Lab1实验：实现LC-3乘法运算

姓名：杨涛
学号：PB20020599

正确性检验方法

LC-3寄存器能够存储16bits的数据，即为2bytes的整型数据，与C语言中的short型整型的行为完全相同，故在C语言中编写，使用gcc编译运算并输出乘数与结果的**位模式**的十六进制表示，并将其乘数结果输入LC3Tools，经过运行后对结果进行比对则可校验结果是否正确.
在这里将C语言代码与样例的运行结果呈现出来.

```c
#include<stdio.h>
int main(){
    short a=-114,b=-233,c=a*b;//a,b为第一个与第二个乘数
    printf("%04X\n",(unsigned short)a);
    printf("%04X\n",(unsigned short)b);
    printf("%04X\n",(unsigned short)c);
    return 0;
}
```

原乘法	乘数1	乘数2	结果
1 * 1	x0001	x0001	x0001
5 * 4000	x0005	x0FA0	x4E20
4000 * 5	x0FA0	x0005	x4E20
-500 * 433	xFE0C	x01B1	xB24C
-114 * -233	xFF8E	xFF17	x67C2

L版本程序设计

起初就确定方向就是简单求和，利用loop最终达到乘法效果，即使是乘以负数根据溢出的性质可知最终并不会影响结果正确性，故大胆将0作为最后的终止条件.
最初是按照while的思路进行设计，即类似C语言代码：

```
c=a;
while(b!=0)
{
    b--;
    a+=c;
}
```

但是LC3汇编实现while需要至少两个跳转指令，这显然不合适.
考虑到do...while的实现只需要一个跳转指令，则得到最终设计.
但这样差别在于b=0时结果是否会有差异，发现根据数学证明即使不需要特殊考虑，最后结果也是为0，由此得到最终版本.

核心代码：

```
LOOP ADD R7, R0, R7
ADD R1, R1, #-1
BRnp LOOP
```

对应机器码

```
0001111000000111
0001001001111111
0000101111111101
```

按照while思路则需要多1行，即需要4行，而最终设计按照do...while则需要3行.

LC3Tools上的调试代码：

```
.ORIG x3000 ; start the program at location x3000

LOOP ADD R7, R0, R7
ADD R1, R1, #-1
BRnp LOOP

HALT ; halt
.END
```

由于是验证实验，结果与上面正确性所给出的结果相同，故不再给出实验结果.

周期数测试方法

初始化一个计数器，根据每个阶段指令的数量给计数器进行累加，最后计数器内存储的值即为运行指令数.
需要特别注意的是计算计数时不应干扰条件码的判断，因此计数位置应在设置条件码的指令之前.

P版本程序设计

核心思路是仿照竖式乘法运算，a的第i位对应$b \times 2^i$，由此相加则得到结果.
一开始是想用右移来实现a的第i位是否为1的判断，但LC-3中没有移位的操作，但有ADD来间接实现左移，故使用一个掩码x0001不断左移并AND来实现每一位的判断.

核心代码：

```
ADD R6,R6,x1;            R6置1用于AND判断
LOOP2 AND R2,R1,R6;      用于判断R6的对应位是否为1，只用于设置状态码，存储无用
BRz LOOP1;               为0的时候跳过相加
ADD R7,R7,R0;            对应位置为1，加上R0*2^i
LOOP1 ADD R0,R0,R0;      R0*2,左移一位
ADD R6,R6,R6;            R6左移一位，掩码位置向左移动
BRnp LOOP2;              R6非零，继续计算相加
```

对应机械码：

```
0001110110100001
0101010001000110
0000010000000001
0001111111000000
0001000000000000
0001110110000110
0000101111111010
```

乘法调试代码：

```
.ORIG x3000 ;        start the program at location x3000

ADD R6,R6,x1
LOOP2 AND R2,R1,R6
BRz LOOP1
ADD R7,R7,R0
LOOP1 ADD R0,R0,R0
ADD R6,R6,R6
BRnp LOOP2

HALT ;               halt
.END
```

加上计数后：

```
.ORIG x3000 ;         start the program at location x3000

ADD R6,R6,x1
ADD R5,R5,x1;         R5为指令计数器

LOOP2 ADD R5,R5,x2;    ---横杆和箭头显示了该计数器所累加指令数的指令范围
AND R2,R1,R6;           |
BRz LOOP1;              ↓

ADD R7,R7,R0;           ↑
ADD R5,R5,x1;          ---

LOOP1 ADD R0,R0,R0;     ↑
ADD R5,R5,x3;          ---
ADD R6,R6,R6;           |
BRnp LOOP2;             ↓

HALT ;                halt
.END
```

下面是样例所测的R7的存储值，即对应的指令数量

乘法	指令数
1 * 1	82
5 * 4000	87
4000 * 5	83
-500 * 433	86
-114 * -233	93

平均指令数$\frac{82+87+83+86+93}{5}=86.2$，直接分析程序，可知每次循环指令数的差别在于下面这个指令是否被执行

```
ADD R7,R7,R0
```

故每组不同输入可能差别执行该指令0次到16次，分析测试指令数可知总指令数范围在$[81,97]$.

此即为最终版本，故无程序最初运行案例的耗时.