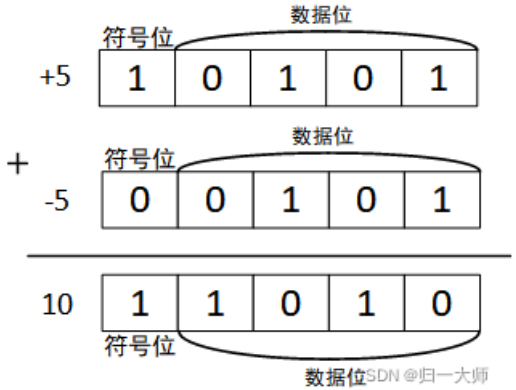
补码符号位的权重

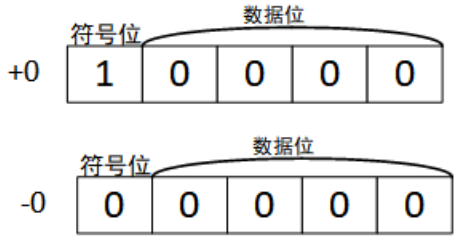
本文完整版链接，从另一个角度看补码，点击查看

补码的由来

由于原码在进行编码时采用了数制与码制相结合的方法，即 最高位 符号位采用0代表正数，1代表负数的码制编码方式，低位采用带有相应权重的数制编码方式进行编码；而码制是不能进行算术运算的，这就会出现5的原码0_0101， 等于10而不是等于0；



而且会出现正零0_0与负零1_0的表达方式，

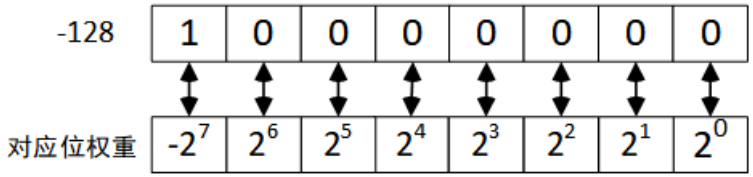


但实际并不存在正零和负零，造成编码的浪费；

补码权重

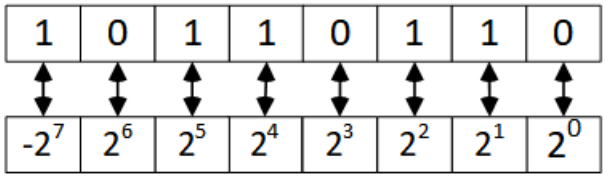
由于原码所存在的问题，就出现了补码，补码等于原码按位取反加一，补码采用数制编码方式进行编码，最高位（符号位）虽然依旧是0代表正数，1代表负数，但是却有权重（表示有大小关系），权重位-2^n；如下图所示：

这也很好理解八位补码中10000000（-128）为什么是最小的，因为最高位的权重为负，而其它位的权重为正，其它为一，就会使求和的结果的数值增加，如下图所示：



补码转十进制，十进制转补码

由于补码符号位具有权重，那么补码便可以直接转化位十进制数，例如：10110110对应位的权重如下图所示：

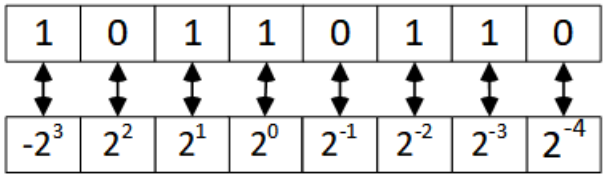


$10110110 = 1*(-2^7)+0*(2^6)+1*(2^5)+1*(2^4)+0*(2^3)+1*(2^2)+1*(2^1)+0*(2^0)$

$=-128+32+16+4+2$

$=-74$

而1101.0110对应位的权重如下图所示：



$1101.0110 = 1*(-2^3)+1*(2^2)+0*(2^1)+1*(2^0)+0*(2^{-1})+1*(2^{-2})+1*(2^{-3})+0*(2^{-4})$

