

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

```
1      .ORIG    x3000
2      LEA      R2, #14      ; x300F → R2
3      AND      R0, R0, #0    ; x0000 → R1
4      JSR      #x           ; x3003 → R7, x3003 + x → PC
5      HALT
6  FUNC  STR      R7, R2, #0    ; R7 → mem[R2]
7      ADD      R2, R2, #y     ; R2 + y → R2
8      ADD      R0, R0, #1     ; R0 + 1 → R0
9      LD       R1, #17        ; mem[x3019] → R1
10     ADD      R1, Rz, #-1     ; Rz - 1 → R1
11     ST       R1, #15        ; R1 → mem[x3019]
12     BRz      #1
13     JSR      #-8            ; x3013 → R7, x3005 → PC // Call FUNC
14     ADD      R2, R2, #-1     ; R2 - 1 → R2
15     INS
16     RET
17     .BLKW    #10            ; x300F ~ x3018
18     .FILL    #5             ; x3019
19     .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:

```
1  FUNC():
2      R0 = R0 + 1
3      R1 = mem[x3019]
4      R1 = Rz - 1
5      mem[x3019] = R1
6      if R1 != 0:
7          FUNC()
8      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:

```
1  FUNC():
2      R0 = R0 + 1
3      mem[x3019] = mem[x3019] - 1
4      if mem[x3019] != 0:
5          FUNC()
6      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

```
1  11100100000001110
2  0101000000100000
3  0100100000000001
4  1111000000100101
5  0111111010000000
6  0001010010100001
7  0001000000100001
8  0010001000010001
9  0001001001111111
10 0011001000001111
11 0000010000000001
12 0100111111111000
13 0001010010111111
14 0110111010000000
15 1100000111000000
16 0000000000000000
17 0000000000000000
18 0000000000000000
19 0000000000000000
20 0000000000000000
21 0000000000000000
22 0000000000000000
23 0000000000000000
24 0000000000000000
25 0000000000000000
26 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

```
1      .ORIG    x3000
2      LD      R1, #21      ; x3016 → R1
3  LOOP1 JSR     DIV         ; Call DIV
4          AND   R2, R1, #7  ; R1 % 8 → R2
5          ADD   R1, R2, R4  ; R2 + R4 → R1
6          ADD   R0, Rx, #-7 ; Rx - 7 → R0
7          BRp   LOOP1      ; fill the missing bit by guessing :)
8          ADD   R0, Rx, #-7 ; Rx - 7 → R0
9          BRn   #1
10         ADD   R1, R1, #-7 ; R1 - 7 → R1
11        HALT
12  DIV    AND   R2, R2, #0   ; clear R2
13        AND   R3, R3, #0   ; clear R3
14        AND   R4, R4, #0   ; clear R4
15        ADD   R2, R2, #1   ; R2 + 1 → R2
16        ADD   R3, R3, #8   ; R3 + 8 → R3
17  LOOP2 AND   R5, R3, R1   ; R3 % R1 → R5
18        BRz   #1
19        ADD   R4, R2, R4   ; R4 + R2 → R4
20        ADD   R2, R2, R2   ; R2 + R2 → R2
21        ADD   Rx, R3, R3   ; R3 + R3 → Ry
22        BRxxx LOOP2
23        RET
24        .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 $R1 \bmod 8$ 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 $R1/8 \rightarrow R4$. 而第 6, 7 行则能保证不断对 $R1$ 进行 $R1\%8 + R1/8 \rightarrow R1$ 操作, 直到 $R1 < 8$, 第 8, 9 行则是处理 $R1 = 7$ 的情况, 此时应该让 $R1 - 7 \rightarrow R1$, 因此第 6, 8 行的 Rx 均为 $R1$.

分析 `DIV` 子程序, 知道它的功能是 $R1/8 \rightarrow R4$, 即为 $R1 \gg 3 \rightarrow R4$, 而其伪代码如下

```
1  DIV():
2      R2 = 1
3      R3 = 8
4      do:
5          if R3 % R1 != 0:
6              R4 = R2 + R4
7              R2 = R2 + R2
8              Rx = R3 + R3
9      while (Rx ...)
10     Return
```

显然 $R3$ 作为掩码, 则 $R2$ 要始终为 $R3 \gg 3$, 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:

```
1  0010001000010101
2  0100100000001000
3  0101010001100111
4  0001001010000100
5  0001000001111001
6  0000001111111011
7  0001000001111001
8  0000100000000001
9  0001001001111001
10 1111000000100101
11 0101010010100000
12 0101011011100000
13 0101100100100000
14 0001010010100001
```

15	0001011011101000
16	0101101011000001
17	0000010000000001
18	0001100010000100
19	0001010010000010
20	0001011011000011
21	0000101111111010
22	1100000111000000
23	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 $288 \equiv 1(\text{mod } 7)$, 程序执行正确.