中国科学技术大学计算机学院
《计算机系统概论实验报告》

实验题目：reveal yourself

学生姓名：叶子昂

学生学号：PB20020586

完成时间：2021年12月23日

0.1 实验目的：

> 阅读代码理解并补充缺失的部分

0.2 实验环境：

- window11 和 wsl2
- VScode以及LC3TooL

0.3 实验过程：

0.3.1 task1

- 在任务1中丢失的第一行为0100100000000000x，这是一条JSR指令，它的下一行为HALT指令，x应该为1使得能够进入HALT后面的子程序。
- 丢失的第二行为0001 010 010 1 0x001结合第一行LEA 指令向R2中装入第一个空白行的地址，后面有R2递减1的操作，判断程序逐行操作的可能性较大，x猜测为0。
- 丢失的第三行为0001 001 x01 1 11111 ，根据指令格式x01为一寄存器，遍观程序未有其他地方使用R5，也未做BR指令判断所用，且最终结果R5依然为0，综合判断x为1。
- 丢失的第四行为01x0 111010000000，它可能为JSR(JSRR) 或LDR指令，若为JSR则程序会跳转至x3000以前这是错误的，故应改为LDR指令，x为1 。

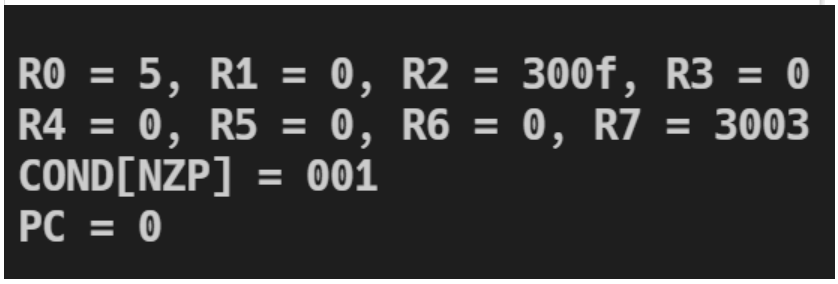
修复后代码如下：

```
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
```

将修改后的程序在LC3tool和LabS的simulator中运行结果如下：

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	32769	CC: P
PC	x3003	12291	
MCR	x0000	0	



结果一致且符合所给结果。

0.3.2 task2

根据实验提示及大致浏览代码，task2中代码应为对第23行的数字对七取余结果存放在R1中 。使用的方法为“divided by 8”具体思想是将求七的余数转化为求8的余数。具体方法是：

若$n = 8 \times k + m$

则$n = 7 \times k + m + k$

若$m + k$大于7，则令n=m+k，迭代直至m+k小于等于7。小于七m+k即为结果，等于7则结果为0 。

阅读程序可知，子程序是用以求$n = 8 \times k + m$中的k，主程序计算m同时计算$m + k$并判断是否需要继续迭代。

- 第一个缺失的为00010000xxx11001，它为ADD，根据上一行为0001 001 010 000 100;R1=R2+R4。可知这里为R0=R1-7，从而xxx为011。
- 第二个缺失的为00000011xxx11011，它为BRP,若上一行的R0=R1-7大于0则应继续迭代，从而它跳转的目的地为JSR指令，则xxx为111。
- 第三个缺失的为00010000xxx11001，它为ADD，根据主程序计算$m + k$并判断是否需要继续迭代的功能。可知这里依然为R0=R1-7，设置条件码供下一行BR指令使用，从而xxx为011。
- 第四个缺失的为0001xxx011000011，遍观整个子程序求$n = 8 \times k + m$中的k的方法为获得n的前13位（后三位为余数m）这通过R3检测，R2表示当前位数，R4存储结果实现，主要还是使用掩码的方法，故此处为R3<<1，xxx为011 。

- 第五个缺失的为0000xxx111111010，该句用以判断是否检测完毕，当R3移至1000000000000000，即以检测完毕应当跳出，当然继续检测也可以也可移至0000000000000000，即判断R3大于0或R3不等于0继续循环均可故xxx为001或101。

修复后代码如下：

```
0010 001 000010101;21;288
0100 1 00000001000
0101 010 001 1 00111;mod/8
0001 001 010 000 100;R1=R2+R4
0001 000 001 1 11001;R0=R1-7
0000 001 111111011;P T JSR/
0001 000 001 1 11001;R0=R1-7
0000 100 000000001;N T HALT
0001 001 001 1 11001;R1=R1-7
1111000000100101;HALT
0101 010 010 1 00000;R2=0//TO
0101 011 011 1 00000;R3=0
0101 100 100 1 00000;R4=0
0001 010 010 1 00001;R2+=1
0001 011 011 1 01000;R3+=8
0101 101 011 000 001;R5=R3&R1
0000 010 000000001;Z T NEXT
0001 100 010 000 100;R4=R2+R4
0001 010 010 000 010;R2<<
0001 011 011 000 011;R3<<
0000 001 111111010;TO ZNEXT
1100 000 111 000000;RET
0000000100100000
```

将修改后的程序在LC3tool和LabS的simulator中运行结果如下：

R0	xFFFA	65530	
R1	x0001	1	
R2	x0000	0	
R3	x8000	32768	
R4	x0001	1	
R5	x0000	0	
R6	x0000	0	
R7	x3002	12290	
PSR	x8004	32772	CC: N
PC	x3009	12297	
MCR	x0000	0	

```
R0 = fffa, R1 = 1, R2 = 0, R3 = 8000
R4 = 1, R5 = 0, R6 = 0, R7 = 3002
COND[NZP] = 100
PC = 0
```

结果一致且符合288%7=1。

0.4 实验总结与思考：

- 本次实验主要关于子程序的使用与阅读，熟悉各种指令格式，能根据较完整的代码理解其功能和原理，并最终补全代码缺失的部分。
- 通过本次实验，我熟悉了子程序的使用（调用和返回），进一步熟悉LC3指令的格式和用途，提高了阅读LC3代码的能力，了解了LC3对非2^n类数取余数的方法。