**ICS Lab0 Report**

虞佳焕

PB17121687

1. **算法设计**

以4位有符号数为例说明，16位有符号数只需要进行简单拓展。

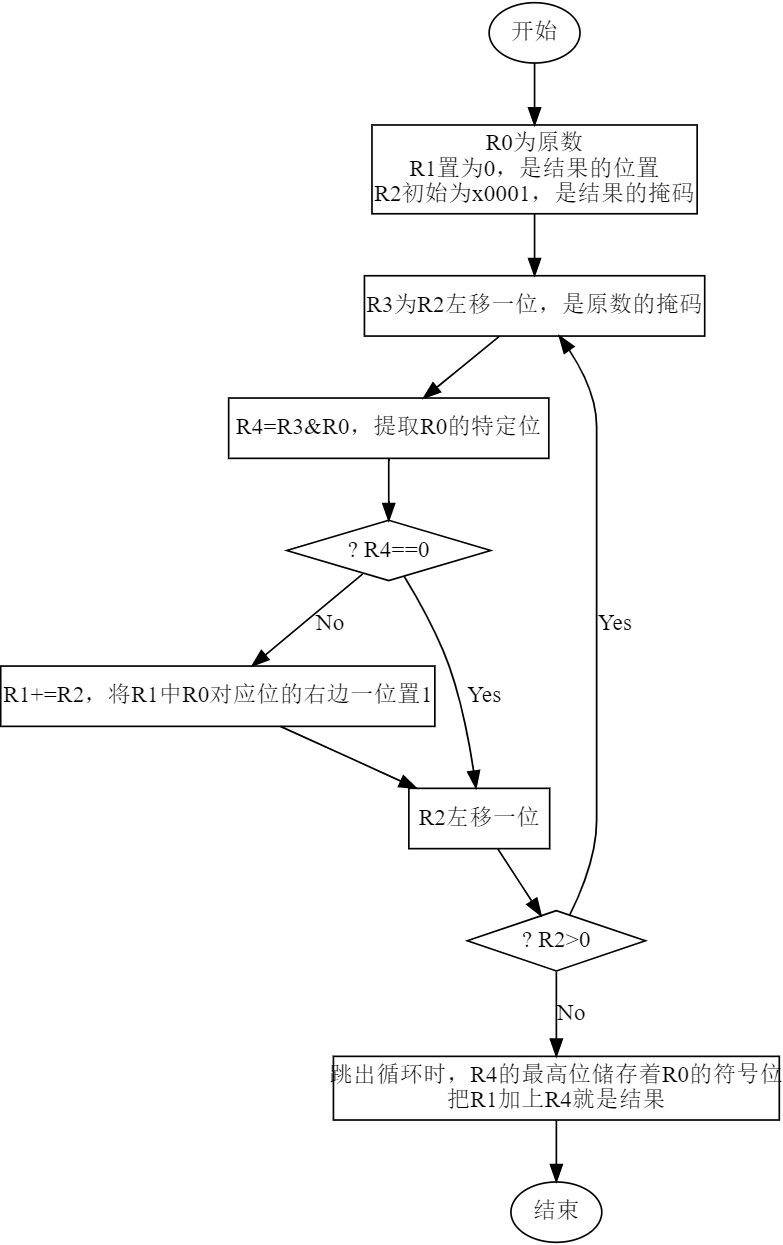
设4位有符号数为 ，右移后变为 ，可以利用 的掩码功能，设置一个只有第i位为1的原数掩码和一个只有第i-1位为1的结果掩码。

