ICS LAB_01 乘法器的实现 实验报告

L版本

代码实现（机器码书写，通过汇编语言加C风格注释辅助说明）

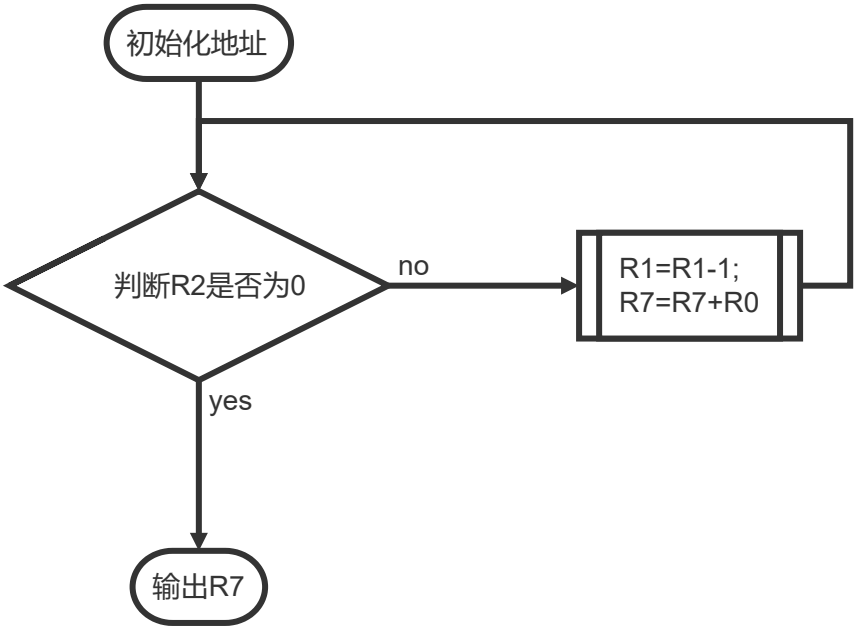
```
0001010001100000;
0000010000000010;
0001111000001001;
0001001001111111;
0000101111111101;

.orig x3000 //初始化地址
ADD R2,R1,#0; //判断R2是否为0
BRz ENDL; //为0则直接输出R7=0
LOOP ADD R7,R0,R7; //R7循环加R1个R0存储的值
ADD R1,R1,#-1; //R1为计数器
BRnp LOOP; //判断R1是否完成计数
ENDL HALT; //
.end //结束
```

设计思路

$$Sn = a1 + a2 + \ldots + an（a1 = a2 = \ldots = an）$$

从最原始的数列求和公式出发，数的乘法实现就是乘数个被乘数相加。由于补码中负数的特性，一个负数的补码减一恰好是绝对值小它1的数的补码值，所以我们根据上述求和公式可以设计出以下流程图。



代码优化

初始代码只包括3行

```
0000010000000010;
0001111000001001;
0001001001111111;
```

显然初始时并没有对乘数等于0这一情况进行检验，所以代码正确性是不足的，经过修改得到上文所述最终版本，通过增加寄存器R2来判断是否可以直接输出R7为0。同时，为了实现"L"的初衷，这里并没有对被乘数为0进行额外判断。

最终结果为5行机器码。

样例测试与正确性检验

本次正确性检验直接通过程序运行来执行

- **计算1*1(按运行前后状态排序)**

Registers			
R0	x0001	1	
R1	x0001	1	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x0000	0	
PSR	x8002	-32766	CC: Z
PC	x3000	12288	
MCR	x0000	0	

Registers			
R0	x0000	0	
R1	x7FFF	32767	
R2	x0001	1	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x2FFC	12284	
R7	x0001	1	
PSR	x0002	2	CC: Z
PC	x036C	876	
MCR	x0000	0	

- **计算 5 * 4000(按运行前后状态排序)**

Registers			
R0	x0005	5	
R1	x0FA0	4000	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x0000	0	
PSR	x8002	-32766	CC: Z
PC	x3000	12288	
MCR	x0000	0	

Registers			
R0	x0000	0	
R1	x7FFF	32767	
R2	x0FA0	4000	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x2FFC	12284	
R7	x4E20	20000	
PSR	x0002	2	CC: Z
PC	x036C	876	
MCR	x0000	0	

