王海印　编著

微型计算机的原理、安装与使用

科学出版社

微型计算机的原理、安装与使用

王海印 编著

[illegible]出版社

（京） 新登字 092 号

内 容 简 介

本书深入浅出地讲述了计算机的工作原理和基本概念，介绍了计算机中常用的芯片和电路板，如CPU、存储器、I/O接口部件等，以及计算机的外部设备和操作系统、硬件与软件的安装方法等。本书是作者在多年教学经验的基础上写成的，实用性非常强。通过本书不但能从理论上较深入地了解计算机，而且能够掌握计算机的选购、安装和使用技巧，提高计算机的使用效率。

本书可作为大学和中等专科学校微机概论的教材、微机原理的参考书，也可作为中等文化程度的人员自学、培训用书。

图书在版编目(CIP)数据

微型计算机的原理、安装与使用/王海印编著. —北京：
科学出版社，1996
ISBN 7-03-005361-3

Ⅰ.微… Ⅱ.王… Ⅲ.微型计算机-基本知识 Ⅳ.TP36

中国版本图书馆CIP数据核字(96)第05714号

科学出版社出版
北京东黄城根北街16号
邮政编码：100717

中国科学院印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1996年10月第 一 版 开本：787×1092 1/16
1996年10月第一次印刷 印张：18 3/4
印数：1—6 700 字数：431 000

定价：22.00元

前　言

随着微型计算机价格的降低和可靠性的提高，其应用日益广泛。事实上，微型计算机的使用已不再是少数专业人员的事情，社会各界和家庭都正在越来越多地使用计算机。

大量关于计算机操作、语言和文字处理方面的图书，对计算机的普及起了推动作用。但是，一般人对计算机的基本原理并不太了解，对计算机仍有神密感。即使能够上机操作，甚至可以编一些简单程序的人，对一些基本概念也不是十分清楚。计算机的飞速发展和新元件、新术语的不断出现，更使人感到茫然。因此，人们迫切地需要能用较少的时间深刻理解计算机的工作原理和实质。

从这一情况出发，本书运用深入浅出的讲述方法，讲解了计算机工作的原理和基本概念；介绍了计算机中常用的芯片和电路板、计算机的外部设备和操作系统、硬件与软件的安装方法等。以期读者通过阅读本书，不仅从理论上能得到一定的提高，而且能够掌握计算机的安装、使用技术。

本书可作大学和中专微机概论的教材、微机原理的参考书，也可作为自学和培训用的微机原理的教材。

只要具备数字逻辑电路和二进制的基本知识的读者都可以从第三章开始阅读，而缺乏这方面知识的读者，通过第一和第二章的学习也完全可以理解本书的以后各章。

为保持本书的逻辑性和简捷性，正文中只叙述了基本部分。附录一至附录六提供了较为详细的资料，以作为对正文的补充，供需要较详细了解的读者阅读。

本书的前六章由王海印同志负责编写，后四章由王军同志负责编写，并由王海印同志负责全书的统稿工作。由于作者水平有限，不当之处望广大读者指正。

本书插图由王芸绘制，在此表示感谢。

目　录

第一章　数与符号

电子计算机快速而准确的计算能力常令人感叹不已，而实际上，计算机的神奇计算能力是建立在一种比我们熟悉的十进制数更简单的数制——二进制数的基础上的。计算机只认识“0”和“1”，或者说电平的“高”与“低”。大千世界的各种事物、数、符号和代码要让计算机理解和进行处理，就必须变换为用“0”和“1”的组合所表示的数。这种只用“0”和“1”组成的数制叫做二进制。由此可见，要和计算机打交道，就必须了解二进制。

§1-1　二进制

一、十进制

人最熟悉、最常用的数制是十进制。十进制有0到9共10个数字，数字的个数称为基数。十进制的基数是10。每个数所处的位置不同其值也不同，例如1122.34就是多项式 $(1\times10^3)+(1\times10^2)+(2\times10^1)+(2\times10^0)+(3\times10^{-1})+(4\times10^{-2})$ 的缩写。

在十进制中，每个数都要乘以基数10的幂次。上列数字的两个1，一个1在千位，它表示 1×10^3；另一个1在百位，它表示 1×10^2。其中 10^3 和 10^2 分别表示千位和百位的“权”。

