编码

如果将编码本身当成二进制数来读，那么编码实际上是将一个数做一些运算，或者说，编码本身就是做一种映射。

- **首先是原码，用一个单独的位来表示正负是很显然的，一个数字的绝对值有一个二进制表示位：假设为1010，如果是正数，那么就在前面加0,为：01010，如果是负数，那就加1，为11010,这样的编码源于朴素的加一个符号位的思想，它可以表述为上述的规则，第一个位置为符号位，0表示正，1表示负，后面跟着原本那个数的绝对值。但是如果我们将原码当成二进制数来读，那么就有了一个公式是：**
- $$[x]_{原} = \begin{cases} x & 0 \leq x < 2^n \\ 2^n - x & -2^n < x \leq 0 \end{cases}$$

  这是对于定点整数，需要注意的是，这里的n表示的原本的数的位数，这是因为，我们加的那个符号位的权值是$2^{n+1-1} = 2^n$,x本身的绝对值大小为$0 \leq x 2^n(n$个位都是1最大，为$2^n - 1)$。当x<0时，符号位的1后面跟的是它的绝对值$= -x$,所以实际上是减去x
  如果是对于小数，那么符号位位于的是个位的地方，所以要把2^n改为1，即：

$$[x]_{原} = \begin{cases} x & 0 \leq x < 1 \\ 1 - x & -1 < x \leq 0 \end{cases}$$

再次强调，对于原码,它来源于加一个符号位的思想，但是将原码当成二进制数来读会得出这样的公式。当我们构建一个原码的时候，自然还是这样的思路最为简洁：第一个位置为符号位，0表示正，1表示负，后面跟着原本那个数的绝对值。

- 举例说明，对于定点整数，
  - 当数值位为3位时，算上符号位一共四位，此时，我们会发现，它能表示得数有15个，也就是0,1,2,3,4,5,6,7,-1,-2,-3,-4,-5,-6,-7,即7个正数，7个负数，还有0，有正0和负0。
  - 若原码整数的位数是8位,其表示的最大值、最小值 8位: 127，-127（2^7-1，也就是数值为七个位置都为1）

一般说位数说的是整个的字长，比如八位，那就是说有一位是符号位，剩下七位是数值位。

- 可以总结出，对于整数，它的数值为是n位，原码需要n+1位，左边第一位为符号位，可以编码的区间为：$-(2^n - 1)$ ~~~~ $2^n - 1$(整数)
- 对于定点小数则不同，因为要注意他们数值位的权重为2的负数幂，数值为是n位，原码需要n+1位，左边第一位个位为符号位，可以编码的区间是$-(1 - 2^{-n})$ ~~~~ $1 - 2^{-n}$(整数)
- 结论：
  - 原码为符号位加上数的绝对值，0正1负;
  - 原码零有两个编码，+0和 -0编码不同;
  - 原码加减运算复杂，乘除运算规则简单;
  - 码表示简单，易于同真值之间进行转换。

然而

> '编码作为一种运算，或者说映射'

给了我们启示，原码的思路固然简洁明了，但是这是对于人脑而言，人很容易分清符号位和数值位，但是对于计算机，是很难的，特别判断符号位然后再进行运算很浪费资源，并且原码还有一个致命的缺陷，那就是：对于0这个值它可以有两个，正0和负0，也就是

0.0000，和1.0000（以数值位为四位为例。所以我们想要找到一种运算，或者说一种映射，使得可以给0一个唯一的编码，并且最好能将正数和负数以及0用一个统一的运算来编码，模运算正好可以完成这个目标。

考虑$n+1$个位，这n+1个位作为二进制数可以表示的范围是0到$2^{n+1}-1$，也就是2^{n+1}个数(这比原码的表示多了一位),这2^{n+1}个数也正好是以2^{n+1}为模数的所有结果，我们用$mod\ (2^{n+1})$这种运算可以将这2^{n+1}个数编码（尽管他们本身就长这样，但是为了加深映射的思想，我们这里还是这样表述），但是这都是正数和0，我们希望给负数留一些位置，而模运算有一个特性是，$(x+a)\mod a=x\mod a$,因此，对于负数，我们加上模数再模，就可以转化为正数，当然，要满足绝对值小于等于模数。
好了，下面我们用mod运算来做一个统一的映射，我们的定义域是n位二进制数,也就是说他们的绝对是是n位的二进制数：-2^n到2^n-1,这正好是2^{n+1}个数，也就是 $\mod 2^{n+1}$的结果个数，映射方式是$[x]_{补}=(x+2^{n+1})\mod 2^{n+1}$。

这也就是补码的定义，分开来看

- 对于非负数x，$(x+2^{n+1})mod(2^{n+1})=x$
- 对于负数，$(x+2^{n+1})mod(2^{n+1})=(2^{n+1}-|x|)mod(2^{n+1}).$

