# 电子计算机ЭВМ

▼ 编码

## 1. 原码

原码就是符号位加上真值的绝对值, 即用第一位表示符号, 其余位表示值. 比如如果是8位二进制:

> [+1]原 = 0000 0001
>
> [-1]原 = 1000 0001

第一位是符号位. 因为第一位是符号位, 所以8位二进制数的取值范围就是:

> [1111 1111 , 0111 1111]

即

> [-127 , 127]

原码是人脑最容易理解和计算的表示方式

## 2. 反码

反码的表示方法是:

*正数的反码是其本身*

*负数的反码*是在其原码的基础上, *符号位不变，其余各个位取反*.

> [+1] = [00000001]原 = [00000001]反
>
> [-1] = [10000001]原 = [11111110]反

可见如果一个反码表示的是负数, 人脑无法直观的看出来它的数值. 通常要将其转换成原码再计算

## 3. 补码

补码的表示方法是:

*正数的补码就是其本身*

*负数的补码*是在其原码的基础上, *符号位不变, 其余各位取反, 最后+1*. (即在**反码的基础上+1**)

> [+1] = [00000001]原 = [00000001]反 = [00000001]补
>
> [-1] = [10000001]原 = [11111110]反 = [11111111]补

对于负数， 补码表示方式也是人脑无法直观看出其数值的。通常也需要转换成原码在计算其数值

▼ 为什么需要？

计算十进制的表达式: 1-1=0

> 1 - 1 = 1 + (-1) = [00000001]原 + [10000001]原 = [10000010]原 = -2

如果用原码表示, 让符号位也参与计算, 显然对于减法来说, 结果是不正确的.这也就是为何计算机内部不使用原码表示一个数.

为了解决原码做减法的问题, 出现了反码:

计算十进制的表达式: 1-1=0

> 1 - 1 = 1 + (-1) =
> =[0000 0001]原 + [1000 0001]原=
> =[0000 0001]反 + [1111 1110]反 =
> =[1111 1111]反 = [1000 0000]原 =
> =-0

发现用反码计算减法, 结果的真值部分是正确的. 而唯一的问题其实就出现在"0"这个特殊的数值上. 虽然人们理解上+0和-0是一样的, 但是0带符号是没有任何意义的. 而且会有[0000 0000]原和[1000 0000]原两个编码表示0.

于是补码的出现, 解决了0的符号以及两个编码的问题:

> 1-1 = 1 + (-1) =
> =[0000 0001]原 + [1000 0001]原 =
> =[0000 0001]补 + [1111 1111]补 =
> =[0000 0000]补=[0000 0000]原

- 1-127的结果应该是-128, 在用补码运算的结果中, [1000 0000] 就是-128. 但是注意因为实际上是使用以前的-0的补码来表示-128, 所以-128并没有原码和反码表示.(对-128的补码表示[1000 0000]补算出来的原码是[0000 0000], 这是不正确的)

  补

  原

使用补码, 不仅仅修复了0的符号以及存在两个编码的问题, 而且还能够多表示一个最低数. 这就是为什么8位二进制, 使用原码或反码表示的范围为[-127, +127], 而使用补码表示的范围为[-128, 127].

因为机器使用补码, 所以对于编程中常用到的32位int类型, 可以表示范围是: [-231, 231-1] 因为第一位表示的是符号位.而使用补码表示时又可以多保存一个最小值.