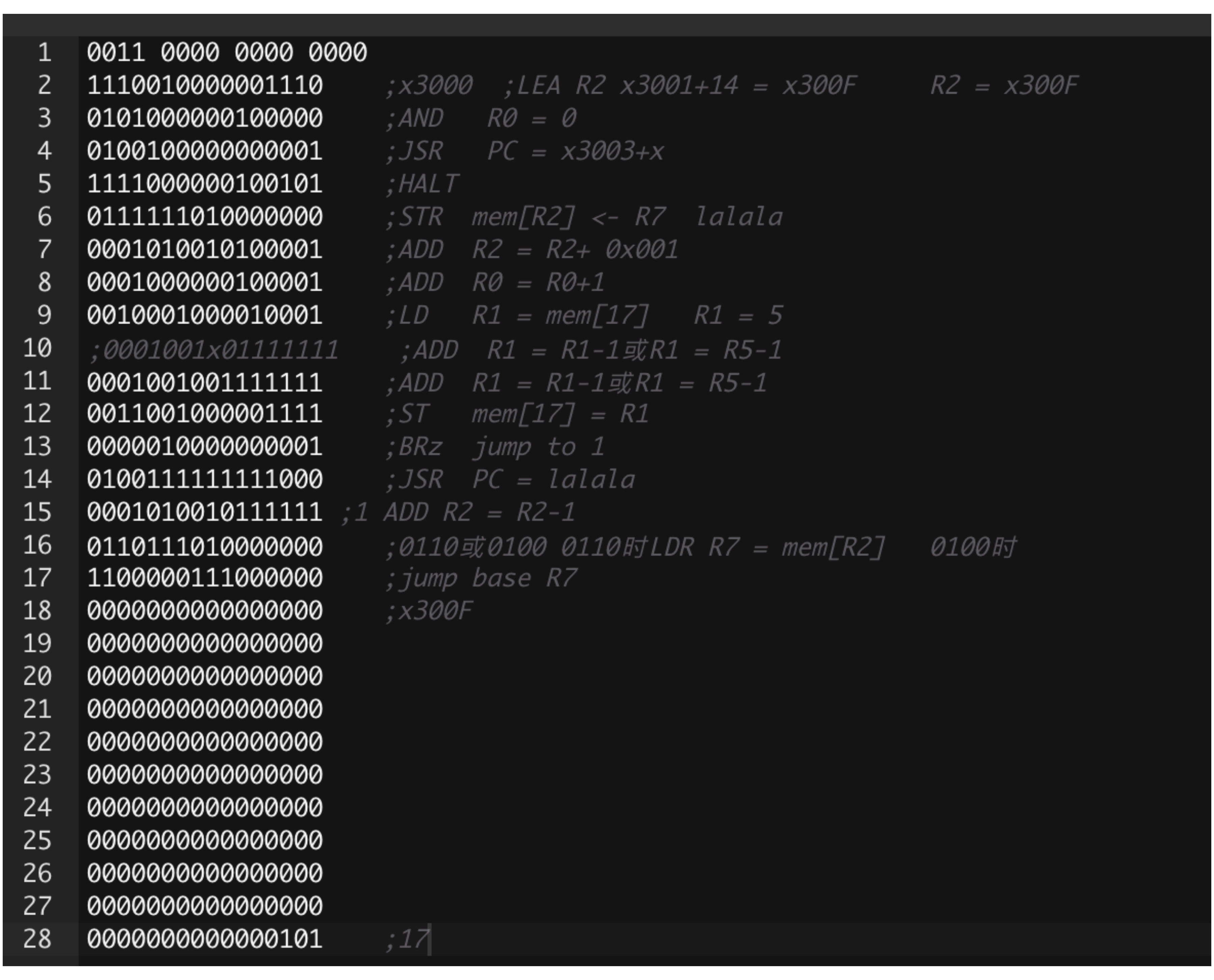
Lab4实验报告

task1

本次实验的任务一是补全机器码，下面是助教给出的有未知数字的机器码：

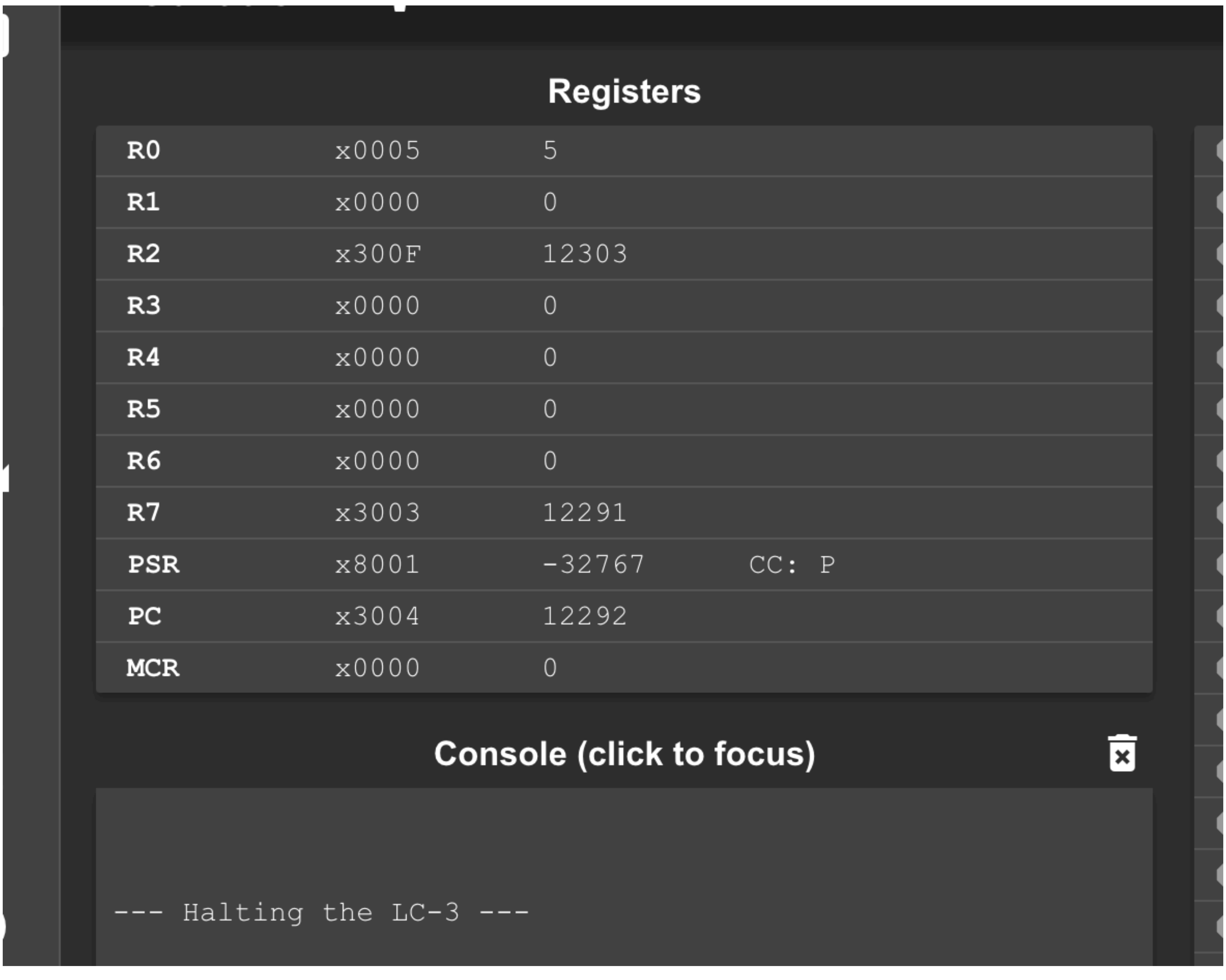
```
1110010000001110
0101000000100000
010010000000000x
1111000000100101
0111111010000000
000101001010x001
0001000000100001
0010001000010001
0001001x01111111
0011001000001111
0000010000000001
0100111111111000
0001010010111111
01x0111010000000
1100000111000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000101
```