在上例中，用十进制数小数点把数分为整数部分1122和小数部分0.34。

可见数制有下列特征：

(1) 数字的个数等于基数（十进制中为10）；

(2) 最大的数字（十进制中为9）比基数少1；

(3) 每个数字，根据它所处的位置，乘以该位的“权”，即基数的幂次。

(4) 进位时逢基数进1，十进制中为逢十进一。

二、二进制

二进制中，每一位只能取0或1，即基数为2。进位时“逢二进一”。例如，二进制数1111.1111相当于十进制的

$(1\times2^3)+(1\times2^2)+(1\times2^1)+(1\times2^0)+(1\times2^{-1})+(1\times2^{-2})+(1\times2^{-3})+(1\times2^{-4})=8+4+2+1+0.5+0.25+0.125+0.0625=15.9375$

可见，各位的“权”分别为8，4，2，1，0.5，0.25，0.125，0.0625。

三、十进制数和二进制数的相互转换

1. 二进制数转换为十进制数

二进制数转换为十进制数比较简单，就是上述例子中所用的方法，为了区别所表示

的是二进制数还是十进制数，常用下标来加以区分。下面再举两个例子加以说明。

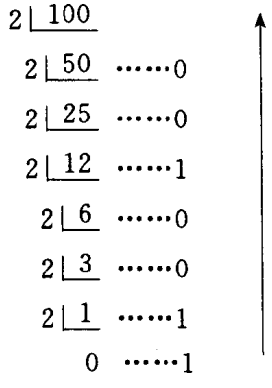
$(1100100)_2=(1\times2^6+1\times2^5+0\times2^4+0\times2^3+1\times2^2+0\times2^1+0\times2^0)_{10}=(64+32+0+0+4+0+0)_{10}=(100)_{10}$

$(0.001)_2=0\times2^{-1}+0\times2^{-2}+1\times2^{-3}=(0.125)_{10}$

2. 十进制数转换为二进制数

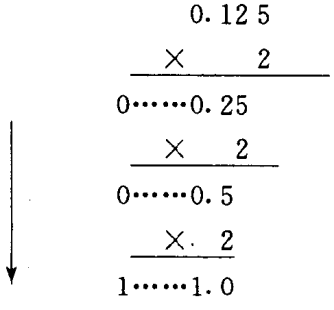
十进制整数转换为二进制数的方法是把该十进制数连续用基数 2 来除，依次记下余数，直到不能除为止。

第一个余数即为整数部分的最低位，最后一个余数即为整数部分的最高位，可概括为“除 2 取余”。比如，把十进制数 100 转换为二进制数，方法为



即 $(100)_{10}=(1100100)_2$

十进制数的小数部分要转换成二进制数时，可以把十进制数乘以基数 2，乘积的整数部分即为所求小数的最高位。把乘积的小数部分再乘以 2（乘积的整数部分不再乘以 2），这一次乘积的整数部分即为二进制小数的第二位。依此类推，直到不能进行或达到所需的精确度时为止。这一方法可概括为“乘 2 取整”。比如，把十进制数 0.125 转换为二进制的方法是



即 $(0.125)_{10}=(0.001)_2$

对于混合小数，要把整数部分和小数部分分别换算，然后再把整数部分和小数部分加起来。

四、二进制数的运算

（1）二进制加法的规则

0+0=0

0+1=1

1+0=1

1+1=10

例如：

$$\begin{array}{r} 1110 \\ +1100 \\ \hline 11010 \end{array}$$

（2）二进制减法的规则

0−0=0

1−0=1

1−1=0

10−1=1

例如：

$$\begin{array}{r} 10110 \\ -\ 1100 \\ \hline 1010 \end{array}$$

（3）二进制乘法的规则

0×0=0

0×1=0

1×0=0

1×1=1

例如：

$$\begin{array}{r} 1101 \\ \times 1011 \\ \hline 1101 \\ 1101 \\ 0000 \\ +1101 \\ \hline 10001111 \end{array}$$

（4）二进制除法

$$\begin{array}{r} 10100\cdots\cdots\text{（商）} \\ 1001\overline{)10110101} \\ \underline{1001} \\ 1001 \\ \underline{1001} \\ 001\cdots\cdots\text{（余数）} \end{array}$$

