为什么 $x-y==x+(-y)_{模}$

在学习补码的时候，我们说利用补码，就可以把计算机中的减法运算转换成为加法运算，这是因为在计算机系统中 $x-y==x+(-y)_{模}$，这是为什么呢？

（不是一般性，这里我们将 x、y 视为整数）。

这要从模运算讲起：

1. 模运算和模同余的概念

（1）模运算

“模”来自于“Mod”的音译。**模运算就是取余运算**，即求两个整数相除时的余数，如 5 除 3，得商 1，**余数 2**。

（2）模同余

在一个以 M 为“模”的模运算系统中（这里 M 是整数），若二个整数 x、y 及 M 间满足下列关系：

$$x=y+k\times M，k 为整数，$$

则称 y 和 x 为**模 M 同余**，即 x 和 y 各除以 M 的余数相同。记为：

$$x \equiv y \pmod M。$$

2. 现实世界中模运算系统的一个具体例子：时钟。时钟是一种模 12 系统。

对于时钟，有下面的现象：

假定钟表时针指向 10 点，要将它拨向 6 点，则有两种拨法：

① 做减法：倒拨 4 格，即 10-4 = 6

② 做加法：顺拨 8 格，即 10+8 = 18 ≡ 6 **(mod 12)**

可见，在模 12 系统中：10 减 4 等价于 10 加 8，也记为：

$$-4 \equiv 8 \pmod{12}，即-4 与 8 模 12 同余$$

这里称 8 是-4 对模 12 的补码，即 8=12+(-4)。

所以，**(负数的）原码和补码其实就是关于一个模互补的一对数**，（可以看做

一正一副）**两者差的绝对值等于模。**

基于上面的时钟例子，可以看到：**在模运算系统中，减去一个数等于加上这个数负数的补码**：

$$10 - 4 \equiv 10 + (-4) \equiv 10 + (12-4) \equiv 10+8 \equiv 6 \pmod{12}$$

3. 模运算规则

规则 1） 一个负数的补码等于模减该负数的绝对值。

如：M=12 时，-4 的补码 = 12-4 = 8

规则 2） 对于一个确定的模，某个数 A 减去小于模的另一个数 B，可用 A 加上(-B)的补码来代替 —— 这在模同余的前提下是等价的。

如：$10-4 \equiv 10+(12-4) \equiv 10+8 \pmod{12}$

所以，**模运算系统实现了加减运算的等价转换**：用加法来实现减法运算

—— 减一个数等于加上这个(负)数的补码(模 M)。

4. 计算机是一个模运算系统

为什么说计算机是一个模运算系统呢？这是因为，**在计算机中的存储、运算和传送数据的部件都只有有限位数，简单的理解，就是计算机中的任何数据的长度都是有限的，内存中仅用有限数量的位表示一个数**，比如 int 只有 32 位，short 只有 16 位。

如果某种运算的结果超过了相应数据类型的固有长度，会怎么样呢?

比如：设 x 和 y 都是 16 位的 unsigned short int 型数据，且 x=y=65535，那么 x+y 等于多少呢？

如果不考虑溢出，算上**最前面的进位**，总的结果是：

$(11111111111111110)_2$，**17bit**。

但要记住：short/unsigned short 是 16 位的，所以上面的**结果事实上只能保留 16 位，而且是低 16 位，最高位 1 被舍去了**，**实际的结果是**

$(1111111111111110)_2$，**16bit**。

这样的一个结果相当于什么呢?

取模，即，不管在不考虑进位溢出情况时值有多大，实际得到的结果是不考虑进位溢出情况下的值**除模 M 的余数——相当于舍弃进位，只保留低 n 位**。

就如同时钟系统，过了 12 点再加 1，不考虑进位溢出（超过 12 点）是 13 点，但表盘上，实际显示的值是 1 点，因为$(12+1)_{(\mathrm{mod}\ 12)} \equiv 1$

所以，不管是时钟模的 12 也好，还是 short 型整数的模 2^{16}、或者 int 型整数的模 2^{32}，**都是具有同类性质的模运算系统**。

那么，在计算机系统中，如同时钟一样，同样具有模同余的性质，而所谓的**一个负数的补码，事实上就是这个负数关于其模的那个“互补”码**。

如：16 位 short int 数-1，

原码：$(1000000000000001)_2$

补码：$(1111111111111111)_2$

原码+补码=2^{16} **注：2^{16} 是 1 后面跟 16 个 0**

这样就有和前面时钟同样的性质，减去一个数，等于加上这个（负）数的补码:

如，$5-1 \equiv 5+(-1)_{补}$

即：$(0000000000000101)_2-(0000000000000001)_2$

$\equiv (0000000000000101)_2+(1111111111111111)_2$

$= (0000000000000100)_2$ 注：最前面有个进位的 1，但只保留低 16 位，所以这个进位事实上是被舍去了，只留下$(0000000000000100)_2$，即十进制的 4，**结果正确**。