链滴

# 原码、反码、补码

作者：JavaIsRubbish

原文链接：https://ld246.com/article/1525836744064

来源网站：链滴

学过计算机原理的人都知道原码、反码、补码，但是有多少人知道为什么会有这三种码呢，这三种码是用来干嘛的呢。

众所周知，在计算机的世界只有01，那么显然所有的数都得转成二进制，这样计算机才能够理解。何将一个十进制的数转成二进制就不说了，说下原码，正数的原码就是十进制转成二进制得到的二进值，而负数是对应的正数转成二进制得到的二进制值，然后将最高位（符号位）置为1表示这是一个数，如-10:10001010。

### 1. 原码

计算机进行算术运算时为了简单效率所以要求能够使用加法代替减法，如1-1==1+(-1)==0，那么们先看看原码能不能实现这种需求。

示例：

```
计算76-10==66
  十进制     二进制
  76        01001100
 +  -10      10001010
---------------------
  66        11010110（-86）
```

### 2. 反码

从上面算出的结果可见原码是无法完成对减法的运算需求的，那么由于1-1==1+（-1），所以人类找到了一个看似能够解决这个问题的解决方法——反码，即将负数的符号位不变其余位取反。下面我再看看反码能不能解决问题。

示例1：计算15-125

计算15-125==-110
  十进制    二进制原码    二进制反码
  15       00001111    00001111
 +  -125    11111101    10000010
--------------------------------
  -110     11101110    10010001
得到10010001(反码)==11101110(原码)==-110，正确。注意：使用反码计算得到的结果也是反码需要再次转换成原码。

示例2：计算76-10

计算76-10==66
  十进制    二进制原码    二进制反码
  76       01001100    01001100
 + -10     10001010    11110101
--------------------------------
  66       01000010    101000001==01000010
这里得到的值超过8bit，所以最高的1需要丢弃，丢弃后需要在最低位+1，得到01000010(反码)==01000010(原码)==66，正确。

示例3：计算1-1

计算1-1==0
  十进制    二进制原码    二进制反码
  1        00000001    00000001
 + -1      10000001    11111110
--------------------------------
  0        10000000    11111111
得到11111111(反码)==10000000(原码)==-0，-0？通过反码计算会出现+0和-0，一个0对应了两码，显然是不合理的。

从上面三个例子可以看出使用反码进行减法运算时存在两个问题：

1. 当计算结果溢出时需要额外进行+1操作，使得运算多了一步，效率降低

2. 0存在+0和-0两种存在方式，不方便理解

### 3. 模与互补、同余

在看补码之前，先介绍三个概念——模、补数、同余。我们从现实生活举例来看：

- 我们将一个时钟的分针往前拨20分钟，和往后拨40分钟，得到的结果是一样的。
- 把你的属年(属猴)往后退5年，和往前进7年，一样都是属兔。
- 把数字 87，减去 25，和加上 75，在不考虑百位数的条件下，得到的结果都是62。

上述几组数字，有这样的关系：

  20 + 40 = 60

  5 + 7 = 12

  25 + 75 = 100

式中的 60、12 和 100，就是“模”。

式中的 20和40、5和7，以及25和75，就是一对对“互补”的数字。

而且20，80，140在模是60的情况下就是互为“同余”的数字。

通俗解释下模、补数、同余的概念：

- **模**：就是一个轮回，比如分针转一圈，十二生肖一轮等等。
- **互补**： 一个数值针对某个模的互补值就是这个数值加上或者减去多少能够等于模，或者等于模的同值。
- **同余**： 一个数值加上或者减去模的整数倍得到的所有数值即为该数值的同余值**(也就是除上模，数是一样，所以叫同余)**，0是模的同余，-模也是模的同余。

理解了什么是模，什么是互补、什么是同余，那么如果给一个模，以及一个值a，如果计算a的补数(a互补的值)呢，其实很简单，只需要拿模-a即可，计算同余值可以直接加上或者减去模的整数倍即可。

### 4. 那么互补的值有什么用呢?

如果我们在进行减法运算时，用与减数互补的值代替减数与被减数进行加法运算会发生什么呢？废不多说，看示例。

示例1：在分钟刻度下，计算55分钟往后拨动34分钟，转化成数学计算就是：55-34

```
被减数     55
减数       34
减数补数   60-34==26
最终结果    55+26==81
--------------------
```

用减数补数代替减数得到结果为81,81在分钟刻度盘上正好是21，也就是81是21的同余值，和55-34一样的。注意：这里涉及到类似上面的87+75的情况，即忽略了进位。

示例2：在十二生肖中，计算猴年往后退11年，转化成数学计算就是：9-11

```
被减数     9
减数       11
减数补数   12-11==1
最终结果    9+1==10
------------------
```

用减数互补值代替减数得到结果为10,10对应到十二生肖正好是鸡，和猴年往后退11年是一样的，所得到的也是一个同余值。

从上面的示例可以看出，使用互补值计算出的结果与实际值其实是**同余**的关系。

### 5. 二进制的模

上面看了分钟刻度盘的模，十二生肖的模，以及两位整数的模，那么对于一个8bit的字节的模是多呢?