例如对于原数 ，原数掩码 ，结果 初始置为0，结果掩码 。计算 即可判断 是否为1，如果 为1，则把 加上 ，效果等价于把 置为与 相同。

在程序中把结果掩码 初始置为1，每次循环都左移一位，直到为1000时跳出循环。

对于符号位 ，只需单独加上即可。

1. **编程**
   1. **根据以上算法，可绘制以下算法流程图**：



* 1. **用LC-3汇编语言实现以上流程图：**

AND R1, R1, #0 ;R1置为0，是结果的储存位置

ADD R2, R1, #1 ;R2初始置为1，是R1的掩码

;循环开始

LOOP ADD R3, R2, R2 ;R3为R2左移一位，是R0的掩码

AND R4, R0, R3 ;R4为提取R0的特定位

BRz #1 ;R0此位为0时不需要进行操作

ADD R1, R1, R2 ;R0此位为1时，用掩码将R1对应位的右边一位置为1

ADD R2, R2, R2 ;R2左移

BRp LOOP ;R2为x8000时跳出循环

;跳出循环时，R4储存着R0的符号位

ADD R0, R1, R4 ;补上R1最左侧的符号位

HALT ;结束

* 1. **将汇编语言翻译成指令和机器码：**

十六进制 二进制 指令  
x5260 0101001001100000 AND R1, R1, #0  
x1461 0001010001100001 ADD R2, R1, #1  
x1682 0001011010000010 ADD R3, R2, R2  
x5803 0101100000000011 AND R4, R0, R3  
x0401 0000010000000001 BRz #1  
x1242 0001001001000010 ADD R1, R1, R2  
x1482 0001010010000010 ADD R2, R2, R2  
x03FA 0000001111111010 BRp #-6  
x1044 0001000001000100 ADD R0, R1, R4  
xF025 1111000000100101 TRAP x25

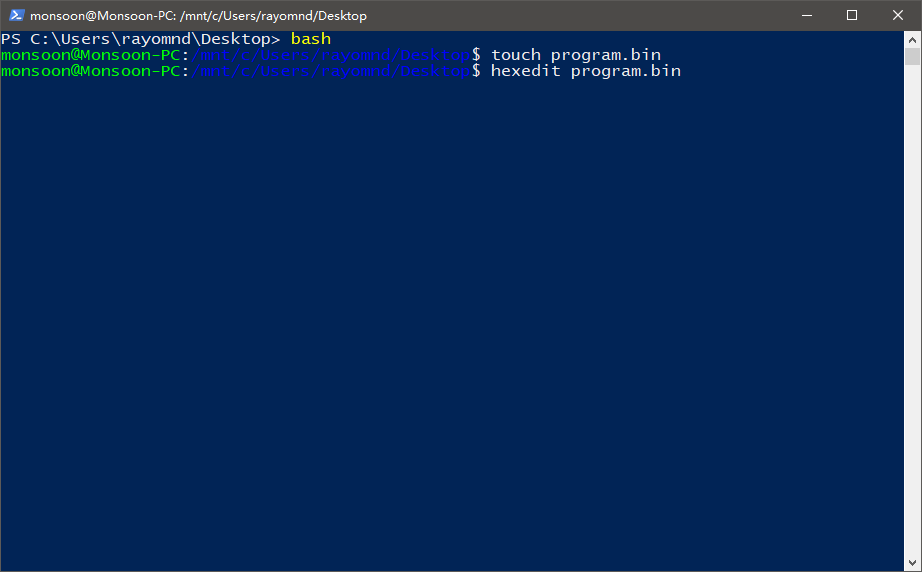
* 1. **使用hexedit编写二进制文件program.bin**

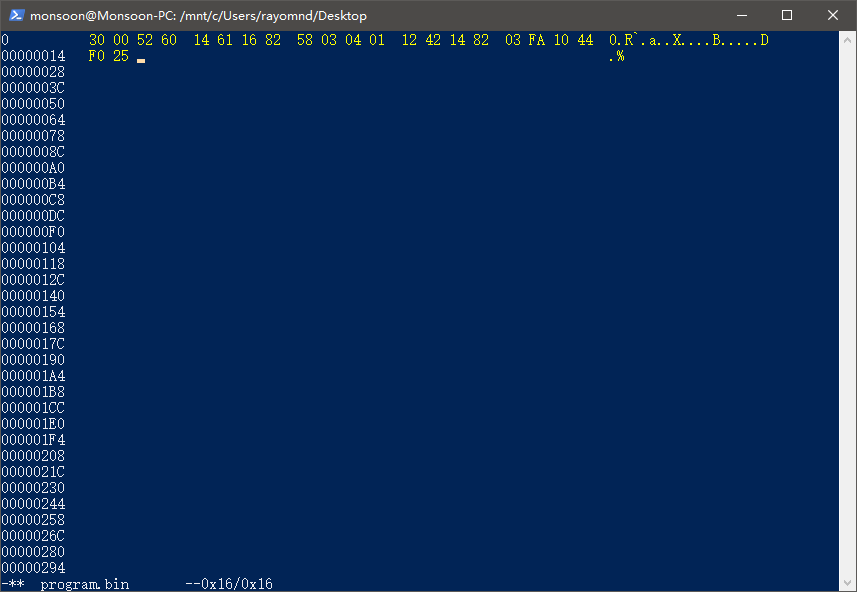
由于需要将程序加载到LC3 Simulate的x3000位置开始执行，program.bin的前两个字节应为x3000，使其符合LC3 Simulator的obj文件格式。

