清华大学计算机系列教材

数字逻辑及数字集成电路

王尔乾　巴林凤

清华大学出版社

清华大学计算机系列教材

数字逻辑及数字集成电路

王尔乾　巴林凤　编

清华大学出版社

内 容 简 介

本书系统地阐述数制和码制、逻辑代数及逻辑函数化简、基本逻辑电路及触发器、各种集成化组合逻辑电路的设计与应用、同步时序电路及异步时序电路的设计与分析、集成化时序电路、逻辑电路的参数、集成化存储器电路等内容。

本书可作为高等学校计算机专业"数字逻辑"课程的教材，亦可供从事计算机、自动化及电子学方面生产、科研人员及有关人员参考，本书还是学习"逻辑电路"的参考书。

(京)新登字 158 号

高等学校计算机系列教材

数字逻辑及数字集成电路

王尔乾 巴林凤 合编

责任编辑 [illegible]

清华大学出版社出版

北京 清华园

清华大学印刷厂印刷

新华书店总店科技发行所发行

☆

开本：787×1092 1/16 印张：18.75 字数：445 千字

1994 年 3 月第 1 版 1994 年 3 月第 1 次印刷

印数：0001—5000

ISBN 7-302-01410-8/TP・545

定价：8.90 元

前　言

“数字逻辑”是计算机专业本科学生的一门主要课程。它是“计算机组成原理”课程的主要先导课之一，是计算机及其应用专业关于计算机系统结构方面四门主干课程（数字逻辑、计算机组成原理、微机与接口技术、计算机系统结构）的第一门课。本课程的主要目的是使学生了解和掌握从对数字系统提出要求开始，一直到用集成电路实现所需逻辑功能为止的整个过程的完整知识。课程的基本要求是系统掌握逻辑电路（重点是组合逻辑电路和同步时序电路）的分析、设计与应用。

数字集成电路是数字系统也是计算机功能实现的物质基础。由于数字集成技术的发展，迄今人们已不再用分立器件去实现逻辑功能部件了，而是用标准集成电路去构成系统。各种标准数字集成电路本身就是优美的逻辑设计作品。因此把数字逻辑和数字集成电路结合起来讲授学习，既使读者掌握数字逻辑部件的分析与设计方法，又使他们了解标准数字集成电路的原理与使用方法，无疑，这是理论结合实际的学习方法。由于篇幅及学时限制，本书不去讨论集成电路、单元电路的线路设计技术和集成电路制造工艺，只限于介绍集成电路的逻辑结构与其应用，并以介绍广泛应用的 TTL 集成电路为主。

本书是按上述思路并结合多年来教学实践经验来编写的。为突出集成电路在功能部件实现中的重要性，本书取名为“数字逻辑和数字集成电路”。本课程的参考学时数为 40～60 学时，其先修课程是电子线路。

由于编者水平有限，书中肯定会有许多缺点和错误，殷切希望广大读者批评指正。

编　者

1992 年 12 月

目　录

第一章　数制和编码

1.1　数　　制

数制是人们对数量计数的一种统计规律。日常生活中最常遇到的进位计数制是十进制，在数字系统中，广泛采用的则是二进制、八进制、十六进制。

一种进位计数包含着两个基本的因素：

(1) 基数：它是计数制中所用到的数码的个数，一般地说，基数为 R 的计数制(简称 R 进制)中，包含的是 0、1、…、R_0、R_{-1}等数码，进位规律是“逢 R 进一”，即每个数位计满 R 就向高位进 1，称为 R 进位计数制。

(2) 位权：在一个进位计数制表示的数中，处在不同数位的数码，代表着不同的数值，某一个数位的数值是由这一位数码的值乘上处在这位的一个固定常数。不同数位上的固定常数称为位权值，简称位权。不同数位有不同的位权值。例如，十进制数个位的位权值是 1，十位的位权值是 10^1，百位的位权值是 10^2。

广义地说，一个 R 进制数 N，可以有两种表示方式：

① 并列表示方式，也称位置计数法：

$$(N)_R = (K_{n-1}K_{n-2}\cdots K_1K_0.K_{-1}K_{-2}\cdots\cdots K_{-m})_R \tag{1-1}$$