§1-2 八进制和十六进制

由于二进制码记忆不便，且书写容易出错，所以人们希望有一种较为简短和容易记忆的表示方法，于是十六进制和八进制应运而生。

多数人认为十六进制的代码表示法最好。十六进制是把四位二进制数作为一组用一个符号来表示，这些符号就是数字0—9和字母A—F。

与十六进制不同的另一种码制是八进制。在八进制中，二进制数被三位分为一组，每一个八进制位表示十进制数的0—7。表1.1是各种数制的对照表。

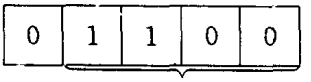
表1.1 各种数制对照表

十进制	二进制	八进制	十六进制	十进制	二进制	八进制	十六进制
0	0000	0	0	8	1000	10	8
1	0001	1	1	9	1001	11	9
2	0010	2	2	10	1010	12	A
3	0011	3	3	11	1011	13	B
4	0100	4	4	12	1100	14	C
5	0101	5	5	13	1101	15	D
6	0110	6	6	14	1110	16	E
7	0111	7	7	15	1111	17	F

这两种数制（十六进制和八进制）与十进制数之间的数据转换，与二进制和十进制之间的转换方法类似，不再赘述。当把十六进制数和八进制数转换为十进制数时，也可以先把它们变为二进制数，然后按二到十进制的转换方法进行转换，反之亦然。

§1-3 原码、反码和补码

在代数中正数和负数分别用“+”号和“−”号来表示。由于计算机只能识别“0”和“1”两种信息，所以“+”号和“−”号也必须用“0”和“1”来表示。通常用“0”表示“+”号，“1”表示“−”号，符号位设在数的最高位之前。例如十进制数+12和−12可分别用带符号的二进制数表示如下：



符号位+ 数12

符号位− 数12

一、数的定点制和浮点制

在计算机中用二进制表示数，有定点制和浮点制两种表示方法。

在定点制中，小数点的位置是固定不变的，规定放在符号位之后和数的最高位之前，因此参与运算的数N的绝对值应小于1。运算后的结果的绝对值也应小于1，否则将产生

“溢出”。定点机产生“溢出”后，将迫使计算机停止原有的运算，转入“溢出”错误处理。

若参加运算的数是大于 1 的数，则应将其变为纯小数与一个基数的乘方来表示。例如，十进制数 $A=25.02$，$B=12.12$，根据定点数的要求，A 表示为：0.2502×10^2，B 表示为 0.1212×10^2，求其和时，先尾数相加：$0.2502+0.1212=0.3714$，再将结果 0.3714 乘上 10^2，即为正确的结果 37.14。

若有两个二进制数 $A=1001.101$ 和 $B=11.011$ 相加，则应改写为 $A=0.1001101\times10^{100}$，$B=0.0011011\times10^{100}$，其中 0.1001101 和 0.0011011 称为尾数，指数 100 称为阶。按定点计算时，把尾数相加：

```
    0.  1 0 0 1 1 0 1     被加数
+   0.  0 0 1 1 0 1 1     加  数
    0.  1 1 0 0 1 0 0       和
        ↑ ↑   尾数
        | 小数点
符号位
```

计算得到的尾数和为 0.1101000，再乘以 10^{100} 即得真正的和。计算前对尾数相加的结果要加以估计，应该为小于 1 的数，否则把小数点再向左移，使尾数的和小于 1，以不产生“溢出”。

在浮点制中，数的阶是变化的，即小数点的实际位置是变动的。一个数用浮点制表示时要用二组数码，即一组表示尾数，一组表示阶。例如，二进制数 $1001.101=0.1001101\times10^{100}$，指数 100 为二进制数，即相当于十进制数 4。此数在计算机中用浮点制表示为：

二、原码

一个数用原码表示时，符号位用 0 表示正数，用 1 表示负数，其余各位表示尾数本身。

正数 $P=0.1101$ 的原码表示为

```
  0 1 1 0 1
    ↑  尾数
符号位
```

负数 $N=-0.1101$ 的原码表示为

```
  1 1 1 0 1
    ↑  尾数
符号位
```

对于0，可以认为它是（+0），也可以认为它是（−0），因此0在原码中有两种表示方法：

$$(+0)=0.0000$$
$$(-0)=1.0000$$

采用原码进行加减运算比较复杂。例如，进行加法运算时，计算机首先要判断两个数的符号是否相同。如果相同，则进行加法；如果符号不同，则要进行减法。而且要判断哪一个数的绝对值大，再由大数减去小数，并给出正确的符号。设计这样的计算比较复杂，所以在计算机中多采用补码进行计算。

