lab1:实现LC3乘法

郑子涵　　PB20000248

实验要求

- 两个运算数分别储存到**R1**和**R0**,其结果储存到**R7**，其他寄存器的结束状态不做要求。
- 初始状态：除**R1**和**R0**外的所有寄存器都为0。
- **l**部分：写出实现乘法的同时，尽可能少的机械码指令***行数***。
- **p**部分：写出实现乘法的同时，执行指令***条数***的次数最少。
- 要求实现与C语言的**short**型变量乘法结果相同。

设计实现

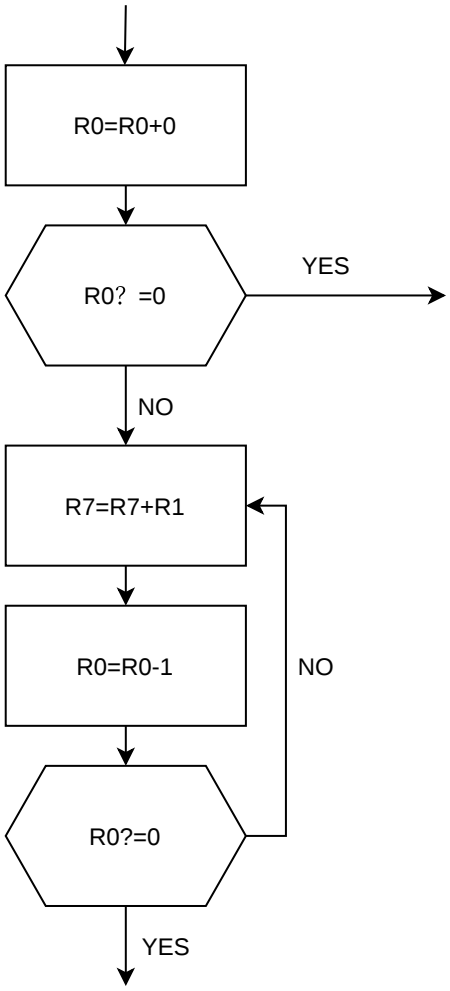
l部分

设计思路

初始

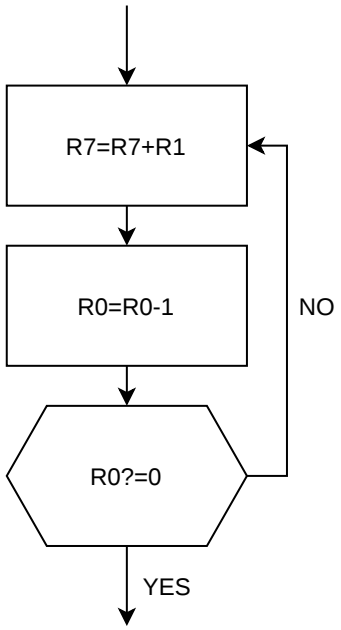
因为LC3机械可以实现加法，所以实现乘法的最直接的想法就是把乘法转化为多次加法来实现

根据这个思想可以画出流程图如下：



最终

根据观察，不需要最开始判断R0是否为0，即上述流程图中两个判断`R0? =0`是一样的，所以最终简化流程图如下



具体机械码实现

```
0011 0000 0000 0000;start at x3000

;my programe
0101 111 111 1 00000;清零R7

0001 111 111 0 00 001;R7=R7+R1
0001 000 000 1 11111;R0--
0000 101 111111101;如果R0不等于0，跳到R7=R7+R1
1111 0000 00100101;halt
```

总结

- 最开始是考虑把正负数分开来考虑，但发现按照补码的定义(负数的补码是数值位取反后加一)，最后都会因为进位而不会被保存下来，所以**有符号数和无符号数的乘法是一样的**，但初始版本还是考虑了有一位数是0的情况，在最终版本中发现0的情况也会最终因为最高位进位溢出而没影响。
- 最初的核心代码为5行。
- 最终得到的机械码如上，用了**3**行核心代码。

