转---原码，反码，补码的深入理解与原理。

这里涉及到有关原码，反码，补码的基础知识，在网上查阅资料，发现一篇讲的很不错的文章。

【https://www.imooc.com/article/16813?block_id=tuijian_wz】

以下是文章内容：

本文从原码讲起。通过简述原码，反码和补码存在的作用，加深对补码的认识。力争让你对补码的概念不再局限于：负数的补码等于反码加一。

接触过计算机或电子信息相关课程的同学，应该都或多或少看过补码这哥仨。每次都是在课本的最前几页，来上这么一段：什么反码是原码除符号位，按位取反。补码等于反码加一。然后给整得莫名其妙，稀里糊涂地，接着就是翻页，反正后面的内容也跟三码没多大关系。

我原来也是看了好几遍都没看懂。古人云：事不过三。学 C 语言的时候，看过一次。不懂？看《计算机基本组成原理》的时候看过，还是不懂！到了大三，上《单片微机原理与接口技术》的时候仍旧是不懂。到了期末，复习的时候，和宿舍的人瞎聊。说讲讲这些码呀，我说我也不是很清楚呀。然后就一边说怎么求码，一边算。玩着玩着，突然就明白了。我说好，打住。不说了，放假我在好好整理下思路，于是就有了这篇额。。算讨论帖吧。

好了，废话不多说。开始我们的原码，反码，补码之旅。

（一）预备知识

认识二进制，十六进制。会二进制与十进制的相互转化运算

由计算机的硬件决定，任何存储于计算机中的数据，其本质都是以二进制码存储。

根据冯~诺依曼提出的经典计算机体系结构框架。一台计算机由运算器，控制器，存储器，输入和输出设备组成。其中运算器，只有加法运算器，没有减法运算器（据说一开始是有的，后来由于减法器硬件开销太大，被废了 ）

所以，计算机中的没法直接做减法的，它的减法是通过加法来实现的。你也许会说，现实世界中所有的减法也可以当成加法的，减去一个数，可以看作加上这个数的相反数。当然没错，但是前提是要先有负数的概念。这就为什么不得不引入一个该死的符号位。

1. 而且从硬件的角度上看，只有正数加负数才算减法。
2. 正数与正数相加，负数与负数相加，其实都可以通过加法器直接相加。

原码，反码，补码的产生过程，就是为了解决，计算机做减法和引入符号位（正号和负号）的问题。

本文可能比较长，没必要一下子读完。原码，反码，补码，按章读。

重点在于讲补码，到了补码可能有些绕，建议带着笔，写出二进制数一起算。

表达可能不够清楚严谨，望见谅。

（二）原码

原码：是最简单的机器数表示法。用最高位表示符号位，‘1’表示负号，‘0’表示正号。其他位存放该数的二进制的绝对值。

若以带符号位的四位二进值数为例

1. 1010 ： 最高位为‘1’,表示这是一个负数，其他三位为‘010’，
2. 即（0*2^2）+（1*2^1）+（0*2^0）=2（‘^’表示幂运算符）
3. 所以 1010 表示十进制数（-2）。

下图给出部份正负数数的二进制原码表示法

	正数
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

	负数
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

OK，原码表示法很简单有没有，虽然出现了+0 和-0，但是直观易懂。

于是，我们高兴的开始运算。

1. 0001+0010=0011 （1+2=3）OK
2. 0000+1000=1000 （+0+（-0）=-0） 额，问题不大
3. 0001+1001=1010 （1+（-1）=-2）

噢，1+（-1）=-2，这仿佛是在逗我呢。

于是我们可以看到其实正数之间的加法通常是不会出错的，因为它就是一个很简单的二进制加法。

而正数与负数相加，或负数与负数相加，就要引起莫名其妙的结果，这都是该死的符号位引起的。0 分为+0 和-0 也是因他而起。

所以原码，虽然直观易懂，易于正值转换。但用来实现加减法的话，运算规则总归是太复杂。于是反码来了。

（三）反码

我们知道，原码最大的问题就在于一个数加上他的相反数不等于零。

例如：0001+1001=1010 (1+(-1)=-2) 0010+1010=1100 (2+(-2)=-4)

于是反码的设计思想就是冲着解决这一点，既然一个负数是一个正数的相反数，那我们干脆用一个正数按位取反来表示负数试试。

> **反码**：正数的反码还是等于原码
>
> 负数的反码就是他的原码除符号位外，按位取反。

若以带符号位的四位二进制数为例：

1. 3 是正数，反码与原码相同，则可以表示为 0011
2. -3 的原码是 1011，符号位保持不变，低三位（011）按位取反得（100）
3. 所以-3 的反码为 1100

下图给出部分正负数的二进制数反码表示法

反码 **原码**

	正数
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

	负数
-0	1111
-1	1110
-2	1101
-3	1100
-4	1011
-5	1010
-6	1001
-7	1000

	负数
-0	1000
-1	1001
-2	1010
-3	1011
-4	1100
-5	1101
-6	1110
-7	1111

对着上图，我们再试着用反码的方式解决一下原码的问题

0001+1110=1111 （1+（-1）=-0）

互为相反数相加等于 0，解决。虽然是得到的结果是 1111 也就是-0

好，我们再试着做一下两个负数相加

1110（-1）+1101（-2）=1011（-4）

噢，好像又出现了新问题

（-1）+（-2）=（-4）?