我们利用LC3，先将代码翻译，如下所示：



```
0011 0000 0000 0000
1110010000001110    ;x3000  ;LEA R2 x3001+14 = x300F     R2 = x300F
0101000000100000    ;AND   R0 = 0
0100100000000001    ;JSR   PC = x3003+x
1111000000100101    ;HALT
0111111010000000    ;STR  mem[R2] <- R7  lalala
0001010010100001    ;ADD  R2 = R2+ 0x001
0001000000100001    ;ADD  R0 = R0+1
0010001000010001    ;LD   R1 = mem[17]   R1 = 5
;0001001x01111111    ;ADD  R1 = R1-1或R1 = R5-1
0001001001111111    ;ADD  R1 = R1-1或R1 = R5-1
0011001000001111    ;ST   mem[17] = R1
0000010000000001    ;BRz  jump to 1
0100111111111000    ;JSR  PC = lalala
0001010010111111 ;1 ADD R2 = R2-1
0110111010000000    ;0110或0100 0110时LDR R7 = mem[R2]   0100时
1100000111000000    ;jump base R7
0000000000000000    ;x300F
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000101    ;17
```

我们跟据代码的分析来确定程序的原始代码。 首先，对于第三行的JSR命令来说，最后一位x一定是1，如果它是0的话，会直接跳转到HALT指令处，这样之后的程序会毫无意义。而对于第二个未知的代码命令，首先我们分析，它是一条ADD指令，其实现的结果为 R2 = R2+0x001，我们猜测在这里，x为0的可能性很大，故先假设x为0，之后在整体分析整个代码的功能。而对于第三处来说，我们翻译得到这是一条ADD指令，其结果根据x的取值意思分别R1 = R1-1和R1 = R5-1，而对于后者，由于初始时寄存器置0，整个代码中未出现过给R5赋值的场景，所以我们可以得到该部分代码应该为前者。现在我们需要解决的就是最后一个代码命令，我们首先翻译这段命令，根据x的取值，该部分代码可以被翻译成LDR指令或者是JSR指令。我们根据两种情况都试一此，得到结果如下图所示：



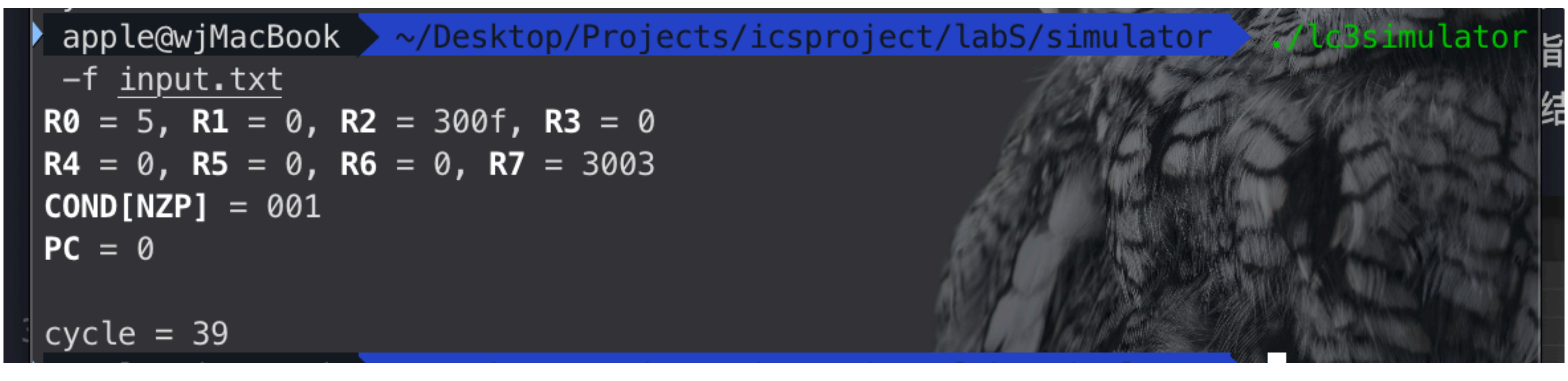
Registers

R0	x0005	5	
R1	x0000	0	
R2	x300F	12303	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x3003	12291	
PSR	x8001	-32767	CC: P
PC	x3004	12292	
MCR	x0000	0	