其中，n 为整数部分的数位；m 为小数部分的数位；R 表示基数；K_i 为不同数位的数值：

$$0 \leqslant K_i \leqslant R-1$$

② 多项式表示法，也称以权展开式：

$$(N)_R = (K_{n-1}R^{n-1} + K_{n-2}R^{n-2} + \cdots + K_1R^1 + K_0R^0 + K_{-1}R^{-1} + \cdots + K_{-m}R^{-m}) \tag{1-2}$$

或者写成和式：

$$(N)_R = \left(\sum_{i=-m}^{n-1} K_iR^i\right)_R$$

其中：R 代表进位制的基数；m、n 为正整数，n 代表整数部分的位数；m 代表小数部分的位数；K 代表 R 进位制中 R 个数字符号中的任何一个：

$$0 \leqslant K_i \leqslant R-1$$

1.1.1　二进制

1. 二进制数的表示

基数 R＝2 的数制为二进制。二进制数的数值表示符号只有“1”和“0”，进位规律是“逢二进一”，任意一个二进制数 N 的多项式表示为：

$$(N)_2 = K_{n-1}2^{n-1} + K_{n-2}2^{n-2} + \cdots + K_12^1 + K_02^0 + K_{-1}2^{-1} + \cdots + K_{-m}2^{-m}$$

$$=\left(\sum_{i=-m}^{n-1} K_i 2^i\right)_2 \tag{1-3}$$

其中：K_i 为 0 或 1，2 为位权。

例如：二进制数 1011.101 可以展开为：

$$1011.101 = 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3}$$

2. 二进制数的运算

二进制数的算术运算比较简单，只要记住二个二进制整数的和与积的运算规律就可以了。

加法规律为：

$$0+0=0 \qquad 0+1=1+0=1 \qquad 1+1=10$$

乘法规律为：

$$0\times 0=0 \qquad 0\times 1=1\times 0=0 \qquad 1\times 1=1$$

由于二进制数每位只可能有两种数值 0 或者 1，在数字系统中，可用电子器件的两种不同的状态来表示一位二进制数，因此实现起来非常方便。例如，我们在数字系统中，用晶体管的导通表示“0”，而晶体管的截止表示“1”；或用低电位表示“0”、高电位表示“1”。所以二进制数的物理实现简单、易行、可靠，并且存储和传送也方便。其运算规则也很简单。但二进制书写位数太多，不便记忆。为此，通常用八进制和十六进制数作为二进制数的缩写。

1.1.2 八进制

八进制数的数位符号有八个，即 0～7，进位规律是“逢八进一”，基数 R＝8，任意的一个八进制数 N 的多项展开式为：

$$(N)_8 = K_{n-1}8^{n-1} + K_{n-2}8^{n-2} + \cdots + K_1 8^1 + K_0 8^0 + K_{-1}8^{-1} + \cdots + K_{-m}8^{-m}$$

$$=\left(\sum_{i=-m}^{n-1} K_i 8^i\right)_8 \tag{1-4}$$

其中：K_i 表示为 0～7 中的任意一个。

1.1.3 十六进制

一个十六进制数的表示符号有 16 个，即 0～9，以及用 A、B、C、D、E、F 分别表示 10～15。进位规律为"逢十六进一"，基数 R＝16，任意的一个十六进制数 N 的多项表示式为：

$$(N)_{16} = K_{n-1}16^{n-1} + K_{n-2}16^{n-2} + \cdots + K_1 16^1 + K_0 16^0 + K_{-1}16^{-1} + \cdots + K_{-m}16^{-m}$$

$$=\left(\sum_{i=-m}^{n-1} K_i 16^i\right)_{16} \tag{1-5}$$

其中，K_i 表示为 0～9 以及 A、B、C、D、E、F 中的任意一个。

1.1.4 二进制与八进制、十六进制之间的转换

1. 八进制转换为二进制

把每位八进制数用三位二进制数表示。

例如$(312.64)_8$，3、1、2、6、4 各用三位二进制数表示：

3	1	2	.	6	4
011	001	010	.	110	100

所以$(312.64)_8=(11001010.1101)_2$

2. 二进制数转换为八进制

二进制转换为八进制时，整数部分从低位向高位每三位分为一组，最高一组不够时，用 0 补足；小数部分从高位向低位每三位一组，最后不足三位的，在低位补 0，然后把每三位的二进制数用相应的八进制数表示。