不过好像问题不大，因为 1011（是-4 的反码，但是从原码来看，他其实是-3。巧合吗？）

我们再看个例子吧

1110（-1）+1100（-3）=1010（-5）

确实是巧合，看来相反数问题是解决了，但是却让两个负数相加的出错了。

但是实际上，两个负数相加出错其实问题不大。我们回头想想我们的目的是什么？是解决做减法的问题，把减法当成加法来算。

两个正数相加和两个负数相加，其实都是一个加法问题，只是有无符号位罢了。而正数+负数才是真正的减法问题。

也就是说只要正数+负数不会出错，那么就没问题了。负数加负数出错没关系的，负数的本质就是正数加上一个符号位而已。

在原码表示法中两个负数相加，其实在不溢出的情况下结果就只有符号位出错而已（1001+1010=0011）

反码的负数相加出错，其实问题不大。我们只需要加实现两个负数加法时，将两个负数反码包括符号位全部按位取反相加，然后再给他的**符号位强行置‘1’**就可以了。

所以反码表示法其实已经解决了减法的问题，他不仅不会像原码那样出现两个相反数相加不为零的情况，而且对于任意的一个正数加负数，如：

0001（1）+1101（-2）=1110（-1） 计算结果是正确的。所以反码与原码比较，最大的优点，就在于解决了减法的问题。

但是我们还是不满足为什么 0001+1110=1111 （1+（-1）=-0） 为什么是-0 呢

而且虽然说两个负数相加问题不大，但是问题不大，也是问题呀。好吧，处女座。接下来就介绍我们的大 boss 补码。

（四）补码

补码：正数的补码等于他的原码

负数的补码等于反码+1。

（这只是一种算补码的方式，多数书对于补码就是这句话）

在《计算机组成原理中》，补码的另外一种算法 是

负数的补码等于他的原码自低位向高位，尾数的第一个‘1’及其右边的‘0’保持不变，左边的各位按位取反，符号位不变。

OK，补码就讲完了。再见！！

还是莫名其妙有没有，**为什么补码等于反码加 1，为什么自低位向高位取反**.....?

