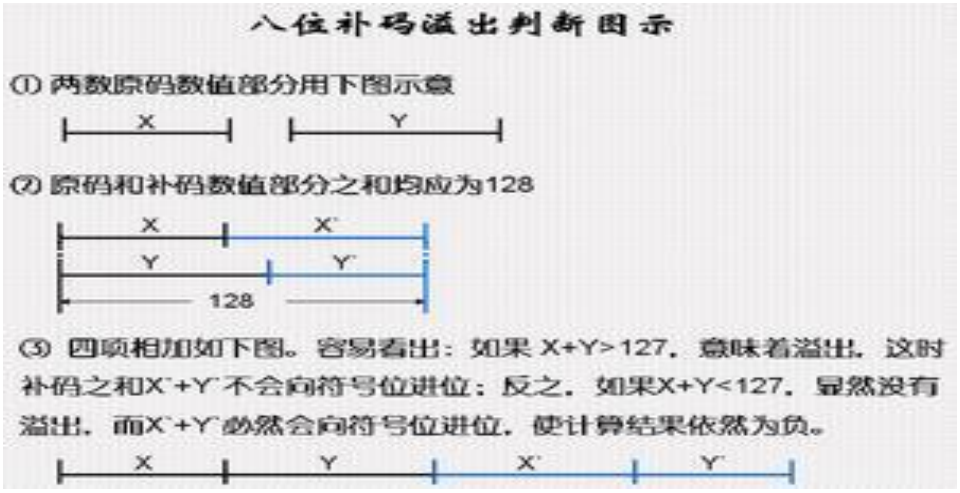
溢出判断

补码运算时的溢出判断



当两个以补码表示的负数相加时，会遇到两个问题。第一是两个负数的符号位相加，1+1 后，本位为零，似乎负数相加变成了正数；其二是两个负数的数值部分之和，如果不向符号位进位，是不是就说明运算结果没有溢出？但不进位最终将导致两个负数相加成了正数，显然是错误的，这该怎么解释？如果两个以补码表示的负数的数值部分之和向符号位进位，会使运算结果依然为负数，那么这个运算结果是正确的吗？下面我们分析一下这个问题：

①只有真正意义上的相加才可能溢出，比如：

正+正，负+负，正-负，负-正

纯粹的减法是不可能溢出的，这一点仅需常识即可作出判断，所以遇到不是真正意义上的加法运算（当然，包括乘法和左移等）要你判断是否有溢出，直接就可以回答：OF=0；

②两正数之和的数值部分向符号位进位,显然是运算结果超过了指定位数的带符号数的表示范围，这就是典型的溢出；

③两负数之和的溢出判断是我们讨论的重点。我们先考察一下负数原码和补码数值部分之间的关系：以 8 位补码为例，负数原码和补码数值部分之和始终等于 128（见上图）。由于这种关系，当原码数值大时对应的补码数值就小，反之也一样。所以，当两补码表示的负数的数值部分之和没有向符号位进位，说明两负数的原码之和必然向符号位进位，即发生溢出；反之，当两补码表示的负数的数值部分之和向符号位进位,那么对应两负数原码的数值之和就不可能向符号位进位，即运算结果没有溢出；并且在这种情形下补码之和的数值部分向符号位的进位,修正了两负数符号位相加本位为零的问题,使得两负数之和依然是个负数。

下面看两个负数补码相加溢出判断的实例：

例一：085h + 9ch

= 10000101b + 10011100b

两数相加，数值部分不会向符号位进位，这是不是就说明没有溢出呢？但由于计算结果为正，显然不对。我们还是看看两个数的原码之和再说：

10000101b 的原码 = 11111011b(-123)

10011100b 的原码 = 11100100b(-100)

显然，原码之和的数值部分将向符号位进位，显然是溢出无疑。

例二：0e7h + 0b3h

=11100111b + 10110011b

两数相加，数值部分会向符号位进位，这进位是溢出吗？还是看看原码吧!

11100111b 的原码 = 10011001b(-25)

10110011b 的原码 = 11001101b(-77)

容易看出，两数原码之和没有向符号位进位，即没有发生溢出。

其实归结起来，补码的溢出判断规则就一句话：

同号数相加如果结果的符号位和两加数不同，既是溢出。

这自然说明了：

⑴不是同号数相加，则不可能溢出；

⑵同号数相加有可能溢出；

⑶同号数相加如果结果的符号位和两加数不同，既是溢出。

补码加、减运算规则及溢出判断

1、运算规则

[X + Y]补= [X]补 + [Y]补

[X - Y]补= [X]补 + [- Y]补

若已知[Y]补，求[- Y]补的方法是：将[Y]补的各位（包括符号位）逐位取反再在最低位加 1 即可。

例如：[Y]补= 101101 [- Y]补= 010011

2、溢出判断，一般用双符号位进行判断：

符号位 00 表示正数 11 表示负数

结果的符号位为 01 时，称为上溢；为 10 时，称为下溢

例题：设 x=0.1101，y= - 0.0111，符号位为双符号位

用补码求 x+y，x - y

[x]补+[y]补=00 1101+11 1001=00 0110

[x - y]补=[x]补+[- y]补=00 1101+00 0111=01 0100

结果错误，正溢出