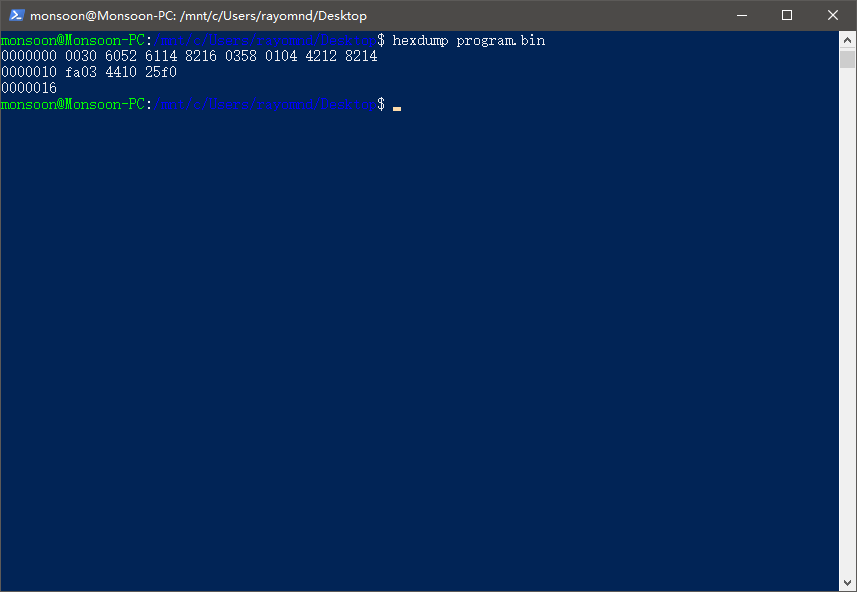
需要写入的内容为：

x3000  
x5260  
x1461  
x1682  
x5803  
x0401  
x1242  
x1482  
x03FA  
x1044  
xF025

