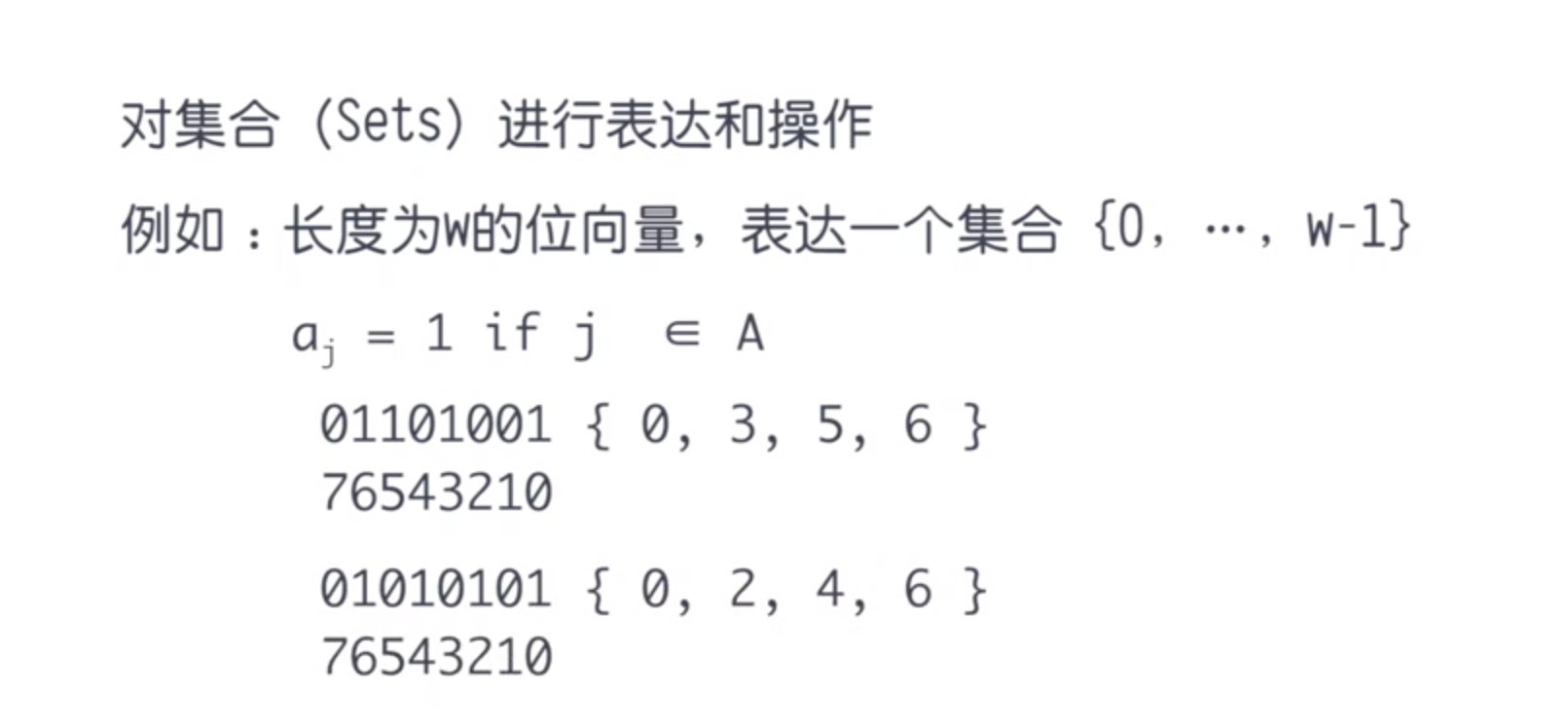
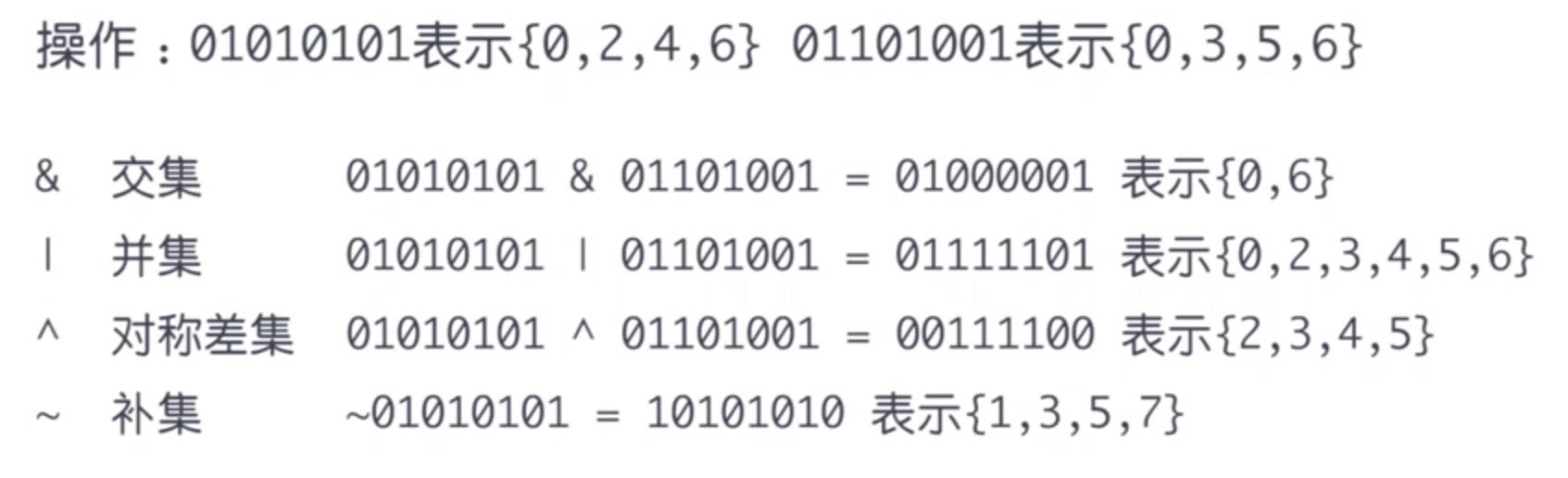
**按位运算的神操作**