其实上面那两段话，都只是补码的求法，而不是补码的定义。很多人以为求补码就要先求反码，其实并不是。

那些鸡贼的计算机学家，并不会心血来潮的把反码+1 就定义为补码。只不过是补码正好就等于反码加 1 罢了。

所以，忘记那些书上那句负数的补码等于它的反码+1。就这句话把我们带入了理解的误区。

这就是后来我明白为什么我看的那本《计算机组成原理》，要特意先讲补码，再讲反码。

然后说负数的补码等于他的原码自低位向高位，尾数的第一个‘1’及其右边的‘0’保持不变，左边的各位按位取反，符号位不变。

但是上面这句话，同样不是补码的定义，它只是补码的另外一种求法。它的存在，告诉我们忘记那句该死的‘反码+1’它并不是必须的。

如果你有兴趣了解，补码的严格说法，我建议你可以看一下《计算机组成原理》。它会用‘模’和‘同余’的概念，严谨地解释补码。

接下来我只想聊聊补码的思想。

(五)补码的思想

补码的思想，第一次见可能会觉得很绕，但是如果你肯停下来仔细想想，绝对会觉得非常美妙。

补码的思想其实就来自于生活，只是我们没注意到而已。时钟，经纬度，《易经》里的八卦。

补码的思想其实就类似于生活中的时钟

好吧，我其实不想用类似，好像这种词，因为类比的，终究不是事物本身。而且不严谨会让我怀疑我不是工科僧，说得好像我严谨过似的，哈哈

如果说现在时针现在停在 10 点钟，那么什么时候时针会停在八点钟呢？

简单，过去隔两个小时的时候，是八点钟。未来过十个小时的时候也是八点钟

也就是说时间正拨 10 小时，或是倒拨 2 小时都是八点钟。

也就是 10-2=8，而且 10+10=8（10+10=10+2+8=12+8=8）

这个时候满 12 说明时针在走第二圈了，又走了 8 小时，所以时针正好又停在八点钟。

所以 12 在时钟运算中，称之为模，超过了 12 就会重新从 1 开始算了。

也就是说， 10-2 和 10+10 从另一个角度来看是等效的，它都使时针指向了八点钟。

既然是等效的，那在时钟运算中，减去一个数，其实就相当于加上另外一个数（这个数与减数相加正好等于 12，也称为同余数）

这就是补码所谓模运算思想的生活例子

在这里，我们再次强调原码，反码，补码的引入是为了解决做减法的问题。在原码，反码表示法中，我们把减法化为加法的思维是减去一个数，等于加上一

个数的相反数，结果发现引入了符号位，却因为符号位造成了各种意向不到的问题。

但是从上面的例子中，我们可以看到其实减去一个数，对于数值有限制，有溢出的运算（模运算）来说，其实也相当于加上这个数的同余数。

也就是说，我们不引入负数的概念，就可以把减法当成加法来算。所以接下来我们聊 4 位二进制数的运算，也不必急于引入符号位。因为补码的思想，把减法当成加法时并不是必须要引入符号位的。

而且我们可以通过下面的例子，也许能回答另一个问题，为什么负数的符号位是‘1’，而不是正数的符号位是‘1’。

(六)补码实例

好吧，接下来我们就做一做四位二进制数的减法吧（先不引入符号位）

0110（6）-0010（2）【6-2=4，但是由于计算机中没有减法器，我们没法算】

这个时候，我们想想时钟运算中，减去一个数，是可以等同于加上另外一个正数（同余数）

那么这个数是什么呢？从时钟运算中我们可以看出这个数与减数相加正好等于模。

那么四位二进制数的模是多少呢？也就是说四位二进制数最大容量是多少？其实就是 2^4=16=10000B

那么 2 的同余数，就等于 10000-0010=1110（14）

既然如此

0110（6）-0010（2）=0110（6）+1110（14)=10100（20=16+4）

OK，我们看到按照这种算法得出的结果是 10100，但是对于四位二进制数，最大只能存放 4 位（硬件决定了），如果我们低四位，正好是 0100（4），正好是我们想要的结果，至于最高位的‘1’，计算机会把他放入 psw 寄存器进位位中。8 位机则会放在 cy 中，x86 会放在 cf 中（这个我们不作讨论）

这个时候，我们再想想在四位二进制数中，减去 2，就相当于加上它的同余数 14（至于它们为什么同余，还是建议看《计算机组成原理》）

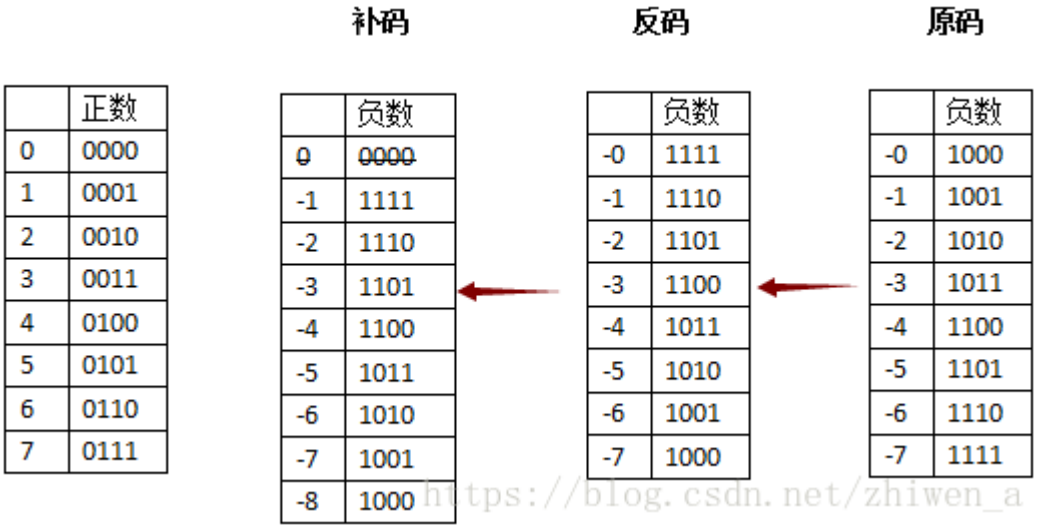
但是减去 2，从另外一个角度来说，也是加上（-2）。即加上（-2）和加上 14 其实得到的二进制结果除了进位位，结果是一样的。