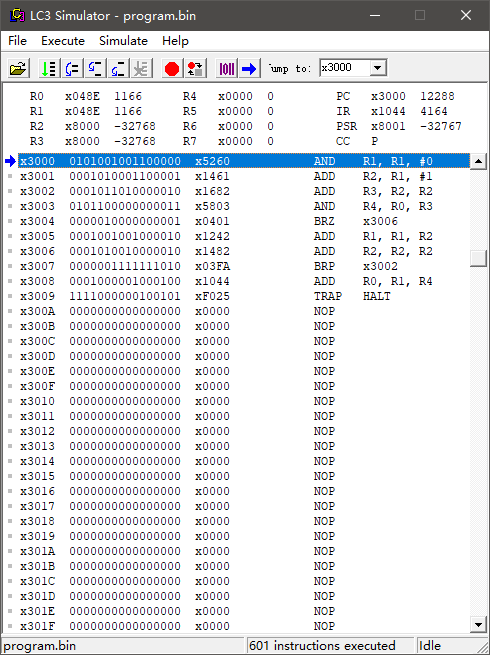
以下是写入过程的部分截图：







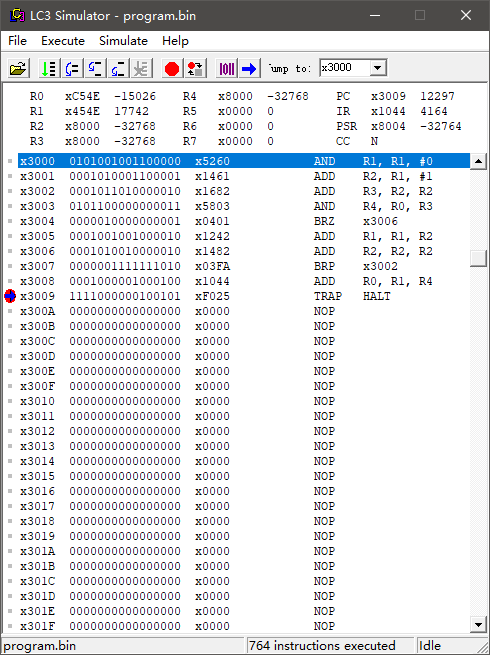
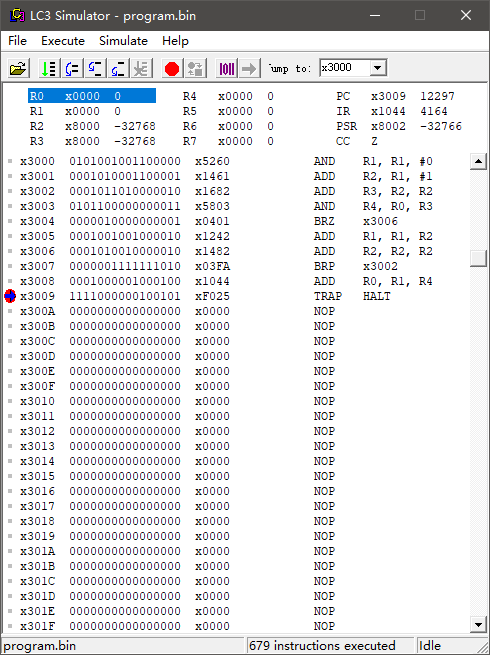
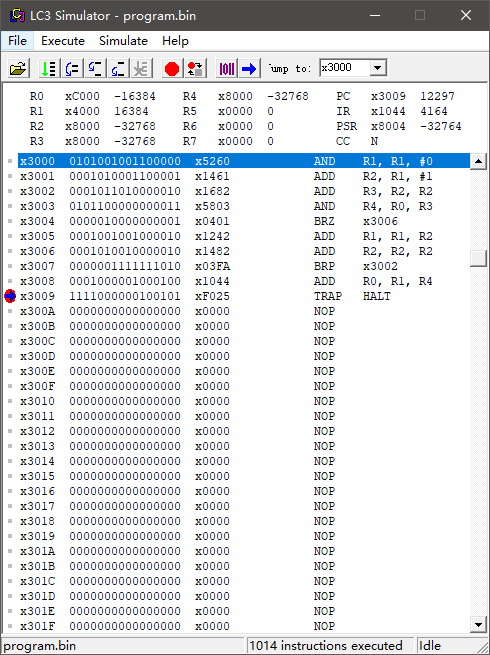
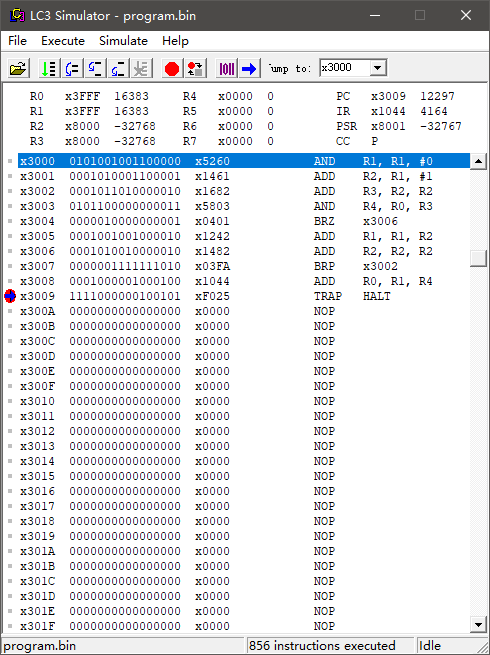
生成的文件能成功加载到LC3 Simulate。