三、反码

反码也称1的补码，可以这样来定义：

1. 当$1>x\geq 0$时，x的反码$[x]_{反}$就是x本身。

例如，$x=+0.1001$，则

$[x]_{反}=$ | 0 | 1 0 0 1 |

符号位 ↑ 尾数

2. 当$0\geq x>-1$时，把x的每一位变反，即原来为0的变为1，原来为1的变为0，然后再在符号位上放1即得x的反码。

例如，$x=-0.1001$，则

$[x]_{反}=$ | 1 | 0 1 1 0 |

符号位 ↑ 尾数

3. 如果x为0，它的反码有$[+0]_{反}$和$[-0]_{反}$两种表示法。

$[+0]_{反}=$ | 0 | 0 0 0 0 |

$[-0]_{反}=$ | 1 | 1 1 1 1 |

综上所述，反码的定义为：

设$x=+0.x_1x_2\cdots\cdots x_n$

或$x=-0.x_1x_2\cdots\cdots x_n$

若x_i变反后用$\overline{x_i}$表示，则

$$[x]_{反}=\begin{cases}x & \text{当 } 1>x\geq 0 \text{ 时}\\ 1.\ \overline{x_1}\,\overline{x_2}\,\overline{x_3}\cdots\overline{x_3} & \text{当 } 0\geq x>-1 \text{ 时}\end{cases}$$

四、补码

1.、模为12的补码

为了理解补码的意义，先举一个钟表对时的例子。设准确的时间为2点整，而有一个钟却已9点整。为此要把时针倒拨7格，也就是$9-7=2$。另一种方法是把时针向前拨5格，也可使它指到2点整。这是因为时针按正方向转动时，到12点就从0重新开始，相当于自动丢失一个数12，因此把时针向前拨5格时有

$$9+5=12\text{（自动丢失）}+2=2$$

这个自动丢失的数（12）就叫做“模”。上述加法可称为“按模 12 的加法”，用数学符号可写为

$$9+5=2\ (\mathrm{mod}\ \ 12)$$

由于时针转一圈会自动丢失一个数12，因此可以把 $9-7=2$ 这个减法运算改为下列加法运算：先把 12 加到减数（-7）上，即 $12+(-7)=5$，然后把 9 和 5 相加，

$$9+[12+(-7)]=9+5=12+2=2$$

（12 自动丢失）

模和某个数 x（例如 -7）相加所得的数 $[12+(-7)=5]$ 称为该数的补数，用 $[x]_{补}$ 表示，即

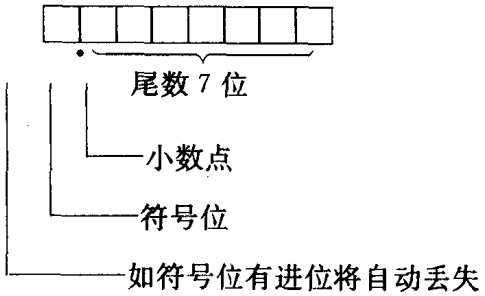
$$[x]_{补}=模+x$$

上例中 $[x]_{补}=12+(-7)=5$。找出 $x=-7$ 的补数 $[x]_{补}=5$ 以后，原来的减法 $9-7=2$ 就可以改用加法来做：$9+5=12$（自动丢失）$+2=2$。

为了求出（-7）的补数，仍然用了减法（$12-7$），并没有避免减法运算。下面将会看到，在计算机中可以用其它方法求二进制数的补码。

2. 模为 2 的补码

设采用定点值，数的绝对值均小于 1，在数的前面有一个表示正负的符号位，小数点在符号位之后和数的最高位之前。如计算机为 8 位，数的格式如下：



在上述格式中，对于正数小数点前有一个符号位 0，对负数来说符号位上有一个 1。如果符号位上再加一个 1 就要产生进位，但是由于计算机符号位前没有数值了，所以符号位产生的进位将自动丢失。这个丢失的进位数为二进制数 10，即十进制数 2，因此采用以 2 为模的补码。

首先通过例子来看一个正数的补码。例如 $x=+0.1101$，对 2 的补码为

$$[x]_{补}=模+x=10.0000+0.1101=1\,0.1\,1\,0\,1=0.1101$$

（1 自动丢失）

当计算机小数点前只有 1 位数时，上式中小数点前第二位的“1”自动丢失，故正数 x 的补码 $[x]_{补}$ 就是正数 x 本身。

对于负数来说，如 $x=-0.1011$，其补码为

$[x]_{补}=$模$+x=$ $(10.0000)+(-0.1011)$

由于二进制数 $10.0000=1.1111+0.0001$，故 $[x]_{补}=(1.1111+0.0001)-0.1011$ $=(1.1111-0.1011)+0.0001=1.0100+0.0001$。