例如$(10110.11)_2$

010	110	.	110
2	6	.	6

$(10110.11)_2=(26.6)_8$

3. 十六进制转换为二进制

把每位十六进制数用相应的四位二进制数表示。

例如$(21A.5)_{16}$

2	1	A	.	5
0010	0001	1010	.	0101

$(21A.5)_{16}=(1000011010.0101)_2$

4. 二进制转换为十六进制

整数部分由小数点向左，每四位一组，最高一组不足四位数前面补 0；小数部位由小数点向右，每四位一组，最后不足四位的，在低位补 0，然后，把每四位二进制数用相应的十六进制数表示。

例如：$(1100101.101)_2$

0110	0101	.	1010
6	5	.	A

$(1100101.101)_2=(65.A)_{16}$

由于八进制、十六进制比二进制书写简短、容易读写，也便于记忆，并且转换为二进制也较方便。因此，在数字系统中得到普遍应用。

1.1.5 二进制与十进制之间的转换

1. 二进制转换为十进制

把二进制数按权展开，利用十进制运算法则，求出其值，即可将二进制数转换为十进制数。

例如：

$$(101101.101)=1\times2^5+1\times2^3+1\times2^2+1\times2^0+1\times2^{-1}+1\times2^{-3}$$
$$=32+8+4+1+0.5+0.125$$
$$=(45.625)_{10}$$

2. 十进制转换为二进制

一个具有整数部位和小数部位的十进制数转换为二进制数时，应当分别将其整数部分和小数部分转换为二进制数，然后用小数点将两部分连接起来。现举例说明。

例如：将十进制数$(157)_{10}$转换为二进制数：

根据公式(1-2)：

$$(157)_{10}=K_{n-1}2^{n-1}+K_{n-2}2^{n-2}+\cdots+K_12^1+K_02^0$$

显然，等式右边除K项外都有2的因子。因此，用2除$(157)_{10}$，所得余数即为K_0，

$$2\ \big|\ 157$$
$$78\cdots\cdots\text{余数}\ 1=K_0$$

并得到等式：

$$(78)_{10}=K_{n-1}2^{n-2}+K_{n-2}2^{n-3}+\cdots+K_22^1+K_1$$

同样，再用2除$(78)_{10}$，余数则为K_1，

$$2\ \big|\ 78$$
$$39\cdots\cdots\text{余数}\ 0=K_1$$

再用这样的方法一直继续下去，直到商为0为止。

2	157	
2	78	余数为1，所以$K_0=1$
2	39	余数为0，所以$K_1=0$
2	19	余数为1，所以$K_2=1$
2	9	余数为1，所以$K_3=1$
2	4	余数为1，所以$K_4=1$
2	2	余数为0，所以$K_5=0$
2	1	余数为0，所以$K_6=0$
	0	余数为1，所以$K_7=1$

得$(157)_{10}=(10011101)_2$

十进制小数转换为二进制小数的方法是：不断用2乘要转换的十进制小数，将每次所得的整数(0或1)，依次记为K_{-1}、K_{-2}、…。若乘积的小数部分最后能为0，那么最后一

次乘积的整数部分记作 K_{-m}，则 $0.K_{-1}K_{-2}\cdots K_{-m}$ 即为十进制小数的二进制表达式。因为十进制小数，并不都是能用有限位的二进制小数精确表示，通常则是根据精度要求 m 位，作为十进制小数的二进制的近似表达式。

例如：将 $(0.913)_{10}$ 转换为二进制数，根据公式(1-2)：

$$(0.913)_{10} = K_{-1}2^{-1} + K_{-2}2^{-2} + \cdots + K_{-m}2^{-m}$$

两边乘以 2，则得：

$$(1.826)_{10} = K_{-1} + K_{-2}2^{-1} + \cdots + K_{-m}2^{-m+1}$$

所以 $K_{-1}=1$；那么，$(0.826)_{10}=K_{-2}2^{-1}+K_{-3}2^{-2}+\cdots+K_{-m}2^{-m+1}$

两边再乘以 2，则得：

$$(1.652)_{10} = K_{-2} + K_{-3}2^{-1} + K_{-4}2^{-2} + \cdots + K_{-m}2^{-m+2}$$

