lab1:实现LC3乘法

郑子涵　　PB20000248

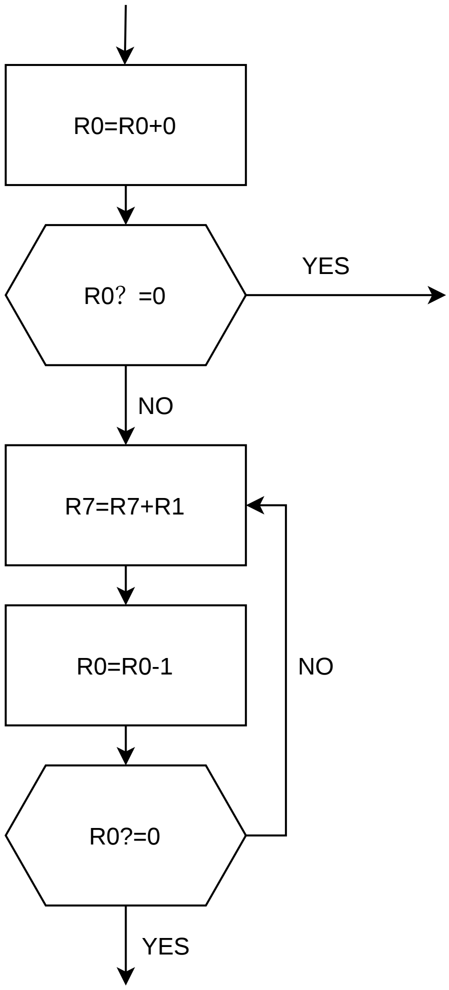
实验要求

- 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

> 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：
>
> 
>

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 000 000 1 00000;R0=R0+0
> 0000 010 000000011;如果等于R0=0，跳到X3006
> 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x3003
> 1111 0000 00100101;halt
> ```

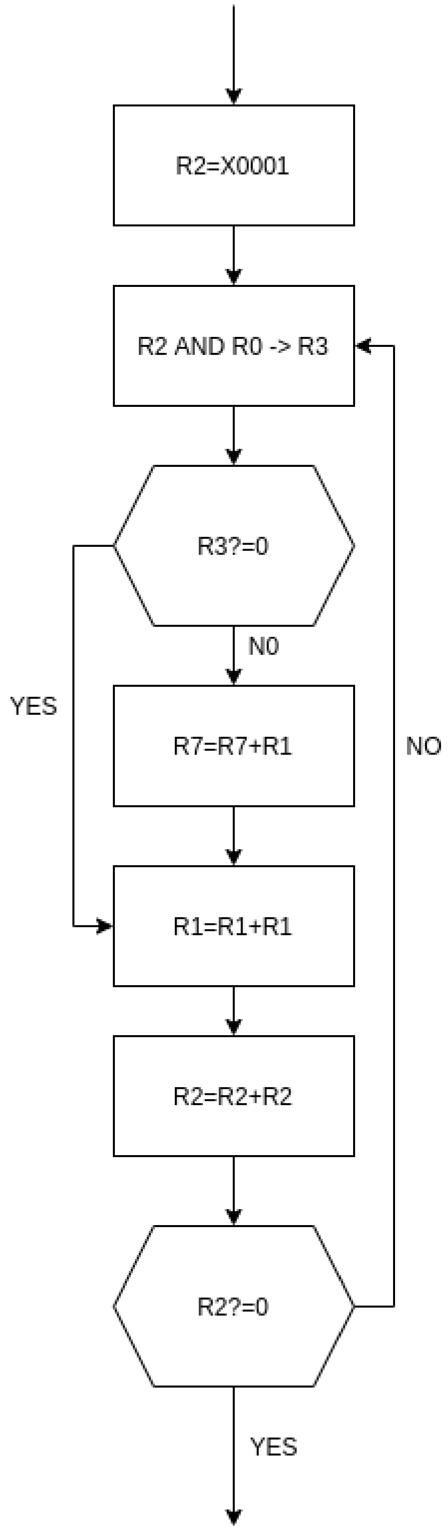
总结

- 最初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**。
- 最终得到的机械码如上，用了**5**行。