- **计算 4000*5 (按运行前后状态排序)**

Registers			
R0	x0FA0	4000	
R1	x0005	5	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x0000	0	
PSR	x8002	-32766	CC: Z
PC	x3000	12288	
MCR	x0000	0	

Registers

R0	x0000	0	
R1	x7FFF	32767	
R2	x0005	5	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x2FFC	12284	
R7	x4E20	20000	
PSR	x0002	2	CC: Z
PC	x036C	876	
MCR	x0000	0	

- **计算 -500 * 433(按运行前后状态排序)**

Registers

R0	xFE0C	-500	
R1	x01B1	433	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x0000	0	
PSR	x8002	-32766	CC: Z
PC	x3000	12288	
MCR	x0000	0	

Registers

R0	x0000	0	
R1	x7FFF	32767	
R2	x01B1	433	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x2FFC	12284	
R7	xB24C	-19892	
PSR	x0002	2	CC: Z
PC	x036C	876	
MCR	x0000	0	

- **计算 -114*233(按运行前后状态排序)**

Registers

R0	xFF8E	-114	
R1	xFF17	-233	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x0000	0	
PSR	x8002	-32766	CC: Z
PC	x3000	12288	
MCR	x0000	0	

Registers

R0	x0000	0	
R1	x7FFF	32767	
R2	xFF17	-233	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x2FFC	12284	
R7	x67C2	26562	
PSR	x0002	2	CC: Z
PC	x036C	876	
MCR	x0000	0	

P版本

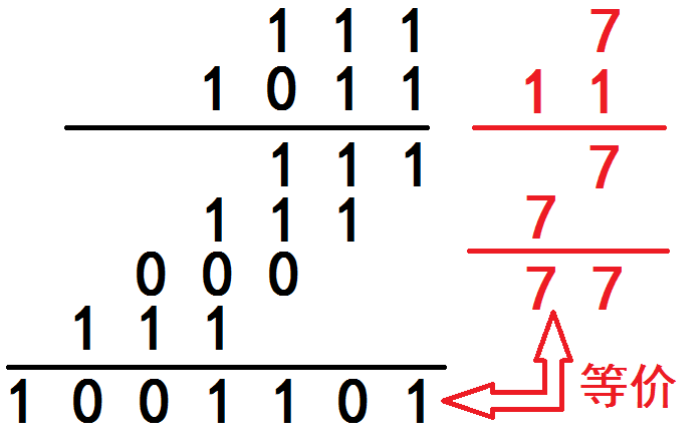
代码实现（机器码书写，通过汇编语言加C风格注释辅助说明）

```
0001010010100001;
0001011000100000;
1001100000111111;
0000100000111111;
1001101001111111;
0000100000000010;
0001000100100001;
0001001101100001;
0101100001000010;
0000010000000001;
0001111011001111;
0001011011000011;
0001010010000010;
1001101010111111;
0000100111111001;
1111000000100101;

.orig x3000 //设置初始地址
ADD R2,R2,#1; //R2自加1
ADD R3,R0,#0; //R3初始化为R0（被乘数）
NOT R4,R0; //R4标记为RO取反（判断被乘数符号）
BRn LOOP; //R0为非负数则不需要继续判断
NOT R5,R1; //R5标记为R1取反（判断乘数符号）
BRn LOOP; //R1为非负数则不需要继续判断
ADD R0,R4,x1; //求负数R0的绝对值
ADD R1,R5,x1; //求负数R1的绝对值
LOOP AND R4,R1,R2; //R1,R2求与，判断乘数当前位是否为0
BRz NEXT; //为0，不需要进行计算
ADD R7,R3,R7; //不为0，R7=R7+R3，R3是R0（被乘数）的某进位值
NEXT ADD R3,R3,R3; //R3=R3*2,进位操作
ADD R2,R2,R2; //R2进位
NOT R5,R2; //R5=R2取反，判断程序是否结束
BRn LOOP; //符号位未出现进位，程序继续执行
.end //结束
```

