


Lab2 Report

522030910109 刘翰文

1. Modular Operation

为了实现给计算机添加取余操作 $A \% B$ ，需要在 Main Algorithm 部分加上取余符号的判断以及判断成功之后的取余操作。对于取余操作，它的主要流程是：首先从栈顶 POP 出两个数，若两次 POP 均成功则判断 A 的符号，若 $A < 0$ ，则重复 $A = A + B$ 直到 $A \geq 0$ ，此时的 A 便是所要求的结果。若 $A \geq 0$ ，则先将 $B = -B$ ，在重复 $A = A + B$ 直到 $A < 0$ ，此时 $A - B$ 便是所要求的结果。得到结果之后将结果存入 R_0 后判断范围，若范围合理则 PUSH 回栈并输出，否则将 A、B 重新放入栈中，完成取余的操作。

以下是进行取余操作的实际演示图：



```
LC3 Console
Enter a command:16
+016
Enter a command:5
+005
Enter a command:%
+001
Enter a command:
```

本次计算了 $16 \% 5$ ，可以从结果看到得到了 1 这个正确答案，接下来计算 $-16 \% 5$ ：



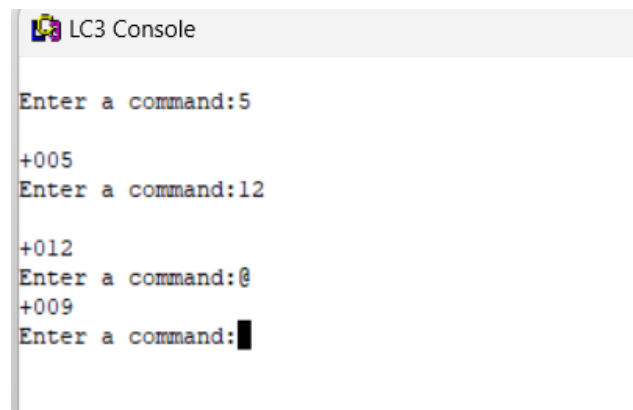
```
LC3 Console
Enter a command:16
+016
Enter a command:-
-016
Enter a command:5
+005
Enter a command:%
+004
Enter a command:
```

可以从结果看到计算得到了 4，和正确答案一致。

2. XOR Operation

与取余操作类似，为了实现异或操作 $A \oplus B$ ，需要在 Main Algorithm 部分加上取余符号的判断以及判断成功之后的异或操作。对于异或操作，它的主要流程是：首先从栈顶 POP 出两个数，若两次 POP 均成功则先初始化一个寄存器 R_2 为 1， R_3 为 0（存放结果），通过它与 A、B 进行按位与操作得到 A 和 B 最后一位的值 a，b，随后将 $a + b$ 的结果与 R_2 的值相加，将得到的结果与 R_3 相加，最后将 R_2 的值翻倍。重复以上过程（从用 R_2 与 A 和 B 进行按位与操作开始）直到取遍 A、B 的每一位为止（通过判断 R_2 是否溢出变回 0 来判断）， R_3 中的值便是所要求的结果。最后将 R_3 的值存入 R_0 后判断范围，若范围合理则 PUSH 回栈并输出，否则将 A、B 重新放入栈中，完成异或的操作。

以下是进行异或操作的实际演示图：



```
LC3 Console

Enter a command:5
+005
Enter a command:12
+012
Enter a command:@
+009
Enter a command:
```

本次是计算 $5 \oplus 12$ ，用二进制表示即为 $0101 \oplus 1100$ ，异或的正确结果应为 1001 ：9，得到了正确答案。