假如精度要求 m=4，转换过程如下：

$$\begin{array}{r} 0.913 \\ \times \quad 2 \\ \hline 1.862 \end{array}$$ 整数部分为 1，所以 $K_{-1}=1$；

$$\begin{array}{r} 0.826 \\ \times \quad 2 \\ \hline 1.652 \end{array}$$ 整数部分为 1，所以 $K_{-2}=1$；

$$\begin{array}{r} 0.652 \\ \times \quad 2 \\ \hline 1.304 \end{array}$$ 整数部分为 1，所以 $K_{-3}=1$；

$$\begin{array}{r} 0.304 \\ \times \quad 2 \\ \hline 0.608 \end{array}$$ 整数部分为 0，所以 $K_{-4}=0$；

因此，$(0.913)_{10}=(0.1110)_2$

任意进制与十进制之间的转换原理及方法，同二进制与十进制之间的转换原理及方法相类似，不再重复。

而任意两种进制之间的转换，一般说来是先由一种进位制转换为十进制，再由十进制转换为另一种进制，把十进制作为桥梁。例如实现 $(N)_a$ 转换为 $(N)_b$ 时，首先是将 $(N)_a$ 转换为 $(N)_{10}$，也就是将 $(N)_a$ 展开为 a 进位制的多项式，在十进制中计算其值，然后再利用基数乘除法，将 $(N)_{10}$ 转换为 $(N)_b$。

1.2 编　码

1.2.1 带符号的二进制数的编码

在通常的算术运算中，用“+”号表示正数，用“−”号表示负数。而在数字系统中，正、负数的表示方法是：把一个数的最高位作为符号位，并用“0”表示“+”；用“1”表示“−”。连同符号位在一起作为一个数，称之为机器数，它的原来的数值形式则称为这个机器数的真值。

例如：　　$X_1=+0.1101$；　　$X_2=-0.1101$

表示成机器数为：　　$X_1=0.1101$；　　$X_2=1.1101$

在数字系统中，表示机器数的方法很多，目前常用的有原码、反码和补码。

1. 原码(True Form)

原码表示法又称符号—数值表示法。正数的符号位用“0”表示；负数的符号位用“1”表示；数值部分保持不变。

(1) 小数原码的定义：

若二进制数 $X=\pm 0.X_{-1}X_{-2}\cdots X_{-m}$

1) $X>0$ 时

$$X=+0.X_{-1}X_{-2}\cdots X_{-m}$$

$$(X)_{原}=0.X_{-1}X_{-2}\cdots X_{-m}$$

2) $X<0$

$$X=-0.X_{-1}X_{-2}\cdots X_{-m}$$

$$\begin{aligned}(X)_{原}&=1.X_{-1}X_{-2}\cdots X_{-m}\\&=1-(-0.X_{-1}X_{-2}\cdots X_{-m})\\&=1+X\end{aligned}$$

例如：$X_1=+0.1101$　　则 $(X)_{原}=0.1101$

$X_2=-0.1101$　　则 $(X)_{原}=1-(-0.1101)=1.1101$

3) 零的原码有两种表示形式

$$(+0)_{原}=0.00\cdots0$$

$$(-0)_{原}=1.00\cdots0$$

所以小数原码表示为：

$$(X)_{原}=\begin{cases}X & \text{当 } 0\leqslant X<1\\ 1-X & \text{当 } -1<X\leqslant 0\end{cases}$$

(2) 整数原码的定义

若 $X=\pm X_{n-1}X_{n-2}\cdots X_0$

1) $X>0$ 时，则　　$X=+X_{n-1}X_{n-2}\cdots X_0$

$$(X)_{原}=0X_{n-1}X_{n-2}\cdots X_0$$

2) $X<0$ 时，则　　$X=-X_{n-1}X_{n-2}\cdots X_0$

$$
\begin{aligned}
(X)_{原} &= 1X_{n-1}X_{n-2}\cdots X_0 \\
&= 2^n + X_{n-1}X_{n-2}\cdots X_0 \\
&= 2^n - (-X_{n-1}X_{n-2}\cdots X_0) \\
&= 2^n - X
\end{aligned}
$$

例如：X＝－1101

$$
\begin{aligned}
(X)_{原} &= 11101 \\
&= 10000 + 1101 \\
&= 10000 - (-1101) \\
&= 2^n - X
\end{aligned}
$$

因此，整数原码定义为：

$$
(X)_{原} = \begin{cases} X & 当\ 0 \leqslant X < 2^n \\ 2^n - X & 当\ -2^n < X \leqslant 0 \end{cases}
$$