分钟刻度盘的模为什么是60？是因为他的值是从1-59，总共60个值，十二生肖以及两位整数也是一的，所以我们只需要看看一个8bit的字节的所有取值一共是多少个就是他的模，显示8个bit可表示的小值是00000000==0，最大值是11111111==255，那么从0到255一共是256个值，所以一个8bit字节的模就是256了。但是其实在计算机中为了能够表示负数，所以讲8bit的字节的最高位设为符号，0表示整数，1表示负数，所以能够表示数值的也就只有7bit，如果我们忽视符号位，那么剩下7bit

模就是128，而不是256了。**下面在计算时我们会直接使用128而非256！**

### 6. 使用互补值进行二进制的减法计算

下面我们就来看看如果使用互补值来进行二进制的减法计算，我们先来看一个公式：假设模式M，们计算X-Y，然后我们使用减数的补数来计算，看看下面的换算：

X-Y == X+(M-Y) == X+((M-1)-Y+1)

下面我们来看示例，这个公式在下面会用到的。

示例1：计算15-125

| | 十进制 | 二进制 |
|---|---|---|
| 被减数 | 15 | 0001111 |
| 减数 | 125 | 1111101 |
| 减数补数 | 128-125==3 | 0000011 |
| 最终结果 | 18 | 0010010 |

---

得到0010010==18，在模式128的情况下，18正好是-110的同余值，跟上面现实的例子是一样的！

示例2：计算76-10

| | 十进制 | 二进制 |
|---|---|---|
| 被减数 | 76 | 01001100 |
| 减数 | 10 | 00001010 |
| 减数补数 | 256-10==246 | 11110110 |
| 最终结果 | 322 | 101000010 |

---

得到101000010==322，在模式256的情况下，322正好是66的同余值，结果还是一样！

从上面两个例子我们应该可以看出，如果我们使用减数的补数进行加法运算，那么得到的结果就是个与正确结果同余的值。在现实生活中，我们可以直接把两个同余的值看做是相同的，例如分钟20和钟80完全就是一样的，那么在计算机里我们可以这么假设吗？答案是可以的，看下面。

试想当计算机使用一个7bit的空间保存一个数值时是如何保存的，比如18，我们可以这么推算，首分配一个7bit的空间，每个bit上的值都是0，那么如何表示18呢？我们可以这样理解：往这个7bit的间内进行18次加1操作，满2就进1，最终就会得到0010010。那么如何表示-110，我们可以理解为往个7bit的空间内进行110次减1操作，一开始全是0，那么如何减1呢？很简单直接减成1111111即可可以这样理解，分钟在0刻度，你往后拨一下就会指向59，这里也是这个道理，所以连续减110次，会得到0010010，根18是一样的，所以在计算机看来18和-110是一样的。

也就是说**15-125 == 15+(128-125) == 15+(127-125+1)** (上面的公式)，也就是说-125被127-12+1代替了，那么**127-125+1（M-）**又是什么？

### 7. 补码

上面一路走来终于证明了使用补数可以代替减法，下面我们要解决的问题是M-1-Y+1是啥。

我们直接看二进制如何计算M-1-Y+1。

示例：计算M=128，Y=110

十进制　二进制
M-1　127　1111111
-Y　110　1101110
　　0010001
---
M-1换算成二进制就是N位1，那么N位1减去任何一个N位的二进制是啥呢？其实就是按位取反！因遇到0,1-0==1，取反，遇到1,1-1==0，取反，所以整体就是按位取反，也就是反码。

+1　1　0000001
　　0010010
---
所以总体就是在110的二进制基础上按位取反然后加1,也就是110的反码加1。

看了上面的示例，应该知道M=128，Y=110，M-1-Y+1就是Y的反码加1，也就是说，如果我们需计算X-Y，只需要计算X+(Y的反码+1)，由于我们得出这个结论是使用**补数替代减法**得到的，所以*Y的反码+1**就被叫做Y的**补码**。

到这里我们知道了110的补码，上面我们介绍了计算机使用1字节的最高位表示符号位，1表示负数所以-110的最高位是1，由于在使用补码进行减法运算过程中最高位并不参与运算，所以这个最高位该是固定不动的，所以负数的反码补码最高位始终都是1。**也就得到了-110的补码是：10010010**对于正数，符号位是0，那么反码补码最高位就始终是0，而且对于正数在计算时也无需使用其补码进操作，但是为了统一都是用补码，所以定正数的反码补码都等于原码。

根据补码的计算过程有些文章会说一个负数X的补码对应的值==2^n-|X|，理解了上面的过程这个式就自然懂了，不过这个公式没啥用，也没必要记。

到这里终于把**补码**的来历说清楚了，至少我自己是明白了，但愿读者也可以明白吧！

**一些补码的其他知识**

上面我们看了7bit的模式128，也就说是能表示0-127共128个数值，加上最高位的符号位就成了-12-127共计255个数值，因为没有-0这个数字。但是实际对于计算机来说8bit的空间是可以表示256个字的，那么还有一个数字是啥呢？正是：**10000000**(注意：这是补码，因为计算机都存的补码)。我可以试着计算下10000000的原码，可以得到10000000的原码就是10000000，也就是-0，但是如果在+0和-0两个计算机码对应一个值(+0和-0都是0)，那么显然是没必要的，而且还会造成混乱，所以为的规定**10000000表示-128**。所以一个8bit的空间可以表示的数字就是从-128到127了，而不是-12-127！

https://blog.csdn.net/vickyway/article/details/48788769