如果我们把 1110（14）的最高位看作符号位后就是（-2）的补码，这可能也是为什么负数的符号位是‘1’而不是‘0’，

而且在有符号位的四位二进制数中，能表示的只有‘-8~7’，而无符号位数（14）的作用和有符号数（-2）的作用效果其实是一样的。

那正数的补码呢？加上一个正数，加法器就直接可以实现。所以它的补码就还是它本身。

下图给出带符号位四位二进制的补码表示法



到这里，我们发现原码，反码的问题，补码基本解决了。

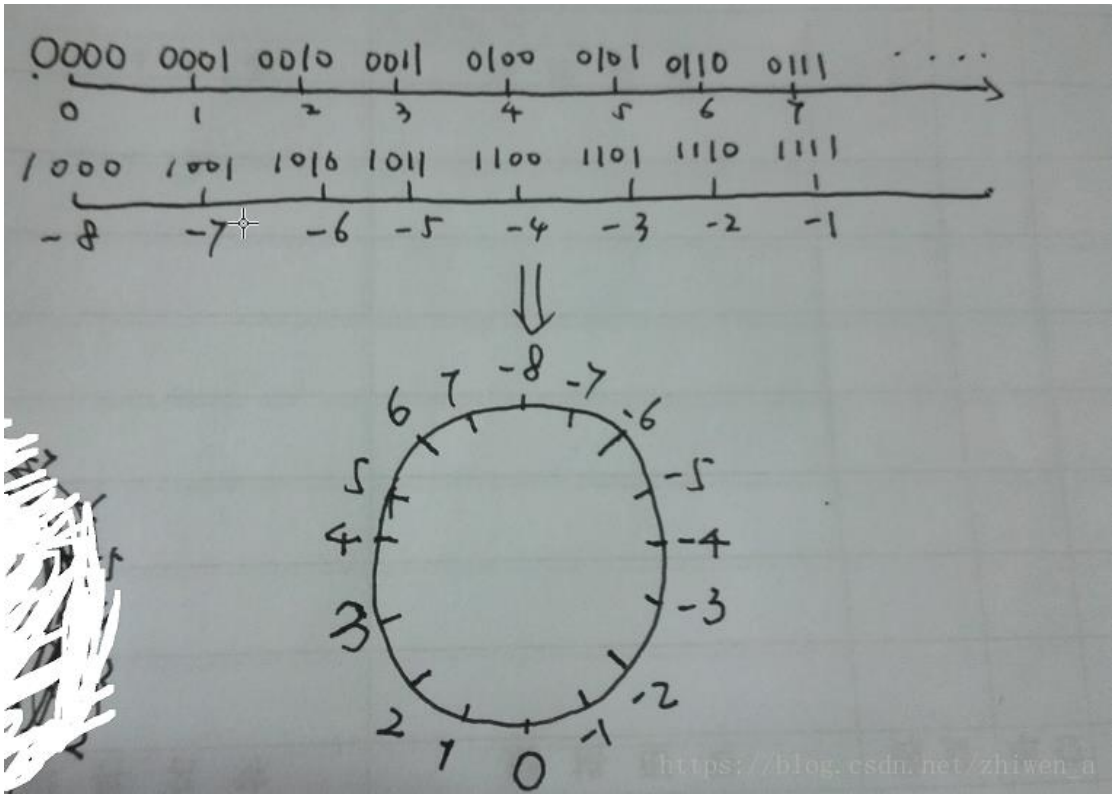
在补码中也不存在负零了，因为 1000 表示-8

这是因为根据上面的补码图，做减法时，0001（1）+1111（-1）=0000

我们再也不需要一个 1000 来表示负 0 了，就把它规定为-8

负数与负数相加的问题也解决了 1111（-1）+1110（-2）=1101(-3)

可能说得有点绕，但是实在是没办法。其实我觉得补码还可以这样画。



很优美有没有，如果你想想地理课本，0 不就相当于本初子午线，-8 不就是 180°，而正数相当于西经，负数相当于东经。

(七)为何这样求补码

然后我们再来看看为什么负数的补码的求法为什么是反码+1

因为负数的反码加上这个负数的绝对值正好等于 1111，再加 1，就是 1000，也就是四位二进数的模

而负数的补码是它的绝对值的同余数，可以通过模减去负数的绝对值，得到他的补码。

所以 负数的补码就是它的反码+1。

有点绕吧，只能说很难算清楚，你们还是自己算算吧。还有上面我提到的另外一种算法。

接下来，我要说一下我自己算补码的小技巧。

看上面那个图。

如果我们把-8 当成负数的原点。那么-5 的补码是多少呢？

```
-5=-8+3
```

-5 的补码就是-8 的补码加 3

```
1000（-8） +0011（3）=1011(-5)
```

所以完全可以口算出-5 的补码是 1011

当然，也可以记住-1 的补码是 1111 口算减法得出

对于八位加法器的话，可以把-128 当补码原点。十六位可以把-32768 当补码原点。

是的，128 是 256（八位二进制数的模）的一半，32768 是 65536（十六位二进数的模）的一半

也很方便有没有，而且简单的是

补码原点总是最高位是'1'，其他位是'0'

所以做加法总是简单得可以口算。

OK，原码，反码，补码之旅就到这里结束。补码第一次看总会觉得很绕，想言简意赅，就怕哪里遗漏了。讲得细致，又不免连自己都觉得啰里啰嗦。谢观！