P部分

设计思路

> 乘法除了可以改为加法来实现外，还可以通过结合**移位[^左移可以通过加自身来实现]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



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x3007
```

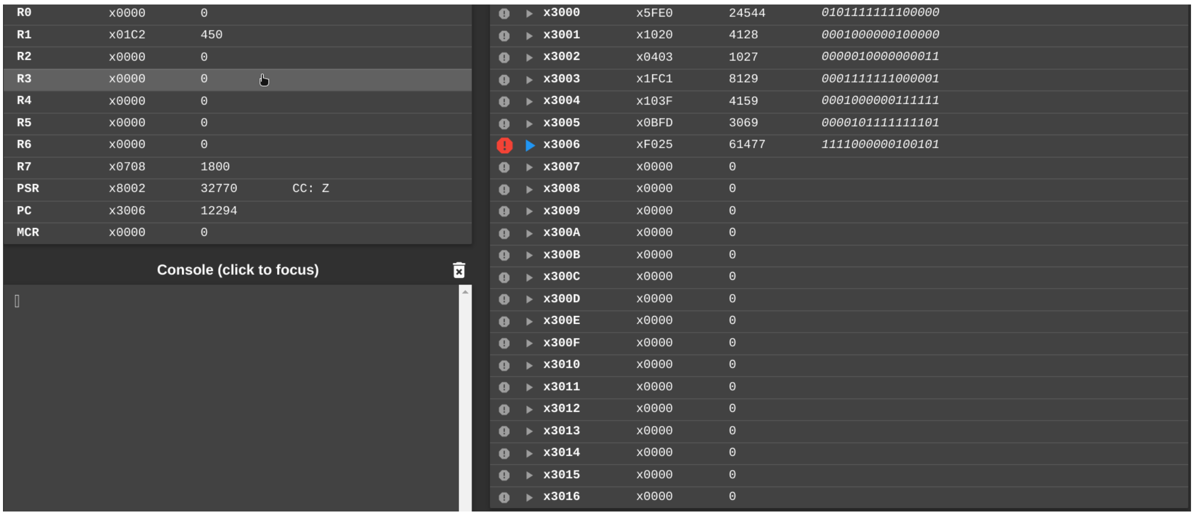
```
0001 111 111 0 00 001;R7=R7+R1
0001 001 001 0 00 001;R1=R1+R1(左移一位)
0001 010 010 0 00 010;R2=R2+R2(左移一位)
0000 101 111111010;如果R2!=0, 则跳到x3004
1111 0000 00100101;halt
```

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]，最开始便想到移位，这便是初始版本。
- R2经过16次循环后会变为0二这个循环大小最大为6个指令，再加上开始的赋值指令，所以max=6 * 16 + 1=97，最多会用97次指令，循环最小为5个指令，所以min=5 * 16 + 1=81，平均而言会用89条指令。

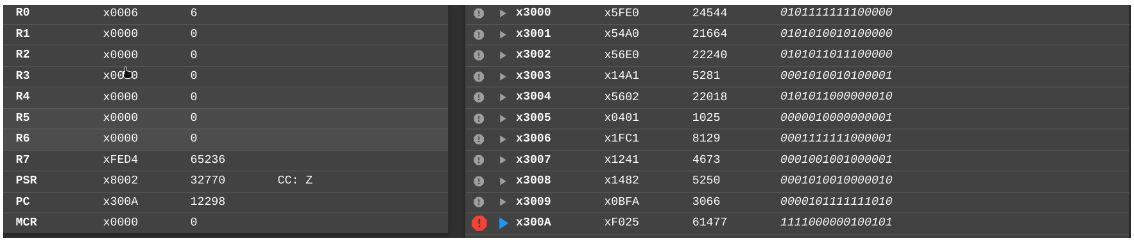
样例测试

- 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