仔细观察就可以发现，上式结果中的第一项 1.0100 正好是所给的要求其补码的负数 -0.1011 的反码，即原来为 1 的位变为 0，原来为 0 的位变为 1。由此可以引入求负数的补码的方法如下：

(1) 求负数 x（如 $x=-0.1011$）的反码，即把负数 x 的每一位求反（0 变为 1，1 变为 0）。在本例中，$[x]_{反}=1.0100$。

(2) 在反码的最后一位加 1，即得负数 x 对 2 的补码。在本例中 $[x]_{补}=1.0100+0.0001=1.0101$。

求补码的一般表示式为：$[x]_{补}=[x]_{反}+2^{-m}$，其中 m 为小数点后数字的个数（本例中 $m=4$）。

在计算机中对每一位求反，再在最后一位加 1 是很容易用硬件实现的，所以在计算机中求负数的补码不必做减法。

求负数 x 对 2 的补码还有一种方法。从负数 x 的最低位开始向左检查，在遇到第一个 1 以前，包括第一个 1，重写每一位，即原来是 0 仍写 0，遇到的第一个 1 仍写 1，以后使所有各位均取反。例如 $x=-0.1010$，求它的补码时，最后 1 位的 0 和倒数第二位的 1 均保持不变，其余各位均变反，可得到 $[x]_{补}=1.0110$。

对于补码来说 $[+0]_{补}$ 和 $[-0]_{补}$ 均为模 2（二进制 10）加零，由于模 10 中的 1 自动丢失，故在计算机中，$[+0]_{补}$ 和 $[-0]_{补}=0.0000$。

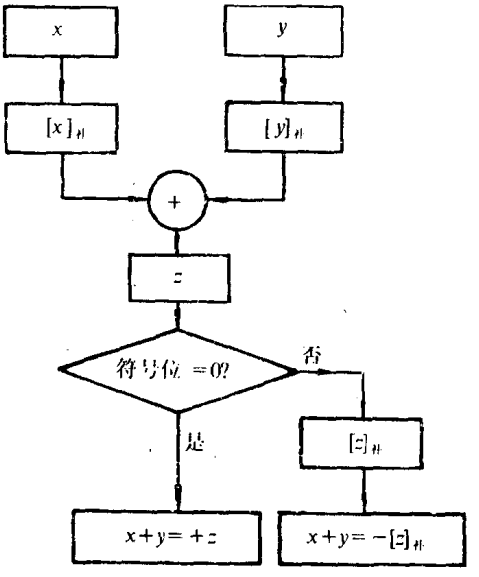


图 1.1 用补码进行加法计算

下面以实际例子说明，不论 x 和 y 是正数还是负数，下列公式都是成立的：

$$[x]_{补}+[y]_{补}=[x+y]_{补}$$

如果要将结果用原码表示，可作如下进一步处理。设上式中 $[x+y]_{补}=z$。如果 z 的符号为 0，表示是正数，因为正数的补码就是该数本身，故有 $(x+y)=(x+y)_{补}=z$。如果 z 的符号位为 1，则表示 z 是一个负数的补码，将补码再求补可获真值，故 $(x+y)=-[z]_{补}$。

因此，可以将 x 和 y 变为补码，按上述补码相加的公式求出 $z=[x+y]_{补}$，然后根据 z 的符号位的情况，再求出 $x+y$。这一计算过程可用图 1.1 表示。

3. 用补码表示时的加法运算

根据 x 和 y 的符号及绝对值大小，可分四种情况，下面讨论每种情况下的加法运算过程。

(1) x 为正，y 为负，且$|x|\geq|y|$，即 $1>(x+y)\geq 0$ 时，求 $(x+y)$ 的值。

例 1：设 $x=+0.1101$，$y=-0.0101$，求 $x+y$。

解：$[x]_{补}=0.1101$，$[y]_{补}=1.1011$，

$$\begin{array}{rl} & 0.\ 1\ 1\ 0\ 1=[x]_{补} \\ +) & 1.\ 1\ 0\ 1\ 1=[y]_{补} \\ \hline \text{自动丢失}\leftarrow\boxed{1} & 0.\ 1\ 0\ 0\ 0=z \end{array}$$

