第一次实验实验报告

L 程序

1. 实现

计算两数相乘 a x b，即将 a 累加 b 次。当 b 为负数时，即将 a 累加 x10000-$|b|$ 次，因为溢出特性，最终结果仍为 a x b

代码思路为：

> R7=R7+R0
>
> R1=R1-1
>
> 如果 R1 不为 0，重复执行上述步骤；若 R1 为0，则程序终止

核心代码（不含 start, halt 以及寄存器初始化）为：

```
0001 111 111 0 00 000
0001 001 001 1 11111
0000 101 111111101
```

2. 代码分析

- 核心代码共 3 行
- 因为核心代码就是在循环执行上述三条指令，所以除去 start 和 halt 指令，指令执行条数为：
  - 若 b 非负，则执行 3b 条
  - 若 b 为负数，则执行 3 (x10000-$|b|$) 条
- 计算的正确性判断：利用如下 C 语言程序：（P 版本也是这种方法）

```
#include <iostream>
using namespace std;
int main()
{
    while (1)
    {
        short a, b;
        cin >> a >> b;
        cout <<short( a * b) << endl;
    }
    return 0;
}
```

- 若为无符号数，那么上述方法在乘数为负数时，会出现错误，所以当被乘数为负数时需要先对乘数进行取反加一，然后将被乘数累加，最后将结果取反加一，实现思路为：

> R1=0，直接结束
>
> R1<0，对 R1 的值取反加一
>
> R7 等于 R0 累加 R1 次
>
> 若 R1>0，程序结束

> 若 R1<0，对 R7 取反加一

代码共12行，代码如下：

```
0001 011 001 1 00000
0000 010 000001010
0000 011 000000010
1001 001 001 111111
0001 001 001 1 00001
0001 111 000 0 00 111
0001 001 001 1 11111
0000 001 111111101
0001 011 011 1 00000
0000 001 000000010
1001 111 111 111111
0001 111 111 1 00001
```

P程序

1. 实现

思路：采用列竖式的方式计算 a x b，列竖式的第 i 项时，判断 b 的从低位起第 i 位是否为 1，若是，则将已经左移过（i-1）次的 a ,加到 R7 中。然后 a 再次左移一位

实际实现：

- 从低位开始判断 b 的每一位：寄存器R2 最初置1，每次列完竖式，都将 R2 的值左移，在下一次判断 b 的某位时，只需将 R2 & b，结果存储在 R3，若 R3 为正数，则说明 b 的当前位为 1，则执行 R7 的加操作
- 左移：自己加自己
- 程序结束：R2 最初为 1，在左移 16 次后，R2=0，此式乘运算也结束，根据 R2=0 终止程序

代码思路：

> R2=R2+1
> 循环16次：
> R3=R1&R2
> 判断 R3，如果不为 0，则 R7=R7+R0
> R0=R0+R0
> R2=R2+R2
> 判断 R2，若不为零，则继续执行循环操作(PC-5)，若为 0，则程序结束

核心代码（不含 start, halt 以及寄存器初始化）为：

```
0001 010 010 1 00001
0101 011 001 0 00 010
0000 010 000000001
0001 111 111 0 00 000
0001 000 000 0 00 000
0001 010 010 0 00 010
0000 101 111111010
```

2. 代码分析

- 核心代码共 7 行
- 指令执行条数：
  - 最初 R2 置 1，需要 1 条指令
  - 在每次循环中，若 b 的当前位为 1，则执行 6 条指令；若 b 当前位为 0，则执行 5 条指令
  - 共执行 16 次循环
  - 记 b 中 1 的个数为 n ，那么总指令执行条数 = 1+6n+5(16-n) = 81+n 条
- 优化考虑：
  - 本程序一个特点是，无论 b 什么取值，都至少会执行 81 条指令，当 b=0 或 b 为较小的正数时，本程序做了许多无用的运算。可以考虑一种优化，记为优化 Y： 根据 b 的实际情况来进行左移 a 以及求和，如 b 为110，那么 a 只需要左移 2 次即可。
  - 但考虑到，一方面，若 b 为负数，那么竖式就会包含 a 左移15位，所以优化 Y 并没有大幅度减少总指令条数。另一方面，若 b 为 16 位非负整数，那么 b 的 bit[14] 为 1 的概率接近 50%，b 的最高位高于 bit[11] 的概率超过 93% ，所以优化 Y 也只能在较少的情况下起到大幅度减少总指令条数的效果
  - 另外，本程序核心部分为5行的循环（不考虑实现循环的跳转指令），相较于我自己根据 Y 版本构想的优化程序要清晰得多，更易于阅读。所以未采用优化 Y 的思路

总结

- 通过本次实验，我练习了如何用机器码完成乘法操作，对于 LC3 的指令集有了更深入的了解与掌握，对于通过 NZP 寄存器来实现循环的思想有了进一步的认识
- 通过完成 L 程序和 P 程序，锻炼了自己在写程序时，考虑代码长度以及代码效率的习惯。写一段代码，不仅仅是完成既定功能，还要考虑代码的高效性，这种思想对以后的编程有较大帮助
- 另外也深刻感受到高级语言的“高级”，在 C 语言中一行就能完成的操作，在使用机器码时却变得很复杂，而且如果没有 LC3 指令集对照表，也很难一眼看出来这段机器码完成的是什么任务