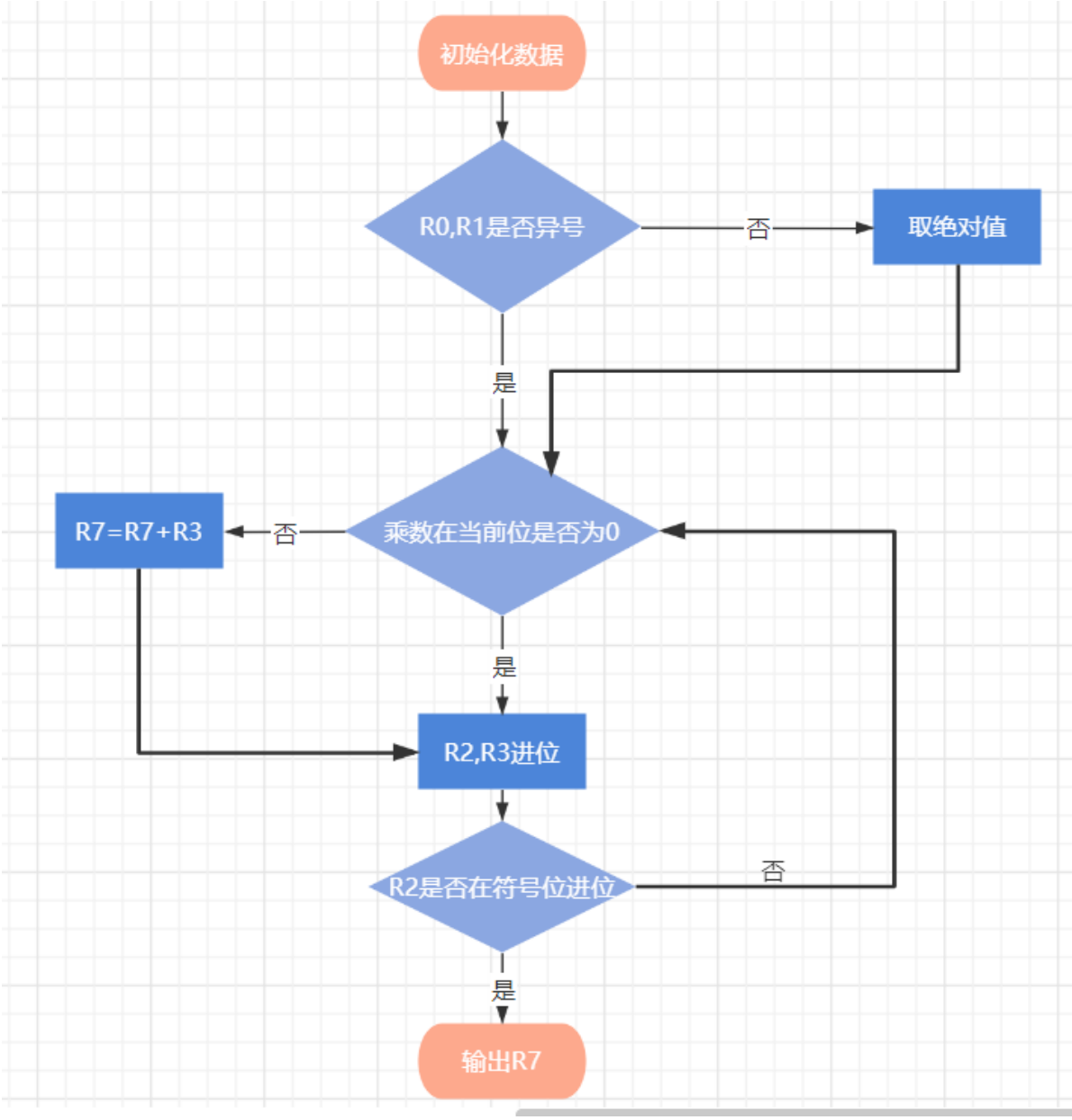
设计思路

我们笔算多位乘法时，往往习惯于进行列竖式运算。由于补码运算对列竖式计算也是适用的，我们可以对竖式运算进行模拟，当乘数某一位值不为0，则进相应位数进行竖式加法。（例如下图）



同时，考虑到乘数与被乘数符号一致时等于绝对值运算，为避免造成指令过分浪费，这里对均为负数的情况进行求绝对值操作。

因此，我们可以画出以下流程图。



样例检测与正确性检验

通过step调试确定指令数

- **case1 1*1**

Registers			
R0	x0000	0	
R1	x7FFF	32767	
R2	x8000	-32768	
R3	x8000	-32768	
R4	x0000	0	
R5	x7FFF	32767	
R6	x0000	0	
R7	x0001	1	
PSR	x8002	-32766	CC: Z
PC	x3000	12288	
MCR	x0000	0	

