原码的乘除法运算

一.总体思路

1. 将原码限定为 8bits
2. 从标准输入读入两个十进制数保存在两个 int 当中
3. 通过 itot 函数将两数转变成字符串形式的原码并输出
4. 利用已经实现的加减法运算来模拟原码的乘除发法运算，并输出结果（乘法得到 16bits 的得数）

二.算法证明

1. 乘法

   通过多次的加法来实现。

   以 00101011 * 00000101 为例：

   首先忽略符号位，

   ```
          0101011
   *      0000101
   ------------------------
          0101011
         0000000
        0101011
   ------------------------
       110010111
   ```

   通过类似与补码乘法运算的硬件实现，先将乘数放在积的末尾，当积的末位为 1，则加上被乘数后右移 1 位，若为 0，则直接右移 1 位。

   以上运算结束后，判断相乘两数的符号位，对得数的符号位进行赋值。

2. 除法

   通过多次的减法来实现

   首先忽略符号位，将除数移到高位，然后开始判断，若被除数够一次减法，则得数上该位为 1，并做一次减法，若不够则为 0，然后将除数右移，继续进行判断。

   最后判断符号位，若同号则为 0，异号为 1。

   同样也可以用类似补码的交叉加减法的运算，运算效率应当会有所提高。

三.使用方法

源文件 true_form.c 在文件夹 src 当中，可直接编译运行。

输入两个在-127~127 之间的运算数，能得到 6 行结果，分别是运算数的原码，和与差的原码以及和与差的十进制数。

若在 linux 系统中，可直接使用 make 命令编译得到可执行程序，再通过命令 cat test.txt | ./true_form.out 来得到实例分析中的测试结果。

四.特殊处理

整数除法的余数也给予了输出， 在 remain 当中

五.实例分析

15 20
num1 = 00001111
num2 = 00010100
sum = 00100011 overflow = 0
difference = 10000101 overflow = 0
product = 0000000100101100
quotient = 00000000 remain = 00001111
sum = 35
difference = -5
product = 300
quotient = 0 remain = 15

127 1
num1 = 01111111
num2 = 00000001
sum = 00000000 overflow = 1
difference = 01111110 overflow = 0
product = 0000000001111111
quotient = 01111111 remain = 00000000
sum = 0
difference = 126
product = 127
quotient = 127 remain = 0

127 -127
num1 = 01111111
num2 = 11111111
sum = 10000000 overflow = 0
difference = 01111110 overflow = 1
product = 1011111100000001
quotient = 10000001 remain = 00000000

sum = 0
difference = 126
product = -16129
quotient = -1 remain = 0

-120 -10
num1 = 11111000
num2 = 10001010
sum = 10000010 overflow = 1
difference = 11101110 overflow = 0
product = 0000010010110000
quotient = 00001100 remain = 00000000
sum = -2
difference = -110
product = 1200
quotient = 12 remain = 0