原码表示法简单易懂，但在数字系统中，要进行两个异号原码的加法运算时，需先判两数的大小，然后才能从大数中减去小数。最后，还要判结果的符号位，这就增长了运算时间。

2. 反码(One's complement)

反码的符号位表示法与原码相同，即符号“0”表示正数，符号“1”表示负数。与原码不相同的是反码数值部分的形成和它的符号位有关。正数反码的数值和原码的数值相同，而负数反码的数值是原码的数值按位求反。

(1) 整数的反码

若　$X = \pm X_{n-1}X_{n-2}\cdots X_0$

则定义为：

$$
[X]_{反} = \begin{cases} X & 当\ 0 \leqslant X < 2^n \\ (2^{n+1} - 1) + X & 当\ -2^n < X \leqslant 0 \end{cases}
$$

例如：$X_1 = +1101$，则$(X_1)_{反} = 01101 = 1101$

$X_2 = -1101$，则
$$
\begin{aligned}
(X_2)_{反} &= (2^5 - 1) + X \\
&= (100000 - 000001) + (-1101) \\
&= 11111 - 1101 \\
&= 10010
\end{aligned}
$$

(2) 小数反码的定义

若 $X = \pm 0.X_{-1}X_{-2}\cdots X_{-m}$

则定义为：

$$
[X]_{反} = \begin{cases} X & 当\ 0 \leqslant X < 1 \\ 2 - 2^{-n} + X & 当\ -1 < X \leqslant 0 \end{cases}
$$

例如：$X_1 = +0.1101$，则$[X_1]_{反} = 0.1101$

$X_2 = -0.1101$，则
$$
\begin{aligned}
[X_2]_{反} &= 2 - 2^{-4} + X \\
&= 10.0000 - 0.0001 - 0.1101
\end{aligned}
$$

$$=1.0010$$

(3) 零的反码有两种形式：

$$[+0]_{反}=0.00\cdots0$$

$$[-0]_{反}=1.11\cdots1$$

作反码加、减法时，要将运算结果的符号位产生的进位(0 或 1)加到和的最低位，才能得到最后结果。

例如：将 X_1、X_2 作反码加：$X_1=+1101$，　$X_2=-0101$

$$[X_1]_{反}=01101，\quad [X_2]_{反}=11010$$

$$[X_1]_{反}+[X_2]_{反}=01101+11010$$

$$=\text{“1”}00111$$

将符号位产生的进位“1”，加到最低位，即为

$$[X_1+X_2]_{反}=00111+1=01000$$

在反码表示中，±0 的表示不是唯一的，因此，使用反码不很方便。

3. 补码(Two's complement)

补码的符号表示和原码相同。“0”表示正数；“1”表示负数。正数的补码和原码、反码相同，就是二进制数值本身。负数的补码是这样得到的：将数值部分按位求反，再在最低位加 1 。

(1) 整数的补码

若 $X=\pm X_{n-1}X_{n-2}\cdots X_0$

则定义为：

$$[X]_{补}=\begin{cases}X & \text{当 } 0\leqslant X<2^n\\ 2^{n+1}+X & \text{当 } -2^n\leqslant X<0\end{cases}$$

例如：$X_1=+1101$，　则$[X_1]_{补}=1101$

$X_2=-1101$，　则$[X_2]_{补}=2^5+X$

$$=100000-1101$$

$$=10011$$

(2) 小数的补码

若 $X=\pm0.X_{-1}X_{-2}\cdots X_{-m}$

则定义为：

$$[X]_{补}=\begin{cases}X & \text{当 } 0\leqslant X<1\\ 2+X & \text{当 } -1\leqslant X<0\end{cases}$$