需要注意的是，如果原本的数是n位的，比如四位 1011，那么我们加的这个
2^{n+1}并不是在左边加一位，而是在左边加2位，因为权值的幂是比位数少一的，也就是说，这里应该是$1011->101011->(101011)mod(2^5)->01011$,最终的补码位数又是n+1位的，这是因为n+1位的数才能有2^{n+1}位置来存数。

而对于负数，比如-1011，它的补码应该是：-1011 ->100000-1011->10101

倘若我们要归纳一个直接写出补码的口诀，那么，

- 对于整数或者0，就是在前面加一个0，作为符号位，后面就是它本身
- 对于负数，则有一个巧妙的方法：先在左边写一个1，然后把后面的数值位全部取反，然后再加1。
  - 这是为什么呢？这是因为，我们定义的补码映射里面，对于负数是 $(2^{n+1}-|x|)mod(2^{n+1})$,当做了$(2^{n+1}-|x|)$之后已经是一个介于0到$2^{n+1}$的数了,因此mod不用考虑，主要是前面这个$(2^{n+1}-|x|)$，我们考虑一个负数它的数值位是 1011，而我们的2^{n+1}是100000，用100000去减1011对于我们来说还是太麻烦了，因此我们考虑把100000写成011111+1，这样我们先用011111来减去，由于他全都是1，所以很好写出来答案就是10100，毕竟它不用借位每一位都直接减去，然后我们在把1加上来。
  - 为什么左边这一位可以当成符号位呢，因为我们加的模数是比我们x多两位的，所以说进行减法的时候左边多一位的这个位置一定是1，而只有负数是进行的减法。

写到这里，再次提醒不要忘记了，把编码当成一个二进制数来看，编码本身只是个映射而已。

- 奥对，上面还只是定点正数，对于定点小数呢，将模数变为2，也就是2^1，注意权值为2不是小数点左边第一位，而是第二位。上面的口诀还是适用。

下面我们来看为什么补码有这样好的性质：

- 补码最高一位为符号位，0正1负；
- 补码零有唯一编码；
- 补码能很好用于加减运算。
  (4) 特点
- 补码满足 $[-x]_{补}+[x]_{补}=0$
  - $[+7]_{补}=00111$，$[-7]_{补}=11001$
- 最高位参与演算，与其它位一样对待。

这些都是因为，定义本身：

$[x]_{补}=(x+2^{n+1})mod(2^{n+1})$,那么很显然任意两个数之和x+y的补码为$[x+y]_{补}=(x+y+2^{n+1})mod(2^{n+1})=(x+2^{n+1})mod(2^{n+1})+(y+2^{n+1})mod(2^{n+1})=[x]_{补}+[y]_{补}$
因此上述的结论都是因为mod运算的性质而已。
当补码运算发生溢出时，应当直接舍去，因为补码本身有一个mod的操作，而溢出，最多溢出一位，而无法溢出到2^{n+1}那一位。

- 补码还有一个性质是，扩展方便。5位的补码扩展为8位
  - 00111-> 00000111
  - 11001-> 11111001
- 这里的扩展之所以可以直接重复0和1，是因为在每扩展之前模数的影响只到这个位置，扩展之后后面的位是不会受到影响的（对于负数的那个情况的操作而言）

算术移位。假设 $[x]_{补}= x0.\ x1\ x2\ \cdots\ xn$,

$[x/2]_{补}= x0.\ x0\ x1\ x2\ \cdots\ xn-1$

原符号位不变，符号位与数值位均右移一位，

[X]补 =10010　　则　[X/2]补 =11001

> 最大的优点就是将减法运算转换成加法运算。
>
> $[X+Y]_{补}= [X]_{补} + [Y]_{补}$
>
> $[X-Y]_{补}= [X]_{补} + [-Y]_{补}$
>
> 例如：$X=(11)_{10}=(1011)_2$ ，$Y=(5)_{10}=(0101)_2$ ，已知字长n=5位，则
>
> $[X]_{补}+ [-Y]_{补}= 01011 + 11011 = 100110 = 00110 = (6)_{10}$
>
> 注：　最高1位已经超过字长故应丢掉。
>
> $[X - Y]_{补}= [0110]_{补}=00110$

关于补码的一些运算：

- 已知原码求补码
  - 正数 $[X]_{补} = [X]_{原}$
  - 负数 符号除外，各位取反，末位加1,相当于从真值转补码已经完成了添加符号位

例： X= -1001001

$[X]_{原}$=11001001

$[X]_{补}$=10110110+1=10110111

$[X]_{补}= 2^{7+1} +X=100000000-1001001= 10110111$

$$\begin{array}{r} 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ -\quad\ 1\ 0\ 0\ 1\ 0\ 0\ 1 \\ \hline 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1 \end{array}$$