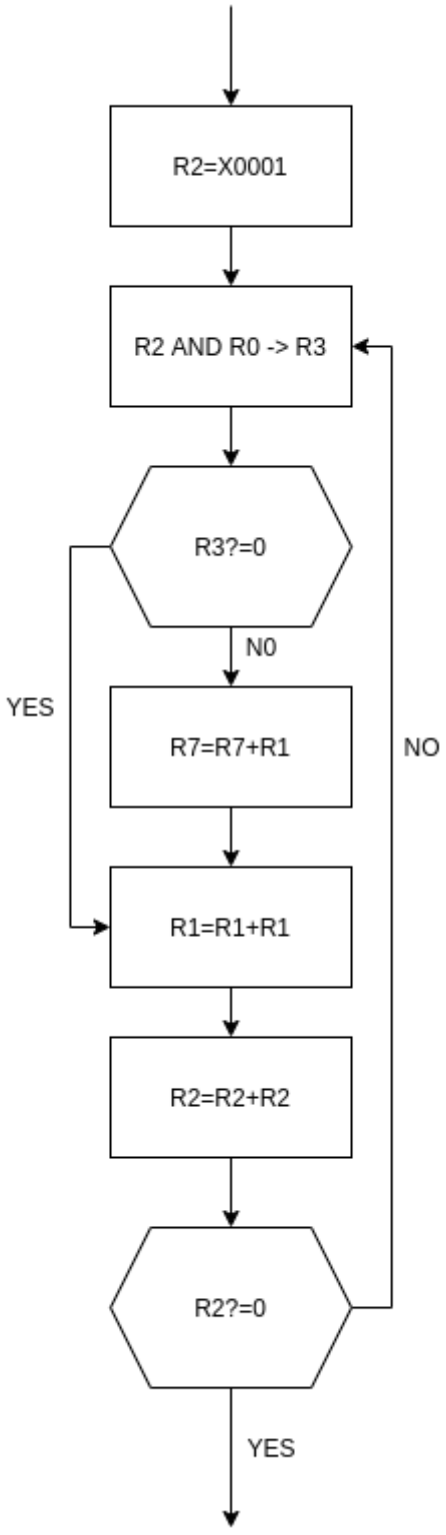
P部分

设计思路

初始

乘法除了可以改为加法来实现外，还可以通过结合**移位[^左移可以通过自己加自己来实现]和加法**来实现，就像平时10进制列的式子一样。我们可以从低位到高位依次取R0的值，如果是1的话，就加上对应的移位后的R1，如果是0则不加（要求每次R0取下一位前都要对R1进行移位到对应的权值），一直到取到最高位为止。

流程图如下



最终

可以发现最后的一个BR语句可以不用，循环可以分开来写，这样可以省去BR语句，但命令的条数会跟多。

具体机械码实现

```
;start the programe at x3000
0011 0000 0000 0000



;my programe
0101 111 111 1 00000;清零R7
0101 010 010 1 00000;清零R2
0101 011 011 1 00000;清零R3
0001 010 010 1 00001;设置R2为x0001
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
```

```
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 R1=R1+R1
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0101 011 000 0 00 010;R2 AND R0-> R3
0000 010 000000001;如果R3=0则跳到 halt
0001 111 111 0 00 001;R7=R7+R1
1111 0000 00100101;halt
```