(3) 零的补码只有一种形式：

$$[0]_{补}=0.000\cdots0$$

引入补码以后，可将数字系统的减法运算用加法实现。在求得和的结果中，要将运算结果产生的进位丢掉，才得到正确结果。

例如：在补码系统中，实现 X_1-X_2 运算：

$$X_1=1101，\quad X_2=0101$$

求出：$[X_1]_{补}=1101$，　$[-X_2]_{补}=11011$

做$[X_1]_{补}+[-X_2]_{补}=1101+11011$

$=$“1”01000

丢掉最高位的“1”，即得$(X_1-X_2)_{补}$的正确结果。

显然，两数相减时，用补码求和运算比用原码求和要简单。但补码的缺点是负数用补码表示不直观。

表 1-1 给出了几个典型数的真值、原码、反码、补码的表示。

表 1-1

X	$[X]_{原}$	$[X]_{反}$	$[X]_{补}$	X	$[X]_{原}$	$[X]_{反}$	$[X]_{补}$
+1001	1001	1001	1001	−0.0000	1.0000	1.1111	0.0000
+0001	0001	0001	0001	−0.0010	1.0010	1.1101	1.1110
+0.1101	0.1101	0.1101	0.1101	−0011	1.0011	1.1100	1.1101
+0.0000	0.0000	0.0000	0.0000	−1010	1.1010	1.0101	1.0110

1.2.2 带小数点的数的编码

一个数既有小数部分又有整数部分，在数字系统中是如何表示的？一般有两种方式：定点表示和浮点表示。

任何数制的数 N，均可以表示为：

$$N = R^E \times M \tag{1-6}$$

其中：R 为进制的基数；E(Exponent)为阶码，取值为整数；M(Mantissa)为数 N 的尾数，取值为整数或小数。

例如：$(N_1)_{10}=(516000)_{10}=10^3\times 516$

$(N_2)_{10}=(0.25)_{10}=10^{-2}\times 25$

数 N_1 的阶码 $E_1=3$、尾数 $M_1=516$；数 N_2 的阶码 $E=-2$，尾数 $M=25$。

对于二进制数，

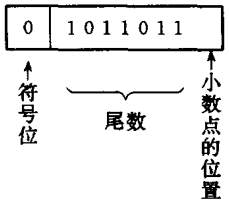
$$N = 2^E \times M \tag{1-7}$$

1. 定点表示法：

所谓的定点表示法就是在一个数中小数点的位置在数中是固定不变的。这个固定的位置是事先约定好的，不必用符号表示。在定点表示中，阶码 E 为零。

当 E=0，尾数 M 为纯整数时，则认为小数点在尾数 M 最低位右边，为整数定点。

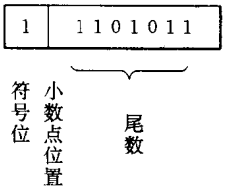
例如：N=+1011011，则表示为



当 E=0，尾数 M 为纯小数时，则认为小数点的位置在 M 最高位的左边，定点数只能

表示小数，为小数定点。

例如：N＝－0.1101011，则表示为

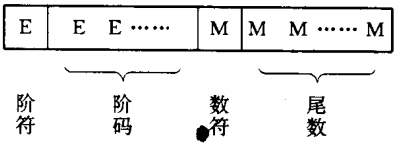


定点数表示法，数N范围是限定的，在小数定点时，当用8位二进制数表示一个数时，1位符号位，7位表示数值，若只考虑绝对值，最大数取值为$(0.1111111)_2=(127\div128)_{10}$

最小数取值为$(0.0000001)_2=(1\div128)_{10}$

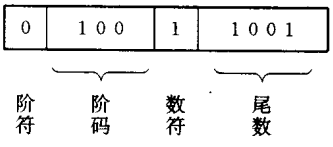
2. 浮点表示法

在数中小数点的位置不是固定不变的，而是可以变化的，这种表示法称为浮点表示法。阶码E和尾数M各自可分别采用原码、反码和补码的形式。若尾数M采用反码表示，阶码E采用补码表示，表示格式如下：



例如：$N=2^4\times(-9)$，用二进制数表示为　　$N=2^{100}\times(-1001)$

浮点表示的格式如下：



尾数M表示了数N的全部有效数字；而阶码E指明了小数点的位置。小数点移动的原则是小数点向左移一位，相当于尾数的数码向右移一位，而阶码加1。

浮点数的运算规则要比定点数复杂。如阶码相同的两浮点数求和，只要将尾数作相加运算，所得为和数的尾数，而阶码仍为原来的阶码。如阶码不等的两数求和，则先将阶对齐，然后才能对尾数求和。例如：

$$a=2^{11}\times0.1001$$

$$b=2^{01}\times0.1100$$

首先对阶，让小阶向大阶看齐，阶小的尾数小数点要左移一位阶码加1，直到阶码相同为止。对a、b来说，必须使b的尾数左移两位，阶码加2，得：

$$b=2^{11}\times0.0011$$