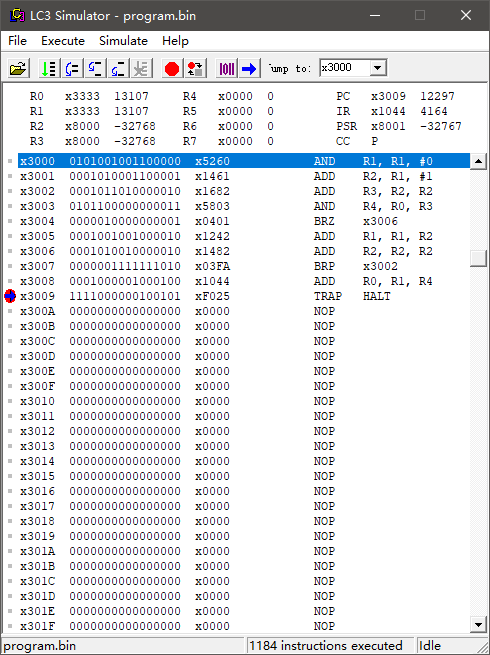
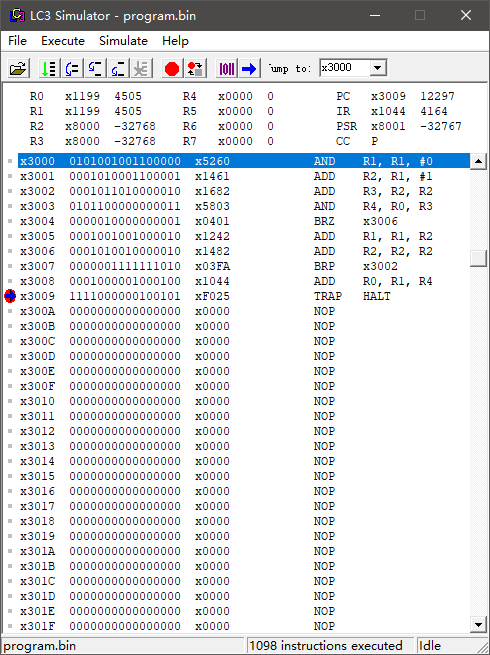


1. **测试**
   1. **个例测试**

选择一些具有代表性的随机值和一些边界值进行个例测试。选择的量中应包含0、边界值、正数和负数。

分别设置R0为x0000，x8A9C，x7FFF，x8000，x2333，x6666进行测试，结果应该分别为x0000，xC54E，x3FFF，xC000，x1199，x3333。



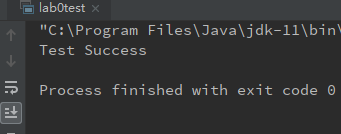
个例测试结果全部符合预期。

* 1. **自动测试**

使用Java语言编写程序模拟指令的执行，以下为测试代码：

public class Lab0Test {  
 public static void main(String args[]) {  
 short[] R = new short[8]; //8个寄存器  
 for (int i = Short.*MIN\_VALUE*; i <= Short.*MAX\_VALUE*; i++) {  
 R[0] = (short) i;  
 //被测试代码开始  
 R[1] = (short) (R[1] & 0); //AND R1, R1, #0  
 R[2] = (short) (R[1] + 1); //ADD R2, R1, #1  
 do {  
 R[3] = (short) (R[2] + R[2]); //ADD R3, R2, R2  
 R[4] = (short) (R[0] & R[3]); //AND R4, R0, R3  
 if (R[4] != 0) //BRz #1  
 R[1] = (short) (R[1] + R[2]); //ADD R1, R1, R2  
 R[2] = (short) (R[2] + R[2]); //ADD R2, R2, R2  
 } while (R[2] > 0); //BRp LOOP  
 R[0] = (short) (R[1] + R[4]); //ADD R0, R1, R4  
 //被测试代码结束  
 if (R[0] != (i >> 1)) {  
 System.*out*.println("Test Failed");  
 return;  
 }  
 }  
 System.*out*.println("Test Success");  
 }  
}

运行结果如下：



程序执行结果符合预期。