Console (click to focus)

```
--- Halting the LC-3 ---
```

将该代码在Labs的模拟器上运行结果如下所示：



```
apple@wjMacBook  ~/Desktop/Projects/icsproject/labS/simulator  ./lc3simulator
 -f input.txt
R0 = 5, R1 = 0, R2 = 300f, R3 = 0
R4 = 0, R5 = 0, R6 = 0, R7 = 3003
COND[NZP] = 001
PC = 0

cycle = 39
```

同样验证了该代码的正确性，最终得到代码如下所示：

```
1110010000001110
0101000000100000
0100100000000001
1111000000100101
0111111010000000
0001010010100001
0001000000100001
0010001000010001
0001001001111111
0011001000001111
0000010000000001
0100111111111000
0001010010111111
0110111010000000
1100000111000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000000
0000000000000101
```

task2

第二个任务同样是补全遗漏的代码，根据提示我们可以知道这个代码的目的是取余,即求7的余数，提示为"divide by 8"。下面是我们得到的不全的代码：

```
0010001000010101
0100100000001000
0101010001100111
0001001010000100
00010000xxx11001
00000011xxx11011
00010000xxx11001
0000100000000001
0001001001111001
1111000000100101
0101010010100000
0101011011100000
0101100100100000
0001010010100001
0001011011101000
0101101011000001
0000010000000001
0001100010000100
0001010010000010
0001xxx011000011
0000xxx111111010
1100000111000000
0000000100100000
```

然后我们对该部分代码进行翻译，得到翻译后的代码, 下面是我思考是的思路：

```
1 LD pc+21 => R1
2 JSR 8+3 => pc 3=> R7
3 AND R1 & offset => R2
4 ADD R2+R4 => R1
5 ADD Rx+11001(signextend) => R0
6 BRp 7+()=>pc
7 ADD Rx+11001=>R0
8 BRn 9+1 => pc
9 ADD R1+11001 => R1
10 HALT
11 AND 0 => R2
12 AND 0 => R3
13 AND 0 => R4
14 ADD R2+1 => R2   r2 = 1
15 ADD R3+8 => R3   r3 = 8
16 AND R1 & R3 => R5
17 BRz 18+1 => pc
18 ADD R2+R4 => R4
19 ADD R2+R2 => R2
20 ADD R3+R3 => Rx
21 BR xxx 22-6 => pc
22 JMP R7 => pc
```

之后我们再来分析代码的结构，我们根据提示，思路引向取余的方向。只要弄懂了算法的原理，就容易完成代码。我们首先分析HALT之后的代码部分，首先将各寄存器的值置为0，然后将R2与R3赋予初值，每次循环时都会进行对R1的某位的判断，若为1，则R4+R2，R2左移，R3左移。最后得到的应该是R1除以8取下整的结果。我们首先观察几个式子： $16\ mod\ 7 = 2,32\ mod\ 7 = 4,72\ mod\ 7 = 9,$ 故我们可以得到下列公式:、 $num = (8 * num)\ mod\ 7$ ，故这样就对应了我们之前得到的R1除以7求余数的过程。我们再由该算法可以比较容易的得到得到代码如下所示：

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

以上算法实现了除以7得到余数的算法过程。

实验总结

本次实验实验形式新颖，难度适中，但在不明白算法的意思下猜测代码难度较高，一开始有些摸不着头脑，浪费了许多的时间。不过在弄懂算法的意思后，特别是第二题提示明显，并且除以8这一提示可以帮助理解算法，得到一个目的明确的算法，题目完成起来要简单许多。