运行结果如上，至HALT执行了**92**条指令。

- **case2 5 * 4000**

Registers			
R0	x0000	0	
R1	x7FFF	32767	
R2	x8000	-32768	
R3	x8000	-32768	
R4	x0000	0	
R5	x7FFF	32767	
R6	x2FFE	12286	
R7	x4E20	20000	
PSR	x0002	2	CC: Z
PC	x036C	876	
MCR	x0000	0	

运行结果如上，至HALT指令执行了**101**条指令。

- **case3 4000*5**

Registers			
R0	x0000	0	
R1	x7FFF	32767	
R2	x8000	-32768	
R3	x0000	0	
R4	x0000	0	
R5	x7FFF	32767	
R6	x2FFE	12286	
R7	x4E20	20000	
PSR	x0002	2	CC: Z
PC	x036C	876	
MCR	x0000	0	

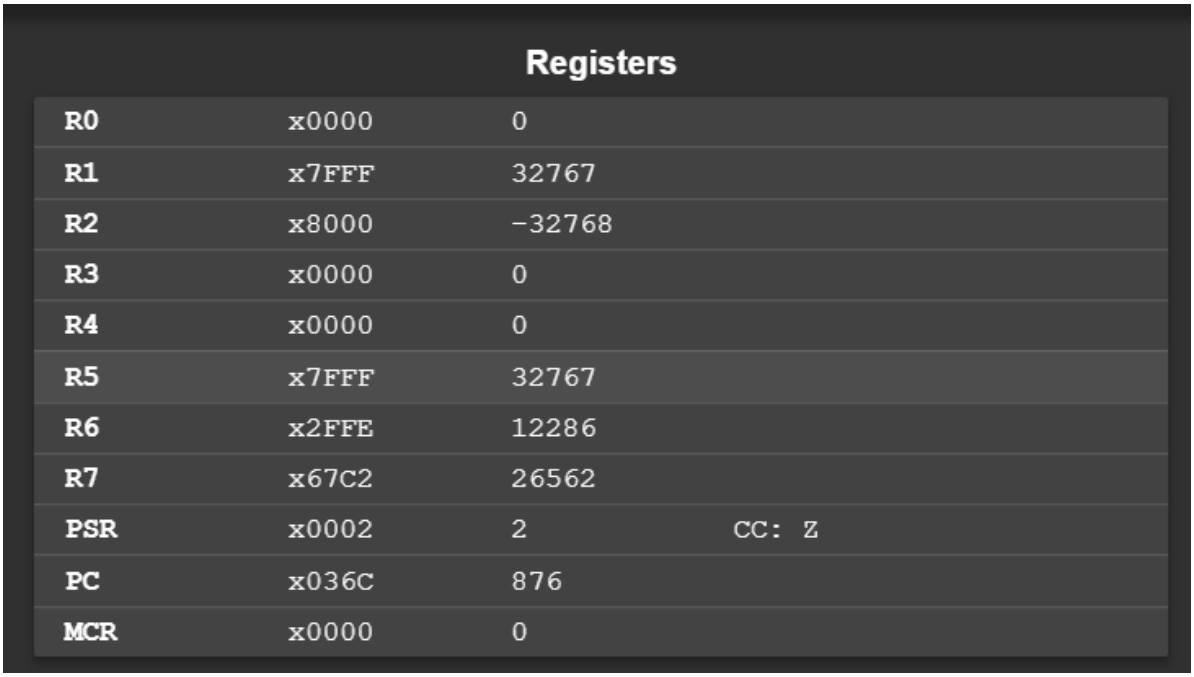
运行结果如上，至HALT指令执行了**96**条指令。

- **case4 -500*433**

Registers			
R0	x0000	0	
R1	x7FFF	32767	
R2	x8000	-32768	
R3	x0000	0	
R4	x0000	0	
R5	x7FFF	32767	
R6	x2FFE	12286	
R7	xB24C	-19892	
PSR	x0002	2	CC: Z
PC	x036C	876	
MCR	x0000	0	

运行结果如上，至HALT指令执行了**101**条指令。

- **case5 -114*-233**



运行结果如上，至HALT指令执行了**103**条指令。

综上所述，5条样例平均指令数为98.6条

代码优化

初始版本不存在halt指令和绝对值判断，在被乘数与乘数均不为0时，运行指令数要**比现在版本少3~5条。**但是，当出现双方均为负数时，代码指令数将多15次。