我们可以注意 图片中，每个位置下标（0，1，2，3）如果这个二进制上面是1，代表这个下表数字存在，集合中3，在第四位二进制中就为1.

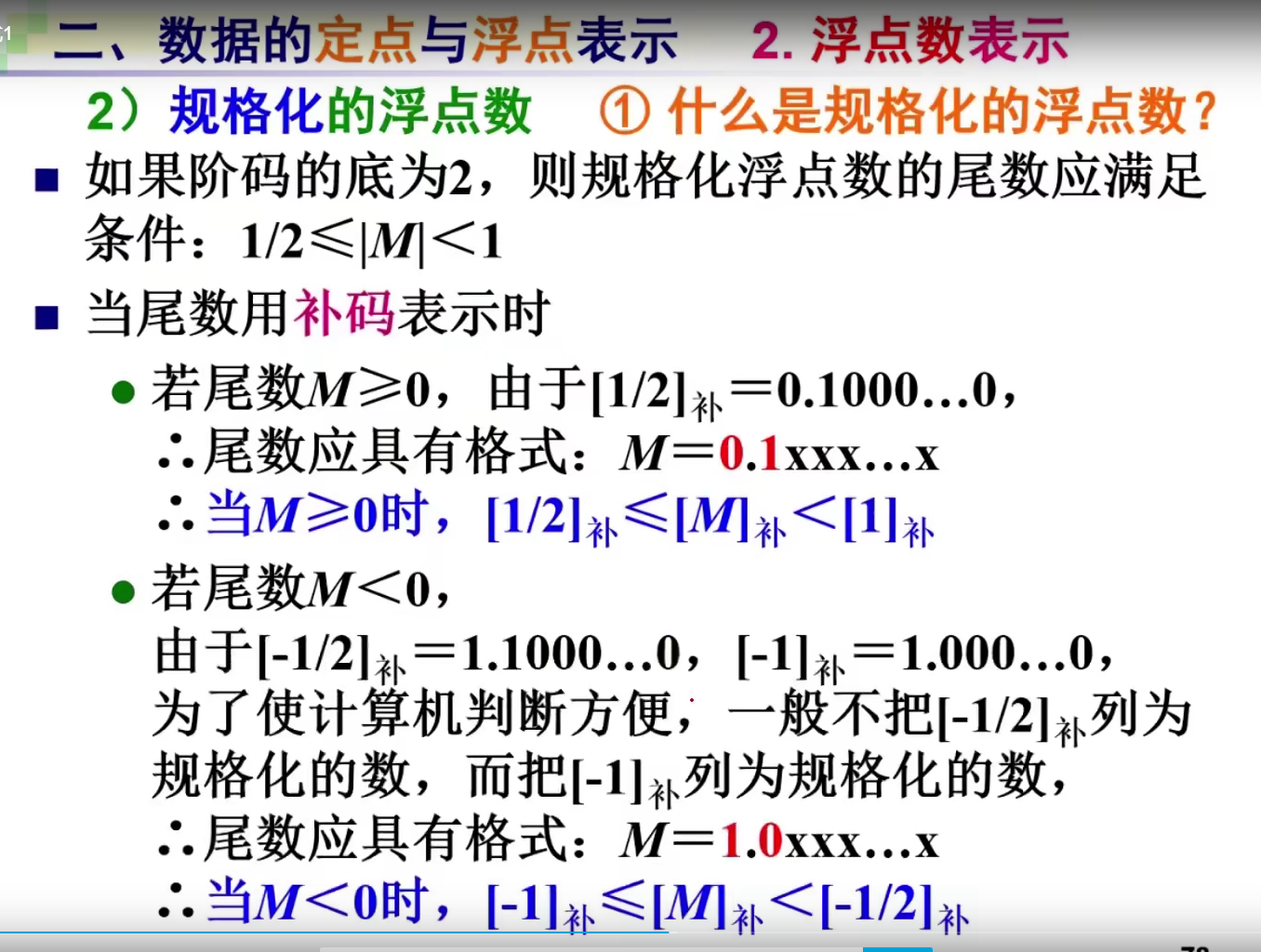


**思路打开**

不同编码真值不同，所以我们发现移吗可以做到方便比较大小



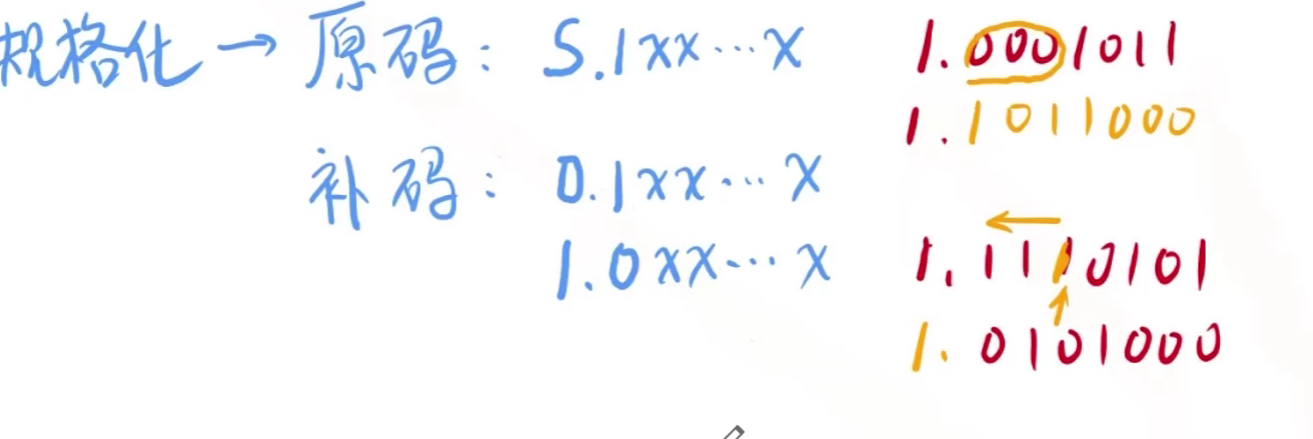
**浮点数的规格化（p28）**



浮点数的部分规格化 补码和原码的表示不同。

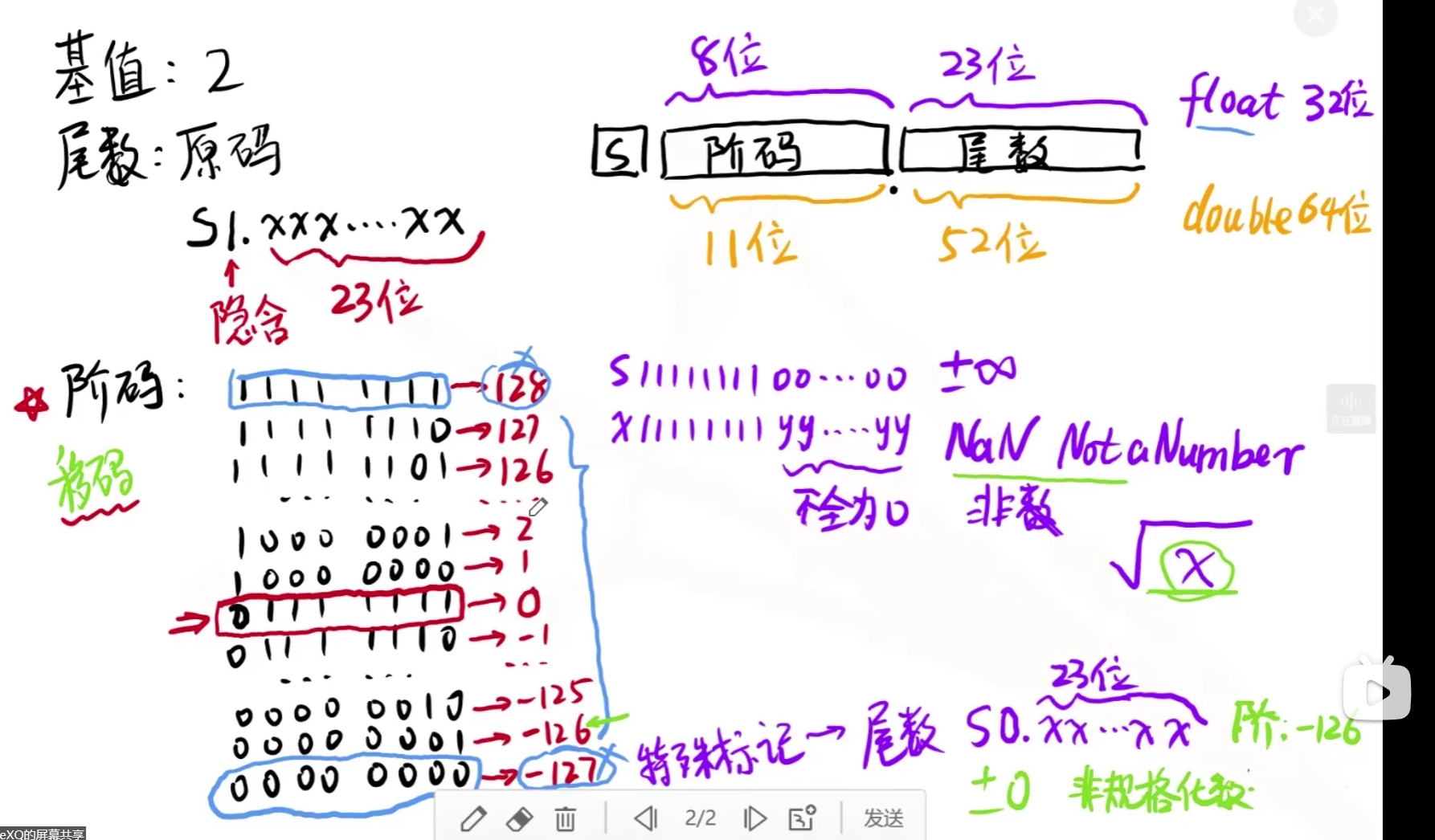
补码 当M<0我们定义 -1.0【补】=1.0000000，（原因看补码部分）。所以是右边图片的表示范围