总结

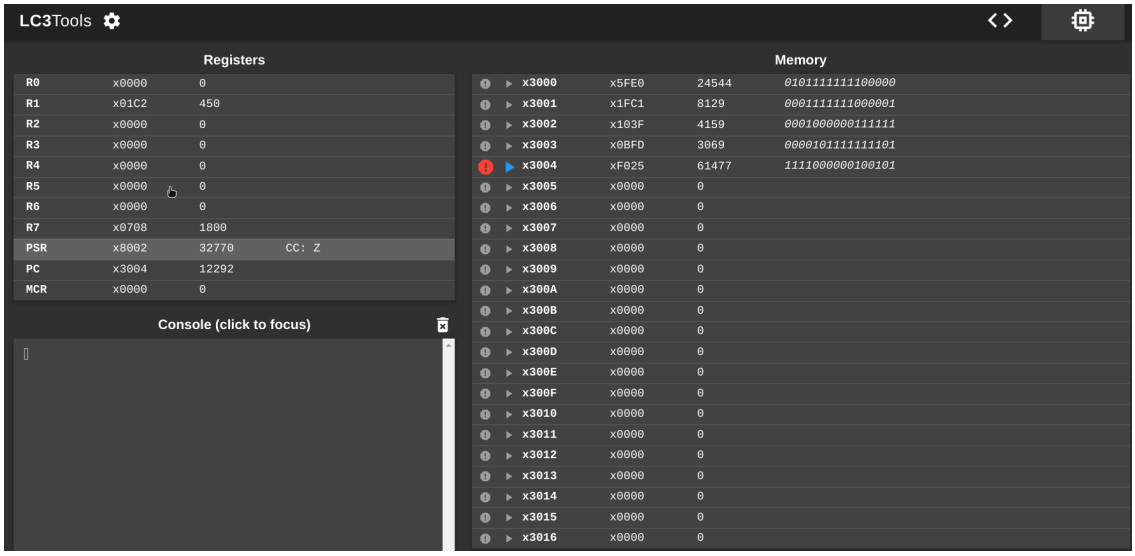
- 因为移位所需要操作数与数值的大小无关，所以选择移位操作使得操作的次数最小，l部分的时间复杂度为O（n）[^n为R0的大小] 。
- 在核心语句中（从R2=X0001开始），最长的执行条数（即从头执行到halt前一条）为79条，最短为63条。所以平均执行的命令条数约为**71**条。
- 优化思路：可以把一些情况分开来讨论，以减少特殊情况的命令执行数，如：把其中有个数为0的情况分开来描述。

实验结果

- L部分最终用了**3**行核心代码。
- P部分最终平均执行了**71**条核心语句。

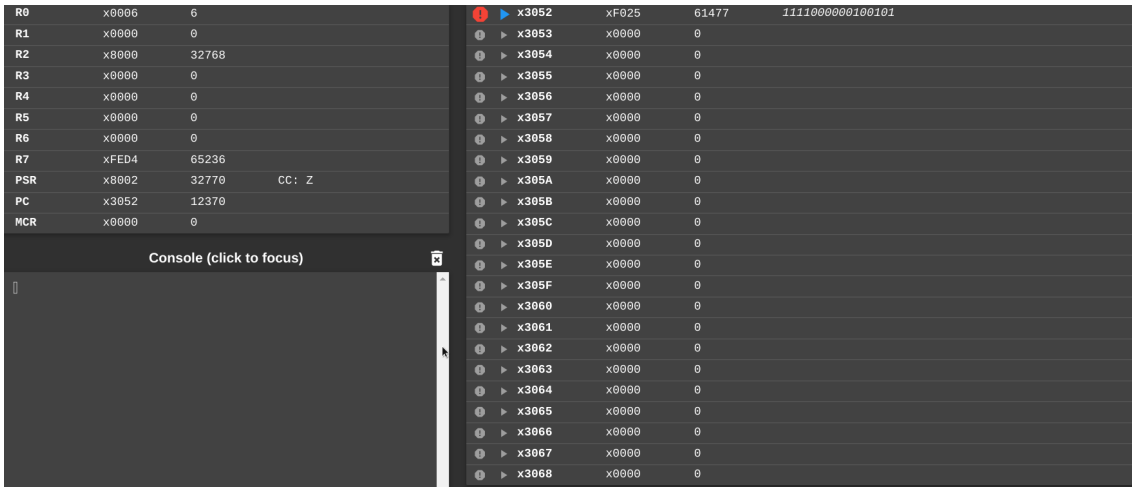
样例测试

- 4 * 450(L部分)



结果正确，且按照算法R0清为0停止，R1不会改变，过程也正确

- 6 * -50（P部分）



结果正确，xFED4对应的数值为-300，R0不变，R1会因为移位而变成0，符合过程