Lab 1 实验报告

- 姓名：谭旭
- 学号：PB20030775

实验内容

使用 LC3 实现乘法，写出对应程序机器码。

要求有两个版本， L 版本要求使用最少的行数完成乘法； P 版本要求使用最少的指令条数完成乘法。

实验过程

L 版本

考虑到 L 版本要求使用最少的行数完成乘法，因此应该减少代码中复杂的逻辑判断。

考虑到乘法本质上是多个相同数的加法，而 LC3 指令集中包含了加法指令，因此可以直接使用一个计数器来记录求和的个数，用求和的结果作为乘法的结果。

实现方法如下：

将两数记为 `R0` 、 `R1` ，求和结果 `R7` ，则每次循环对 `R1` 进行自减操作，并将 `R7` 加上一个 `R0` 。

如此，得到的结果便是 b 个 a 的和。

bin 代码如下：

```
0001111111000000    ;   R7 <= R7 + R0
0001001001111111    ;   R1 <= R1 - 1
0000101111111101    ;   BR np Line1
```

以上代码在对于 `R1` 为非负数时显然是正确的，当 `R1` 为负数时，代码正确性还需要判断。

由于 LC3 中对于整数的存储使用补码，若 `R1` 为负数，则实际上对 R0 的求和次数为

$$R_1' = R_1 + 2^{16}$$

即，求和的实际结果为

$$Sum' = R_0(R_1 + 2^{16}) = R_0R_1 + 2^{16}R_0$$

使用补码进行数据存储，则实际存储的求和结果为

$$Sum = Sum' \mod 2^{16} = R_0R_1$$

因此上述代码对 `R1` 为负数同样成立。

该程序最初使用了 3 行，最终版本使用了 3 行。

P 版本

P 版本要求使用尽量少的指令数来完成乘法，这样一来上述 L 般的程序所需要的命令数显然过多，在乘数 `R1` 的补码足够大的时候（如 -1 ），程序需要执行的循环次数已经高达 $2^{16}-1$ 次，因此继续沿用上述作法是不可行的。

而要求使用尽量少的指令数，实际上就是在要求程序的时间复杂度尽量低，朴素乘法的时间复杂度为 $O(n^2)$ ，其实现原理即为高精度乘法。

在十进制高精度乘法中，对于两数中的每一位，都要进行一次乘法操作，而在二进制乘法中，由于只有两个数字 0 和 1 ，对于乘数 0 ，其与被乘数的乘积一定为 0 ；对于乘数 1 ，其与被乘数的乘积一定为被乘数本身。因此，使用高精度乘法的原理，计算过程中只需要使用加法即可完成。

依照上述原理，使用一个变量 `R2` ，使其始终等于 2^i ，并使用 `R2` 判断 `R1` 的每一位是否为 1 ，如果为 1 ，那么就将最终结果加上左移 i 位后的 `R0` 。

bin 代码如下：

```
0001010010100001    ;   R2 <= R2 + 1
0101011001000010    ;   R3 <= R1 & R2
0000010000000001    ;   BR z Line5
0001111111000000    ;   R7 <= R7 + R0
0001000000000000    ;   R0 <= R0 + R0
0001010010000010    ;   R2 <= R2 + R2
0000101111111010    ;   BR np Line2
```

上述作法对于 `R1` 为非负数是显然成立的，对于 `R1` 为负数时正确性的证明和 L 版相同。

由代码可知，假设第 3 行的 BR 指令从未跳转过，那么该程序的实际指令执行条数为 $1+6\times 16=97$ 条指令；

假设第 3 行的 BR 指令每次都跳转，那么该程序的实际指令执行条数为 $1+5\times 16=81$ 条指令。

即，上述代码实际执行的指令条数一定处于 $[81,97]$ 的范围内。

该程序仅有上述一个版本，未在此基础上做出改进。

代码正确性

上文已给出对于 R1 为负数时，程序正确性的证明。

在实际实验过程中，还需要对各种测试样例进行测试分析。

由于 LC3 Tools 中寄存器不显示负数，本次实验中使用 `c++` 中的 `unsigned short` 来进行对拍。

将负数输入对拍程序和 LC3 Tools 并计算后，均会输出一个非负整数，即实际答案在模 2^{16} 下的结果，通过对比两数据，可以确定代码的正确性。

对拍代码如下：

程序 1

```cpp
#include <iostream>
using namespace std;
int main()
{
    unsigned short a, b, ans;
    cin >> a >> b;
    ans = a * b;
    cout << ans;
    return 0;
}
```

程序 2

```cpp
#include <iostream>
using namespace std;
int main()
{
    unsigned short a, b, ans = 0, temp = 1;
    cin >> a >> b;
    while (temp != 0)
    {
        if (b & temp)
        {
            ans += a;
        }
        temp <<= 1;
        a <<= 1;
    }
    cout << ans;
    return 0;
}
```

程序 1 给出了两数相乘的结果；程序 2 模拟了 P 版程序，给出了 P 版程序的运算结果。

编译指令：`g++ -std=c++17 source.cpp`

使用方法：输入两个使用空格分隔的整数，程序将输出它们在 `unsigned short` 下的乘积。

代码指令数

上文已分析出 P 版代码实际执行指令数一定在 $[81, 97]$ 的范围内。

在数据充分随机的测试样例下，该程序的平均指令数应为 89 条。

在实际测试过程中，通过 LC3 Tools 中的 Step In 功能，统计程序共需要步进的次数，即为程序实际执行指令数。

对于文档中提供的测试样例，测试结果如下表格

R0	R1	R7	指令数
1	1	1	82
5	4000	20000	87
4000	5	20000	83
-500	433	45644	86
-114	-233	26562	93