# ICS 2021 LAB01

# 实验报告

PB20111699 吴骏东 2021.11.28

#### 一. 实验内容

众所周知,LC3 机器是没有可以直接使用的乘法指令的。本次实验任务是实现乘法,写出对应程序机器码。两个运算数分别放置于 RO 和 R1,结果需要存储到 R7,其他寄存器状态我们不做限制(即不限结束状态)。

初始状态: RO 和 R1 存放待计算数,其余寄存器全部为 0。

请评估自己程序的代码行数、完成实验功能所需要执行的指令数,并将统计方法写在报告中。

要求提交两个版本的代码。L版本尽量编写更少的代码行数,P版本尽量让程序执行更少的指令。

### 二. L 版本代码分析

L 版本要求用更短的代码行数实现乘法。我们可以以最简单粗暴的方式实现: 循环加法模拟乘法。基本思路是,结果不断累加一个加数,并对另一个加数进行递增递减操作。该思路的流程图如图 2.1.1 所示.

随后,我们发现该过程中关于 R1 的正负讨论其实可以合并成一类。这是二进制补码运算的一个特征。这样一来代码的行数就可以进一步减少。改良后的设计思路如下图所示。

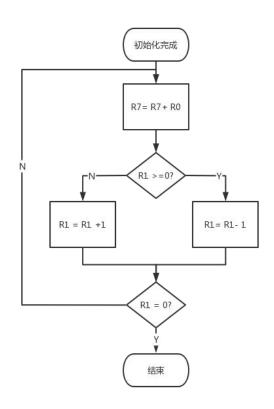
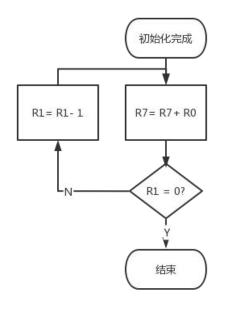
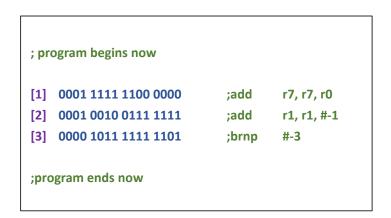


图 2.1.1





L 版本最终代码

图 2.1.2

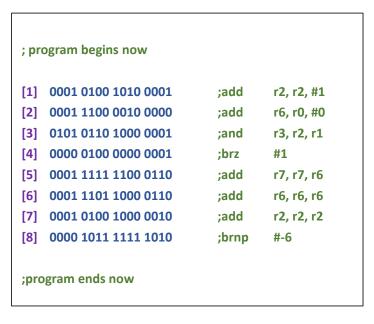
这样一来,无论 R1 大小如何, 在执行至多 65536 次循环后程序一定会终止。可验证上面的程序可以正确模拟短整型整数乘法的结果。最终版本的程序共用了 3 行。

### 三. P 版本代码分析

P 版本要求总代码执行条数最少。这样一来我们就必须改进乘法的算法。事实上乘法算法的理论最优情况就是查表法,但碍于范围过大在这里不再适用。为此我参考了《模拟与数字电路》课程中的二进制乘法器介绍,决定采用**竖式乘法**的方式进行计算。下面是一个简单的例子:

				1	0	1	0	
_		×		0	1	1	0	
				0	0	0	0	
			1	0	1	0		
		1	0	1	0			
_	0	0	0	0				_
	0	1	1	1	1	0	0	

不难发现,若被乘数的某一位是 0,则结果不会发生改变;若该位是 1,则结果加上乘数左移对应位的值。为此我们可以先判断被乘数特定位是否为 0,再让结果加上移位后的乘数即可。该算法对应的流程图如图 3.1 所示。其中 R2 的作用是判断左移的次数是否达到 16 次,R6 的作用是存储每次左移后乘数的值。对于每一次的乘法运算,该代码会固定执行 2+6×16=98 条指令,符合最终的要求。





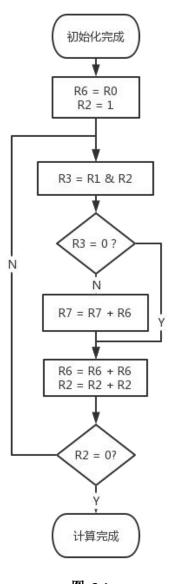


图 3.1

### 四. 小结

完成实验后,我们发现 P 版本的代码行数也没有超过 14 行,所以本实验中是可以做到一份代码同时满足两项要求的。Do not set limit for yourself. 看来还是学无止境,天外有天呀。