源码 他就根本没有-1.0，也没有1.0，所以M>=0和补码一样，M<0，应该是1.1开头，就是符号位改变一下，因为原码负数和正数一样。补码不太一样



规格化的过程中，不标准，所以我们需要将源码和补码进行左右移动，源码他的符号位不要动进行移动，补码跟着符号位移动，所以得出有边的结果

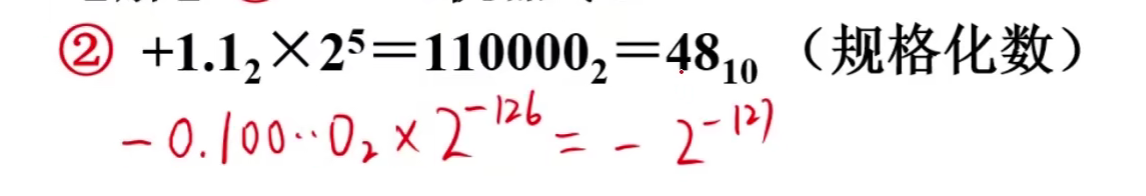
**IEEE -754标准**



1.我们注意到在float32位的时候 实际上尾数原码23位不加上符号位，当是在这个标准中我们假设 尾数的原码为了规范化（这个标准规范化），我们让小数点前的第一位都是一，既然都是1了，索性我们直接隐藏这个1，那么这个23位的尾数表示了24位。

2.阶码 使用移码表示，和我们前面学习不同的是，关于偏移量的选择，就像图片中的那样，而前面学习的是1000000（在这里补码符号位取反就不行了计算移码），所以数字表示范围反过来了，这里是正数128 负数127。

3.如果阶码是-127和128的时候，不再表示一个数了，而是一个特殊标记，如图所示，-127的时候（非规格化数），这个-127替代前面所说的隐藏1了，表示为0，如果后面23位，都是0的话就表示+-0了，这样做的好处就是，他定义了一个下限，阶码都是-127了，是不是2（-127）已经够小了，所以直接处理为0，这样防止了溢出不是吗 前面不是提到过如果是-127的时候就是非规格化数，不是我们说-127只是一个标记吗，但-127依旧比-126小，所以-127不能用，那就用-126表示最小



4.如果是128的时候，如图所示，定义了一个上限，下面的一行非数是表示，不是数字，比如根号-12，这个数字没有意义，就可以用它来表示。