- 由$[X]_{补}$求$[-X]_{补}$：将$[X]_{补}$连同符号一起将各位取反，末位再加1
  - 由于$[X]_{+}[-X]_{补} = 0_{补}$
  - 所以$[-X]_{补} = 0_{补} - [X]_{补}$
  - 所以$[-X]_{补} = (0 - X)mod(2^{n+1} = (-X)mod(2^{n+1})$
  - 如果-X是个负数，那么将等于$(2^{n+1} - |X|)mod(2^{n+1})$,而由于此时X是个正数，所以$(2^{n+1} - |X|)mod(2^{n+1})$的操作步骤是在X的数值前加一个符号位1，然后取反然后加1，但是由于$[X]_{补}$是X的数值前加了个0，所以这里只需要连同符号位一起取反再加1
  - 如果-X是个正数，那么X是个负数，$[X]_{补}$是将X的数值位前加了1，然后后面全部取反，又加了1，也就是说要得到$[-X]_{补}$，应该减去1，再把所有位都取反，而这相当于做了这样一个操作$11111111... - \ ([X]_{补} - 1) \ = 11111111... - ([X]_{补}) + 1$
    - 显而易见的是，$11111111... - \ ([X]_{补})$相当于给所欲的位全部取反，因此，由$[X]_{补}$求$[-X]_{补}$,被统一到：
    - **将$[X]_{补}$连同符号一起将各位取反，末位再加1**
- 补码转真值：
  - 对于正数，由于正数的补码就是在左边加个符号位0，所以后面的数值位就是真值
  - 对于负数，由于负数的补码是在数值的左边加个符号位1，取反再加1，那么想要转换成真值，应当先减去1，再对整体取反（符号位从1变成0可以归纳进全部取反），然后记得加个负号
    *以$[11101]_{补}$为例,先减去1->$[11100]$,再全部取反->$[00011]$,这就是数值了。但是这样计算很麻烦，我们考虑负数的补码：
    - $(x + 2^{n+1})mod(2^{n+1}) = (2^{n+1} - |x|)mod(2^{n+1}).$
    - 那么对于原本是n位数值位的负数，它的补码实际上是这样得到的，$2^{n+1} - |X|\text{mod}(2^{n+1})$,我们知道X是n位的，而$2^{n+1}$是出现在n+2位，它相当两个n+1位，即$2^{n+1} - |X|\text{mod}(2^{n+1}) = (2^n + 2^n - |X|)\text{mod}(2^{n+1})$,第一个$2^n$相当于在数值左边加了一位1作为符号位，第二个则是去减去数值位的，它相当于先变为11111...然后减去（方便计算，可以直接取反），然后再加1。因此对于补码它的数值位部分，相当于是一个$2^n-|X|$,而$|X|$的范围是不超过2^n$的,所以不会溢出。因此有了这样一个办法：
    - **负数的补码求真值，对于符号位不用管，后面的求补得到数值部分，或者另一个理解，将补码看成一个二进制数，但是第一个那个符号位，它的权值为负数，这样省的加负号了。**

反码

我们在补码的时候，对于负数使用了一个取反的技巧，那是因为我们将(2^{n+1})先减去1，变成全都是1的效果，然后再减，这样可以直接取反。对于反码，不使用模运算，在 处理负数的时候，直接用$(2^{n+1} - 1)$去减，于是有这样的效果：

- 对于正数，符号位为0，其余为本身。
- 对于负数，符号位为1，其余为本身取反
- 如果是小数，符号位是个位

不难发现，对于负数，补码就是反码加1.

移码

移码采用的映射是$[X]_{移} = X + 2^n$,对于整数，直接是在左边加一个1，对于负数，那就是先减去1，变为全都是1的状态，然后可以让X数值位全部取反，再加1

(3)移码的特点

- 在移码中，最高位为0表示负数，最高位为1表示正数，这与原码、补码、反码的符号位取值正好相反。
- 移码为全0时所对应的真值最小，为全1时所对应的真值最大!

因此，移码的大小直观地反映了真值的大小，这将有助于两个浮点数进行阶码大小比较。

- 真值0在移码中的表示形式是唯一的，即：

$$[+0]_{移} = [0]_{移} = 100\cdots00$$

- 移码把真值映射到一个正数域，所以可将移码视为无符号数，直接按无符号数规则比较大小。
- 同一数值的移码和补码除最高位相反外，其他各位相同。

几种机器编码简便方法对比

机器码	真值为正数	真值为负数
原码	符号位为零，等于真值本身	符号位为一，数值位为真值本身 简便编码方法：加符号位
补码	符号位为零，等于真值本身	符号位为一，逐位取反，末位加一
反码	符号位为零，等于真值本身	符号位为一，逐位取反
移码	符号为一，数值位为真值本身	符号位为零，数值位逐位取反，末位加一

40

码制表示法小结

[X]原、[X]反 、[X]补用“0”表示正号，用“1”表示负号；

[X]移用“1”表示正号，用“0”表示负号。

如果X为正数，则[X]原=[X]反=[X]补。

如果X为0，则 [X]补 、[X]移有唯一 编码，

[X]原、[X]反 有两种编码。

移码与补码的形式相同，只是符号位相反。