOP1   JSR     DIV         ; Call DIV
        AND     R2, R1, #7  ; R1 % 8 → R2
        ADD     R1, R2, R4  ; R2 + R4 → R1
        ADD     R0, Rx, #-7 ; Rx - 7 → R0
        BRp     LOOP1       ; fill the missing bit by guessing :)
        ADD     R0, Rx, #-7 ; Rx - 7 → R0
        BRn     #1
        ADD     R1, R1, #-7 ; R1 - 7 → R1
        HALT
DIV     AND     R2, R2, #0  ; clear R2
        AND     R3, R3, #0  ; clear R3
        AND     R4, R4, #0  ; clear R4
        ADD     R2, R2, #1  ; R2 + 1 → R2
        ADD     R3, R3, #8  ; R3 + 8 → R3
LOOP2   AND     R5, R3, R1  ; R3 % R1 → R5
        BRz     #1
        ADD     R4, R2, R4  ; R4 + R2 → R4
        ADD     R2, R2, R2  ; R2 + R2 → R2
        ADD     Rx, R3, R3  ; R3 + R3 → Ry
        BRxxx   LOOP2
        RET
        .FILL   #288        ; x3016
```

题目中指出这段代码是用于求余的, 注意到

$$x \equiv \left\lfloor \frac{x}{8} \right\rfloor + (x \bmod 8) \pmod 7$$

而第 4 行的指令将 `R1 mod 8` 赋值给了 `R2`, 而第 5 行的指令将 `R2 + R4` 重新赋值给了 `R1`, 因此可以猜测从第 12 行开始的子程序 `DIV` 的作用为 `R1/8 → R4`. 而第 6 ~ 10 行则能保证不断对 `R1` 进行 `R1%8 + R1/8 → R1` 操作, 直到 `R1 < 7`, 因此第 6, 8 行的 `Rx` 均为 `R1`.
分析 `DIV` 子程序, 知道它的功能是 `R1/8 → R4`, 即为 `R1>>3 → R4`, 而其伪代码如下

```
DIV():
    R2 = 1
    R3 = 8
    do:
        if R3 % R1 != 0:
            R4 = R2 + R4
        R2 = R2 + R2
        Rx = R3 + R3
    while (Rx ...)
    Return
```

显然 `R3` 作为掩码, 则 `R2` 要始终为 `R3 >> 3`, 因此 `Rx` 即为 `R3`, 且循环结束的条件应该为 `R3 = 0`, 因此汇编代码第 22 行应该为 `BRnp LOOP2`.

最后得到的机器码为:

```
0010001000010101
0100100000001000
0101010001100111
0001001010000100
0001000001111001
0000001111111011
0001000001111001
0000100000000001
0001001001111001
1111000000100101
0101010010100000
0101011011100000
0101100100100000
0001010010100001
0001011011101000
```

```
0101101011000001
0000010000000001
0001100010000100
0001010010000010
0001011011000011
0000101111111010
1100000111000000
0000000100100000
```

同样使用 LabS 的模拟器运行程序, 输出为:

```
> lc3simulator.exe -f mod.txt
R0 = fffa, R1 = 1, R2 = 0, R3 = 0
R4 = 1, R5 = 0, R6 = 0, R7 = 3002
COND[NZP] = 100
PC = 0

cycle = ec
```

其中 $288 \equiv 1 (\mathrm{mod}\ 7)$, 程序执行正确.