$=-8+4+1+0.25+0.125$

$=-2.625$

以上就是二进制补码　转十进制的方法，相比于把补码转化成原码在转化成十进制快很多；

十进制转补码运算时采用反码加一也是比较方便，只是注意在对带有小数的源码取反之后，所加的这个一是加在最右边那一位之上的；

比如：

-2.625

2.625=110.101

原码：1110.101

反码：1001.010

得补码=1001.011

当然也可以不经过反码；

$-2.625=-4+1+0.25+0.125=1*（-2^2）+1*(2^0)+1*(2^{-2})+1*(2^{-3})$

所以补码=1001.011

比如：

$-93=-128+32+2+1=1*（-2^7）+1*(2^5)+1*(2^1)+1*(2^0)$

所以补码为10100011

补码位数的扩展就是符号位扩展

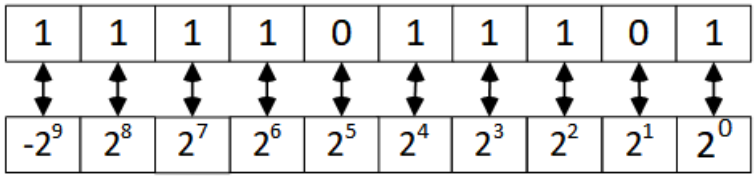
补码进行位数的扩展，直接在高位扩展符号位即可，正数最高位扩展0大小不变，而负数的补码n位补码为1010……1101，则最高位的权重为-2^{n-1}，当扩展成n+1位补码11010……1101时，变化的就是最高位权重以及次高位的权重，最高位的权重为-2^n，而次高位权重为2^{n-1},这两位求和为$1*（-2^n）+1*(2^{n-1})=-2^{n-1}$，大小不变；所以补码扩展位数直接对符号位进行扩展即可；

在对补码进行十进制转化时可能得到简化，比如：

1111011101=1011101=-2^6^+2^4^+2^3^+2^2^+2^0^=-64+16+8+4+1=-35

原因用图解的方式进行打开：

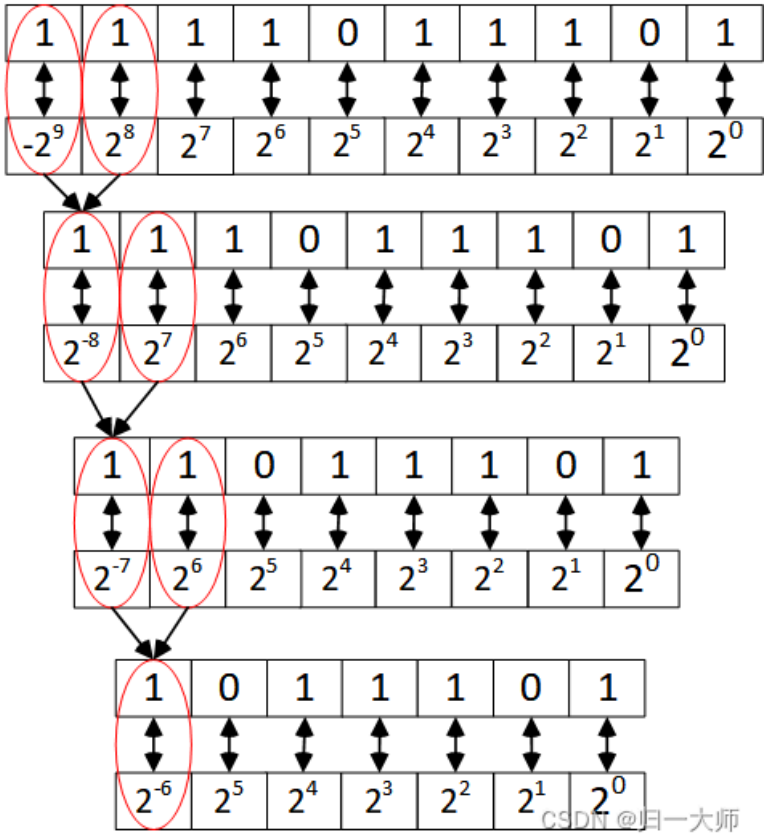
1111011101各位对应的权重如下：



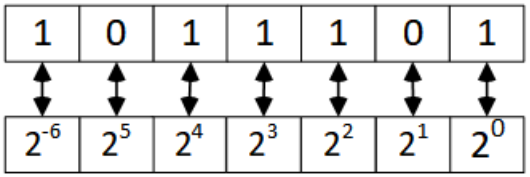
最高位和次高位因为都为1，但是所在位权重是相反的，相加的结果是1*（-2^9）+ 1*(2^8) = 1*（1-2）* 2^8 = -2^8，这不就是第九位为补码最高位时的权重？

还可以发现因为第八位也为1，这个运算结果还可以与第八位进行相同运算，1 *（-2^9）+ 1* 2^8 + 1 * 2^7 = 1*（1 + 2 - 4）* 2^7 = -2^7，这不就是第八为补码最高位时的权重吗？

当然这里因为第七位数据依旧是1，还可以与前面结果进行计算，得到的结果当然是-2^6。运算过程如下：



最终会发现补码 1111011101 的高四位全一的数据与权重相乘在求和运算的结果却与 1011101 的最高位权重是一致的，均为-2^6。



而其余位是相同的，这两个数相等；这也说明，补码在进行位扩展时，要保证大小不变，必须进行符号位的扩展；

您的支持是我更新的最大动力！将持续更新工程，如果本文对您有帮助，还请多多点赞👍、评论💬和收藏⭐！