直接进行 $(x+y)$ 计算，即

$$x+y=0.1101+(-0.0101)=0.1000$$

这是一个正数，它的补码 $[x+y]_{补}$ 等于 $(x+y)$。由此证明，当 z 的符号位为 0 时，$z=[x+y]_{补}=(x+y)=0.1000$。

(2) 如 x 为正，y 为负，且$|y|>|x|$，这时 $(x+y)$ 为负，它的值在 0 和 -1 之间。

例 2：设 $x=0.0101$，$y=-0.1101$，求 $x+y$。

解：$[x]_{补}=0.0101$，$[y]_{补}=1.0011$，

$$\begin{array}{rl} & 0.\ 0\ 1\ 0\ 1=[x]_{补} \\ +) & 1.\ 0\ 0\ 1\ 1=[y]_{补} \\ \hline & 1.\ 1\ 0\ 0\ 0=z \end{array}$$

如果直接求 $(x+y)$，则

$$x+y=0.0101+(0.1101)=-0.1000$$

这个负数的补码为

$$[x+y]_{补}=1.1000$$

由此证明，当 z 的符号为 1 时，$z=[x+y]_{补}$。

当 z 的符号位为 1 时，根据 z 求 $(x+y)$ 的方法为：先求出 z 的补码 $[z]_{补}$，再加一负号，就是 $(x+y)$，即 $(x+y)=-[z]_{补}$，即 $x+y=-0.1000$。

(3) x 和 y 均为正数时，因为正数的补码就是正数本身，故有

$$[x]_{补}+[y]_{补}=[x+y]_{补}=x+y$$

对于两个正数相加，它的和有两种情况：一种情况是 $x+y<1$；另一种情况是 $x+y\geq 1$，此种情况即发生“溢出”的情况。

判别溢出的方法举例说明如下：

例 3：设 $x=0.0101$，$y=0.1001$，$x+y<1$，求 $x+y$。

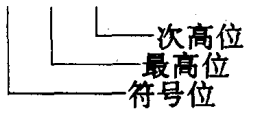
解：c_{s+1} c_s……进位

0 0 0 0 1……进位

0. 0 1 0 1 = $[x]_{补}=x$

+) 0. 1 0 0 1 = $[y]_{补}=y$

0. 1 1 1 0



由于 $x+y<1$，x 的最高位、y 的最高位和“次高位的进位”三者相加不产生进位，即 $c_s=0$。由于 x 和 y 均为正，它们的符号位均为 0，故 x 的符号位、y 的符号位和进位 c_s 三者之和不会产生进位，即 $c_{s+1}=0$。由此可见，当 $c_{s+1}=0$ 和 $c_s=0$ 时，不会发生“溢出”。实际上，只要 c_{s+1} 和 c_s 相等，就不会发生溢出。

例 4：设 $x=0.1101$，$y=0.1001$，$x+y>1$，

c_{s+1} c_s ……进位位

1 0 0 1……进位

0. 1 1 0 1 = $[x]_{补}$

+) 0. 1 0 0 1 = $[y]_{补}$

1. 0 1 1 0

由于 $x+y\geq 1$，x 和 y 的最高位和“次高位的进位”相加时将产生进位，即 $c_s=1$。由于 x 和 y 均为正，它们的符号位均为 0，故 x 的符号位、y 的符号位和进位 c_s 三者之和不会产生进位，即 $c_{s+1}=0$。由此可见，当 $c_s=1$，$c_{s+1}=0$ 时会发生溢出。事实上，只要 c_s 和 c_{s+1} 的符号不同，就发生溢出。发生溢出时，将产生错误的结果，如本例中两个正数相加的结果得到了一个负数，这显然是错误的。

(4) 两数均为负数，进行相加时有两种情况：一是 $0\geq(x+y)>-1$，这时不产生溢出；二是 $(x+y)$ 的值在 -1 与 -2 之间，这时要产生溢出。

例 5：设 $x=-0.1001$，$y=-0.0101$，$-1<x+y<0$，求 $x+y$。

解：按照通常的计算方法：

$x+y=-0.1001-0.0101=-0.1110$

$[x+y]_{补}=1.0010$

采用补码计算时，x，y 的补码分别为 $[x]_{补}=1.0111$，$[y]_{补}=1.1011$，$[x]_{补}+[y]_{补}$ 为

c_{s+1} c_s ……进位位

1 1 1 1 1……进位

1. 0 1 1 1 = $[x]_{补}$

+) 1. 1 0 1 1 = $[y]_{补}$

自动丢失 1 1. 0 0 1 0 = z